平成 21 年 3 月修了

博士（工学）学位論文

セルフタイム型パイプライン・システムの構成法に関する研究

A study on design scheme for self-timed pipelined systems

平成 20 年 12 月 12 日

高知工科大学大学院　工学研究科　基盤工学専攻 (基盤工学コース)

学籍番号　1086108

三宮　秀次

SANNOMIYA Shuji

要 旨

セルフタイム型パイプライン・システムの構成法に関する研究

三宮 秀次

将来の数十〜数百億個のトランジスタを用いた大規模システム集積において，トランジスタを活用して，処理性能や設計生産性を含む総合的なシステム性能の向上に直接的に転換するには，ソフト／ハードの組合せの爆発的な増加に対して，従来の大域的あるいは静的な制御を改良・複雑化するのではなく，計算に必要な制御を局所化・極小化する，自律的な計算を実現できる，システム構成原理の確立が極めて重要である．セルフタイム型パイプライン（STP）は，隣接するパイプライン段間の配線のみを用いて局所的にデータを転送する構成により，配線局所化による信号スキュー問題の緩和のみならず，パイプライン段水準で自律的に動作する．STP システムの設計では，並列性を徹底して抽出できるよう，パイプライン構造をアプリケーション毎に特化させて，高いデータ流量を維持することが高性能化の要となる．また，全システム要素を集積して処理性能と生産性の向上を狙う SoC（System-on-Chip）／ SiP（System-in-Package）構成を用いて，アプリケーション毎にパイプライン構成を最適化する特定用途向けシステムを実現するには，設計の初期段階から定量的なシステム評価に基づいて要求に見合う多種多様なアーキテクチャから最適なものを選択する必要があり，したがって，システム評価が設計のボトルネックとなり得る．

本研究では，大規模システム集積に STP を徹底して活用するため，STP を蜘蛛の巣（web）のように自由に張り巡らせた上で大規模な並列処理を直接的に展開できる，ウェブパイプライン構成法を提案する．また，このウェブパイプラインを実現するため，パイプライン間の任意の演算とデータ流制御を実現する相互作用制御回路の構成法を提案する．さらに，提案回路に基づき，専用処理用 LSI 回路を設計し，十分な処理性能が得られることを示す．これに加えて，STP の振舞いをマクロに捉える軽量かつ正確な動作モデルを提案

し，提案モデルに基づくシステム評価をチップ上で直接，高速に実行できる，オンチップ・シミュレーション手法を提案する．また，提案モデルに基づくオンチップ・シミュレータは，従来のシステム評価手法と同程度の精度を持つことを示す．さらに，シミュレーション回路を搭載する場合，通常の処理コア回路に対する回路規模の増加が約 5 割程度に収まる一方で，実システムの実行時間の 80 倍以内に収まるシミュレーション時間で高速に評価可能であることを示す．

キーワード　セルフタイム型パイプライン，パイプライン間相互作用，シミュレーション，性能見積りモデル

目次

目次

第 1 章

序論

半導体素子の微細加工集積技術の進展により，同一の集積回路チップ上で利用可能な半導体素子すなわち計算資源（トランジスタ）は膨大な量になりつつある．今後の半導体チップの集積度の指標としては，国際半導体技術ロードマップ（ITRS）があり，集積回路チップ上の最小配線間距離（プロセス・ルール）を基準に，設計面や製造面の技術目標が整理されている [1]．今日までそれらの技術目標を達成するための研究開発の努力によりプロセス・ルールの断続的な縮小が達成されており，今後は，同一チップ上に集積できるトランジスタの数は数十億となる，ギガ・スケール集積の実現が期待される．

今後は，こうした膨大な量の計算資源を活用した，情報処理通信システムの高性能化が期待できる．例えば，アプリケーション毎の処理やデータの並列度に応じて多数の処理回路を搭載することによる処理の高並列化が可能である．また，例えば無線処理用の回路などの周辺装置を含む情報通信処理システムの全機能をチップに収める，システム集積により，同一チップ内の低遅延の信号の伝達が実現でき，処理速度は飛躍的に向上する．一方で，こうした大規模なシステムの設計では，規模に応じて，ソフトとハードの組合せは爆発的に増加するため，設計作業量の加速度的な増加は必至であり，設計に掛かるコストを無視してシステムの性能は議論できなくなる．すなわち，こうした将来のシステムの集積では，計算資源を活用して，処理性能と共に設計や生産の容易性も含めた総合的なシステム性能を向上できる， システム構成法の確立が必要である．

システム集積では，並列度に応じた処理装置に加えて，設計の各工程を支援し，作業を容易化するための回路を積極的に集積することにより，システム性能の向上が期待できる．設計工程支援回路には，例えば，回路レベルのテスト支援のための BIST（built-in self-test）

回路 [2] や，回路最適化を支援するオンチップ波形測定回路 [3] などがある．これらをチップに組込めば，時間の掛かるオフライン（オフチップ）の模擬等が不要になるうえに，チップ上の情報を利用し易くなり，したがって，設計の大幅な容易化が期待できる．こうした，システム集積の構想には，同一チップ上への集積を追及する SoC（System-on-Chip）と，同一パッケージ中の複数チップ上に分割して集積してチップ単位の再利用性や歩留りの向上を狙う SiP（System-in-Package）があるが，どちらも全システム要素の集積の点では同じである．

そうした SoC/SiP 構成は，回路集積の容易性や計算のオーバヘッドの観点から，もはや単一の中央集権的な機構を想定する従来の大域的な制御による実現は困難になる．既に，消費電力対性能および信頼性の向上を阻害する消費電力／発熱に対して，高電圧高周波数で動作する単一のプロセッサに代わり，消費電力の少ない低電圧低周波数で動作する多数のプロセッサ（処理コア）を用いて，アプリケーションの並列性をプロセッサ水準で活用して，より少ない電力消費で全体としてのスループットを維持あるいは向上させるマルチコア技術が研究されている [4, 5]．こうした多数の処理コアと命令の組合せは，コア数とソフトウェアの規模に応じて爆発的に増加する．これに対して，ソフト面からは，OS（operating system）やコンパイラによる大域的あるいは静的な計算資源の割当てや制御は困難となり，したがって，必ず，アプリケーションの計算自体に対するオーバヘッド，あるいは設計のボトルネックとなる．また，ハード面からは，大域同期信号（クロック）やバス方式による計算の制御は，制御下にある回路の部分的な機能の追加や変更であっても回路全体に影響が波及し得るため，回路の階層的ないし局所的な管理が困難であり，今後の大規模化によって設計限界が必ず訪れる．特に，主流となってしまったクロックによるシステム構成法では，集積回路の製造工程において回路素子の特性の制御が困難となる今後の微細加工技術における性能向上は困難である．既に，現行の製造技術でさえ約 30%もの実効性能のバラつきが報告されている [6]．これを鑑みると，バラつきの影響を局所的に留めることが困難であるクロック同期式の回路では，今後の微細化に伴った，歩留りや性能の向上が期待できない．以上のことから，将来のシステム集積には，計算のための制御を局所化および極小化できる，

自律的な計算を実現できるシステム構成原理が不可欠である．

セルフタイム型パイプライン（STP: self-timed pipeline）は，時間的に並行する計算を徹底してパイプライン段に分割して統治することで，各パイプライン段が自律的に計算するパイプライン・システム構成原理である．STP に基づけば，パイプライン段水準でシステムの構成要素の追加や変更が容易にできるため，設計時の組合せ論的爆発の問題を回避できる [7]．また，配線はパイプライン段間に局所化されるため，大域に亘る長距離の回路素子間の配線に伴う高消費電力／発熱化や EMI（電磁妨害）といった回路集積を阻害する要因は緩和される．STP は，データ転送時にのみ電力を消費する自律的な省電力機能も有している [8, 9]．既に，これらの STP の特徴を活用して，高速データパス回路，非同期回路プロセッサ，大域非同期局所同期システムなど多数の応用が検討されている [10, 11, 12, 13]．筆者らも，既に，データ駆動原理に基づき抽出された並列性に応じて自然に徹底して高速パイプライン並列処理するデータ駆動プロセッサ（DDP）構成を研究しており，映像信号処理やネットワークプロセッサへの応用を検討している [8, 14]．DDP 構成では，マルチコア化により，プロセッサ水準から命令水準までの粒度に応じて徹底した並列処理が可能である．

本論文は，STP をシステム構成原理として徹底して活用するために，アプリケーションに内在する並列性を最大限に引き出すパイプライン・システム構成法を提案し，さらに，パイプライン・システム設計を支援するための評価回路を搭載するシステム構成法を提案するものである．

STP は，隣接するパイプライン段間の転送制御（ハンドシェイク）のみを保証すれば，全体の動作を保証できるため，柔軟なパイプライン構造を実現できる．この柔軟性を活用し，並列処理の形態に沿ってパイプラインを展開すれば，大規模なパイプライン並列処理による処理性能の向上が期待できる．データフロー計算原理の観点から，処理は，データストリームにおける個々のデータ間の演算と流れ制御の組合せ（相互作用）と考えられる．この相互作用を，アプリケーション毎のデータ流に沿って，直接的にパイプライン段間で実現できれば，データ流に内在する並列性を最大限に引き出せる．既存技術としては，二本のパイプライン間で双方向の相互作用を実現する転送制御 [15] が研究されている．しかし，これは，

パイプライン間の任意の経路を想定しておらず，柔軟な流れ制御の実現が困難である．したがって，パイプライン間の任意の経路に対する相互作用制御が必要となる．

一方，こうしたアプリケーションに特化したシステムの設計現場では，ユーザ要求の多様化に伴う製品ライフサイクルの短期間化に，設計コストを抑えて応える必要がある．これに対して，設計の初期段階からシステム全体の定量的評価に基づいて要求に見合ったシステム・アーキテクチャを選択しておけば，設計の手戻りを局所化かつ極小化することができ，設計の短期間・低コスト化が期待できる．そのためには，初期の設計段階から，異なる命令セットを持つプロセッサや特定の処理に特化した専用命令／エンジンを含むマルチコアを搭載した SoC ／ SiP 構成を想定する必要がある．この際，新規の SoC ／ SiP 構成を容易にシミュレーションできれば，応用分野の専門家はハードウェア実装を待たずに，直接的に SoC ／ SiP 構成を最適化できるため，システム開発期間の大幅な短縮が期待できる．ところが，性能見積りのための，従来の回路シミュレーションやシステムレベルの動作シミュレーションは多大な時間を要するため [16]，明らかに適用が困難である．一般にセルフタイム型パイプライン・システムでは，信号の遷移やデータ授受などのあらゆるイベントが，クロック同期システムのように同期して離散化されず，すべて因果関係のみによって生起する．これに対して，マルコフ連鎖による確率過程解析モデル [17] や確率的なタイミング解析 [18] 等による非同期式システムの振舞いの解析手法が研究されている．これらは，イベントの生起確率が定めやすい定常状態の解析には効果がある．しかし，プログラムの解釈・実行結果によって振舞いが大きく異なり得る，実システムの性能見積りへの適用では，確率的なモデル化自体が困難であり，直接的な適用ができない．また，ステージ毎のパケットの転送を模擬して，パケットの追跡を可能とする手法として，半導体レベルの転送時間をモデル化した Charie 図を用いる手法 [19] や，時間ペトリネットによる離散事象シミュレーションの高速化手法 [20] 等が研究されているが，いずれの手法でも，すべてのパケットに対してステージ毎に転送を逐一模擬する必要があり，システムが大規模化するにつれてステージ毎の状態管理コストが増大し，高速なシミュレーションは望めない．さらに，今後のシステムの大規模化を考慮すると，シミュレーション時間対実行時間比の増加が予測されるため，ソフトウェ

ア（オフライン）・シミュレーションによる大幅な高速化は期待できない.

これらに対して，本論文は，任意のパイプライン並列処理構造を実現するウェブパイプラインを提案し，これを実現するためのパイプライン間相互作用制御回路を提案する．また，STP 中のデータの挙動をチップ上で直接的に模擬してシミュレーション時間を大幅に短縮できるオンチップ・シミュレーション手法を提案し，これを実現するための STP の動作モデルとセルフタイム型パイプライン回路構成を提案する.

以下に本論文の構成を示す.

第 2 章においては，パイプライン・システム構成の要件を議論した上で，セルフタイム型パイプラインの動作特性を示し，大規模システムへの適用要件として，パイプライン間相互作用および高速システム評価が必要であることを明らかにする.

第 3 章では，セルフタイム型パイプラインをチップ全域に展開するためのパイプライン間相互作用制御回路の構成法を示す.

第 4 章では，システム検証において，セルフタイム型パイプラインの振舞いをデータが転送される速度に着目してマクロに捉えることで，シミュレーション・コストを大幅に削減するマクロフロー・モデルを示す．そして，マクロフロー・モデルに基づく高速な性能評価法を示す.

第 5 章では，高速システム評価を実現するマクロ・シミュレーション手法とトレース駆動エミュレーション手法を示し，そのセルフタイム型パイプライン回路構成を示す．また，これらを搭載したデータ駆動型プロセッサ・コア構成を示す.

第 6 章では，本研究で提案したウェブパイプラインの実現可能性および有効性を評価する．また，提案したシミュレーション手法の精度の評価，及びオンチップ・シミュレーション回路の FPGA 実装に基づく実現可能性を示す.

第 2 章

パイプライン・システム構成の要件

2.1 緒言

パイプライン処理機構はチップ上の素子の稼働率を向上する常套手段として，現在では，広くアーキテクチャ水準から回路水準に至るまで活用されている．しかしながら，いずれも直線状ないしは環状のパイプラインとして部分的に活用されている．これでは，データの経路が限られるため，演算と演算間のデータ流を直接的に写像した高並列パイプライン処理が実現できない．すなわち，大規模 SoC ／ SiP レベルの膨大な素子を並列処理に有効活用するには，柔軟なパイプライン構成が重要である．

こうした柔軟なパイプライン構造による大規模パイプライン並列処理では，演算の実現は多様な形態をとり得る．まず，最も直接的なパイプライン展開として，パイプライン段に演算を割当て，さらに演算間のデータの転送をパイプライン段間のデータ転送で実現するパイプライン処理形態がある．この場合，演算の粒度が細かいほど，より高並列なパイプライン処理が可能となる一方で，演算毎にパイプライン・レジスタが必要であるため，パイプライン回路の占める面積は大きくなる．これに対して，汎用的な演算をパイプライン段に割当て，データ毎に演算を指定して，演算器を複数の演算に再利用する，パイプライン処理形態がある．この場合，より小規模に回路を構成できる一方で，データ毎の演算の指定や，適切な演算器へのデータの転送が付加的な制御，すなわちオーバヘッドとなり，したがって，パイプライン処理の効率は低下する．つまり，並列性を活用する上で，回路規模と処理性能はトレードオフの関係となる．演算の実行において，付加的な制御は，すなわち処理を制御する命令であり，命令の解釈・実行と演算の実行を実現するパイプラインを合わせて，プロ

セッサ・コア（形態）と呼ぶ．これに対して，相対的に見て，命令を専用化し命令の解釈・実行が簡易ないしは無いパイプラインを専用処理エンジン（形態）と呼ぶ．以上のことから，システムの設計では，要求制約を満足できるように，プロセッサ・コアや専用処理エンジンと，それらを制御する一連の命令，すなわちハードウェアとソフトウェアを適切に切り分けて決定する必要がある．

このソフト／ハードの切り分けは，システム設計の手戻りを局所化かつ最小化する観点から，初期の設計段階で，定量評価に基づき決定しておく必要がある．この際，要求性能が満足できなければ，命令セット，専用処理エンジン，およびプロセッサ・コアの構成の異なる，新規アーキテクチャの設計が必要になる．このとき，システムの定量評価が設計のボトルネックにならないよう，新規アーキテクチャ上で実行される応用プログラムの模擬をなるべく高速に行うことが重要である．一般に，ソフトウェア・シミュレーションは，システムの実行時間に比べて低速である．これに対して，再構成可能なデバイスを用いて比較的柔軟に高速なエミュレーションを行う手法 [21] なども提案されている．しかし，一般的なエミュレーションの場合，ハードウェア実装が完了しないとシステム全体の評価ができないため，初期の設計段階における定量評価には適さない．

これらに対して，本章では，まず，チップ上の膨大な素子を余すところなく稼働させて超並列処理を実現できる，新しいパイプライン・アーキテクチャとして，セルフタイム型ウェブパイプラインの考え方を示し，その実現に必要となる要素技術について議論する．次に，シミュレーションのための回路をプロセッサに搭載してチップ上で高速に模擬を行う手法を示し，その実現に必要となる動作モデルについて議論する．

2.2　ウェブパイプライン構成

ウェブパイプラインは，図 2.1 に示すように，蜘蛛の巣（web）状に複数のパイプライン機構を相互に接続して，それらの間の相互作用機構を含めて，チップ上に展開できる構成の総称である．このような構成によって，対象とするアプリケーションの同時並行・パイプラ

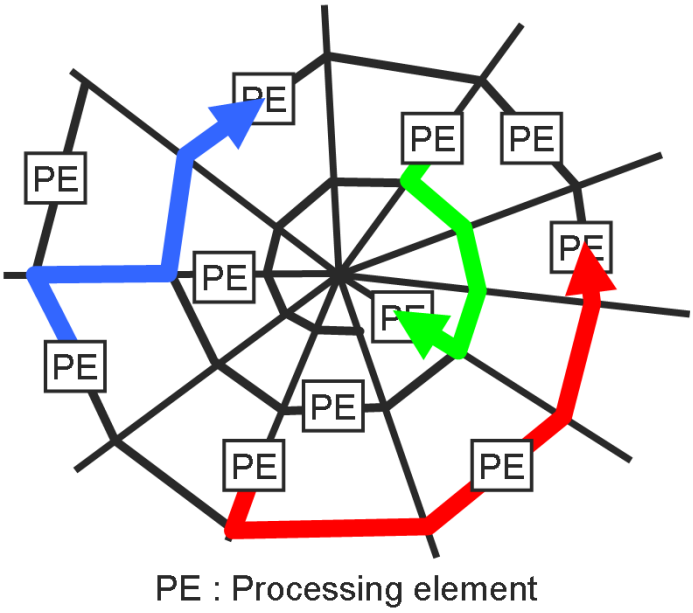

図 2.1　ウェブパイプライン (概念図)

イン並列処理の構造に適した形態で，処理対象データの論理的な流れ（データフロー）を物理的なパイプライン処理機構上に自然に写像することが可能になる．

より一般的には，このような大規模 LSI チップ上では複数のアプリケーション（ないしは，サブ・アプリケーション）を多重並列に実現する必要がある．当然のことながら，アプリケーション毎に要求性能が異なるため，それに応じた写像が必要となる．典型的には次のような写像方法が想定できる．

(a)　最も単純な写像である．図 2.2(a) に示すように，アプリケーションの論理的なデータフローをそのまま物理的なパイプライン機構上に写像する．

(b)　単純な写像では要求性能を満たさない場合に，同種のハードウェア機能を複数カスケードした，アクセラレータ・パイプライン上に論理的データフローを写像する（図 2.2(b)）．

(c)　単純な写像では過度な性能になる場合には，多機能な ALU を搭載したプロセッサ（ないしは，マルチプロセッサ）上に写像して，ソフトウェアをパイプライン並列に実行する（図 2.2(c)）．

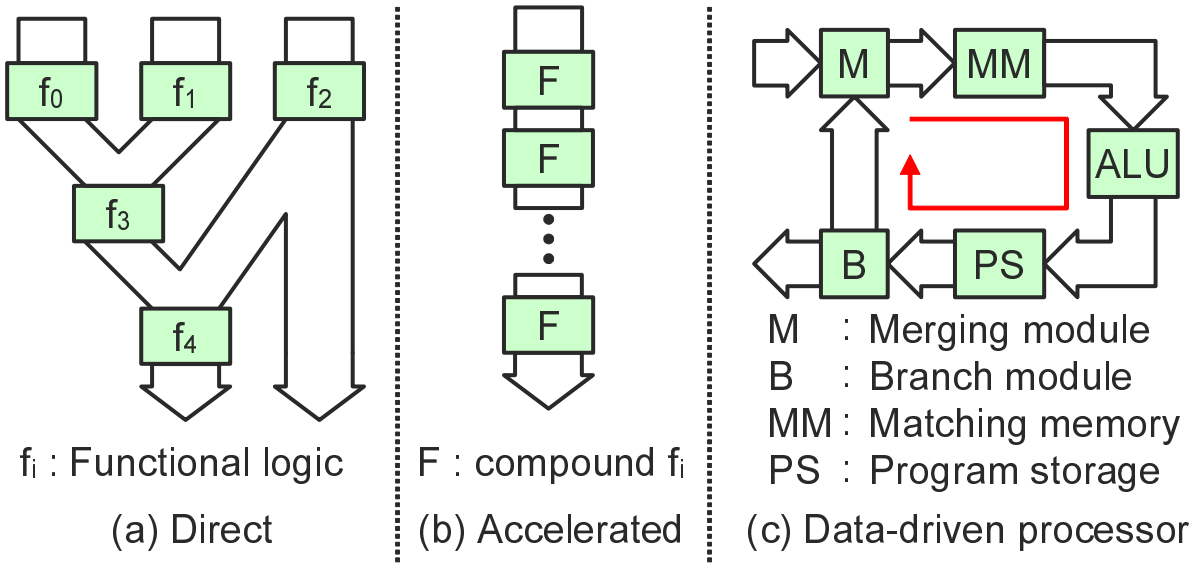

図 2.2　パイプライン処理の実現形態

例えば，図 2.1 中の PE は，写像法 (a)(b) による専用エンジン，あるいは，写像法 (c) によるプロセッサコアとして構成できる．さらに，それらをパイプライン状に接続することによって，多様な粒度のパイプライン並列処理を同時に実現することが可能になる．

これらの写像法は，典型的な形態であり，一般には，その組合せ，あるいは，折衷的な写像法を導入して，実現する必要がある．したがって，ウェブパイプライン機構では，このような写像を自由に実現するために，複数のパイプライン間で，自由にデータの流れを切り替える機能（流れ制御），および，相互のデータ系列を対象にして指定された加工を施す機能（流れ処理），ないしはその複合機能（流れ処理制御）が必須となる．

本論文では，これらパイプライン段間の流れ制御，流れ処理，および流れ処理制御を，相互作用と総称する．この相互作用を，あるパイプライン内で作用させると，データの複製や消滅が起こったかのように見え，パイプライン内のデータ流に一時的に擾乱が生じる．クロック同期方式によるパイプライン機構を用いた場合，このような局所的なイベントの影響を逐一すべて監視して，パイプライン全体の動作を制御する必要がある．これに対して，相互作用の実現に適したセルフタイム型パイプライン STP の基本構成を示す．

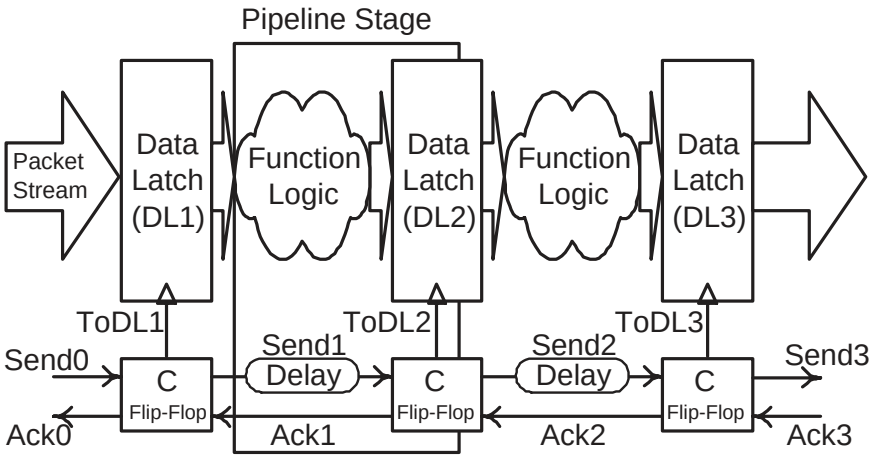

図 2.3　セルフタイム型パイプラインの基本構成

2.2.1　セルフタイム型パイプライン

セルフタイム型パイプライン STP の基本構成を図 2.3 に示す．STP は，マイクロパイプライン [9] に基づいた非同期式パイプライン・システムの構成原理である．非同期式パイプラインの構成法には，1 ビットのデータ表現に 2 つの信号線を用いる 2 線方式と，N ビットのデータ転送を 2 個の転送制御信号で支配する束データ方式がある [22]．STP では，束データ方式を採用している．束データ方式では，転送制御信号の遅延時間を，データパス中の処理回路のクリティカルパス及びデータラッチのセットアップホールド時間に一致するように設計し，データ値の正当性を保証する．このため，N ビットのデータパスを構成する場合，必要となる信号線が，2 線方式では $2N$ 本であることに対して，束データ方式では，$N + 2$ 本に収まる．したがって，例えば，単一命令と複数のデータを同時に発行して並列演算する SIMD（single instruction multiple data）などのマイクロ・アーキテクチャ構成において，必要な信号線を抑えることができ，回路コストを抑えて高い並列処理性能が得られる．

STP のパイプライン段は，データラッチ，処理回路，及び一致記憶フリップフロップ（Coincidence flip-flop: C 素子）により構成される．パイプライン段のデータ転送は，C 素子間の転送制御信号の授受（ハンドシェイク）により制御される．STP 中の各データは，

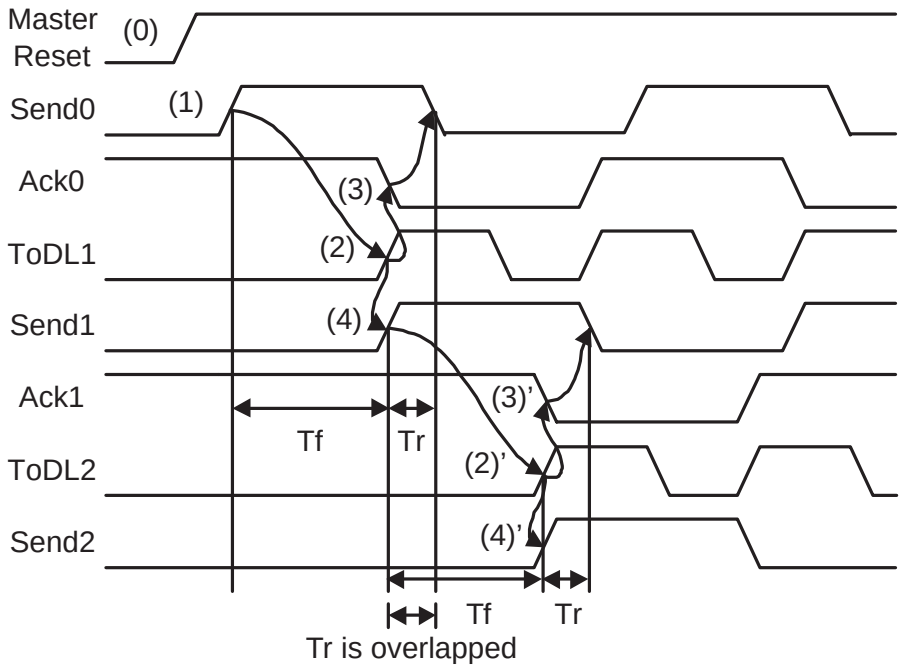

図 2.4　ハンドシェイクのタイミング・チャート

データ本体と各ステージにおける処理内容を示すタグにより構成される．これをパケットと呼ぶ．直線状の STP 内では各パケットはハンドシェイクによって自律的に移動する．ハンドシェイク信号が遷移するタイミングを図 2.4 に示す．ハンドシェイクの手順は，次の通りである．

まず，マスターリセット時には，すべての C 素子は後続ステージへ ack 信号を送信する（図 2.4(0)）．

1.（パケットの転送開始）C 素子が先行ステージへ send_{i-1} 信号を送信する．同時に，データラッチが先行ステージへパケットを送信する．
2.（ハンドシェイク）C 素子は send_{i-1} 及び ack_i 信号が到着すると，データラッチ DL_i にトリガを入れ，パケットが転送される．
3.（ack 信号遷移）同時に C 素子は，ack_{i-1} 信号を後続ステージに送信し，後続パケットの転送を許可すると共に，
4.（send 信号遷移）send_i 信号を先行ステージに送信し，先行ステージへのパケットの転

送を開始する．

5. パケットがある限り（2）～（4）を繰り返す．

厳密には，ハンドシェイクのプロトコルには，パケットの転送が完了するまでに 2 回の信号遷移を伴う 2 相式と，4 回の信号遷移を伴う 4 相式がある [23]．このうち，図 2.4 は 2 相式の信号タイミングを示している．4 相式では，1 回のハンドシェイクの間に，send 信号と ack 信号が，それぞれ 2 回遷移するが，パケットの転送手順は同じである．

STP における一連のデータ転送制御は，隣合った転送制御回路間のハンドシェイク信号の授受に局所化され，他のパイプライン段のデータ転送制御に直接的な影響を与えない．このため，隣接ステージ間のハンドシェイクのタイミングのみを守れば，パイプライン全体の動作を保証できる．よって，VLSI 設計で問題視される，タイミング制約やシグナルインティグリティの確保に有効なアーキテクチャとして期待できる．例えば，スキューの影響が各ステージ内に局所化されるため，パイプライン分割が容易になり，スループットを向上できる．また，容易に深いパイプラインを構成できるため，高い並列処理性能を達成できる．加えて，クロック制御による同期式パイプラインのように全体が同時に動作することがなく，データの有無によって間欠的に動作できるため，原理的に省電力かつ低 EMI で動作する特徴を持つ．また，STP に基づけば，前述のパイプライン段間相互作用を搭載しても，各パイプライン段が自律分散動作できるため，パイプライン全体を制御する付加的な回路は一切不要になる．

次節では，このような STP の原理的な特徴を維持したままウェブパイプラインを構成するために，複数のパイプラインの間で相互作用を可能とする，基本的なデータ転送制御回路の要件を明らかにする．

2.2.2　パイプライン間相互作用

データフロー・コンピューティングの観点から，パイプライン処理の出力は，入力データストリームを処理するパイプライン間の相互作用の結果である．通常，相互作用は，単純な

データストリームの分流と合流のみならず，例えばデータの大小関係が成立する場合のみデータを交換する，といった複雑なデータ流の制御を伴い得る．こうした相互作用の一部は既に研究されており，例えば，優先度付きキューイング機能 [24] の実現に用いられている．しかしながら，筆者の知る限り，二本のパイプライン間でさえ，すべての相互作用は明らかにされていない．本論文では，二本のパイプライン間の相互作用を，パイプライン間の（静的な）データ転送経路と，パイプライン間の（動的な）データ流制御の組合せを用いて明らかにする．

静的なデータ転送経路

STP 間の任意の相互作用には，パイプライン間の任意の経路においてデータを転送できる必要がある．パイプライン段間のトポロジーに着目すると，任意のデータ転送経路は，片方のパイプラインから他方のパイプラインへのデータの流出とその組合せにより表現できる．図 2.5 に，二本のパイプライン A と B 間のデータの流出を示す．図中，A_b と B_b はそれぞれ，データ流出のための分流点を示し，また，A_m と B_m はそれぞれ合流点を示す．ここで，前方のパイプライン段へのデータ転送経路に加えて，片方のパイプラインの出口におけるデータ転送が一時的に停止した場合であっても，パイプライン間のデータ転送を可能とするための，横方向への流出が考えられる．これらのすべてのデータ転送経路の組合せを表 2.1 に示す．データの流出の組合せによるデータ転送経路は，それぞれ，分流（Divide），合流（Merge），一方向（One-way），双方向（Bi-directional），および横方向流出付き一方向（Sideward one-way）と呼ぶ．表中では，例えば，A_b と B_b から A_m の組合せである合流（Merge）と，A_b と B_b から B_m の組合せである合流といった等価なデータ転送経路は，パイプライン A から B への経路のみを掲載することで省略している．

動的なデータ流制御

相互作用では，前述の静的なデータ転送経路の上で，処理の対象となるデータを検出する必要がある．対象となるデータ対が，事前に決定している，決定的な処理では，ストリーム間のデータ対を厳密に検出する必要がある．これに対して，対象となるデータ対が，事前に決定しておらず実行時に決まる，非決定的な処理では，すでに到着しているデータに応

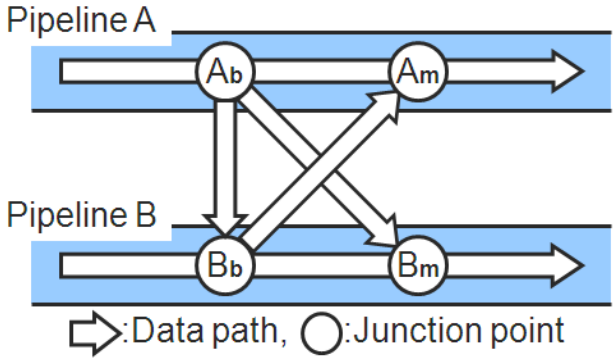

図 2.5　データ転送経路

じて，データ対を柔軟に決定する必要がある．これは，例えば，片方のデータの到着のみであっても演算が適用できる場合に，他方のデータの到着を待つことによるパイプライン処理のスループットの低下を避けるためである．

本論文では，これら決定的および非決定的な処理のための相互作用を実現するため，二つの相互作用モードを導入する．ひとつは，相互作用を行うパイプライン段において，片方のパイプライン中を流れるデータは他方のデータ対の到着を待って相互作用を適用するモードである．もうひとつは，相互作用を行うパイプライン段へ，ある定められた時間以内に到着したデータのみに相互作用を適用するモードである．前者を決定的モード，後者を非決定的モードと呼ぶ．

