論文 / 著書情報
Article / Book Information

題目(和文)	ハードウェアアクセラレータを活用したクラウド基盤技術 “ Video Service Function Chaining ” に関する研究
Title(English)	
著者(和文)	右近祐太
Author(English)	Yuta Ukon
出典(和文)	学位:博士（工学）, 学位授与機関:東京工業大学, 報告番号:甲第11937号, 授与年月日:2021年3月26日, 学位の種別:課程博士, 審査員:髙橋 篤司,一色 剛,本村 真人,中原 啓貴,原 祐子
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第11937号, Conferred date:2021/3/26, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

TOKYO INSTITUTE OF TECHNOLOGY
DEPARTMENT OF INFORMATION AND COMMUNICATIONS ENGINEERING

令和 2 年度 学位論文

ハードウェアアクセラレータを活用したクラウド基盤技術 “Video Service Function Chaining” に関する研究

東京工業大学
工学院 情報通信系 情報通信コース

右 近　祐 太

令和 3 年 3 月

概要

本論文では，各ユーザの目的に合わせてカスタマイズしたクラウド画像処理サービスを柔軟に提供するために，ネットワーク上で画像処理機能を組み合わせる Video Service Function Chaining（VSFC）を提案する．この提案で高品質なサービスを提供するための最も重要な要件の一つは，サーバノードの処理遅延が小さいことである．そこで，2 種類の技術によりサーバノード内の遅延を削減し，VSFC による高品質なクラウド画像処理サービスを実現する．

近年，コンピュータビジョンや人工知能（AI）などの高度なクラウドサービスの需要が高まっている．これらのサービスを高速かつ低消費電力で提供するために，ハードウェアアクセラレータ（HWA）を用いたシステムが検討されている．これらのサービスに対するユーザの要求は様々であり，ユーザの目的に応じてサービスをカスタマイズできることが望ましい．しかし，既存のシステムは不特定多数のユーザに向けて単一のサービスを提供することを目的としており，ユーザの目的に合致したサービスを提供できない．また，クラウド上のプラットフォームを利用してユーザが独自のサービスを構築することもあるが，開発に時間がかかる．

一方，通信事業者は低コストで柔軟なネットワークを実現する Software-defined network（SDN）や Network function virtualization（NFV）などのネットワーク仮想化技術に注目している．これらの技術により汎用サーバを用いてネットワーク機能が提供されるようになった．さらに，ユーザごとにカスタマイズしたネットワークサービスを提供するために，汎用サーバ上のネットワーク機能を柔軟に組み合わせる Service Function Chaining（SFC）が検討されている．

SFC 技術はネットワークサービスを柔軟に提供することを目的に検討されているが，より高度な情報処理サービスに対しても有用な技術である．例えば，SFC 技術を利用した映像監視サービスでは，監視カメラの置かれた環境や検出したい人/モノによって検出アルゴリズムを変えることで，ユーザごとに監視条件が異なった場合も高品質なサービスを提供

できる．しかし，著者の知る限りこのような画像処理サービスは存在しない．

このような背景から，本研究では HWA 搭載サーバを用いて画像処理機能を組み合わせる Video Service Function Chaining（VSFC）を提案する．VSFC ネットワークは SFC 技術を利用して複数のサーバノードに実装された画像処理機能を組み合わせることで，ユーザの目的に合致した画像処理サービスを提供する．しかし，各サーバノードで発生する遅延がサービス遅延として累積するため，VSFC による画像処理サービスは性能が低下する可能性がある．したがって，この提案で高性能な画像処理サービスを提供するための最も重要な要件の一つは，サーバノードの処理遅延が小さいことである．

典型的な CPU-HWA サーバアーキテクチャでは低速な CPU を用いて複数のパケットから画像データを再構築した後，アプリケーションレベルで画像データを HWA に転送するため，大きなデータ転送遅延が発生する．また，HWA に実装される画像処理回路は単一のクロックに同期して動作する固定レイテンシ回路であり，多くのアイドル時間により処理に時間がかかる．このように，従来のサーバノードは大きな処理遅延が発生するため，VSFC による低遅延なサービスに向いていない．

本研究では 2 種類の技術によりサーバノード内のデータ転送遅延と画像処理遅延を削減し，低遅延な VSFC 対応画像処理サービスを実現する．

まず，パケット順序制御回路を備えた CPU-HWA サーバアーキテクチャを提案する．このアーキテクチャは画像データを含むパケットをネットワークレベルで HWA に転送し，ハードウェアでパケット順序制御と画像処理を行う．この構成では画像処理回路が正しいデータを受信するためにパケット順序制御回路が必要である．パケット順序制御を高速に行うために，正しい順序のパケットを適切なタイミングで転送するリアルタイムパケット順序制御回路を提供する．この構成により，サーバノード内のデータ転送遅延を削減する．

次に，Approximate Completion-Detection（ACD）方式の可変レイテンシ回路を提案する．ACD 方式は組み合わせ回路におけるタスクの完了を大まかに検出し，各タスクの処理時間を変更することでアイドル時間の短い回路動作を実現する．この方式の可変レイテンシ回路はタスク完了を誤検出することで誤った処理を行う可能性があるが，エラー率は制御可能であり，低いエラー率で高速に動作する．この技術により，画像処理回路を高速化する．

提案技術の効果を確認するために，いくつかの実験を行った．第一の実験では提案アーキテクチャにおける画像処理機能の処理遅延を評価し，従来構成と比べて高速に画像処理を行えることを示した．第二の実験では ACD 方式の画像処理回路の処理性能を評価し，わずかな計算エラーを許容することで，従来方式の回路と比べて処理性能が向上することを示した．第三の実験では，2 種類の画像処理機能（HOG 特徴量計算，フレーム間差分法

による移動物体検出）からなる映像監視サービスを開発し評価した．実験において，提案アーキテクチャを FPGA 搭載サーバで実現し，2 種類の画像処理機能を組み合わせることで，VGA 映像をリアルタイムに監視できることを確認した．また，画像処理機能に ACD 方式の可変レイテンシ回路を用いることで，サービス品質をほとんど低下させることなく処理性能が向上し，低遅延な映像監視サービスを実現できることを示した．

以上から，提案技術によりサーバノード内のデータ転送遅延と画像処理遅延を削減できることを確認した．また，提案技術を用いたサーバノードで VSFC による低遅延な映像監視サービスを実現できることを確認した．

目次

図目次

表目次

第 1 章

はじめに

1.1 研究の目的

近年，コンピュータビジョンや人工知能（AI）などの高度なクラウドサービスの需要が高まっている．これらのサービスを高速かつ低消費電力で提供するために，Field-Programmable Gate Array（FPGA）や Application-Specific Integrated Circuit（ASIC）などのハードウェアアクセラレータ（HWA）を用いたシステム [1, 2, 3] が検討されている．Microsoft 社の Catapult[4] や Brainwave[5] は，FPGA を用いて検索サービスや推論サービスを高速化した．Google 社は高速なディープラーニングを低消費電力で行うために Tensor Flow Processor（TPU）[6] を開発した．Intel 社は CPU と FPGA を密結合した Xeon+FPGA プラットフォーム [7] による CPU と FPGA 間の高速通信を検討している．

高度なクラウドサービスに対するユーザ要求は様々であり，ユーザの目的に応じてサービスをカスタマイズできることが望ましい．しかし，前述のシステムは不特定多数のユーザに向けて単一のサービスを提供することを目的としており，ユーザの目的に合致したサービスを提供できない．また，Amazon EC2 F1 インスタンス [8] や Google Cloud Platform[9] などのクラウド型プラットフォームを利用してユーザが独自のサービスを構築することもあるが，アプリケーションの開発に時間がかかる．

一方，通信事業者は低コストで柔軟なネットワークを実現する Software-defined network（SDN）[10] や Network function virtualization（NFV）[11] などのネットワーク仮想化技術に注目している．これらの技術により，ネットワーク機能は汎用サーバを用いて提供されるようになった．さらに，ユーザの目的に応じたネットワークサービスを柔軟に提供するために，汎用サーバ上のネットワーク機能を組み合わせる Service function chaining（SFC）[12] が検討されている．図 1.1 は SFC 技術を利用したネットワーク（SFC ネットワーク）

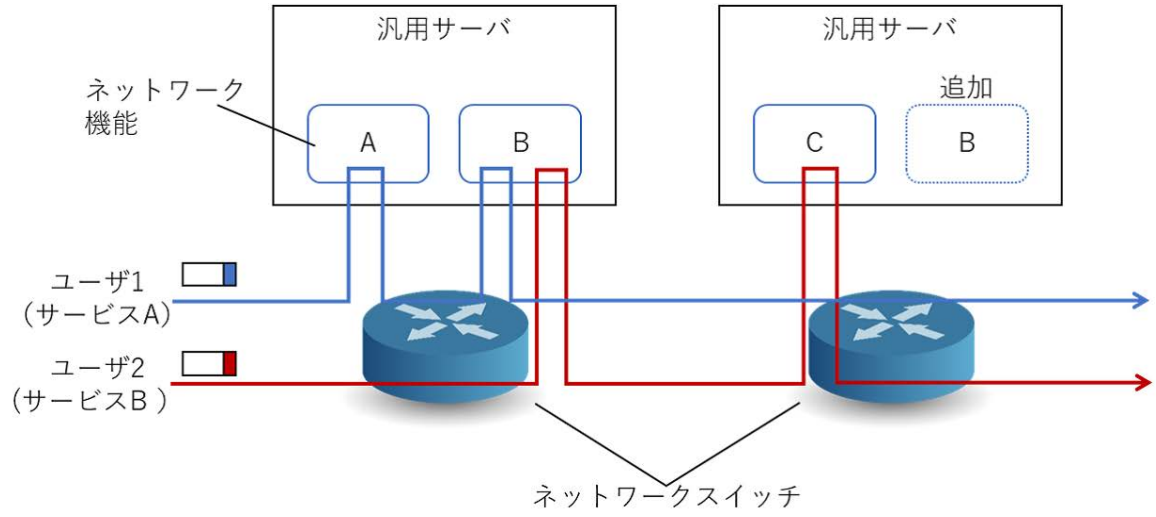


図 1.1　SFC ネットワークにおけるサービスチェイニング

におけるサービスチェイニングの例である．このネットワークはユーザ 1 とユーザ 2 のパケット転送経路を変えて異なるネットワーク機能を組み合わせることで，各ユーザの目的に応じたネットワークサービスを提供する．ネットワーク機能は汎用サーバに実装されるため，機能追加が容易である．そのため，ネットワーク機能 B のように複数のユーザが利用してネットワーク機能の負荷が増加しても，計算リソースが余っている別サーバに同じ機能を追加し，片方のユーザのパケット転送経路を変更することで各ユーザのサービス品質を維持できる．このように，SFC ネットワークでは各ユーザの目的に合致したネットワークサービスが効率的に提供される．

SFC 技術はネットワークサービスを柔軟に提供することを目的に検討されているが，より高度な情報処理サービスに対しても有用な技術である．例えば，SFC 技術を利用した映像監視サービスでは，監視カメラの置かれた環境や検出したい人/モノによって検出アルゴリズムを変えることで，ユーザごとに監視条件が異なった場合も高品質なサービスを提供できる．しかし，著者の知る限りこのような画像処理サービスは存在しない．

このような背景から，本研究では HWA を備えた汎用サーバを用いて画像処理機能を組み合わせる **Video Service Function Chaining（VSFC）**を提案する．VSFC ネットワークは SFC 技術を利用して複数のサーバノードに実装された画像処理機能を組み合わせることで，ユーザの目的に合致した画像処理サービスを提供する．しかし，各サーバノードで発生する遅延がサービス遅延として累積するため，VSFC による画像処理サービスは性能が低下する可能性がある．したがって，この提案で高品質なサービスを提供するための最も

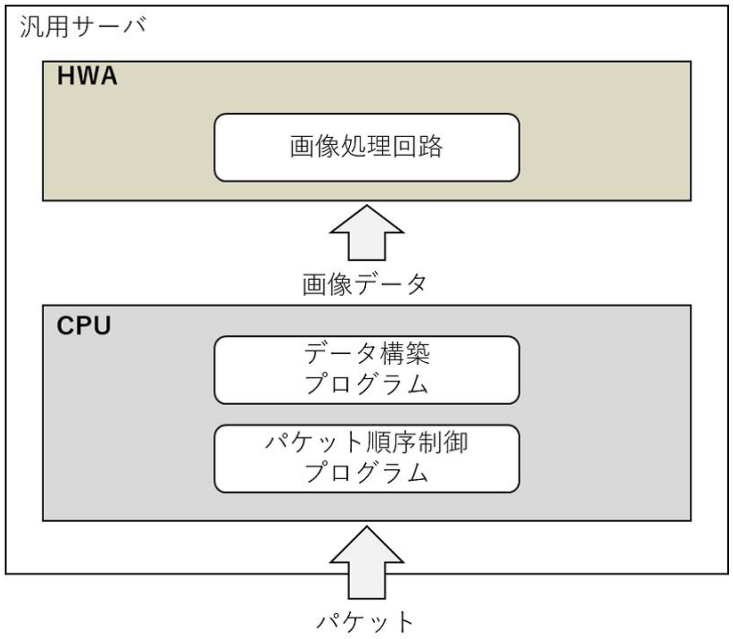


図 1.2 典型的な CPU-HWA サーバアーキテクチャにおける処理手順

重要な要件の一つは，サーバノードの処理遅延が小さいことである．

図 1.2 は典型的な CPU-HWA サーバアーキテクチャにおける処理手順を示している．このアーキテクチャは低速な CPU を用いて複数のパケットから画像データを再構築した後，アプリケーションレベルで画像データを HWA に転送するため，大きなデータ転送遅延が発生する．また，HWA に実装される画像処理回路は単一のクロックに同期して動作する固定レイテンシ回路であり，多くのアイドル時間により処理に時間がかかる．このように，従来のサーバノードは大きな処理遅延が発生するため，VSFC による低遅延な画像処理サービスに向いていない．

本研究は低遅延なサーバノードを開発し，VSFC による柔軟かつ低遅延な画像処理サービスを実現することを目的とする．この目的を達成するために，本研究ではサーバ内のデータ転送遅延と画像処理遅延の削減を検討した．データ転送遅延の削減に向けて**パケット順序制御回路を備えた CPU-HWA サーバアーキテクチャ**を提案する．また，画像処理の高速化に向けて **Approximate Completion-Detection（ACD）方式**の可変レイテンシ回路を提案する．実験により，提案技術によってサーバノード内の遅延が削減できることを示す．また，提案技術を用いたサーバノードにより，VSFC による低遅延な映像監視サービスを実現できることを示す．

1.2　研究の意義

本研究による貢献を以下に挙げる．

パケット順序制御回路を備えた CPU-HWA サーバアーキテクチャの提案

画像データを含むパケットをネットワークレベルで HWA に転送し，ハードウェアでパケット順序制御と画像処理を行う構成を提案した．この構成ではパケット順序制御回路が必要になる．パケット順序を高速に制御するために，パケットの順番が入れ替わった時だけパケットをバッファに保持するリアルタイムパケット順序制御回路を開発した．この構成により，画像処理回路へのデータ転送遅延を削減できることを示した．

Approximate Completion-Detection 方式の可変レイテンシ回路の提案

いくつかの画像処理サービスは必ずしも正確な計算を必要としない．そこで，計算エラーを許容するアプリケーションを高速化するための回路動作方式を提案した．提案方式は組み合わせ回路におけるタスクの完了を大まかに検出し，各タスクの処理時間を変える．この方式の可変レイテンシ回路はタスク完了を誤検出することで誤った処理を行う可能性があるが，エラー率は制御可能であり，低いエラー率で高速に動作する．この技術により，画像処理回路を高速化できることを示した．

VSFC 対応映像監視サービスの開発

VSFC による柔軟な画像処理サービスの例として，2 種類の画像処理機能（HOG 特徴量計算，フレーム間差分法による移動物体検出）からなる映像監視サービスを開発した．実験において，提案アーキテクチャを FPGA 搭載サーバで実現し，2 種類の画像処理機能を組み合わせることで，50 台のネットワークカメラからの 60fps の VGA 映像をリアルタイムに監視できることを確認した．また，画像処理機能に ACD 方式の可変レイテンシ回路を用いることで，サービス品質をほとんど低下させることなく処理性能が向上することを確認した．これらの結果から，提案技術を用いたサーバノードにより，VSFC による低遅延な映像監視サービスを実現できることを確認した．

1.3　本論文の構成

本論文の構成を以下に示す．本論文は全 6 章から成る．

「**第 2 章 Video Service Function Chaining**」では，画像処理機能を組み合わせて柔軟な画像処理サービスを実現する VSFC を提案する．まず，既存の SFC ネットワークの基本構成を紹介し，次に，SFC 技術を利用して複数のサーバノードに実装された画像処理機能を組み合わせる VSFC ネットワークのコンセプトを説明する．

「**第 3 章 パケット順序制御回路を備えた CPU-HWA サーバアーキテクチャの提案**」では，サーバノード内で低遅延データ転送を実現する CPU-HWA アーキテクチャを提案する．始めに開発の動機について述べ，提案アーキテクチャのコンセプトと構成を説明する．このアーキテクチャではハードウェアで高速にパケット順序制御を行う必要がある．そこで，小さな遅延でパケットを並び替えることができるリアルタイムパケット順序制御回路を提供する．実験では，HOG 特徴量計算を，提案アーキテクチャで行うと，従来構成に比べてデータ転送および処理を高速かつ低消費電力で実現でき，パケット順序の入れ替えが大きい場合でも高速な処理が可能であることを示す．

「**第 4 章 Approximate Completion-Detection 方式の可変レイテンシ回路の提案**」では，計算エラーを許容するアプリケーションにおいて，処理を高速化するための回路動作方式を提案する．本章では開発の動機を述べた後，提案方式の回路動作，回路構成，および回路設計手法を説明する．実験では，画像処理機能を提案の ACD 方式の可変レイテンシ回路で実現し，わずかな計算エラーを許容することで従来方式の回路と比べて処理性能が向上することを示す．また，設計した画像処理回路は，従来の回路構成と比べて面積遅延積が小さいことを示す．

「**第 5 章 VSFC 対応映像監視サービスの評価**」では，2 種類の画像処理機能（HOG 特徴量計算，フレーム間差分法による移動物体検出）からなる VSFC 対応映像監視サービスの性能を評価する．まず，提案アーキテクチャを FPGA 搭載サーバで実現し，2 種類の画像処理機能を組み合わせることで，VGA 映像をリアルタイムに監視できることを示す．次に，画像処理機能に ACD 方式の可変レイテンシ回路を用いることで，サービス品質をほとんど低下させることなく処理性能が向上し，低遅延な映像監視サービスを実現できることを示す．

最後に，「**第 6 章 おわりに**」で本論文をまとめる．本研究の貢献を整理し，今後の展望について述べる．

第 2 章

Video Service Function Chaining

本章では，画像処理機能を組み合わせて柔軟な画像処理サービスを実現する VSFC を提案する．始めに前提となる SFC について説明し，SFC ネットワークの基本構成を紹介する．次に SFC 技術を利用して複数のサーバノードに実装された画像処理機能を組み合わせる VSFC ネットワークのコンセプトを説明する．

2.1 SFC

SFC[13] はネットワーク機能を動的に選択し連携させる**サービスチェイニング**を実現するためのパケット転送方式である．この方式はパケットにネットワーク機能の組み合わせ（**サービスチェイン**）を記載したタグを付与し，タグに基づいて適切なネットワーク機能に適切な順序でパケットを転送することでサービスチェイニングを実現する．SFC によるネットワーク（SFC ネットワーク）では，ユーザの目的に応じたサービスチェインを記載したタグをパケットに付与することで，各ユーザに適切なサービスを柔軟に提供することができる．

従来の IP ルーティングによってサービスチェイニングを行うと，ユーザを追加する度に複雑なネットワーク設計が必要になるため，迅速な対応が難しい．一方，SFC ネットワークはパケットにタグを付与するだけで新規ユーザを既存サービスに参加させることができる．また，新しいサービスを開始する場合はいくつかのネットワークスイッチへの設定が必要になるが，経路情報がパケットに付与されるためわずかな設定で済む．これらの特徴により，SFC ネットワークはユーザ数の増加に対して迅速に対応することができる．

図 2.1 に SFC ネットワークの基本構成を示し，各機能の役割を以下で説明する．SFC ネットワークを構成する機能 [14] は Internet Engineering Task Force（IETF）において標

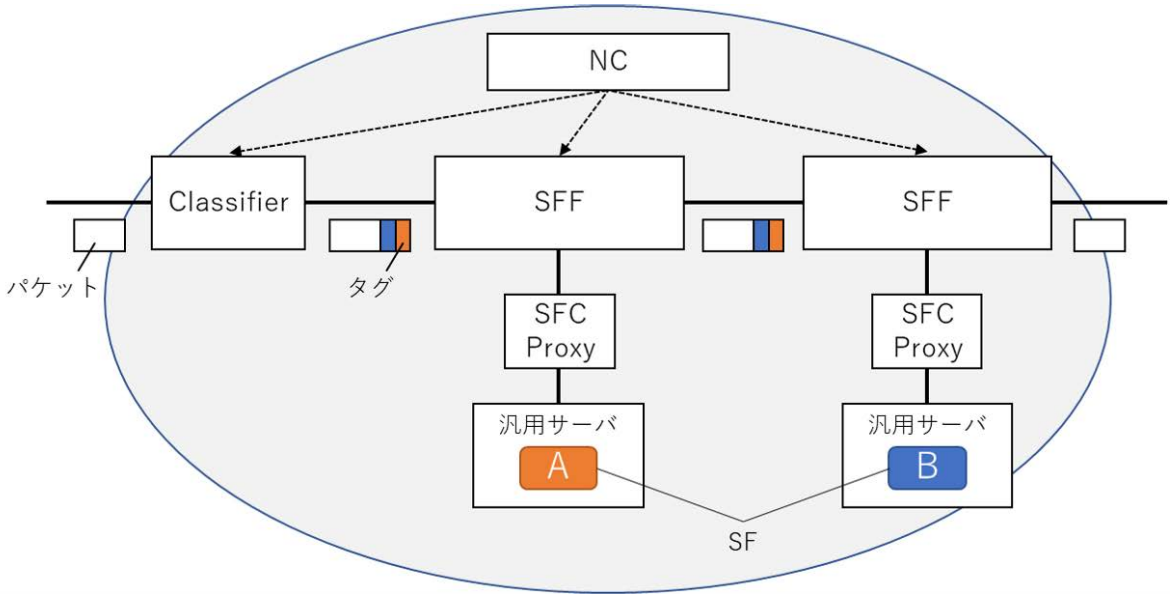


図 2.1 SFC ネットワーク構成

準化のための議論が行われている．さらに，SFC に関する研究 [15, 16] が行われている．

Service Function（SF）

SF はサービスチェインを構成するネットワーク機能である．NFV においては汎用サーバ上や仮想マシン上にソフトウェアとして実装される．文献 [17] や [18] はネットワーク機能を高速化するために FPGA を用いたプラットフォームを提案している．文献 [14] は，SF の例としてファイアウォールや Network Address Translators（NAT），ロードバランサなどを検討している．

