修士論文

FPGA を用いたリアルタイム AI カメラシステムの設計と評価

Design and evaluation of a real-time AI camera system using FPGA

報 告 者

学籍番号: 1235119

氏名: 亀阪 亮紀

指 導 教 員

星野 孝総 准教授

令和 3 年 2 月 12 日

高知工科大学 電子・光工学コース

目 次

第1章　はじめに

1.1.　研究背景

深層学習（Deep Learning）は人工知能（Artificial Intelligence）を支える大きな技術であり，様々な分野に応用され始めている．画像認識，音声認識，自然言語処理などの様々な分野で深層学習が適用され，日常生活やビジネスシーン等のあらゆる分野の様々なアプリケーションに用いられている [1, 2, 3]．その中でも，特に近年では，IoT（Internet of Things）デバイス等の組み込みシステムでの利用が期待されている [4, 5, 6]．画像認識分野においては，畳み込みニューラルネットワークにより，多くのタスクで高い精度を示している．これは，2012 年の大規模画像データセットを用いた画像認識コンペティション（Large Scale Visual Recognition Challenge:ILSVRC）で Hinton 教授らのチームが深層学習を用いて画像物体識別タスクにおいて，非常に高い認識精度を示したことが背景にある [7, 8]．特に，ImageNet などの大規模画像データセットを用いて学習した畳み込みニューラルネットの中間層から抽出される特徴量は非常に汎用性が高くさまざま分野・領域で利用可能であることが示唆されている [9, 10]．本研究においては，深層学習の中でも，画像認識分野の畳み込みニューラルネットワークにターゲットをおくこととした．

次に，組み込みシステムについて述べる．組み込みシステムへの要求事項として，低消費電力化等に対するリソース制約，機械や機器の制御といった観点から誤動作が人命にかかわる事態になりうるため，高い信頼性があげられる．また，処理時間においても，単に計算速度が早く，レスポンス時間が短いだけではなく，リアルタイム性が求められる．制御システムの対象となる機器によって定められている要求時間を満たして動作する必要がある [11, 12]．そのため，計算コストが高い深層学習を IoT デバイス等へ実装することは困難である [13, 14]．特に現在の畳み込みニューラルネットワーク技術によるリアルタイムのオブジェクト検出には GPU（Graphics Processing Unit）が不可欠であり，ほとんどのコンピューティングライブラリは，GPU システムを使用することを前提に設計されている [15, 16, 17]．そのため，畳み込みニューラルネットワークの高速化には一般的に GPU が必須となる．畳み込みニューラルネットワークの最大の処理は，GPU アーキテクチャによる畳み込み処理と行列演算である [18, 19, 20]．さらに，高速実行には，高速なメモリアクセスと積和演算の並列計算パフォーマンスが必要である．そのため，厳しいリソース制約とリアルタイム性が要求される組み込みシステムにおいては，その計算のためのストレージや複雑度，消費電力が応用範囲を制限している．

また，その一方で，近年では，自動車やセキュリティーシステム，工業製品の欠陥検査などの組み込み機器における画像処理分野に FPGA が応用されており，日々重要性が高まっている．FPGA は Field Programmable Gate Array の略で既にチップ化されている LSI である．しかし，チップ内部は構成されておらず，ユーザーが任意の回路をハードウェア記述言語（Hardware Description Language: HDL）を用いて設計し，プログラム

をして書き込むことができる．組み込み機器における深層学習のアクセラレータとして，CPU（Central Processing Unit）・GPU・ASIC（Application Specific Integrated Circuit）・FPGA 等が挙げられる．それぞれの特徴について表 1.1 に示す [21, 22, 23]．FPGA は消費電力の観点から CPU や GPU に比べエネルギー効率が高く，パフォーマンス性能が高い特徴がある．また，FPGA の書き換え可能という観点から ASIC と比較するとアルゴリズムの変更にも柔軟に対応することができる．そこで，本研究では，FPGA を用いて深層学習のハードウェア化を目指す．

表 1.1: 深層学習向けアクセラレータの特徴比較

	CPU	GPU	FPGA	ASIC
Cost	◯	◯	△	×
Power Consumption	△	×	◯	◯
Ease of development	◎	◯	◎	×
Data Control(Bit width operation)	◯	×	◎	◎