以上のことから，相互作用における任意の流れ制御は，データ転送経路と相互作用モードの組合せにより実現できる．相互作用においては，検出された処理の対象となるデータに対して，演算の適用，行先の設定，あるいは優先度の設定などを行う．これらは，次の情報のいずれかまたは複数に基づき決定される．

1. 対象データの値あるいは演算結果
2. データの到着順序（例：first-come-first-served）
3. パイプライン中のデータ流量
4. パイプライン段の特定の属性（例：事前に付与されたパイプライン段の番号）

表 2.1　有効なデータ転送経路 (パイプライン A から B へ)

Interaction name (based on static data path)		Active paths		
		to A_m	to B_m	to B_b
Divide	A_b	◯	◯	—
	B_b	—	—	—
Merge	A_b	—	◯	—
	B_b	—	◯	—
One-way	A_b	◯	◯	—
	B_b	—	◯	—
Bi-directional	A_b	◯	◯	—
	B_b	◯	◯	—
Sideward one-way	A_b	◯	—	◯
	B_b	◯	◯	—

◯: existent, —: nonexistent

演算の適用や行先の設定を終えた後，図 2.5 のデータ転送経路に沿って，データは転送される．この際，行先のパイプラインが同一の場合，衝突が生じる．衝突が生じた場合，上記の情報に基づき，優先的に転送するデータを決定し，排他的にデータ転送を制御する．表 2.2 に排他的に行うべきデータ転送制御を列挙する．

こうした相互作用における排他的なデータ転送を実現するには，他方のパイプラインからデータの流入がある場合，片方のパイプラインのデータ転送を一時的に停止する必要がある．転送制御が複数のパイプライン段に跨って行われる場合，制御を実現するパイプライン段は新たにデータを受け入れることはできないため，回路規模の増加のみならず，転送制御自体がパイプライン処理のボトルネックとなり得る．このため，相互作用は，対象となるパイプライン段間のみの局所的な制御により実現する必要がある．

既に，パイプライン間を接続するための転送制御回路はいくつか研究されている．一般的

に利用される，fork と join により，パイプライン間を接続する構成がある [25]．この fork と join は，それぞれ分流と合流に相当する転送制御を行う．しかしながら，これらは同一のパイプライン段で組合せて用いられることを想定していないため，例えば，双方向相互作用を制御するには複数の join と fork のパイプライン段が必要となる．また，カウンターフロー・パイプラインにおける転送制御回路 [15] は，双方向相互作用の一種を実現できる．しかしながら，パイプライン間のデータ転送経路が限られており，任意の相互作用は実現できない．筆者が知る限り，二本のパイプライン間の任意のデータ転送経路に対して相互作用を制御する転送制御回路はない．したがって，ウェブパイプラインの実現には，新たなパイプライン間相互作用制御回路が必要となる．

ここまでに，パイプライン間相互作用に基づくウェブパイプライン構成を示し，必要な転送制御を明らかにした．次章からは，パイプライン・システム設計を支援するオンチップ・シミュレーション手法とその要件を示す．

表 2.2　データ転送制御（パイプライン A から B へ）

	Transfer control
Divide	$A_b \rightarrow A_m$
	$A_b \rightarrow B_m$
Merge	$A_b \rightarrow B_m$
	$B_b \rightarrow B_m$
One-way	$A_b \rightarrow A_m$, $B_b \rightarrow B_m$
	$A_b \rightarrow B_m$
	$B_b \rightarrow B_m$
Bi-directional	$A_b \rightarrow A_m$, $B_b \rightarrow B_m$
	$B_b \rightarrow A_m$
	$A_b \rightarrow A_m$
	$A_b \rightarrow B_m$
	$B_b \rightarrow B_m$
	$A_b \rightarrow B_m$, $B_b \rightarrow A_m$
Sideward one-way	$A_b \rightarrow A_m$, $B_b \rightarrow B_m$
	$B_b \rightarrow A_m$
	$A_b \rightarrow A_m$
	$A_b \rightarrow B_b$, $B_b \rightarrow B_m$
	$A_b \rightarrow B_b$
	$B_b \rightarrow A_m$

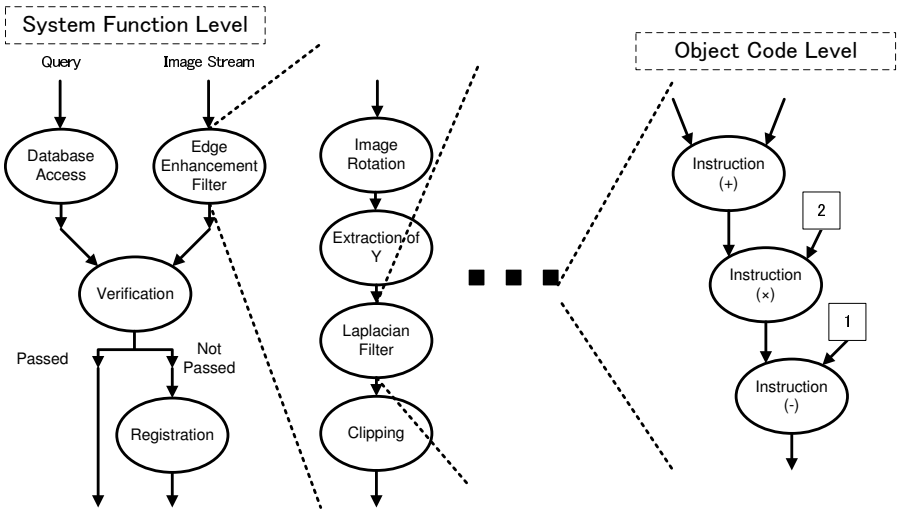

図 2.6　データフロー・グラフの階層性

2.3　オンチップ・シミュレーション手法

セルフタイム型パイプライン STP によるデータ駆動型マルチプロセッサ DDP を活用した応用システムの開発では，通常，データフロー・グラフ（DFG: data-flow graph）を用いて応用システムの機能仕様を定義する手法が有効である．

DFG は，処理内容を示すノードとノード間の依存関係を示すアークで構成されている．DFG の解釈では，データフロー・計算モデル [26] が前提となっている．つまり，データはアークに沿ってノードに向かい，ノードでは，処理内容に必要なデータが揃った時点で処理が着手される．処理が完了した後，処理結果は次ノードのデータとして再びアークを沿って次ノードに向かう．DFG では，処理（ノード）間の依存関係がアークにより明示的に示されるため，ノードは独立しており，ノード毎に階層的に詳細化できる．したがって，システムの仕様から実行コードまで，DFG による一貫した記述が可能である．図 2.6 に DFG による階層的な記述例を示す．一方，データ駆動プロセッサでは，割込み制御やスケジューリング等，本来の処理とは無関係の付加的なコードを必要としないため，詳細化により得られた実行コードを直接的に実行できる．このため，DDP システムでは，DFG を階層的に精錬化すれば，機能仕様から実行コードまでシームレスに詳細設計が可能になる．

この段階的設計の過程では，応用システムの性能要求を満たせるように，ソフトウェアで

実装する機能とハードウェアで実装する機能を切り分ける必要がある．この際，ソフトウェアやハードウェア構成を応用分野に最適化させることで，性能対コスト効率に優れたシステムを実現できる．この最適化の過程では，ソフト／ハード構成に対して，システム性能を定量化するため，ハードウェア部分上でソフトウェア部分の解釈・実行を模擬するシステム評価が必要となる．このとき，既に流通している製品チップ上で,（新規の）対象のシステムの動作が模擬できれば，定量評価に基づき応用分野の専門家が直接，命令セット，プロセッサ・コアの数や構成，さらに専用処理エンジンの有無といったアーキテクチャ項目を最適化できる．つまり，回路実装を待たずにシステムが最適化できるため，設計期間の大幅な短縮が期待できる．このようなシステム評価を実現するため，チップ上で高速にシステム全域をシミュレーションする，オンチップ・シミュレーションを示す．本論文では，模擬対象のチップをターゲット，模擬を行う既存のチップをホストと呼ぶ.

DDP システムの実効性能は，DFG 上のデータの流量で決まる．一方，データ駆動原理の下では，DFG 中の命令間にはデータの依存関係しか存在しないため，DFG 中の各命令の論理的な動作は，実行されるタイミングによらず保証される．すなわち，データの到着順序によらずに，DFG の演算結果は保証される．したがって，DDP システムの機能は，性能とは独立に評価できる．この機能と性能の独立性を活かして，オンチップ・シミュレーションでは，トレース駆動方式 [27] を採用して，命令の論理的な動作の評価と実行タイミングの評価の二段階に分けて，システムを評価する．前者を機能検証，後者を性能検証と呼ぶ. 図 2.7 に機能検証と性能検証の流れを示す.

機能検証

実アプリケーションは，いくつかの基本的な算術演算命令を中心に記述できる [28]．この点に着目し，ターゲットの DFG 中の新命令を含む各命令を，ホストの命令を用いて，機能的に等価なサブグラフに置き換える．これにより得られた DFG を，実データと共に，ホスト上で直接実行することにより，ターゲットの実行時の処理結果を得ることができる．得られた処理結果を基に，システム機能の検証を行う．このような DFG の実行は，ほぼ実行時間内で評価できるため，オフラインのシミュレーションに比べて，遥かに高速である.

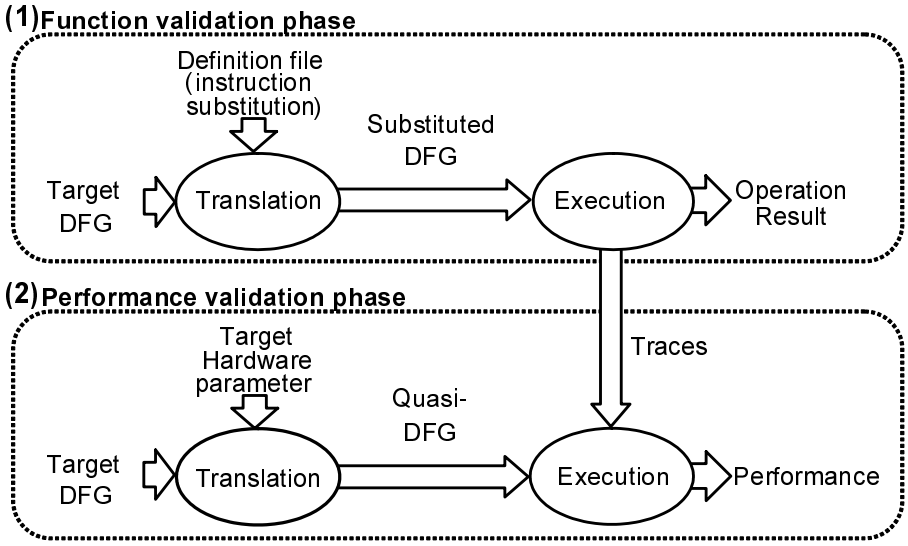

図 2.7　DDP システム検証の流れ

このホスト上での DFG の実行時に，性能を左右する実行時情報（トレース）を取得する．性能評価では，正確な性能見積りを行う必要があり，各命令の実行タイミングをターゲットとホストで一致させる必要がある．命令の実行タイミングを一致させるには，実行命令数と各命令の実行時間を一致させる必要がある．ところが，データ値に基づく分岐により実行命令数が変化し得る．また，命令が搭載されるコアのパイプライン段数やデータの転送時間に応じて命令の実行時間が変化し得る．これに対して，分岐方向と,（実行時間を区別するための）命令コードをトレースとして取得しておき，トレースに基づきターゲットの DFG を解釈・実行することで，正確な性能値を得る．

性能検証

機能検証で取得したトレースの評価は，擬似命令を用いて行う．擬似命令には，分岐方向を再現する命令と，命令の実行時間を再現する命令がある．この擬似命令を用いて，ターゲットの DFG 中の全命令を置換し，擬似 DFG を得る．この擬似 DFG を実データとトレースに基づき解釈・実行することで性能を見積り，得られた性能値を基にシステム性能の検証を行う．

このように対象命令をバイナリ変換する手法では，サイクルタイムの正確さが保証できな

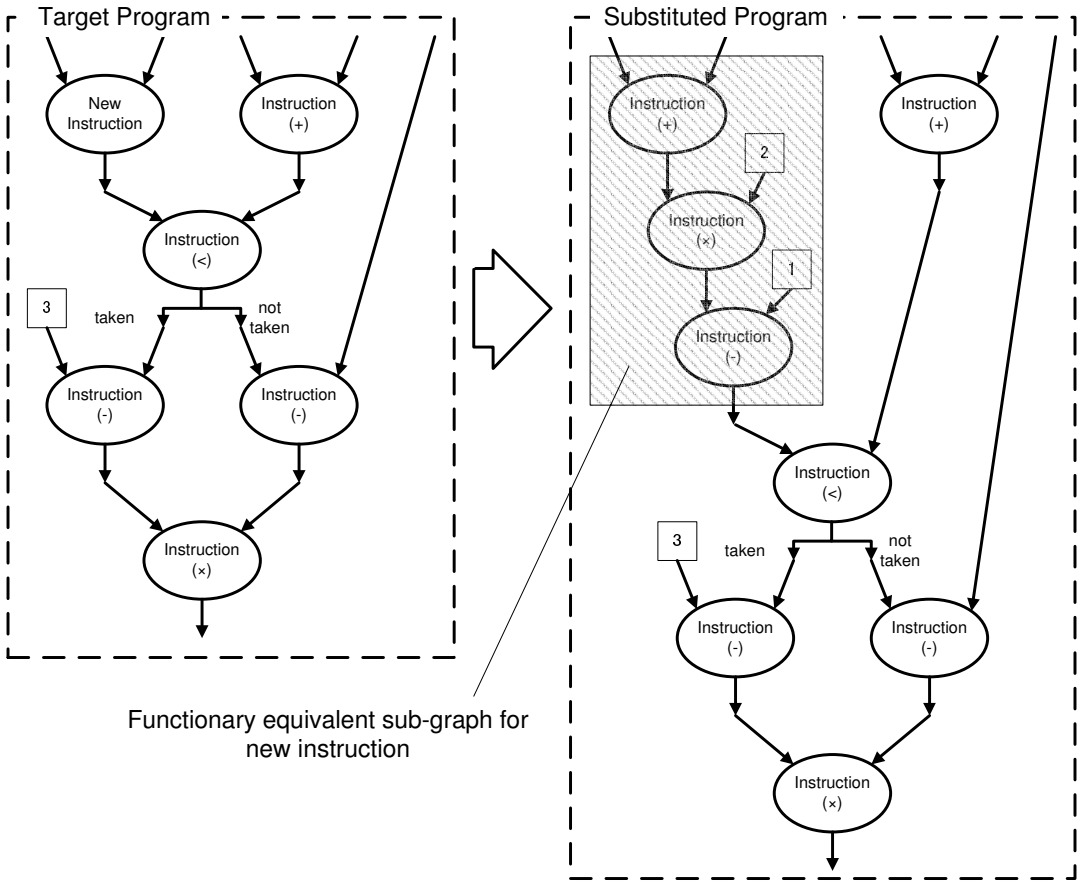

図 2.8　命令置換による対象プログラムの変換

いという問題が指摘されている [29]．この要因としては，メモリ階層の構成が性能を支配する点が大きい．これに対して，DDP システムでは DFG でプログラムが与えられるため，依存関係にあるデータが明示的に示されており，データのプリフェッチが容易に行える．このため，こうしたデータ配置の問題を容易に回避できる．一方，STP は負荷の変動に対する緩衝能力を備えており，性能検証では，この緩衝能力を正確に模擬する必要がある．

DDP システムでは，データは DFG を沿って STP 中を流れるため，命令の実行タイミングは，STP 中のデータ流量と STP のステージ数とステージ毎のデータ転送時間に支配される．このため，擬似 DFG の解釈・実行時には，STP の挙動を正確に模擬する必要がある．これに対して，従来までの，ステージ毎にデータの転送を逐一綿密に評価する手法では，STP の全ステージの状態を管理する必要があった．ところが，STP では，転送制御は

非同期に行われるため，多大な状態数を扱う必要があり，状態の保存には多大なハードウェア資源が必要となり，チップ上での模擬は難しかった．

このため，オンチップ・シミュレーションの実現には，STP の状態管理コストを低減できる，新たな動作モデルが必要となる．次節では，STP の構成と動作特性を示し，新たな動作モデルの要件を示す．

2.3.1　セルフタイム型パイプラインの動作特性

STP のハンドシェイク制御を逐一模擬すれば，STP システム全体の挙動や性能を把握できる．しかし，ステージ毎の状態変化，すなわちハンドシェイク信号の遷移を追跡する必要があり，複雑かつ高価なシミュレーション回路が必要になる．これに対して本論文では，STP 内で転送されるパケットの速度に着目して，より簡易に STP システムの挙動を把握する方法を示す．

本論文における，パケットの速度の定義を示す．

定義 1（パケットの速度（$V(t)$））あるパケットが単位時間あたりに転送されるステージ数を，$V(t)$[ステージ/秒] とする．

$V(t)$ は STP 内で転送されているパケットの疎密状況によって変動する．これは，各パケットがハンドシェイク制御により転送されるためである．以下では，この疎密状況を定量的に議論するために，パケット間の距離（$D(t)$[秒]）を定義した後，$V(t)$ と $D(t)$ の相関を説明する．

定義 2（パケット間距離（$D(t)$））ある時刻 t において，隣接する 2 つのパケット間の時間的な距離，すなわち後続パケットと先行パケットの間にある回路のクリティカルパス上の信号伝播時間を，$D(t)$ とする．

あるステージに着目すると，パケットが当該ステージを通過する最小時間であるハンドシェイク時間は，物理的な回路構成要素に基づき次のように定義される．

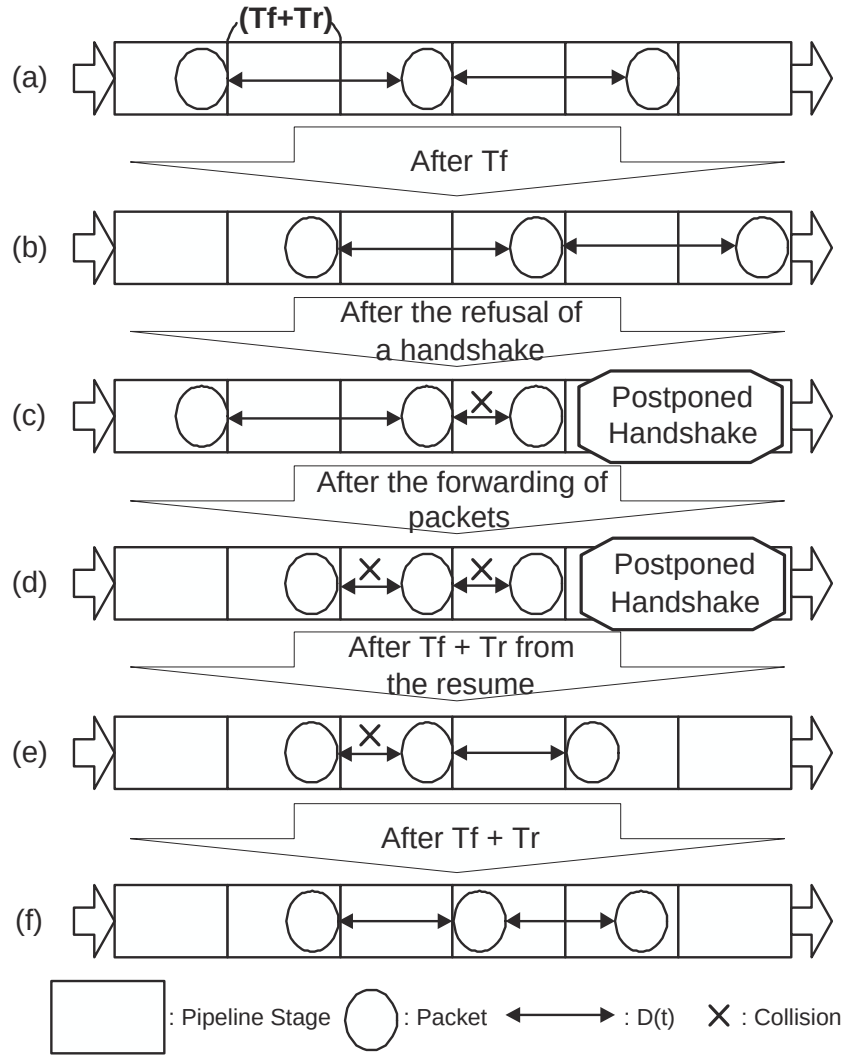

図 2.9　STP のパケット転送

定義 3（ハンドシェイク時間）$(T_f + T_r)$ は，ハンドシェイク時間とする．ここで，T_f と T_r は，それぞれ *send* 信号と *ack* 信号の伝播時間である．

図 2.9 に，$D(t)$ と $V(t)$ の関係を模式的に示す．ここでは，簡単化のため，全ステージの $(T_f + T_r)$ は同一として，$D(t)$ と $V(t)$ の関係を説明する．

$D(t) \geq (T_f + T_r)$ となる場合， send 信号の到達に先立ち，ack 信号が到着し，T_r は無視できる．この様子を，図 2.4 に示す．このため， $D(t) \geq (T_f + T_r)$ となる範囲内であれば（図 2.9(a)），各パケットは $V(t) = \frac{1}{T_f}$ で前進する（図 2.9(b)）．この様子を，図 2.9 に示す．一方，ハンドシェイクが延期され，パケットが一時的に停止する場合，後続パケットは，

send 信号の到達後も，ack 信号の到着を待つ（図 2.9(c)）．これを衝突と呼ぶ．この場合，$D(t) < (T_f + T_r)$ となり，$V(t) < \frac{1}{T_f}$ となる．

STP では，このような一時的な衝突は，自律的に緩衝される．まず，先行パケットが衝突した場合でも，後続パケットは可能な限り前進できる（図 2.9(c)∼(d)）．また，STP では，ハンドシェイクが完了すると同時にパケットは前進を再開し，1 ステージの空きがあれば処理が継続される．このように，STP には，パケット密度の増加に伴うハンドシェイクの延期により，たとえ一部のパケット群で一時的に $D(t) < (T_f + T_r)$ となっても，$D(t) \geq (T_f + T_r)$ を維持する，エラスティック能力がある（図 2.9(e)∼(f)）．したがって，一時的な衝突が $V(t)$ に与える影響も緩和される．

2.3.2 セルフタイム型パイプライン・システムの性能見積り

前節では，全ステージの $(T_f + T_r)$ が均一であると仮定したが，実際には，各ステージの $(T_f + T_r)$ は異なる．例えば，複数の STP との合流点での調停などにより，ハンドシェイクが延期される場合，これは直前のステージの T_r が伸びたと見なせる．また，設計ツールや製造環境に委ねられる回路実現工程において，設計段階で定めた平均遅延を厳密に保つのは困難であるため，$(T_f + T_r)$ にバラつきが生じる．これらのことから，実用的な STP の模擬には，ステージ毎に T_f または T_r の異なる STP をモデル化する必要がある．

このような STP では，最大の $(T_f + T_r)$ を T_{max} とおくと，パケットは自律的に $D(t) \geq T_{max}$ を維持しようとする．パケット数の増加により $D(t) < T_{max}$ となると，常に衝突が発生し，待ち時間 T_{max} を後続パケットで緩衝する．後続する個々のパケットは $D(t) - T_{max}$ だけ時間的な余裕を持っており，この余裕で待ち時間を吸収する．このため，STP へ流入するパケットに対する衝突の影響は隣接パケット間の $D(t)$ の総和，つまり，両端のパケット間の $D(t)$ で決まる．よって，衝突の影響は，個々のパケットの分布に依らず，後続パケット数によって決まる．したがって，$V(t)$ はパケット数を基準に算出でき，個々のパケットの疎密状況を追跡せずとも，STP の挙動が模擬できる．

衝突の影響は，実際の STP チップ（実 STP）では，3 段階の不連続な性能低下として現

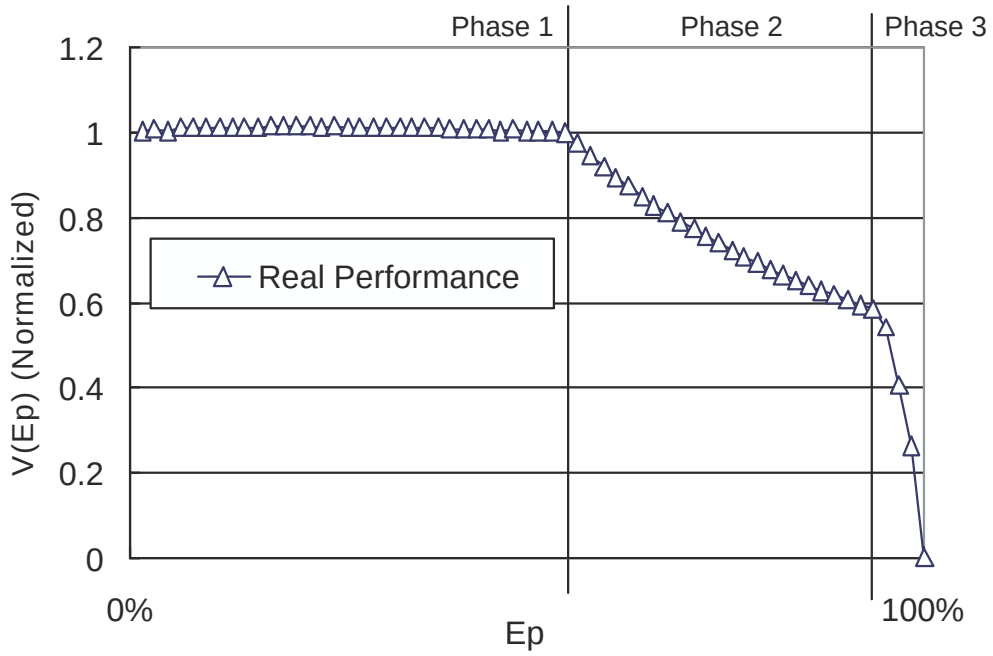

図 2.10　実システムにおけるパケットの速度 $V(t)$

れることが知られている [30]．実 STP として，STP の典型例である現行の DDP チップで負荷に応じた $V(t)$ を観測した結果を，図 2.10 に示す．グラフの横軸は負荷すなわちステージ当りのパケット数であり，縦軸は $V(t)$ の最大値を 1 として正規化したものである．$V(t)$ に基づき，性能値（スループット）を算出した結果を，図 2.11 に示す．図 2.10 及び図 2.11 共に 3 段階（図中 Phase 1 から 3）の性能変化が見られる．

このように，衝突の影響が 3 段階の不連続な性能特性として現れる主因は，DDP コアの環状のパイプライン構造において，過負荷時に衝突の影響が STP を 1 周して新たな衝突にまで波及するためである．環状のパイプライン構造は，命令の解釈・実行を担うプロセッサ・コアに不可欠である，ある命令の演算結果データを次命令の入力データとして供するための，データ・パスのフィードバック・ループを実現するためである．

図 2.11 中の Phase 2 となる状態では，過負荷時であっても衝突の影響を緩衝して，最大性能を維持できることを示している．STP システムの開発では，STP の模擬に基づく設計の試行により，ステージ数等のハードウェアコストの要求の範囲内で定常的に高い性能値を得ることが目標となる．この際，STP の緩衝能力を活用し，一時的な過負荷の状態（図中

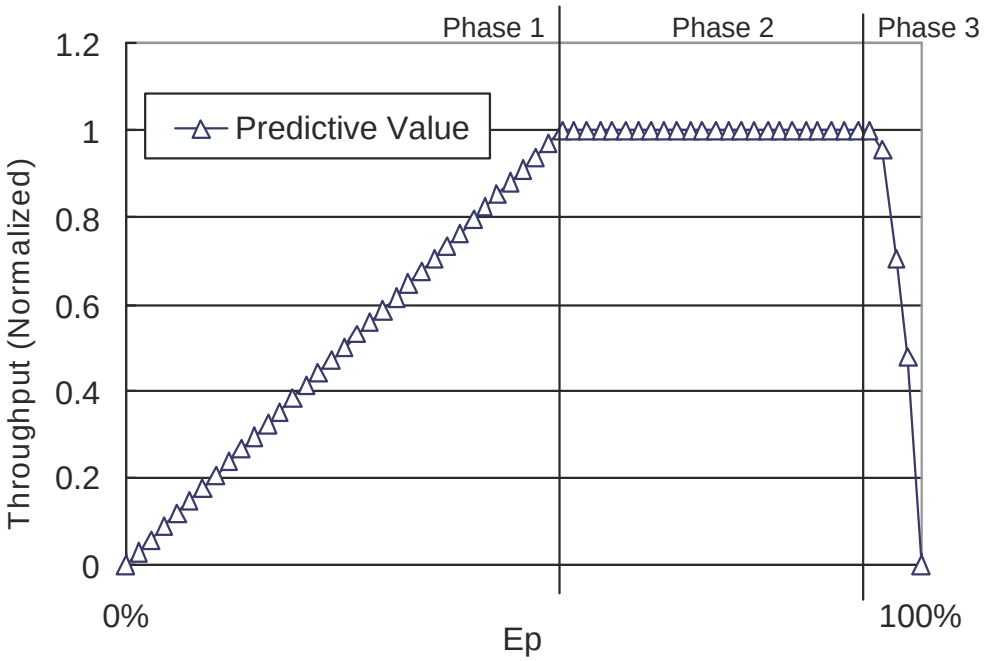

図 2.11　STP システムのスループット

Phase 2 及び 3）を許容することで，ハードウェア構成を最小化つまり最適化できる．このため，STP の模擬には，図中 Phase 3 における急激な性能低下を含む，全域的な性能変化を正確に反映できる STP のモデルが不可欠となる．

以上のことから，STP システムの性能見積りには，ステージ毎に T_f または T_r の異なる環状の STP において，$V(t)$ に対する衝突の影響を解析し，3 段階に変化する性能特性をモデル化する必要がある．

DDP コアを構成する環状の STP のパケットの速度が解析できれば，コア単位で並列したシミュレーションが可能になる．シミュレーション回路は，シミュレーション時間の短縮に大きく貢献する一方で，システム本来の計算を補助しない点で，回路規模は可能な限り抑える構成が望ましい．したがって，動作モデルに基づく並列シミュレーションの実装では，できる限り回路規模の増加を抑える必要がある．

2.4　結言

本節では，まず，大規模パイプライン並列処理を実現する，ウェブパイプライン構成を示し，ウェブパイプラインを構成するパイプライン処理の形態を分類した．各処理形態における柔軟なパイプライン構造を実現するには，パイプライン間における相互作用，すなわちデータの流れを自由に切り替えて指定された加工を施す流れ処理制御が必要である．これに対して，束データ方式に基づくセルフタイム型パイプライン STP の基本構成を示し，STP 間相互作用を実現する転送制御を明らかにし，新たに転送制御回路が必要であることを示した．

また，パイプライン・システムを迅速に評価するため，トレース駆動方式に基づく，オンチップ・シミュレーション手法を示し，シミュレーション回路規模を増大させる状態管理コストの削減には，新たな動作モデルが不可欠であることを示した．これに対して，STP 中のパケットの速度に着目すれば，ステージ毎のデータ転送を逐一模擬せずとも，STP のマクロな挙動を把握できることを示した．パケットの速度に着目することで，ハンドシェイクのプロトコルによらずに STP の挙動を示せるため，汎用性の高い動作モデルを定義できる．新たな動作モデルの要件として，STP システムでは，(a) 環状のパイプライン構造と (b) 回路実装におけるデータ転送時間 $(T_f + T_r)$ のバラつきにより，3 段階に変化する性能特性を正確に表現する必要性を示した．さらに，シミュレーション回路はシステム本来の計算を補助しない点で，回路実装において可能な限り回路規模を抑える必要があることを示した．

第 3 章

パイプライン間相互作用制御

3.1 緒言

前章では，セルフタイム型パイプライン STP の局所制御性をさらに徹底的に活用して，広大な LSI チップ上にパイプライン処理機構を蜘蛛の巣（web）のように自由に張り巡らせることによって，様々なパイプライン並列処理を実現できる，セルフタイム型ウェブパイプラインの構成法を明らかにし，基本となる二本のパイプライン間の相互作用とそれを制御する転送制御回路が必要であることを示した．ウェブパイプライン構成では，相互作用の適用に際して，例えばデータの到着タイミングやデータの値といった条件を設定すれば，有意義な機能性を持つ構成の実現が期待できる．しかしながら，STP の転送制御におけるすべての事象は因果関係にあり，非同期に遷移するすべての信号に関して，タイミング制約を満たす必要がある．このため，相互作用を実現する転送制御回路の設計において，人手による信号タイミングの保証は困難であり，将来の大規模パイプライン・システムの設計は不可能である．これに対して，本章では，パイプライン間相互作用制御回路の構成法を提案する．

3.2 相互作用制御回路

相互作用を実現するため，前章で示した静的な転送経路に沿ってデータの転送先をマルチプレクサ（MUX）で選択できるように構成し，さらに，動的なデータ流制御を実現するための相互作用制御回路により MUX を制御するよう構成する．ブロック図を図 3.1 に示す．また，相互作用制御回路では，ハザードフリーなハンドシェイクを保証できるよう，回路を

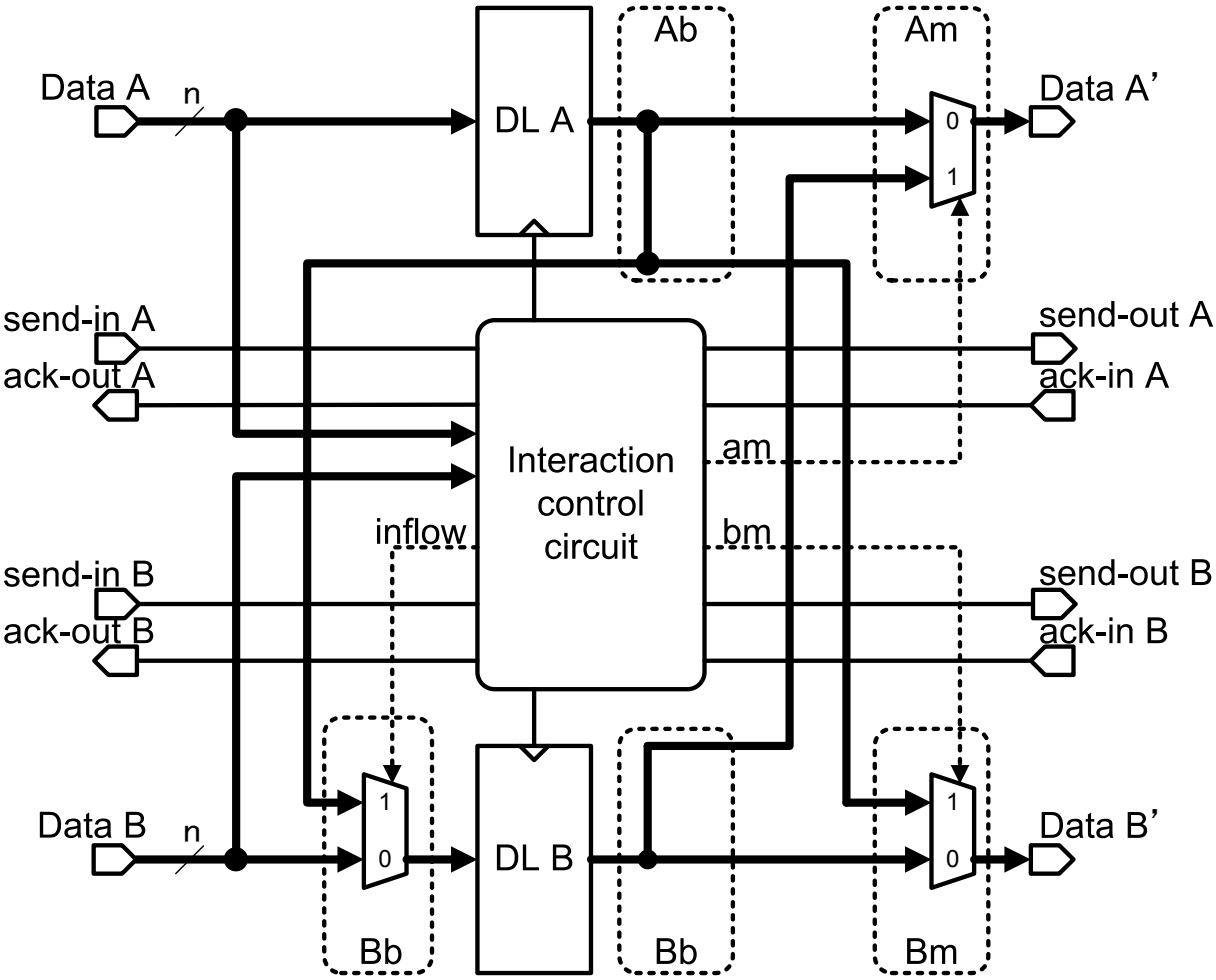

図 3.1　Interaction stage.