Classifier

Classifier は受信したパケットをユーザごとに識別し，その結果に応じてパケットにタグを付与する．IETF は Network Service Header（NSH）[19] を用いたタグの付与を検討している．NSH ヘッダフォーマットを図 2.2 に示す．このパケットヘッダにはサービスチェインの識別子である Service Path Identifier（SPI），サービスチェイン内の現在位置を示す Service Index（SI），およびメタデータが含まれる．後述する SFF が SPI と SI の組み合わせからサービスチェインに含まれる SF とその順序を識別し，適切な SF にパケットを転送する．SFC ネットワーク内でタグに基づいたパケット転送を行うために，Classifier は外部ネットワークと SFC ネットワークの境界に設置される．

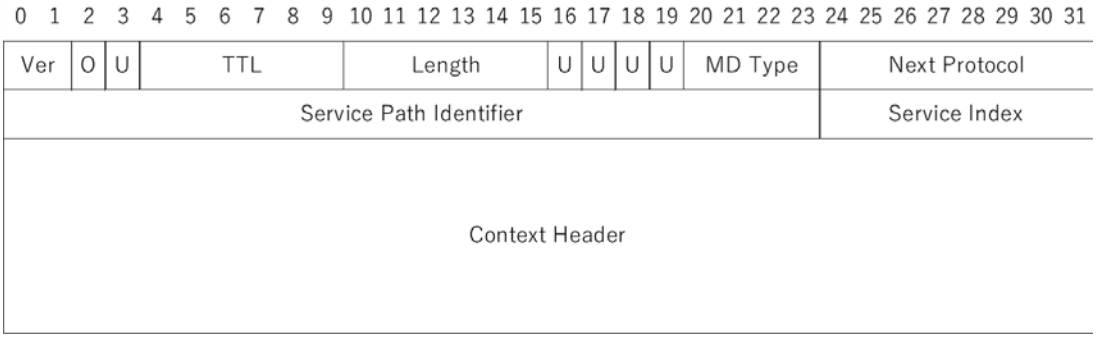


図 2.2　NSH ヘッダフォーマット

Service Function Forwarder（SFF）

SFF はパケットに付与されたタグを参照し，パケットを適切な SF に転送する．サービスチェインを構成するすべての SF で処理が終わると，SFC ネットワークの端に設置された SFF がタグを削除する．そのため，SFC ネットワーク内のパケット転送は外部ネットワークに影響しない．

SFC Proxy

SFP Proxy は SFC に対応していない SF の代わりにタグの除去と再付与を行う．この機能により SFC 非対応のネットワーク機能を SFC ネットワーク上で利用することができる．

Network controller（NC）

NC はアプリケーションとスイッチコントローラから構成される．アプリケーションはユーザが要求したネットワークサービスと SFC ネットワーク内の SF の配置状況からパケットに付与するタグを計算する．この計算に向けて複数のスケジューリングアリゴリズム [20, 21, 22] が提案されており，アプリケーションはこれらのアルゴリズムを用いて最適なサービスチェインを計算することができる．スイッチコントローラはアプリケーションからタグを受け取り Classifier に登録する．また，スイッチコントローラは SFF にタグに対応したフローエントリを登録する．これらの処理は SFF とスイッチコントローラが OpenFlow[23] に対応することで簡単に行うことができる．OpenFlow に対応したスイッチコントローラは Ryu[24] や Trema[25]，Floodlight[26] などがある．

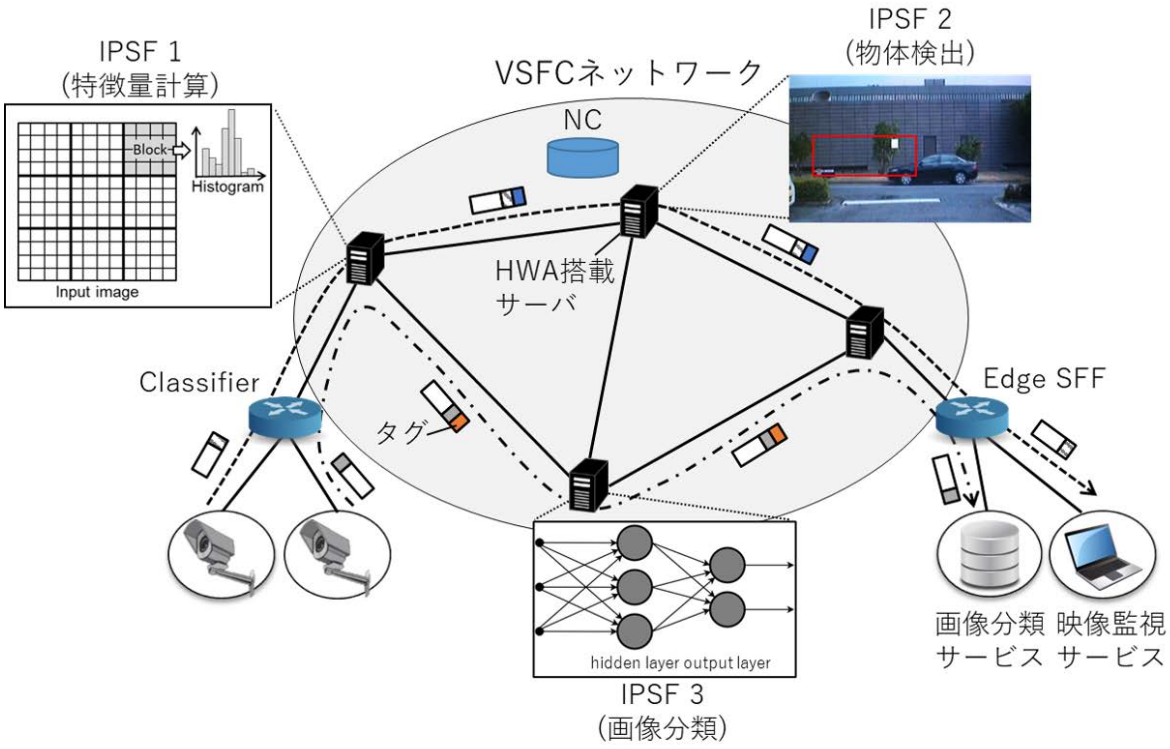


図 2.3　VSFC のコンセプト

2.2　VSFC ネットワーク

VSFC は画像処理サービスを柔軟に提供するためのコンセプトである．VSFC によるネットワーク（VSFC ネットワーク）は画像処理の SF（Image processing SF，IPSF）を組み合わせて複数の画像処理サービスを提供する．VSFC の利点の一つは IPSF を複数のユーザで共有することで，少ないリソースで画像処理サービスを効率的に提供できることである．例えば，図 2.3 の VSFC ネットワークは IPSF1（特徴量計算）と IPSF2（物体検出）を組み合わせて映像監視サービスを提供し，IPSF1 と IPSF3（画像分類）を組み合わせて画像分類サービスを提供する．このようにサービスごとに IPSF を用意しなくても IPSF を共用して複数のサービスを提供することが可能である．また，VSFC ネットワークがアルゴリズムの異なる特徴量計算機能を複数持つことで，それらを IPSF2 や IPSF3 と組み合わせて異なるタイプの映像監視サービスや画像分類サービスを提供できる．

VSFC ネットワークは SFC 向けの Classifier，SFF，SFC Proxy，NC と IPSF で構成する

ことができる．これらの機能は異なるサーバに実装することもできるが，一つのサーバにいくつかの機能をまとめて実装するのが望ましい．特に，SFF，SFC Proxy と複数の IPSF を一つのサーバに実装することでネットワーク内のサーバ台数を減らせるだけでなく，パケット転送が効率化されサービス遅延を減らすことができる．サーバの詳細な構成は 3 章で説明する．

VSFC ネットワークにおけるサービスチェイニングを図 2.3 を用いて説明する．始めに VSFC ネットワークの入り口に設置した Classifier を用いてパケットにサービスチェインを識別するタグを付与する．その後，パケットを最寄りの SFF に転送し，さらにタグ情報をもとに適切な IPSF に適切な順序で転送する．IPSF の前後では SFC Proxy を用いてタグの削除と再付与を行う．VSFC ネットワークの端に設置された SFF（Edge SFF）はサービスチェイニングが完了したパケットからタグを削除し，外部ネットワークにパケットを送信する．NC はタグの計算と Classifier へのタグ登録，SFF へのフローエントリ登録を行う．このように SFC 技術を利用した VSFC ネットワークはユーザごとの画像処理サービスを柔軟に提供することができる．

VSFC ネットワークでは画像処理サービスを構築するために複数のサーバに分散した IPSF を連携させるため，各サーバで発生した処理遅延やサーバ間の通信遅延が累積する．その結果，VSFC による画像処理サービスの遅延は一台のサーバで提供されるサービスの遅延よりも大きくなる．サーバ間の通信遅延はサーバを物理的に近づけることで小さくできるが，依然としてサーバ内で大きな遅延が発生する．したがって，サーバ内の遅延を削減することが VSFC による画像処理サービスを低遅延で提供するために重要である．

第 3 章

パケット順序制御回路を備えた CPU-HWA サーバアーキテクチャの提案

本章では，サーバノード内のデータ転送遅延の削減に向けてリアルタイムパケット順序制御回路を備えた CPU-HWA サーバアーキテクチャ [27, 28] を提案する．実験では，提案アーキテクチャにおける画像処理機能の処理遅延と消費電力を評価した．さらに，パケット順序が入れ替わる状況での処理性能を評価した．

3.1 開発の動機

複数の HWA 搭載サーバを用いて提供される VSFC 対応画像処理サービスは，1 台のサーバで提供されるサービスよりもサービス遅延が大きい．例えば，図 2.3 における映像監視サービスでは，2 台の HWA 搭載サーバの処理遅延とサーバ間の通信遅延がサービス遅延として累積する．サーバ間の通信遅延は同じデータセンタ内のサーバを用いて距離を近くすることで小さくできる．一方，サーバ内ではアプリケーションレベルのデータ転送により大きな処理遅延が発生する．よって，サーバ内のデータ転送遅延を減らすことを検討した．

ネットワーク上で動作する HWA 搭載サーバは，一般に複数のパケットに分割された画像データを受信する．従来の CPU-HWA サーバアーキテクチャでは CPU で画像データに再構築してから HWA に転送する．この手順は以下の理由で大きなデータ転送遅延が発生する．

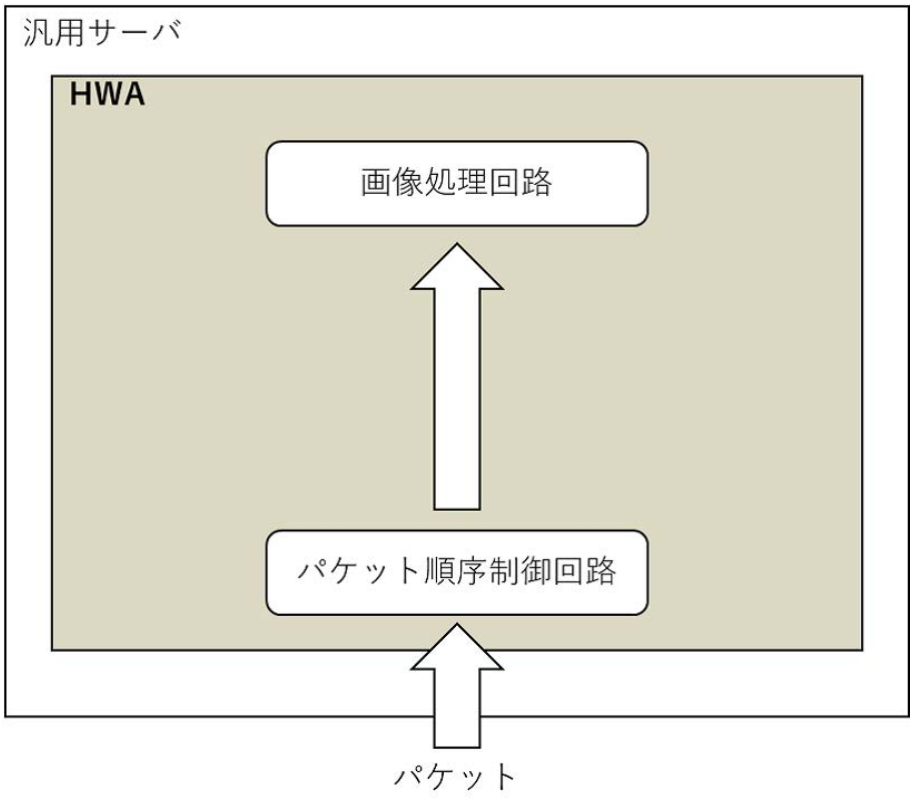


図 3.1　提案アーキテクチャにおける画像処理手順

- CPU と HWA 間の画像データ転送をアプリケーションレベルで制御することによる通信オーバヘッドが発生する．
- パケットが異なるネットワークルータを通過することで，サーバは順不同のパケットを受信する可能性がある．これにより誤った画像データが再構築されると，適切な画像処理を行うことができない．したがって，低速な CPU を用いてパケット順序を制御する必要がある．

3.2　提案アーキテクチャ

我々のアーキテクチャは画像データを含むパケットをネットワークレベルで HWA に転送し，ハードウェアでパケット順序制御と画像処理を行う（図 3.1）．この構成では受信したパケットを直接 HWA に転送するため，CPU で発生するアプリケーションレベルのオーバヘッドを回避できる．既存研究 [29, 30, 31] は CPU と FPGA 間で 40 Gbps を超える高

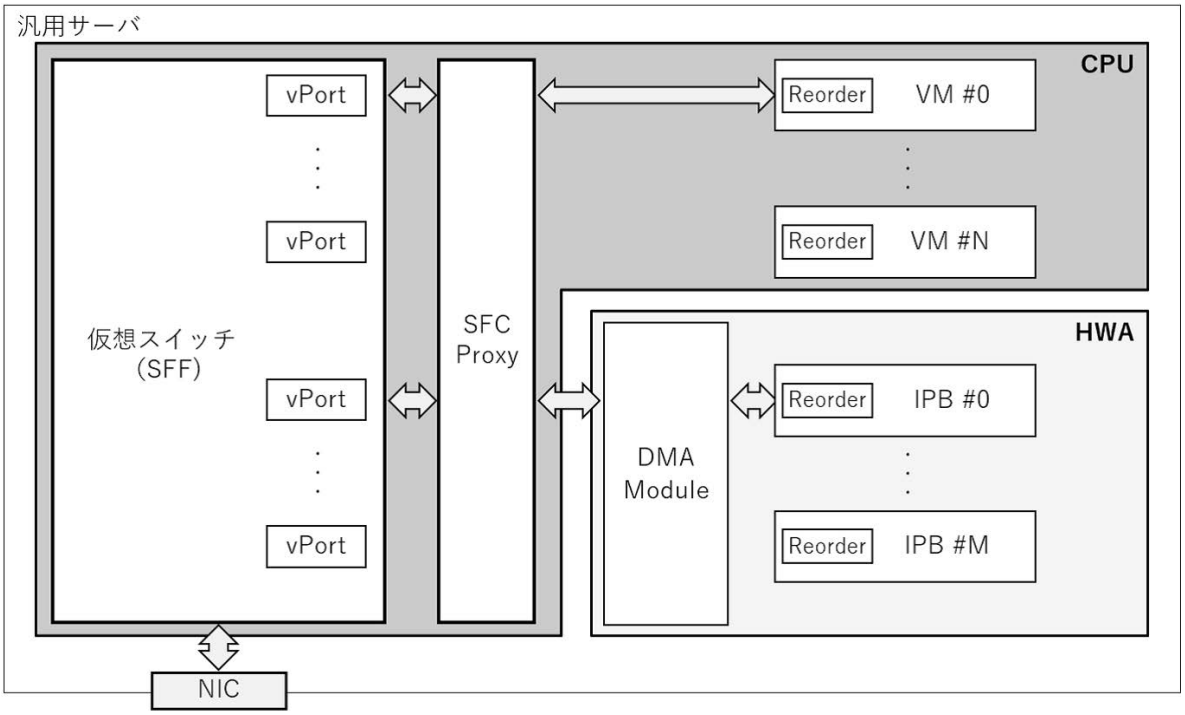


図 3.2　VSFC サーバノードの構成

速パケット転送を実現した．一方，これらの研究はパケット単位で処理を行う NFV アプリケーションを対象としており，パケット順序は考慮されていない．よって，この構成では正しいデータに基づく画像処理を行うために，パケット順序制御回路が必要である．

提案アーキテクチャの VSFC サーバノードを図 3.2 に示す．このサーバノードは IPSF を実行するための**画像処理ブロック（Image Processing Block，IPB）**および仮想マシン（Virtual Machine，VM）を備える．IPB はハードウェア処理により単純かつ計算量の多い IPSF を高速に実行する．一方，VM はハードウェアアクセラレーションが効かない IPSF を実行する．本システムはこれら 2 種類のプラットフォームにより様々な IPSF に対応する．また，本システムは IPB と VM を追加，削除することで IPSF の種類や数の変更に対応する．IPB は FPGA に実装することで全体再構成や部分再構成 [32] により変更できる．

仮想スイッチ（SFF）は受信したパケットのタグを確認して，別サーバの仮想スイッチ，IPB，もしくは VM のいずれかにパケットを転送する．外部との通信は Network Interface Card（NIC）を介して行い，IPB や VM との通信は仮想ポートを介して行う．IPB との通信はアプリケーションレベルの通信オーバヘッドを避けるために，Direct Memory Access（DMA）転送 [33, 34] で行う．また，仮想スイッチと IPB，VM の間に置いた SFC Proxy

でパケットのタグを管理する．この構成では IPB と VM に転送するパケットの順序が保証されない．よって，IPB と VM にパケット順序制御機能を具備する．

VSFC サーバノードの仮想スイッチは複数の IPB と VM，および外部からパケットを受信するため，大量のパケットを転送する必要がある．高速パケット転送を行う仮想スイッチは Open vSwitch[35] や VPP[36]，Lagopus[37] などがある．VPP を改良した CuVPP[38] は 80Gbps の入力レートに対するパケット転送を実現している．これらの仮想スイッチは現時点で NSH ヘッダをサポートしておらず，タグによるパケット転送は行えないが，オープンソースソフトウェア（OSS）であるため短期間の開発で対応可能である．仮想スイッチと SFC Proxy を NC で管理することで，ユーザごとのサービスチェインを制御できる．

以上の構成によりアプリケーションレベルの通信オーバヘッドを回避し，小さな遅延で画像データを IPB に転送できる．なお，VM に関しては本研究の対象外のため詳細な検討は行わない．

3.3　パケット順序制御

提案アーキテクチャはハードウェアによるパケット順序制御を必要とする．本節では，ハードウェア実装に適した低遅延かつ省リソースなパケット順序制御回路を提案する．

3.3.1　従来手法

パケット順序逆転 [39] はインターネットにおいて一般的な現象である．そのため，パケット順序を制御する方法が古くから検討されている．古典的なパケット順序制御アルゴリズムである Calendar Queues[40] はキューのサイズを動的に変更することで定数時間でのパケット順序制御を実現している．しかし，ハードウェアのキューサイズは製造段階で固定されるため，このアルゴリズムは残念ながらハードウェア実装に適していない．Priority Queue[41] は複数のシフトレジスタとコンパレータを用いてパケット順序を制御する．このキューは多くのリソースを必要とするため，IPB の性能に悪影響を与える可能性がある．Data Plane Development Kit（DPDK）[42] は CPU でパケット処理を行うためのデファクトスタンダードな技術であり，その機能の一つとして DPDK Reorder Library が提供されている．DPDK Reorder Library は Order buffer と Ready buffer と呼ばれる 2 つのバッファを用いてパケット順序を制御する（図 3.3）．前者のバッファはパケットの並び替えを行い，後者のバッファは正しい順序のパケットを後段に転送する．DPDK Reorder Library は具体的に以下の手順でパケットを並び替える．

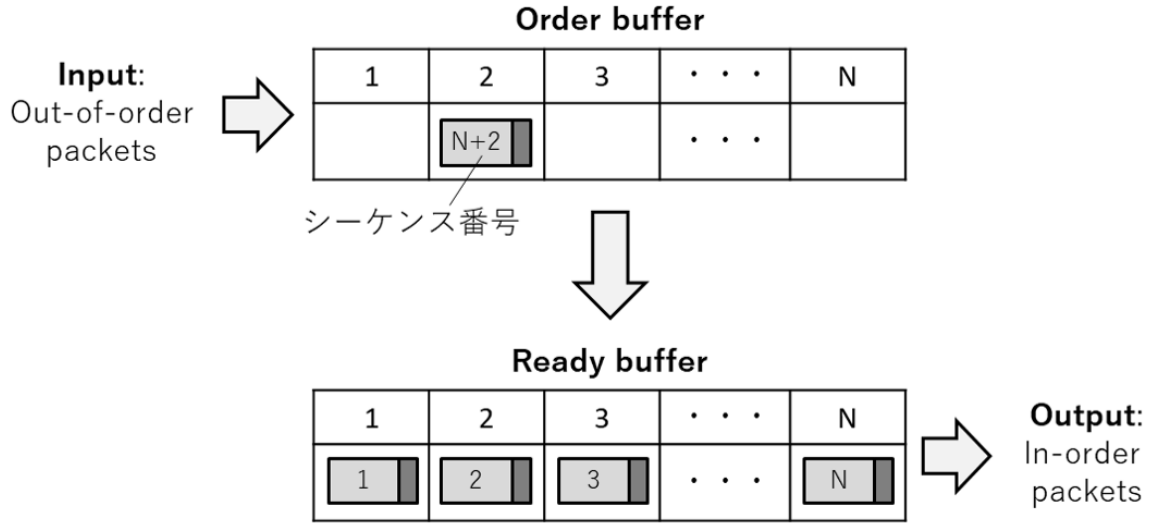


図 3.3　DPDK Reorder Library[43] のパケット順序制御

1. Order buffer は受信したパケットのシーケンス番号を参照し，格納先を計算する．
2. シーケンス番号に応じた格納先が空であれば，パケットをその位置に保持する．
3. 格納先が空でなければ，Order buffer は保持するパケットの中でシーケンス番号が最も小さいものを Ready buffer に渡す．
4. 3 の操作を Order buffer がパケットを受け入れ可能になるまで繰り返す．
5. パケットを格納する．

DPDK Reorder Library は各バッファを別スレッドに割り当てることでパケットの並び替えとパケット送信を並列に実行できる．しかし，パケット順序が入れ替わらない状況では Order buffer が一杯になるまでパケットは Ready buffer に渡されない．すなわち，順序の正しいパケットも一定時間バッファ内で待機するため，遅延が大きい．さらに，このアルゴリズムは 2 つの同じ大きさのバッファのために多くのメモリリソースを必要とする．以上から，このアルゴリズムはハードウェア実装に適していない．

3.3.2　リアルタイムパケット順序制御回路

リアルタイムパケット順序制御

従来の CPU-HWA サーバアーキテクチャは遅延の大きいパケット順序制御を行った後で画像処理を行う．これらの処理は逐次実行されるためシステム全体の処理遅延が大きい（図 3.4a）．一方，提案アーキテクチャはパケット順序制御と画像処理を可能な限り並列実

