平成 13 年度

学士学位論文

FPGA を用いたデータ駆動プロセッサの回路実現法

FPGA-Specific Circuit Implementation of Data-Driven Processor

1020267　　小倉 通寛

指導教員　　岩田 誠

2002 年 2 月 8 日

高知工科大学 情報システム工学科

要 旨

FPGA を用いたデータ駆動プロセッサの回路実現法

小倉 通寛

近年の半導体集積化技術の発展により、数億個のトランジスタＵＬＳＩを搭載したプロセッサが登場してきた。しかしながら、これら主流のプロセッサは同期クロック方式のノイマン型プロセッサであり、極度に集積化が進むにつれて、さらなる高速化、集積化、省電力化を達成することが困難になってきている。この問題は、ノイマン型処理方式が中央集中制御による逐次的処理を動作原理としていることにある。この問題を解決し得るアーキテクチャとして自己タイミング型パイプライン機構が提案されている。この自己タイミング型パイプライン機構と親和性が高いデータ駆動方式を採用したＶＬＳＩ（ＤＤＭＰ）が、すでに開発され実用化されている。このアーキテクチャによって、柔軟なデータ転送が可能になると共に、配線を局所化することができる。そのため、同期クロック方式のノイマン型プロセッサに対して省電力、集積化、スループット性能において高い潜在能力を有している。

本論文では、この自己タイミング型パイプライン機構を基本アーキテクチャとした特定処理向きデータ駆動プロセッサの開発環境を構築することを目的として、FPGA チップ上にデータ駆動プロセッサを実装するために必要な自己タイミング型パイプライン機構の実現法と、パケットの複製制御回路、消去制御回路を提案した。そして、提案回路を実装したデータ駆動プロセッサと既存のソフトウェアシミュレータを比較し、機能検証を高精度、高速に行えることを確認した。さらに、FPGA のリソース消費量を調査して、マルチプロセッサの機能検証について考察した。結果、大規模な DDP システムを FPGA チップ上に実装可能なことが確認された。

キーワード　自己タイミング型パイプライン機構、データ駆動型プロセッサ、FPGA

Abstract

FPGA-Specific Circuit Implementation of Data-Driven Processor

OGURA, Michihiro

By evolution of senmiconductor integration technology, the processors that carry hundreds of millions of transistors had been developed. Almost of those current processors implement the von Neumann principle and those are realized by synchronous circuits. However, it becomes difficult to attain the further improvement in the speed, integration, and power-saving. These problems arise from the nature of the sequential processing. The self-timed pipeline was introduced to settle these problems. The self-timed pipeline has a strong affinity to the data-driven principle. Transmission of the data is asynchronously conducted by local transfer control units in the self-timed data-driven architecture (DDP). Therefore, flexible data transmission can be realized and wiring can be localized by these control units. Since the data-driven processor becomes free from the problems mentioned above, it has more high capability than the von Neumann type processor.

This paper aims at building the developping environment of the application-specific data-driven processors made of the self-timed pipeline. To implement a data-driven processor on an FPGA chip, the self-timed pipeline, the packet copy control circuit, and packet elimination control circuit were proposed. The data-driven processor which implemented the proposed circuit was compared with the existing software simulator. Consequently, they have high precision and high speed for functional verification. Furthermore, the amount of resource consumption of FPGA was investigated and functional verification of a multiprocessor was considered and showed a good result for large scale DDP system.

key words self-timed pipelene, data-driven processor, FPGA

目次

第 1 章　序論　1

第 2 章　自己タイミング型パイプライン機構とデータ駆動プロセッサ構成　5

2.1　緒言 5

2.2　自己タイミング型パイプライン機構 5

2.3　データ転送制御回路 7

2.4　DDP 設計方針 9

2.4.1　カスタム LSI 設計フロー 9

2.4.2　FPGA による回路設計 11

2.5　結言 12

第 3 章　FPGA チップ上での自己タイミング型パイプラインの実現法　14

3.1　緒言 14

3.2　クロック入力データ転送制御回路 14

3.3　パケット複製制御回路 17

3.4　パケット消去回路 20

3.5　結言 22

第 4 章　DDP の FPGA 実装と機能検証　23

4.1　緒言 23

4.2　データ駆動プロセッサ使用 23

4.2.1　主な仕様 23

4.2.2　DDP 内部パケットフォーマット 25

4.2.3 実装命令 25
4.3 データ駆動プロセッサ実装 28
4.3.1 合流部構成 28
4.3.2 待ち合わせメモリ部構成 29
4.3.3 演算部構成 34
4.3.4 プログラム保管部構成 34
4.3.5 分岐部構成 38
4.4 DDP 機能検証 39
4.5 結言 41

第 5 章 性能評価 **42**
5.1 緒言 42
5.2 性能評価 42
5.3 結言 44

第 6 章 結論 **45**
6.1 結論 45

謝辞 **48**

参考文献 **49**

図目次

2.1 自己タイミング型パイプライン機構 6
2.2 自己タイミング型パイプライン動作図 6
2.3 C 素子回路図 7
2.4 C 素子状態遷移図 8
2.5 FPGA 構成 10
2.6 FPGA を用いたディジタル回路設計フロー 12

3.1 C 素子回路図（クロック入力） 15
3.2 自己タイミング型パイプライン構成（クロック入力） 16
3.3 自己タイミング型パイプライン構成（クロック入力） 16
3.4 遅延回路を用いたパケット複製制御回路 17
3.5 遅延回路を用いないパケット複製制御回路 18
3.6 Copy Control の状態遷移図 19
3.7 Pulse Generator 回路図 19
3.8 遅延を用いたパケット消去制御回路図 20
3.9 遅延を用いないパケット消去制御回路図 21
3.10 パケット消去回路図 21
3.11 パケット消去回路状態遷移図 22

4.1 DDP 基本構成要素 24
4.2 DDP 内部基本フォーマット 25
4.3 MM から FP へのパケットフォーマット 25
4.4 FP から CPS へのパケットフォーマット 25
4.5 Merge 構成図 28

4.6 ArbitrationControl 回路図 29
4.7 MM 構成図 30
4.8 Hash Address Generator, CST input Packet Format 32
4.9 CST output Packet Format 32
4.10 Flag Generator input Packet Format 32
4.11 Flag Generator output Packet Format 33
4.12 Mode Selector output Packet Format 33
4.13 Hash Memory input output Data Packet Format 33
4.14 FP 部構成図 34
4.15 Cache Program Memory Format 35
4.16 CPS 部構成図 35
4.17 CPS input Packet 37
4.18 Load Cache Program output Packet Format 37
4.19 VCG output Packet Format 38
4.20 Selector output Packt Format 38
4.21 Branch 部構成図 39
4.22 機能検証フローグラフ 40

表目次

2.1 C 素子の入力論理表 7
2.2 LSI 実現方式別の評価 11

4.1 オペレーションコードの定義 26
4.1 オペレーションコードの定義 27
4.2 Merge における信号線定義 29
4.3 Mode Selector における信号線定義 30
4.4 Mode flag の定義 33
4.5 パケットフォーマットにおけるフラグの定義 34
4.6 分岐命令・分岐命令結果フラグの定義 37
4.7 VCG Code の定義 39
4.8 投入パケット 40
4.9 出力パケット 40

5.1 フローグラフ実行結果 43
5.2 20 万ゲート FPGA リソース消費量 43

第１章

序論

マイクロプロセッサは１９７０年代初頭の誕生以来、著しい性能向上を続けてきた。このような著しい性能向上をもたらした最大の要因は、プロセス技術の進展によってもたらされた素子の微細化である。これによって集積規模の拡大や比例縮小に伴うスイッチング速度の向上がもたらされ、基本設計を変更することなく性能を向上させることができた。このため、新しいアーキテクチャを採用するリスクを負うよりも、従来のノイマン型処理方式がマイクロプロセッサのアーキテクチャとして、マイクロプロセッサ誕生以来ずっと採用されてきた。しかしながら、ノイマン型処理方式は、極度に集積化が進むにつれて高速化、集積化、省電力化を達成することが困難になってきている。この問題は、ノイマン型処理方式が中央集中制御による逐次的処理を動作原理としていることに起因している。このような状況を反映して、自己タイミング型パイプライン [1] を用いた並列処理の導入によるアーキテクチャの改善が性能の限界を打ち破る技術として期待を集めている。上記したノイマン型処理方式と比較すると以下のようなことが言える。

- 高速化
  - トランジスタの微細化技術の発展によって素子の動作速度は向上するが配線遅延時間は向上しない。そのためバスを介したノイマン型処理方式のデータ通信速度は相対的に遅くなる。さらに 1 チップに高度なシステム機能を詰め込んだシステムＬＳＩが注目されている今日、集積規模が増大する傾向にあるためバス配線長が長くなり、そのバスを介したデータ通信速度がさらに遅くなる。また、高速化のためにノイマン型のアーキテクチャに. おいても機能モジュールの並列化やパイプライン

化による処理スループット向上が図られている。しかし、中央集中制御による逐次処理を前提としたアーキテクチャの場合、読み出した命令と実行中の命令との間にデータ依存関係が存在すると、複数の機能モジュールや深いパイプライン段を活性化するだけの並列度が得られず、性能向上に限界が生じる。一方、自己タイミング型パイプライン機構の場合、配線の局所化によって平均遅延性能を上げることができる。また、データ駆動処理方式はフローグラフを用いたプログラムの記述を用いるとパイプラインハザードが存在しないようなプログラムを記述できるので、深いパイプライン段を活性するだけの並列度が得られる。

- 消費電力
  - 自己タイミング型パイプライン機構はデータ駆動方式と非常に親和性が高い。（この処理方式は処理に必要なデータが揃う事によって駆動する）よって、この処理方式を用いれば必要なときに必要な場所でのみ回路が動作する。一方、クロックで同期を採っているノイマン型の場合、基本的に回路全体が常に駆動しており、また回路の末端までクロック信号を伝達するために高い電圧を掛ける必要があるため、より多くの電力を消費する。よって前者が低消費電力の点で有利なことは明らかである。
- 安定性
  - クロック信号を発信して同期をとらないために環境変動への体制が優れており、タイミングのずれをほとんど気にしなくても、回路は動作し続けるので温度変動や電源電圧の変動に非常に強い回路が実現できる。また、ノイズ発生が大きく低減されるため回路の誤動作も防ぐことができる。

上記のような利点を有する自己タイミング型パイプライン機構をもとにしたシステムであり、また処理方式としてデータ駆動方式を採用したデータ駆動型ＶＬＳＩマルチプロセッサ DDP（Data Driven Processor）[1] が実用化されている。上記の優れた特徴をもつ DDP であるが、回路の集積化、命令の複合化が進み、システムの大規模化・複雑化が進ん

でいるため、システムの事前評価が容易に行えない。よって、本研究では、FPGA（Field Programmable Gate Array）チップ上に DDP を実装し、DDP の開発環境を構築することを目的とする。FPGA チップ上に DDP を実装する事により、自己タイミング型パイプライン機構を基本アーキテクチャとした特定処理向き DDP の機能検証、及び性能見積もりを高精度で高速に行えるようになる。現在、ＤＤＰにおける特定処理向きの事前評価はソフトウェアによるシミュレータで行っているが、処理に膨大な時間やマシンパワーを必要とするため、計算量削減の手段として評価対象の回路を抽象化している。そのために、高い精度でシミュレーションが可能な RTL(Register Transfer Level) 検証を行えない。また、短い開発サイクルの実現、設計資産の効率的な蓄積等、開発環境の飛躍的な改善を行う事ができる。これによって、特定処理向けにＰＥ (Processing Element) の構成を変えても設計資産の再利用ができるため、柔軟な対応ができ迅速な開発を行うことができる。

現在、殆どの FPGA は同期回路に特化して設計されているため、遅延回路をサポートしていない。これは、データ転送制御回路間に入れるべき遅延を適切に挿入することができないことを意味している。また、DDP を構成する上で必要となるパケットの複製制御回路と消去制御回路は、遅延回路を用いて実装されており、FPGA チップ上に DDP を実装する場合、LSI で実装されている DDMP のパケット複製制御回路、消去制御回路を利用することが不可能である。よって、これらを解決するために本論文では、解決策としてクロックをデータ転送制御回路に入力し、これを遅延の代わりとした。さらに、FPGA チップ上に実装可能な DDP の回路を実現するために、遅延回路を用いないパケットの複製制御回路、消去制御回路を提案する。