分割して，ハンドシェイクを実現する回路モジュールを他と独立させる．相互作用制御回路のブロック図を図 3.2 に示す.

相互作用制御回路は，C 素子，MUX，アービタ，およびルータと呼ぶ 4 つの回路モジュールから構成する．まず，C 素子によりハンドシェイクを実現し，各データラッチの開閉を制御する．静的な転送経路に沿ったハンドシェイクを実現できるよう，転送要求 send 信号と転送許可 ack 信号を MUX を用いて選択する．MUX を制御する信号は，相互作用におけるデータ流の調停を行うアービタが生成する．アービタは，ハンドシェイク信号すなわち，send-in A，send-in B，ack-in A，および ack-in B を入力として監視し，データの到着順序を検出して，転送経路を切換えるタイミングを調節する．さらに，ルータは，データの値あ

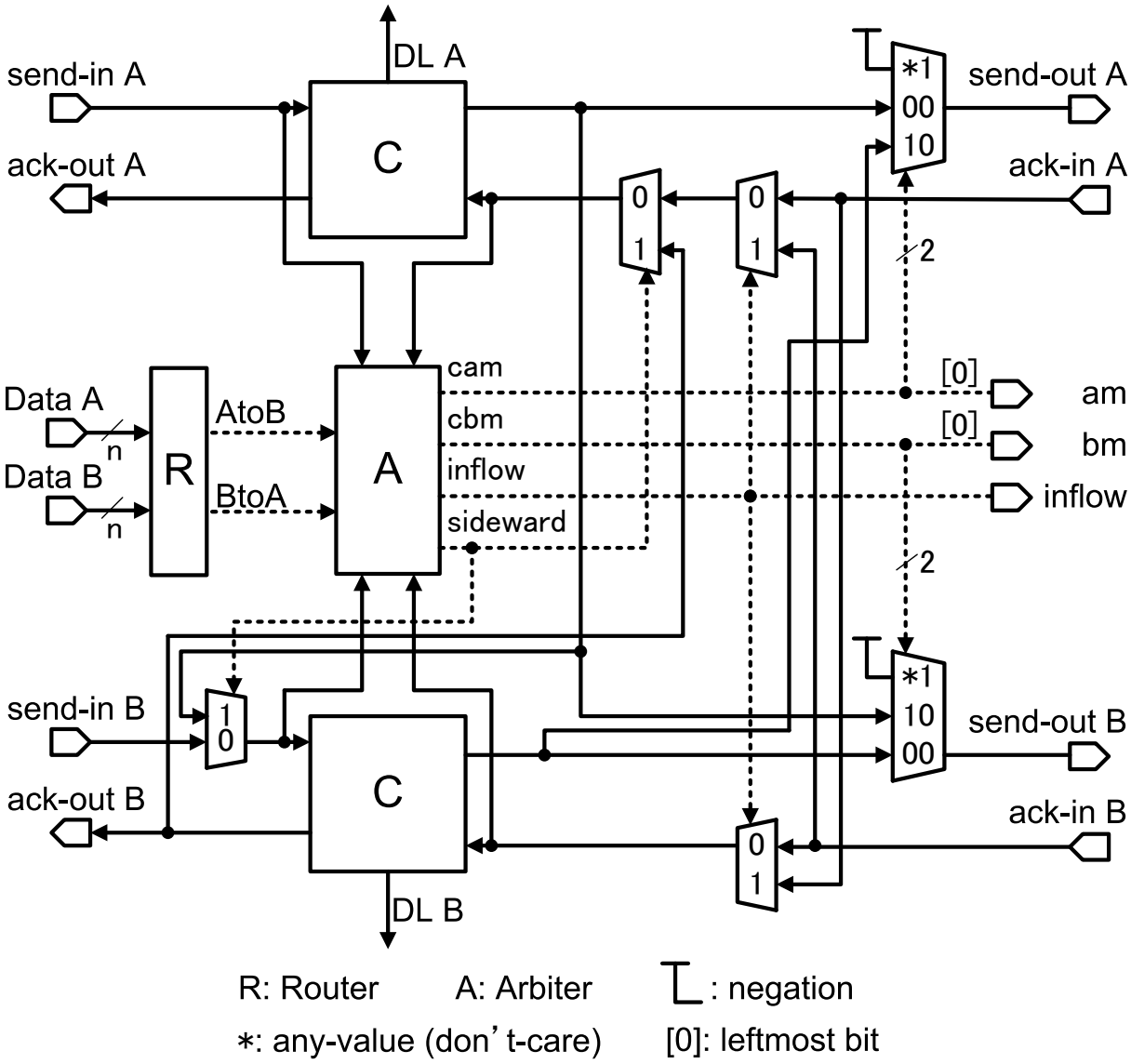

図 3.2　Interaction control circuit.

るいはデータの演算結果を基に，転送経路を検出しアービタに知らせる．図 3.2 では，アービタとルータの出力を破線で示している．これらの回路モジュールの出力信号の値を表 3.1 に示す．出力信号値の詳細は後述する．このように回路を分割し，C 素子を独立させることで，C 素子の設計に非同期回路設計手法を適用できる．図 3.1 と 3.2 に示したブロック図は，すべての相互作用を実現するための構成であり，これらは，対象アプリケーションに必要な静的な転送経路と動的な流れ制御や相互作用の条件に応じて縮退される．

C 素子（C）

STP のような非同期式回路の設計手法はいくつか研究されており，ハフマン回路に基づく方式（ハフマン回路方式）とマラー回路に基づく方式（マラー回路方式）に大別される [23]．マラー回路方式は，一般化 C 素子と呼ばれる回路素子を想定している．ところが，この一般化 C 素子は，LSI メーカーから提供される標準的な回路ライブラリ（標準セル・ライブラリ）に含まれておらず，また，トランジスタ特性のバラつきが深刻となる今後のトランジスタ微細加工技術の上で動作の保証が困難であるため，広くは用いられていない．これに対して，ハフマン回路方式は，標準セル・ライブラリのみで設計可能である．本論文では，この設計容易性の観点から，ハフマン回路方式を適用する．

ハフマン回路方式では，一種の状態マシンであるバーストモード（BM）マシンにより仕様を記述し，BM マシンから，組合せ回路と遅延素子で構成されるハフマン回路を導出する．この際，ハザードフリーなハフマン回路を導出するために，同時に変化できる入力信号数に制限がある．C 素子の send-in や ack-in の場合のように，多入力の信号変化を持つ回路の仕様記述には，拡張 BM（XBM）マシンを用いる [23]．提案した相互作用制御回路における C 素子の仕様の記述では，同時に変化する入力信号を，XBM マシンで記述できる範囲に収めるため，C 素子間のハンドシェイクに，新たな 4 相式プロトコルを導入する．導入する 4 相式プロトコルは，第 2 章で示したものと，転送要求信号（send-out 信号）をアサートするタイミングが違っている．詳細な手順は次の通りである．

1. まず，i 番目のステージの C 素子（C_i）は，C_{i+1} に対して $send-out_{i+1}$ 信号をアサートする．同時に，DL_i のゲート・オープン信号をアサートする．
2. $send-outi+1$ 信号がアサートされた後，C_{i+1} は C_i に対して $ack-out_{i+1}$ をアサートする．
3. $ack-out_{i+1}$ を受け取った C_i は，$send-out_{i+1}$ および DL_i のゲート・オープン信号をネゲートする．
4. $send-out_{i+1}$ を受け取った後，C_{i+1} は $ack-out_{+1}$ をネゲートする．同時に，もし $send-out_{i+1}$ と $ack-out_{i+2}$ が共にネゲートされていれば，C_{i+1} は，$send-out_{i+2}$

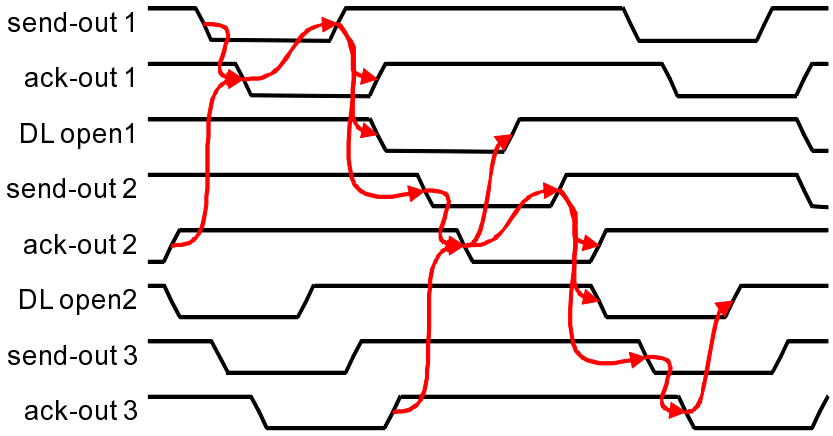

図 3.3　4 相式ハンドシェイクプロトコルのタイミングチャート

をアサートする．$ack - out_{i+1}$ を受け取った C_i は，次のパケット転送を受け入れ可能となる．

パケットの転送が終わるまで，上記の手順を繰返す．ここで，C 素子は負論理であり，アサートとネゲートは，それぞれ low（‘0’）と high（‘1’）の信号レベルを意味する．導入したプロトコルのタイミングチャートを図 3.3 に示す．図に示す通り，このプロトコルでは，$ack - out$ 信号のネゲートを待ってから，$send - out$ 信号がアサートされるため，$ack - out$ 信号の到着が遅れるとき，すなわち過負荷時には，ハンドシェイク時間が延長されて，パイプライン・スループットが低下する．しかしながら，この性能低下は，パイプラインが過負荷とならないよう，適切な負荷分散を行うことで，回避することができる．

相互作用回路の C 素子の XBM マシンは，導入した 4 相式ハンドシェイクプロトコルに基づき合成する．図 3.4 に合成した XBM マシンを示す．この XBM マシンの状態 $S0$ は，回路がリセットされた初期状態を示す．図中では，信号の立上りは+と，また，信号の立下りは-と表記している．パケットを転送する場合，信号の立上り／立下りに伴い，状態 $S1$，$S2$，および $S3$ に遷移する．また，$send - in*$ は，有効ドントケアであり，$send - in$ 信号は遷移しない，あるいは一度のみ遷移することを示している．合成した XBM マシンは，C 素子間すなわち XBM マシン間のすべての信号遷移を評価して，検証した．

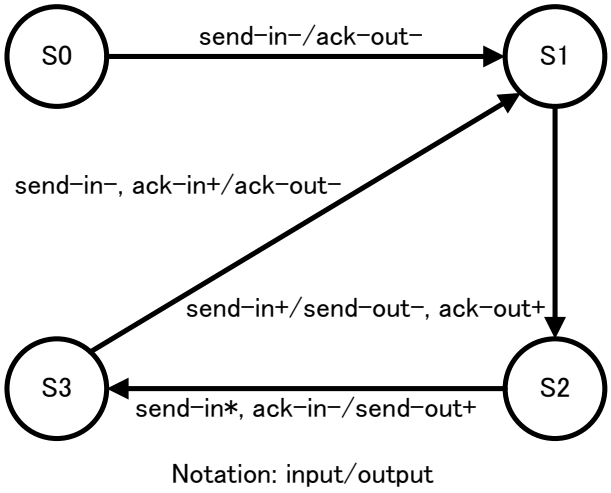

図 3.4　C 素子の XBM マシン

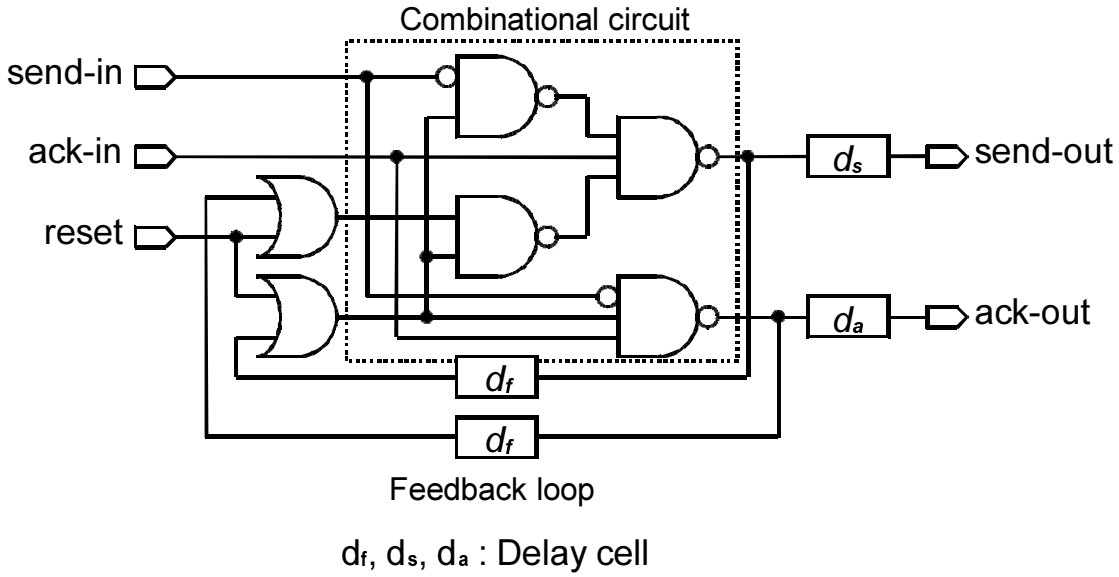

図 3.5　C 素子のハフマン回路

C 素子のハフマン回路は，ハフマン回路方式に基づき，XBM マシンから機械的に導出した．このハフマン回路の導出は定式化されているため，CAD による自動化が可能である．導出したハフマン回路を図 3.5 に示す．ハフマン回路は，次状態を決定するための組合せ回路，および状態を一時的に保持するためのフィードバック・ループにより構成される．また，d_s，d_a，および d_f は遅延素子である．それぞれの遅延量は後述する．

アービタ（A）

アービタは，ルータの出力と C 素子間のハンドシェイク信号から，データの転送経路を切換えるタイミングを決定し制御する．ルータの出力である，AtoB 信号は，パイプライン A のデータの転送経路を示しており，1 のときにパイプライン B へ，0 のときにパイプライン A への転送を意味する．同様に，BtoA 信号はパイプライン B のデータの転送経路を示す．ここで，AtoB と BtoA 信号により同じ転送先が指定された場合，衝突が発生する．この場合，アービタは，send-in 信号の到着タイミングに基づき，いずれかの転送経路を選択する．こうして選択された転送経路に基づき，MUX を制御して，転送経路を切換える．表 3.1 に MUX を制御するためのアービタの出力信号を示す．表中では，選択した転送経路毎に出力信号を示している．

非決定的な相互作用の場合，どちらかの send-in 信号の立下りを検出すれば，直ちに転送経路を決定して MUX への出力信号を決定する．その後，一連のデータ転送が終わるまで，すなわち ack-in 信号が立上るまで，出力を保持する．データ転送が終われば，転送経路を解放する．この際，データ転送中に他方のパケットが到着した場合，到着したパケットを「二番目に到着したパケット」として扱い，相互作用の条件に従って転送経路を決定する．一方，決定的な相互作用の場合，どちらかの send-in 信号の立下りを検出すれば，後方のステージへ send-out 信号が伝播しないよう，直ちに MUX を制御する．その後，他方の send-in 信号の立下りを検出した後，相互作用の条件に従って，転送経路を選択し出力を決定する．さらに，すべてのデータ転送が完了すれば，すなわちすべての ack-in 信号が立上れば，転送経路を解放する．

ルータ（R）

ルータは，データの値に基づき，データの転送先を決定する組合せ回路である．ルータの実装は，対象のアプリケーションにおける相互作用の条件に従い決定する．例えば，データの優先度を示す領域の値を用いる比較器などとなる．

表 3.1　相互作用の調停（パイプライン A から B へ）

	Route		Selected	Arbitration			
	AtoB	BtoA	transfer control	cam	cbm	inflow	sideward
Divide	0	*	Ab → Am	00	*1	0	0
	1	*	Ab → Bm	*1	10	1	0
Merge	*	*	Ab → Bm	*1	10	0	0
			Bb → Bm	*1	00	1	0
One-way	0	*	Ab → Am, Bb → Bm	00	00	0	0
	1	*	Ab → Bm	*1	10	1	0
			Bb → Bm	*1	00	0	0
Bi-directional	0	0	Ab → Am, Bb → Bm	00	00	0	0
	0	1	Bb → Am	10	*1	0	0
			Ab → Am	00	*1	0	0
	1	0	Ab → Bm	*1	10	1	0
			Bb → Bm	*1	00	0	0
	1	1	Ab → Bm, Bb → Am	10	10	1	0
Sideward	0	0	Ab → Am, Bb → Bm	00	00	0	0
one-way	0	1	Bb → Am	10	*1	1	1
			Ab → Am	00	*1	0	0
	1	0	Ab → Bb, Bb → Bm	*1	00	0	1
	1	1	Ab → Bb	*1	*1	0	1
			Bb → Am	10	*1	1	1

3.3 タイミング制約

本節では，まず，前節で提案した相互作用制御回路中のハフマン回路である C 素子モジュールのハザードフリーを保証するためのタイミング制約を導出する．ハフマン回路では，同時に変化できる入力信号の数に関する制約を満足できるように信号が変化するタイミングを保証する必要がある．

ハフマン回路は，入力信号と現状態から次状態を決定する組合せ回路，および現状態を一定期間保持するための遅延素子を持つフィードバック・ループから構成される，非同期式状態マシンである．このハフマン回路における，状態遷移を保証するには，入力信号が変化する最小時間 [sec.]（以降，d_i）が $2d_{max} + d_f$ より大きい必要がある [23]．ここで，d_{max} は，組合せ回路の入力から出力までの最大の遅延時間 [sec.] であり，d_f は，フィードバック・ループ中の遅延素子の遅延量 [sec.] である．

導入した 4 相式プロトコルに基づくハンドシェイクでは，ack-in+と send-in-は同時に起こり得る．ところが，send-in-は有効ドントケアとして定義しているため，入力として取り扱う必要がない [23]，すなわち，d_i に影響しない．この同時変化を除くと，入力変化は，send-in-から send-in+，send-in+から ack-in-，および ack-in+から ack-in-の 3 つとなる．説明を簡単化するため，着目するステージ（$stage_i$）と隣接するステージ，すなわち前方のステージ（$stage_{i+1}$）および後方のステージ（$stage_{i-1}$）は同一の相互作用制御回路を持つと仮定する．このとき，相互作用制御回路の論理ゲートを追跡すると，これら 3 つの入力変化における d_i は，すべて $2d_{min} + d_s + d_a + 3d_{mux}$ である．ここで，d_{min} はハフマン回路の入力から出力までの最小の遅延時間 [sec.]，d_{mux} は MUX の入力から出力までの遅延時間 [sec.] である．さらに d_s は，send-out 信号を遅延させるための遅延素子の遅延量 [sec.] であり，また，d_a は ack-out 信号を遅延させるための遅延素子の遅延量 [sec.] である．以上より，C 素子は次式を満たすように設計する必要がある．

$$2d_{min} + d_s + d_a + 3d_{mux} > 2d_{max} + d_f \tag{3.1}$$

また，$stage_{i+1}$ と $stage_{i-1}$ が $stage_i$ と異なる転送制御回路を持つ場合，前述した 3 つの入

力変化に対して回路を追跡して式（3.1）の対応する項目を置き換えればよい．

得られた C 素子の制約に基づき，ルータおよびアービタの制約を明らかにする．ルータは，相互作用制御回路中の他の回路モジュールからの入力を持たず，したがって，パイプラインの機能ロジック回路の一部として捉えられるため，機能ロジック回路の完了の保証と同様に，ルータの遅延時間だけ d_s を増加させれば，動作を保証できる．これに対してアービタは，send-in-を検出し，ルータの出力に従い MUX への制御信号を出力する必要がある．send-out 信号の伝播は MUX が制御するため，アービタの出力信号は，MUX に send-out-が到着するまでに決定する必要がある．すなわち，アービタの遅延量 [sec.]（以降，d_{arb}）は，send-in-から send-out-までに収まる必要がある．この send-in-から send-out-までには，ack-out-，send-in+，そして send-out-の信号変化を経る．図 3.3 に従うと，まず，send-in-は $stage_i$ の C 素子を通過して ack-out-となり，次に，ack-out-は，$stage_{i-1}$ の C 素子を通過して send-in+となる．さらに，send-in+は，$stage_i$ の C 素子を通過して，send-out-となる．すなわち，send-in-から send-out-までに，信号は C 素子を 3 回通過するため，send-in-から send-out-までの最小間隔は，$3d_{min} + 2d_s + d_a + 3d_{mux}$ となる．したがって，この最小間隔時間内に，アービタの入力から MUX の出力までの時間（$d_{arb} + d_{mux}$）を収める必要がある．以上のことから，アービタと MUX は，次式を満たすように設計する必要がある．

$$3d_{min} + 2d_s + d_a + 3d_{mux} + d_x > d_{arb} + d_{mux} \tag{3.2}$$

上述の設計制約において，各遅延量にバラつきがある場合，バラつきの範囲の最小値を左辺に，またバラつきの範囲の最大値を右辺に設定することで，例えば，プロセス・電圧・温度（PVT）のバラつきによるトランジスタの特性のバラつきに対しても動作が保証できる．また，導出した設計制約では，同期化障害のための時間的な余裕も定量化できている．アービタは，信号到着の検出や，出力を一時的に保持するために順序回路（フリップフロップ）を持つ．非同期式回路における順序回路では，入力が同時に変化した場合に出力信号の安定が遅れる，メタステーブル状態による同期化障害が知られる [23]．これに対しては，複数のフリップフロップを冗長にカスケード接続してメタステーブル状態の生起確率を実用上問

題ない程度まで低下させることができる．これに基づき，設計制約の範囲内で，フリップフロップを冗長に構成すれば，実用上の問題はない．

以上より，式（3.1）と（3.2）を満たすように回路を設計すれば，相互作用のハザードフリーが保証できる．この設計制約の導出により，回路上のすべての遅延の組合せに対して遅延を調節する必要がなくなるため，相互作用に基づくウェブパイプラインの設計は飛躍的に容易になる．

3.4　結言

本章では，セルフタイム型パイプライン STP に基づくウェブパイプライン構造の基本的な構成要素として，二本のパイプライン間の相互転送制御回路を提案し，その設計制約を明らかにした．提案した回路に基づけば，複数のパイプライン中のデータが相互に影響し合って転送経路を局所的に選択できるため，自律的かつ動的に経路を選択する柔軟なパイプライン構造を LSI 上に構成可能となる．このウェブパイプラインの特徴を生かせば，アプリケーションのデータフローを LSI 上に直接展開することで，高性能専用処理エンジンを構成することができる．

将来の大規模システム LSI において，STP の局所的な転送制御を活用すれば，処理性能の向上のみではなく，ウェブパイプライン化により並列化された処理モジュール毎の，細粒度の省電力化が期待できる．課題としては，ウェブパイプラインの応用として，コア間ネットワークなどの実用的なアプリケーションを検討し，提案回路の一般性を明らかにする必要がある．

第 4 章

シミュレーション・モデル

4.1 緒言

第 2 章では，セルフタイム型パイプライン STP の振舞いはパケットの速度で捉えることができ，また，環状のパイプライン構造を持つ DDP システムでは，負荷に応じた性能特性が 3 段階に不連続に変化することを示した．環状のパイプライン構造は，直線状の STP の出入り口を接続して構成される．本論文では，リング型 STP と呼ぶ．DDP システムでは，直線状の STP にデータ駆動処理モジュールを実装して，プロセッサ間ネットワークにより出入口を接続することで，プロセッサの環状データパスを実現している．すなわち，プロセッサ・コア（PE）はリング型 STP で構成されている．したがって，リング型 STP の動作を模擬できれば，PE 単位で並列して高速にシミュレーションできる．リング型 STP のシミュレーションでは，パケットの速度を定量化しておくことで，ハンドシェイクを逐一模擬せずともパケットの振舞いを模擬することができる．リング型 STP では，衝突の影響が新たな衝突に波及し得るため，パケットの速度は負荷に応じて 3 段階に変化する．これに対して，負荷に対するパケットの速度を定量化できれば，負荷を基準にしてシミュレーションできる．本章では，リング型 STP のモデルと，各ステージのデータ転送時間に基づき，リング型 STP 中のパケットの速度に対する衝突の影響を代数的に表現するマクロフロー・モデルを示す．提案モデルでは，不連続に変化するパケットの速度 $V(t)$ を，ハンドシェイク時間 $(T_f + T_r)$ とパケット間距離 $D(t)$ の関係から，代数的に表現する．

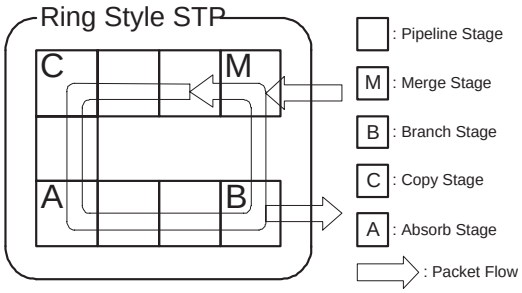

図 4.1　リング型 STP のモデル

4.2　リング型セルフタイム型パイプライン

直線状の STP を環状に結合したリング型 STP は，プログラムの実行を行う環状データパス [8] や，演算アルゴリズム中の繰り返し構造を STP に写像した ALU 回路 [31] 等への応用が可能なため，大規模 STP システムを組織的に構成する上で不可欠な基本構成の一つである．

リング型 STP は，図 4.1 に示すように，直線状の STP の出口を入口に接続して，環状のパイプライン内でパケットを転送する構成である．一般的には，2 つのパケット流を調停し合流させるマージステージ（M），及び 1 つのパケット流を選択し分流するブランチステージ（B）を配し，更にパケットの消去/複製を行うステージ（A/C）を内蔵する．この構成では，パイプライン容量が許す限りパイプライン処理を時間的に重畳できるため，高いパイプライン並列処理性能が期待できる．

このようなリング型 STP（以降，単に STP）で構成するシステムの設計では，定常的に高いパイプライン利用率を維持するように，ハードウェア構成，すなわち M・B・A・C の配置やステージ数等と，応用プログラムを設計すれば，STP の処理能力をできる限り活用できる．こうした設計に基づくシステムでは，システム内に存在するパケットの総数（以降，P_{total}）の時間的な変動は緩やかであるため，あるパケットが STP 内を一周する間は，巨視的に，P_{total} は単調に増減すると見なせる．

前章に述べたように，直線状の STP では，$D(t)$ と $(T_f + T_r)$ の大小関係によって，衝突が生じない状態（I）と生じる状態（II）がある．一方，リング型 STP では，衝突が生じる状

態には，さらに，その影響が STP のエラスティック性により解消される状態（II-1）と，解消されきれずにその影響が一周回する状態（II-2）がある．これらの STP の状態は，P_{total} に応じて遷移する．本論文では，それぞれを状態 I，II-1，および II-2 と呼ぶ．

4.3 マクロフローモデル

本節以降では，$D(t)$ と $(T_f + T_r)$ を用いて，各状態における $V(t)$ を求めると共に，各状態における P_{total} の上・下限を明らかにする．以降，STP のステージ数，ステージ i における $(T_f + T_r)$，全ステージの T_f の合計，及び全ステージの T_r の合計を，それぞれ，pl，$(T_{fi} + T_{ri})$，$\sum T_f$，及び $\sum T_r$ とおく．

4.3.1 状態 I

全ステージにおいて $D(t) \geq (T_{fi} + T_{ri})$，すなわち衝突がない状態を状態 I と呼ぶ．この状態では，STP のエラスティック能力により，$(T_{fi} + T_{ri}) = T_{max}$（$(T_f + T_r)$ の最大値）であるステージ i（以降，最大ステージ）を通過したパケット群では $D(t)$ は T_{max} 以上に保たれる．この時，各ステージでのパケットの転送時間において T_r は無視でき，パケットが一周する時間は $\sum T_f$ である．したがって，$V(t)$ は定数，

$$\frac{pl}{\sum T_f} \tag{4.1}$$

となる．

全ステージのハンドシェイク時間から $D(t) = \frac{\sum T_f + \sum T_r}{P_{total}}$ であるが，状態 I では衝突が発生せず，$\sum T_r = 0$ である．このため，図 4.2 に示すように，P_{total} は，$D(t) \geq T_{max}$ となるよう，

$$\sum T_f \geq T_{max} \times P_{total} \tag{4.2}$$

を満たす必要がある．すなわち，状態 I における P_{total} の上・下限は，

$$0 \leq P_{total} \leq \frac{\sum T_f}{T_{max}} \tag{4.3}$$

である．

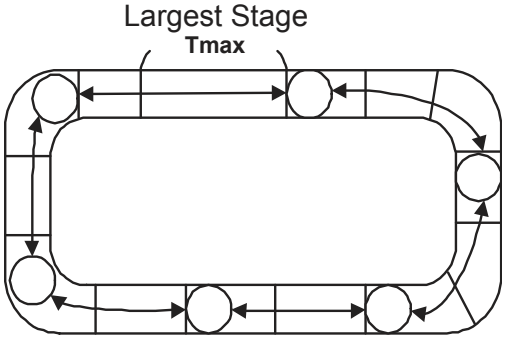

図 4.2　パケットの模式図（状態 I）

4.3.2　状態 II-1

P_{total} の増加により，式（4.3）が満たされないとき，最大ステージの通過に際し，最初のパケット衝突が発生する．この時，STP のエラスティック能力により，最大ステージを通過するパケットは，T_{max} 以上に隔てられるため，パケットは最大ステージの直前で，最大 T_{max} の間待つことになる．つまり，パケットが周回する時間は，式（4.3）の右辺を超過するパケット 1 つにつき，T_{max} 増加する．この状態を状態 II-1 と呼ぶ．したがって，状態 II-1 における $V(t)$ は，

$$\frac{pl}{\sum T_f + T_{max} \times P_{over}} \tag{4.4}$$

となる． ここで，P_{over} は，式（4.3）の上限を超過したパケット数である．

状態 II-1 では，図 4.3 に示すように，衝突は，最大ステージを起点に後方ステージに伝播する．この範囲は，各ステージが持つ時間的な余裕で決まる．最大ステージを S_1，後方のステージを順に S_2，S_3，$\cdots$，S_{pl} とおくと，S_i が持つ余裕時間は，$T_{max} - (T_{fi} + T_{ri})$ となる．よって，衝突に影響されるステージ数 n は，

$$P_{over} \times T_{max} \leq \sum_{i=1}^{n} \{T_{max} - (T_{fi} + T_{ri})\} \tag{4.5}$$

を満たす最小数となる．

状態 II-1 は，式（4.5）より，P_{over} が

$$P_{over} \leq \frac{\sum_{i=1}^{n} \{T_{max} - (T_{fi} + T_{ri})\}}{T_{max}} \tag{4.6}$$

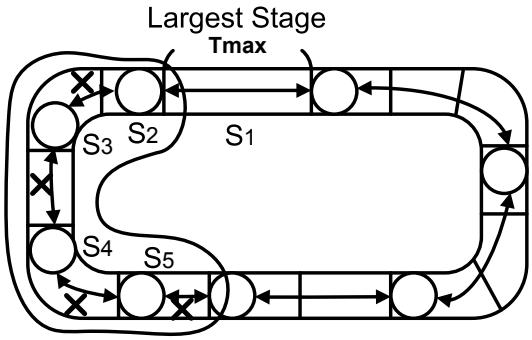

図 4.3　パケットの模式図（状態 II-1）

を満たす状態である．左辺は，式（4.3）より，

$$P_{over} = P_{total} - \lfloor \frac{\sum T_f}{T_{max}} \rfloor \tag{4.7}$$

となる．右辺は衝突が一周する $n = pl$ のとき最大で，

$$\begin{aligned} &\frac{\sum_{i=1}^{pl}\{T_{max} - (T_{fi} + T_{ri})\}}{T_{max}} \\ =&\frac{\sum_{i=1}^{pl} T_{max}}{T_{max}} - \frac{\sum_{i=1}^{pl} T_{fi} + \sum_{i=1}^{pl} T_{ri}}{T_{max}} \\ =&pl - \frac{\sum T_f + \sum T_r}{T_{max}} \end{aligned} \tag{4.8}$$

となる．式（4.6）に式（4.7）と式（4.8）を代入すると，

$$\begin{aligned} P_{total} - \lfloor \frac{\sum T_f}{T_{max}} \rfloor &\leq pl - \frac{\sum T_f + \sum T_r}{T_{max}} \\ \lceil \frac{\sum T_r}{T_{max}} \rceil &\leq pl - P_{total} \end{aligned} \tag{4.9}$$

となる．すなわち，状態 II-1 における P_{total} の上限は，

$$P_{total} \leq pl - \lceil \frac{\sum T_r}{T_{max}} \rceil \tag{4.10}$$

である．

4.3.3　状態 II-2

さらなる P_{total} の増加により，式（4.10）が満たされないとき，図 4.4 に示すように，各ステージに衝突を緩衝する余裕がないので，衝突の伝播は一周回し，最大ステージでの新

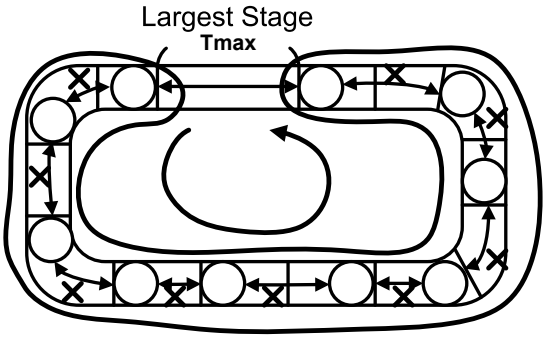

図 4.4　パケットの模式図（状態 II-2）

たな衝突にまで波及する．すなわち，一旦発生した衝突は STP を周回し続け，すべてのパケットがステージを前進するたびに衝突する．この状態を状態 II-2 と呼ぶ.