ハードウェアアクセラレーションは，処理速度を向上させる手法の 1 つであり，画像処理の分野でよく用いられる [24, 25, 26]．これらに関する FPGA の研究は多数あり，FPGA を使用してハードウェア画像処理フィルターを実装することにより，CPU と比較してパフォーマンスが向上するという報告が数多くされている．Jeremy らの報告 [27] によると，フィルター処理を FPGA に実装し，CPU や GPU と性能比較すると，画像サイズが大きい場合 FPGA の方が GPU よりも優れた性能を示している．更には，消費電力においても，CPU や GPU と比較して優れた性能を示している．これは，画像処理フィルターや特徴抽出などの各処理は通常，独立しているためである．特にカメラからの RAW データを直接処理する場合，GPU や CPU では一旦全データをメモリに格納した後からしか処理できないが，カメラからの同期信号に合わせたリアルタイム処理を行う場合，FPGA はカメラから転送されてくるデータをメモリに入れることなく，流し込み処理（ストリームデータ処理）のパイプライン処理構造にすることで高速化が期待できる [28]．特に，本研究においてターゲットとなっているニューラルネットワークの演算は，データフロー構成の演算と親和性が高く，メモリに入れない FPGA を用いたパイプライン演算に適している．カメラ信号処理の FPGA を用いた関連研究として，宮口らの報告 [29] によると，1024 個の PE（Processing Elements）を統合することにより，標準の DSP を超える高速処理速度を実現する汎用画像処理プロセッサ（Scan-line Video Processor：SVP）を提案し，EDTV2(The 2nd generation Enhanced Definition TV) のデコーダーへの応用を示している．

1.2. 研究目的

本研究では，FPGA を用いて畳み込みニューラルネットワークのハードウェア化を目的とした．畳み込みニューラルネットワークの画像処理演算部を FPGA 向けにメモリアクセスを削減・効率化しパイプライン化することで，演算コストを減らす．これにより，CPU が非力なエッジマシンでカメラデバイスを用いたリアルタイム AI 画像処理を実現することを目指す．

次に，本稿における実装対象とするターゲットアプリケーションについて述べる．本研究では，野外に設置されたカメラ画像を用いて，人工知能システムによる多様な画像識別を行うことを想定している．一般的な多クラス分類では，図 1.1 に示すように GPU を用いたアプローチが一般的で，電力問題から，エッジ側では処理を行わずネットワークを介して AI サーバー側で動作させる．しかし，この方法では，ネットワークがボトルネックとなり，リアルタイム性を損なわせてしまう．また，サーバー側では，カメラ情報を一旦サーバーのメインメモリに格納し，その後，パラメータを GPU メモリに転送してから計算が始まるため，サーバー側では，高速なメモリ確保と転送が求められる．これに対して，図 1.2 に示すように，AI の処理を前処理とし，サーバー側では，エッジ側での推論結果等のデータを用いて学習により，AI を更新する構成にする．このように，カメラデバイス側で推論処理させることで，処理の大幅な高速化が期待できる．

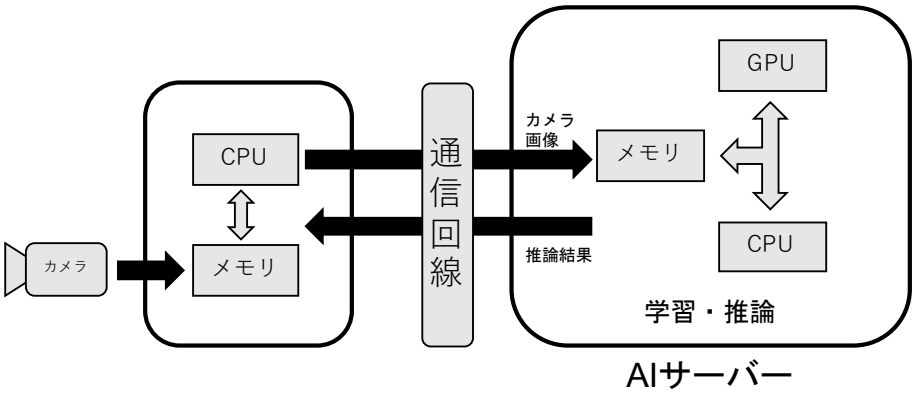


図 1.1: 従来型 AI カメラシステム

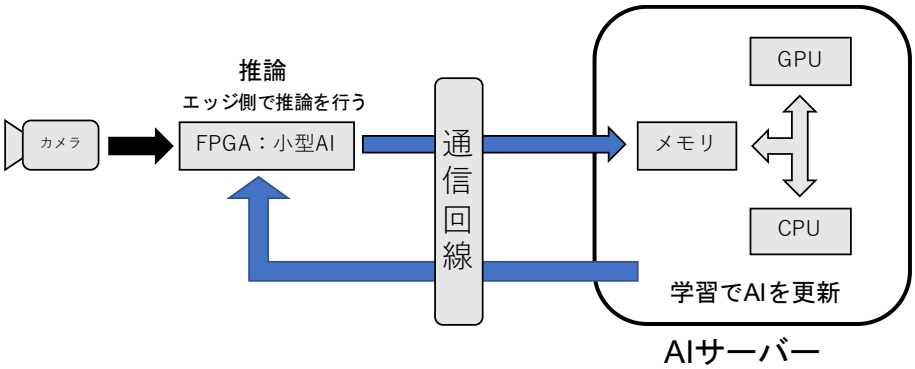


図 1.2: 提案する AI カメラシステム

本稿では，高知県大豊町への設置を想定した AI 搭載型害獣捕獲システム [30]（図 1.3）をターゲットアプリケーションと設定した．本害獣捕獲システムは，太陽光パネルと鉛蓄電池のみで駆動するスタンドアローンシステムであり，消費電力の制約がとても大きい．現状のシステムでは，複数の低消費電力マイコンを使用し，システム内の役割を細かく分割し，必要なときにそれぞれのマイコンが起動して現地では必要最小限の処理のみを行う仕組みになっている．また，低消費電力マイコン上で深層学習を用いることは現実的ではないため，人工知能システムを動作させるサーバー及び Web サーバーは別の基地局へ設