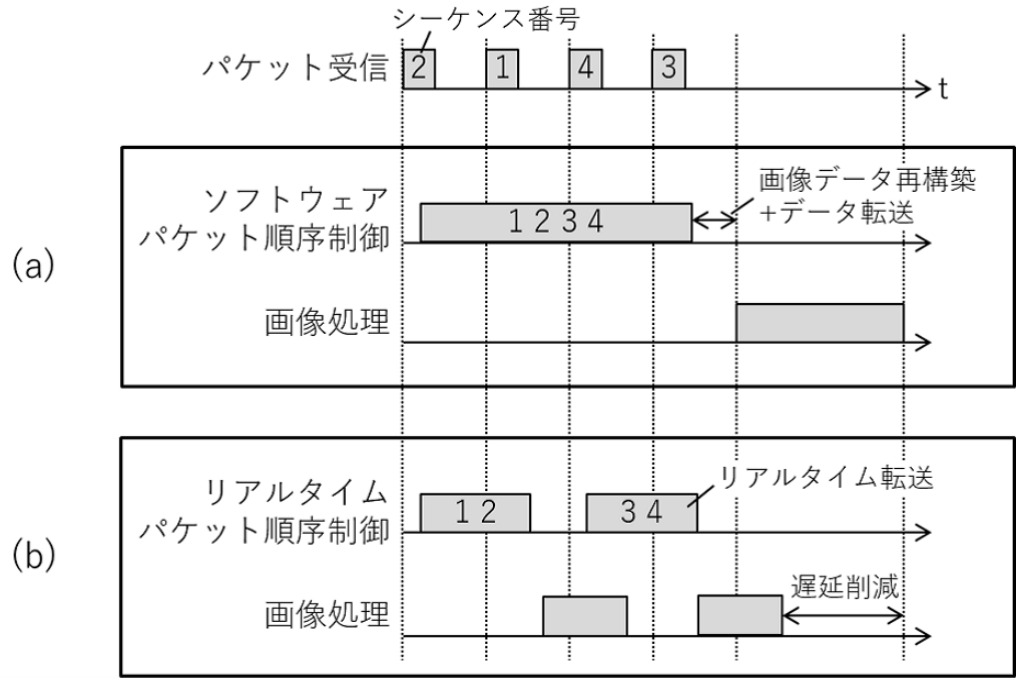


図 3.4　(a) 従来構成と (b) 提案構成における処理遅延

行することで低遅延な処理を実現する（図 3.4b）．この動作を実現するためのリアルタイムパケット順序制御回路を提案する．この回路は正しい順序のパケットをすぐに IPB に転送するリアルタイムパケット順序制御を行う．これにより IPB は必要なデータが揃った段階ですぐに処理を開始できる．さらに，IPB は小さなデータに対して画像処理を行うため，完全な画像データを処理する回路と比べて回路規模が小さくて済む．

リアルタイムパケット順序制御回路はシーケンス番号を有するパケットを受信することを想定している．シーケンス番号を有する通信プロトコルは Transmission Control Protocol（TCP）[44] や Real-time Transport Protocol（RTP）[45] がある．RTP は動画などのストリームデータをリアルタイムに配送するための軽量な通信プロトコルであり，低遅延を必要とする画像処理サービスに適している．

回路構成

リアルタイムパケット順序制御回路の構成を図 3.5 に示す．この回路は順序が入れ替わったパケットを識別するための ID comparator と，パケットを格納する Reorder buffer から構成される．ID Comparator は自身が持つ参照値（送信 ID）とパケットのシーケンス番号を比較し，パケットを Reorder buffer に転送するかどうかを決める．Reorder buffer は

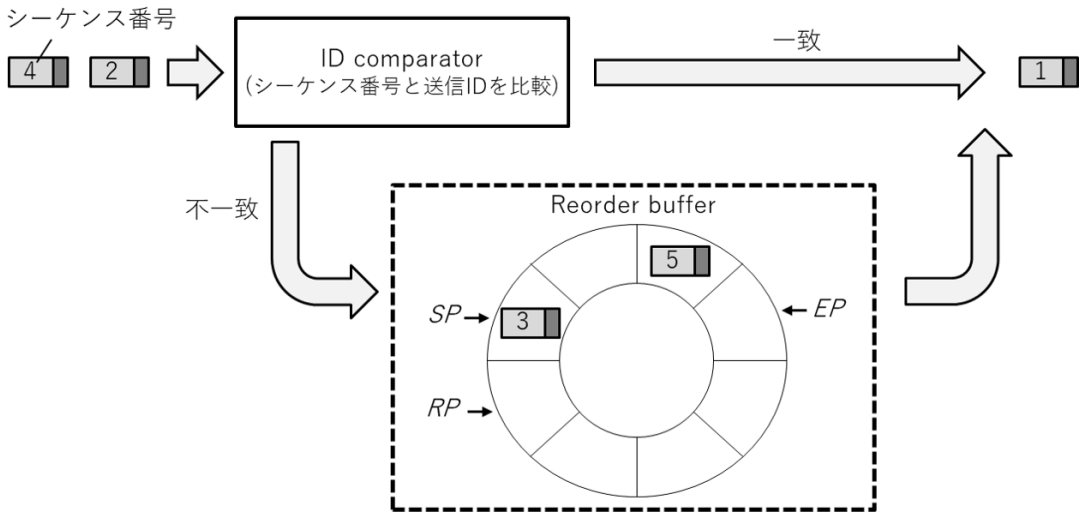


図 3.5　リアルタイムパケット順序制御回路

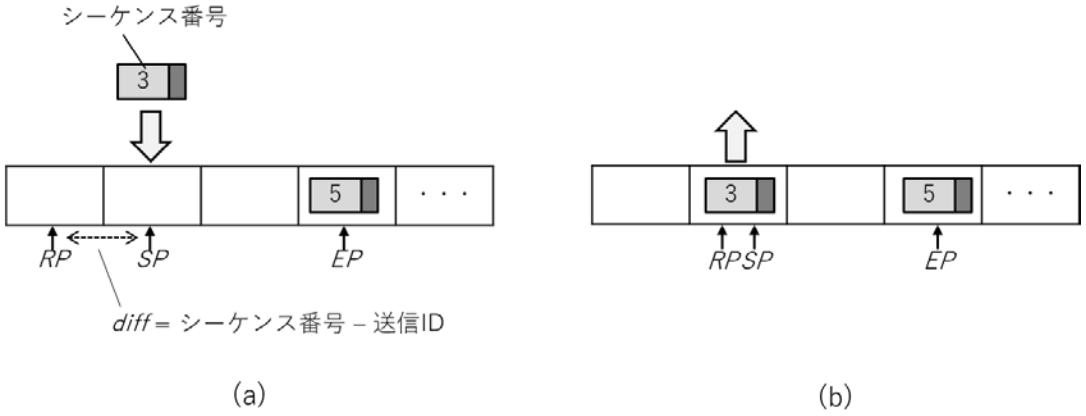


図 3.6　制御変数を用いた（a）パケット格納処理と (b) パケット取り出し処理

Reference pointer（RP），Start pointer（SP）および End pointer（EP）の 3 つの制御変数を用いてリングバッファ内のパケットを管理する．RP はバッファ内の現在位置を示し，SP と EP はバッファ内で最小および最大のシーケンス番号を持つパケットの格納位置を示す．これらの制御変数により，Reorder buffer は適切な位置にパケットを格納し，適切なタイミングでパケットを取り出すことができる（図 3.6）．

リアルタイムパケット順序制御回路の動作を説明する．この回路がパケットを受信したときのフローチャートを図 3.7 に示す．この回路がパケットを受信すると，ID comparator によってパケットのシーケンス番号と送信 ID が比較される．2 つの値が同じだった場合，ID comparator はパケットを直接 IPB に転送し，送信 ID と RP をインクリメントする．そ

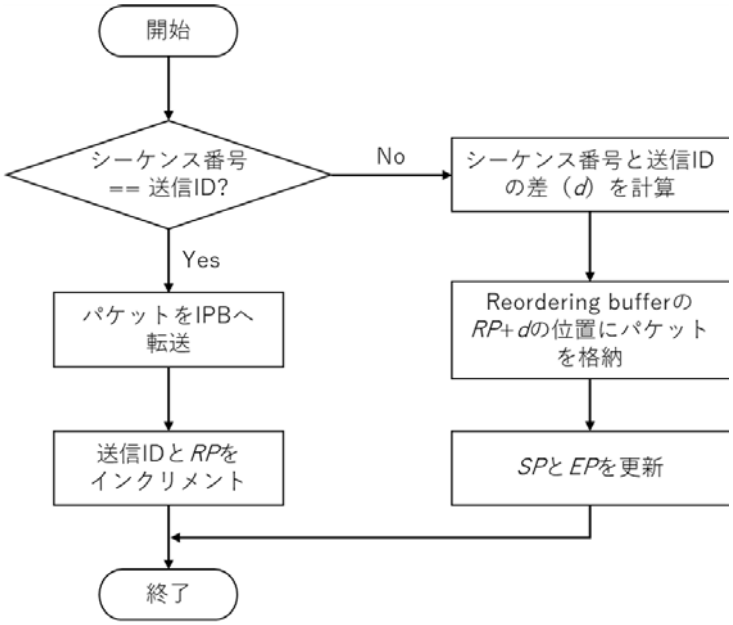


図 3.7　リアルタイムパケット順序制御回路がパケットを受信した時のフローチャート

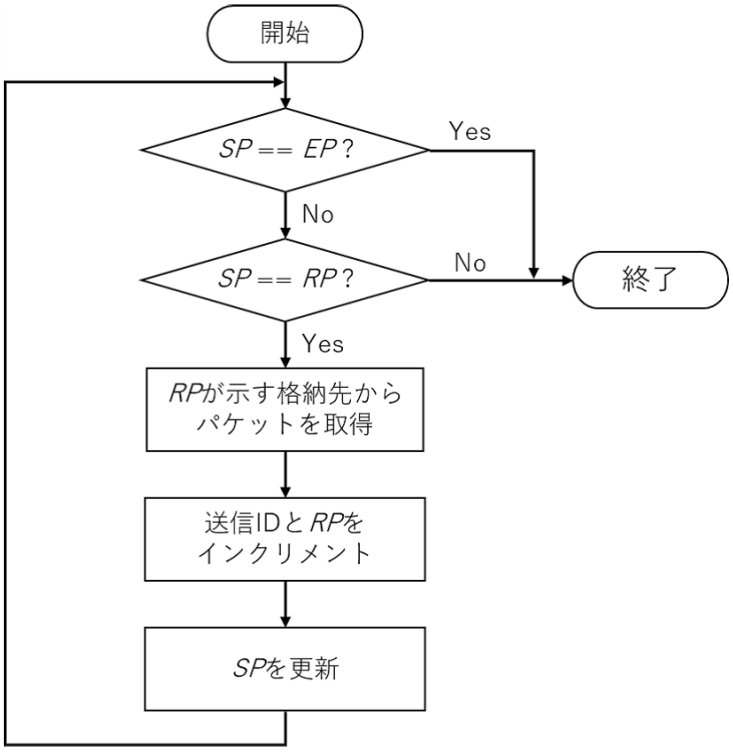


図 3.8　Reorder buffer からパケットを取り出す時のフローチャート

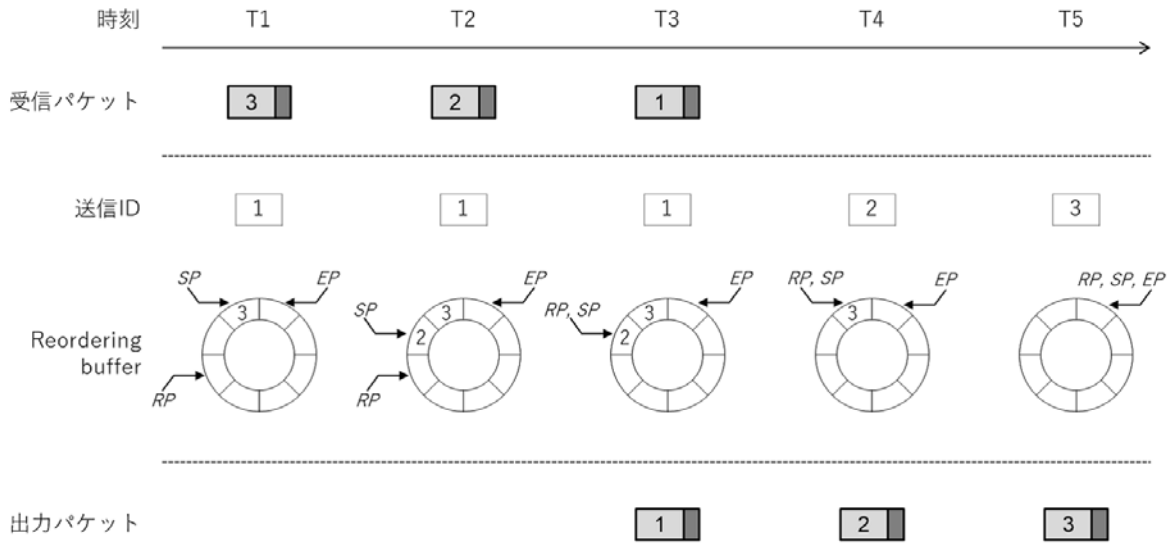


図 3.9　Reorder buffer の動作例

れ以外の場合，ID comparator はパケットを Reorder buffer に転送する．Reorder buffer はシーケンス番号と送信 ID の差（d）を計算し，RP+d の位置にパケットを格納する．さらに，格納先が SP より前だった場合，SP をその位置に更新する．また，格納先が EP よりも後だった場合，EP をその位置の次に更新する．

Reorder buffer からパケットを取り出すタイミングは RP を更新したときである．そのときのフローチャートを図 3.8 に示す．更新した RP が SP と同じ位置を指していた場合，その位置のパケットを Reorder buffer から取り出し，IPB に転送する．その後，送信 IP と RP をインクリメントする．さらに，SP をインクリメントしてバッファ内で最小のシーケンス番号のパケットを検索する．その結果，SP と RP が再び同じ位置を指した場合，パケットを取り出す．この処理は SP と RP が異なる場合，もしくは SP と EP が同じ場所を指した場合に終了する．前者の状況はバッファ内に正しい順序のパケットがないことを意味し，後者はバッファ内のすべてのパケットを IPB に転送したことを意味する．

Reorder buffer の動作例を図 3.9 に示す．この回路は送信 ID の初期値を 1 とする．時刻 T1 と T2 おいて，送信 ID と一致しないシーケンス番号のパケットを受信したとき，Reorder buffer はそれらのパケットを適切な位置に格納する．その後，時刻 T3 においてシーケンス番号 1 のパケットが IPB に転送されると，Reorder buffer はパケット取り出し処理を開始する．Reorder buffer は RP と SP が同じ位置を指している間，その位置のパケットを IPB に転送する（時刻 T4，T5）．そして SP と EP が同じ位置を指した結果，パ

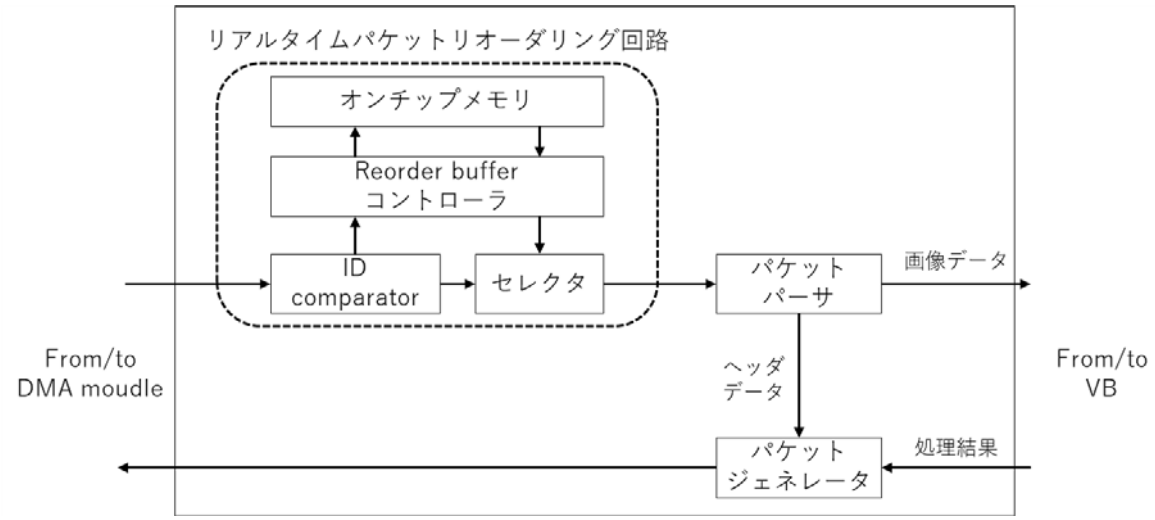


図 3.10　IPB インタフェースのブロック図

ケットの取り出し処理を終了する．このように，3 つの制御変数を用いることで 1 つのバッファでリアルタイムにパケット順序を制御できる．

ネットワーク上でパケットロスが発生すると，送信 ID と一致するシーケンス番号を持つパケットを受信できないため，リアルタイムパケット順序制御回路はストールする．この状況に陥った回路はタイムアウト機能によって Reorder buffer 内のパケットを IPB に転送し，送信 ID と制御変数を更新する．これにより回路は動作を継続できるが，IPB に不完全な画像データを渡すことになる．不完全な画像データへの対応はアプリケーションによって異なり，不完全な画像データを無視する方法や周辺ピクセルから欠けたピクセルを補間する方法がある．例えば，物体検出アプリケーションは一部の画像データが失われても物体を検出できるため，このアプリケーションでは不完全な画像データを無視できる．

リアルタイムパケット順序制御回路の重要な特徴は正しい順序のパケットをすぐに IPB に転送することである．したがって，この回路はバッファ内で無駄な待機時間が発生しない．さらに，単純なコンパレータと 1 つのバッファでパケット順序を制御するため，この回路は少ないリソースで実現できる．

3.4　IPB インタフェース

IPB が正しい画像データを受信し，処理結果を送信するために，そのインタフェースはパケット順序制御機能だけでなく，パケットに含まれる画像データを取り出す機能と処理

表 3.1 評価実験に用いたデバイス

項目	デバイス
CPU	Xeon Gold 6132 @2.60 GHz
FPGA	Arria 10 GX 1150
Memory	64GB

結果をパケット化する機能を備える必要がある．図 3.10 にこれらの機能を備えた IPB インタフェースの構成を示す．このインタフェースはパケットを受信するとリアルタイムパケット順序制御回路で高速に並び替えた後，パケットパーサで画像データとヘッダデータに分離する．そして，画像データを IPB に転送し，ヘッダデータをパケットジェネレータに転送する．パケットジェネレータは IPB の処理結果とヘッダデータを組み合わせてパケットを作成し，DMA モジュールに送信する．以上の処理により IPB はこのインタフェースを介してパケット化されたデータを送受信できる．

IPB インタフェースは複雑な計算を行わないため，少ない回路リソースで実装することができる．ただし，Reorder buffer のサイズを小さくするとパケットロスが発生し，IPB に不完全なデータを渡す可能性が高くなる．そのため，Reorder buffer の大きさは想定する外部環境に応じて慎重に決めなければならない．

3.5 性能評価

提案アーキテクチャの有効性を確認するため，IPB インタフェースと画像処理回路で構成される画像処理機能を開発し，その処理遅延と消費電力を評価した．さらに，パケット順序が入れ替わる状況でのスループットを評価した．本節では，評価結果について説明する．

3.5.1 実装

評価実験のために，図 3.11 に示す 3 種類の構成の画像処理機能を開発した．これらの機能は表 3.1 のデバイスに実装した．SW-SW 構成と SW-HW 構成の機能ではパケット順序制御を CPU で行い，画像処理をそれぞれ CPU と FPGA で行う．一方，HW-HW 構成の機能ではパケット順序と画像処理を FPGA で行う．HW-HW 構成が IPB に対応する．

CPU を用いたパケット順序制御のために DPDK Reorder Library ベースのプログラムを

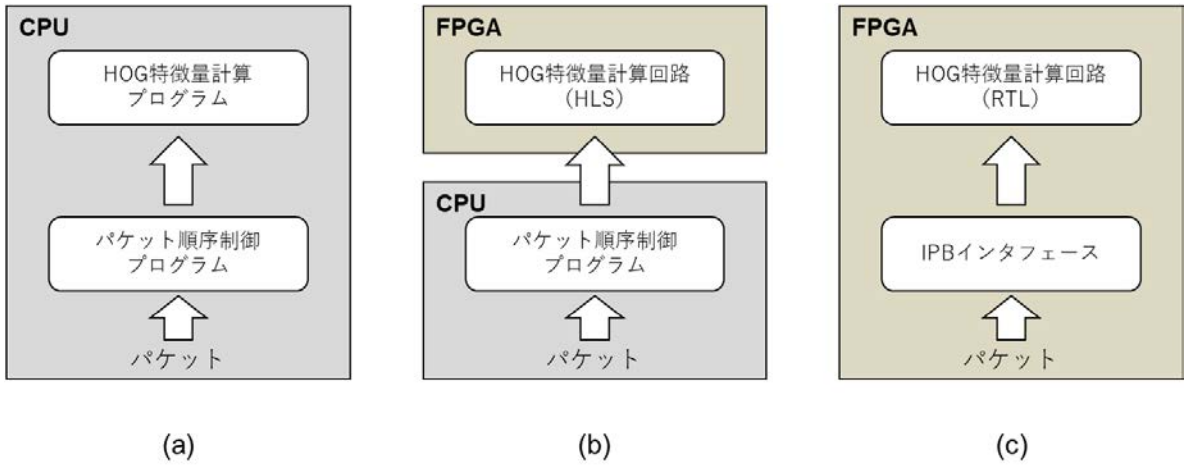


図 3.11　(a)SW-SW 構成，(b)SW-HW 構成，(c)HW-HW 構成の HOG 特徴量計算機能

開発した．付録 A.1 にプログラムのソースコードを示す．一方，FPGA を用いたパケット順序制御のために IPB インタフェースを開発した．これらのパケット順序制御では，VGA 画像 1 枚分のパケットまで保持できるようにした．また，Histograms of Oriented Gradients（HOG）特徴量 [46] を計算する画像処理プログラムおよび回路を開発した．HOG 特徴量計算は物体検出や画像分類などの画像処理アプリケーションで行われる処理であり，高負荷な処理例として選択した．FPGA に実装する HOG 特徴量計算回路は文献 [47] を参考に開発し，Register transfer level（RTL）と，Intel FPGA SDK for OpenCL[48] を用いた高位合成（High-Level Synthesis，HLS）で設計した．HLS 設計は CPU と FPGA を用いたシステムの開発に適しており，抽象度の高い設計により短期間で開発が可能である．また，Intel FPGA SDK for OpenCL は CPU と FPGA 間で通信するための API を提供している．これらの理由で SW-HW 構成の機能における HOG 特徴量計算回路を HLS 設計で開発した．

RTL 設計と HLS 設計によって開発した HOG 特徴量計算回路の FPGA リソース使用量を表 3.2 に示す．各回路の FPGA リソース使用量は Intel Quartus Prime Design Software[49] を用いて推定した．RTL 設計の回路は HLS 設計の回路と比べて約 2 倍のロジックリソースと DSP リソースを使用するが，メモリリソースを使用しない．これは設計者が手動で回路をチューニングした結果である．一方，HLS 設計の回路は HLS コンパイラを用いて自動チューニングした結果，すべての種類の FPGA リソースを使用する．

次に，RTL 設計によって開発した IPB インタフェースの FPGA リソース使用量を表 3.3 に示す．この回路のロジックリソース使用率は FPGA 全体の 1% 未満であり，メモリリソースの使用率は約 20% である．