状態 II-2 では，全ステージで衝突が発生しており，パケットはパイプライン・バブルとすれ違わない限り前進できない．STP 中のバブルの総数を B_{total}（$= pl - P_{total}$）とすると，パケットは，バブルの一団と 1 回すれ違うことで，B_{total}[ステージ] だけ前進できるから，パケットが STP を 1 周するには，$\frac{pl}{B_{total}}$ 回だけバブルの一団とすれ違うことになる．また，1 回すれ違うために，バブルの一団は，$pl - B_{total}$[ステージ] だけ前進する必要がある．これらのことから，パケットが STP を 1 周する時間は，バブルの一団が STP を $\frac{pl}{B_{total}} \times (pl - B_{total}) \div pl = \frac{pl - B_{total}}{B_{total}}$ 回だけ周回する時間で規定できる．ここで，バブルが STP を 1 周する時間は，$\sum T_r$ である.

以上より，状態 II-2 における $V(t)$ は,

$$pl \div \frac{(pl - B_{total}) \times \sum T_r}{B_{total}} = \frac{pl}{\sum T_r} \times \frac{pl - P_{total}}{P_{total}} \tag{4.11}$$

となり，P_{total} に関して定義できる.

状態 II-2 における P_{total} は，pl 未満であるため，上限は,

$$P_{total} < pl \tag{4.12}$$

である.

以上より，リング型 STP の性能が，P_{total} に関する 3 つの状態において，$V(t)$ を用いて統一的に扱えるマクロフローモデルが定義できる.

4.4 実システムのシミュレーション

マクロフローモデルでは，単調に増減する P_{total} に関して $V(t)$ を定めた．一方，実システムでは，M・B・A・C の動作により，P_{total} が増減する．したがって，シミュレーションでは，P_{total} が変化した時点で，STP 中のパケットの $V(t)$ を再設定する．すなわち，P_{total} に変化がある時刻にイベントを発生させ，イベント内で，P_{total} に応じた $V(t)$ に基づきパケットを移動させて，P_{total} が変化する時刻に新たにイベントを発生させる，イベント駆動シミュレーションを採用する．これにより，動的に負荷の変動する実システムの振る舞いを，パケットの転送タイミングを逐一追跡する従来の手法に比べて，高速に模擬できる．

また，実システムにおいて，合流 M では，M ステージへの send 信号の到着が遅い方のパケットの転送が，最大で 1 パケット分の転送時間だけ遅れる．さらに，複製 C では，ラッチしたパケットを前方ステージに連続して 2 回転送することで，1 パケット分の転送時間の遅延のみで複製が完了する．これらの遅延の影響は，後続パケットを遅延させ，M ／ C ステージから離れるほど小さくなる．これに対して，M ／ C を持つ構成のシミュレーションにおいては，常に P_{total} を $P_{total}+1$ と見なして，1 パケット分の転送時間だけ全パケットを遅延させる．これにより，付加的な処理なく，M ／ C の影響を最大に見た安全側の見積りが期待できる．

4.5 パラメータ設定法

マクロフローモデルに基づくシミュレーションでは，pl と各状態における $V(t)$ および P_{total} の上限値が必要である．ここで，状態 I における P_{total} の上限値および状態 II-1 における P_{total} の上限値を，ぞれぞれ，P_{IU} および P_{II-1U} と呼ぶ．これらのパラメータは，システムの設計時に見積られた，pl と，$(T_{fi}+T_{ri})$ から算出する．つまり，システムの機能をパイプライン分割する際に見積られる論理ゲートや配線の遅延に基づき，T_{fi} と T_{ri} を決定し，パラメータ値を求める．

マクロフローモデルは，P_{IU}，P_{II-1U}，及び $V(t)$ は，pl，$\sum T_f$，$\sum T_r$，及び T_{max} か

ら導出できることを示している．すなわち，$(T_{fi} + T_{ri})$ の組合せに対して，すべての組合せではなく，pl，$\sum T_f$，$\sum T_r$，及び T_{max} が変化する場合のみパラメータを設定して，シミュレーションを行えば十分である．よって，各ステージ毎にランダムなバラつきが想定される場合でも，効率的に性能見積りが行える．

P_{IU} と P_{II-1U} は，それぞれ式（4.3）と式（4.10）より，

$$P_{IU} = \lfloor \frac{\sum T_f}{T_{max}} \rfloor, P_{II-1U} = pl - \lceil \frac{\sum T_r}{T_{max}} \rceil \tag{4.13}$$

であり，設計値により定まる．また，$V(t)$ は，式（4.1），式（4.4），及び式（4.11）より，設計値により定まる．

4.6　結言

本章では，パケットの平均的な速度 $V(t)$ に着目して，ステージ毎のハンドシェイク時間 $(T_{fi} + T_{ri})$ との関係を，総パケット数 P_{total} を基準にして代数的に規定した．これにより，P_{total} に応じた $V(t)$ を管理するのみで，パケットのフローを近似可能とした．また，$(T_{fi} + T_{ri})$ のバラつきを見込んだ $V(t)$ を予め設定することで，設計の初期段階から軽量かつ安全なシミュレーションが可能となる．提案モデルを用いれば，$V(t)$ に応じてパケットの速度を動的に調整さえすれば，全てのパケットのフローを追跡できるため，シミュレータの実装を軽量化できる．また，STP においてパケットの流量に応じて変動する，消費電力量などの動的なシステム性能の見積りへの応用も期待できる．課題としては，パイプライン間相互作用に基づくウェブパイプラインによる柔軟なパイプライン構造のモデル化が残されている．

第 5 章

オンチップ・シミュレーション手法

5.1 緒言

本章では，オンチップ・シミュレーションを実現するためのセルフタイム型パイプライン STP 回路を示す．第 2 章で示したとおり，STP システムの性能検証は，STP 中のパケットの転送タイミングの模擬と，転送タイミングを支配するデータ駆動プログラムの評価からなる．

転送タイミングの模擬は，対象のチップ（ターゲット）のパイプライン構造を基に，パラメータを調節して，シミュレーションを実行するチップ（ホスト）上でマクロフロー・モデルに基づくシミュレーション（本論文では，マクロ・シミュレーションと呼ぶ.）を実行することで達成できる．一方，ターゲットとホストのパイプライン構成において，ステージ数が一致している場合，ステージ毎の転送時間をターゲット／ホスト間で一致させるのみで，タイミングが模擬できるため，シミュレーションの簡略化が期待できる．本論文では，前者を単にシミュレーション，後者をエミュレーションと呼ぶ.

シミュレーション及びエミュレーションでは，ターゲットのパケット・フローを模擬して，ターゲット／ホスト間のパケット流量の変動を一致させる必要がある．このため，データ駆動プログラムの解釈・実行が必要となり，オンチップ・シミュレーションのボトルネックとなり得る．これに対して，命令の解釈・実行を STP に実装した DDP コアのパイプライン構成を直接的に活用すれば，パケットの転送タイミングの模擬と，データ駆動プログラムの評価を同時に行うことによる，シミュレーション時間の短縮化が期待できる.

本章では，DDP コアの構成を示し，データ駆動プログラムの解釈・実行がシミュレーショ

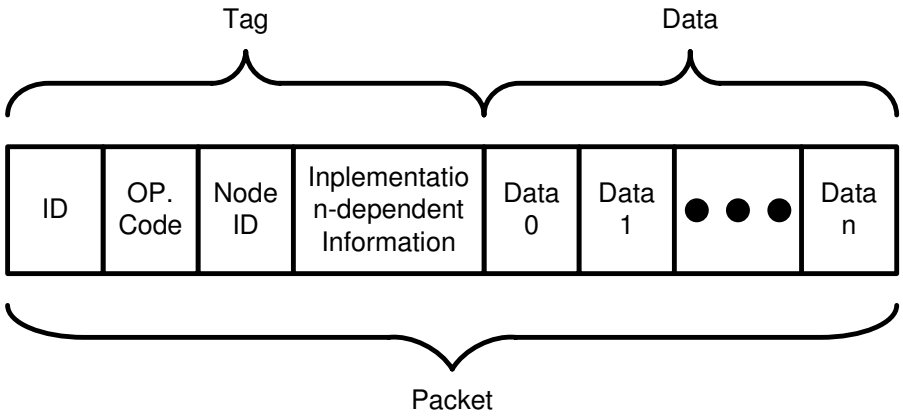

図 5.1　DDP パケットの構成

ンのオーバヘッドにならないよう負荷に応じてパケットの転送時間を動的に調整する機構を追加した，シミュレーション／エミュレーション回路を示す.

5.2　データ駆動プロセッサ・コア

現行の DDP では，データフロー・グラフ DFG の同一アーク上に複数のトークン（パケット）の存在が許容される動的データ駆動方式を採用しているため，命令が処理可能（発火可能）になれば直ちに実行（発火）でき，同時実行可能な命令数に応じて空間的な並列性が自然に抽出できる．動的データ駆動方式では，複数のトークンを識別するため，トークン毎にカラーを付与する必要がある．これに対して，DDP では，データに加えて，データの識別子となるタグを付加して，各パケットを構成する．図 5.1 にパケットの構成を示す．タグには，識別子に加えてオペ・コードと，次ノードを示すノード識別子を持つ.

DFG の解釈・実行は，ノード単位で行われる．ノード毎の一連の実行は，パケットの待合せ，演算実行，次命令フェッチの 3 つの過程を経る．すなわち，まず，演算対象のパケットを待ち合わさせ，すべてのパケットが揃った時点で，演算パケットを生成する．次に，演算パケットに対して，オペ・コードに対応する演算を適用して演算結果を得る．さらに，DFG の接続情報に基づき，次命令のオペ・コードと次ノード識別子を更新する．図 5.2 にこれらの過程を示す.

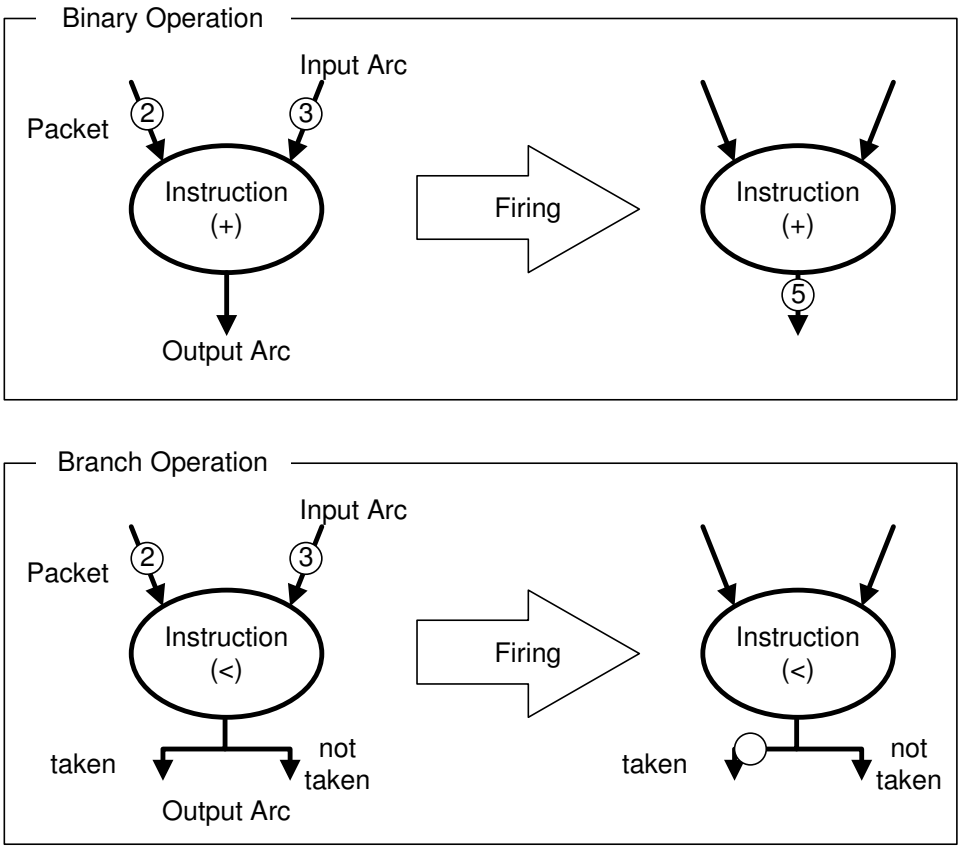

図 5.2　データ駆動処理方式

DDP コアは，これらの，データ駆動原理に基づき DFG を解釈・実行するための，パケットの待合せ・発火，演算実行，及び次命令フェッチといった処理モジュールを実装した STP で構成される．DDP コアのブロック図を図 5.3 に示す．この構成では，各パケットは処理コンテキストを持って STP 中を転送されるため，複数のプログラムを同時に実行する場合に，パイプライン・インタロック等による明示的なコンテキスト・スイッチが不要であり，したがって高いパイプライン利用効率が期待できる．また，動的データ駆動方式により抽出された空間的な並列性は，STP 上で時間的（パイプライン的）に重畳されるため，各種粒度の並列性に応じて高いパイプライン処理性能を発揮できる．

各 DDP コアは，データを分流／合流するモジュール（B/M）により出入り口を接続してリング型の STP を形成している．また，多段パケット・ルータなどのプロセッサ間相互結合網により，コア間を結合することで，マルチコア構成が実現できる．図 5.4 に，マルチコ

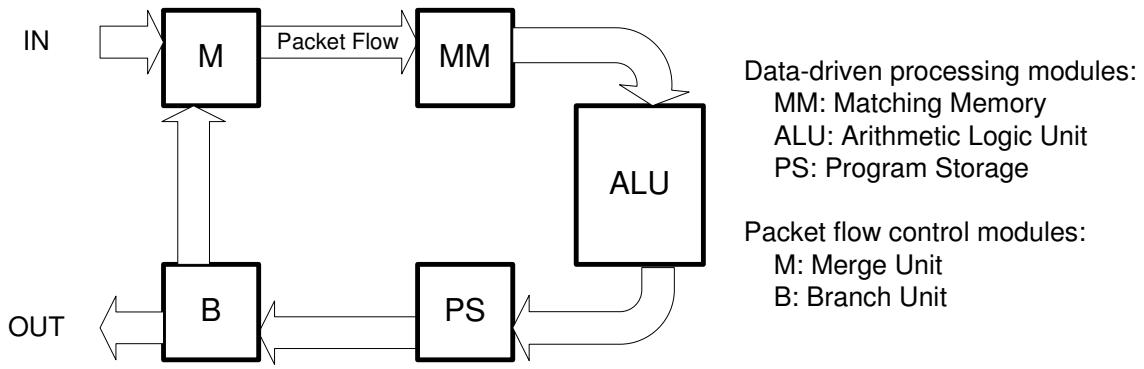

図 5.3　データ駆動プロセッサ・コア

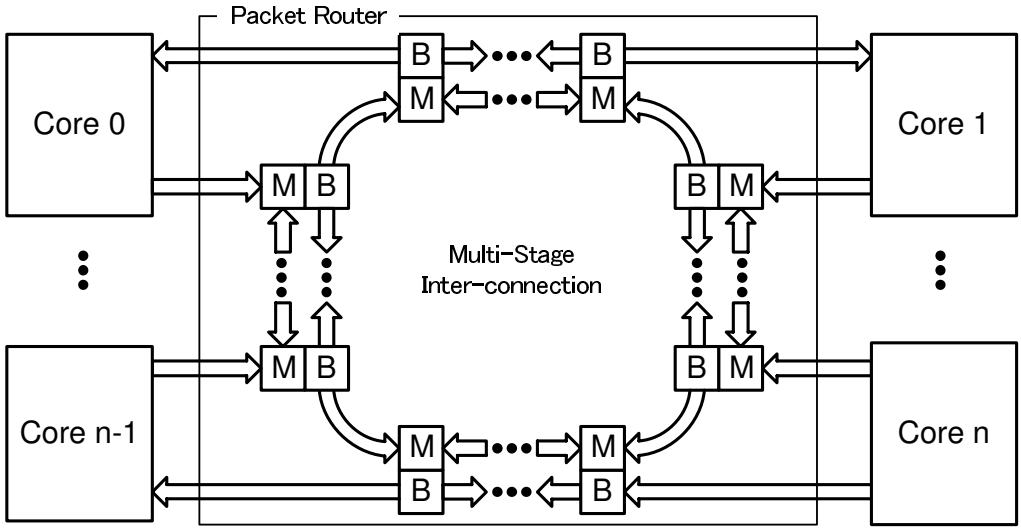

図 5.4　マルチコア構成

ア構成のブロック図を示す．このような構成では，DFG の 1 ノード（命令）の実行は，リング型 STP をパケットが 1 周することに相当する．したがって，性能検証には，リング型 STP をパケットが 1 周する時間，すなわち周回時間を模擬する必要がある．

これに対して，エミュレーションでは，ホスト上でステージ毎のパケットの転送時間を動的に調節することで，振舞いをターゲットと一致させることができる．一方，マクロ・シミュレーションでは，パケットの速度 $V(P_{total})$ に基づいて周回時間を動的に調整する．このためには，パケットの流量の増減を正確に追跡する必要がある．

パケット流量は，パケットの入力，出力，消去，及び複製により変動する．現行の実装で

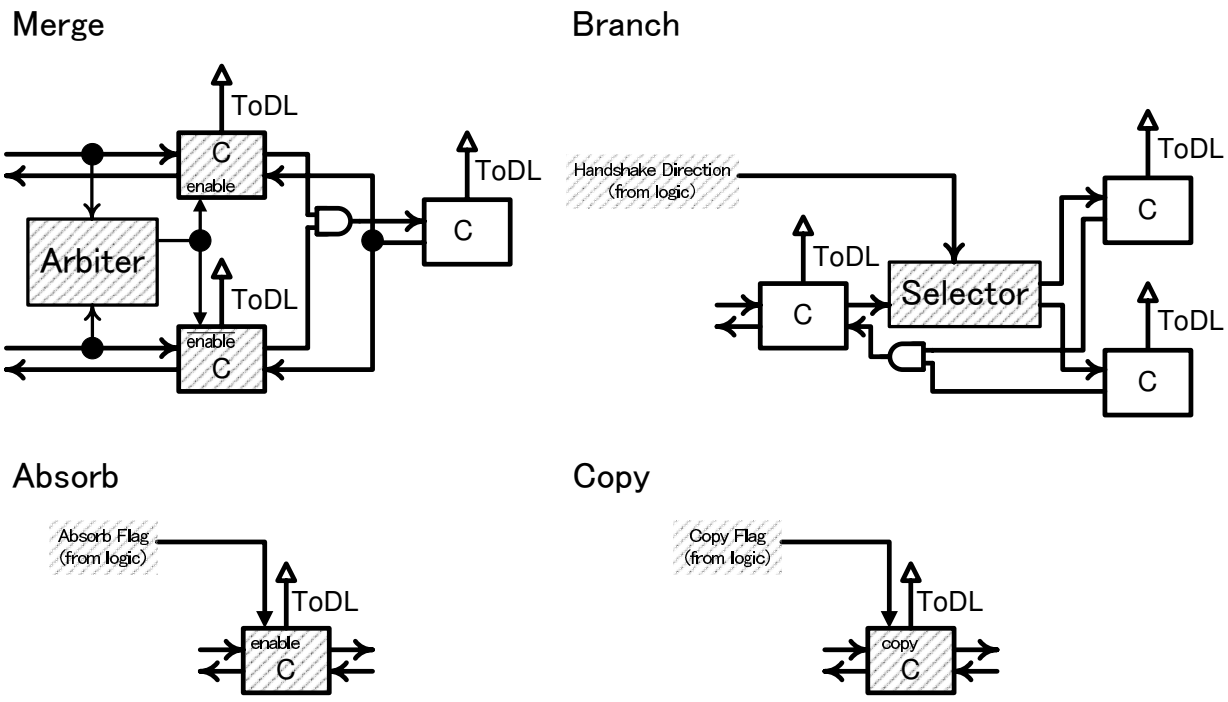

図 5.5　M・B・A・C の回路構成

は，これらは，それぞれ，合流 M，分岐 B，消去 A，及び複製 C の機能を持った転送制御回路が，データ駆動処理モジュール側（データパス）からの制御信号に応じて機能することで実現されている．図 5.5 に，データパスと転送制御パスのブロック図を示す．また，各転送制御回路の動作を次に示す．

M の動作

外部からパケットの入力と内部からのパケットの転送を調停する．調停は，send 信号の到着順序に基づいており，M ステージへの send 信号の到着順序が早いパケットを後続ステージへ転送し，send 信号の到着が遅い方のパケットの転送を最大で 1 ステージ分遅らせることで合流を実現している．

B の動作

経路信号に基づき，パケットを指定された方向に転送する．経路信号に対応する後続ステージとのみハンドシェイクを行う．これにより，排他的な経路にパケットが転送され，分流が完了する構成である．

A の動作

消去信号に基づき，パケットを消去する．後続ステージとのハンドシェイクを取り止める．これにより，ラッチしたパケットは後続するパケットに上書きされ，消去が完了する構成である．

C の動作

複製信号に基づき，パケットを複製する．後続ステージとのハンドシェイクを連続して 2 回行う．これにより，ラッチしたパケットが後続ステージに連続して 2 回転送され，1 ステージ分の遅延のみで複製が完了する構成である．

このような構成により，プログラムから明示的な複製や消去ができるため，定常的に高いパケット流量の確保が容易化される．

このような構成では，パケット流量の変動を正確に追跡するためには，プログラムの解釈・実行に基づいて，適切なタイミングで入力，出力，消去，及び複製を模擬する必要がある．ところが，第 2 章で述べたとおり，プログラムの解釈・実行には，トレースに基づく擬似 DFG の実行・解釈が必要であり，シミュレーションのオーバヘッドとなり得る．

これに対して，トレースを評価する機構と，動的なパケット数を計測するモジュールを DDP コアに搭載し，擬似 DFG を直接的に実行・解釈すると同時に，周回時間を調整する手法を示す．

5.3　トレース駆動エミュレーション手法

前章で述べたとおり，STP の性能はパケット流量で決まる．DDP 構成では，パケット流量を変化させるアーキテクチャ項目は，命令セットに関係するものとハードウェア構成に関係するものに大別できる．前者は，ターゲットに搭載する命令，演算に対する待合せ・発火ポリシーであり，後者は，STP の総ステージ数，M・B・A・C の機能とステージ位置，ハンドシェイク時間である．

これらのアーキテクチャ項目がターゲットとホスト間で異なる場合，マクロ・シミュレー

ションによる模擬が必要となる．一方，新システムの設計において，命令の追加や変更では，変化するアーキテクチャ項目が限定される場合があり，この場合，パケット流量を一致させるための追加回路を簡略化することができ，すなわちマクロ・シミュレーションによらずとも，直接的なターゲットの模擬が可能である．

パケットは DFG に沿って STP 内をフローするため，対象プログラムと擬似 DFG の構造（接続関係）を一致させる必要がある．しかし，命令の追加／変更では，命令置換において，機能的に等価なホストの命令がない場合がある．これに対して，新規命令を意味する擬似命令で置換することで，対象プログラムの構造と実行命令数を保存する．置換後に得られたプログラムを，擬似プログラムと呼ぶ．擬似プログラムでは，新規命令の機能はホストに実装されていないため，DFG 上のパケットの値を保証することができない．このため，値分岐を行う分岐命令の分岐先の実効命令数が同一でない場合，パケット流量が変動し得る．図 5.6 ／ 5.7 に，値分岐によりパケット流量が変動する／しない DFG を例示する．図 5.6 では，分岐命令に続くノード数が，すべての分岐方向で同じである，すなわち，実行命令数が分岐方向によらずに一定であるため，パケット流量は変化しない．一方，図 5.7 では，分岐先で実行命令数が変化するため，パケットが STP を周回する回数が変化し，パケット流量が変化し得る．このように値分岐によりパケット流量を変化させ得る分岐命令も擬似命令で置換しておき，分岐命令の実行時に，値に関係なく，対象プログラムにおける分岐方向を設定すれば，擬似命令はあたかもターゲット上の新規命令あるいは分岐命令として機能する．図に 5.8 擬似命令を用いて，対象プログラムから擬似プログラムを得る流れを示す．

また，新規命令の追加では，例えば，メモリに対してデータ読み出し，演算，書込みといった一連の操作が頻出する場合，これらの操作を 1 命令で行う複合命令の追加により，大幅な性能改善が見込める場合がある．このとき，メモリ容量の変更などで，データの読み書き時のアクセス・レイテンシが伸縮する場合，データの読み書きが保証できるように，ハンドシェイク時間を伸縮する必要がある．以上のことから，擬似命令の実行時に，分岐方向の保証と，ステージ毎のハンドシェイク時間を調節すれば，パケット流量は一致する．

ハンドシェイク時間の調節では，周回時間を調節するパケットを特定する必要があるため，

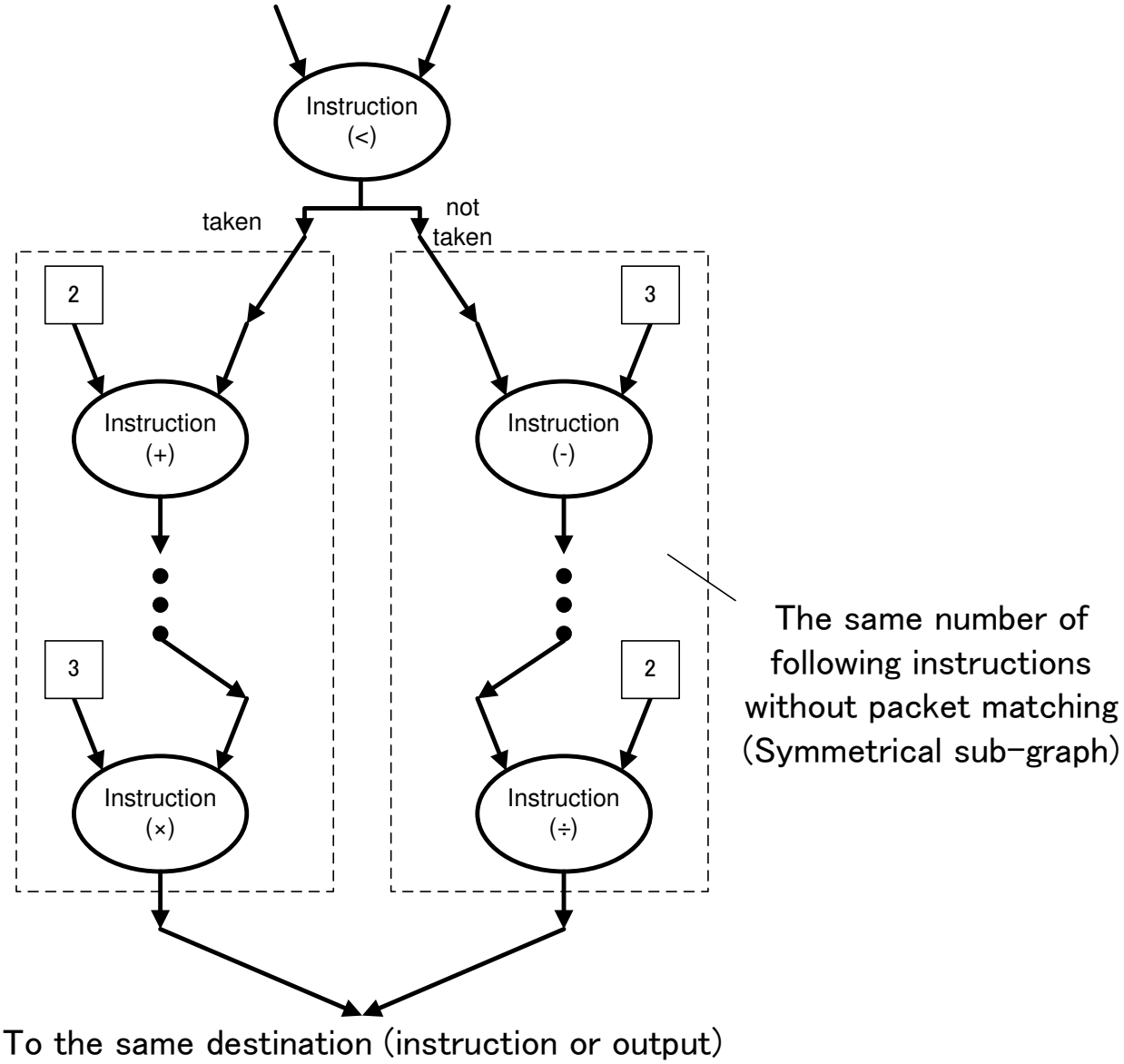

図 5.6　値分岐によりパケット流量が変化しない例

トレースとして，命令名が必要となる．また，分岐方向の保証では，分岐方向に加えて，設定対象のパケットを特定するために，ノード識別子とパケットの識別子が必要となる．これらのトレースは，DDP パケットが処理コンテキストを保持していることを活用すれば，実行時に容易に取得できる．すなわち，パケットが分岐命令を通過した直後にパケット中のタグ情報から得ることができる．これに基づき，機能検証において，置換プログラムにトレースを取得するための命令である，記憶命令を追加することに，トレースを取得する．図 5.9