以下、第２章では、各データ転送制御回路間に適切な遅延が採られている LSI での自己タイミング型パイプライン機構とデータ駆動プロセッサ構成について詳しく述べる。また、フルカスタム LSI 設計の問題点をあげ、FPGA を用いた DDP の回路設計法について述べる。第３章では、FPGA チップ上で動作する、遅延をクロックで入力して動作するデータ転送制御回路について述べるとともに、それを用いたパケット複製制御回路、消去制御回路を提案する。第４章では、本研究で実装した DDP の構成について詳細に述べる。第５章では、

第３章で提案した回路を DDP に実装し、既存のソフトウェアシミュレータと比較をおこない、性能検証を行う。第６章は、本論文の結論であり、自己タイミング型パイプライン機構を有するデータ駆動処理方式のプロセッサを FPGA 上に実装するために不可欠な、パケット複製制御回路、消去制御回路に関する本研究の成果を総括し結論を述べる。

第 2 章

自己タイミング型パイプライン機構とデータ駆動プロセッサ構成

2.1 緒言

この章では、まずノイマン型処理方式の根源に内在する問題点を原理的に解決可能な自己タイミング型パイプライン機構の構成、特質を詳しく述べる。次にフルカスタム LSI 設計の問題点をあげ、その解決法として FPGA チップ上に設計した回路を実装して機能検証する方法を提案する。つづいて、それを用いた設計手法について述べる。

2.2 自己タイミング型パイプライン機構

自己タイミング型パイプライン機構の基本構成は、図 2.1 に示すように、パイプライン間で転送されるデータを保持するデータラッチ DL_i（フリップフロップの接続と考えられる)、データ処理回路 L_i、ならびに DL_i の開閉をハンドシェイクプロトコルによって制御するデータ転送制御回路 C_i から構成される。

自己タイミングパイプライン機構の基本的な処理の流れは次の 2 通りである。

状態その 1

図 2.2 に示すように DL_i が有意なデータを保持していない場合には、データラッチ DL_{i+1} の状態に関係なく DL_{i-1} より DL_i にデータが転送される。

状態その 2

図 2.2 に示すように DL_{i+1} に有意なデータが保持されているならば、DL_i はデータの転送を待機する。

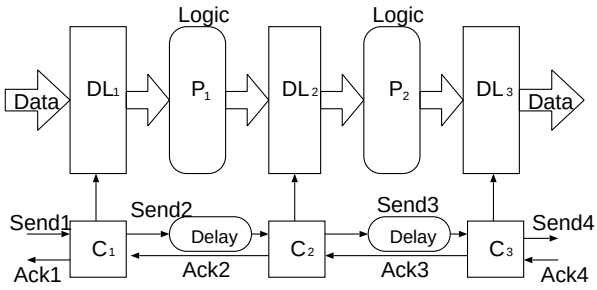

図 2.1　自己タイミング型パイプライン機構

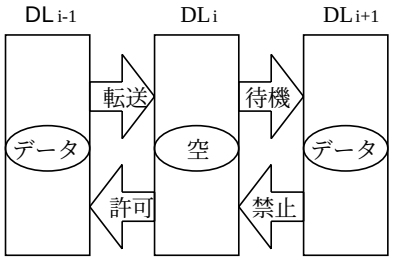

図 2.2　自己タイミング型パイプライン動作図

このような構成を採ることによって、パイプライン中の任意の場所で、データの流れが一時的に停止状態になったとしても、停止場所より後段のデータラッチにデータが順に詰められることになる。このため、自己タイミング型パイプライン機構は、一種のデータの緩衝能力を有している。

自己タイミング型パイプラインにおけるデータ転送制御は、データラッチに接続された C 素子と呼ばれる転送制御回路によって行われている。C 素子は転送要求信号（図中の Send）と転送許可信号 (図中の Ack) がデータ処理回路 P_i の処理遅延時間と完全な同期を行った上で動作し、それに伴ってデータラッチに開閉信号を発生するメカニズムになっている（図 2.1)。

2.3　データ転送制御回路

前節において、自己タイミング型パイプライン機構の構成及び動作について説明した。本節では、データ転送回路について詳しく述べる。C 素子の基本動作は、前段からの転送許可信号と後段からの転送要求信号が入力された時、前段への転送許可信号、及び後段への転送許可信号を出力する。つまり、2 つの信号の入力を受信して 2 つの信号の出力を発信する順序回路と言える。次に現在、実際にデータ駆動プロセッサ DDMP に使用されている C 素子の回路を図 2.3 に示す。この C 素子は、転送要求信号と転送許可信号を表 2.1 のように定

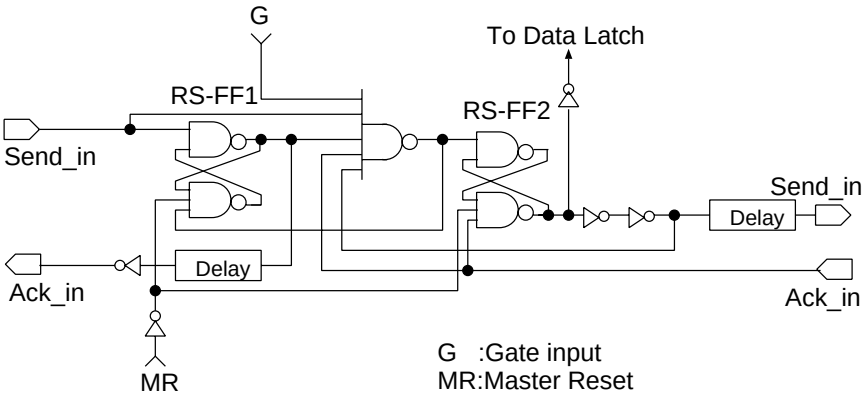

図 2.3　C 素子回路図

られており、ゲートには通常 “1” が入力される。もし、“0” が入力されるならば、その C 素

表 2.1　C 素子の入力論理表

	0	1
Send_in	要求	待機
Ack_in	禁止	許可

子はノンアクティブになる。MR に “1” が入力されると、その C 素子は初期状態にリセットされる。DDP の内部では、データの分岐や合流が頻繁に行われるため、様々なタイミングで信号が C 素子に入力されるため、RS-FF1 や RS-FF2 によって、様々なタイミングで入力される転送要求信号 (Send_in) や転送許可信号 (Ack_in) のタイミングのずれを完全に

押さえ込んでいる。初期状態で Send_in にパルスが入力されると、5 入力 nand の入力が全て “1” になり、RS − FF の R 側の入力に “0” が入力され DL が開く。そして、Send_out からは転送要求信号が送信され、その信号がすぐさま Ack_in に入力されて DL が閉じる。データ転送が正常に行われるときは、この動作を繰り返す。図 2.4 に示したのは、この C 素子の状態遷移図である。この遷移表から、正常なデータ転送が行われているときは Q1 →

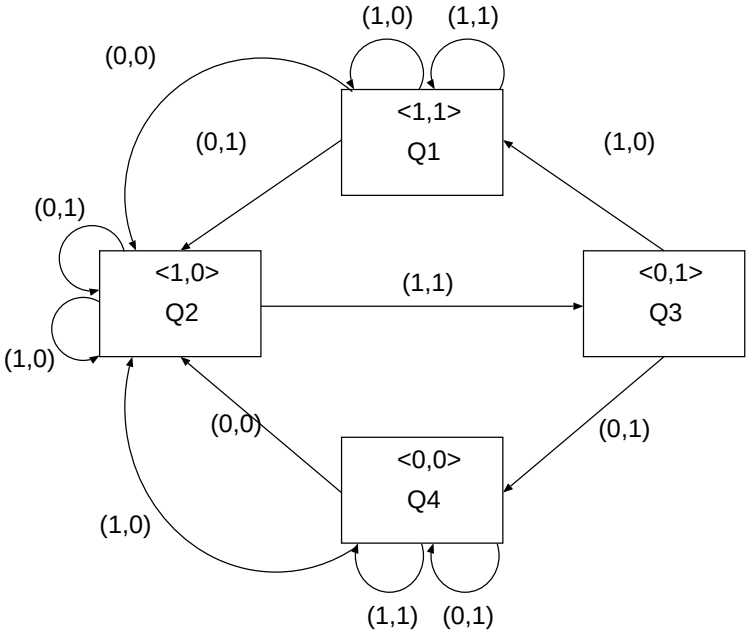

図 2.4　C 素子状態遷移図

Q2 → Q3 の順で遷移する事がわかる。また、不測の事態が生じても、Q4 → Q2 と正常な状態に戻ることができるのを見て分かるように、信頼性の高いデータ転送制御回路になっていることが分かる。

2.4 DDP 設計方針

この節では、LSI の実現法として FPGA とフルカスタムを併用し、DDP の開発をどのように効率的に行っていくかを述べる。

2.4.1 カスタム LSI 設計フロー

現行のフルカスタム LSI の設計の処理フロー [2] は

1. 仕様設計
2. HDL 記述設計
3. 論理設計
4. 回路設計
5. レイアウト設計

と言った順になっている。動作仕様に従って論理設計を行い、回路図、HDL(Hardware Description Language)[3] 記述からの論理合成などを用いてトランジスタや論理ゲートなどの回路素子間の接続関係を表すネットリストを作成する。HDL とはディジタル回路の動作や構造を形式的に記述するための言語である。次にこのネットリストをもとにレイアウト設計を行う。フルカスタム LSI の設計では、このレイアウト設計に莫大なコストがかかっている。

- LSI メーカーで製造しなければならず、デバイスが手元に届くまでの TAT（Turn Around Time）が長い
- 開発の固定コストが高い
- 単品種を大量に発注しなければならない

開発期間の短縮やユーザからの要求が多様化してきている現在、これでは効率的な DDP の開発を行うことはできない。そこで、本研究では DDP の開発環境を構築する手段として FPGA に注目した。FPGA は小規模な論理セルを単位として、それらの間を配線してでき

た構造の高集積 PLD(programmable logic device) である。FPGA は、ユーザーが設計し

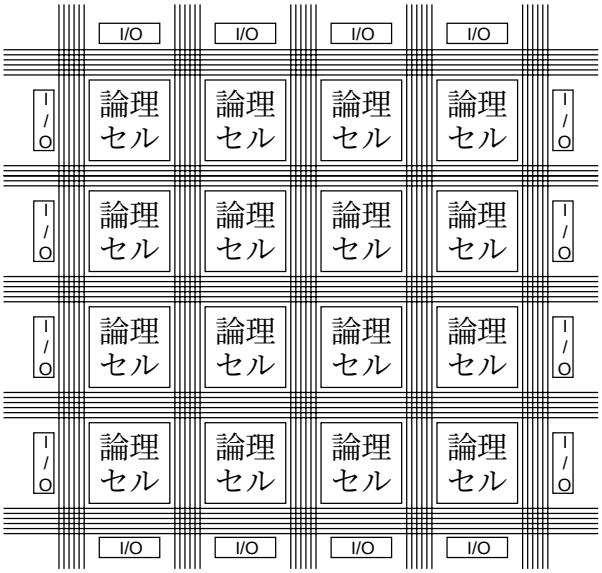

図 2.5 FPGA 構成

た任意のロジック回路を 1 個の LSI 上に実現できるデバイスであり、フルカスタム LSI のようにメーカーに発注して製造するのではなく、ユーザープログラマブル（ユーザー自身が手元で LSI に機能を書き込める）という大きな特徴を持っている。そのため、FPGA は，開発期間の短縮や固定コストの削減が要求されるシステム開発において，とても重要な役割を果たすデバイスとして注目を集めている。開発期間の短縮が急務となっている DDP にとって問題を解決する有力な解決策であることが分かる。さらに，メモリや PCI，LVDS(Low Voltage Differential Signaling) などの高速 I/O もチップ上に搭載できることから，プログラマブルデバイス上でシステムを構築することも可能である。よって FPGA を用いて、設計した回路を LSI に実装して機能検証を行う。その他の LSI の実現方式についての比較を表 2.2 に示す。そして、FPGA での RTL における高精度な機能検証が完了した後、プロトタイプのチップを LSI メーカーに発注して DDP の開発を行う。