表 3.2 HOG 特徴量計算回路の FPGA リソース使用量（括弧内は Arria 10 GX 1150 FPGA に対する使用率）

Design technology	ALMs	M20Ks	DSPs
RTL	99910 (23.39%)	0 (0.00%)	192 (12.65%)
HLS	59730 (13.98%)	152 (5.60%)	80 (5.27%)

表 3.3 IPB インタフェースの FPGA リソース使用量（括弧内は Arria 10 GX 1150 FPGA に対する使用率）

Modules	ALMs	M20Ks	DSPs
Real-time packet reordering circuit + Packet parser	1627	512	0
Packet generator	475	13	0
Total	2102 (0.49%)	525 (19.35%)	0 (0.00%)

3.5.2 評価結果

処理性能評価

各構成において，パケット順序制御の開始から画像データが画像処理プログラムもしくは回路に到着するまでのデータ転送遅延を測定した．実験では，CPU もしくは FPGA に実装した画像データ送信機能から 800 個のパケットに分割した VGA 画像（640×480 pixels）を入力した．

各構成における平均データ転送遅延を図 3.12 に示す．これらの結果は 1000 回測定したときの平均値であり，パケット順序制御による遅延と CPU-FPGA 間の通信時間，システムコールなどのオーバヘッドを含む．また，エラーバーは測定したデータ転送遅延の範囲を示している．

SW-SW 構成と SW-HW 構成における平均データ転送遅延はそれぞれ 0.64 ms と 5.06

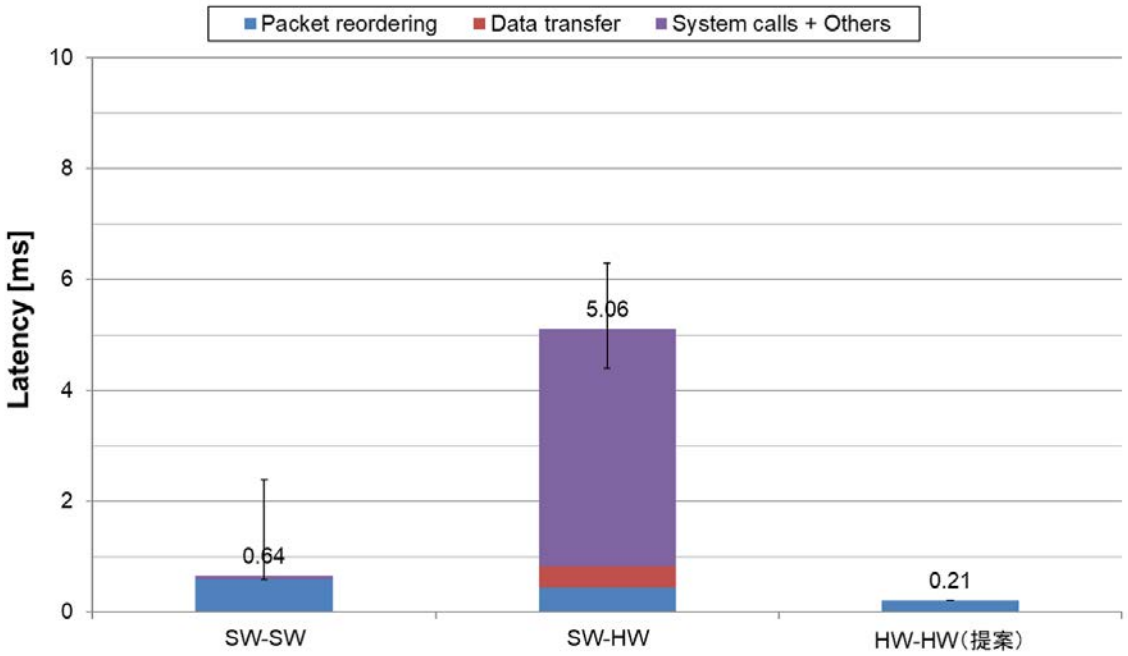
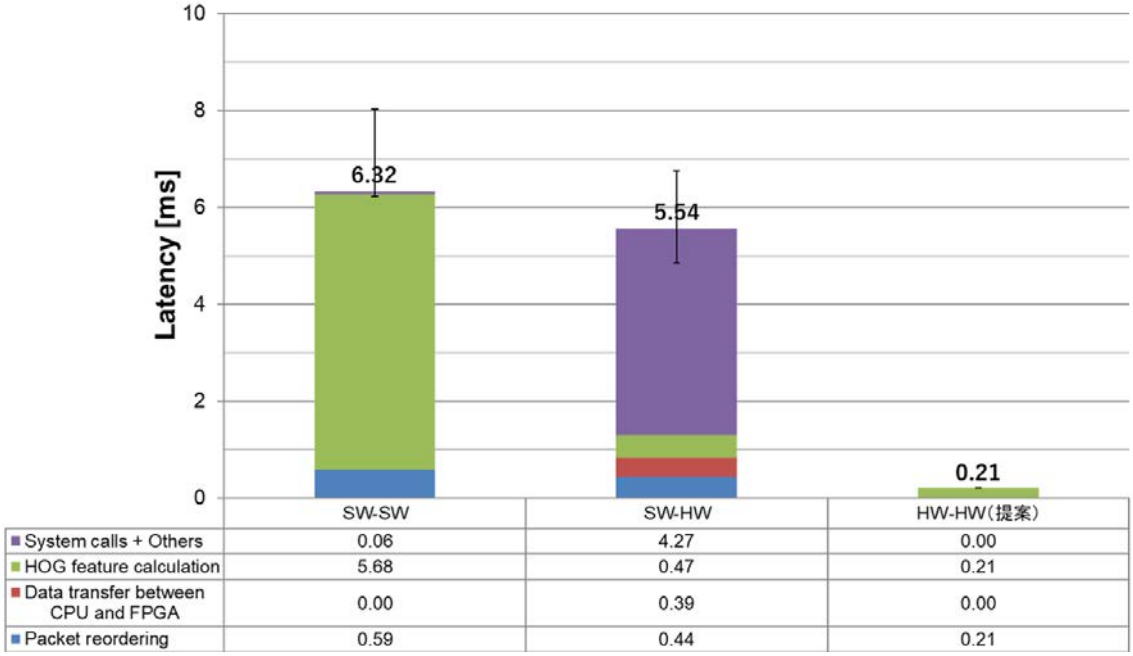


図 3.12　HOG 特徴量計算機能の平均データ転送遅延

ms であった．SW-HW 構成では CPU と FPGA 間の通信に時間がかかったため，より大きなデータ転送遅延が発生した．また，これらの構成ではソフトウェアのパケット順序制御により大きな遅延が発生した．一方，HW-HW 構成では，IPB インタフェースが正しい順序のパケットをすぐに HOG 特徴量計算回路に転送したことで，平均データ転送遅延は 0.21 ms であった．これらの結果から，HW-HW 構成は，SW-SW 構成と SW-HW 構成と比べて約 3 分の 1 と 24 分の 1 の遅延で画像データを HOG 特徴量計算回路に転送できることを確認した．

次に，パケット順序制御の開始から HOG 特徴量の計算完了までの機能全体の処理遅延を測定した．測定結果を図 3.13 に示す．これらの結果は 1000 回測定したときの平均値である．また，エラーバーは測定した処理遅延の範囲を示している．SW-SW 構成は HOG 特徴量の計算に時間がかかり，平均処理遅延は 6.32 ms だった．一方，SW-HW 構成と HW-HW 構成はより短い時間で HOG 特徴量を計算した．そのため，SW-HW 構成の平均処理遅延は 5.54 ms であり，SW-SW 構成より小さくなった．また，HW-HW 構成はパケット順序制御と画像処理が並列に実行されたことで，平均処理遅延は 0.21 ms だった．これらの結果から，HW-HW 構成は，SW-SW 構成と SW-HW 構成と比べて約 30 分の 1 と 26 分の 1 の遅延で処理を行えることを確認した．



	SW-SW	SW-HW	HW-HW(提案)
System calls + Others	0.06	4.27	0.00
HOG feature calculation	5.68	0.47	0.21
Data transfer between CPU and FPGA	0.00	0.39	0.00
Packet reordering	0.59	0.44	0.21

図 3.13　HOG 特徴量計算機能の平均処理遅延

一般的に，計算量の多い画像処理は FPGA にオフロードすることで大幅に高速化できる．しかし，HOG 特徴量の計算時間が短縮されたにも関わらず，SW-HW 構成では大きな処理遅延が発生した．その要因は CPU と FPGA 間の通信オーバヘッドである．通信オーバヘッドはテクノロジによって変化する．そこで，SW-HW 構成で通信オーバヘッドが全く発生しない場合を考える．その場合，図 3.13 の結果から処理遅延は 1.30 ms である．よって，HW-HW 構成の平均処理遅延よりも大きい．これはパケット順序制御と画像処理が逐次実行されたためである．一方，HW-HW 構成はパケット順序制御と画像処理の間のパイプライン処理により，短い時間で処理を行えた．

これらの実験では，1 つの CPU コアを用いてパケット順序制御プログラムを動作させた．したがって，複数の CPU コアを使用することでソフトウェア処理の時間を短縮し，機能全体の処理遅延を減らせる可能性がある．しかし，パケット順序制御に割り当てる CPU リソースを増やすことで画像処理プログラムに悪影響を与える可能性がある．また，複数の CPU コアを使用する並列処理はコア間でデータ移動を行うため，ソフトウェアのオーバヘッドが増加する可能性がある．

HOG 特徴量計算機能のユースケースとして，映像監視サービスを考える．例えば，60 fps のネットワークカメラを入力ソースとする映像監視サービスでは，送られてきた画像

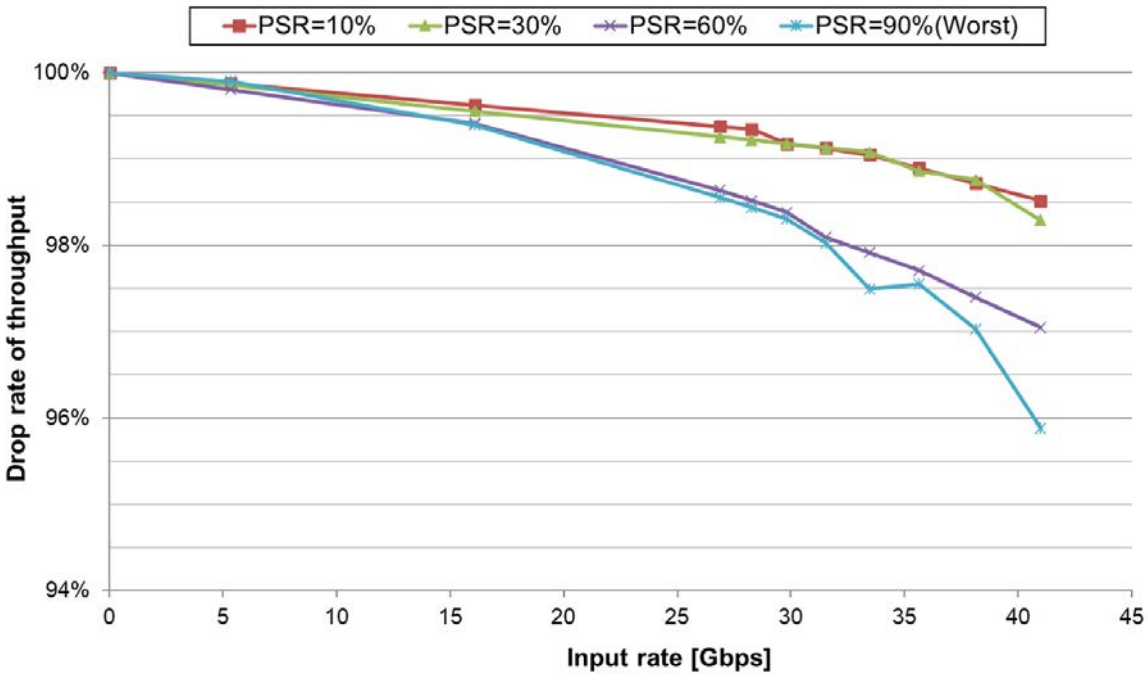


図 3.14　HW-HW 構成の HOG 特徴量計算機能のスループット低下率

データを 16 ms 以内に処理する必要がある．図 3.13 の結果はいずれの機能も 16 ms 未満の遅延で VGA 画像を処理できることを示している．しかし，より大きな画像データに対しては処理が間に合わない可能性がある．例えば，4K 画像（4096 × 2160 pixels）の処理量は VGA 画像の処理量の 28.8 倍であるため，SW-SW 構成と SW-HW 構成は処理が間に合わない．一方，HW-HW 構成は要求された遅延未満で処理が可能である．

パケット順序逆転によるスループット低下率の評価

リアルタイムパケット順序制御回路は誤った順序のパケットをバッファに格納するため，パケット順序逆転が機能のスループットに影響する．そこで，パケット順序逆転率と入力レートを変えて HW-HW 構成の HOG 特徴量計算機能のスループットを測定した．画像の送信間隔はパケットの送信間隔よりもはるかに長いため，パケットの順序逆転は画像内および近くのパケット間でランダムに発生すると仮定した．パケット順序逆転率（Packet Swap Rate，PSR）をパケットの総数に対する誤った順序のパケット数の比率と定義する．

図 3.14 に入力レートと PSR を変えてスループット低下率を評価した結果を示す．この結果は入力レートおよび PSR が高くなると，システムのスループットがわずかに低下することを示している．例えば，10Gbps の転送速度の Lagopus から PSR が 90% のパケット

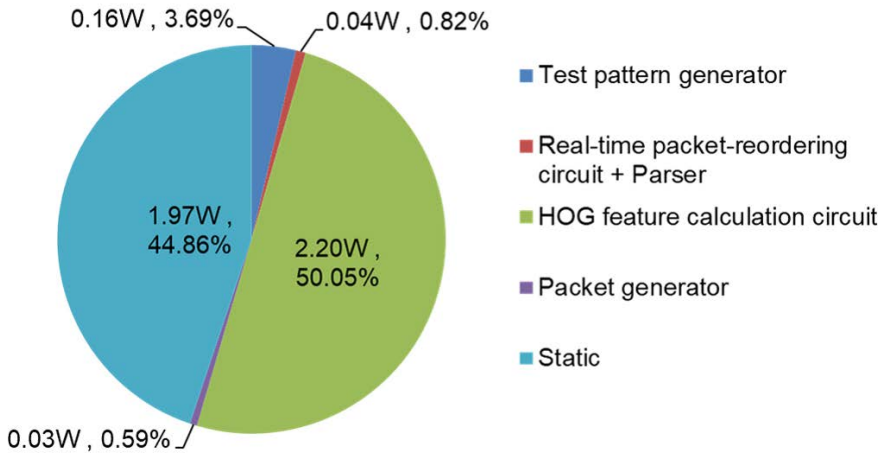


図 3.15　IPB インタフェースおよび HOG 特徴量計算回路の消費電力

を受信する場合，このシステムのスループット低下率は 0.5% 未満である．また，Lagopus と FPGA NIC の組み合わせで実現できる 40Gbps の入力レートの場合，PSR が 90% の最悪の条件でスループット低下率は 3.7% である．よって，ほとんどのパケットが入れ替わる状況でも HW-HW 構成では高速に画像処理を行える．

消費電力の評価

HW-HW 構成の HOG 特徴量計算機能の消費電力を PowerPlay Early Power Estimator を用いて推定した．図 3.15 にその結果を示す．この図は，画像データ送信回路を除く機能全体の消費電力が 4.24W であることを示している．また，IPB インタフェース（リアルタイムパケット順序制御回路とパケットパーサ，パケットジェネレータ）の消費電力は高々 0.07W である．一方，1 つの CPU コアで実行した SW-SW 構成の機能の消費電力は平均 6.70W だった．これらの結果から，HOG 特徴量計算機能を FPGA にオフロードすることで，消費電力が 37% 削減されることを確認した．

以上の評価結果から，提案アーキテクチャにより HOG 特徴量計算機能が低遅延かつ低消費電力で動作することを確認した．

第 4 章

Approximate Completion-Detection 方式の可変レイテンシ回路の提案

本章では，回路のアイドル時間を減らし高速処理を達成するために，組み合わせ回路におけるタスクの完了を大まかに検出し，各タスクの処理時間を変更する ACD 方式の可変レイテンシ回路 [50] を提案する．実験では，設計した ACD 方式の画像処理回路の処理性能と回路面積，消費電力を評価した．

4.1 開発の動機

画像処理サービスなどの高度な情報処理サービスは，高負荷な処理を高速に行うための高性能なハードウェア（デジタル回路）を必要とする．例えば，映像監視サービスでは複数の防犯カメラの映像をリアルタイムに監視するために，高速な画像処理回路が必要である．いくつかの画像処理サービスは，必ずしも正確な計算を必要としておらず，サービス品質を低下させない計算エラーを許容できる．本研究では，計算エラーを許容するアプリケーションの高速化に焦点を当てる．

現在主流のデジタル回路はクロック信号に同期して動作する同期回路である．これまで同期回路の高速化にはプロセスの微細化が大きく貢献してきたが，近年ムーアの法則 [51] が当てはまらなくなった．これは，微細化による処理性能の改善が困難になったことを意味する．さらに，最先端テクノロジノードでは様々なタイプの遅延変動が回路性能に悪影響を与えている．例えば，フリップフロップ（Flip-Flop，FF）間で発生する最大遅延の差

や入力パターンに応じた動的遅延の違いにより，組み合わせ回路におけるプリミティブな処理にアイドル時間が発生する．これらの遅延は単一の固定クロックサイクルで動作する一般的な同期回路で問題になる．そのため，関連研究はこれらの遅延を削減して回路の高速化を達成した．

文献 [52, 53] は FF 間の最大遅延を可能な限り削減することで，FF 間の最大遅延の差によって生じるアイドル時間を短縮した．しかし，すべての信号伝搬経路の最大遅延を等しくすることは困難であり，最適化された回路でもその差は残る．この問題を解決するため，文献 [54, 55] は一般同期回路を提案した．一般同期回路は各 FF へのクロック供給タイミングを変えることで最大遅延の差を完全になくし，普通の同期回路よりも高い性能を達成した．文献 [56, 57] は信号伝搬経路の最小遅延を大きくすることでクロックスケジューリングの柔軟性を高め，一般同期回路のクロック周期をさらに短縮した．これらの研究のおかげで，FF 間の最大遅延の差によるアイドル時間は十分に短縮された．

アプロキシメートコンピューティング手法を導入した回路 [58, 59, 60] は，計算エラーを許容した回路構成により動的遅延の違いによって生じるアイドル時間を短縮する．この手法は，例えば高い信頼性と精度を必要としない画像処理アプリケーションを高速化するのに効果的である．しかし，サービス品質を低下させないようにエラー率を制御しながら性能目標を達成する回路を設計することは困難であり，この手法は少数のアプリケーションにのみ使用される．この手法のもう 1 つの問題は，ユースケースごとの開発に時間がかかることである．これらの問題により，高性能サービスを迅速に提供することが難しい．

タイミングエラー検出回復（EDC）方式 [61, 62, 63] は，正しい計算を保証しながら最大遅延未満のクロック周期での回路動作を可能にする．EDC 方式による可変レイテンシ回路はタイミングエラーを監視しながら，高速クロックを用いて投機的実行を行う．タイミングエラーを検出すると，通常処理を中断して対応する処理の出力を複数のクロックサイクルで修正する．したがって，この回路はほとんどの処理で投機的実行に成功すると高速に動作する．例えば，文献 [64, 65] は EDC 方式の加算器が高速に動作することを示した．EDC 方式の問題はタイミングエラーを適切に検出して修正するための厳しい設計制約 [66, 67] があり，処理性能の改善が制限されることである．文献 [68] と [69] はタイミングエラーを予測することで可変レイテンシ回路を実現した．前者はカナリア FF[70] を用いてサブスレッショルド回路のエネルギー効率を改善した．後者はクリティカルパスの一部を監視し，タイミングエラーの兆候が検出されたときに回路動作を変更することで高速処理を実現した．これらの方法は動的遅延の影響を軽減できるが，回路規模が大きくなる可能性がある．また，タイミングエラーを回避するための厳しい設計制約がある．

本研究では可変レイテンシ回路による処理性能の改善を検討した．我々が提案する ACD

方式はプリミティブな処理の完了を大まかに検出して動作を変えることで，動的遅延の違いによるアイドル時間を短縮する．ACD 方式の回路は誤った完了検出のために誤った計算を行う可能性があるが，正しい計算を保証する回路よりも高速に動作する．エラー率は制御可能であり，それを最小化するための回路設計手法も開発した．

4.2　ACD 方式

本稿では単一の固定クロックサイクルで動作する回路を**固定レイテンシ回路**と呼ぶ．一方，動的遅延に応じて動作が変化する回路を**可変レイテンシ回路**と呼ぶ．ACD 方式による可変レイテンシ回路は **Approximate Completion-Detection 機構（ACDM）**を用いて実現する．ACDM は組み合わせ回路の出力が一定時間変化しないと処理が完了したと見なす．そのためにこの回路はメインクロックよりも高速なサブクロックを用いて組み合わせ回路の出力を監視する．ACDM を使用することで動的遅延に応じて処理の完了が迅速に検出される．本節では，ACD 方式による可変レイテンシ回路の動作と性能推定について説明する．また，関連研究の EDC 方式について説明する．

4.2.1　回路動作

ACD 方式による可変レイテンシ回路の動作を図 4.1 に示す．この回路は組み合わせ回路の出力（ロジック出力）が変化しない間，ACDM でサブクロックごとにカウントを行う．そして，入力 FF と出力 FF にメインクロックが入力されたとき，ACDM のカウント値が事前に設定した閾値と等しいと次の処理を開始する．この回路は動的遅延の違いを効果的に利用するために最大遅延よりも短い周期のメインクロックを必要とする．また，メインクロック周期よりも短いサブクロック周期を用いて処理の完了を迅速に判断する．

ACDM は組み合わせ回路の出力変化を検出するために，ロジック出力をサブクロックサイクルごとに FF に保持し，直近のロジック出力と比較する．両方が同じ値であれば，サブクロックごとのカウントを行う．具体的には，両方の値が同じであればローレベル，異なっていればハイレベルの Diff 信号を生成し，Diff 信号がローレベルの間カウントを行う．カウント値が閾値に達するとオーバーフローによる誤動作を防ぐためにカウントを止める．閾値は回路設計によって決める．閾値の決め方については 4.3 節で説明する．一方，出力が変化するとハイレベルの Diff 信号によりカウント値をゼロにリセットする．これにより，ACDM は組み合わせ回路の出力が一定時間連続して変化しない場合のみ，処理が完了したと判断する．