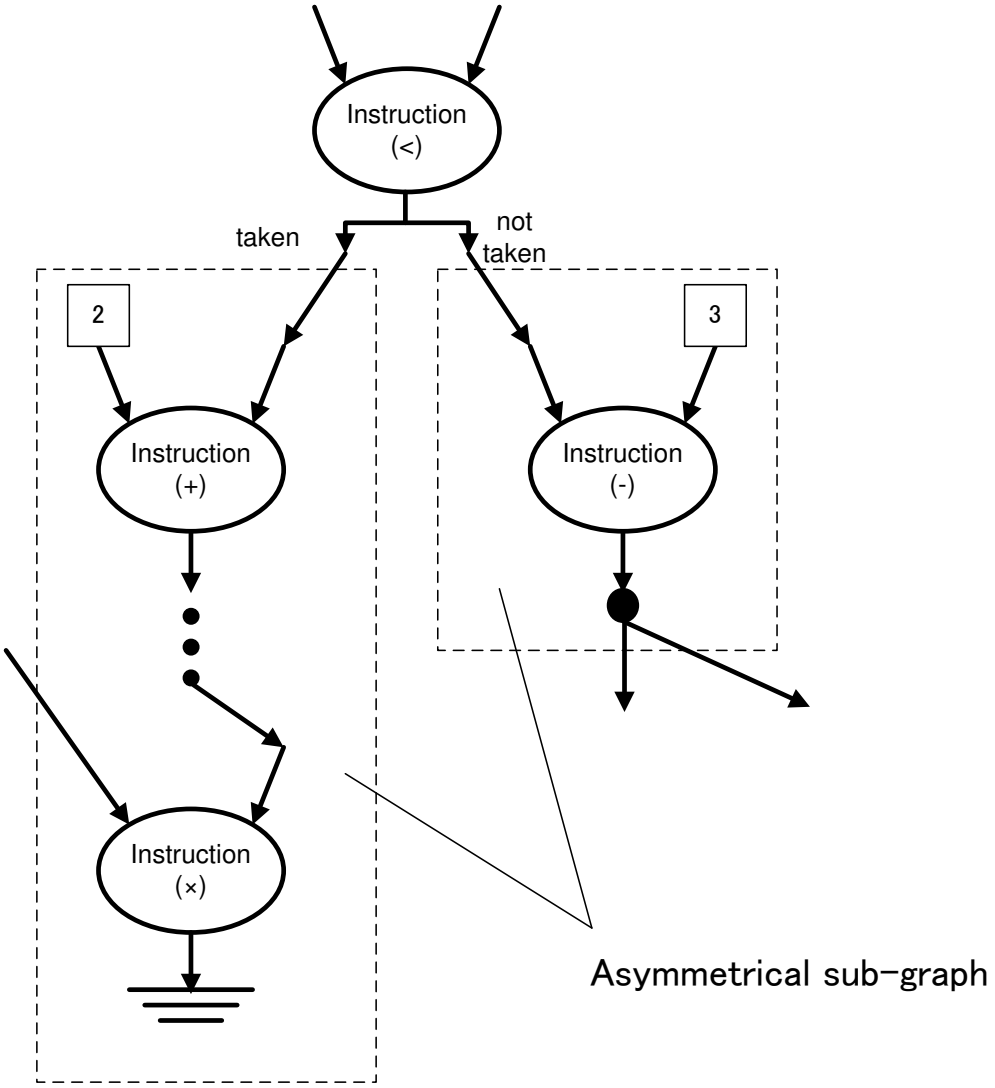

図 5.7　値分岐によりパケット流量が変化する例

に記憶命令を追加した置換プログラムを例示する．記憶命令（record）は，ノード番号及びパケットの識別子をアドレスとして，1bit の分岐方向（taken を ’1 ’，not taken を ’0 ’）を格納する命令である．

次節では，トレースに基づいて擬似命令を実行するためのエミュレーション機構を示し，それを搭載した DDP コアの構成を示す．

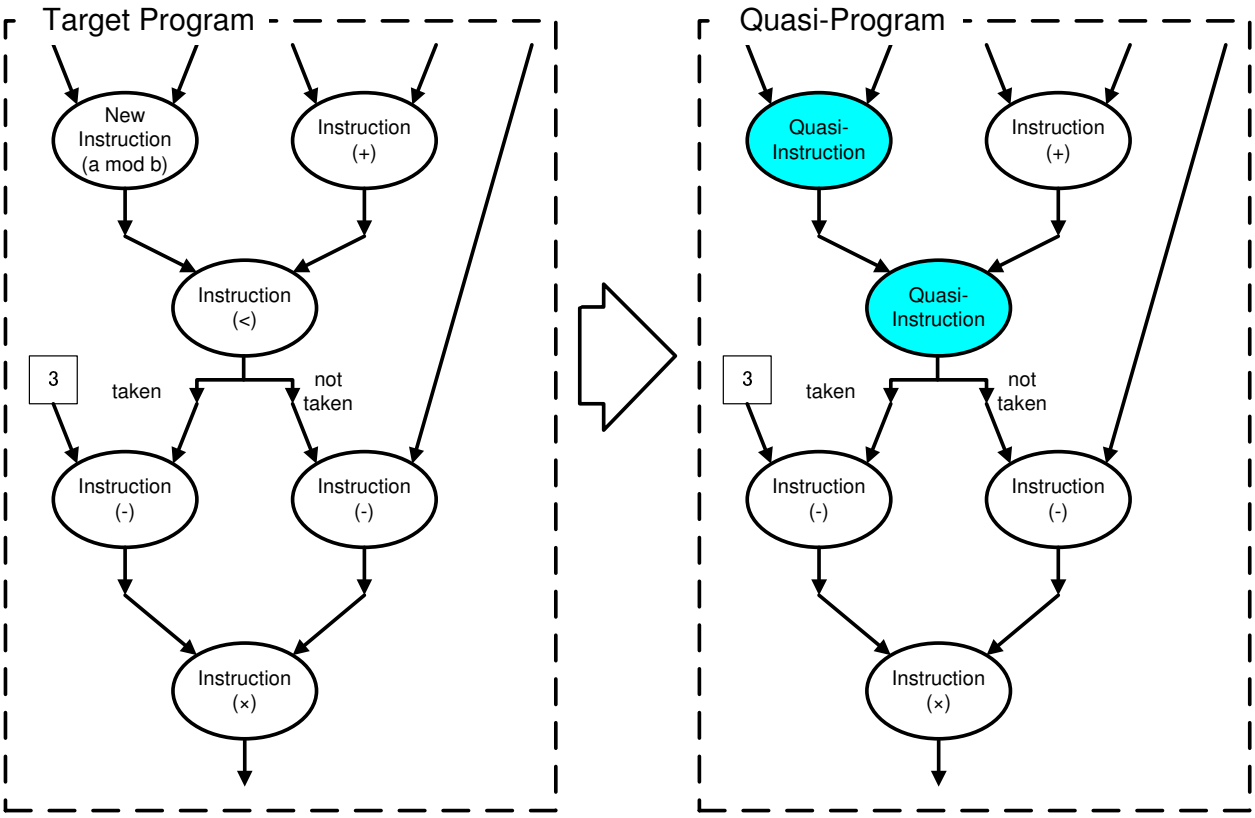

図 5.8　擬似命令による対象プログラムの置換

5.4　エミュレーション機構

オンチップ・エミュレータは，擬似プログラムの実行において，擬似命令の実行時にハンドシェイク時間を調節する可変遅延機構（VD）と記憶した分岐方向の記憶を行う分岐履歴機構（BHM）を，DDP コアに搭載して構成する．図 5.10 にオンチップ・エミュレータのブロック図を示す．VD は，データ駆動処理モジュールに搭載して，オペ・コードに基づき，動的にハンドシェイク時間を調節する．また BHM は，ALU 部に搭載して，パケットが擬似命令を通過する時に記録した分岐方向を設定することで，ターゲットのパケット流量を模擬する構成である．

本構成により，ホスト上でターゲットのパケット流量を模擬することができ，実行時間以内で対象システムが模擬できる．このためには，トレースの評価自体を，パケット流量を変化させないように，擬似プログラムの実行中に隠蔽する必要がある．

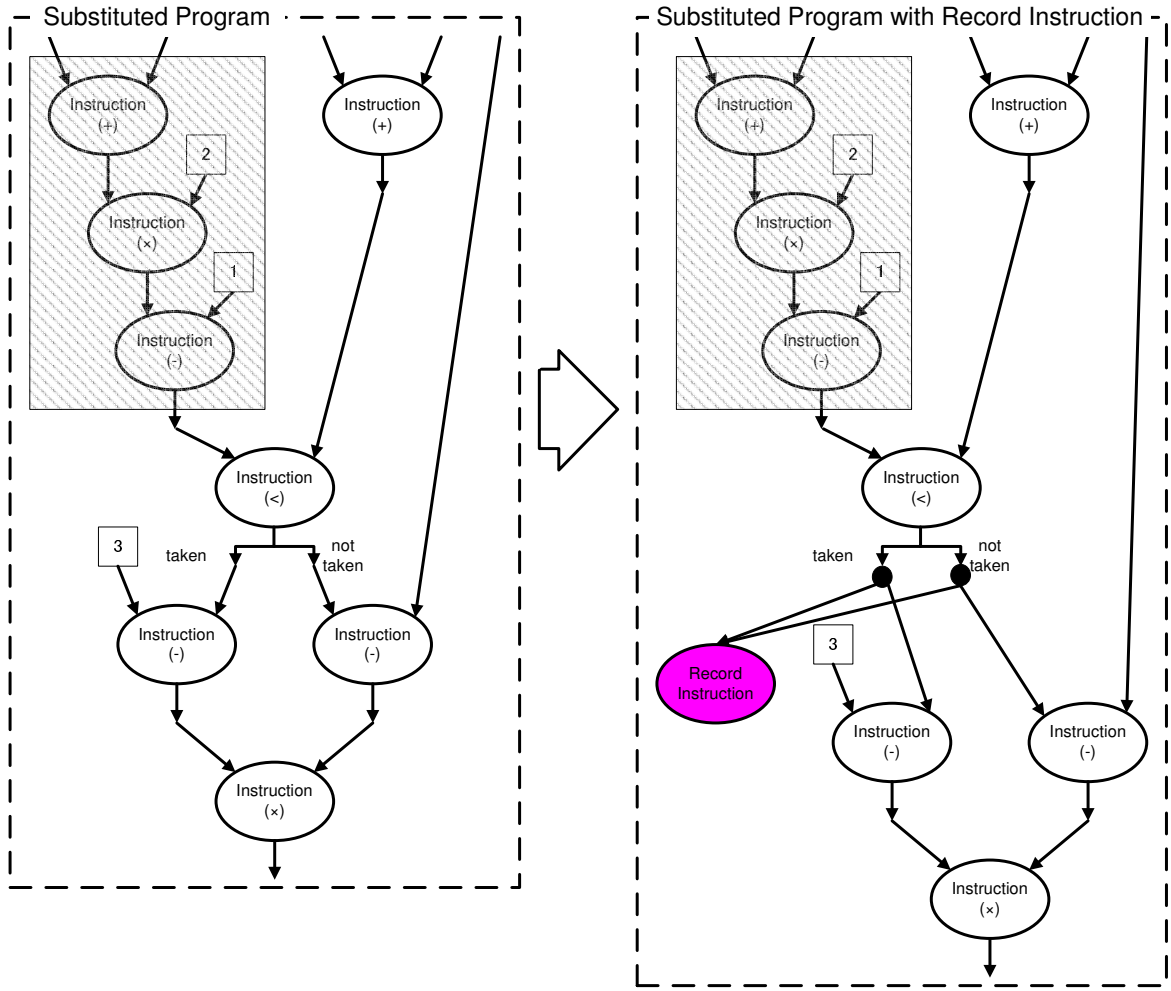

図 5.9　分岐記憶命令付き置換プログラム

本論文では，VD と BHM をエミュレーション機構と呼ぶ．次節以降では，エミュレーション機構の機能と STP 回路構成法を示す．

5.4.1　分岐履歴機構

DDP では，パケットのタグ中に，ノードの接続先毎に，結果パケットの出力の有無を示す 1bit のフラグを設定している．現行の実装は，最大で 2 出力を許容するための 2 フラグを持つ．各フラグでは，消去が ’0 ’，通常転送が ’1 ’として表現される．パケットの複製の場合，全フラグを ’1 ’とし，分岐の場合，分岐先への出力に対応するフラグのみ ’1 ’とする構成である．表 5.1 に，フラグビットの値と機能の対応を示す．このような実装によ

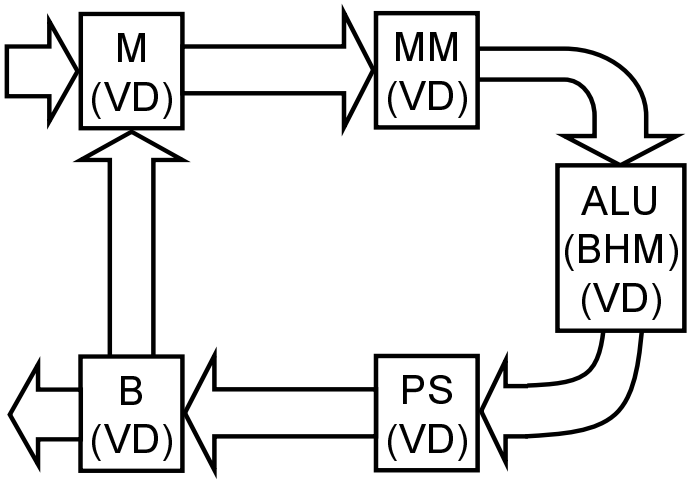

図 5.10　オンチップ・エミュレータのブロック図

表 5.1　フラグビットの意味

	00	01	10	11
Ordinary Instruction	absorb	-	output	copy
Branch Instruction	absorb	not taken	taken	-

り，STP を一周する間に，命令実行と同時にパケットの消去／複製を実現している．現行の DDP の実装では，命令の分岐方向は taken 方向と not taken 方向へのパケットの出力の有無を示す 2bit で表現されている．

トレース駆動エミュレーション手法では，パケット流量を変動させ得るすべての分岐方向を記憶する必要があるため，エミュレーション可能なプログラムの規模は，分岐方向を記憶するメモリ容量で決まる．DDP では 2bit で表現される分岐方向に対して，分岐履歴（BHM）機構では，格納される分岐方向を，taken を’1’ とし，not taken を’0’ として，1bit に符号化することで，必要となるメモリ容量を抑える．符号化された分岐方向を EBD（embedded branch direction）と呼ぶ．BHM 機構は，EBD をパケット識別子及びノード識別子でアドレッシングすることで，分岐方向を読み書きする構成である．

BHM は，書き込みと読み出しに，それぞれ record と qbranch のニーモニックを持った

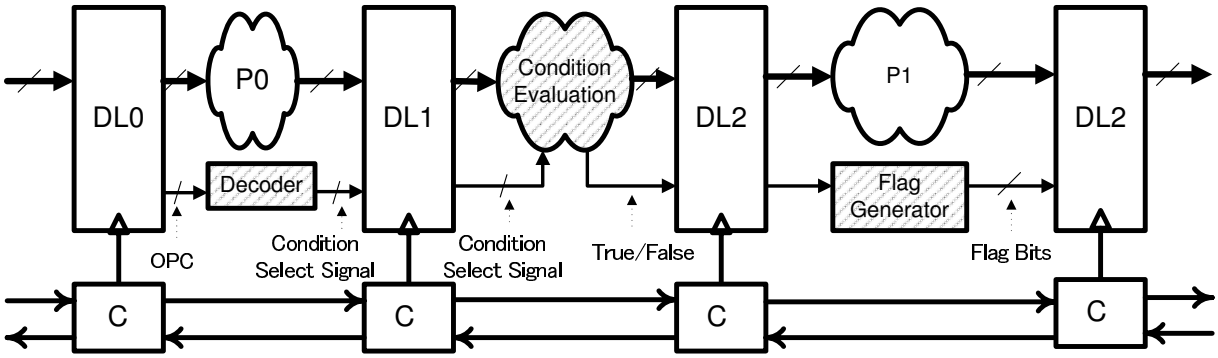

図 5.11　分岐命令のパイプライン回路

擬似命令に対して動作する．record 命令は機能検証時，また qbranch 命令は性能検証時に用いる．

機能検証時

record 命令を追加した置換プログラムに対してデータを投入して，機能検証を行う．このとき，すべての分岐方向は record 命令で記憶される．record 命令に到着したパケット（分岐結果パケットと呼ぶ）は，タグ中に識別子，ノード識別子，及び分岐方向を持つ．record 命令は，分岐方向を EBD に符号化して BHM へ格納する．格納後，分岐結果パケットは不要であるため，record 命令の実行終了後に消去する．

性能検証時

新規命令およびパケット流量を変動させ得る分岐命令を qbranch 命令で置換した，擬似プログラムに対してデータを投入して性能検証を行う．このとき，アセンブラ等を用いて，qbranch 命令のノード識別子を，対応する record 命令の識別子と一致させておけば，機能検証時に記憶された EBD が読み出される．この EBD を複合してフラグビットを生成してパケットに設定することで，分岐方向を保証する．qbranch 命令を実行するパイプライン回路は，オーバヘッドなく，あたかも対象プログラム中の分岐命令として動作する必要がある．

図 5.11 に分岐命令を実装した STP のブロック図を示す．分岐命令の実行は，通常，命令コードの解釈，条件式の評価，及び分岐方向の選択といった，3 ステージにパイプライン分割される．

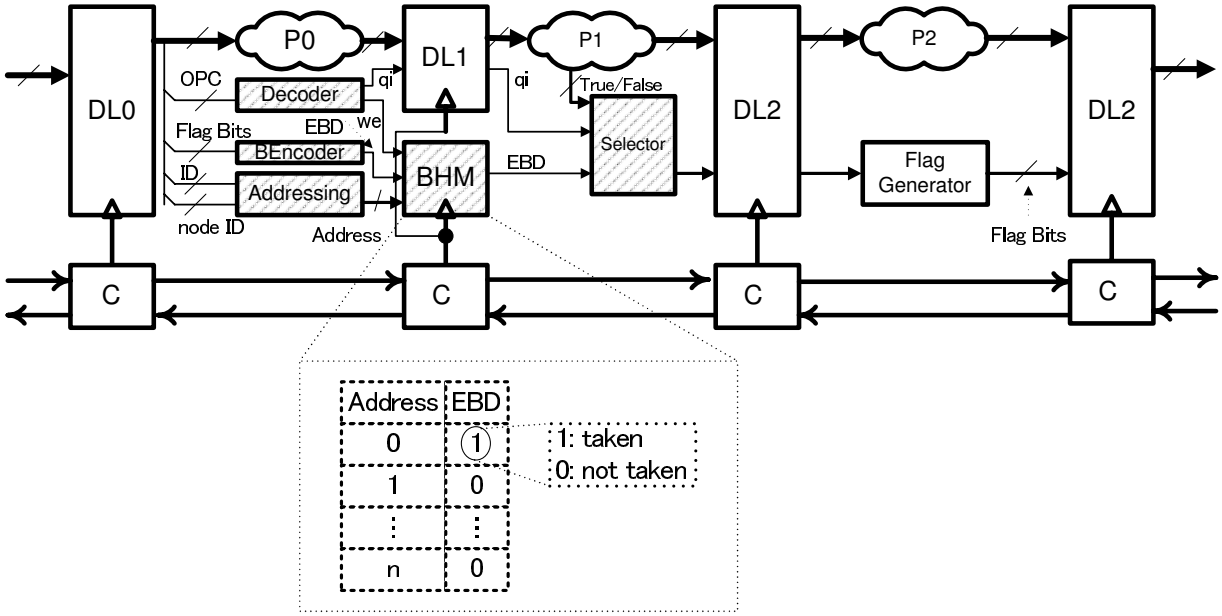

図 5.12　BHM 機構のパイプライン回路構成

この構成に基づけば，条件式の評価を，BHM のアクセスに切り替えることで，record 及び qbranch 命令を実現できる．図 5.12 に BHM 機構のパイプライン回路を示す．すなわち，まず，命令コードの解釈において，命令コードが record であった場合，書き込み許可信号（we）をアサートする．一方，命令コードが qbranch であった場合，qi フラグを立てることにより，擬似命令の実行を後続ステージに伝える．分岐命令の実行は，パケット識別子（ID）とノード識別子（node ID) により検知できる．これらの情報に基づき，Addressing は，BHM の実効アドレスを算出する．第 2 ステージには，BHM が SRAM として実装されており，EBD を出力する．また，qi フラグにより，通常の分岐命令の条件式の評価結果（True/False）と，EBD を選択することにより，通常の分岐命令の実行も可能とする．このため，選択器（Selector）は qi フラグが立っていれば，すなわち，qbranch 命令の実行であれば，EBD を出力する．第 3 ステージでは，EBD あるいは条件式の評価結果をフラグビットに復号した結果をパケットに付帯させる．このような構成では，BHM に対するアクセス時間をデータパス上の機能ロジック（P1）に隠蔽することができるため，分岐方向を保証することによるレイテンシの増加はない．

5.4.2　可変遅延機構

ハンドシェイク・プロトコルの下では，データ転送要求 send 信号とデータ転送許可 ack 信号の到着によってのみ，ハンドシェイクが進行する．このため，send あるいは ack 信号を遅延させても，ハンドシェイク時間が伸張するのみである．ここで，4.3 節で示した性能特性の 3 状態の条件において，ack 信号を遅延させる，すなわち，T_{ri} を増加させた場合，状態 II-2 の占める割合が増加するため，ハードウェア利用効率が低下する．一方，send 信号を遅延させる，すなわち T_{fi} を増加させた場合，状態 I の割合が増加し，パイプライン並列処理の向上が期待できる．したがって，データパスの処理を保証するためのハンドシェイク時間の延長には，通常，send 信号に十分な遅延を設定する．

VD 機構は，排他的に選択できる複数の遅延素子（D）を持ち，VD 機構を send 信号線上に挿入することで，ハンドシェイク時間を延長させる．VD には，遅延量の設定方法の点で 2 つの実現法がある [32]．すなわち，遅延量を変更する仕組みを持ち，動作時に任意の遅延量を設定する可変遅延量方式と，遅延の量と種類を予め固定しておき，動作時に選択する固定遅延量方式である．図 5.13 に可変遅延量方式の実装例を示す．オンチップ・エミュレータでは，全てのデータ駆動モジュール中の転送制御パスに VD を組込むため，STP のステージ数に応じた回路投資が必要となり，柔軟性よりも回路コストを優先するべきである．これに対して，前者は，柔軟な遅延量を設定できるが，非同期式カウンタと比較器を持っている．一方，後者は，比較的簡素なマルチプレクサの構成で実現できる．本研究では，固定遅延方式を用いる．

固定遅延方式の実装では，N 個の遅延を提供する場合，N 入力のマルチプレクサを用いる並列実装法と，2 入力のマルチプレクサを用いる直列実装法が考えられる．図 5.14 と 5.15 に，それぞれ並列実装法と直列実装法による VD の回路構成を示す．

並列実装法では，N 種類の遅延を提供するために，$2 \times N + 2[gate]$ の回路が必要である．一方，直列実装法では，$6 \times \log_2 N[gate]$ である．すなわち，$N \leq 8$ のときに並列実装法を，また $N > 8$ のときに直列実装法を用いれば，回路コストを抑えることができる．

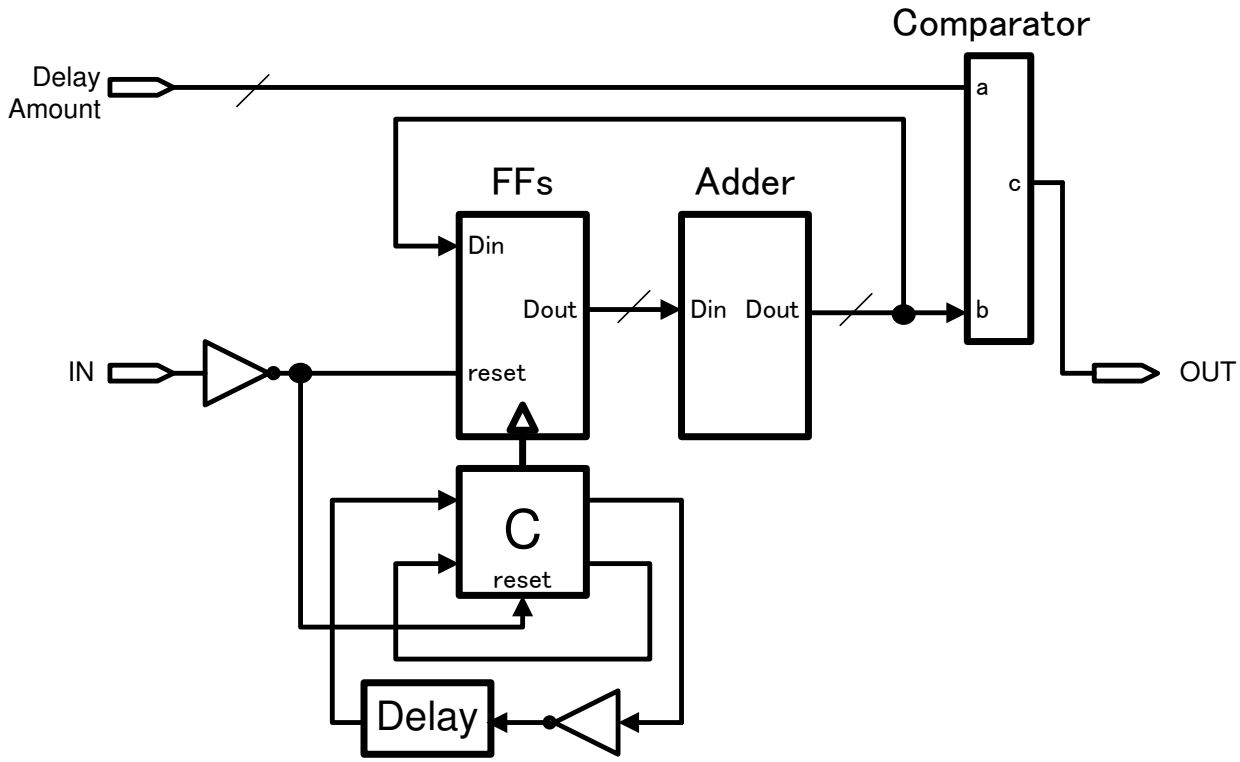

図 5.13　VD の可変遅延量方式による実装例

エミュレーションでは，この VD 機構の遅延選択信号として，擬似命令の命令コードを与えることで，命令に応じてパケットの周回時間を動的に調節する．このように動的なハンドシェイクの調節を保証するには，転送制御信号にハザードが生じないように，send 信号の到着に先立って，選択信号は安定している必要がある．この制約を満たせるように，VD を搭載した STP 回路の構成を図 5.16 に示す．図では，8 種類の遅延を備えた VD を想定している．

VD 機構は，2 ステージにパイプライン分割する．すなわち，まず，第 1 ステージで，VD_Dec は命令コード（OPC）を解釈し，擬似命令に応じた遅延選択信号（Select Signal）を生成する．その後，後続するステージで，VD の遅延が選択信号により有効となる．このようなパイプライン実装により，VD_Dec の処理時間は，データパス中の機能ロジック（P0）に隠蔽できるため，ハンドシェイク時間を動的に調節することによるレイテンシの増加はない．

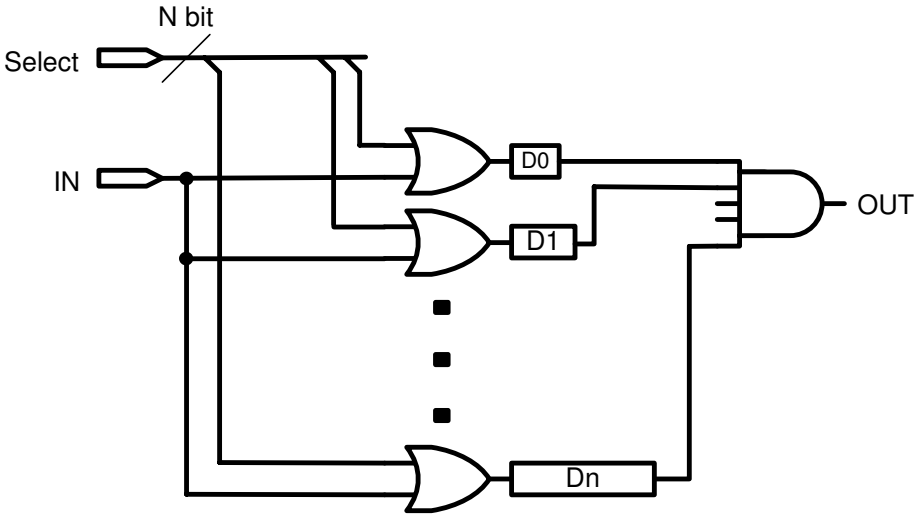

図 5.14　VD の並列実装法

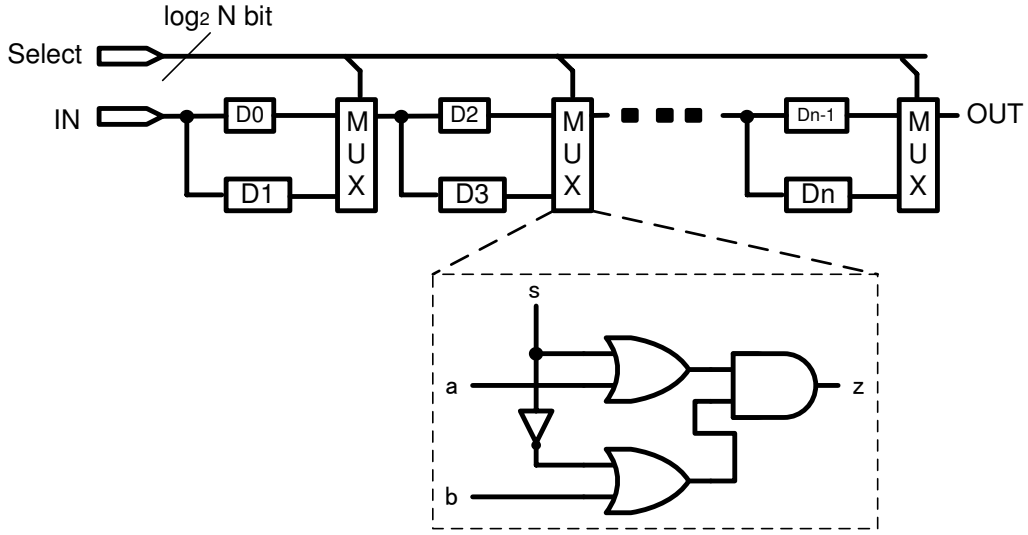

図 5.15　VD の直列実装法

以上のパイプライン回路構成法により，BHM と VD によるトレースの評価時間は隠蔽でき，擬似プログラムを直接的に実行し，正確なタイミングで結果パケットを出力できる．また，両機構とも，擬似命令でのみ駆動しかつ他の命令の実行には影響しない．したがって，オンチップ・エミュレータは通常の DDP コアとして機能して，必要時にのみエミュレータとして機能する．

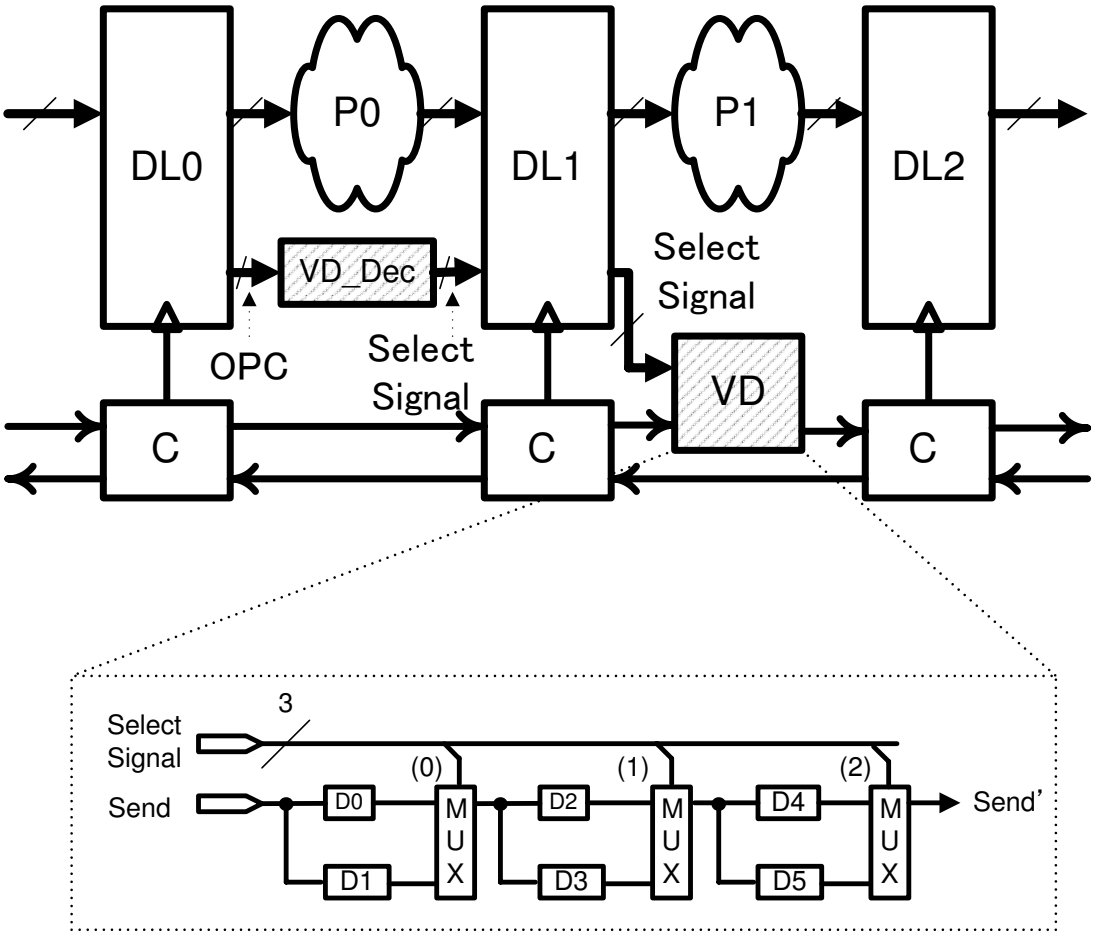

図 5.16　VD 機構のパイプライン回路構成

5.5　オンチップ・マクロ・シミュレーション手法

第 2 章で述べたとおり，大規模なプログラムの開発では，DFG で記述したシステム機能を階層的に具体化する．この際，プログラムを機能単位にサブグラフへ分割することで，可読性や再利用性を高めることができ，生産効率の向上が期待できる．このようなプログラムの部品化には，サブグラフ内での負荷変動を極力抑えることが重要である．これは，サブグラフ毎に負荷が大きく変動する場合，ピーク時が重なる可能性が大きくなり，結合後に状態 II-2 あるいはオーバフローに至る危険性があるためであり，したがって，負荷の変動を極力抑えて定常的に高いパケット流量を保つ必要がある．これには，システム設計者の経験や知識あるいはコンパイラの支援を得るなどして，プログラムの構造をハードウェアに最適化す