表 2.2 LSI 実現方式別の評価

評価項目	FPGA	ゲートアレイ	スタンダードセル	フルカスタム
設計の自由度	△	◯	◎	◎
回路の規模	小 ～ 中	中 ～ 大	中 ～ 大	大
性能	△	◯	◎	◎
機能	△	◯	◎	◎
開発時間	短い	中位	やや長い	長い
開発費	小	中位	大	大
量産時の価格	×	△	◯	◎

2.4.2 FPGA による回路設計

ここでは、FPGA を用いてどのように DDP を設計するかを詳しく述べる。HDL で回路を記述し、デザインツール (Renoir)、シミュレータツール (ModelSim)、論理合成ツール (Leonardo Spectrum)、配置配線ツール (Quartus) を使用してＦＰＧＡボード上に回路をダウンロードする。図 2.6 に FPGA のディジタル回路設計フローを示す。従来、ディジタル回路の設計作業の多くは、人間の手作業によって行われてきたが、半導体の集積化技術の飛躍的な向上に伴い、設計する回路の規模は増大の一途を辿っており、人間の手作業による回路図設計はすでに限界となっている。このため近年、ＨＤＬを用いた回路設計が行われている。本研究ではＨＤＬを用いてトップダウン設計手法を用いＤＤＰを設計する。ＨＤＬを用いたトップダウン設計の利点は以下の通りである。

- ゲートレベル設計の自動化
  - ＨＤＬを用いてＲＴＬ記述を行えば、論理合成が自動化される。その結果、ＲＴＬ設計とゲートレベル設計を完全に分離できることから、設計書はＲＴＬ設計に専念

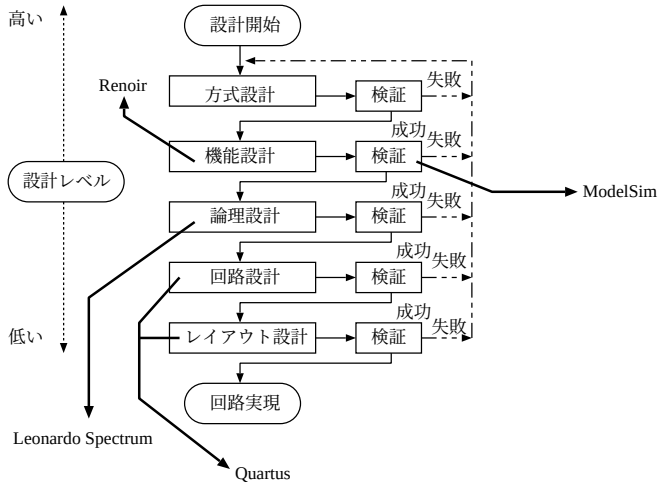

図 2.6　FPGA を用いたディジタル回路設計フロー

できるようになり設計効率が向上する。しかもゲートレベルへの展開が自動化されたことは人手によるケアレスミスの発生を減らし、設計品質の向上をもたらす。

- シュミュレーションの早期開始が可能
  - ボトムアップ的なゲートレベル設計では、シュミュレーションはゲートレベルに展開してから行っていたため、シュミュレーションでバグが見つかっても、それが上位レベルでのバグであると、バグ発生の個所の特定と修正に時間がかかってしまう。一方、ＨＤＬでＲＴＬ設計を行えば、ＲＴＬの段階でシュミュレーションを行えるので、設計の早い段階で高い品質を確保できる。

以上の方針に従い機能検証・性能評価の観点から、DDP の基本構成要素の中でもいつくかの独立した処理をモジュール化し、基本構成要素を１つの大きなモジュールとして設計を行う。これによって、設計資産の再利用効率を高め、設計期間の短縮が望める。

2.5　結言

本章で述べたように、自己タイミング型パイプライン機構は並列度が高く、長いパイプライン段を有効に活用できる画像や通信などの信号処理に向いているシステムである。また、

データ転送制御回路においては、C 素子間の信号線に処理回路の遅延と、整合が取れる遅延が挿入されることを記した。これは、自己同期でデータ転送を行う上で重要な要素になる。さらに、DDP の効率的な開発の手法について、FPGA とフルカスタム LSI を併用して開発を行う方法を示した。

第 3 章

FPGA チップ上での自己タイミング型パイプラインの実現法

3.1 緒言

前章では、自己タイミング型パイプライン機構のデータ転送制御は、データラッチに接続された C 素子によって行われている。そして、データ転送には遅延が重要になることを示した。FPGA は、同期回路に特化して設計されており、遅延回路を用いることができない。そのことにより、自己タイミング型パイプライン機構を従来の構成では実装できない。さらに、DDP を構成する上で重要なパケット複製制御回路と消去制御回路も、遅延回路を用いることができないために、従来の構成をそのまま FPGA チップ上に実装できない。従って本章では、FPGA チップ上に実装可能な自己タイミング型パイプライン機構と新しくパケット複製制御回路と消去回路を提案する。

3.2 クロック入力データ転送制御回路

自己タイミング型パイプライン機構において遅延回路を利用できないため、ハンドシェーク信号が DL 間の処理回路よりも早く、次段の C 素子に到達してしまう。そのことは、処理回路の遅延時間との同期がとれないことを意味し、正常なデータ転送が行われない。

この問題に対して、本節では C 素子のゲートポートに、同期回路の制御に使用されているクロックを入力する方法を提案する。データ転送を正常に行うには、入力パケットが処理回路を経て状態が確定するまで、次段の C 素子から DL を開く信号を発信させなければ良い。つまり、処理回路の遅延時間より、長い時間、次段の C 素子をノンアクティブにすればよい。2 章で C 素子には、C 素子の動作をアクティブ、ノンアクティブにするための入力ポートが存在することを示した。よって、クロックをゲートポートに入力することとする。また、隣接する C 素子には位相が逆のクロックを入力することによって、前後する C 素子を交互に動作させることが可能になり、上で述べた処理回路の遅延時間と整合が取れる遅延を C 素子間に挿入することが可能になる。ゲートにクロックを入力する C 素子の回路図を図 3.1 に示す。実際には、ゲートにクロックを入力するのではなく、ゲートの信号線が存在

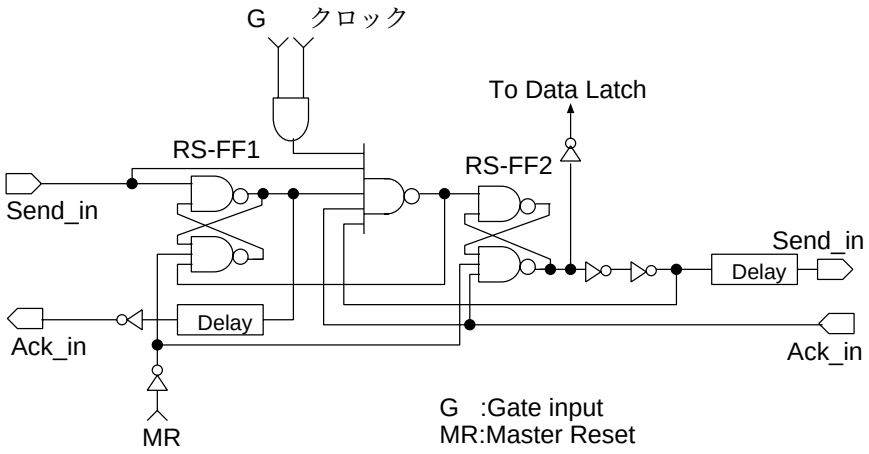

図 3.1　C 素子回路図（クロック入力）

していたポートにゲートとクロックの AND をとった信号を入力している。ゲートの入力は通常 “1” となる。ここでクロックが立ち下がると、つまり “0” が入力される場合、パケットが送信されたことを意味をするパルスが Send_in から入力されていても、クロックが “0” のため、5 入力 NAND の出力は “1” ならない。よって、DL 開閉信号は “1” となり得ず、C 素子が不活性の状態になる。次に、クロックが立ち上がると 5 入力 NAND の出力が “0” になり、DL 開閉信号は “1” となってパケットが DL に格納され、C 素子が活性の状態になっているのが分かる。次にクロックを使用した自己タイミング型パイプライン構成を示す。ク

ロックの立ち上がりで 1 段目と 3 段目の C 素子が活性化され、パケットが処理回路に投入される。次に、クロックの立下りで 2 段目の C 素子が活性化され、クロックの立ち上がりで

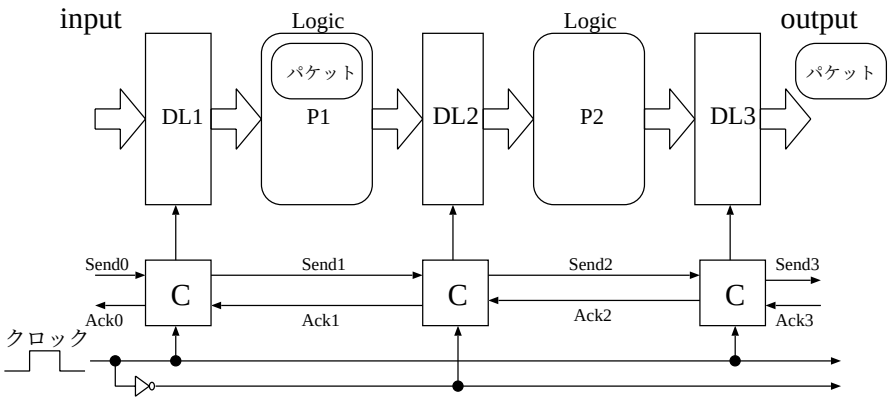

図 3.2　自己タイミング型パイプライン構成（クロック入力）

投入されたパケットが DL に格納され処理回路に投入される。上記の方法を用いることでク

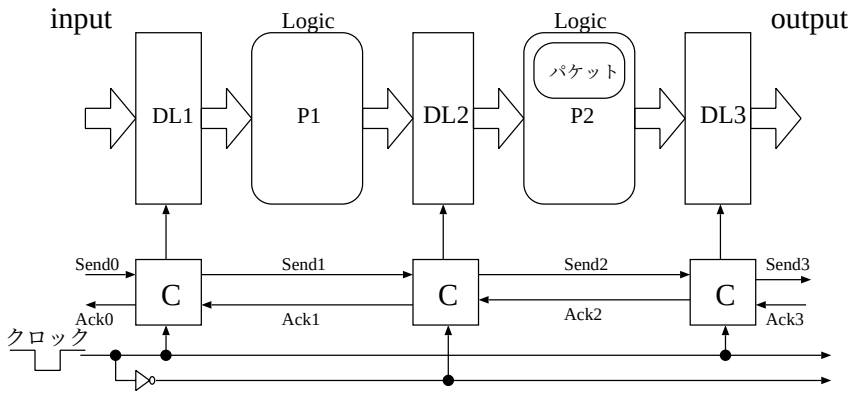

図 3.3　自己タイミング型パイプライン構成（クロック入力）

ロック周波数の 1/2 の時間分だけ遅延を与えることが可能になり、自己タイミング型パイプライン機構を擬似的に FPGA チップ上に実装可能である。ここで擬似的というのは、厳密に自己タイミング型パイプライン機構を再現できないためである。クロックを入力して遅延を挿入する方法では、Send 信号と Ack 信号の遅延の割合を等しくすることしかできない。実際の LSI に実装されている自己タイミング型パイプライン機構では、Send と Ack の遅延は 4 対 1 の割合で挿入されている。パイプラインのパケット充足率が高くなると、パイプラ