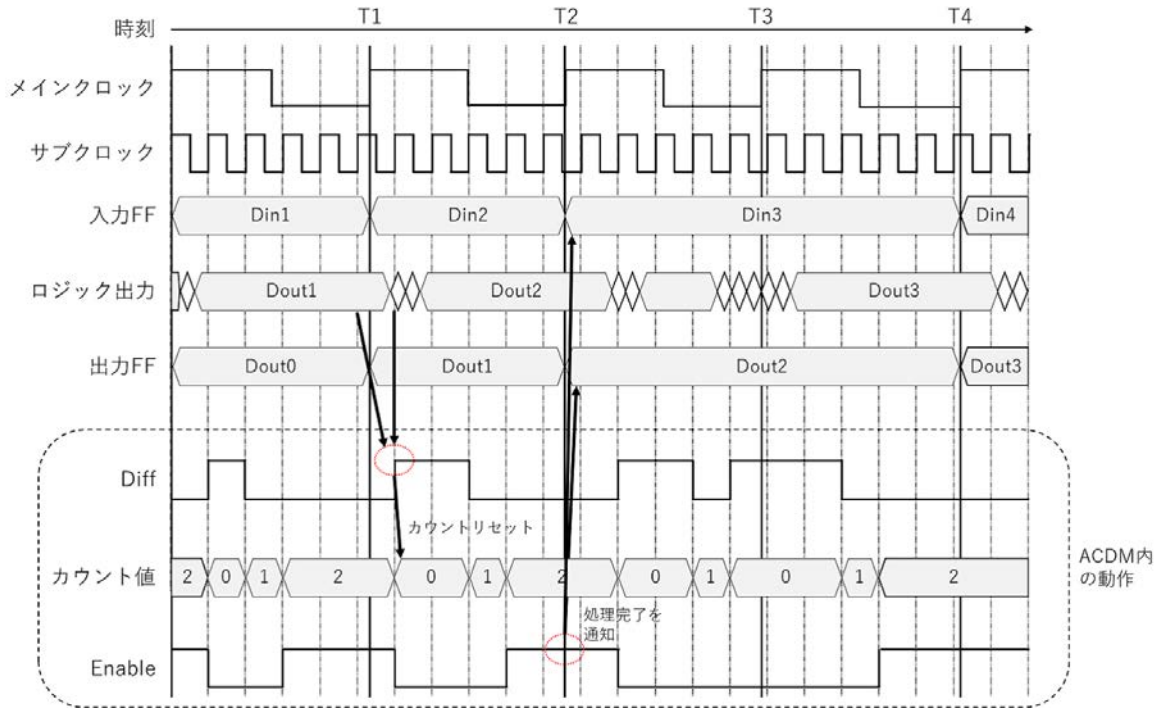


図 4.1 ACD 方式による可変レイテンシ回路の動作例．この回路の ACDM は同じロジック出力を 2 回続けて取得すると処理の完了を検出する．

ACDM は Enable 信号を使って判断を各 FF に通知する．Enable 信号がハイレベルの場合，入力 FF と出力 FF は前段のロジック出力をメインクロックの立ち上がり時に保持する．それ以外の場合，各 FF は以前のデータを保持し続ける．したがって，処理時間がメインクロック周期より短い場合に 1 サイクルで処理が完了する．一方，メインクロック周期より処理時間が長いと処理に数サイクルかかる．

次に，図 4.1 を用いて ACD 方式による可変レイテンシ回路の具体的な動作を説明する．この回路の ACDM は同じロジック出力を 2 回続けて取得すると処理の完了を検出する．時刻 T1，T2 および T4 では，カウント値が 2 であるため，イネーブル信号がハイレベルである．よって，入力 FF と出力 FF はメインクロックの立ち上がり時に前段のロジック出力を保持する．一方，時刻 T3 では，ロジック出力の変化によりカウント値がリセットされたことで，イネーブル信号がローレベルである．そのため，各 FF は以前のデータ（Din3，Dout2）を保持し続ける．以上の動作により，ACD 方式の回路は処理完了を判断する少しのオーバヘッドで次の処理を開始できる．

ACDM が一時的に安定したロジック出力に対して誤って処理完了を判断すると，誤った

計算結果が出力される可能性がある．また，処理完了の判断がメインクロックの立ち上がりに間に合わないと，回路は同じ入力に対する処理を継続し，無駄なアイドル時間が生じる．これらの望ましくない状況は設計で減らす必要がある．

4.2.2　実効クロック周期

可変レイテンシ回路は，通常，最大遅延よりも周期が短いクロックを使用するため，処理に複数のクロックサイクルがかかる場合がある．つまり，処理にかかる時間がクロック周期と異なる．本稿では，可変レイテンシ回路の実効的な速度を**実効クロック周期**と呼ぶ．ACD 方式による高速な可変レイテンシ回路を設計するには，実効クロック周期を推定する必要がある．そこで，実効クロック周期の計算方法を検討した．本項では，実効クロック周期を推定するための計算式について説明する．

ACD 方式の回路に N 個のデータを入力したときの実効クロック周期を考える．この回路はすべての ACDM が処理の完了を判断するまで次の処理を開始しない．したがって，処理に必要なクロックサイクル数は最も大きな動的遅延に依存する．$w(i)$ を i 番目（$1 \leqq i \leqq N$）の処理における最も大きな動的遅延とする．ACDM は処理の完了を判断するのに一定の時間を必要とする．このオーバヘッド時間は，ロジック出力をチェックする回数を C_{th}，サブクロック周期を T_{sub} とすると，$C_{\mathrm{th}} \times T_{\mathrm{sub}}$ で表すことができる．したがって，処理の開始から正しい完了判断が行われるまでの経過時間は次の式で表される．

$$t_{\mathrm{proc}}(i) = \left\lceil \frac{w(i)}{T_{\mathrm{sub}}} \right\rceil \times T_{\mathrm{sub}} + C_{\mathrm{th}} \times T_{\mathrm{sub}} \tag{4.1}$$

ここで，$t_{\mathrm{proc}}(i)$ は i 番目の処理の経過時間である．式 4.1 の右辺の第一項は，ACDM が安定したロジック出力のチェックを開始するまでの時間である．ロジック出力はメインクロックの立ち上がり時に FF に保持されるため，処理に必要なクロックサイクル数は $t_{\mathrm{proc}}(i)$ を用いて次の式で表される．

$$N_{\mathrm{cycle}}(i) = \left\lceil \frac{t_{\mathrm{proc}}(i)}{T_{\mathrm{main}}} \right\rceil \tag{4.2}$$

ここで，$N_{\mathrm{cycle}}(i)$ は i 番目の入力を処理するために必要なクロックサイクル数を表し，T_{main} はメインクロック周期を表す．以上から実効クロック周期 T_{eff} は次の式で推定できる．

$$T_{\mathrm{eff}} = \frac{1}{N} \times \sum_{i=1}^{N} N_{\mathrm{cycle}}(i) \times T_{\mathrm{main}} \tag{4.3}$$

すべての ACDM がメインクロック周期よりも短い時間で処理の完了を判断すると，回路は 1 周期のメインクロックサイクルで動作し，それ以外の場合は複数周期のメインクロックサイクルで動作する．回路が 2 周期以上のメインクロックサイクルで動作する場合，固定レイテンシ回路よりも処理時間が長くなる可能性がある．しかし，大部分の入力を最大遅延未満で処理することで，N 個の入力に対する合計処理時間は短くなる．ACD 方式の回路の実効クロック周期を短くするには，C_{th}，T_{main} および T_{sub} のパラメータを適切に設定する必要がある．パラメータ設定については 4.3 節で詳しく説明する．

4.2.3 関連研究

EDC 方式 [71] は FF の一部を投機 FF(Speculative FF, S-FF) に置換し，エラー検出回復機構を付加することで可変レイテンシ回路を実現する．そのため，回路規模は大きくなるが幅広い回路に適用できる．例えば，文献 [72, 73] では一般同期回路への EDC 方式の導入が検討されており，文献 [74, 75] では FPGA 上で EDC 方式の回路を実現した．文献 [76, 77] では EDC 方式を用いた回路でのアプロキシメートコンピューティングが検討されている．

EDC 方式では，エラー検出回復機構により回路動作の正当性が保証される．そのため，FF 間の最大遅延よりも小さいクロック周期のもとで回路を投機的に動作させることが可能となる．この動作により，通常処理における処理時間は削減される．一方，タイミングエラーが発生すると回復処理を行うため時間損失が生じる．ゆえに，速度性能はクロック周期低減による処理時間の低減が回復処理による時間損失を上回る場合において向上する．

EDC 方式による可変レイテンシ回路の動作を図 4.2 に示す．この回路の処理は通常処理と回復処理の 2 つに分類される．通常処理では，最大遅延未満のクロック周期のもとで回路は投機的な動作を行う．このとき，回路は絶えずタイミングエラーの発生を監視する．タイミングエラーが発生した場合，すなわち Error 信号がハイレベルになった場合，通常処理を中断して外部に出力する信号を正当な信号に修正する回復処理を行う．このとき，外部回路では回復処理に対応した適切な動作を行う．タイミングエラーが発生した次の周期の通常処理では，回路は前の入力に対する処理を再度行い，外部回路は訂正した値に対して処理を行う．

EDC 方式の設計制約

EDC 方式では FF 間の最大遅延に縛られない設計が可能となり，最大遅延による制約を回避できる．しかし，文献 [78] によると，設定クロック周期 T と，spFF と cfFF のクロッ

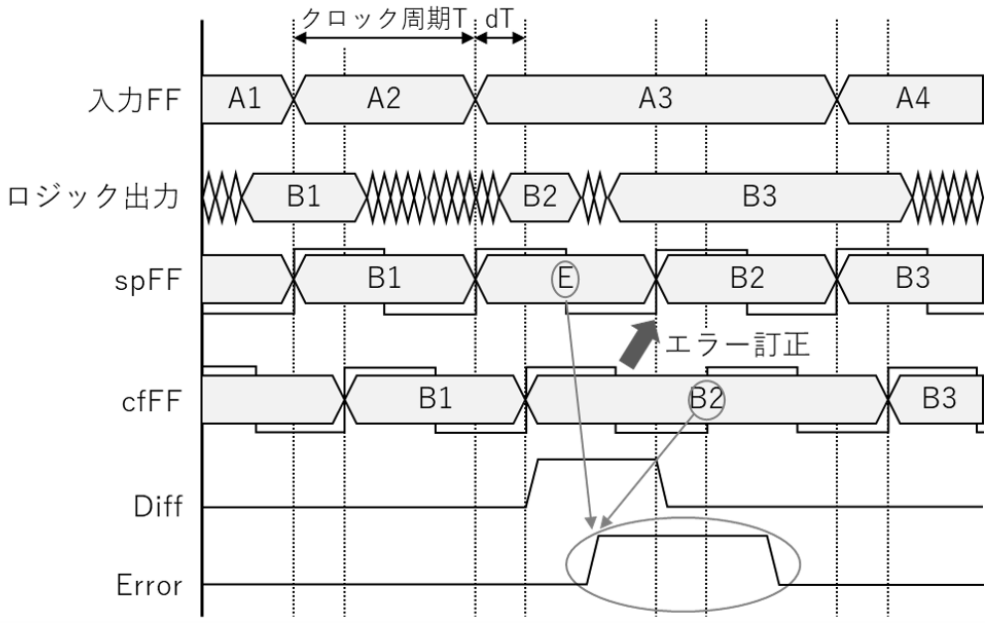


図 4.2　EDC 方式による可変レイテンシ回路の動作例．この回路は回復処理を 1 周期の間に終了する．

クタイミング差 dT に以下の設定許容範囲が存在する．

設定クロック周期制約

$$\begin{aligned} T \geqq \max(& \max_{(a,b)\in P_{\mathrm{hs}} \wedge b \in F_{\mathrm{nr}}} w_{\max}(a,b), \\ & \max_{(a,b)\in P_{\mathrm{hs}} \wedge b \in F_{\mathrm{cf}}} (w_{\max}(a,b) - w_{\min}(a,b)), \\ & \max_{(p,q)\in P_{\mathrm{cor}}} w_{\mathrm{cor}}(p,q)) \end{aligned} \tag{4.4}$$

クロックタイミング差制約

$$\begin{aligned} & \max_{(a,b)\in P_{\mathrm{hs}} \wedge b \in F_{\mathrm{cf}}} (w_{\max}(a,b) - T) \leqq dT \leqq \\ & \min(\min_{(a,b)\in P_{\mathrm{hs}} \wedge b \in F_{\mathrm{cf}}} w_{\min}(a,b), \\ & \min_{(p,q)\in P_{\mathrm{cor}} \wedge p \in F_{\mathrm{cf}}} (T - w_{\mathrm{cor}}(p,q))) \end{aligned} \tag{4.5}$$

ここで，$w_{\max}$ と $w_{\min}$ は FF 間の最大遅延と最小遅延を表し，w_{cor} は回復処理に要する信号伝搬遅延を表す．式 4.4 は設定クロック周期の下限が FF 間の最大遅延と最小遅延の差によることを示している．また，式 4.5 はクロックタイミング差を最も小さな最小遅延以下にしなければならないことを示している．これらの制約により，EDC 方式による高速化は制限される．

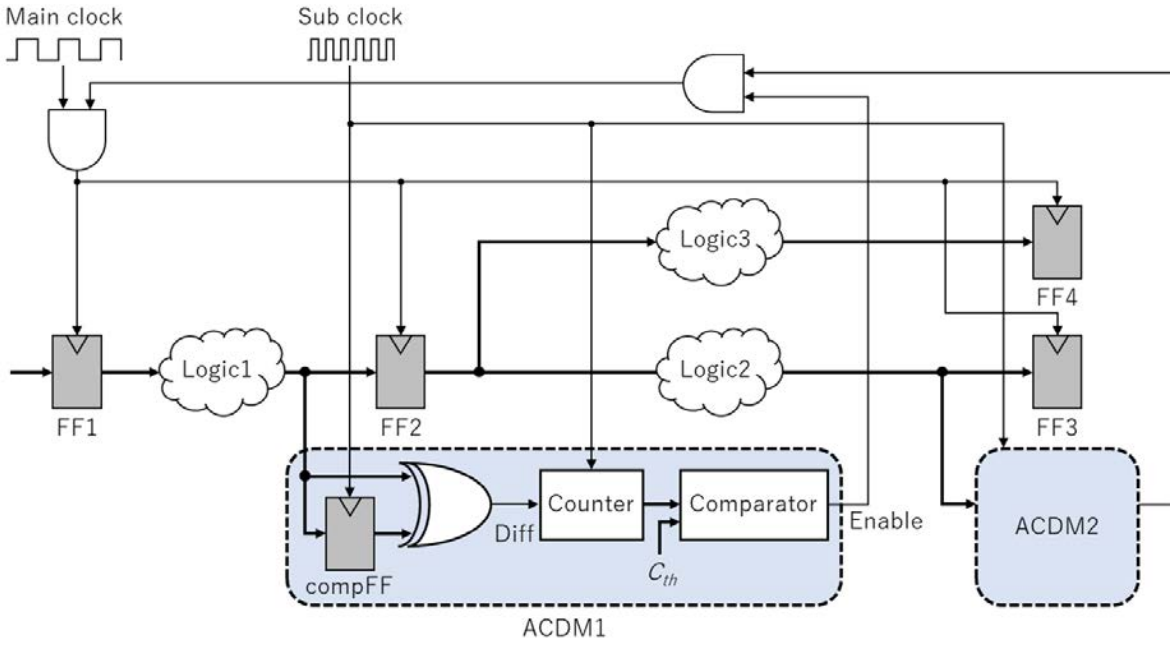


図 4.3　ACDM を用いた回路の構成

4.3　ACDM を用いた可変レイテンシ回路

ACD 方式による可変レイテンシ回路は，一般的な同期回路に ACDM を追加することで実現できる．本節では，ACDM を用いた回路構成と設計手法について説明する．

4.3.1　回路構成

ACDM を用いた回路構成例を図 4.3 に示す．この回路は組み合わせ回路（Logic1，Logic2，Logic3），フリップフロップ（FF1，FF2，FF3，FF4）および ACDM（ACDM1，ACDM2）で構成されている．ACDM は Logic1 と Logic2 の出力を定期的にチェックし，出力が一定時間変化しない場合に処理完了を判断する．具体的には，ACDM は XOR ゲートを使用して組み合わせ回路の出力を compFF の出力と比較し，Diff 信号を生成する．Diff 信号がローレベルの場合，Counter はサブクロックが入力される度，カウント値をインクリメントする．一方，Diff 信号がハイレベルであればカウント値を 0 にリセットする．Comparator はカウント値を閾値である C_{th} と比較する．カウント値が C_{th} と等しくなると，Comparator は Enable 信号をハイレベルにして，処理の完了を各 FF に通知する．正

確には，AND ゲートツリーによって集約された Enable 信号がメインクロックの立ち上がりタイミングを制御する．いずれかの ACDM の Enable 信号がローレベルである場合，メインクロックはハイレベルにならない．したがって，各 FF は以前のデータを保持し続ける．以上の動作により，すべての ACDM が処理の完了を検出したときに次の処理を開始できる．

複数の ACDM を用いる回路は，複雑な AND ゲートツリーの処理遅延により Enable 信号が各 FF に届くまでに時間がかかる．処理完了の判断が FF に届くのが遅れた場合，処理に余計なメインクロックサイクルが必要になり，実効クロック周期が悪化する．この余計なメインクロックサイクルは，ACDM をボトルネックになっている FF 間にのみ追加することで削減できる．回路面積の増加を抑える観点からも，少数の ACDM を用いて大きな動的遅延が生じる FF 間のみ監視することが望ましい．さらに，監視する信号線の数を減らすことで ACDM の回路規模を小さくできる．

4.3.2　設計手法

ACDM を用いた回路の設計手法を説明する．図 4.4 は設計フローの概要を示している．この設計フローでは従来の同期回路設計フローに ACDM 追加プロセスが追加される．この設計手法の手順は以下のとおりである．まず VHDL や Verilog，SystemVerilog などのハードウェア記述言語（HDL）を用いてソースファイルを作成する．次に，ソースファイルに対して論理合成を行い，ネットリストを生成する．ACDM 追加プロセスは，ネットリストが入力されると ACDM を含むネットリストを生成し，最適なパラメータを提案する．このプロセスは ACDM の追加先を選択するステップ，T_{eff} を推定するステップ，およびシミュレーションにより実効クロック周期とエラー率を評価するステップで構成される．最後のレイアウトプロセスは ACDM を含むネットリストをもとにコンポーネントを対象のデバイスに配置し，コンポーネント間を配線する．この設計フローの大部分のプロセスは従来と同じである．よって，設計者は既存ツールを使用することができ，設計スキルとノウハウを活かした回路設計を行える．

ACDM 追加プロセスの詳細フローを図 4.5 に示す．このプロセスは論理合成プロセスからネットリスト，設計者から制約条件（性能目標と T_{sub}）を受け取ると，いくつかのステップを経て，ACDM を含むネットリストを生成し，最適な T_{main} と C_{th} を提案する．T_{sub} はテクノロジノードに応じて設計者が決める．高速サブクロックを使用すると ACDM のオーバヘッド時間を短縮できるが，誤判定の可能性が増す．また，判定時間が FF 間の最小遅延より短いと，ACDM は完了した処理の出力をチェックすることで誤った判定を行う．

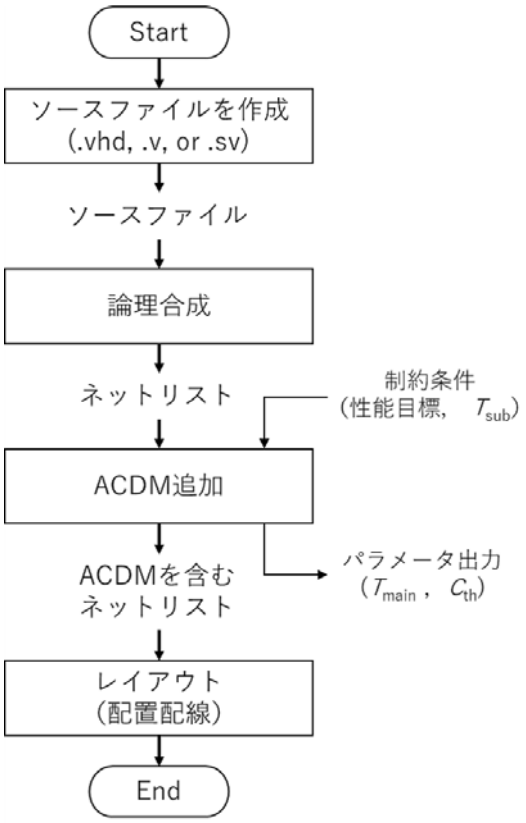


図 4.4　ACDM を用いた回路の設計フロー概要

したがって，T_{sub} は少なくとも FF 間の最小遅延より大きくする必要がある．

このプロセスは，始めにネットリストをもとに回路の動的遅延を解析し，遅延分布を取得する．この解析は文献 [79] の推定手法を用いると高速に行える．そして，遅延分布から ACDM の追加先を選ぶ．具体的には，動的遅延の最大値が目標とする実効クロック周期を超える FF 間を選択する．続いて，遅延分布，T_{main} および C_{th} を用いて式 4.1 から 4.3 により T_{eff} を計算する．T_{eff} が目標の実効クロック周期より大きかった場合，候補が残っていれば T_{main} と C_{th} を変えて ACDM の追加先を選び直す．それ以外の場合，論理シミュレーションにより回路の実効クロック周期とエラー率を評価する．エラー率 P_{err} は次の式を使用して計算する．

$$P_{err} = \frac{N_{err}}{N_{sample}} \times 100 \tag{4.6}$$

ここで，N_{err} は計算を誤った回数を表し，N_{sample} は入力の総数を表す．

シミュレーションの結果，目標の実効クロック周期が達成されない場合，候補が残って

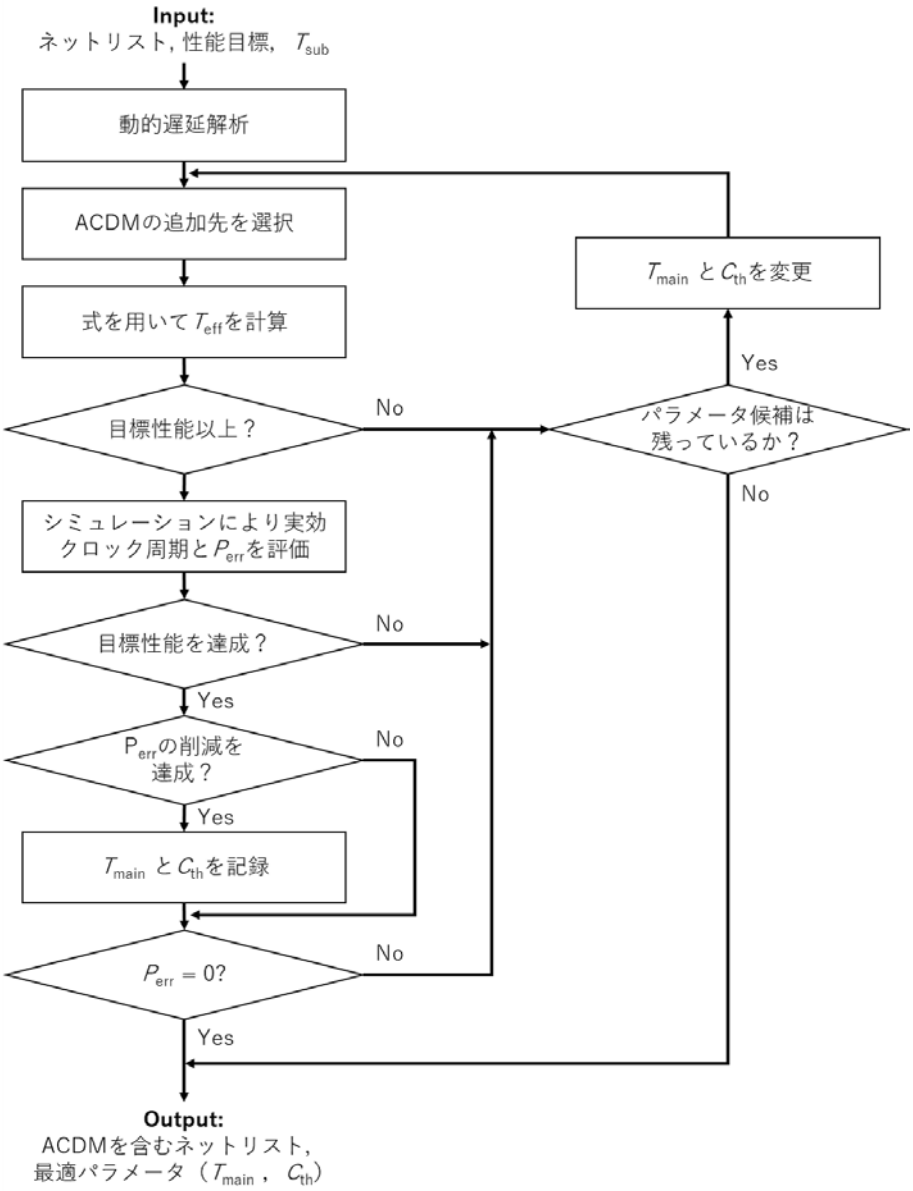


図 4.5　ACDM 追加プロセスの詳細フロー

いれば T_{main} と C_{th} を変えて ACDM の追加先を選ぶ処理に戻る．これは T_{eff} が最低限の条件で推定されるためにおこる可能性がある．それ以外の場合，P_{err} が削減されていれば T_{main} と C_{th} を記録する．ACDM 追加プロセスは P_{err} が 0 になったとき，もしくは候補となる T_{main} と C_{th} がなくなったときに終了する．後者の場合，プロセスは記録した次善のパラメータセットを出力する．