る必要がある．

本論文では，サブグラフでは負荷の変動が緩やかである，すなわち，あるパケットの周回中はパケット流量が変動しない程度の緩やかな負荷変動であることを前提として，チップ上でマクロ・シミュレーションを実行する，基本的なシミュレーション回路構成を示す．

マクロ・シミュレーションでは，STP システムを模擬するためには，STP 中の総パケット数 P_{total} に対して規定された STP 中のパケットの速度 $V(t)$ を，全パケットに一律適用する必要がある．このため，チップ上で，マクロ・シミュレーションを実行する場合，対象 DFG を解釈・実行すると同時に，P_{total} を監視して，P_{total} に応じてパケットの周回時間を調節する機構が必要となる．これに対して，トレース駆動エミュレーション手法を用いて対象 DFG を解釈・実行すると同時に，キュー（FIFO）を用いて P_{total} を検出して，さらにエミュレーション機構を使って $V(t)$ を調節する手法を示す．

以降，P_{total} に対する $V(t)$ を $V(P_{total})$ とする．

擬似プログラムの実行において，パケットが 1 週する間の P_{total} の変化が無視できる範囲では，STP 中をフローするパケットを一箇所にキューイングしておき，VD を用いてパケットの周回時間を調節して $V(P_{total})$ の速度で，キューへの到着時刻順に 1 パケットずつエミュレーション回路を周回させることで，パケット毎の周回時間が模擬できる．この方法では，ターゲットとホスト上でパケットが STP を周回し終える時刻（周回時刻）が異なるため，パケット毎に周回時間を計測及び累算してターゲット上での周回時刻を保持できる機構が必要となる．また，パケットの入出力及び消去／複製による P_{total} の増減を求める機構が必要となる．求められた P_{total} に基づき，VD を用いて $V(P_{total})$ をパケットに適用するには，パケットが，$V(P_{total}$ で STP を一周する時間に一致するように，VD を選択すれば良い．このとき，$V(P_{total})$ と VD の選択信号を対応させる機構が必要となる．以上を踏まえて構成した，オンチップ・マクロ・シミュレータ回路のブロック図を図 5.17 に示す．

シミュレータ回路は，エミュレーション機構を搭載した DDP コアにさらに，a. パケットの周回時間を計測しターゲット上での時刻を更新する Start Time Stamper 及び Lap Time Stamper，b. STP 中をフローするパケットをキューイングして P_{total} を検出し，時刻順に

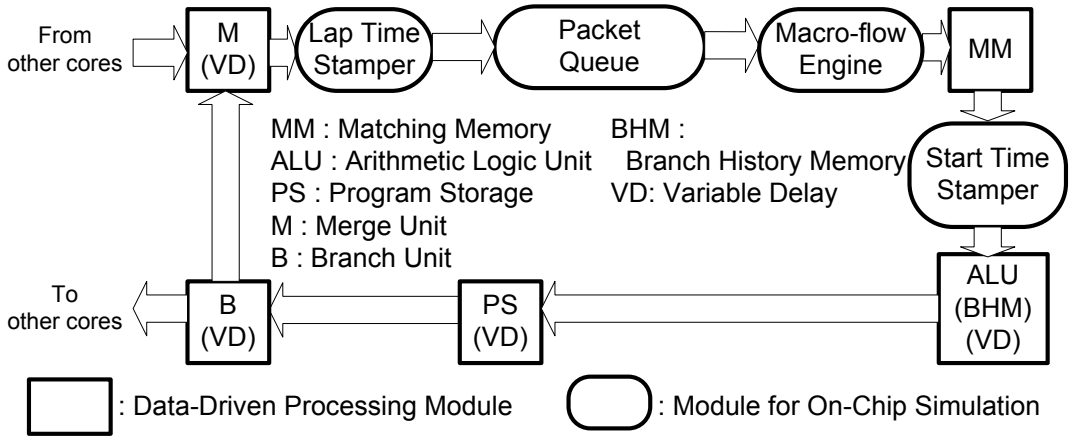

図 5.17　オンチップ・マクロ・シミュレータ回路

パケットを出力する Packet Queue，及び c. パケットに P_{total} に対する VD 選択信号を与える Macro-flow Engine といった 3 つのシミュレーション機構を搭載して構成する．

Start Time Stamper & Lap Time Stamper

パケットの周回時刻は，周回開始時刻と周回完了時刻の差から求められる．周回開始／完了時刻の押印に，Start Time Stamper ／ Lap Time Stamper を用いる．カウンタ回路を用いてシミュレーション開始時からの経過時刻を管理している．図 5.18 及び 5.19 に，それぞれ，Start Time Stamper 及び Lap Time Stamper のパイプライン回路を示す．Start Time Stamper は，パケットが通過する際に，経過時刻を周回開始時間として，パケットに付帯させる構成である．また，Lap Time Stamper は，パケットが通過する際に，パケットに付帯している周回開始時刻と経過時刻の差（すなわち周回時間）求め，パケットの周回時刻に加算して更新させる．この手法では，両機構の時刻を同期させる必要があるため，カウンタ回路のカウントアップ信号には，同一のクロック信号を使用する．

以上より，ターゲット上の時刻を管理するには，周回開始時刻と周回時刻をパケットに保持させる必要があるが，パケットのデータ部に保持しておけば，時刻を保持することによる回路コストの増加はない．これは，トレース駆動エミュレーションでは，擬似分岐命令を用いることで値に関係なく，正確にパケット流量を模擬できるためである．

DDP コアでは，オペランドの待合せ時に，パケットはすべてのオペランドが揃うまで待

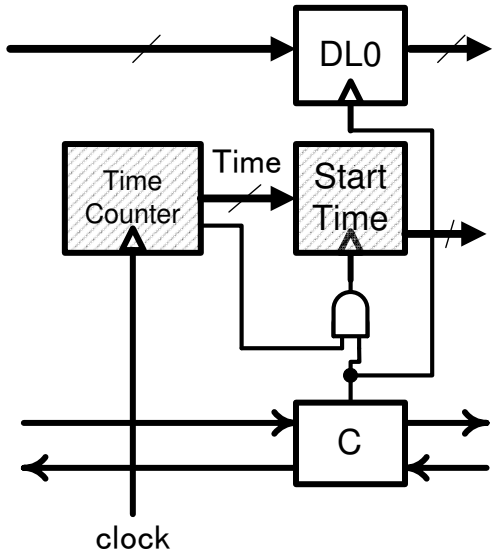

図 5.18　Start Time Stamper のパイプライン回路

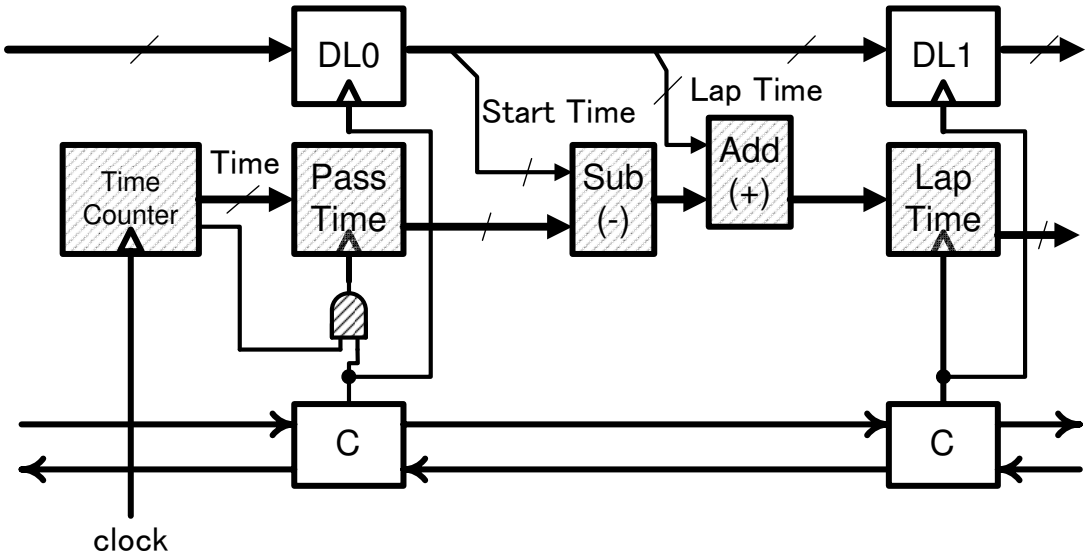

図 5.19　Lap Time Stamper のパイプライン回路

合せメモリに格納され，最後に到着したパケット（オペランド）により読み出されて，演算パケットが生成され，周回を開始する．このため，待ち合わせメモリに格納されたパケットの周回時間において，MM で滞在する時間は無視され，最後に到着するパケットの周回時間と一致する．オンチップ・シミュレータでは，MM における滞在時間を無視するために，図 5.17 に示した構成のとおり，MM の直後から周回時間を計測する．このとき，MM を通過

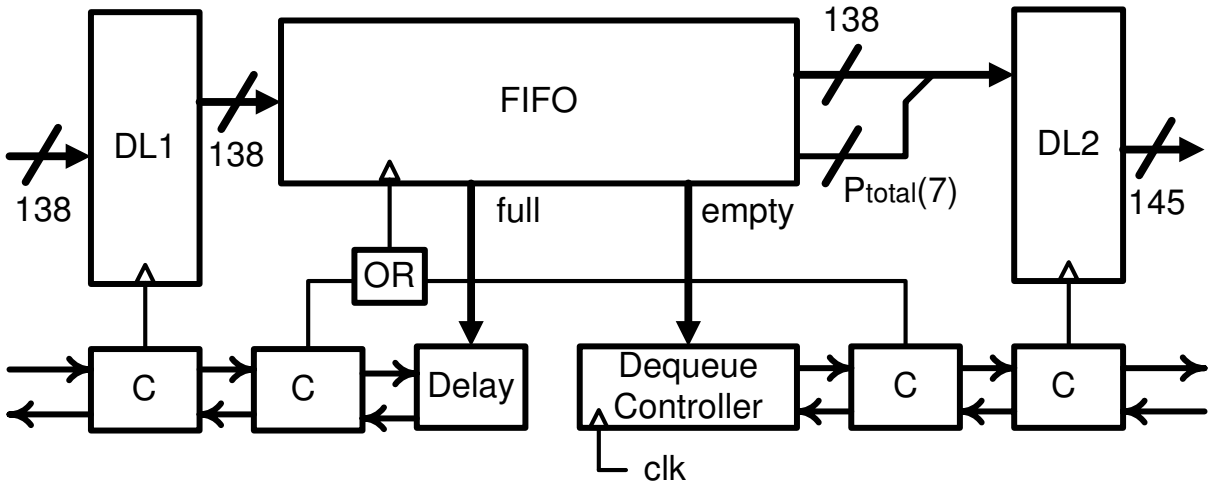

図 5.20　Packet Queue のパイプライン回路

する時間が無視されるので，その分を含めた遅延量を後続の VD で選択する．

Packet Queue

Packet Queue は，STP 中をフローする全パケットを格納するキューである．つまり，Packet Queue に格納されているパケット数は P_{total} となる．Packet Queue は，P_{total} をパケットに付帯させて，FIFO に基づき出力する．1 パケット毎のマクロ・シミュレーションの実現では，パケットの周回を待って次パケットの周回を開始する必要があるため，キューからパケットを出力する間隔は，パケットが STP を 1 周する最大の時間に設定しておけばよい．このとき，パケットの出力や消去は，一定間隔の時間が経過した後もキューイングされないため，P_{total} の減少として自然に検出できる．また，複製は，一定間隔の時間が経過する前に連続して複数パケットがキューに格納されるため，P_{total} の増加として自然に検出できる．シミュレーション開始時には，投入時刻を周回時刻に設定して，入力パケットを投入時刻順にキューイングしておく．このような構成の実現では，パケットが到着次第キューに格納して，一方，一定間隔毎にキューからパケットを取り出す機構が必要である．

図 5.20 に Packet Queue のパイプライン回路を示す．Packet Queue は FIFO キュー・メモリと，キューに対する格納／読み出しを制御するモジュール（Enqueue Controller ／ Dequeue Controller）により構成される．

Enqueue Controller は，キューへのパケットの格納を制御する．キューに格納されたパケットに対するハンドシェイクの要求を後続ステージに伝えないための回路であり，send 信号を ack 信号に接続した構成である．また，キューが一杯になったとき，すなわち，オーバフローのときは，send 信号と ack 信号の接続を断ち切り，ハンドシェイクを停止する．

Dequeue Controller は，カウンタ回路により一定時間の経過を検出して，検出後にキューからパケットを取り出すと同時に，ハンドシェイク信号を生成して後続ステージにハンドシェイクを伝播させる回路である．キューが空のときは，カウントアップを停止して，ハンドシェイクを停止する．

FIFO キュー・メモリは，1 パケットを 1 ワードとして格納できるリングバッファの構造を持つメモリであり，キューの容量と格納されたパケット数に関して，空であるか一杯であるかを判断する．また，格納されたワード数すなわち P_{total} を出力する．出力された P_{total} はパケットと同時に，次ぎステージの Macro-flow Engine に転送される．

Macro-flow Engine

パケットの速度の調節では，pl と $V(P_{total})$ から算出される周回時間を実現できるように，予め適切な VD の選択信号を求めておき，P_{total} と複製の有無に応じて VD の選択信号をパケットに付与することで実現する．複製が有る場合の周回時間は，$V(P_{total+1})$ から求める．

Macro-flow Engine は，$V(P_{total})$ に対する VD の選択信号を保持しておき，P_{total} を基に，パケットに付与する．図 5.21 に Macro-flow Engine のパイプライン回路を示す．Macro-flow Engine は，命令デコードと選択信号読み出しの 2 ステージで構成される．また，VD の選択信号を格納するためのメモリを持っており，選択信号を格納するための特別な制御命令である，vwrite 命令により書き込みを行う．まず，第 1 ステージで命令をデコードして，vwrite 命令であれば，メモリに対する書き込み許可信号（we）をアサートする．また，パケットに付帯している P_{total} からメモリの実効アドレスを得る．次いで，第 2 ステージでは，メモリにアクセスして VD の選択信号を更新あるいはルックアップする．この構成では，後続ステージに VD の選択信号を伝播させる必要があるが，トレース駆動エミュレーション手法では，値に関係なく正確にパケット流量を模擬できるため，パケットのデータ部

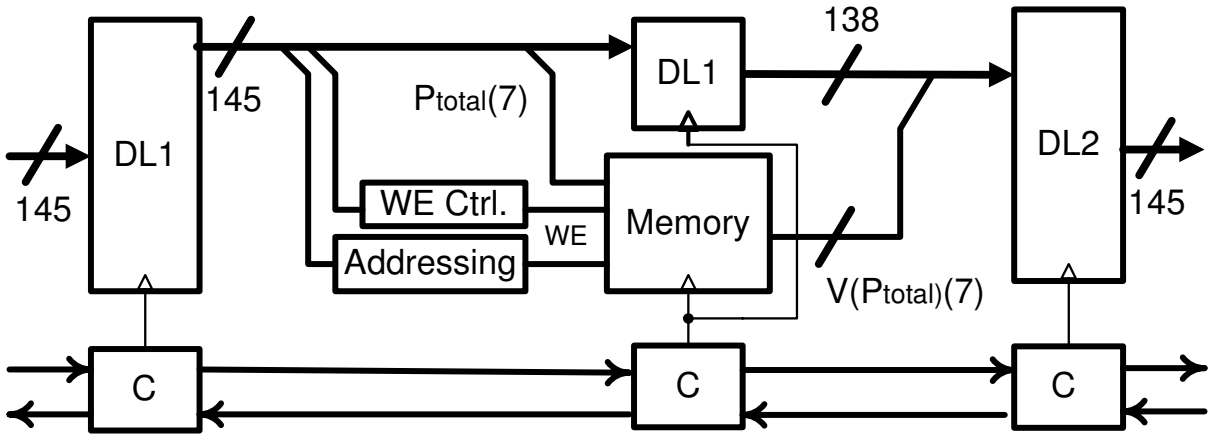

図 5.21　Macro-flow Engine のパイプライン回路

に選択信号を格納することができる．このため，通常の DDP コアの回路は増加しない．

シミュレーションの流れ

本シミュレータ構成では，擬似 DFG を直接的に評価する．すなわち，まず，評価に先立って，$V(P_{total})$ を Macro-flow Engine に格納する．また，トレース情報と擬似 DFG を，それぞれ，BHM と PS に格納する．入力されたパケットは，Packet Queue で一時的にキューイングされる．Packet Queue は，キューイングされたパケット数から，P_{total} を決定する．Macro-flow Engine は，ルックアップテーブルを持ち，$V(P_{total})$ を読み出す．読み出した $V(P_{total})$ を模擬できる遅延を VD から選出する選択信号をパケットに付与する．これにより，算出された P_{total} に基づき，各 VD が適切に選択され，パケットの周回時間が模擬される．また，タイムスタンプ値に周回時間を積算する Lap Time Stamper により，パケットの周回時刻を更新する．

このようなシミュレーションにより，1 命令の模擬が，ほぼ 1 命令の実行時間内で行えるため，シミュレーション時間は大幅に短縮できる．

提案したオンチップ・シミュレータ構成は，DDP コアとエミュレーション回路を内包している．このため，シミュレーション機構をバイパスする経路を設けるだけで，通常は DDP コアとして機能し，必要に応じてシミュレータ／エミュレータとして機能できる．このため，全コアにシミュレーション／エミュレーション機構を搭載しておけば，システム全域に亘る

高速な模擬が実現できる．

5.6　結言

本章では，シミュレーション／エミュレーション機能を持つ DDP コアの構成法を示した．提案した構成では，トレース駆動方式に基づいて，DDP コアを直接的に用いた対象プログラムの解釈・実行を可能としており，オンチップ・シミュレーション手法の実現に伴う回路規模の増加を抑えている．また，対象プログラムの実行タイミングをターゲットと一致させるためのエミュレーション／シミュレーション回路の構成法を示した．エミュレーション回路では，ステージ毎のハンドシェイク時間を動的に調節する機構により，ステージ数が一致していれば直接的な模擬を可能とした．シミュレーション回路では，エミュレーション回路を活用してパケットの速度を変化させることで，マクロフロー・モデルに基づくシミュレーションの実現のための回路規模の増加を抑えた．

提案方式に基づけば，マクロフロー・モデルによりモデル化された対象を高速に模擬可能となる．また，ウェブパイプラインなどを含むシステムの場合，DDP コアを用いてモデル化されていない部分をソフトウェアでシミュレーションし，さらにシミュレーション／エミュレーション結果を統合すれば，システム全体が模擬できる．課題としては，マルチコア構成において，実行時間の異なるシミュレーション／エミュレーションを限りなく並行して実行する手法の確立が残されている．

第 6 章

評価

6.1 緒言

本章では，提案した相互作用回路に基づくウェブパイプライン，およびマクロフロー・モデルに基づくオンチップ・シミュレーション手法を評価する．

まず，ウェブパイプラインの実現可能性および妥当性を検証するため，ウェブパイプライン専用処理エンジンの LSI 回路を設計する．設計結果に基づき，処理性能と設計容易性を評価する．セルフタイム型パイプライン STP では，隣接するステージ間のハンドシェイク時間がパイプライン・スループット [packet/sec.] を決定する．このため，設計した回路の配置配線後の遅延情報を用いた RTL（register-transfer level）シミュレーションを行い，ハンドシェイク時間を測定する．また，設計容易性の定量的な評価として，2 年間の CAD ツールの使用経験を持つ修士学生が，対象回路の HDL（hardware description language）記述から回路の配置配線までに要した期間を計測する．

次に，オンチップ・シミュレーション手法の評価精度，シミュレーション時間，および回路規模を評価する．オンチップ・シミュレーション手法は，マクロフロー・モデルに基づくマクロ・シミュレーション手法と STP 回路を直接用いるエミュレーション手法で構成される．エミュレーション手法は，STP のパケット流量とハンドシェイク時間をターゲットとホストで一致させるため，ターゲットの実行結果と同一の結果を得ることができる．一方，マクロ・シミュレーション手法では，シミュレーション精度はマクロフロー・モデルに依存する．本章では，マクロフロー・モデルに基づく（ソフトウェア）シミュレータを実装して，実プロセッサの実効性能の見積りに適用した結果に基づき，提案モデルの妥当性を示す．ま

た，シミュレーション／エミュレーション機構を FPGA を用いて実装し，有効性を示す．

6.2　ウェブパイプラインの評価

決定的および非決定的な相互作用を用いた，ウェブパイプライン専用処理エンジンとして，パイプライン・キュー，オメガ網ルータ，およびパイプライン・ソータを対象とし，TSMC 社 180nm CMOS 6 層メタル 1.8V の標準セル・ライブラリを用いて LSI 回路を設計した．各エンジンにおける相互作用制御回路は，図 3.1 および 3.2 の回路構成を仕様に基づき縮退して得た．得られた相互作用制御回路の各回路モジュールは，表 3.1 に示した入出力信号の仕様に従い，すべての入力の組合せに対して動作を検証した．さらに，配置配線後の遅延情報を用いた RTL シミュレーションにより，導出したタイミング制約を満足することを確認した．

6.2.1　セルフタイム型パイプライン・キュー

セルフタイム型パイプライン・キュー（SPQ）[24] は，直線状のパイプラインを折返して，各ステージ間にバイパス経路を設けることで構成される．SPQ のブロック図を図 6.1 に示す．SPQ では，対向するパイプライン間の相互作用によりバイパスを実現する．この相互作用において高優先度を持つパケットを優先的にバイパスすることで，明示的なスケジューリングなしに，優先度付き可変長キューイングが実現できる．この機能性は，ネットワーク・トラフィック制御における QoS 制御に有効である [34]．

パイプライン間のバイパスは，一方向の転送経路を持ち，またデータは常に対になるとは限らないため，非決定的な一方向相互作用を用いる．導出した相互作用制御回路を図 6.2 に示す．ルータは，相互作用に先立って，データの優先度から転送経路を決定する．この転送経路に対して，アービタは，図 6.3 に示すように，データラッチを用いて，send-in 信号を監視し，衝突の有無を検出する．衝突がある場合，パイプライン A のデータの優先度が高い場合はパイプライン B へバイパスする．一方，パイプライン A のデータの優先度が低

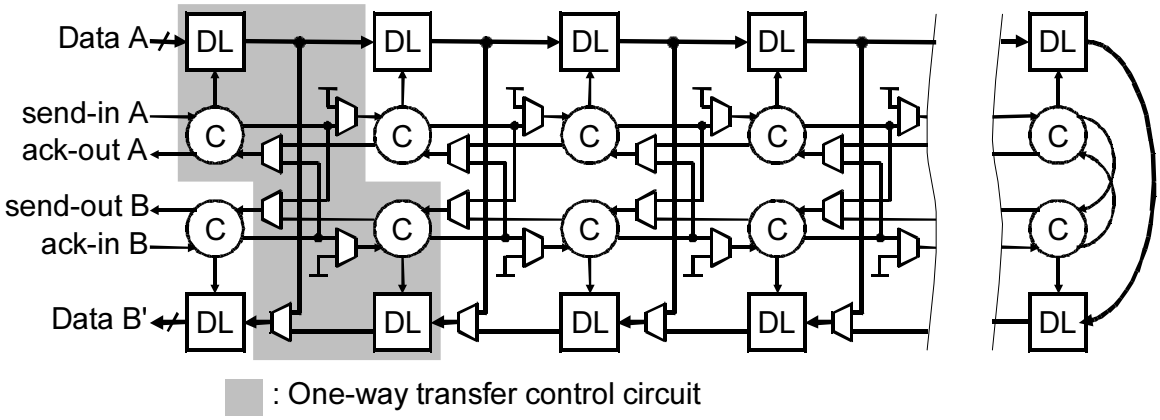

図 6.1　セルフタイム型パイプライン・キュー

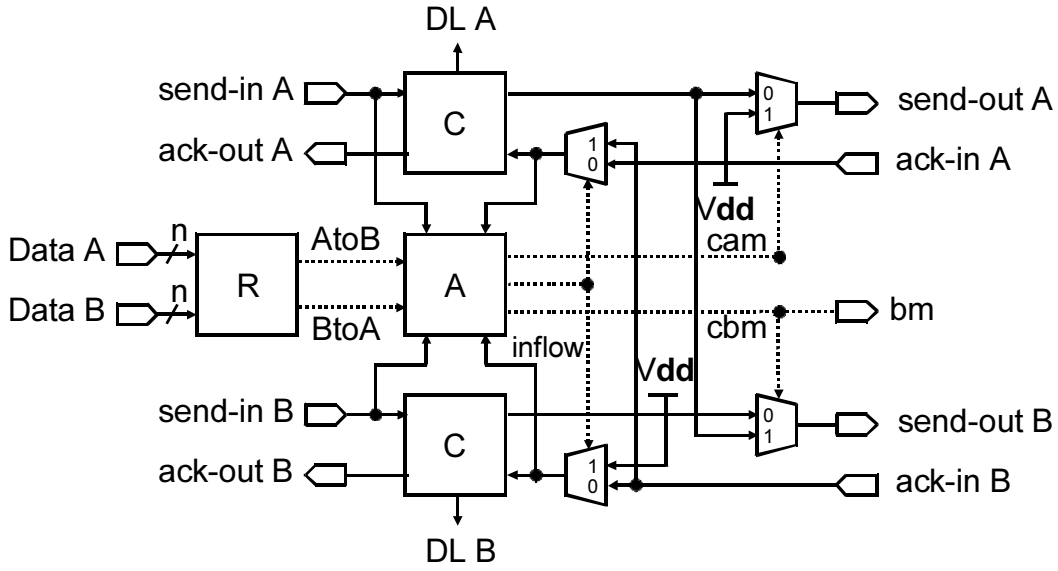

図 6.2　一方向相互作用制御回路

い場合は，パイプライン A のデータを前方のステージへ転送すると同時にパイプライン B のデータも前方のステージへ転送するよう MUX を制御する．この際，MUX の出力を V_{dd}（すなわち，ネゲート）へ吊上げて，前方の転送先ではないステージへの send-out 信号の不必要な伝播を防ぐ．

設計した SPQ は，128 段のバイパス経路を持ち，256 個のパケットをキューイングする構成である．設計したウェブパイプライン回路と，既存の人手により設計した回路 [24] の比較を表 6.1 に示す．結果，設計した回路は人手により設計した回路と同等の処理性能である，

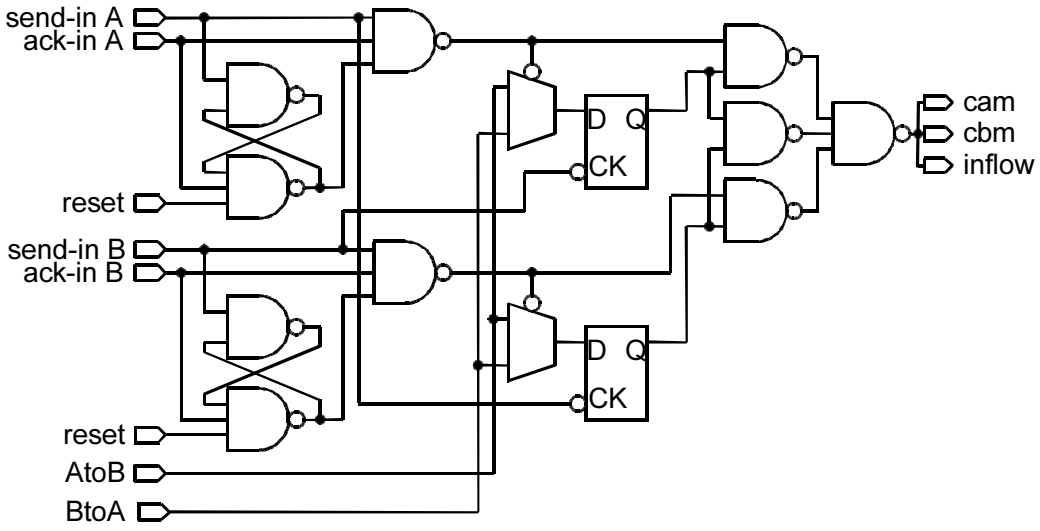

図 6.3　SPQ のアービタ

表 6.1　人手による回路設計との比較

	Logic area [mm^2]	Throughput [packet/sec.]	Design term [man/month]
Proposed	1.25	100 M	0.25
Manual design	2.01	106 M	3

100M[packet/sec.] を達成することを確認した．一方で，回路設計に要する期間を 1/6 まで短縮できることを確認した．

6.2.2　セルフタイム型オメガ網ルータ

データ駆動プロセッサ DDP コア間を接続するチップ内ネットワーク（NoC）の実現のため，一般的なオメガ網による多段相互接続網をウェブパイプラインにより設計した．設計したセルフタイム型オメガ網ルータ（SPR）は，一般的な 8 コア間の接続を想定し，8 ポートの構成とした．図 6.4 にブロック図を示す．

オメガ網は，2 入力 2 出力のスイッチング・セルで構成される．スイッチング・セルでは，双方向の転送経路が必要であること，およびデータ対が必要とは限らないことから，非決定

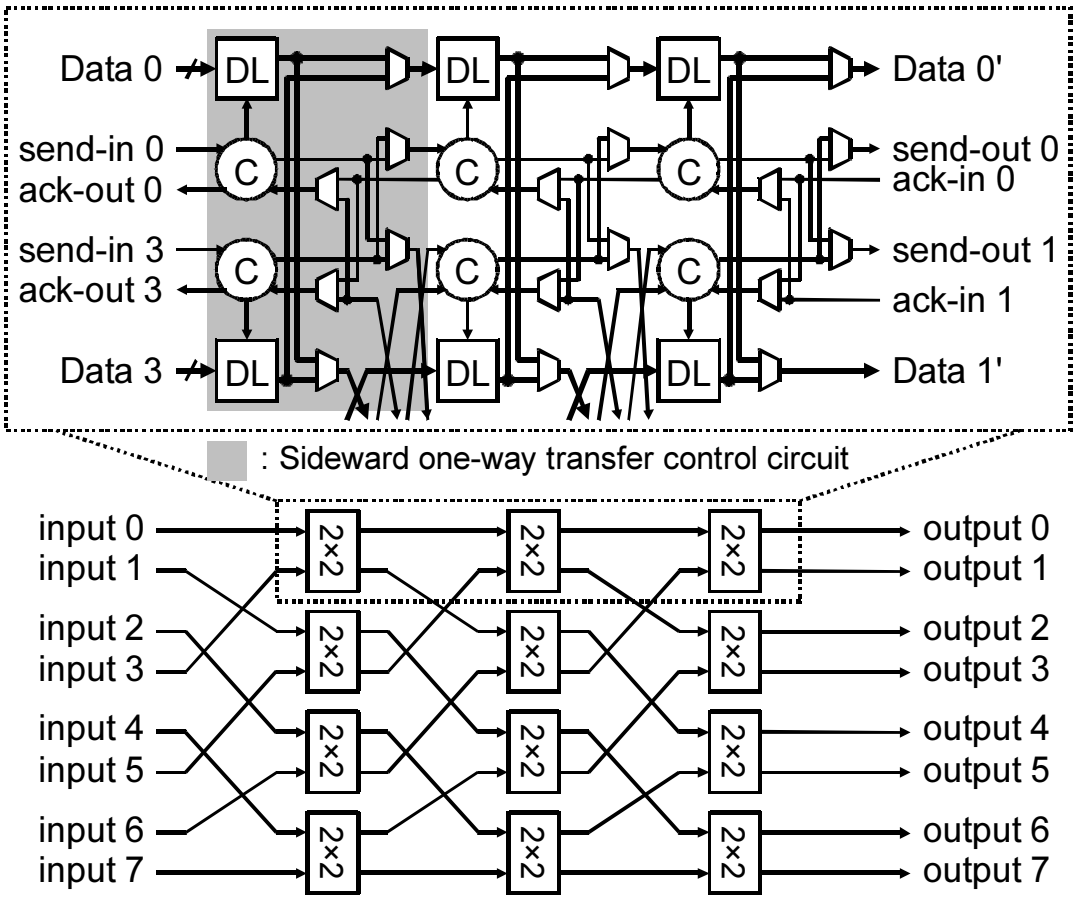

図 6.4　セルフタイム型オメガ網ルータ

的な双方向相互作用を用いる．導出した制御回路を図 6.5 に示す．ルータは，データの行先情報に基づき，転送経路を決定する．行先が同じ，すなわち，衝突する場合，アービタの動作により，各データは先着順に排他的に転送される．一方，衝突しない場合，アービタは，それぞれのデータを行先に同時に転送できるように MUX を制御する．アービタは，図 6.6 に示すように，フリップフロップを用いてパケットの到着順序を検出し，さらに，ルータの出力に基づき衝突の有無を判断する．