インの所々でパケットの転送が停止する部分が出てくる。そして、一度停止したパケットが再度動き出すときに、遅延の割合が 4 対 1 と 1 対 1 では、4 対 1 の割合のほうが当然パケットの動きの回復速度が速い。よって、この動作を正確にシミュレーションすることはできないが、1 つ 1 つのパイプライン段間の処理回路を通るパケットのシミュレーションができるため、本研究の目的に沿うものとして実装する。

3.3　パケット複製制御回路

4 章で説明するが、DDP の基本構成要素の CPS にはパケット複製制御回路が含まれており、DDP を構成する上で必要不可欠な回路であることが分かる。このパケット複製制御回路も遅延を用いて実現されており [4]、FPGA チップ上に実装するために新しいパケット複製制御回路を提案する。最初に遅延回路を用いたパケット複製回路を示す（図 3.4)。図 3.4

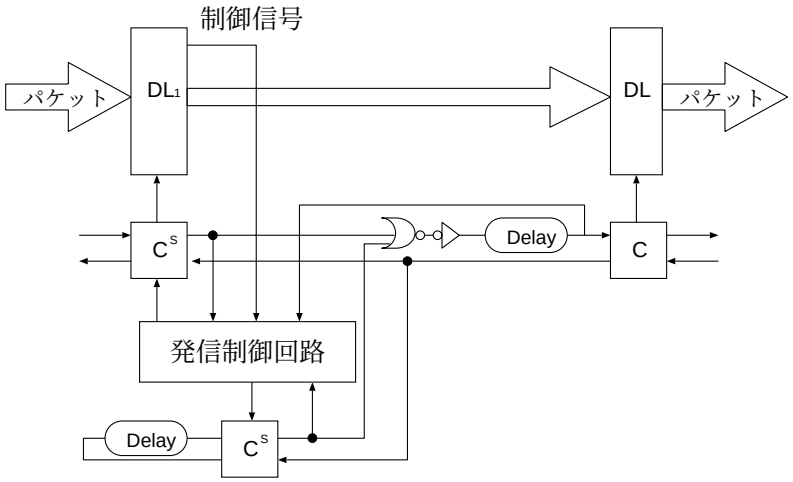

図 3.4　遅延回路を用いたパケット複製制御回路

に示すように、パケット複製制御回路はストップ入力付きの 2 つの C 素子、1 つの基本 C 素子、及び C 素子へのストップ信号を生成する制御回路から構成されている。制御回路は、制御信号の値に応じて 2 つの C 素子 C_s へのストップ信号の値を切り替え、何れか一方のみをアクティブにする回路となっている。下段の C_s へのストップ入力がノンアクティブである限り、この C 素子は自己発信を続け、パケットを送出し続ける仕組みとなっている。上段

の C 素子間の遅延は、3．2 節で述べた方法で置換できるが、下段の C 素子の遅延は 3．2 節で述べた方法では置換できない。この問題を解決する方法として 2 つの方法を考案した。

- 遅延をフリップフロップに置換する
- 下段の C 素子の変わりにパルスジェネレータを使用する

1 番目に挙げた解決法は、殆ど構成を変えずに実現できる。2 番目の解決法は、C 素子はパルスジェネレータを新しく備え付けなければならないが、C 素子と比べてゲート数を少なくできる利点がある。パルスジェネレータが C 素子に置換できる理由は次の通りである。2 章の C 素子の状態遷移から分かる様に、C 素子にデータが転送されたことを認識させるには、下に凸のパルスを発信する必要がある。言い換えれば C 素子はデータを転送する時にしたのパルスを発信する回路といえる。よって、この場合 C 素子をパルスジェネレータに置き換えることができる。本研究では、2 番目の解決法としてあげた下段の C 素子の変わりにパルスジェネレータを使用する解決法をゲート数を少なくできる理由から採用する。以下に提案するパケット複製制御回路図を示す（図 3.5)。提案するパケット複製制御回路は、Send パ

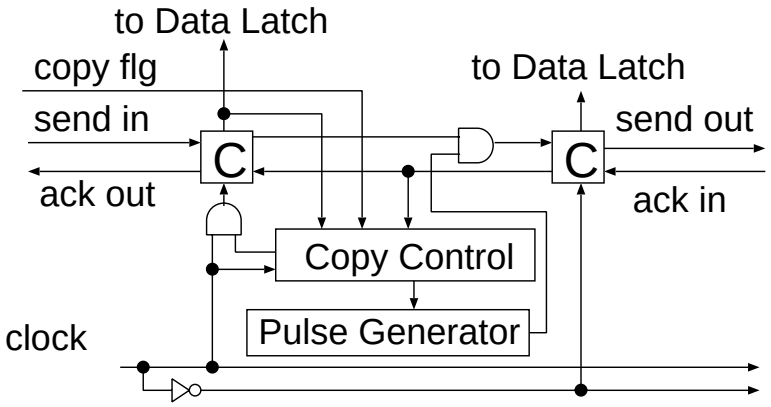

図 3.5　遅延回路を用いないパケット複製制御回路

ルスを生成する Pulse Generator と Pulse Generator に複製のタイミングを計ってパルスの生成許可信号を発信する、順序回路の Copy Control から構成される。以下に回路動作の説明を示す。前段の DL にはパケットが格納さてないとする。Copy Control は予め、Copy flg により複製するべきパケットが後段の DL に格納されることを知り、後段の C 素子にク

ロックが入力されると、後段の C 素子から DL を開く信号が発信される。DL を開く信号を Copy Control が受信すると、後段の C 素子のゲートポートに停止信号を送信し、後段に新しいパケットが格納されるのを防ぐ。この状態を保持して、先ほど転送されたパケットが次段の DL から転送されるのを待つ。次段のデータラッチからパケットが転送されてから、後段の C 素子を活性化させるクロックが入力されると、Copy Control は Pulse Generator にパルス生成許可信号を発信し、Pulse Generator は Send パルスを次段の C 素子に発信する。これにより、次段の DL には先ほど転送されたパケットと同じパケットが格納され、複製が行われたことになる。次に示すのは、Copy Control の状態遷移図である（図 3.6）。この状態遷移図を見れば Copy Control の具体的な信号遷移が理解できるであろう。また、

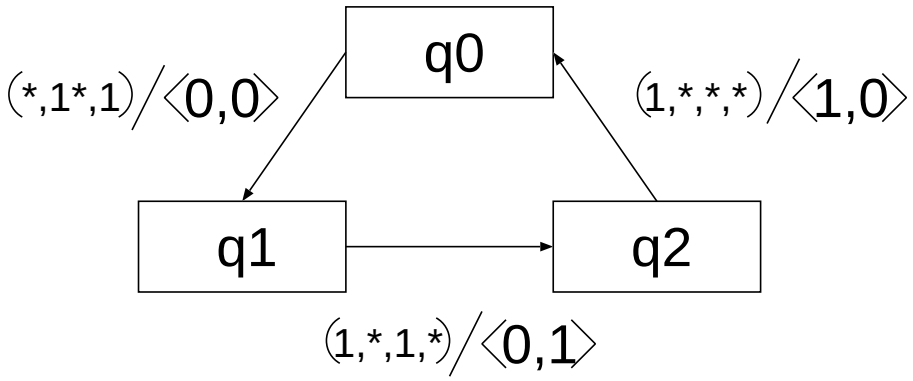

図 3.6　Copy Control の状態遷移図

Pulse Generator の回路構成は以下の通りである (図 3.7)。入力信号線に “0” が入力され

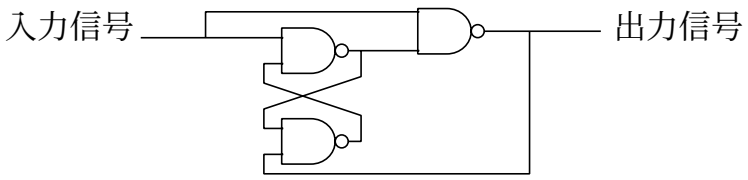

図 3.7　Pulse Generator 回路図

ると出力信号線からは “1” が出力され、“1” が入力されると出力は “0”→“1” と変化しパルスが生成される。

3.4　パケット消去回路

４章で説明するが、基本構成要素 MM と CPS にパケット消去制御回路が組み込まれていることから、DDP を構成する上でパケット消去制御回路は重要な回路と言える。この回路もパケット複製制御回路と同じ問題を有しており、遅延回路を用いない新しいパケット消去制御回路を提案する。LSI に実装されていた遅延を用いたパケット消去制御回路 [4] を下に示す（図 3.8)。遅延を用いたパケット消去制御回路は、消去信号がアクティブな時には、次

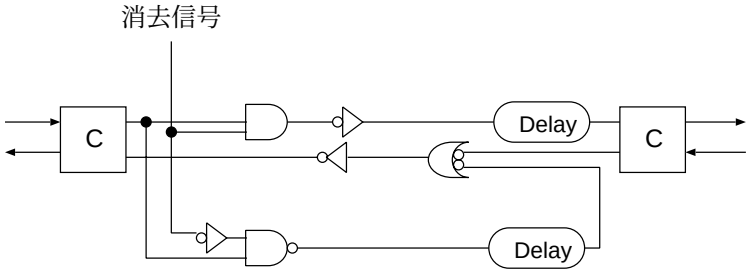

図 3.8　遅延を用いたパケット消去制御回路図

段への Send 信号の伝達を無効化するとともに、ハンドシェイクのプロトコルを完了させるための仮の Ack 信号を後段の C 素子に与えることによって入力パケットの消去を行う。つまり、後段の C 素子には転送パケットが前段の C 素子に届いたことを伝え、次段の C 素子にはパケットが転送されて来ていないように Send 信号を無効にすることによって、新しいパケットが転送されたパケットを上書きする形でパケットの消去を行っている。この回路で FPGA チップ上に実装するときに問題となるのは Ack の遅延である。しかし、ここでは遅延を挿入することを考察するのではなく、Ack 信号の動作に着目した。パケット消去制御回路で Ack が行っている動作は以下の通りである。

1. 消去信号を受信し、後段の DL を閉める Ack 信号を発信する
2. DL を閉める Ack 信号を発信したら Ack の許可信号を発信する。

つまり、後段の C 素子に下に凸のパルスを発信していることになる。よって、これらの機能を満たす遅延を用いないパケット消去制御回路（図 3.9）を提案する。この遅延を用いな

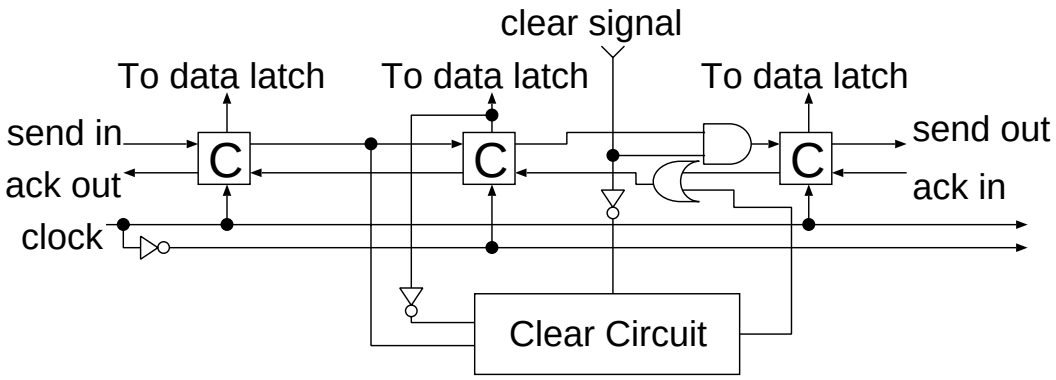

図 3.9　遅延を用いないパケット消去制御回路図

いパケット消去制御回路は、Send パルスを無効にするのと Ack パルスを生成するのに消去信号を用いるところまでは、遅延を用いたパケット消去制御回路と同じである。相違点は Ack パルスを生成するのに消去信号だけを用いるのではなく、2 番目の C 素子に入力される Send 信号と 2 段目の DL に入力される DL 開閉信号を用いるところにある。次に Clear Circuit の回路図を示し、回路全体の動作を説明する。消去すべきパケットが処理回路に投