実効クロック周期は小さな T_{main} と C_{th} を用いることで短くなる．一方，これらのパラ

メータを用いた回路は ACDM の誤った完了判断に従う可能性が高くなるため，エラー率が高くなる．よって，ACDM 追加プロセスは低いエラー率と短い実効クロック周期を達成するために，小さなパラメータから評価するのがよい．

回路のすべてのデータパスに対してすべての入力パターンを評価することは，膨大な時間が必要なため現実的ではない．そのため，ACDM 追加プロセスはサンプルデータを用いた論理シミュレーションにより，評価時間を短縮する．この方法では正確な実効クロック周期とエラー率は得られないが，サンプルデータの数を増やすことで信頼性を向上できる．論理シミュレーションにはランダムデータまたはアプリケーションデータを使用することを想定している．特にアプリケーションデータを用いると，偏ったデータを処理する回路の実効クロック周期とエラー率を高精度で評価できる．例えば，画像データには類似した色のピクセルが複数含まれる．画像処理回路がそれらのピクセルを連続して処理する場合，信号がほぼ同じ経路を通るため小さな動的遅延が発生する．したがって，ACD 方式の画像処理回路はその入力パターンを高速に処理することで，実効クロック周期が短くなる．また，ACDM の誤検出が減るため，エラー率が低くなる．

実効クロック周期の評価精度と設計時間の間にはトレードオフがある．大量のサンプルデータを用いた論理シミュレーションにより評価精度は向上するが，評価時間は長くなる．したがって，論理シミュレーションを繰り返して最小のエラー率を得るのに時間がかかる．パラメータの微調整はより適切なパラメータを見つけるのに役立つ場合があるが，設計時間が長くなる．したがって，必要な精度とローンチまでの時間を考慮して，T_{main} と C_{th} の範囲と分解能を制限する必要がある．

4.4 性能評価

ACD 方式により回路が高速に動作することを確認するため，ACD 方式を導入した桁上げ伝搬加算器（Ripple Carry Adder, RCA），離散コサイン変換回路および HOG 特徴量計算回路の実効クロック周期とエラー率を評価した．複数の回路を評価することで，ACD 方式が異なるタイプの回路に有効であることを確認した．さらに，これらの回路の面積と電力を評価し，ACDM を追加するオーバヘッドを確認した．本節では，まず設計した回路について説明し，次に評価結果を示す．

4.4.1　準備

実験の準備として，RCA，2 次元離散コサイン変換（2D-DCT）回路および HOG 特徴量計算回路を設計した．RCA は幅広い回路に使用される基本モジュールである．2D-DCT 回路 [80] は画像圧縮や映像圧縮などのアプリケーションに，HOG 特徴計算回路は物体検出や画像分類などのアプリケーションに用いられる画像処理回路である．これらの回路を高速化する回路例として選択した．

設計した ACD 方式の回路を説明する．設計した ACDM のソースコードは付録 B.1 に記載する．ACD 方式の RCA は 2 つの 32 ビット入力を加算し，33 ビットの計算結果を出力する．この回路は C_{th} を 1 とする ACDM で計算結果を監視する．

ACD 方式の 2D-DCT 回路は画像データから複数の 8 × 8 係数行列を計算する．この回路は文献 [81] を参考に設計した．この回路は C_{th} を 1 とする ACDM を 8 個用いて組み合わせ回路から並列に出力した 40 ビットの係数を監視し，1 つまたは複数の ACDM が処理の完了を判断しない場合に現在の処理を継続する．

ACD 方式の HOG 特徴計算回路は，画像データから複数のヒストグラムを計算する．この回路は，輝度値計算回路，勾配強度計算回路および勾配方向計算回路を含む．これらの組み合わせ回路はそれぞれ，24 ビット，25 ビットおよび 3 ビットの計算結果を出力する．設計した HOG 特徴計算回路を図 4.3 に示す．Logic1，Logic2，Logic3 は，それぞれ輝度値計算回路，勾配強度計算回路，勾配方向計算回路に対応している．FF2，FF3，FF4 は，それぞれ 24 ビット FF，25 ビット FF，3 ビット FF である．Logic1（輝度値計算回路）と Logic2（勾配強度計算回路）を監視するために，C_{th} を 6 とする ACDM を追加した．一方，Logic3（勾配方向計算回路）は最大遅延が小さいため，ACDM による監視を行わなかった．

本研究は入力パターンに応じた動的遅延の違いに焦点を当てる．そのため，ROHM 社の 0.18 μm スタンダードセル・ライブラリを用いて組み合わせ回路を設計し，現実的な信号伝搬遅延を得た．このライブラリは大学大規模集積システム設計教育研究センター（VDEC）を通じて提供された．一方，他の影響を排除するために，理想的なクロックと FF を仮定し，メタステーブルの発生は考慮していない．

設計した回路における最も大きな最大遅延を表 4.1 に示す．ACD 方式の回路はこれらの最大遅延よりも短い周期のメインクロックを使って高速に動作する．また，ACDM はメインクロック周期より短いサブクロック周期を用いて処理の完了を判断する．実験では，各回路の ACDM に 0.20 ns 周期のサブクロックを入力した．このサブクロックにより，メインクロックの周期が短くても組み合わせ回路の出力をサイクル内で複数回チェックできる．

表 4.1 設計した回路における最も大きな最大遅延

回路	最大遅延 (ns)
RCA	3.84
2D-DCT	17.92
HOG 特徴量計算	13.60

比較対象として固定レイテンシの RCA，2D-DCT 回路，HOG 特徴量計算回路を設計した．さらに，固定レイテンシ RCA の FF を S-FF に置換し，エラー検出回復機構を追加して EDC 方式の RCA を設計した．付録 B.2 に設計した S-FF のソースコードを記載する．S-FF のパラメータであるクロックタイミング差は，4.2.3 項で説明した設計制約を考慮して 0.24 ns とした．この値は正常動作の範囲内で実効クロック周期を最大限改善するために選んだ．

4.4.2 評価結果

実効クロック周期とエラー率の評価

設計した回路の実効クロック周期とエラー率を論理シミュレーションによって評価した．論理シミュレーションは Synopsys 社の VCS シミュレータ [82] を用いて行った．

図 4.6 と図 4.7 は各 RCA に 100 万個のランダムデータを入力したときの実効クロック周期とエラー率を示している．これらの回路の実効クロック周期は，メインクロック周期を小さくすると短縮された．しかし，ACD 方式と EDC 方式の RCA は一部の計算で複数のメインクロックサイクルを消費したため，固定レイテンシの RCA よりも実効クロック周期が大きくなった．

ACD 方式の RCA は，メインクロック周期が 0.50 ns であっても計算エラーが発生しなかった．この結果は ACDM が処理の完了を正しく検出したことを意味する．一方，固定レイテンシと EDC 方式の RCA はメインクロック周期がそれぞれ 3.75 ns と 3.52 ns 未満のとき，計算エラーが発生した．EDC 方式の RCA は 3.53 ns 以上の周期のメインクロックを用いると，タイミングエラーを適切に検出し回復したことで，計算エラーは発生しなかった．

計算エラーが発生しなかった場合の各 RCA の最小実効クロック周期は 3.75 ns，3.53 ns および 1.36 ns である．これらの結果から，ACD 方式の RCA は他方式の RCA と比べて実効クロック周期を 60% 以上削減できることを確認した．

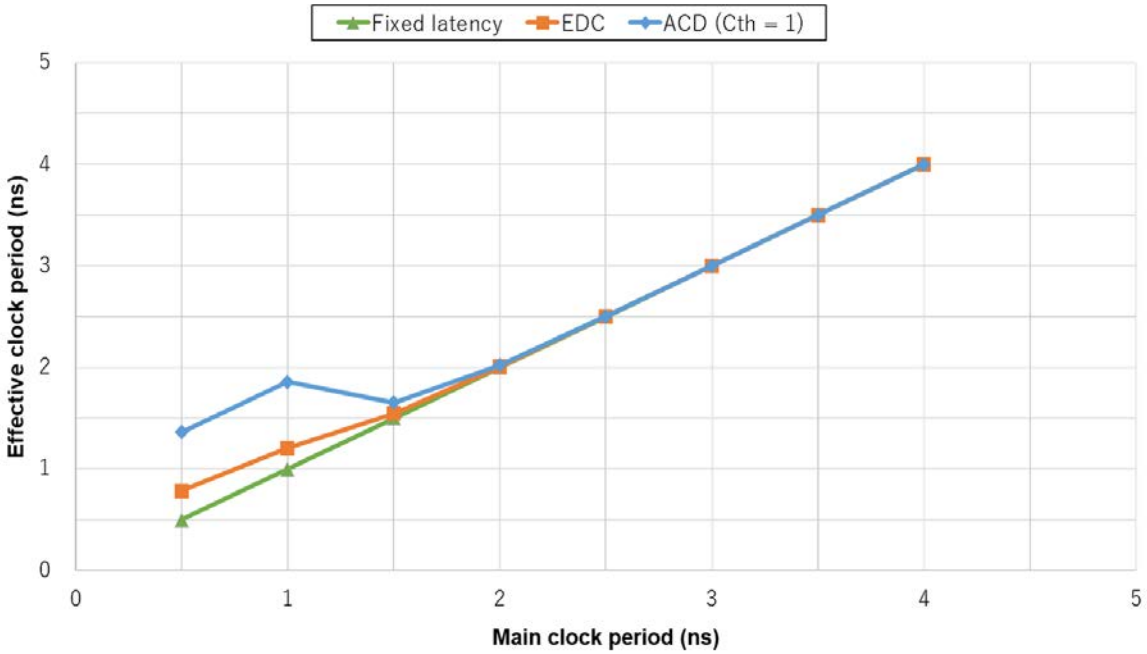


図 4.6　RCA の実効クロック周期

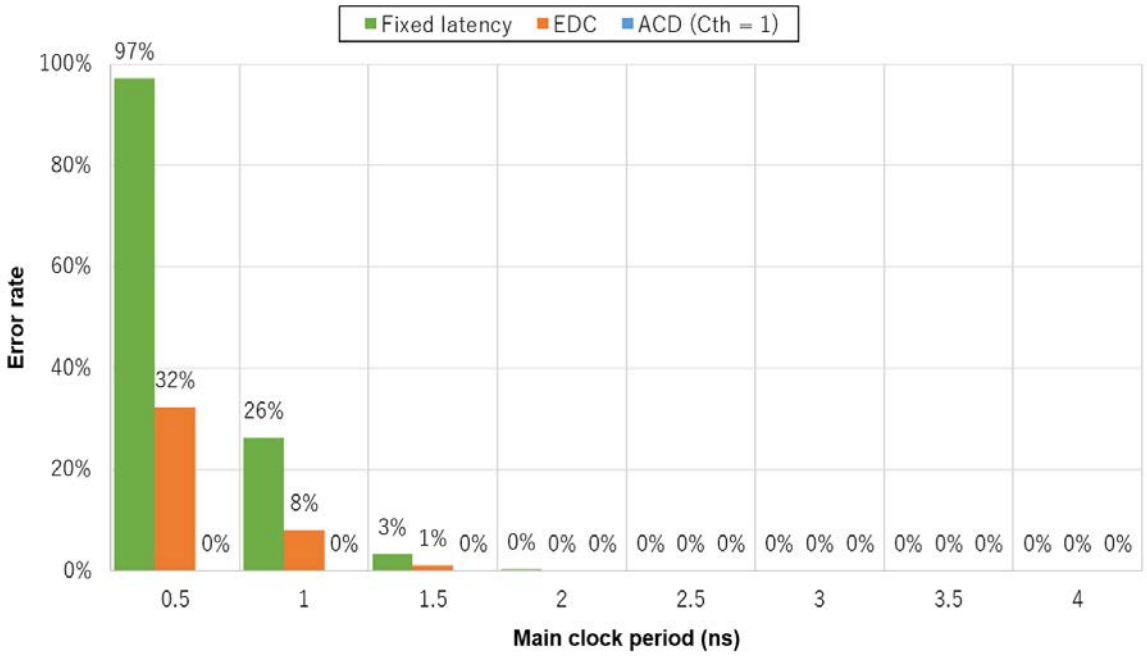


図 4.7　RCA のエラー率

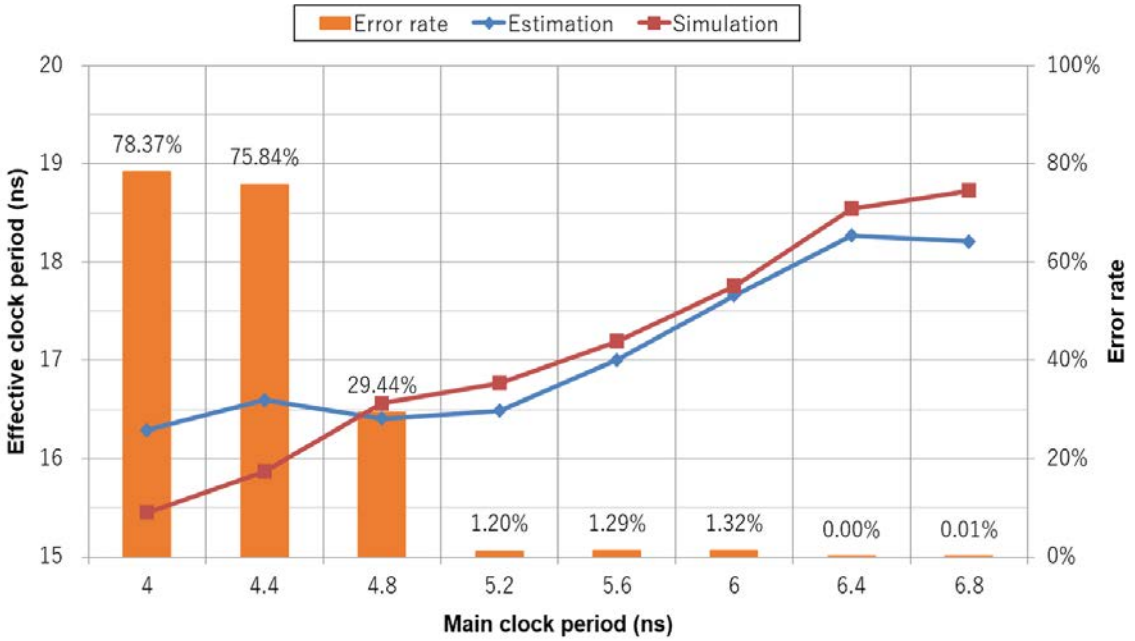


図 4.8　ACD 方式の 2D-DCT 回路の実効クロック周期とエラー率

次に，ACD 方式の画像処理回路の実効クロック周期とエラー率を評価した．これらの実験では，VGA 画像（640 × 480 pixels）をサンプルデータとして入力した．実効クロック周期の推定値は，画像処理回路に対して動的遅延解析を行って得た遅延分布と，式 4.1 から 4.3 を用いて計算で求めた．エラー率は，固定レイテンシと ACD 方式の画像処理回路の計算結果を比較し，式 4.6 を用いて得た．

図 4.8 は ACD 方式の 2D-DCT 回路の実効クロック周期とエラー率を示している．この図は周期の短いメインクロックを用いると，実効クロック周期が短縮されることを示している．メインクロック周期が 4.8 ns 以上のとき，シミュレーション結果は推定値よりも小さい．一方，メインクロック周期をさらに小さくすると，その関係が逆転する．これは，ACDM の誤った判断に基づいて回路が短い時間で処理を行ったからである．よって，メインクロック周期を小さくするとエラー率が高くなった．

我々の設計は性能目標を必要とする．ここではケーススタディとして，固定レイテンシの 2D-DCT 回路より実効クロック周期を 5% 改善することを検討した．目標とする実効クロック周期をさらに小さくする場合，エラー率が急激に悪化する．設計した 2D-DCT 回路の最大遅延が 17.92 ns であることから，目標の実効クロック周期は 17.02 ns 以下である．図 4.8 は，5.2 ns 以下の周期のメインクロックを用いることで目標の実効クロック周期を

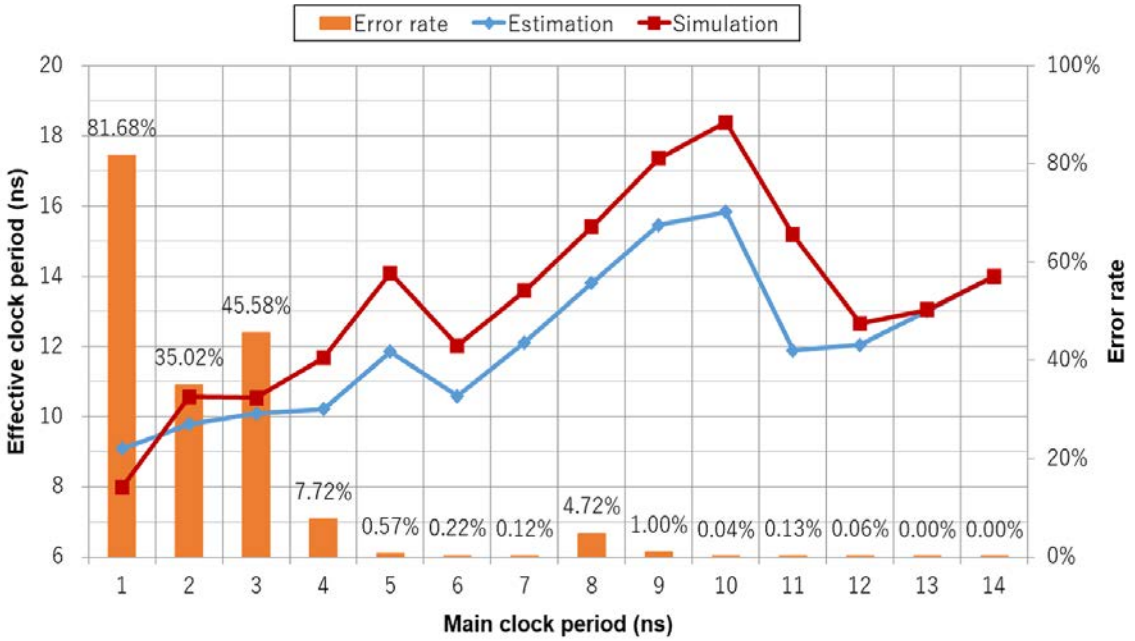


図 4.9　ACD 方式の HOG 特徴量計算回路の実効クロック周期とエラー率

達成できることを示している．最も低いエラー率はメインクロック周期を 5.2 ns にしたときに得ることができ，1.20% のエラー率で実効クロック周期が約 6% 改善された．

図 4.9 は ACD 方式の HOG 特徴量計算回路の実効クロック周期とエラー率を示している．この回路の実効クロック周期は，短い周期のメインクロックを用いることで短縮された．ただし，5 ns，8〜11 ns および 14 ns の周期のメインクロックを使用すると，シミュレートされた実効クロック周期は最大遅延を超えた．これは，回路が処理に複数サイクルをかけたことで，固定レイテンシ回路の処理遅延を超えることがあったためである．このように，C_{th} が 2 以上の ACDM を使用すると，処理完了の誤検出が減る代わりに完了判断に時間がかかり，実効クロック周期が改善されないことがある．

高速画像処理回路を設計するケーススタディとして，HOG 特徴計算回路の実効クロック周期を 10% 改善することを検討した．この実験では，目標性能を達成し，かつエラー率をできる限り小さくするために C_{th} を 6 とする ACDM を用いた．設計した HOG 特徴量計算回路の最大遅延が 13.60 ns であることから，目標の実効クロック周期は 12.24 ns 以下である．図 4.9 の推定結果は 12 ns，11 ns および 7ns 以下の周期のメインクロックを使用すると目標の実効クロック周期が達成されることを示している．一方，シミュレーション結果は，6 ns および 4 ns 以下の周期のメインクロックを使用すると目標の実効クロック周期

が達成されることを示している．これらの違いは，シミュレーションがより複雑な条件を考慮しているからである．最も低いエラー率はメインクロック周期を 6 ns にしたときに得ることができ，0.22% のエラー率で実効クロック周期が約 12% 改善された．

以上の結果から，ACD 方式が異なるタイプの回路の実効クロック周期を低いエラー率で改善できることを確認した．

回路面積と消費電力の評価

ACDM によるオーバヘッドを確認するために，設計した回路の面積と消費電力を評価した．回路面積を表 4.2 に示し，消費電力を表 4.3 に示す．これらは 0.18 μm スタンダードセル・ライブラリと Synopsys 社の Design Compiler[83] を用いて得た．

ACD 方式と EDC 方式の RCA は回路規模がほとんど同じであり，固定レイテンシの RCA よりも 60% 大きかった．これは RCA の回路規模が小さいことが理由である．ACD 方式を用いた 2D-DCT 回路と HOG 特徴量計算回路は，元の回路規模が大きいため，面積の増加率は 1% 未満だった．これらの結果から，回路規模が大きい画像処理回路に対して ACDM のオーバヘッドが小さいことを確認した．一方，ACD 方式を用いることで各回路の消費電力は増加した．これは高速なメインクロックを使ったことが原因であり，データパスにおける消費電力が大きく増加している．

これらの表は面積遅延積（Area-Delay Product, ADP）と電力遅延積（Power-Delay Product, PDP）も示している．ADP は面積と実効クロック周期の積として定義し，PDP は電力と実効クロック周期の積として定義する．固定レイテンシ回路の ADP と PDP を 1 とすると，ACD 方式の RCA，2D-DCT 回路，HOG 特徴量計算回路の ADP はそれぞれ 0.57，0.94，0.89 であり改善した．一方，PDP はそれぞれ 6.23，3.68，2.05 であり悪化した．

表 4.2　（a）RCA，（b）2D-DCT 回路，（c）HOG 特徴量計算回路の回路面積と ADP

(a)

Method	Effective clock period (ns)	Area (μm^2) RCA	Area (μm^2) Extra	Normalized ADP
Fixed latency	3.84	6675	-	1.00
EDC	3.53	-	3995	1.47
ACD	1.36	-	4056	0.57