設計した LSI 回路に基づき，ハンドシェイク時間を測定した結果，160M[packet/sec.] を達成できることを確認した．これは，現行の 140M[packet/sec.] の処理性能を持つ DDP コア間の相互接続網に十分な性能である．また，LSI 回路の設計期間は，1/3[man/month] であった．

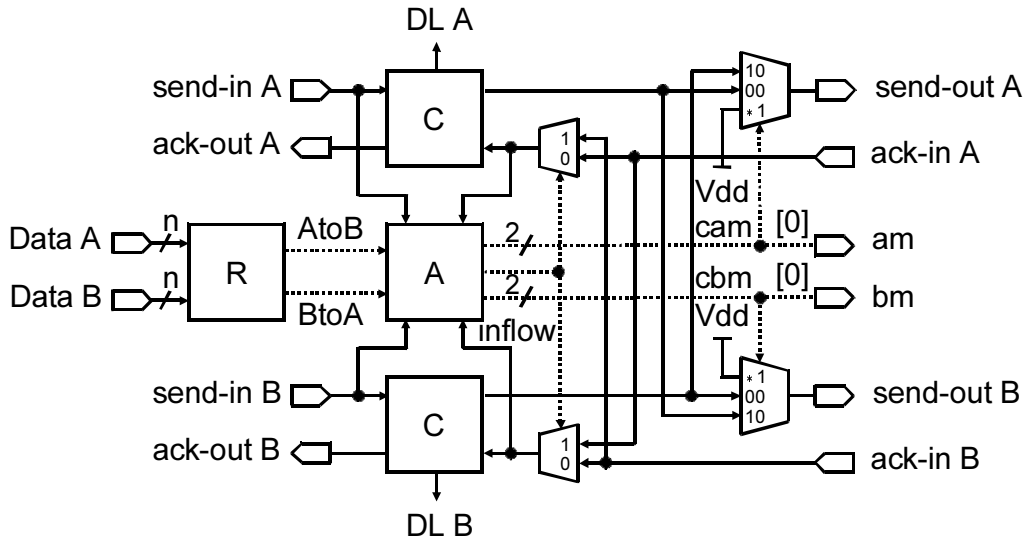

図 6.5　双方向相互作用制御回路

6.2.3　セルフタイム型パイプライン・ソータ

TCP/IP パケット処理などに不可欠であるソート処理を，高並列にパイプライン処理できる，セルフタイム型パイプライン・ソータ（SPS）を設計した．SPS 回路を図 6.7 に示す．

設計した回路は，入力データを流すパイプライン A と，並べ替えたデータを保持するパイプライン B を対向させる構成である．パイプライン B は，ソート処理が終了するまで，データを保持し続けるため，一種のメモリとして振る舞う．二本のパイプライン間の対応するステージ間において，相互作用により，パイプライン A のデータ（E_A）とパイプライン B のデータ（E_B）の比較を実現する．ステージ毎の比較において，E_B の順序を入れ替える必要がある場合，E_B と E_A を入れ替える，すなわち，E_B をパイプライン A へ転送し，E_A を E_B が占有していたパイプライン B のステージへ転送する．入れ替えの必要がない場合，E_A はパイプライン A の次ステージに転送され，新たに比較が行われる．この結果として，全ての入力データがパイプライン B の上でソートされる．こうした相互作用には，一時的に停止したパイプラインのデータを入れ替えが必要であり，また，比較には必ずデータ対が必要なため，決定的な横方向流出付き一方向相互作用を用いる．導出した相互作用制御回路を図 6.8 に示す．ルータは，データの値を比較し，ソートの条件に応じて，転送経路を決

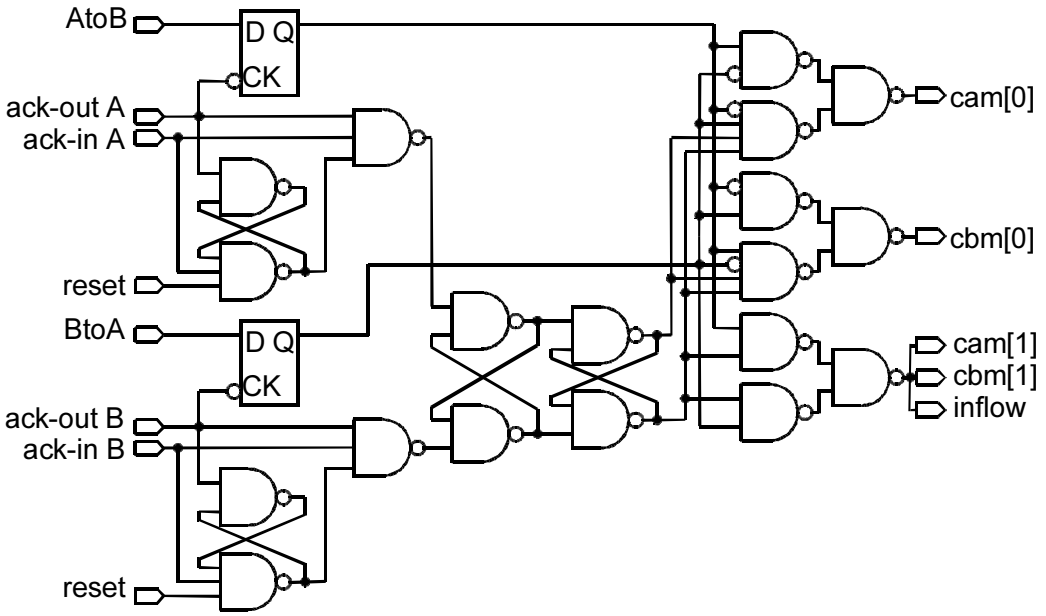

図 6.6　SPR のアービタ

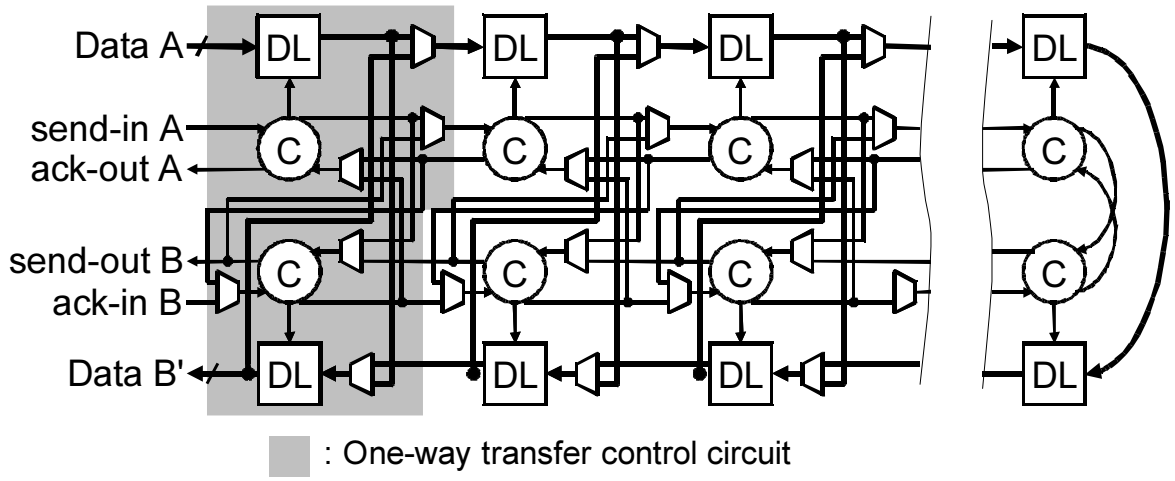

図 6.7　セルフタイム型パイプライン・ソータ

定する．また，アービタは，図 6.9 に示すように，フリップフロップを用いて，データ対の到着を検出し，データの転送を制御する．

回路は，TCP パケット・リアセンブルに必要な 256 個のソート処理を実現できるよう，256 パケットを収容できるように設計した．処理性能を測定した結果，100M[packet/sec.] を達成可能であることを確認した．これは，32[bit] のペイロードの場合，3.2G[bps] のネッ

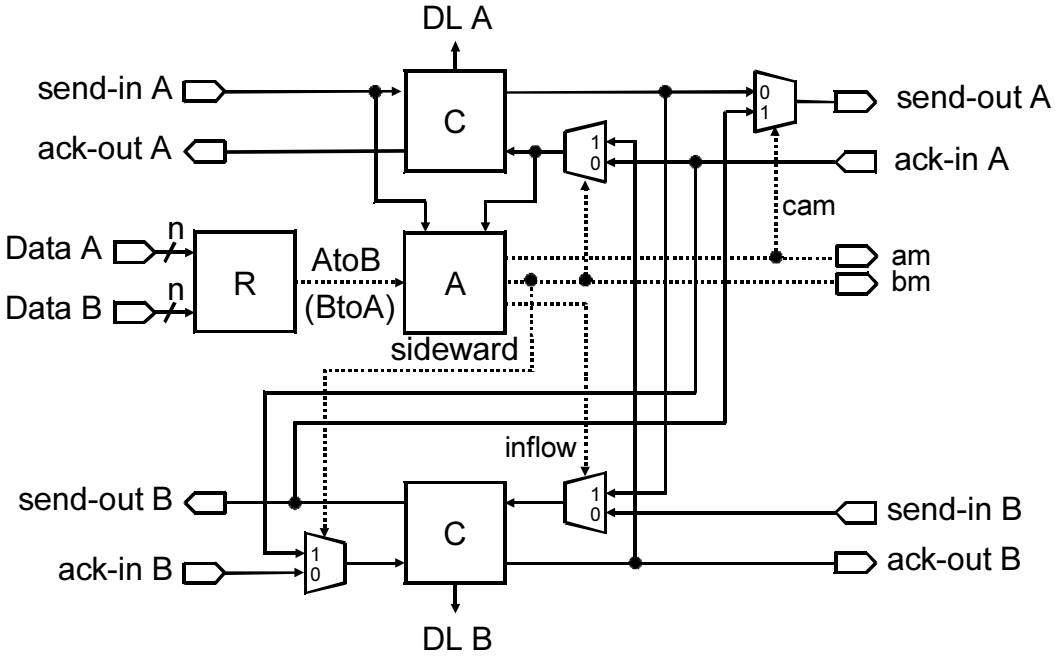

図 6.8　横方向流出付き一方向相互作用制御回路

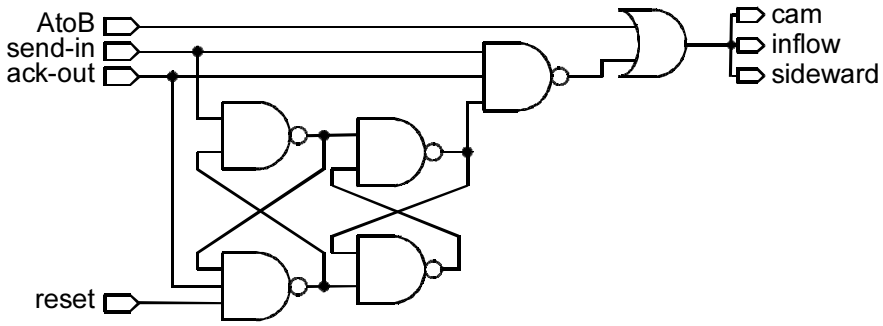

図 6.9　SPS のアービタ

トワークにおけるリアセンブルを実現できる処理性能である．また，設計に要した期間は，1/6[man/month] であった．

以上，3 つのウェブパイプライン専用処理エンジンの設計結果から，提案した相互作用制御回路は，十分な処理性能を持つ高い機能性を実現でき，また設計期間を大幅に短縮可能であることを確認した．

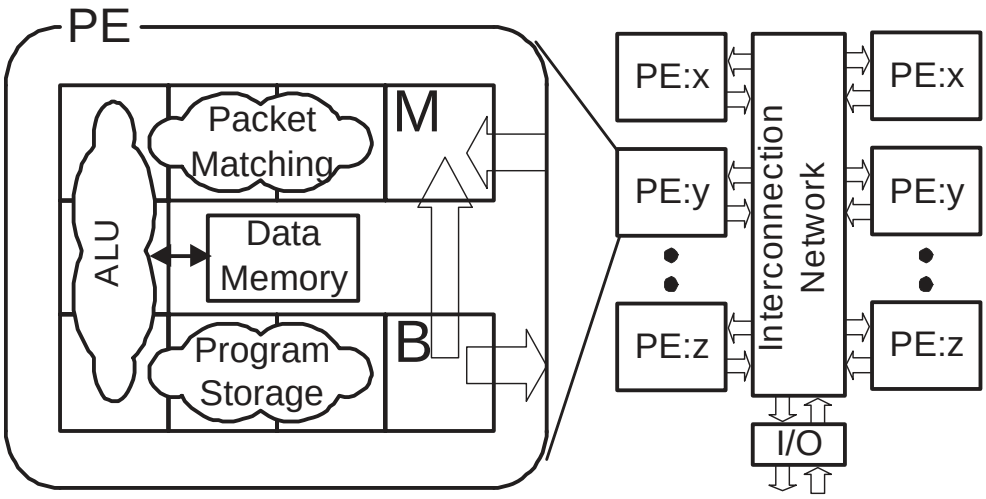

図 6.10　現行の DDP の構成

6.3　マクロフロー・モデルの評価

マクロフロー・モデルの妥当性を示す．セルフタイム型パイプライン機構 STP の実装として現行の DDP チップのプロセッサ・コア（PE）を対象に，パラメータを設定して，シミュレータを実装し，実効性能の見積りに適用した．また，従来の，ハンドシェイクのタイミングを逐一模擬する性能見積りモデル（ナイーブモデル）に基づくシミュレータ（以降，単に従来シミュレータ）と比較し，シミュレーション精度やコストを評価する．

評価手法

図 6.10 に現行の DDP チップの構成を示す．現行の DDP では，PE 間を，M と B で構成する多段パケットルータで結合している．現行の DDP は，10 個の PE を持つが，PE はステージ数と命令セットにより 3 つに分類され，ここでは，それぞれ $PE:x$，$PE:y$，及び $PE:z$ とする．各 PE は，内部メモリアクセス命令等の共通の命令以外に，$PE:x$ は外部メモリアクセス，$PE:y$ はタグ処理及び算術演算，更に $PE:z$ は論理演算のための命令セットを持つ．

図 6.11 は，現行の DDP を搭載した評価ボードの写真である．評価ボードは，DDP チップに加えて，2 つの FPGA チップを搭載している．FPGA には，通過パケットに時刻を押印する機構を備えた DDP コア・プロセッサが実装されており，時刻を押印されたパケット

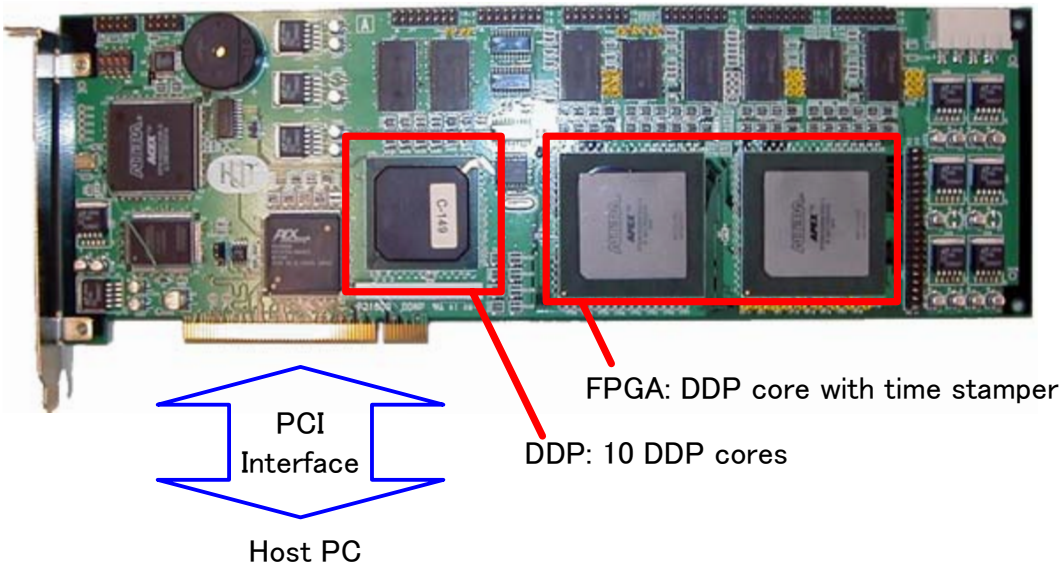

図 6.11　DDP 評価ボード

は，PCI バス・インタフェースを介してホスト PC に転送することができる．ホスト PC では，パケットの時刻から性能値を算出できる構成である．また，DDP チップと FPGA チップ間のハンドシェイク信号をプローブすることができ，詳細なタイミング情報を取得できる．

マクロフロー・モデルに基づくシミュレータは，Java を用いて実装した．本シミュレータは，データ駆動プログラム，入力パケット，及び pl 等の PE 単位の情報を入力として，タイムスタンプを付与した処理結果パケットを出力する．そして，イベント毎に更新されるパケットの位置に基づいて，プログラムに従った演算を適用する構成となっている．同様に，従来シミュレータも，Java を用いて実装した．また，設計では，パケットの入出力や画面表示部分は共通のクラスを用いており，シミュレーション・コアのみモデルに依存した構成にすることで，モデルの計算コスト等の正確な定量評価を可能とした．

シミュレーションでは，$(T_{fi} + T_{ri})$ が必要となる．DDP チップ上の STP では，LSI 製造工程において $(T_{fi} + T_{ri})$ がバラついている可能性がある．このバラつきも反映したパラメータを得るため，式（4.13）に基づけば pl，P_{1-2}，及び P_{II-1U} からシミュレーションに必要なパラメータが調節できることに着目して，1 チップ実装された DDP において，これらのパラメータが採取できるデータ駆動プログラムを実行した．表 6.2 に各 PE のパラメー

表 6.2　DDMP のパラメータ

	$PE:x$	$PE:y$	$PE:z$
pl	63	43	43
P_{IU}	34	31	30
P_{II-1U}	58	39	39

タ値を示す．このパラメータ値に基づき，マクロフロー・モデルでは，T_{max} を 1 とした，$\sum T_f$ 及び $\sum T_r$ を求め，さらにこれらを用いて式（4.1）,（4.4），及び（4.11）より各状態の $V(t)$ を求めた．また，ナイーブ・モデルでは，$(T_{fi}+T_{ri})$ を，パラメータ値を基に調節した．これらは，回路設計時に見積ったパラメータと同等のものである．量モデル及び実 STP では，$V\ (t)$ を状態 I の $V(t)$ を 1 として正規化することで，比較可能とした．

6.3.1　シミュレーション精度

負荷変動がない場合

まず，処理負荷，すなわち，P_{total} が一定になるデータ駆動プログラムを用いて比較評価を行った．実プロセッサの実測値（Real）を図 6.12 に示す．グラフの横軸は STP の負荷すなわち $\frac{P_{total}}{pl}$ であり，縦軸は $V(t)$ である．$PE:y$ と $PE:z$ の結果はほぼ一致しており，グラフ上の差異はないため，図には $PE:x$ と $PE:y$ の結果のみを掲載する．結果から，段数や，各状態におけるパケット数の上限値が異なる構成であっても，マクロフローモデルに基づく見積り（Macro-Sim.）は実測値とほぼ一致している．

ここで，実プロセッサの実測値において，状態 I であってもパケットの速度は微かに変動し，数%の揺らぎがある．これは，設計段階で定めた $(T_{fi}+T_{ri})$ がバラついたことによる，パケット群の偏りが主因であると考えられる．このようなバラつきは，LSI チップの製造バラつきやチップ温度・電圧変動等に起因しており，現行の LSI 製造・実装技術では不可避である．ところが，バラつきによる偏りの性能見積りへの反映は，半導体素子レベルの模擬を

必要とし，設計中には困難である．したがって，性能見積りでは，この揺らぎに対する，設計余裕を見込む必要がある．結果のグラフでは，$P_{total} = P_{IU}$ におけるパケットの速度を 1 として正規化している．この結果，$PE : x$ と $PE : y(PE : z)$ では，それぞれ約 3%と約 4%の設計余裕が必要と判った．

以上のことから，マクロフローモデルによる見積りと従来シミュレータによる見積り（Existing Sim.）の誤差は，この設計余裕内に収まっていれば許容できる．実際，見積りの誤差は，いずれの PE でもすべて設計余裕内に収まっており，マクロフローモデルは，従来シミュレータと同等の十分な精度を持つと言える．

負荷変動がある場合

実際の応用プログラムでは，実行時に動的に処理負荷，すなわち，P_{total} が変動する．DDMP が採用するデータ駆動処理方式では，パケットが処理コンテキストを内包するため，明示的なコンテキストスイッチが不要である．このため，各プログラムの負荷の変動を抑えれば，複数のプログラムを並列実行でき，高スループット化が期待できる．この場合，応用システムの開発においては，各プログラムが要求される入力データ・レートで実行可能かどうかが第一の性能評価指標になる．本評価では，応用プログラムの入力データ・レートを固定して実行し，これと同時並行に，定常的な負荷（Background Load）を発生するプログラムを実行した．追加する負荷の量を変えることによって，状態 I，II-1，および II-2 で応用プログラムが動作する状態を設定でき，3 つの状態を扱えるマクロフローモデルの有効性が判断できる．

また，応用プログラムとしては，P_{total} が一定であることを前提とした提案モデルの見積り精度を評価するために，P_{total} の時間的な増減が比較的大きいラプラシアン近傍フィルタ（Filter）と，比較的小さい高速フーリエ変換（FFT）を選択した． また，両プログラムを実行する命令セットを備えた $PE : x$ を対象とした．図 6.13 に，結果を示す．図中横軸は負荷であり，縦軸は最良値を 1 とした処理レイテンシである．両プログラムとも，マクロフローモデルに基づく見積りは，従来シミュレータの見積りとの誤差が約 3%の設計余裕内に収まっており，従来シミュレータと同等の精度の見積りが可能である．また，実測値と比較

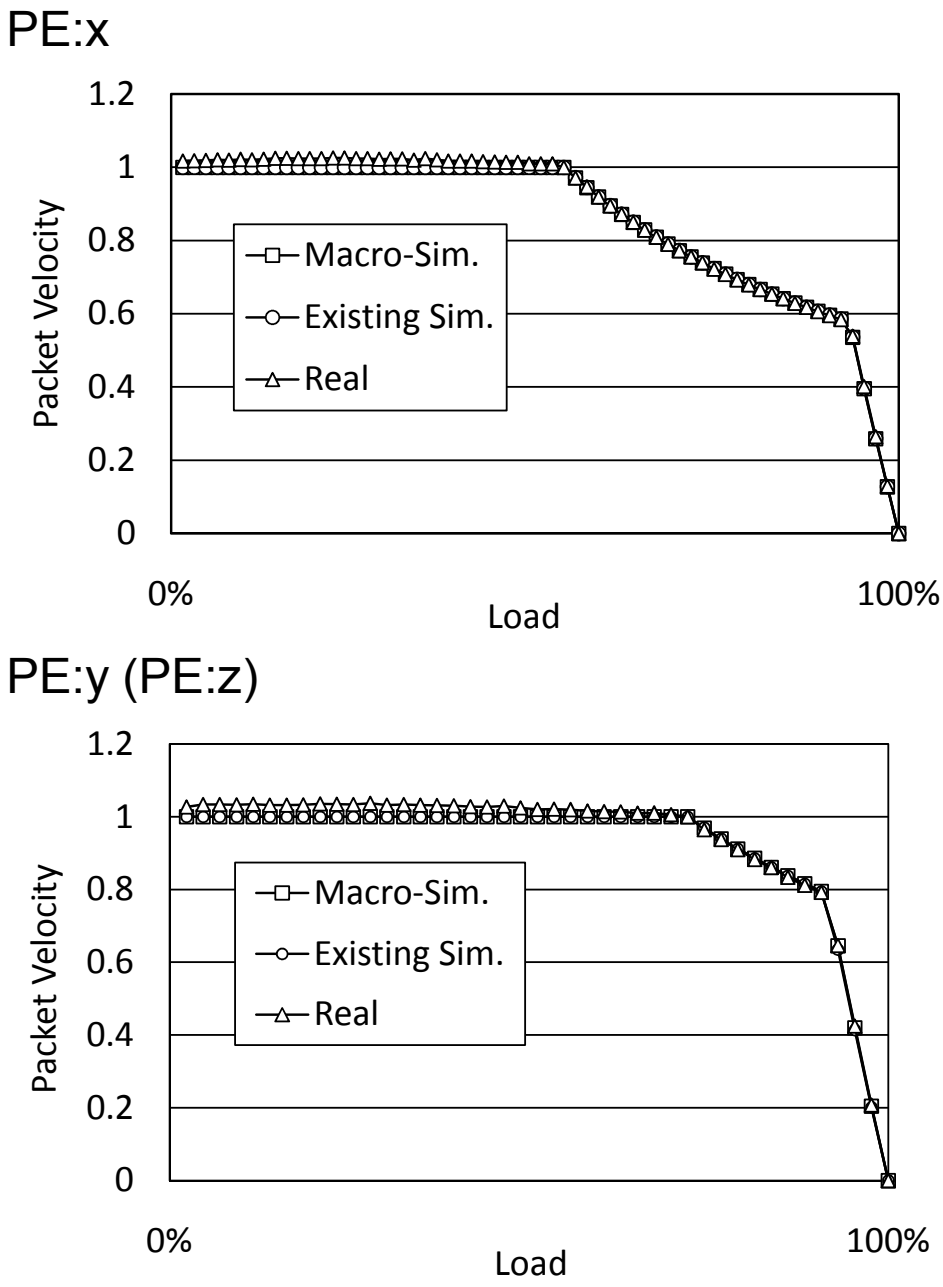

図 6.12　負荷一定の場合の比較

して，オーバフロー点を安全側に見積っていることから，最終的な応用プログラムの動作も保証できる．マクロフローモデルに基づく見積りと実測値との誤差は，約 $\pm 4\%$ であった．

本評価に用いた応用プログラムは，時間的な負荷の増減を抑えるようにプログラムを最適化済である．これは，最適化によって，定常的に高いパイプライン利用率を維持できて高性能化できるためである．実用的な応用プログラムは通常，このような最適化を施されるた

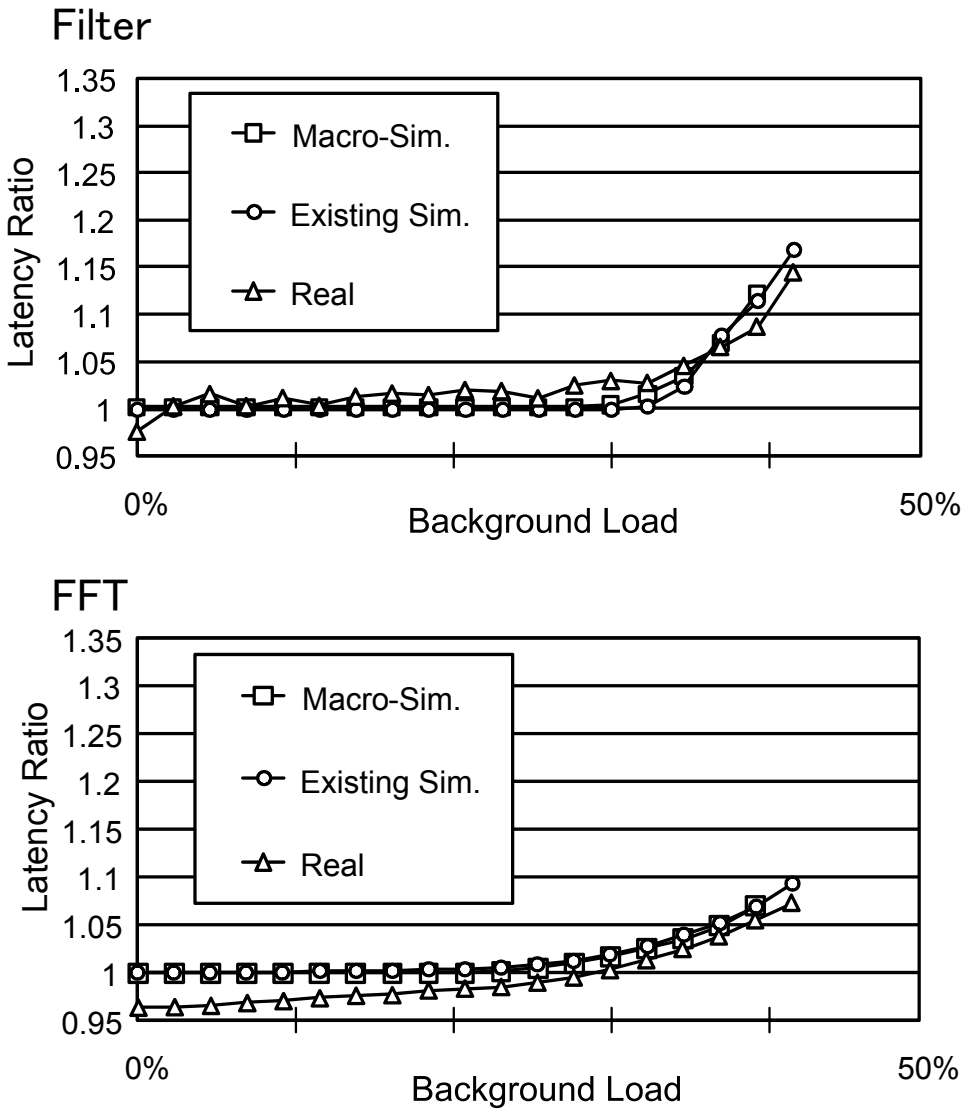

図 6.13　応用プログラムの性能見積り

め，P_{total} の時間的な増減は Filter プログラムと比較して同程度以下であると言える．

以上より，実システムであっても，P_{total} の時間的な変動が極端ではない，すなわち Filter プログラムと同程度以下であれば，マクロフローモデルは，LSI 試作前の事前評価に十分な精度を持つ高速なシミュレーションを実現可能である．

表 6.3　シミュレーション時間の比較

	Simulation Time [sec.]			
	4-Dir. Filter		Expansion Filter	
Added Load	0%	50%	0%	50%
Macro-Simulation	1.8	4.9	1.3	4.8
Naive-Simulation	6.7	24.1	3.8	20.7
Speed-Up Ratio	3.6	4.8	2.8	4.2

6.3.2　計算コスト

マクロフロー・モデルにより削減されたシミュレーションの計算コストを定量的に示すため，従来シミュレータと，シミュレーション時間を比較した．

マクロフロー・モデルに基づくイベント駆動シミュレーションは，従来シミュレータと比較して，計算コストを大幅に削減できる．ナイーブ・モデルでは，1 パケットの移動につき，send 信号の遷移と ack 信号の遷移の 2 イベントが発生する．このため，総イベント数はパケット数 P_{total} に比例する．一方，マクロフロー・モデルでは，全パケットの移動につき，唯一 $V(t)$ を求めるだけであり，イベント数はパケット数に依存しない．そのため，性能見積りに要する時間を 50%以上短縮でき，理想的には $\frac{1}{P_{total}}$ までシミュレーション時間を短縮できる．

両アプリケーションのシミュレーションに要した時間を表 6.3 に示す．マクロフロー・モデルは，P_{total} の増加に対しても，計算量を抑えることができるため，3 倍以上のシミュレーションの高速化を達成している．

6.4　オンチップ・シミュレーション回路の評価

提案したオンチップ・シミュレーション手法の回路コストを評価する．第 5 章で示したとおり，オンチップ・エミュレータは，DDP コアにエミュレーション機構を搭載したもので

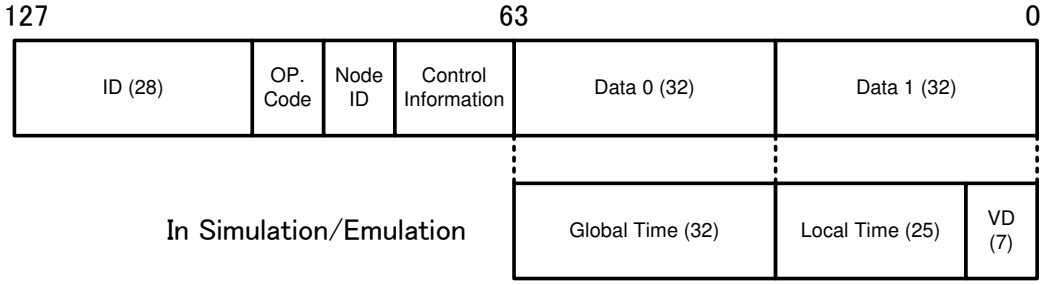

図 6.14　DDP パケット構成の一例

ある．また，オンチップ・シミュレータは，オンチップ・エミュレータにシミュレーション機構を加えたものである．このため，必要な回路は，コア数に応じて増加する．これに対して，DDP コアとそれぞれの機構の回路規模を比較して，オンチップ・シミュレーション手法の実現に必要な回路投資を定量的に示す．

回路の実装には，回路実現の実現可能性を実証するため，FPGA（field programmable gate array）を用いた．

6.4.1　エミュレーション回路

回路構成

エミュレータとして機能する DDP コアは，ALU 部にある分岐命令を実行するデータパスに BHM 機構を搭載すると共に，ステージ毎の転送制御信号線上に VD 機構を挿入する構成である．エミュレーションの実現に必要な回路コストを定量化するため，現行の DDP の ALU 部に BHM と VD を搭載して回路規模の増加率を示す．

現行の DDP コアは，1 データを 32bit で構成し，2 データに対して同時に演算を適用する SIMD 型の演算を実装している．1 パケットは，2 つのデータと演算命令で構成される．パケット構成を図 6.14 に示す．トレース駆動エミュレーションでは，データ値を保証する必要がないため，データ領域を活用して，パケットの時刻の管理や VD の選択信号の格納を行う．第 0 データ領域に，グローバル時刻を格納し，第 1 データ領域の上位 25bit にローカル時刻，下位 7bit に VD 選択信号を格納する．

分岐履歴では，アプリケーションの実行中に生じた分岐の方向を 1bit として格納するため，BH メモリの容量は，実アプリケーションにおける分岐命令の数と実行回数に基づき決まる．つまり，分岐命令数を N，単位時間あたりの分岐命令実行数を B，及び総実行時間を A とすると，BH メモリの容量は，$N \times B \times A[bit]$ である．このとき，部分シミュレーション／エミュレーションでは，対象プログラムを分割できるため，BH メモリの容量は削減できる．例えば，ラプラシアン 4 近傍フィルタでは，VGA 画像を対象に 3 ライン毎に分割して処理が可能であり，その場合，分岐方向の数は 307K である．しかし，一般に，分岐命令の数や実行回数はアプリケーション毎に違っており，筆者が所属する研究室のデータ駆動プログラム資産に基づくと，全命令に対して分岐命令が占める割合は，ラプラシアン 4 近傍フィルタで約 50%，DWT（離散ウェーブレット変換）処理で約 10%，さらに，ファイアウォール向け SPI（Stateful Packet Inspection） [33] で約 2%であった．つまり，BH メモリの容量は対象とするアプリケーションの特性に強く依存する．したがって，評価可能なプログラムの範囲と回路規模のトレードオフに基づき決定する必要がある．本評価では，BH メモリの容量を，研究室の資産を評価できる 400K[bit] として，回路規模を示す．

VD 機構が提供する遅延の数は，エミュレーションで設定する各ステージの遅延の数とマクロフロー・モデルに基づくシミュレーションで必要となるパケットの速度の数の多い方で決まる．マクロフロー・モデルでは，パケット数に応じてパケットの速度を変化できればよいため，60 段（最大で 60 パケット格納）の DDP では，60 種類の遅延が必要となる．また，パケットの複製の影響を反映したシミュレーションでは，複製時の速度を別に用意する必要があることから，60 段の DDP では，120 種類の遅延が必要である．一方，エミュレーションで設定する遅延量は対象システムに依存するため，実システムの設計における要求と回路規模とのトレードオフを見極めて判断する必要がある．本評価では，120 種類の遅延を提供できるように，直列実装法により 128 種類の遅延を選択可能な VD 機構を実装した．

回路規模

DDP コアの ALU に VD と BHM を搭載した回路を実装した．回路の実装に十分なロジック要素（LE）数を持つ，Altera 社の APEX20KE デバイスを用いて実装した．また，論理

表 6.4　回路規模の比較

	DDP	DDP+VD+BHM	Increased
# of LE[K]	44.5	44.7	0.4%

合成には MentorGraphics 社の LeonardoSpectrum を，配置配線には Altera 社の Quartus を用いた．エミュレーション機構を搭載した DDP コア回路の規模を，元の DDP コア回路と比較した結果を表 6.4 に示す.