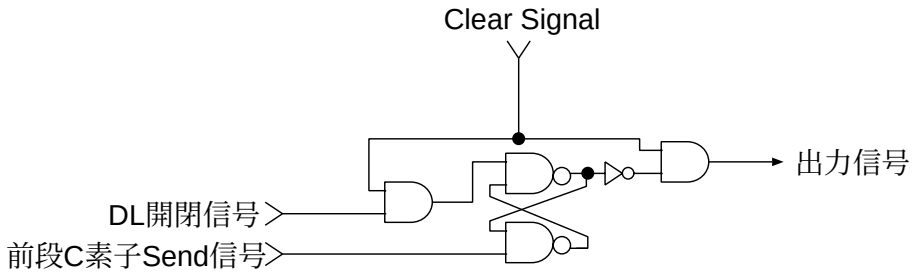

図 3.10　パケット消去回路図

入されると、DL 開閉信号と消去信号が消去回路に入力される。DL 開閉信号の方が、消去信号よりも最初の AND ゲートに到達するので、RS-FF の R には “0” 入力され RS-FF の出力は “1” が出力される。よって、Ack 信号を決定する OR ゲートには “0” が入力され、2 段目の C 素子に Ack の禁止信号が入力され DL が閉じられる。DL が閉じられたので消去回路に入力される DL 開閉信号の値は “0” となるが、RS − FF のに入力されるのは “1” が入力されるので現状維持となり、RS-FF の出力は変化しなため、消去回路の出力も変化し

ない。2 段目の C 素子に転送要求信号が転送されると、消去回路の RS-FF の S に “0” が入力され RS-FF の出力は “0” が出力される。よって、消去回路からの出力は “1” となり、2 段目の C 素子に Ack の許可信号が入力される。これによって、Ack パルスが生成され、3 段目の DL が新しいパケットにより上書きできる状態になった。最後に、新しいパケットが 2 段目の DL に格納されると消去が成される。以下に消去回路の状態遷移図を示す（図 3.11）。

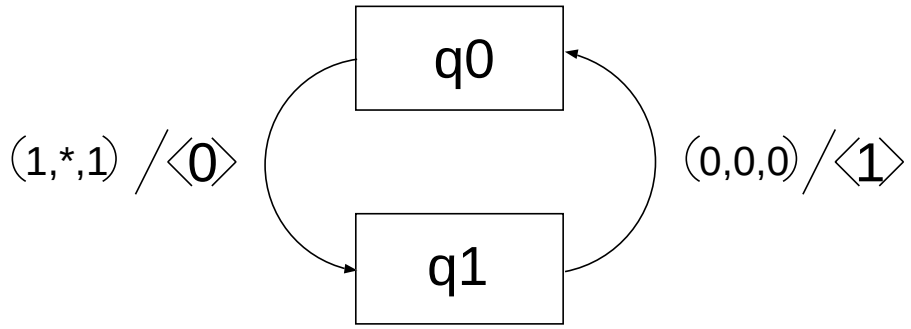

図 3.11　パケット消去回路状態遷移図

3.5　結言

本章では、遅延回路を用いないで自己タイミング型パイプライン機構を実装する方法と、DDP を構成する上で重要となるパケット複製制御回路とパケット消去回路を提案し実装した。これによって、FPGA の遅延回路を利用できない問題点を解決し、FPGA チップ上に実装できる見通しを立てた。

第 4 章

DDP の FPGA 実装と機能検証

4.1 緒言

本章では、3 章で提案した自己タイミング型パイプライン機構とパケット複製制御回路とパケット消去制御回路を用いた DDP の詳細を述べる。まず、主な仕様について述べ、それに従った DDP の各構成要素について説明する。そして、最後に実装した DDP の機能検証を行い、フローグラフを実行する上で最低限必要な処理が正常に行われることを明らかにする。

4.2 データ駆動プロセッサ使用

本節では FPGA に実装する DDP の仕様について述べる。仕様を決定する際に、既に実用化されている DDMP4G を参考とした。

4.2.1 主な仕様

- OCP(operation Code Processor) 部のみを作成することする
- プログラムではメモリを使用しないこととする
- 数値は 2 の補数で表現することとする
- 演算した結果のオーバフロー、アンダフローを考慮しない
- シフトは論理シフトとする
- 小数の表現は固定小数点を採用する

また、DDP は 5 つの基本構成要素をもつ (図 4.1)。

- M(Flow Mergeging Module)[5]
  - ナノプロセッサの外から流れてくるパケットと基本構成要素Dから流れてくるパケットから 1 つのパケットを選択しMMにパケットを流す。
- MM(Matching Memory)
  - CST(matching with constant) と FC(matching with variables) から構成されており、演算のための対になるデータの組み合わせを生成する。
- FP(Functional Processing Unit)
  - MMで発火したデータを演算する。
- CPS(Cache Program Storage)
  - プログラムが記憶されている基本構成要素で、パケットが入ってくるとヘッダの行き先をメモリアドレスとしてメモリを参照し、参照データを新たなヘッダとしてパケットに付加してDにパケットを送信する。
- D(Flow Diverting Module)[5]
  - パケットの行き先情報を参照して、まだ内部で処理が必要であったならMにパケットを流す。そうでない場合、ナノプロセッサの外にパケットを流す。

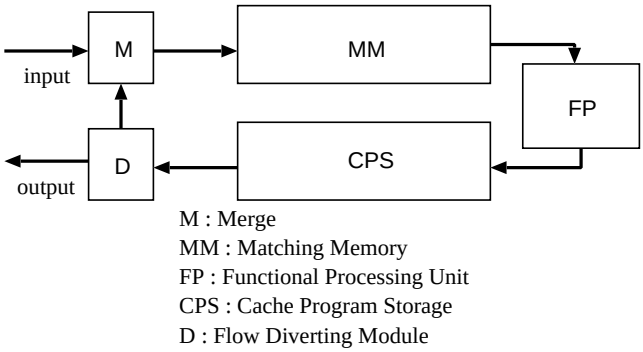

図 4.1　DDP 基本構成要素

4.2.2　DDP 内部パケットフォーマット

ここでは、DDP 内部のパケットフォーマットを示す。特に示さなかった経路のパケットは、基本フォーマットを使用しているものとする。また、それぞれの基本構成要素の中のパケットのフォーマットについては後で述べる。

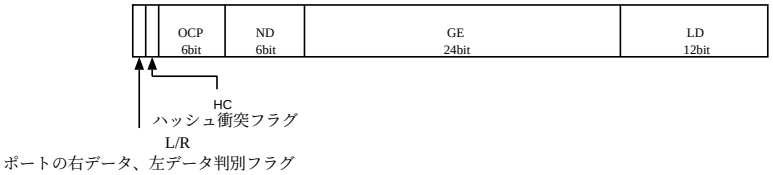

図 4.2　DDP 内部基本フォーマット

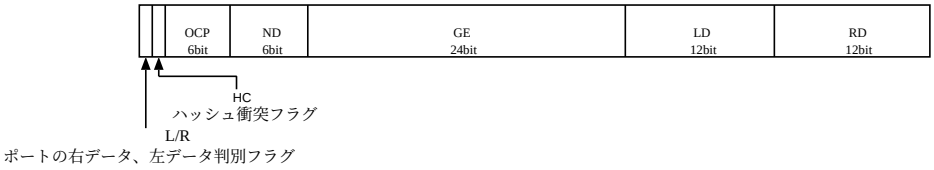

図 4.3　MM から FP へのパケットフォーマット

4.2.3　実装命令

DDP に実装するオペレーションコードと命令の対応表を表 4.1 に示す。今回は、基本的にフローグラフの機能検証を行うための命令しか実装しないこととする。今後、必要と思われる命令は逐次実装する予定である。また、オペレーションコードを圧縮するためにも、オペレーションコードに対する命令の割り当ての最適化も今後の課題である。

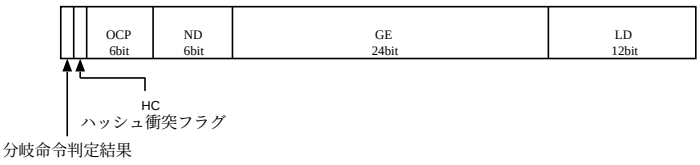

図 4.4　FP から CPS へのパケットフォーマット

表 4.1: オペレーションコードの定義

Operation Code	Operation	命令説明
08	EXOR	排他的論理環（EXOR）の演算をする
09	OR	論理和（OR）の演算をする
0A	NAND	否定論理積（NAND）の演算をする
0B	NOR	否定論理和（NOR）の演算をする
0C	MUL	乗算をする
0D	SYNC	右入力をトリガとして左入力値を出力
0E	NOP	入力値をそのまま出力
10	ADD	加算（左入力＋右入力）をする
11	ABS	加算（左入力＋右入力）した結果を 絶対値表現する
14	SUB	減算（左入力－右入力）をする
15	DIF	減算（左入力－右入力）した結果を 絶対値表現する
18	AGN	世代番号の下位１２ bit に 左入力データを格納
19	SGN	世代番号の下位１２ bit に 右入力データを格納
20	AND	論理和（AND）の演算をする
28	EXNOR	ENOR（排他的否定論理和）の演算をする
29	EQ	左入力=右入力のとき１ そうでないとき０を出力

表 4.1: オペレーションコードの定義

2A	SWEQ	左入力＝右入力のとき True より左入力を出力、 そうでないとき false より 左入力を出力
30	GT	左入力が右入力より大きいとき１ そうでないとき０を出力
31	SWGT	左入力が右入力より大きいとき True より左入力を出力、 そうでないとき false より 左入力をを出力
34	GE	左入力が右入力以上のとき１ そうでないとき０を出力
35	MAX	左入力と右入力を比べて大きい値を出力 両方の値が等しいときは左データを出力
36	MIN	左入力と右入力を比べて小さい値を出力 両方の値が等しいときは左データを出力
37	SWGE	左入力が右入力以上のとき True より左入力を出力 そうでないとき false より左入力を出力
38	LSH	左入力のデータを右入力 （右入力＞０であったら左シフト右入力＜０であったら右シフト） だけシフト

4.3　データ駆動プロセッサ実装

4.3.1　合流部構成

合流部 M(Flow Mergeging Module) はプロセッサ外部から入力されたパケットと、基本構成要素 D の Branch から入力されるパケットを選択出力する機能を持つ基本構成要素である。仮にプロセッサ外部からの入力を InputPathA、Branch からの入力を InputPathB、MM への出力を OutPutPathX とすると、Merge の構成は図 4.5 のようになる。また、信号線の定義を表 4.2 に示す。DL に入力する開閉信号を、相手の C 素子に入力されるゲート信号を生成する NOR ゲートに入力して、一方の C 素子が駆動しているときに片方の C 素子が動作しないようになっている。さらに、DL の開閉信号の機能はこれだけではなく、OutPutPathX に入力するパケットを選択する信号を出力する NOR フリップフロップの入力信号や、調停回路内部の一部の FF を初期化する役目も担っている。 次に調停回路に

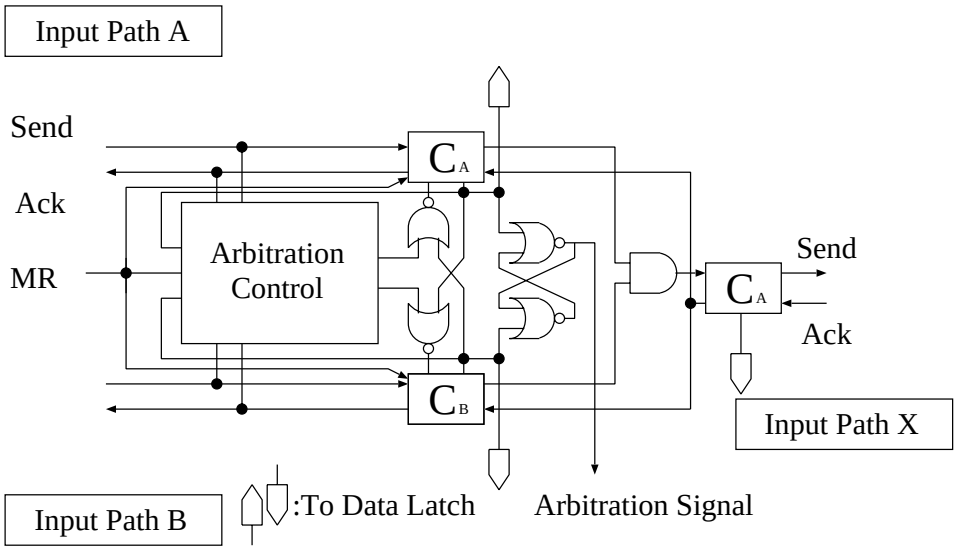

図 4.5　Merge 構成図

ついて詳しく述べる。調停回路は InputPathA と InputPathB の Send、Ack、DL 開閉信号によって状態を遷移する順序回路である (図 4.6)。回路が駆動すると、まず MR が入力され調停回路が初期化される。初期状態では調停回路から InputPathA 側の C 素子を活性