(b)

Method	Effective clock period (ns)	Area (μm^2) 2D-DCT	Area (μm^2) Extra	Normalized ADP
Fixed latency	17.92	4303337	-	1.00
ACD	16.77	-	36402	0.94

(c)

Method	Effective clock period (ns)	Area (μm^2) HOG	Area (μm^2) Extra	Normalized ADP
Fixed latency	13.60	6423705	-	1.00
ACD	12.01	-	6732	0.89

表 4.3 （a）RCA，（b）2D-DCT 回路，（c）HOG 特徴量計算回路の消費電力と PDP

(a)

Method	Effective clock period (ns)	Power (mW) RCA	Power (mW) Extra	Normalized PDP
Fixed latency	3.84	2.43	-	1.00
EDC	3.53	2.64	1.70	1.64
ACD	1.36	18.66	24.07	6.23

(b)

Method	Effective clock period (ns)	Power (mW) 2D-DCT	Power (mW) Extra	Normalized PDP
Fixed latency	17.92	186.14	-	1.00
ACD	16.77	511.02	27.53	3.68

(c)

Method	Effective clock period (ns)	Power (mW) HOG	Power (mW) Extra	Normalized PDP
Fixed latency	13.60	633.93	-	1.00
ACD	12.02	1436.15	38.92	2.05

第 5 章

VSFC 対応映像監視サービスの評価

提案技術を使った VSFC サーバノードにより低遅延な画像処理サービスを実現できることを確認するため，2 種類の画像処理機能からなる VSFC 対応映像監視サービスを開発し評価した．本章では，FPGA 搭載サーバに実装した VSFC 対応映像監視サービスの処理性能の評価結果と，ACD 方式の可変レイテンシ回路により高速化した映像監視サービスの品質の評価結果を示す．

5.1 VSFC 対応映像監視サービスの開発

開発した VSFC 対応映像監視サービスは，HOG 特徴量を計算する機能と移動物体を検出機能から成る．前者の機能は 3.5.1 項で説明した HOG 特徴量計算回路により実現した．また，後者の機能のためにフレーム間差分法による移動物体検出回路を開発した．FPGA 搭載サーバ上に実現した提案アーキテクチャを用いてこれらの機能を組み合わせることで，VSFC 対応映像監視サービスを実現した．

VSFC 対応映像監視サービスの構成を図 5.1 に示す．このサービスは Xeon Gold 6132 CPU と Arria 10 GX 1150 FPGA を用いて実行される．この構成における処理手順は以下のとおりである．

1. テストプログラム（クライアント）が複数のパケットに分割した画像データを IPB #0 に送信する．
2. IPB #0 は画像データの HOG 特徴量を計算し，IPB #1 に計算結果を送る．
3. IPB #1 は HOG 特徴量から画像に含まれる移動物体を検出し，その結果をテストプログラムに送信する．

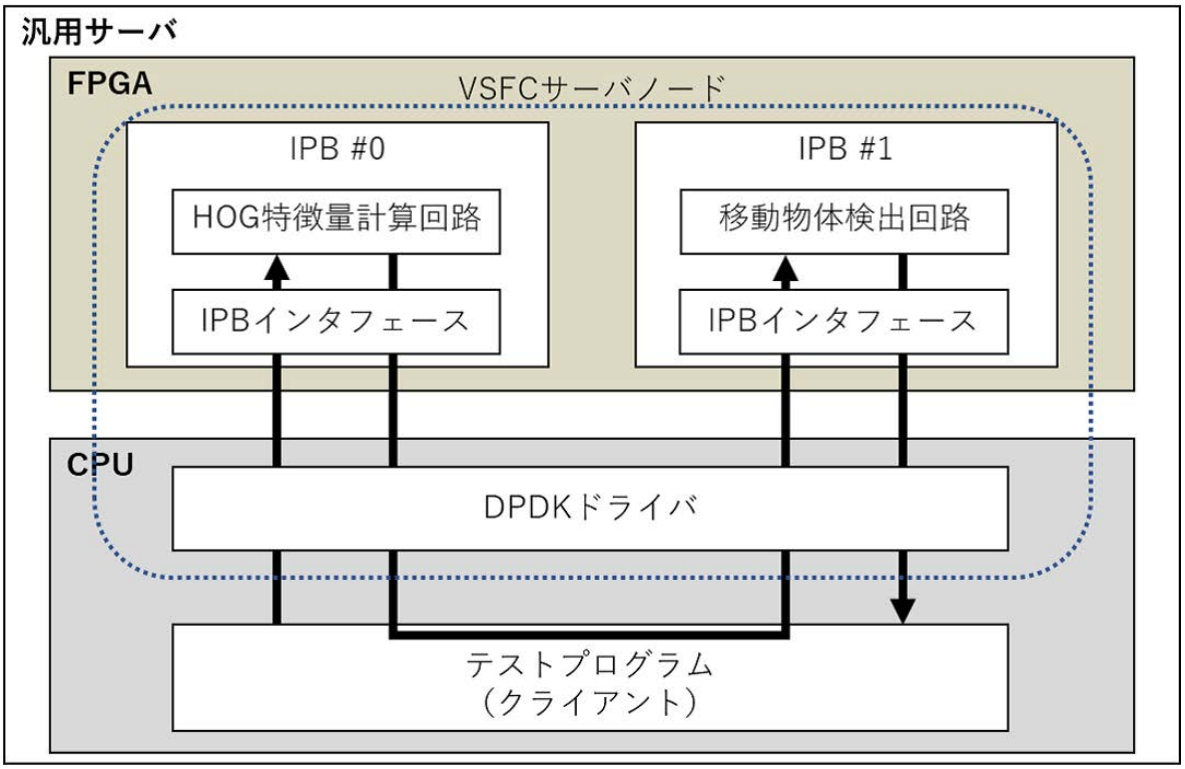


図 5.1　VSFC 対応映像監視サービスの構成

この構成では仮想スイッチの代わりに DPDK ドライバを使ってパケットを転送する．これは開発したサービスの限界性能を測定するためである．なお，各 IPB の処理遅延を測定するために，IPB #0 が送信したパケットを一度テストプログラムに戻している．これにより測定される処理遅延は若干増加する．また，この構成ではサーバ間の通信遅延は発生せず，サーバ内のデータ転送遅延と画像処理遅延のみ発生する．

5.2　VSFC 対応映像監視サービスの性能評価

5.2.1　シミュレーション

シミュレーションにより HOG 特徴量計算回路と移動物体検出回路を組み合わせた映像監視の処理遅延を評価した．この実験では，複数の VGA 画像を HOG 特徴量計算回路に入力し，移動物体検出回路から検出結果を得た．図 5.2 にシミュレーション結果を示す．HOG 特徴量の計算に 208.6 μs，移動物体の検出に 226.7 μs かかった．一方，サービス全体の遅延は 227.8 μs だった．この結果は，リアルタイムパケット順序制御により完全な画

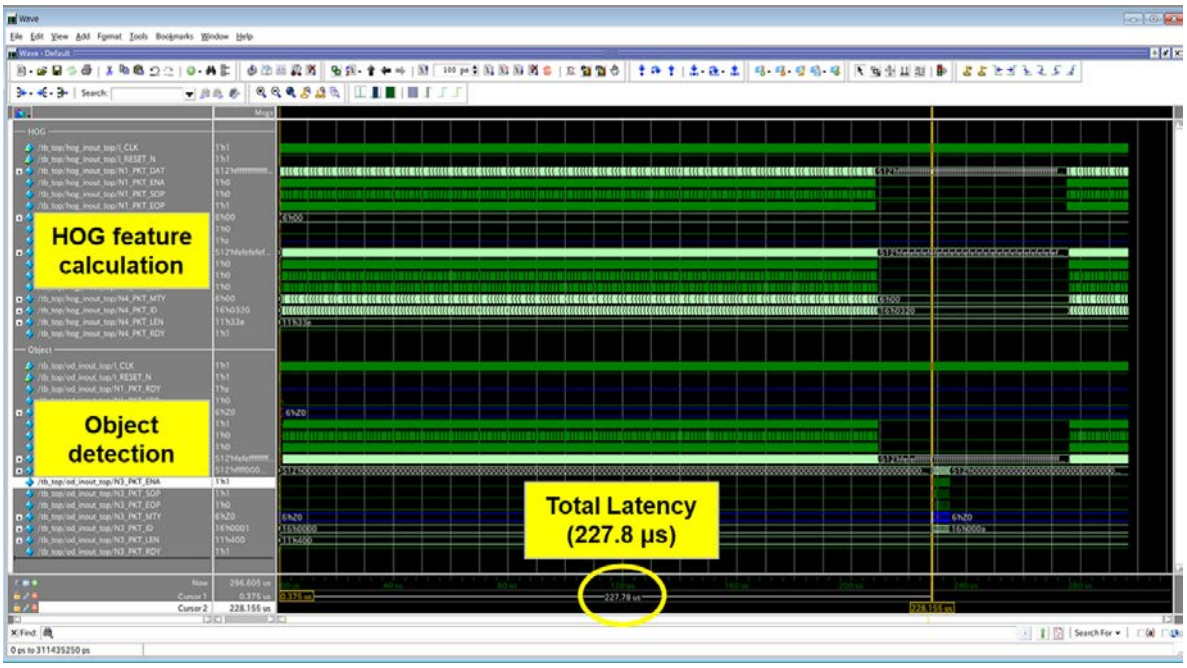


図 5.2　VSFC 対応映像監視サービスのシミュレーション

像データの到着を待たずに HOG 特徴量の計算が行われ，機能間のデータ転送時間が隠蔽されたことを示している．これにより，サービス全体の処理遅延は各回路の処理遅延の合計よりも小さくなった．

5.2.2　実機評価

FPGA 搭載サーバに実装した VSFC 対応映像監視サービスの処理性能を評価した．この実験では，複数の VGA 画像を 40 Gbps のレートで入力した．この実験で各 IPB とサービス全体の処理遅延を測定したところ，IPB #0 と IPB #1 の平均処理遅延はそれぞれ 253.4 μs と 283.2 μs であり，サービス全体の処理遅延は 301.6 μs であった．測定結果はソフトウェアの処理時間と DMA 転送の時間を含んでいるため，シミュレーションの処理遅延よりも大きくなった．図 5.3 は画像ごとに測定した IPB とサービス全体の処理遅延を示している．各 IPB は一定の遅延で処理を行っており，サービスを安定して動作させることができた．

この実験では，複数の入力ソースから画像データを受信する場合のオーバヘッドを考慮していない．IPB #1 は連続する画像フレームのデータを比較して移動物体を検出するため，異なる入力ソースのデータを受信すると参照するデータを変えなければならない．一

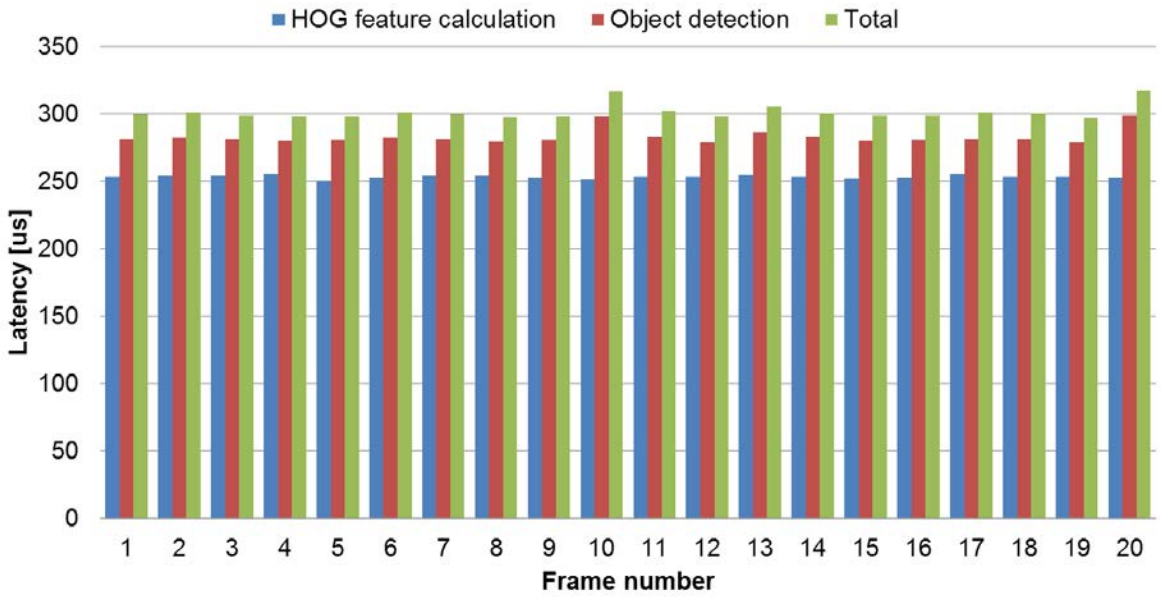


図 5.3 VSFC 対応映像監視サービスの処理性能

方，IPB #0 は内部に参照値を持たないため，複数の入力ソースから画像を受信してもオーバヘッドなしに処理できる．

例えば，IPB #1 が異なるユーザのデータを処理するために DDR3 メモリから参照データを取得する場合，参照データを受信するのに少なくとも 28 μs かかる．この遅延を IPB #1 の処理時間に加えると，機能間のパイプライン処理を考慮してサービス全体の遅延は 329.6 μs になる．この場合，このサービスは毎秒 3000 枚の VGA 画像を処理できる．この処理性能は，例えば 60 fps のネットワークカメラを入力ソースとして仮定すると，50 台のネットワークカメラからの映像をリアルタイムに処理できることを意味する．

これらの結果から，提案アーキテクチャのサーバノードで画像処理機能を組み合わせることで低遅延かつ高スループットの映像監視サービスを提供できることを確認した．

5.3　可変レイテンシ回路を用いた映像監視サービスの品質評価

5.3.1　準備

映像監視サービスは多くの場合サービス品質を低下させない計算エラーを許容できる．そこで，ACD 方式の可変レイテンシ回路により高速化したサービスの品質を評価した．実験のため，ACD 方式の HOG 特徴量計算回路を設計した．また，比較のために，Lower-part OR Adder（LOA）[59] と Segmented Adder（SA）[60] の 2 種類のアプロキシメートコンピューティング手法を用いた HOG 特徴量計算回路を設計した．LOA は OR ゲートと全加算器で構成される近似加算器であり，OR ゲートにより下位ビットを簡単に計算することで高速に動作する．SA は構成要素である全加算器がいくつかのグループに分かれており，グループ内でのみ信号を伝搬させることで高速動作を実現する．

設計した 24 ビット LOA と 48 ビット LOA について説明する．24 ビット LOA は最下位ビット（LSB）から 6 ビット目の加算を OR ゲートにより行い，7 ビット目から最上位ビット（MSB）の加算を全加算器で行う．一方，48 ビット LOA は LSB から 17 ビット目の加算を OR ゲートにより行い，それ以上のビットの加算を全加算器で行う．

次に設計した 24 ビット SA と 48 ビット SA について説明する．24 ビット SA の全加算器は，LSB から 4 ビット目までのグループと，5 ビット目から MSB までのグループに分かれて加算を行う．一方，48 ビット LOA の全加算器は，LSB から 8 ビット目までのグループ，9 ビット目から 28 ビット目までのグループ，および 29 ビット目から MSB のグループに分かれて加算を行う．

5.3.2　評価結果

HOG 特徴量計算回路の計算エラーが映像監視サービスに与える影響を調べた．実験では，設計した HOG 特徴量計算回路に対し，VCS シミュレータを用いて HOG 特徴量を計算した．そして，フレーム間差分法による物体検出プログラムに計算した HOG 特徴量を入力し，移動物体を含む画像ブロックを検出した．この実験には PETS 2009 Benchmark Data[84] に含まれる 3 枚の時間的に連続する VGA 画像を使用した．この画像には 8×8 ピクセルの画像ブロックが 4800 個含まれる．

表 5.1 に HOG 特徴量計算回路の実効クロック周期とエラー率を示す．これらのデータ

は HOG 特徴量を計算するときに合わせて取得した．固定レイテンシ回路と比べて，他の回路は実効クロック周期が約 10% 短縮した．しかし，これらの回路では計算エラーが発生した．ACD 方式の回路のエラー率は 0.3% であった．一方，LOA および SA を用いた回路は近似計算を行ったことで，エラー率が 99% 以上であった．ただし，これらの回路は誤差の小さい計算を行った．

同じ表に物体検出プログラムにより検出した正しい検出点の数と誤検出点の数を示す．正しい検出点は，固定レイテンシ回路が計算した HOG 特徴量を用いて検出した画像ブロックである．誤検出点は，計算エラーが発生する回路が計算した HOG 特徴量を用いたときのみ検出された画像ブロックである．LOA および SA を用いた回路は物体検出プログラムの検出精度を約 92% と 28% 低下させた．さらに，17 個と 31 個の誤検出点が得られた．一方，ACD 方式の回路を用いた場合，物体検出プログラムの検出精度の低下は 1% 未満であり，誤検出点は 1 個だけだった．

これらの結果から，ACD 方式の HOG 特徴量計算回路を用いることで，映像監視サービスの品質をほとんど低下させることなく処理性能を向上できることを確認した．

表 5.1　HOG 特徴量計算回路の実効クロック周期とエラー率，並びに物体検出プログラムの検出精度

Method	HOG feature calculation circuit					Object detection application	
	T_{main} (ns)	C_{th}	T_{sub} (ns)	Effective clock period (ns)	P_{err} (%)	# of correct detection points	# of false-positive points
Fixed latency	13.60	-	-	13.60	-	120	-
LOA	12.34	-	-	12.34 (-9.3%)	99.96	10 (-91.7%)	17
SA	12.19	-	-	12.19 (-10.4%)	99.35	86 (-28.3%)	31
ACD	6.00	6	0.20	12.02 (-11.6%)	0.30	119 (-0.8%)	1

第 6 章

おわりに

6.1 結論

利用者の目的に合致したクラウド画像処理サービスを提供するために，ネットワーク上で画像処理機能を組み合わせる VSFC を提案した．VSFC による柔軟かつ低遅延な画像処理サービスを実現するために，リアルタイムパケット順序制御回路を備えた CPU-HWA サーバアーキテクチャと ACD 方式の可変レイテンシ回路を提案し，サーバノード内の遅延を削減した．

実験により，提案アーキテクチャにおける HOG 特徴量計算機能は，従来の CPU-FPGA 構成と比べて，26 分の 1 の小さな処理遅延で動作することを確認した．この機能は，90% のパケットがランダムに入れ替わる状況でも高速に処理を行った．また，HOG 特徴量計算機能を FPGA にオフロードすることで，ソフトウェア構成と比べて，消費電力が 37% 削減されることを確認した．

次に，ACD 方式を導入した RCA，2D-DCT 回路および HOG 特徴量計算回路の処理性能を評価し，従来の固定レイテンシ回路と比べて処理性能が約 64%，6%，12% 向上，かつエラー率が 1% 程度であることを確認し，ACD 方式により異なるタイプの画像処理回路を高速化できることを示した．さらに，ACD 方式の導入により回路の面積遅延積が小さくなることを示した．

続いて，提案アーキテクチャを FPGA 搭載サーバで実現し，2 種類の画像処理機能を組み合わせた VSFC 対応映像監視サービスの処理性能を評価した．その結果，301.6 μs の処理遅延でリアルタイムに物体検出を行えることを確認した．

最後に，ACD 方式の HOG 特徴量計算回路を用いた映像監視サービスの品質を評価し，1% 未満のサービス品質の低下で処理性能が向上することを確認した．

以上から，提案技術によりサーバノード内のデータ転送遅延と画像処理遅延を削減できることを確認した．また，提案技術を用いたサーバノードにより VSFC による低遅延な映像監視サービスを実現できることを確認した．

6.2　今後の展望

VSFC および VSFC サーバノードに関して以下が今後の課題である．

- 複数ユーザに対して画像処理を行う VSFC サーバノードの開発
- VSFC による柔軟な画像処理サービスの実証
- FPGA 等を用いた ACD 方式の可変レイテンシ回路の実現

リアルタイムパケット順序制御回路を備えた CPU-HWA サーバアーキテクチャにより，VSFC による低遅延な映像監視サービスを実現できることは示せた．しかし，このサービスは経路が固定されている．また，IPB は異なるユーザからデータを受信しても適切な参照値やパラメータを利用できない．よって，VSFC による柔軟な画像処理サービスを実証するには，入力に応じて回路が保持する情報を変更する機能を具備した IPB を開発する必要がある．さらに，VSFC サーバノードを NC から制御できるようにする必要がある．

また，本研究では ACD 方式の可変レイテンシ回路により，サービス品質をほとんど低下させることなく映像監視サービスが高速化されることをシミュレーションにより示した．より高速な VSFC 対応画像処理サービスを実現するために，ACD 方式の可変レイテンシ回路が FPGA などを用いて実現できることを確認する必要がある．

付録 A

ソースコード A.1　ソフトウェアパケット順序制御

```
int dpdk_reorder::input(unsigned char *packet){
    int frame_id = (packet[44] << 8) + packet[45];
    int block_id = (packet[54] << 12) + (packet[55] << 4) + (packet[56] >> 4) - 1;
    int seq_no = PKT_NUM * (frame_id - 1) + block_id;
    int buf_no = block_id % BUFFER_SIZE;
    if(seq_no < 0 || seq_no > 800) seq_no = 0x7FFFFFFF;
    if(seq_no < min_seq_no){ // late
        late_  = 1;
        valid_ = 0;
        early_ = 0;
    }else if(seq_no < min_seq_no + BUFFER_SIZE){ // valid
        for(int i = 0; i < PKT_LEN - HEADER_LEN; i++){
            reorder_buffer[buf_no * PKT_LEN + i] = packet[i + HEADER_LEN];
        }
        late_  = 0;
        valid_ = 1;
        early_ = 0;
    }else{ // early
        for(int i = 0; i < BUFFER_SIZE * PKT_LEN; i++){
            send_buffer[i] = reorder_buffer[i];
        }
        min_seq_no += BUFFER_SIZE;
        for(int i = 0; i < PKT_LEN - HEADER_LEN; i++){
            reorder_buffer[buf_no * PKT_LEN + i] = packet[i + HEADER_LEN];
        }
        ready_ = 1;
        late_  = 0;
        valid_ = 0;
        early_ = 1;
    }
    return ready_;
}
```

付録 B

ソースコード B.1　ACDM

```
module ACDM
(
    input  wire             CLK,
    input  wire             nRST,
    input  wire [WIDTH-1:0] Din,
    output reg              En
    );
    wire            diff;
    reg [WIDTH-1:0] comp_ff;
    reg [3:0]       cnt;

    always @(posedge CLK or negedge nRST)begin
        if(!nRST == 1)begin
            comp_ff <= 0;
        end else begin
            comp_ff <= Din;
        end
    end

    assign diff = |(Din ^ comp_ff);
    always @(posedge CLK or negedge nRST)begin
        if(!nRST == 1'b1)begin
            cnt <= 1'b0;
            En  <= 1'b0;
        end else begin
            if(diff == 1'b1)begin
                nt <= 1'b0;
                n  <= 1'b0;
        end else if(cnt < COMP)begin
                cnt <= cnt + 1'b1;
        end else begin
                cnt <= cnt;
                En  <= 1'b1;
            end
        end
    end
endmodule
```

ソースコード B.2　S-FF

```
module S-FF
    (
        input  wire             spCLK,
        input  wire             cfCLK,
        input  wire [WIDTH-1:0] Din,
        output wire [WIDTH-1:0] Dout,
        output wire             Error
    );
    wire [WIDTH-1:0]    sp_out;
    wire [WIDTH-1:0]    cf_out;
    wire                diff;
    reg                 err;
    assign Dout  = sp_out;
    assign Error = err;
    spFF spff0 (.CLK(spCLK), .Din(Din), .Ref(cf_out), .Error(err), .Dout(sp_out));
    nrFF cfff0 (.CLK(cfCLK), .Din(Din), .Dout(cf_out));
    assign diff = (sp_out != cf_out) && (~spCLK | cfCLK);
    always @(negedge spCLK)begin
        if(err == 0)begin
            err <= diff;
        end else begin
            err <= 0;
        end
    end
endmodule
```

謝辞

はじめに，大阪大学での学部 4 年の 1 年間と修士課程の 2 年間，そして東京工業大学での博士課程の 4 年半，計 7 年半もの間終始熱心なご指導を賜りました，指導教員の高橋篤司教授に深く感謝の意を表します．研究室に配属されたときは研究のイロハも知らなかった私を辛抱強く導いて下さりありがとうございました．また，社会人ドクターとして戻りたいと言った私を快く引き受けて頂けたこと感謝します．先生には，研究の方針から論文執筆に至るまで，あらゆる面において貴重なご助言を頂きました．また，多くの発表の機会を与えていただき，多くの研究者と知り合えたことは貴重な財産です．改めてお礼申し上げます．

佐藤真平助教には，日ごろの活発な議論と多くの貴重な意見を頂きました．先生と行った議論から多くのアイデアが生まれました．また，先輩として博士課程における研究の進め方や研究に対する姿勢について多くのご助言を頂きました．ここに深く感謝いたします．

本論文の審査員として，一色剛教授，本村真人教授，中原啓貴准教授，原祐子准教授には的確な指摘と提案を頂きました．お世話になった多くの先生方に深く感謝いたします．

研究を事務的な面でお手伝い頂いた志村玲子秘書に感謝いたします．いろいろと無茶なお願いをしているにもかかわらず，いつも快く引き受けて頂いて大変有難かったです．

高橋研究室の皆様に感謝いたします．あまり交流を持つ機会がありませんでしたが，ともに研究に励む仲間がいることに心強さを感じていました．

学外における研究の場として，電子情報学会 VLSI 設計技術研究会で活発な議論をした皆様に感謝いたします．

最後に，仕事と子育てで自身も多忙を極めるにも関わらず，献身的に支えてくれた妻の薫に深く感謝いたします．

参考文献

[1] C. Kachris and D. Soudris, “A survey on reconfigurable accelerators for cloud computing,” Proceedings of 2016 26th International Conference on Field Programmable Logic and Applications (FPL), pp. 1-10, 2016.