本評価では，400K[bit] の BH メモリを用いたが，実効アドレスは GE#と ND#より直接求められるため，BH メモリの容量増加に伴う回路規模の増加は僅かな範囲に収まる．結果より，エミュレーション回路の追加による回路規模の増加は，1%未満であった.

6.4.2　シミュレーション回路

回路構成

シミュレータとして機能する DDP コアは，パケットの時刻を計測して更新する Start Time Stamper 及び Lap Time Stamper に加えて，マクロフロー・モデルに基づくシミュレーションを実現するための，Packet Queue 及び Macro-flow Engine を搭載する.

シミュレーション機構の実現可能性を実証するため，シミュレーション機構を搭載した 16bit のデータを持つ 37 段の DDP コアを実装した．前章で示した構成に従い，Start Time Stamper は，25bit のローカル時刻をカウントし，また，Lap Time Stamper は，32bit のグローバル時刻を更新できるように実装した．この際，16bit のデータ領域に格納できない 9bit のローカル時刻と 32bit のグローバル時刻を保持するデータラッチを新たに設けた．Packet Queue は，オーバフローを検出するため対象のプロセッサが収容できる最大のパケット数（すなわち総ステージ数）分の容量が必要である．本評価では，対象とする DDP コアは 62 段として 62 パケットを収容できるキューを実装した．また，Macro-flow Engine は，パケット数に応じてパケットの速度を変化できればよいため，最大数である 62 個のエントリを用意した.

表 6.5　回路規模の比較

	DDP	On-Chip Simulator	Increased
# of LE[K]	10.9	17.1	56%
RAM[K bit]	26.2	33.3	26%

表 6.6　シミュレーション時間

	DDP System	On-Chip Simulation	RTL Simulation
Measured [sec.]	3.5m	272m	$>$ 1 day
Ratio	1	76	$>$ 1M

回路規模

回路の実装には，Altera 社の FPGA である APEX20KC デバイスを用いた．また，論理合成には MentorGraphics 社の LeonardoSpectrum を，配置配線には Altera 社の Quartus を用いた．シミュレーション機構を搭載した DDP コア回路の規模を，元の DDP コア回路と比較した結果を表 6.5 に示す．

結果より，エミュレーション回路の追加による回路規模の増加は，約 5 割程度であった．これに比べて，現行の DDP コア構成に，シミュレーション機構を搭載する場合は，回路規模の増加は少なくなる．まず，設計した VD は規則的に同じものを配置しており，冗長な構成となっている．これに対して，パイプライン段間の遅延量を予め限定して VD を構成する方法がある．また，現行の DDP は，32bit のデータを 2 つ持つ SIMD 構成を取っており，2 つの 32bit のデータ領域を周回時間の格納に用いることができる．

シミュレーション時間

実装した回路よるシミュレーション時間を，現行のシミュレーション手法と比べた結果を表 6.6 に示す．結果より，既存のシミュレーション手法に比べて遥かに高速であり，システムの実行時間に比べて 80 倍以内に収まる高速なシミュレーションを実現できることを確認した．

6.5　結言

本章では，まず，ウェブパイプラインの応用例として，セルフタイム型パイプライン・キュー，セルフタイム型オメガ網ルータ，およびセルフタイム型パイプライン・ソータの LSI 回路を設計し評価した．ウェブパイプライン構成は，これらの高い機能性を実現でき，現行の専用処理に対して十分な処理性能を持つことを明らかにした．また，提案回路構成に基づくことで，従来の人手による設計方法と比べて，約 1/6 まで設計期間を短縮できることを確認した．本評価では，ウェブパイプラインの持つ一部の機能性のみを評価したものであり，今後の課題としては，パイプライン間の相互作用が実現できるすべての機能性を網羅的に検証することが残されている．

次に，マクロフロー・モデルの見積り精度とオンチップ・シミュレーション／エミュレーション回路の回路規模を評価した．提案モデルは，従来のモデルに比べて軽量でありながら，LSI 実装された実プロセッサの性能バラつきを見込んだシミュレーションが，従来モデルと同等の精度で行えることを示した．また，提案モデルに基づきシミュレータを実装した結果，従来モデルに比べてシミュレーションを 3 倍以上高速化でき，パケットを速度で捉えることによるシミュレーションの軽量化を示した．実際，提案モデルに基づくシミュレーション手法を実現するためのシミュレーション／エミュレーション機構を搭載した DDP コアを実装して，回路規模を評価した．結果，コアあたり約 50%の回路規模の増加に収まることが判った．数百以上のコアが収容できる将来の ULSI 集積技術を見据えれば，許容できる回路規模に収まっている．本評価では，シミュレーション／エミュレーションに必要な基本的な構成のみを実装して評価したが，シミュレータ／エミュレータ回路は，回路投資に応じて，柔軟性と高速性を向上させることができる．エミュレータ回路では，可変遅延 VD 機構の遅延の量や数，また分岐履歴 BH 機構のメモリ容量の制限があり，シミュレータ回路では，同時に評価するパケット数の制限などがある．課題としては，評価対象毎の最適な構成の定量的な導出が残されている．

第 7 章

結論

本論文では，今後の回路集積技術の高度化に伴い大規模・複雑化する集積システムの処理性能と設計・生産性を向上するためのシステム構成の要件を明らかにし，セルフタイム型パイプラインを構成原理としたシステム構成法を提示した．この中で，まず，処理性能の向上のために，パイプライン間相互作用制御回路に基づくウェブパイプライン構成を提案した．さらに，設計・生産性の向上のために，マクロフロー・モデルに基づくオンチップ・シミュレーション手法およびパイプライン回路構成を提案した．

ウェブパイプライン構成により，トランジスタ資源の許す限り，データフローに沿ってパイプラインを展開することができ，したがって，アプリケーションに内在する並列性を最大限に活用した，大規模パイプライン並列処理が実現できる．また，オンチップ・シミュレーション手法により，既存システム上で次期システムの模擬が可能になれば，設計現場で応用分野の専門家がハードウェア実装を待たずにシステム全体の定量的評価に基づいて要求に見合ったシステム・アーキテクチャを選択することができるため，設計の手戻りを局所化かつ極小化することができ，したがって，設計の短期間・低コスト化が期待できる．以下に本研究で得られた成果をまとめる．

第 2 章では，まず，大規模パイプライン並列処理を実現する，ウェブパイプライン構成を示し，ウェブパイプラインを構成するパイプライン処理の形態を分類した．各処理形態における柔軟なパイプライン構造を実現するには，パイプライン間における相互作用，すなわちデータの流れを自由に切り替えて指定された加工を施す流れ処理制御が必要であることを示した．これに対して，束データ方式に基づくセルフタイム型パイプライン STP の基本構成を示し，STP 間相互作用を実現する転送制御を明らかにし，新たに転送制御回路が必要で

あることを示した.

また，パイプライン・システムを迅速に評価するため，トレース駆動方式に基づく，オンチップ・シミュレーション手法を示し，シミュレーション回路規模を増大させる状態管理コストの削減には，新たな動作モデルが不可欠であることを示した．これに対して，STP 中のパケットの速度に着目すれば，ステージ毎のデータ転送を逐一模擬せずとも，STP のマクロな挙動を把握できることを示した．パケットの速度に着目することで，ハンドシェイクのプロトコルによらずに STP の挙動を示せるため，汎用性の高い動作モデルを定義できることを示した．新たな動作モデルの要件として，STP システムでは，(a) 環状のパイプライン構造と (b) 回路実装におけるデータ転送時間 $(T_f + T_r)$ のバラつきにより，3 段階に変化する性能特性を正確に表現する必要性を示した．さらに，シミュレーション回路はシステム本来の計算を補助しない点で，回路実装において可能な限り回路規模を抑える必要があることを示した.

第 3 章では，セルフタイム型パイプライン STP に基づくウェブパイプライン構造の基本的な構成要素として，二本のパイプライン間の相互転送制御回路を提案し，その設計制約を明らかにした．提案した回路に基づけば，複数のパイプライン中のデータが相互に影響し合って転送経路を局所的に選択できるため，自律的かつ動的に経路を選択する柔軟なパイプライン構造を LSI 上に構成可能となることを示した．このウェブパイプラインの特徴を生かせば，アプリケーションのデータフローを LSI 上に直接展開することで，高性能専用処理エンジンを構成することができることを示した.

また，将来の大規模システム LSI において，STP の局所的な転送制御を活用すれば，処理性能の向上のみではなく，ウェブパイプライン化により並列化された処理モジュール毎の，細粒度の省電力化が期待できることを示した．課題としては，ウェブパイプラインの応用として，コア間ネットワークなどの実用的なアプリケーションを検討し，提案回路の一般性を明らかにする必要があることを示した.

第 4 章では，パケットの平均的な速度 $V(t)$ に着目して，ステージ毎のハンドシェイク時間 $(T_{fi} + T_{ri})$ との関係を，総パケット数 P_{total} を基準にして代数的に規定した．これによ

り，P_{total} に応じた $V(t)$ を管理するのみで，パケットのフローを近似可能とした．また，$(T_{fi} + T_{ri})$ のバラつきを見込んだ $V(t)$ を予め設定することで，設計の初期段階から軽量かつ安全なシミュレーションが可能となることを示した．提案モデルを用いれば，$V(t)$ に応じてパケットの速度を動的に調整さえすれば，全てのパケットのフローを追跡できるため，シミュレータの実装を軽量化できることを示した．また，STP においてパケットの流量に応じて変動する，消費電力量などの動的なシステム性能の見積りへの応用も期待できることを示した．課題としては，パイプライン間相互作用に基づくウェブパイプラインによる柔軟なパイプライン構造のモデル化が残されていることを示した．

第 5 章では，シミュレーション／エミュレーション機能を持つ DDP コアの構成法を示した．提案した構成では，トレース駆動方式に基づいて，DDP コアを直接的に用いた対象プログラムの解釈・実行を可能としており，オンチップ・シミュレーション手法の実現に伴う回路規模の増加を抑えていることを示した．また，対象プログラムの実行タイミングをターゲットと一致させるためのエミュレーション／シミュレーション回路の構成法を示した．エミュレーション回路では，ステージ毎のハンドシェイク時間を動的に調節する機構により，ステージ数が一致していれば直接的な模擬を可能とした．シミュレーション回路では，エミュレーション回路を活用してパケットの速度を変化させることで，マクロフロー・モデルに基づくシミュレーションの実現のための回路規模の増加を抑えた．

提案方式に基づけば，マクロフロー・モデルによりモデル化された対象を高速に模擬可能となることを示した．また，ウェブパイプラインなどを含むシステムの場合，DDP コアを用いてモデル化されていない部分をソフトウェアでシミュレーションし，さらにシミュレーション／エミュレーション結果を統合すれば，システム全体が模擬できることを示した．課題としては，マルチコア構成において，実行時間の異なるシミュレーション／エミュレーションを限りなく並行して実行する手法の確立が残されていることを示した．

第 6 章では，まず，ウェブパイプラインの応用例として，セルフタイム型パイプライン・キュー，セルフタイム型オメガ網ルータ，およびセルフタイム型パイプライン・ソータの LSI 回路を設計し評価した．ウェブパイプライン構成は，これらの高い機能性を実現でき，

現行の専用処理に対して十分な処理性能を持つことを明らかにした．また，提案回路構成に基づくことで，従来の人手による設計方法と比べて，約 1/6 まで設計期間を短縮できることを確認した．評価では，ウェブパイプラインの持つ一部の機能性のみを評価したものであり，今後の課題としては，パイプライン間の相互作用が実現できるすべての機能性を網羅的に検証することが残されていることを示した．

次に，マクロフロー・モデルの見積り精度とオンチップ・シミュレーション／エミュレーション回路の回路規模を評価した．提案モデルは，従来のモデルに比べて軽量でありながら，LSI 実装された実プロセッサの性能バラつきを見込んだシミュレーションが，従来モデルと同等の精度で行えることを示した．また，提案モデルに基づきシミュレータを実装した結果，従来モデルに比べてシミュレーションを 3 倍以上高速化でき，パケットを速度で捉えることによるシミュレーションの軽量化を示した．実際，提案モデルに基づくシミュレーション手法を実現するためのシミュレーション／エミュレーション機構を搭載した DDP コアを実装して，回路規模を評価した．結果，コアあたり約 50%の回路規模の増加に収まることが判った．数百以上のコアが収容できる将来の ULSI 集積技術を見据えれば，許容できる回路規模に収まっていることを示した．課題としては，評価対象毎の最適な構成の定量的な導出が残されていることを示した．

これらの成果に対して，本論文では，ウェブパイプラインおよびシミュレーション手法を実現する上で基本的な回路構成を示すに留まっており，各章の結言において述べたとおり各種課題が残っている．しかし，それら課題を解決できれば，システム性能を飛躍的に向上させることができる．ウェブパイプラインは，処理を構成する演算と演算間のデータフローを直接パイプライン上に展開することで，演算に必要な転送制御を局所化および極小化して，パイプライン処理のオーバヘッドが極めて小さい並列処理が実現できる．こうした従来のパイプライン処理に加えて，実行時のデータの到着タイミングにより演算のデータ対を決定できる構成により，演算対象のデータ対が実行前に決まらない，ないしは決めなくてもよい処理に対して，実行時の偶然の一致（データ対の決定）を利用したパイプライン処理が実現できる．これにより，データ対の到着を待つ必要がなく，したがって，スムースなパイプライ

ン処理が期待できる．特に，処理の厳密性より，ある程度有意な出力を迅速に求めることが優先される処理への応用が期待できる．例えば，ウェブ検索やデータマイニングなどへの応用が考えられる．一方，オンチップ・シミュレーション手法は，既存手法に比べて飛躍的に迅速なシステム評価を可能とする．例えば，設計現場では，実システムの実行時間を 1 とすると，実システムの模擬には 1000∼10000 倍もの時間が必要と言われており，システムの大規模化を考慮すると実機と模擬の実行時間差は今後ますます大きくなると考えられる．提案手法を用いた場合，シミュレーションでは，1 命令の模擬がターゲットの実行時間と同程度の時間内で行え，またエミュレーションでは，サブ・プログラムの模擬がターゲットの実行時間と同程度の時間内で行えることから，100 倍以上のシミュレーション時間の短縮が期待できる．

将来の集積システムは，半導体集積技術やアナログ・デジタル混載技術の発展に伴い，チップ上に全システム要素を集積することで，複合機能を搭載し小型な高い移動性／携帯性を持った計算機を実現すると考えられる．この結果，通信ネットワーク技術の発達とも相まって，情報サービスのさらなる高度化が期待できる．しかしながら，こうした大規模集積システムの実現の要となる回路微細加工集積においては，トランジスタの微細化につれ，スイッチ動作しない待機時の漏れ電流による消費電力が増大しており，素子の微細化による動作の高速化、省電力化、および高集積化の妨げとなっている．また，消費電力の増大は，発熱による半導体素子特性の変化に起因する誤動作や故障の原因，さらに機器の小型化に不可欠である電源供給源の延命の足枷となっており，したがって，情報処理回路の低消費電力／低発熱化は極めて重要な課題となっている．こうした低消費電力の要求に対して，データが処理されるパイプライン段でのみ電力が消費される自律的な省電力特性を有したセルフタイム型パイプラインに着目して，低消費電力化技術が研究されている．筆者らも，既に，待機状態のパイプライン段への電力供給を局所的に遮断ないし低減する電源遮断回路と電圧スケーリング回路を提案している [35]．こうした構成を活用すれば，今後もシステムの集積化を深化させることができ，セルフタイム型パイプライン構成の優位性を明確に示すことができる．本論文で提案したウェブパイプラインにこれらの省電力化技術を適用して，さらに，

マクロフロー・モデルを消費電力も含むシステム評価に拡張すれば，本論文の提案は，将来的にも有効な手法として発展することが期待できる．

提案手法が，回路投資に見合ったあるいはそれ以上の効果を持って，システム設計の現場で活用されることに期待して，本論文の結びとしたい．

謝辞

本研究の全過程を通じて，終始懇切丁寧なる御指導，御鞭撻を賜った岩田誠教授に心より感謝の意を表します.

本研究の遂行において，懇篤なる御指導をいただき，常に温かい御激励をくださった，大阪大学名誉教授，高知工科大学名誉教授寺田浩詔先生に深謝の意を表します.

本論文の作成にあたり，様々な御高配を賜ると共に，副査として御指導，御助言をいただきました，高知工科大学島村和典教授，冨澤治教授，首都大学東京西谷隆夫教授，および高知工科大学酒居敬一講師に深く感謝いたします.

筆者が岩田研究室に在籍して以来，一方ならぬ御心配りを頂き，随所で大変御世話になりました，九州大学助手大森洋一先生に心より感謝の意を表します.

データ駆動プロセッサや研究アイデアに対して技術的な御討論，御助言をいただいたシャープ株式会社村松剛司氏に心よりお礼申し上げます.

日本テレコム株式会社林秀樹氏には，研究活動に対して数々の貴重な御助言をいただきました．心よりお礼申し上げます.

本研究を進めるにあたって，NEC エレクトロニクス株式会社森川大智氏にはデータ駆動プロセッサ向けシミュレータの実装をはじめ、様々な面でお世話になりました．また，同じく小笠原新二氏，および東北大学小松和寛氏には，共同研究者として支えになって頂きました。心よりお礼申し上げます.

大学院修士課程において，共に研究活動に励んだ，Ruhui Zhang 氏，および志摩浩氏に感謝の意を表します.

最後ではありますが，日頃からの御支援をいただきました岩田研究室の皆様，並びに関係者の皆様に心からお礼申し上げます.

参考文献

[1] International Technology Roadmap for Semiconductors, http://www.itrs.net/.

[2] O. A. Petlin, S. B. Furber, “Built-In Self-Testing of Micropipelines,” Proc. Third International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC ’97), Eindhoven, Netherlands, pp.22–29, Apr. 1997.

[3] M. Takamiya, M. Mizuno, and K. Nakamura, “An On-Chip 100GHz-Sampling Rate 8-channel Sampling Oscilloscope with Embedded Sampling Clock Generator,” ISSCC 2002, Session 11, No. 2, San Francisco, U.S.A., Feb. 2002.

[4] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, K. Yazawa, “The Design and Implementation of a First-Generation CELL Processor,” ISSCC 2005, Session 10, No. 2, San Francisco, U.S.A., Feb. 2005.

[5] M. Edahiro, S. Matsushita, M. Yamashina, N. Nishi, “A Single-Chip Multiprocessor for Smart Terminals,” IEEE Micro, Vol. 20, Issue 4, pp.12–20, 2000.

[6] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, V. De, “Parameter Variations and Impact on Circuits and Microarchitecture,” Proc. 40th Conference on Design Automation, pp.338–342, Anaheim, U.S.A., June. 2003.

[7] 寺田浩詔, “超高速・低電力・高機能な新システム・アーキテクチャの考え方ー刷り込みからの脱却ー”, 信学技報, Vol.103, No.509, pp.29–34, Dec. 2003.

[8] H. Terada, S. Miyata, and M. Iwata, “DDMP’s: self-timed super-pipelined data-driven multimedia processors,” Proc. IEEE, Vol.87, No.2, pp.282–296, Feb. 1999.

[9] I.E. Sutherland, “Micropipelines,” Commun. ACM, Vol.32, No.6, pp.720–738, Jun. 1989.

[10] A. Lines, “Nexus: an asynchronous crossbar interconnect for synchronous system-on-chip designs,” Proc. 11th Annual Hot Interconnects Conf., pp.2-10, Stanford, U.S.A., Aug. 2003.

[11] V. Ekanayake, C. Kelly IV, and R. Manohar, “An Ultra Low-Power Processor for Sensor Networks,” Proc. 11th International Conference on Architectural Support for Programming Languages and Operating Systems, pp.27–36, Boston, U.S.A., Oct. 2004.

[12] A. Takamura, M. Kuwako, M. Imai, T. Fujii, M. Ozawa, I. Fukasaku, Y. Ueno, and T. Nanya, “TITAC-2: a 32-bit asynchronous microprocessor based on scalable-delay-insensitive model,” Proc. 1997 Int. Conf. on Computer Design, pp.288–294, Austin, U.S.A., Oct. 1997.

[13] T. Villiger, H. Kaslin, F.K. Gurkaynak, S. Oetiker, and W. Fichtner, “Self-timed ring for globally-asynchronous locally-synchronous systems,” Proc. 9th Int. Symp. on Asynchronous Circuits and Systems, pp.141–150, Vancouver, Canada, May 2003.

[14] D. Morikawa, M. Iwata, and H. Terada, “Super-pipelined implementation of IP packet classification,” Journal of Intelligent Automation and Soft Computing, Vol.10, No.2, pp.175-184, 2004.

[15] R.F. Sproull, I.E. Sutherland, C.E. Molnar, “The counterflow pipeline processor architecture,” IEEE Des. Test, Vol. 11, No.3. pp.48–59, Jul. 1994.

[16] 鹿毛哲郎, “VLSI 回路シミュレーション,” 信学誌, Vol.83, No.11, pp.838–842, Nov. 2000.

[17] A. Xie and P. A. Beerel, “Accelerating markovian analysis of asynchronous systems using string-based state compression,” Proc. 4th Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, pp.247-260, San Diego, U.S., Mar. 1998.

[18] S. Chakraborty and R. Angrish, “Probabilistic timing analysis of asynchronous

systems with moments of delays," Proc. 8th Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, pp.99–108, Manchester, U.K., Apr. 2002.

[19] J. C. Ebergen, S. Fairbanks, and I. E. Sutherland, " Predicting Performance of Micropipelines Using Charlie Diagrams, "Proc. 4th Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, pp. 238-246, San Diego, U.S.A., Mar. 1998.

[20] W.M. Zuberek, "Event-driven simulation of timed petri net models," Proc. 33rd Annual Simulation Symp., pp.91–98, Washington, U.S.A., Apr. 2000.

[21] 児玉祐悦，片下敏宏，佐谷野健二, "リコンフィギュアラブルシステム REX への並列計算機 EM - X の実装," 研究報告 ARC-152, pp.109-114, Mar. 2003.

[22] J. Sparsϕ, "Asynchronous Circuit Design A Tutorial," Kluwer Academic Publishers, London, 2001.

[23] C.J. Myers，米田友洋（訳），"非同期式回路の設計，" 共立出版社，東京，2003.

[24] M. Iwata, M. Ogura, Y. Ohishi, H. Hayashi, and H. Terada, "100MPacket/s Fully Self-Timed Priority Queue: FQ," ISSCC 2004, San Francisco, U.S.A., Session 8, No.1, Feb. 2004.

[25] S. Furber, "Computing without clocks: Micropipelining the ARM processor," Asynchronous Digital Circuit Design, G. Birtwistle and A. Davis, eds., pp. 211-262. Springer Verlag, 1995.

[26] J. A. シャープ，富田眞治（訳），"データ・フロー・コンピューティング，" サイエンス社，東京，1987.

[27] R. F. Cmelik and D. Keppel, "Shade: A Fast Instruction-Set Simulator for Execution Profiling," ACM SIGMETRICS Performance Evaluation Review，Vol.22, No.1，pp.128-137，1994.

[28] 三宮秀次, 大森洋一, 岩田誠, "シミュレーション対象のハードウェア非依存な命令セット FIS の提案," 平成１３年度電気関係学会四国支部連合大会, 10-3, 2001.

[29] 中田尚，大野和彦，中島浩，"高性能マイクロプロセッサの高速シミュレーション，" 先

進的計算基盤システムシンポジウム（SACSIS 2003）論文集， pp.89–96，May. 2003.

[30] 岡本俊弥，“データ駆動型メディアプロセッサ，” 情報処理学会誌，Vol.39，No.3，pp.208–214，Mar. 1998.

[31] T.E. Williams and M.A. Horowitz, “A zero-overhead self-timed 160-ns 54-b CMOS divider,” ISSCC 91, Session 5, No.5, Feb. 1991.

[32] S. Ogasawara, S. Sannomiya, Y. Omori, and M. Iwata, “An On-Chip Trace-Driven Emulation for Self-Timed Data-Driven Processors,” Proc. International Conference on Next Era Information Networking NEINE’04, Kochi, pp. 435-440, Sep. 2004.

[33] R. Zhang, Y. Shirane, M. Iwata, W. Su and Y. Zheng，“High Speed Stateful Packet Inspection in Embedded Data-Driven Firewall,” Proc. International Conference on Next Era Information Networking NEINE’04，Kochi, Sep. 2004.

[34] 林秀樹, 岩田誠, 島村和典,”異種混合ネットワークにおける自律型フロー分散制御方式,” 情報処理学会論文誌, Vol.45, No.2, pp.426-437, Feb. 2004.

[35] K. Miyagi, S. Sannomiya, K. Sakai, M. Iwata and H. Nisikawa, “Autonomous Power-Supply Control for Ultra-Low-Power Self-Timed Pipeline,” Proc. International Conference on Parallel and Distributed Processing Techniques and Applications, pp.704–709, Las Vegas, U.S.A., Jul. 2008.

付録 A

関連業績

A.1 査読付き論文誌

1. 三宮秀次, 大森洋一, 酒居敬一, 岩田誠, “自己タイミング型パイプラインシステムの性能見積りモデル,” 電子情報通信学会論文誌 A.（条件付き採録）
2. K. Komatsu, S. Sannomiya, M. Iwata, H. Terada, S. Kameda, K. Tsubouchi, “Interacting Self-Timed Pipelines and Elementary Coupling Control Modules,” IEICE TRANSACTIONS on Fundamentals.（条件付き採録）

A.2 国際会議

1. S. Sannomiya, N. Kagawa, K. Sakai, M. Iwata, “A Data-Driven On-Chip Simulation Module and Its FPGA Implementation,” in Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA’08), pp.710–716, 2008.
2. K. Komatsu, S. Sannomiya, M. Iwata, “Essential Building Circuits for Self-Timed Web-Pipeline,” in Proceedings of International Conference on Next Era Information Networking (NEINE’07), pp.148–149, 2007.
3. N. Kagawa, S. Sannomiya, M. Iwata, “Circuit Design of A Data-Driven On-Chip Simulation Module,” in Proceedings of International Conference on Next Era Information Networking (NEINE’07), pp.489–450, 2007.

4. T. Kubo, K. Komatsu, S. Sannomiya, M. Iwata, “A Study on Self-Timed Pipeline Sorter,” in Proceedings of International Conference on Next Era Information Networking (NEINE’07), pp.451–452, 2007.
5. S. Sannomiya, K. Komatsu, M. Iwata, “A Self-Timed Pipeline Circuit for Low-Power Surrounding LSI Chips,” in Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA ’07), pp.613–619, 2007.
6. K. Komatsu, S. Sannomiya, M. Iwata, “A Bi-directional Transfer Control for Multi-dimensional Self-timed Pipeline,” in Proceedings of International Conference on Next Era Information Networking (NEINE’06), pp.399–401, 2006.
7. T.Mitsui, K.Komatsu, S.Sannomiya, M.Iwata, “One-Way Data Transfer Circuit between Self-Timed Pipelines” in Proceedings of International Conference on Next Era Information Networking (NEINE’06), pp.402–404, 2006.
8. S. Sannomiya, Y. Omori, K. Sakai, M. Iwata, “An On-Chip Macro-Simulation Mechanism of Self-Timed Pipelined Systems,” in Proceedings of International Conference on Next Era Information Networking (NEINE’06), pp.133–138, 2006.
9. K. Komatsu, S. Sannomiya, M. Iwata, “Systematic Design of Basic Self-Timed Pipeline Circuit Modules,” in Proceedings of International Conference on Next Era Information Networking (NEINE’05), pp.542–547, 2005.
10. S. Ogasawara, S. Sannomiya, Y. Oomori, M. Iwata, “Implementation of An On-Chip Trace-Driven Emulation for Self-Timed Data-Driven Processors,” in Proceedings of International Conference on Next Era Information Networking (NEINE’05), pp.548–553, 2005.
11. S. Sannomiya, S. Ogasawara, M. Iwata, “A Study on Emulation-Based Rapid Evaluation of Self-Timed Super-Pipelined Systems,” in Proceedings of International Conference on Next Era Information Networking (NEINE’05), pp.554–560, 2005.

12. S. Sannomiya, Y. Omori, M. Iwata, “A Fast Simulation Scheme for Self-Timed Super-Pipelined Systems,” in Proceedings of International Conference on Next Era Information Networking (NEINE’04), pp. 127–138, 2004.
13. S. Ogasawara, S. Sannomiya, Y. Omori, M. Iwata, “An On-Chip Trace-Driven Emulation for Self-Timed Data-Driven Processors,” in Proceedings of International Conference on Next Era Information Networking (NEINE’04), pp.435–440, 2004.
14. S. Sannomiya, Y. Omori, M. Iwata, “A Macroscopic Behavior Model for Self-Timed Pipeline Systems,” in Proceedings of Seventeenth Workshop on Parallel and Distributed Simulation (PADS2003), pp.133–140, 2003.

A.3　国内学会

1. 小松和寛, 三宮秀次, 岩田誠, 亀田卓, 坪内和夫, “セルフタイム型ウェブパイプラインの相互作用回路の検討,” 信学技報, ICD2007-129, pp.53–58, 2007.
2. 三宮秀次, 大森洋一, 酒居敬一, 岩田誠, “自己タイミング型パイプラインシステムのオンチップ・マクロシミュレーション手法,” 情報処理学会研究報告 ARC-169, pp.145–150, Aug. 2006.
3. 三宮秀次, 小笠原新二, 岩田誠, “オンチップ評価機構を搭載した自己タイミング型パイプラインシステムの検討,” 情報処理学会研究報告 ARC-165, pp.93–98, Dec. 2005.
4. 三宮秀次, 大森洋一, 岩田誠, “自己タイミング型パイプラインシステムのマクロシミュレーションモデル,” 情報処理学会研究報告 EVA-8, pp.61–66, Mar. 2004.
5. 三宮秀次, 大森洋一, 岩田誠, “自己同期型パイプラインシステム向けマクロフローモデル,” 情報処理学会研究報告 ARC154, pp.43–48, Aug. 2003.
6. 三宮秀次, 大森洋一, 岩田誠, “自己同期型パイプラインシステムのマクロフローモデルの提案,” Forum on Information Technology, C-017, Sep. 2003.
7. 三宮秀次, 大森洋一, 岩田誠, “シミュレーション対象のハードウェア非依存な命令セッ

ト FIS の提案,” 平成１３年度電気関係学会四国支部連合大会, 10-3, 2001.

A.4　特許

1. データ駆動型情報処理装置（出願中：特願 2008-70446 号）