表 4.2　Merge における信号線定義

	0	1
MR	非初期化	初期化
Arbitration Control	許可	禁止
Arbitration Signal	A からの入力を選択	B からの入力を選択

化する信号が出力される。Send 信号を先に入力した Path から選択されるようになっており、例えば A⇒B の順に入力されたら InputPathA 側の C 素子 ⇒InputPath 側の C 素子の順で活性化される。この調停回路は、InputPathA の Send 信号が入力され調停回路の出力が定まらない間に、InputPathB の Send 信号が入力されても InputPathA 側の C 素子 ⇒InputPath 側の C 素子の順で活性化されるように、Send、Ack 信号のパルスを記憶しており、信頼性の高い構成となっている。

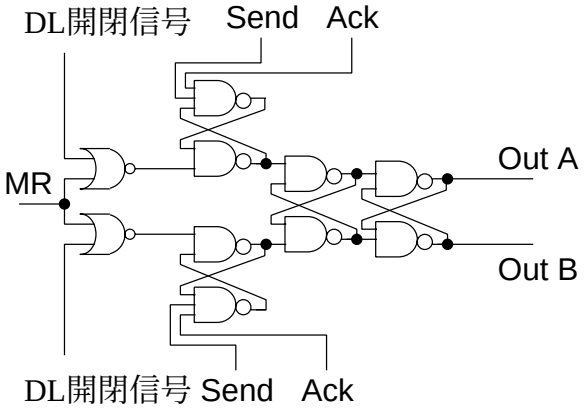

図 4.6　ArbitrationControl 回路図

4.3.2　待ち合わせメモリ部構成

MM(Matching Memory) 部は単項演算用のメモリ (CST) と 2 項演算用のメモリ (Hash Memory) を併用している（図 4.7)。全てのパケットは CST を参照し単項演算であったなら後の処理は未処理で通過する。2 項演算であったなら、入力パケットのタグ情報を、既に

到着済みで未発火の全パケットのタグ情報と比較し、タグ情報の一致するパケットがあれば、発火処理を行い２つのオペランドを有する演算パケットを生成して FP 部に送信する。タグ情報の一致するパケットが無ければ入力パケットは Hash Memory 内に格納され、2 項演算の相手となるパケットの待ち合わせを行う。

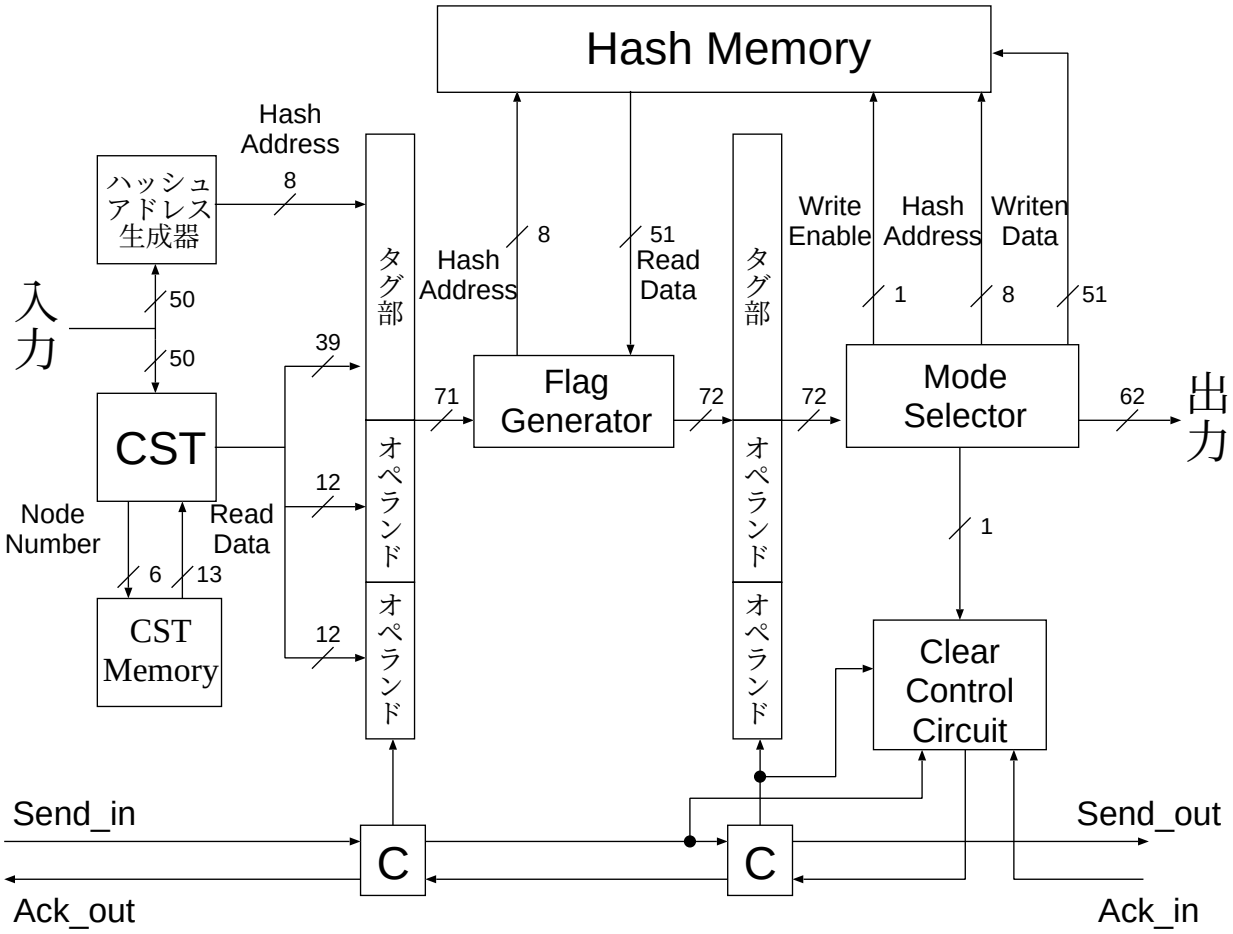

図 4.7　MM 構成図

表 4.3　Mode Selector における信号線定義

	0	1
Write Enable	書き込み禁止	書き込み許可
Clear Signal	消去	通過

各モジュールの機能は以下の通りである。

- Hash Address Generator
  - Operetion Code の下位 2bit と Destination Code、Generation ID のそれぞれ下位 3bit を取り出してハッシュアドレスを生成する。
- CST
  - Destination Code をアドレスとして CST Memory にアクセスする。取り出したデータから単項演算か 2 項演算かを判定し、パケットの最下位に新しいオペランドを追加する。
- Flag Generator
  - ハッシュアドレスでハッシュメモリにアクセスして、読み込んだデータと入力パケットの Destination Code と Generation ID を比較し、
    - 同じならば右ポート、左ポートのオペランドを適切に並び替えて発火したフラグをパケットに立てる。
    - ハッシュ衝突が発生した場合は、ハッシュ衝突のフラグをパケットに立てる。
    - タグ情報の一致するパケットが存在しなかった場合は、未発火のフラグをパケットに立てる。

    上記のどれか 1 つの処理を行う。
- Mode Selector
  - Flag Generator で生成したフラグを元にそれぞれの処理を行う。
    - フラグが発火を示していたら、ハッシュメモリに有効なデータが入っていないことを示すフラグをハッシュメモリに書き込む。
    - ハッシュ衝突フラグが立っていたら、Wirte Enable 信号を Low にする。
    - 未発火のフラグが立っていたら、ハッシュアドレスでメモリにパケットデータを書き込み、パケットを消去する信号を発信する。
- Clear Control Control

– このモジュールについては 3 章のパケット消去制御回路で詳しく述べた。

次に、MM における各モジュールの入出力パケットのフォーマットを示す。

図 4.8 Hash Address Generator, CST input Packet Format

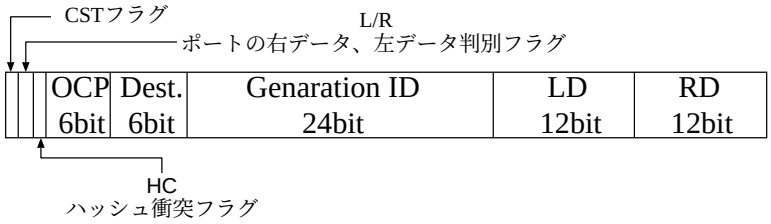

図 4.9 CST output Packet Format

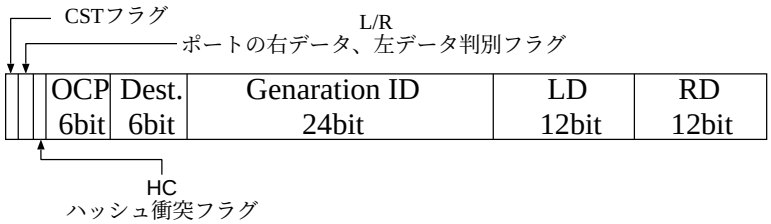

図 4.10 Flag Generator input Packet Format

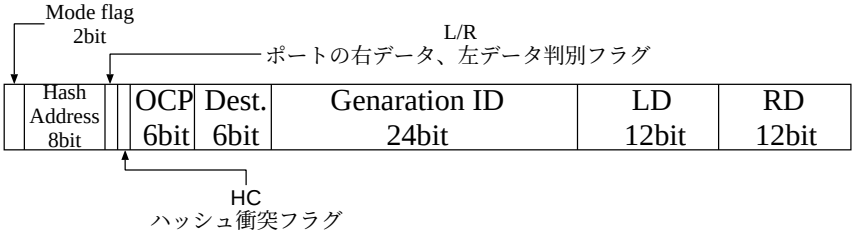

図 4.11 Flag Generator output Packet Format

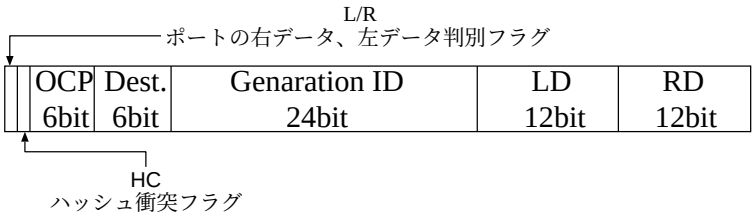

図 4.12 Mode Selector output Packet Format

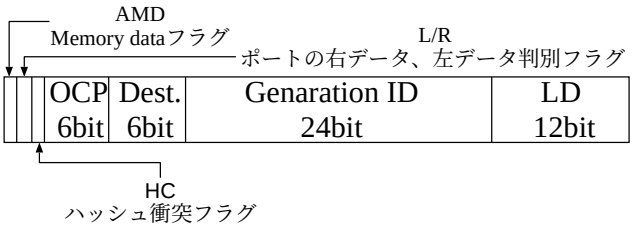

図 4.13 Hash Memory input output Data Packet Format

表 4.4 Mode flag の定義

00	未定義
01	発火
10	未発火
11	ハッシュ衝突

表 4.5　パケットフォーマットにおけるフラグの定義

	0	1
HC	ハッシュ衝突未検出	ハッシュ衝突検出
L/R	右オペランド	左オペランド
CST	2 項演算	単項演算
AMD	無効データ	有効データ

4.3.3　演算部構成

FP(Functional Processing Unit) 部は組み合わせ回路だけで構成されており、入力されたパケットのヘッダに付加されているオペレーションコードをセレクトコードとして、命令演算処理が行われたパケットをを選択する回路である。構成図を図 4.14 に示す。