[2] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, *et al.*, “A cloud-scale acceleration architecture,” Proceedings of 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 1-13, 2016.

[3] I. Magaki, M. Khazraee, L. V. Gutierrez, and M. B. Taylor, “ASIC Clouds: Specializing the Datacenter,” Proceedings of 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), pp. 178-190, 2016.

[4] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, *et al.*, “A reconfigurable fabric for accelerating large-scale datacenter services,” Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA), pp. 13-24, 2014.

[5] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, *et al.*, “Serving DNNs in Real Time at Datacenter Scale with Project Brainwave,” J. IEEE Micro, Vol. 38, No. 2, pp. 8-20, 2018.

[6] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, *et al.*, “In-Datacenter Performance Analysis of a Tensor Processing Unit,” Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA), pp. 1âĂŞ12, 2017.

[7] P. K. Gupta, “Xeon+ FPGA platform for the data center,” The Fourth Workshop on the Intersections of Computer Architecture and Reconfigurable Logic (CARL), 2015.

[8] D. Pellerin, “FPGA Accelerated Computing Using AWS F1 Instances,” AWS Public

Sector Summit 2017, 2017.

[9] Google LLC, “Google Cloud,” https://cloud.google.com/, accessed on December 31, 2020.

[10] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-Defined Networking: A Comprehensive Survey,” Proceedings of the IEEE, Vol. 103, No. 1, pp. 14-76, 2015.

[11] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, “Network Function Virtualization: State-of-the-Art and Research Challenges,” Proc. IEEE Communications Surveys & Tutorials, Vol. 18, No. 1, pp. 236-262, 2016.

[12] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A Survey on Service Function Chaining," J. Network and Computer Applications, Vol. 75, pp. 138-155, 2016.

[13] H. Kitada, H. Kojima, N. Takaya, and M. Aihara, “Service function chaining technology for future networks,” NTT Technical Review Vol. 12, No. 8, pp. 1-5, 2014.

[14] J. Halpern and C. Pignataro, “Service Function Chaining (SFC) Architecture,” IETF RFC 7665, 2015.

[15] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, “Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges,” IEEE Communications Magazine, Vol. 55, No. 2, pp. 216-223, 2016.

[16] Y. Xie, Z. Liu, S. Wang, and Y. Wang, “Service Function Chaining Resource Allocation: A Survey,” arXiv preprint arXiv:1608.00095, 2016.

[17] C. Kachris, G. Sirakoulis, and D. Soudris, “Network Function Virtualization based on FPGAs A Framework for all Programmable network devices,” arXiv preprint arXiv:1406.0309, 2014.

[18] L. Nobach and D. Hausheer, “Open, Elastic Provisioning of Hardware Acceleration in NFV Environments,” Proceedings of 2015 International Conference and Workshops on Networked Systems (NetSys), pp. 1-5, 2015.

[19] P. Quinn, U. Elzur, and C. Pignataro, “Network service header (NSH),” IETF RFC 8300, 2018.

[20] M. T. Beck and J. F. Botero, “Coordinated Allocation of Service Function Chains,” Proceedings of 2015 IEEE Global Communications Conference (GLOBECOM), pp. 1-6, 2015.

[21] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and S. Davy, “Design and

Evaluation of Algorithms for Mapping and Scheduling of Virtual Network Functions," Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft), pp. 1-9, 2015.

[22] L. Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," IEEE Transactions on Communications, Vol. 64, No. 9, pp. 3746âĂŞ3758, 2016.

[23] N. McKeown, T. E. Anderson, H. Balakrishnan, G. Parulkar, L. L. Peterson, J. Rexford, S. J. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, Vol. 38, No. 2, pp. 69-74, 2008.

[24] Ryu SDN Framework Community, "Build SDN Agilely," https://ryu-sdn.org/, accessed on December 31, 2020.

[25] NEC Corporation, "Trema: Full-Stack OpenFlow Framework in Ruby and C," https://trema.github.io/trema/, accessed on December 31, 2020.

[26] L. V. Morales, A. F. Murillo, and S. J. Rueda, "Extending the Floodlight Controller," Proceedings of 2015 IEEE 14th International Symposium on Network Computing and Applications, pp. 126-133, 2015.

[27] Y. Ukon, K. Yamazaki, and K. Nitta, "Real-time Image Processing based on Service Function Chaining using CPU-FPGA architecture," IEICE Transactions on Communications, Vol. E103-B, No. 1, pp. 11-19, 2019.

[28] Y. Ukon, K. Yamazaki, and K. Nitta,"Video Service Function Chaining with a Real-time Packet Reordering Circuit," Proceedings of 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-5, 2018.

[29] Y. Watanabe, Y. Kobayashi, T. Takenaka, T. Hosomi, and Y. Nakamura, "Accelerating NFV application using CPU-FPGA tightly coupled architecture," Proceedings of 2017 International Conference on Field Programmable Technology (ICFPT), pp. 136-143, 2017.

[30] J. F. Zazo, S. Lopez-Buedo, Y. Audzevich, and A. W. Moore, "A PCIe DMA engine to support the virtualization of 40 Gbps FPGA-accelerated network appliances," Proceedings of 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig), pp. 1-6, 2015.

[31] K. Yamazaki, Y. Nakajima, T. Hatano, and A. Miyazaki, "Lagopus FPGA —A reprogrammable data plane for high-performance software SDN switches," Proceedings of 2015 IEEE Hot Chips 27 Symposium (HCS), pp. 1-1, 2015.

[32] K. Cindy, “Benefits of partial reconfiguration,” Xcell journal Vol. 55, pp. 65-67, 2005.

[33] H. Kavianipour, S. Muschter, and C. Bohm, “High Performance FPGA-Based DMA Interface for PCIe,” IEEE Transactions on Nuclear Science, Vol. 61, No. 2, pp. 745-749, 2014.

[34] L. Rota, M. Caselle, S. Chilingaryan, A. Kopmann, and M. Weber, “A PCIe DMA Architecture for Multi-Gigabyte Per Second Data Transmission,” IEEE Transactions on Nuclear Science, Vol. 62, No. 3, pp. 972-976, 2015.

[35] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, “Extending Networking into the Virtualization Layer,” Proc. Eighth ACM Workshop on Hot Topics in Networks (HotNets), October. 2009.

[36] The Fast Data Project, “Vector Packet Processing (VPP),” FD.IO Project, https://fd.io/, accessed on December 31. 2020.

[37] Lagopus project, “Lagopus switch and router,” www.lagopus.org/, accessed on December 31. 2020.

[38] M. Kwon, K. P. Neupane, J. Marshall, and M. M. Rafique, “CuVPP: Filter-based Longest Prefix Matching in Software Data Planes,” Proceedings of 2020 IEEE International Conference on Cluster Computing (CLUSTER), pp. 12-22, 2020.

[39] P. Michal, B. Belter, and A. Binczewski, “Shall we worry about packet reordering?," Computational Methods in Science and Technology, Vol. 11, No. 2, pp. 141-146, 2005.

[40] R. Brown, “Calendar queues: a fast 0(1) priority queue implementation for the simulation event set problem,” Communications of the ACM, Vol. 31, No. 10, pp. 1220-1227, 1988.

[41] S.-W. Moon, J. Rexford, and K. G. Shin, “Scalable Hardware Priority Queue Architectures for High-Speed Packet Switches,” IEEE Transactions on Computers, Vol. 49, No. 11, pp. 1215-1227, 2000.

[42] Intel Corporation, “Developer Quick Start Guide,” https://www.dpdk.org/, accessed on December 31, 2020.

[43] Intel Corporation, “DPDK documentation,” https://doc.dpdk.org/guides/index.html, accessed on December 31, 2020.

[44] P. Jon, “Transmission control protocol,” IETF RFC 793, 1981.

[45] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, “RTP: A transport protocol for real-time applications,” IETF RFC 3550, 2003.

[46] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” Pro-

ceedings of 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), pp. 886-893, 2005.

[47] 露木明宣 富岡洋一, 北澤仁志, "ハードウェア構成に適した HOG 特徴量計算手法と回路構成," 情報科学技術フォーラム講演論文集, Vol. 11, No. 3, pp. 203-204, 2012.

[48] Intel Corporation, "INTEL FPGA SDK FOR OpenCL," https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html, accessed on December 31, 2020.

[49] Intel Corporation, "Intel Quartus Prime Software Suite," https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html, accessed on December 31, 2020.

[50] Y. Ukon, S. Sato, and A. Takahashi, "Design Method of Variable-Latency Circuit with Tunable Approximate Completion-Detection Mechanism," IEICE Transactions on Electronics. (in press)

[51] G. E. Moore, "Cramming more components onto integrated circuits," Electronics, Vol. 38, No. 8, pp. 114-117, 1965.

[52] J. P. Fishburn, "Clock skew optimization," IEEE Transactions on Computers, Vol. 39, No. 7, pp. 945-951, 1990.

[53] E. G. Friedman, "Clock distribution networks in synchronous digital integrated circuits," Proceedings of the IEEE, Vol. 89, No. 5, pp. 665-692, 2001.

[54] A. Takahashi and Y. Kajitani, "Performance and reliability driven clock scheduling of sequential logic circuits," Proceedings of ASP-DAC '97: Asia and South Pacific Design Automation Conference, pp. 37-42, 1997.

[55] A. Takahashi, "Practical Fast Clock-Schedule Design Algorithms," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. 89, No. 4, pp. 1005-1011, 2006.

[56] T. Yoda and A. Takahashi, "Clock Period Minimization of Semi-Synchronous Circuits by Gate-Level Delay Insertion," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. 82, No. 11, pp. 2383-2389, 1999.

[57] Y. Kohira and A. Takahashi, "Clock Period Minimization Method of Semi-Synchronous Circuits by Delay Insertion," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. 88, No. 4, pp. 892-898, 2005.

[58] H. Jiang, J. Han, and F. Lombardi, "A Comparative Review and Evaluation of Approximate Adders," Proceedings of the 25th edition on Great Lakes Symposium on VLSI,

pp. 343-348, 2015.

[59] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, “Bio-Inspired Imprecise Computational Blocks for Efficient VLSI Implementation of Soft-Computing Applications,” IEEE Transactions on Circuits and Systems I: Regular Papers, Vol. 57, No. 4, pp. 850-862, 2009.

[60] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, “Design of voltage-scalable meta-functions for approximate computing,” 2011 Design, Automation & Test in Europe, pp. 1-6, 2011.

[61] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, “Razor: a low-power pipeline based on circuit-level timing speculation,” Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36, pp. 7-18, 2003.

[62] D. Bull, S. Das, B. K. Shivashankar, G. Dasika, K. Flautner, and D. Blaauw, “A power-efficient 32b ARM ISA processor using timing-error detection and correction for transient-error tolerance and adaptation to PVT variation,” Proceedings of 2010 IEEE International Solid-State Circuits Conference (ISSCC), pp. 284-285, 2010.

[63] M. Kurimoto, H. Suzuki, R. Akiyama, T. Yamanaka, H. Ohkuma, H. Takata, and H. Shinohara, “Phase-adjustable error detection flip-flops with 2-stage hold driven optimization and slack based grouping scheme for dynamic voltage scaling,” Proceedings of the 45th annual Design Automation Conference, pp. 884-889, 2008.

[64] 右近 祐太, 高橋 篤司, 谷口 研二, “加算器におけるクロック周期に応じた遅延エラー率の評価,” 電子情報通信学会技術研究報告 (ICD2009-91), Vol. 109, No. 336, pp. 77-81, 2009.

[65] 右近 祐太, 井上 雅文, 高橋 篤司, 谷口 研二, “エラー検出回復方式における加算器の性能評価,” 電子情報通信学会技術研究報告 (VLD2009-121), Vol. 109, No. 462, pp. 133-138, 2010.

[66] S. Sato, H. Nakatsuka, and A. Takahashi, “Performance Improvement of General-Synchronous Circuits by Variable Latency Technique using Dynamic Timing-Error Detection,” Proceedings of the 20th Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI 2016), pp. 60-65, 2016.

[67] 右近 祐太, 井上 雅文, 高橋 篤司, 谷口 研二, “エラー検出回復方式を用いた演算器の性能評価手法,” VDEC デザイナーズフォーラム 2010, 2010.

[68] H. Fuketa, M. Hashimoto, Y. Mitsuyama, and T. Onoye, “Adaptive Performance Com-

pensation With In-Situ Timing Error Predictive Sensors for Subthreshold Circuits," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 20, No. 2, pp. 333-343, 2012.

[69] Y. Shi, H. Igarashi, N. Togawa, and M. Yanagisawa, "Suspicious timing error prediction with in-cycle clock gating," Proceedings of International Symposium on Quality Electronic Design (ISQED), pp. 335-340, 2013.

[70] T. Sato, and Y. Kunitake, "A Simple Flip-Flop Circuit for Typical-Case Designs for DFM," Proceedings of the 8th International Symposium on Quality Electronic Design (ISQED'07), pp. 539-544, 2007.

[71] 井上 雅文, 右近 祐太, 高橋 篤司, 谷口 研二, "エラー検出回復方式回路の回路構成と性能に関するシミュレーション評価," DA シンポジウム 2010 論文集, 情報処理学会シンポジウムシリーズ, Vol. 2010, No. 7, pp. 123-128, 2010.

[72] S. Sato, E. Sassa, Y. Ukon, and A. Takahashi, "A Low Area Overhead Design Method for High-Performance General-Synchronous Circuits with Speculative Execution," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E102-A, No. 12, pp. 1760-1769, 2019.

[73] S. Sato, E. Sassa, Y. Ukon, and A. Takahashi, "A Low Area Overhead Design for High-Performance General-Synchronous Circuits with Speculative Execution," Proceedings of 2019 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-5, 2019.

[74] 右近 祐太, 井上 雅文, 高橋 篤司, 谷口 研二, "FPGA 上に実現した可変レイテンシ回路の動作検証," 電子情報通信学会技術研究報告 (VLD2010-142), Vol. 110, No. 432, pp. 153-158, 2011.

[75] 右近 祐太, 安藤 健太, 高橋 篤司, "FPGA 上に実現した可変レイテンシ回路の性能評価," 電子情報通信学会技術研究報告 (VLD2011-141), Vol. 111, No. 450, pp. 127-132, 2012.

[76] 右近 祐太, 佐藤 真平, 高橋 篤司, "演算器の可変レイテンシ化による処理性能と回路面積のトレードオフに関する評価," 電子情報通信学会技術研究報告 (VLD2017-26), Vol.117, No.97, pp.119-124, 2017.

[77] 佐藤 真平, 右近 祐太, 高橋 篤司, "典型的な回路を用いた近似演算における入力系列の演算精度への影響の調査," 電子情報通信学会技術研究報告 (VLD2016-95), Vol. 116, No. 415, pp. 165-170, 2017.

[78] 井上 雅文, 右近 祐太, 高橋 篤司, "ゲートレベルシミュレーションによるエラー検

出・回復方式回路の評価," 電子情報通信学会技術研究報告 (VLD2010-141), Vol.110, No.432, pp.147-152, 2011.

[79] 秋田 大, 安藤 健太, 高橋 篤嗣司, "動的遅延分布の高速な見積もり手法," 電子情報通信学会技術研究報告 (VLD2012-55), Vol. 112, No. 245, pp. 83-88, 2012.

[80] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete Cosine Transform," IEEE Transactions on Computers, Vol. C-23, No. 1, pp.90-93, 1974.

[81] L. Pillai, "Video Compression Using DCT," Application Note: Virtex-II Series, 2002.

[82] Synopsys, Inc., "Find SoC Bugs Earlier & Faster, Bring-Up Software Earlier & Validate the Entire System," https://www.synopsys.com/verification.html, accessed on December 31, 2020.

[83] Synopsys, Inc., "The Technology Behind 90% of FinFET Designs," https://www.synopsys.com/implementation-and-signoff.html, accessed on December 31, 2020.

[84] J. Ferryman, J. Crowley, and A. Shahrokni, "PETS 2009 Benchmark Data," http://www.cvg.reading.ac.uk/PETS2009/a.html, accessed on December 31, 2020.

著者発表文献

論文誌

1. Y. Ukon, K. Yamazaki, and K. Nitta, “Real-time Image Processing based on Service Function Chaining using CPU-FPGA architecture,” IEICE Transactions on Communications, Vol. E103-B, No. 1, pp. 11-19, 2019.
2. Y. Ukon, S. Sato, and A. Takahashi, “Design Method of Variable-Latency Circuit with Tunable Approximate Completion-Detection Mechanism,” IEICE Transactions on Electronics. (in press)
3. S. Sato, E. Sassa, Y. Ukon, and A. Takahashi, “A Low Area Overhead Design Method for High-Performance General-Synchronous Circuits with Speculative Execution,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E102-A, no. 12, pp. 1760-1769, 2019.

国際会議

1. Y. Ukon, K. Yamazaki, and K. Nitta, “Video Service Function Chaining with a Real-time Packet Reordering Circuit,” Proceedings of 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-5, 2018.
2. S. Sato, E. Sassa, Y. Ukon, and A. Takahashi, “A Low Area Overhead Design for High-Performance General-Synchronous Circuits with Speculative Execution,” Proceedings of 2019 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-5, 2019.

研究会

1. 右近 祐太, 井上 雅文, 高橋 篤司, 谷口 研二, “エラー検出回復方式における加算器の性能評価,” 電子情報通信学会技術研究報告 (VLD2009-121), Vol. 109, No. 462, pp. 133-138, 2010.
2. 右近 祐太, 井上 雅文, 高橋 篤司, 谷口 研二, “FPGA 上に実現した可変レイテンシ回路の動作検証,” 電子情報通信学会技術研究報告 (VLD2010-142), Vol. 110, No. 432, pp. 153-158, 2011.
3. 右近 祐太, 安藤 健太, 高橋 篤司, “FPGA 上に実現した可変レイテンシ回路の性能評価,” 電子情報通信学会技術研究報告 (VLD2011-141), Vol. 111, No. 450, pp. 127-132, 2012.
4. 右近 祐太, 佐藤 真平, 高橋 篤司, “演算器の可変レイテンシ化による処理性能と回路面積のトレードオフに関する評価,” 電子情報通信学会技術研究報告 (VLD2017-26), Vol. 117, No. 97, pp. 119-124, 2017.
5. 井上 雅文, 右近 祐太, 高橋 篤司, “ゲートレベルシミュレーションによるエラー検出・回復方式回路の評価,” 電子情報通信学会技術研究報告 (VLD2010-141), Vol. 110, No. 432, pp. 147-152, 2011.
6. 佐藤 真平, 右近 祐太, 高橋 篤司, “典型的な回路を用いた近似演算における入力系列の演算精度への影響の調査,” 電子情報通信学会技術研究報告 (VLD2016-95), Vol. 116, No. 415, pp. 165-170, 2017.

ポスター・全国大会など

1. 右近 祐太, 高橋 篤司, 谷口 研二, “加算器におけるクロック周期に応じた遅延エラー率の評価,” 電子情報通信学会技術研究報告 (ICD2009-91), Vol. 109, No. 336, pp. 77-81, 2009.
2. 右近 祐太, 井上 雅文, 高橋 篤司, 谷口 研二, “エラー検出回復方式を用いた演算器の性能評価手法,” VDEC デザイナーズフォーラム 2010, 2010.
3. 井上 雅文, 右近 祐太, 高橋 篤司, 谷口 研二, “エラー検出回復方式回路の回路構成と性能に関するシミュレーション評価,” DA シンポジウム 2010 論文集, 情報処理学会シンポジウムシリーズ, Vol. 2010, No. 7, pp. 123-128, 2010.

特許

1. 右近 祐太, 山崎 晃嗣, “経路選択装置、ネットワークシステム、経路選択方法、およびプログラム,” 特開 2018-33016, 2018-3-1.
2. 右近 祐太, 吉田 周平, 山崎 晃嗣, “データ処理装置、ネットワークシステム、パケット順序制御回路、およびデータ処理方法,” WO2018/159677, 2018-9-7.
3. 右近 祐太, 吉田 周平, 山崎 晃嗣, “ネットワークシステム、データ処理装置、画像処理装置、画像処理方法、およびパケットデータ構造,” 特開 2018-148259, 2018-9-20.
4. 右近 祐太, 吉田 周平, 新田 高庸, “アクセス制御方法、アクセス制御装置、およびデータ処理装置,” 特開 2020-77088, 2020-5-21.
5. 吉田 周平, 右近 祐太, 山崎 晃嗣, 新田 高庸, “フロー制御装置および方法,” 特開 2019-140553, 2019-8-22.
6. 吉田 周平, 右近 祐太, 山崎 晃嗣, 新田 高庸, “パラメータ最適化装置、方法、およびプログラム,” 特開 2019-215697, 2019-12-19.