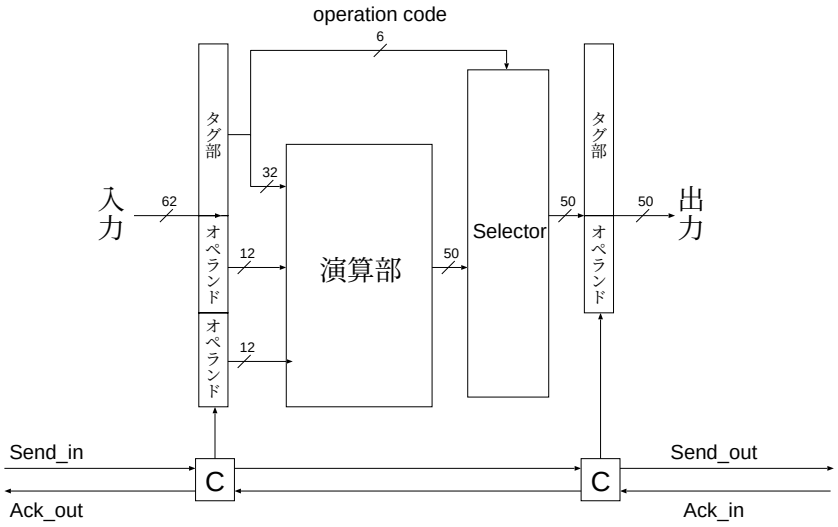

図 4.14　FP 部構成図

4.3.4　プログラム保管部構成

CPS(Cache Program Storage) 内の Cache Program Memory の仕様を示す。分岐命令、コピー、アブソーブを考慮した結果、CPS のメモリを True node と False node に分けて、

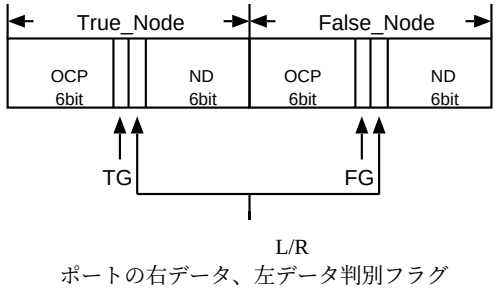

図 4.15　Cache Program Memory Format

それぞれのノードにそのノードが有効か無効かを判定するフラグを設けた。CPS はオペレーションコードを解釈し分岐命令かどうか判別しする、もし分岐命令であったなら、FP 部から送られてくるパケットの分岐命令判定結果のフラグをもとに、True node か False node のどちらか一方を有効データとする。分岐命令でないときは、そのノードが有効であるかないかのフラグ TG,FG を参考に有効データを決める。もし両方に 1 が立っていたのならコピーであり、両方が 0 であったのならアブソーブである。また、Destination Code に全て 1 が入力されているとき、それは DDP から出力することを意味する。

次に、CPS 部の構成を述べる。(図 4.16)。CPS 部は 4 パイプラインステージからなり、

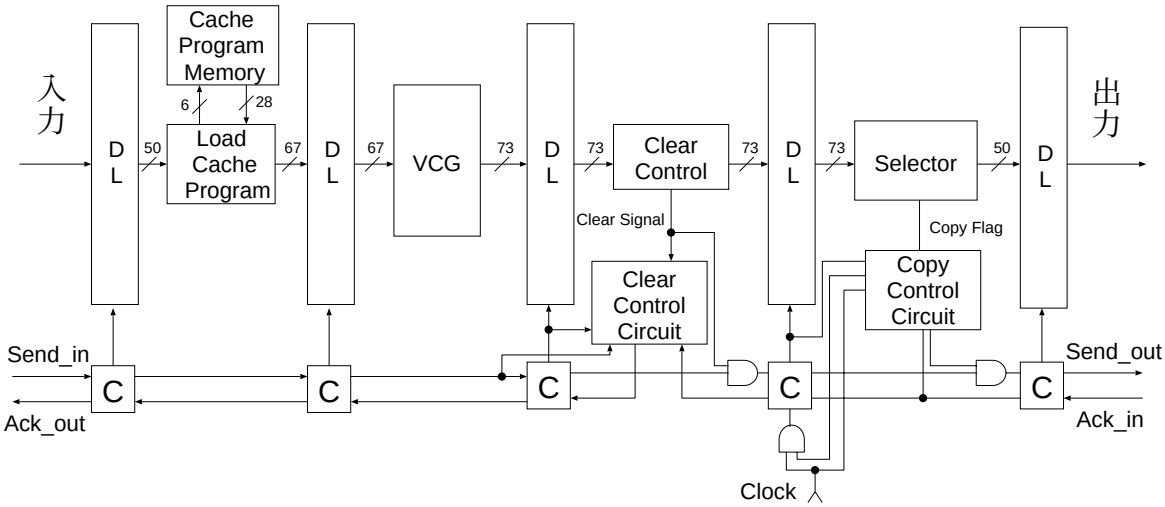

図 4.16　CPS 部構成図

パケットが入力されると、Cache Program Memory から Destination Code でプログラム

を読み込み、読み込んだプログラムに応じて処理コードがパケットに付加され、処理コードに応じて、ハッシュ衝突、分岐、コピー、アブソーブ（消去）の処理が実行される。各モジュールの機能は以下の通りである。

- Cache Program Memory
  - プログラムが格納されているメモリ。
- Load Cache Program
  - 入力されたパケットが分岐命令であるかを調べ、その結果と Cache Program Memory から読み込んだプログラムをパケットに付加する。
- Verdict Code Generator
  - パケットの情報を元に、ハッシュ衝突、分岐、コピー、アブソーブのセレクタコードを生成しパケットに付加する。
- Clear Control
  - VCG で付加されたコードがアブソーブであった場合、Clear Circuit に消去信号を送信する。
- Clear Control Circuit
  - このモジュールについては 3 章のパケット消去制御回路で詳しく述べた。
- Selector
  - ClearControl で処理されなかったハッシュ衝突、分岐、コピーの処理を行う。
- Copy Control Circuit
  - このモジュールについては 3 章のパケット複製制御回路で詳しく述べた。

以下に、CPS における各モジュールの入出力パケットのフォーマットを示す。同じビット幅で繋がれた DL 間のパケットは、同じフォーマットとする。

図 4.17　CPS input Packet

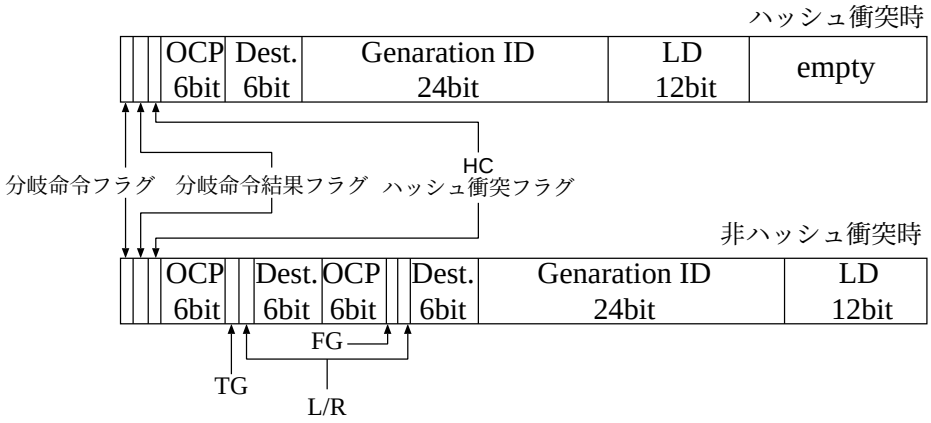

図 4.18　Load Cache Program output Packet Format

また、パケットフォーマットで新しく使用されたフラグの定義を表 4.6 と表 4.7 に示す。なお、VCG Code の表で表記されていないコードについては未定義とする。

表 4.6　分岐命令・分岐命令結果フラグの定義

	0	1
分岐命令フラグ	非分岐命令	分岐命令
分岐命令結果フラグ	false	ture

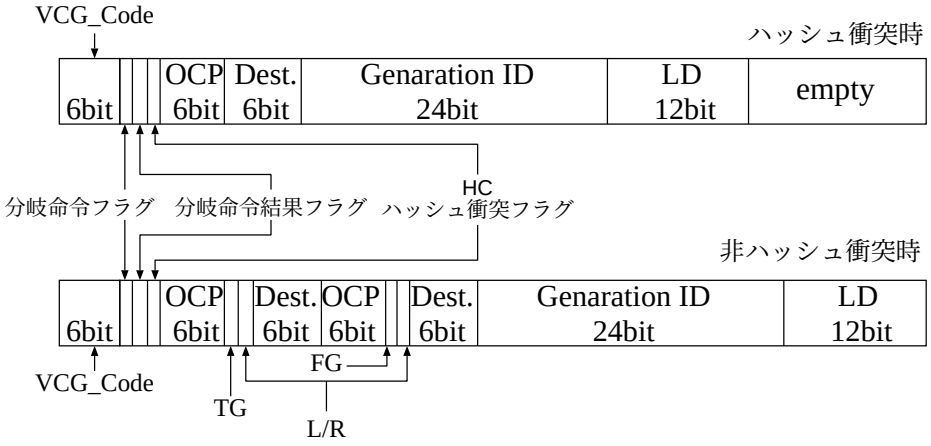

図 4.19　VCG output Packet Format

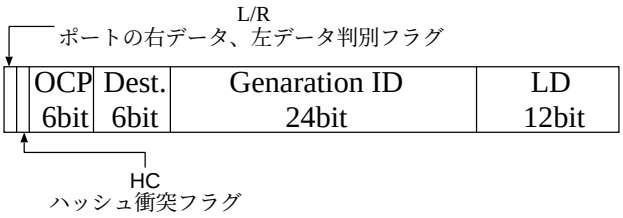

図 4.20　Selector output Packt Format

4.3.5　分岐部構成

分岐部 D(Flow Diverting Module) はプロセッサ内部から入力されたパケットを、基本構成要素 M の Merge に出力するか、DDP の外部へ出力するかを選択する機能を持つ基本構成要素である。仮に Merge への出力を OutputPathX、プロセッサ外部への出力を OutputPathY、分岐部への入力を InputPutPathA とすると、分岐部の構成は図 4.21 のようになる。InputPathA の C 素子から DL を開く信号が発信されると D-FF から Control FLG が OutputPathX,OutputPathY の send 入力につながれている NAND に入力される。NAND に “0” が入力されると、“1” が出力される。つまり、Control FLG が “0” であったときは、そちら側のパスにはパケットが投入されていないこととなり、DL にパケットが格納されない。よって、Control FLG X,Y は逆位相の信号が入力される。このように、分岐部は Control FLG によって C 素子に入力される Send 信号を制御し、データ転送を

表 4.7 VCG Code の定義

100000	ハッシュ衝突処理
010000	複製処理
000100	消去処理
000010	true 処理 (Cache Program Memory から取り出してきたデータの true 部分を使用)
000001	false 処理（Cache Program Memory から取り出してきたデータの false 部分を使用)

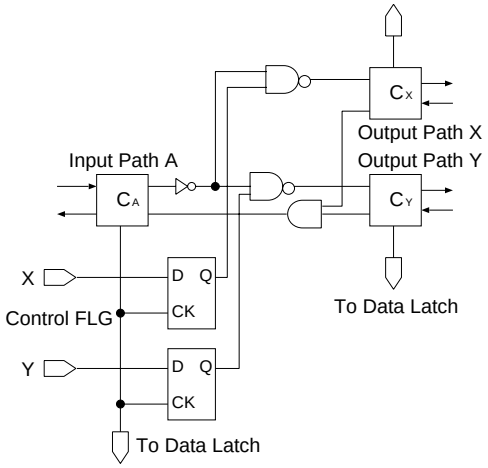

図 4.21 Branch 部構成図

行っている。

4.4 DDP 機能検証

DDP のフローグラフを実行するためには、以下の基本処理が必要になる。

- 複製
- 分岐
- アブソーブ（消去）

この基本処理を正しく実行できれば、フローグラフで記述された処理を正しくシミュレーションできることになる。よって基本処理を全て含むフローグラフを Cache Proguram Memory に書き込み、HDL シミュレータで実行して機能検証を行うこととした。基本処理を含んだフローグラフを以下に示す（図 4.22)。

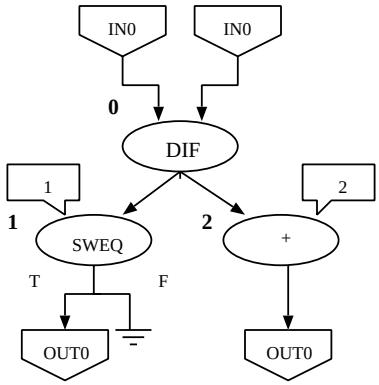

図 4.22　機能検証フローグラフ

このフローグラフに、次のパケットを投入した（表 4.8)。

表 4.8　投入パケット

	OPC	Dest.	Generation ID	operand
左データ	15	0	1	3
右データ	15	0	1	4

次に得られた HDL シミュレータの結果を示す。(表 4.9)。

表 4.9　出力パケット

Generation ID	operand
1	1
1	3

結果を見てみると投入パケットに対して、正しい結果が得られていることが分かる。これに

よって、基本処理を含んだフローグラフを正しく処理できていることが分かり、高精度な回路レベルのシミュレーション結果を得られることが確認された。

4.5 結言

本章では、実際に実装を行った DDP の仕様とそれに沿った DDP の各構成要素の詳細を述べ、その DDP が HDL シミュレータ上で正常に動作していることを示した。このことは DDP の開発環境の基盤を確立できる見通しが立ったことを示す。ただ、MM 部では設計を簡略化するためにハッシュ衝突事前検知回路を実装していないため、データハザードが発生する可能性がある。また、ハッシュメモリにハッシュアドレスを与えて、データを読み込んでくる動作を 1 つのパイプライン段で行うのは処理遅延が大きいため、先に述べたデータハザードを考慮しつつ最適化を行う必要がある。

第 5 章

性能評価

5.1 緒言

本章では、4 章で機能検証した DDP の性能評価を行う。比較対象として、既存の DDMP４ G のアーキテクチャソフトウェアシミュレータを用いた。また、提案した回路を実装した DDP を FPGA チップ上に実装した場合の FPGA チップのハードウェアリソース消費量を解析し、マルチプロセッサの機能検証について考察する。

5.2 性能評価

画像処理向けに特化された DDMP4G のアーキテクチャソフトウェアシミュレータと提案した回路を実装した DDP とでフローグラフの処理時間を計測し、比較することで性能評価を行った。評価には、RGB の画像データを YCrCb に変換するフローグラフと偶数パリティチェックを行うフローグラフを用いた。今回、DDP の処理時間の評価は、DDP を FPGA チップ上に実装できなかったため、測定値ではなく計算式により見積もった。
以下に計算式を示す。

- 最小パケット投入間隔 = 総アーク数 × (2 × 遅延時間)
- レスポンスタイム = CPL の複製以外のアーク数 × DDP のパイプライン段数 + CPL の複製のアーク数 × (DDP のパイプライン段数 + 2) + 遅延時間
- 処理時間 = 最小パケット投入間隔 × (投入パケット数 − 1) + レスポンスタイム

また、最小パケット投入間隔の計算はは DDMP4G シミュレータと同じである。シミュレータを実行してみたところ以下の結果が得られた（表 5.1)。この表を見て分かる通り、既存の

表 5.1　フローグラフ実行結果

	シミュレータ	FPGA
RGP → YCrCb 変換（１０００画素）	3 min	11m sec
偶数パリティチェック (パケット 4096 個投入)	8 min	16m sec

ソフトウェアシミュレータと DDP を実装した FPGA では約 1 万倍 ~3 万倍の性能の差があることが確認された。4 章で示したこととあわせると、高速で高精度なシミュレーションが可能なことが分かる。

次に大規模 1 チップマルチプロセッサの評価への応用を検討するために、論理合成後の 20 万ゲート [6]FPGA チップ上の I/O、Logic Cell、Memory のリソース消費量を調査した。調査結果を以下に示す（表 5.2)。これによると、DDP1 個で LC を 16.85%しか使用し

表 5.2　20 万ゲート FPGA リソース消費量

	Used	Avail	Utilization
I/Os[pieces]	106	376	28.19%
LCs[pieces]	1402	8320	16.85%
Memory Bits[bit]	13056	106496	12.26%

ていないことが分かる。LC の使用率は DDP を搭載する個数に比例して増加するが、I/O は外部との入出力にかかわるプロセッサによって変化する。今回は、外部との通信を行うプロセッサは 1 個として見積もりを算出した。これによって、20 万ゲート FPGA チップ上に約 5 個の DDP を搭載でき、最大 5 個までの DDP を搭載したマルチプロセッサの機能検証が高精度・高速に行えると言える。

5.3 結言

本章では、既存のソフトウェアシミュレータと今回提案した回路を実装した DDP を FPGA チップ上に実装した場合を想定して算出した実行結果を元に比較を行った。比較の結果、高精度で高速にフローグラフの検証が行える事がわかった。また、FPGA のリソース消費量を調べたところ、今回研究で使用した 20 万ゲートの FPGA チップで約 5 個のプロセッサを搭載できることが分かった。最近では、1000 万ゲートの FPGA チップも市場に出回っているため、それらを使用すれば単純計算で 250 個の DDP を搭載することが可能である。よって、これらを利用すれば大規模な DDP システムを 1 つの FPGA チップ上に搭載することができ、開発環境の大幅な改善が見込める。残された課題として、FPGA チップ上に実装した DDP の有効性をさらに明確にするための比較データを収集することがあげられる。

第 6 章

結論

6.1 結論

近年における半導体の微細化技術の発展により、単位面積あたりのゲート数が飛躍的に増えている。それによって、プロセッサ、データパス回路、専用 LSI（ASIC) などの構成要素が、1 つのチップに集約できるようになり、そのことによってシステムが複雑化・大規模化している。一方では市場の要求は多様化してきており、開発期間を短縮することと、要求の変化に追随できる柔軟なシステムを開発することが要求されている。しかし、微細化の影響により大規模化とレイアウト設計時にタイミングを収束させることが難しくなったために、従来のノイマン型処理方式では開発期間が延びる傾向にある。DDP では自己タイミング型パイプライン機構を採用しているため、配線を局所化でき、自己同期で動作するためノイマン型の処理方式よりタイミングの問題を解決しやすい。また、画像や通信など高い並列度が得られ、長いパイプライン段を有効に活用できる特定処理に向いている。この先、この分野での需要はさらに拡大する見込みがあるため、本研究における DDP 開発環境構築が急務となっている。

これを受けて、本研究では、近年、同期回路の開発期間短縮で成果をあげている FPGA に注目し、これを用いて DDP の開発環境構築を試みた。まず、FPGA は同期回路に特化して設計されており、遅延回路を用いないで自己タイミング型パイプライン構成を実装しなければならなかった。これには、C 素子にクロックを入力することで解決した。さらに、DDP を構成する上で必要なパケット複製制御回路、消去制御回路回路も、LSI では遅延回路を用いて実装されていたため、遅延回路を用いない回路を新しく提案した。これらの新し

く提案した方式を DDP に実装し、FPGA チップ上に DDP を実現するための方式を確立した。また、既存のアーキテクチャソフトウェアシミュレータと実行時間を比較し、十分高速に機能検証を行えたことを示した。そして、FPGA のリソース消費量を調べることで、大規模な DDP システムを検証することが出来る見通しを立てた。

今後、提案した回路を実装した DDP を FPGA チップ上に実装し、その有効性を実証するとともに、さらなる開発環境の構築を目指す予定である。具体的には次のようなものを予定している。

1. 各構成要素の最適化を行う
   (a) 今回は DDP を動作させることを第 1 目標として各構成要素を設計したため、各構成要素とも冗長な部分が多い、これを改善する。これによって、DDP のスループット性能が上がり、さらなる高速化が望める。
   (b) CST で使われるメモリの番地に対するデータの割り当てを最適化する。現在、Destination code をメモリのアクセス先にしているため、メモリの使用効率が悪い。
2. 利便性の向上
   (a) 今回、機能検証を行う際に、何度か Cache Program Memory を何度か書き換えたが、手作業でフローグラフをタグデータに変換していたために、かなりのコストが掛かった。これを改善するためにも、フローグラフをタグデータに変換するプログラムを作製する必要がある。
   (b) フローグラフの機能検証だけを行いたいユーザには、PE の再構築などの作業は必要がない。完成された DDP は 1 つにまとめ使いやすいインタフェースを構築する。
3. ライブラリ作製

   基本構成要素の構成を最適化を行った上で、よく使用されるモジュールをライブラリにまとめて、再度利用可能な環境を構築する。これによって、特定処理向きに特化した

PE 構成の検証を行う環境を短期間で構築することが可能になる。

システムが１チップで実現されるようになると、従来ハードウェアで実現されていた処理がソフトウェアでも実現可能となってくる。そのために、ハードウェアで実現するか、ソフトウェアで実現するかが、事前に分からない領域が高度な機能にも拡張されるようになり、ハードウェアとソフトウェアの協調設計が欠かせなくなっている。ハードウェアとソフトウェア双方の性能評価が円滑に進められる開発環境を用いることによって、ハードウェア・ソフトウェア・コデザインが実現される。そして、本研究における目的が達成されると、ソフトウェアシミュレータで最終の実装仕様が決定するまでアルゴリズムの検証を行い、実際に実装仕様が決定してからは、本研究で構築される FPGA チップ上に実装された DDP を使用するといった DDP システムの協調設計が実現される。

謝辞

本研究に於いて懇切なる指導、ご鞭撻を承った岩田 誠教授に心より感謝の意を表します。

本研究を進めるに当たり、親切なご助言、ご指導を賜った大森 洋一助手に深く感謝の意を表します。

本研究室の基礎としているデータ駆動型アーキテクチャを提唱され、様々な御示唆を賜った、寺田 浩詔 教授に心より感謝の意を表します。

本研究に於いて、副査をお受け頂きました、竹田 史章 教授、Ruck Thawonmas 助教授に心より感謝の意を表します。

本研究を進めるにあたり、様々な御助言、御指導を賜った高知工科大学 情報システム工学科の教員、スタッフの方々に心より感謝の意を表します。

日頃から暖かいご支援、並びにご助言を頂いた大学院生の細見 俊彦氏、橋本 正和氏、別役 宣奉氏、森川 大智氏に感謝の意を表します。

日頃からご支援いただいた岩田研究室の方々、三宮 秀次氏、志摩 浩氏、中村 勲二氏、長野 光氏、原田 香織氏、西山 直人氏に感謝の意を表します。

参考文献

[1] H. Terada, S. Miyata and M. Iwata, "DDMP's: Self-timed Super-Pipelined Data-Driven Multimedia Processors ", *Proc. of IEEE*, Vol. 87, No. 2, pp. 282-296, February 1999.

[2] 白井 肇, "わかりやすい システム LSI 入門", Ohmsha, 1999.

[3] Donald E. Thomas, Philip R. Moorby, "The Verilog Hardware Description Language", Kluwer Academic Publishers, 1995.

[4] 小守 伸史, "データ駆動プロセッサの自己同期回路による VLSI 実現に関する研究", データ駆動マイクロプロセッサプロセッサの自己同期回路による VLSI 実現に関する研究報告書, 1995.

[5] H. Kanekura, "Data junction and branching apparatus for an asynchronous data transmission path" , U.S. Patent 5 396 638, March. 1995.

[6] http://www.altera.com/products/devices/apex/apx-overview.html#1.8, APEX20KE Device Data Sheet