置を行っており，学習から推論まで全て携帯電話回線を用いて，ネットワーク経由で行っている．本システムの現状は試作段階ではあるが，運用を考慮した際，多数のデバイスを用いているため，メンテナンスなどが複雑になることが予想される．本研究では，これらの人工知能システムの推論部分・複数の低消費電力マイコン内での画像処理部を電力効率に優れる SoC FPGA を用いてワンチップ化を行い，処理を全て現地のエッジ側で行うようにすることで，システムの性能向上及び，利便性・運用性の向上を目指す．

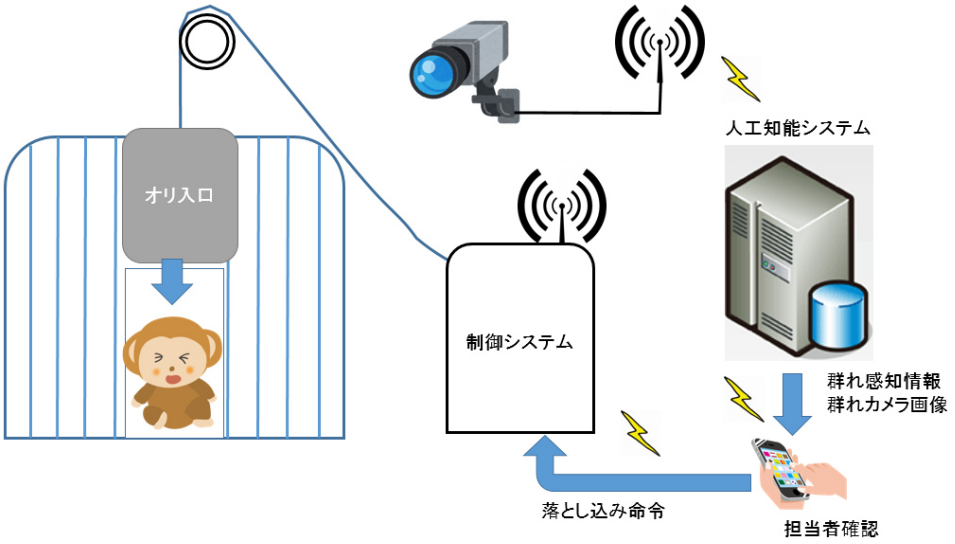


図 1.3: 本研究におけるターゲットアプリケーション

1.3. 本論文の構成

本稿の構成は次の通りである．第 2 章では，プロセッサ別の処理方法の違いについて述べ，本研究におけるリアルタイム処理の定義を行い，本研究用に設計した SoC FPGA による画像処理システムのベースとなる設計概要について述べる．そして，第 3 章では FPGA を用いて畳み込み層を設計し，実際のカメラ信号を用いて，レイテンシーを計測し，リアルタイム性の検証，また，CPU との処理時間の比較を行った実験について述べる．第 4 章では畳み込みニューラルネットワークの全結合層を想定した，多層ニューラルネットワークを実装し，予測時間の検証を行った実験について述べる．第 5 章では畳み込みニューラルネットワークの前処理として対象物体の周辺の抽出を行う背景差分法・ラベリング処理を FPGA 上に実装した結果について述べる．最後に第 6 章では本稿についてのまとめを述べる．そして，今後の研究課題と展望について述べる．

第2章　SoC FPGAによるリアルタイム画像処理システムの設計概要

2.1.　アーキテクチャによる処理方法の違い

本章では，FPGA・CPU・GPU それぞれの処理方法の違いについて述べる．図 2.1 に CPU と FPGA との処理方法の違いを示す．カメラからの RAW データを直接処理する場合，CPU では一度全てのカメラデータをメインメモリに保存する必要があり，1 つの処理毎にメモリを経由して処理が行われる．しかし，FPGA を使った処理ではその必要がなく，カメラからのデータを直接処理できるため，メモリへのアクセス時間を短縮できる．したがって，レジスタ転送レベルで画像処理を実行することが可能で，カメラクロックに同期した画像処理を行うことができる．このように，カメラから転送されてくる画像データをメインメモリに格納する前に画像処理専用回路で処理を行うことで，元画像をメインメモリへ転送終了後，ほぼゼロに近いレイテンシーで画像処理結果を得ることができる．

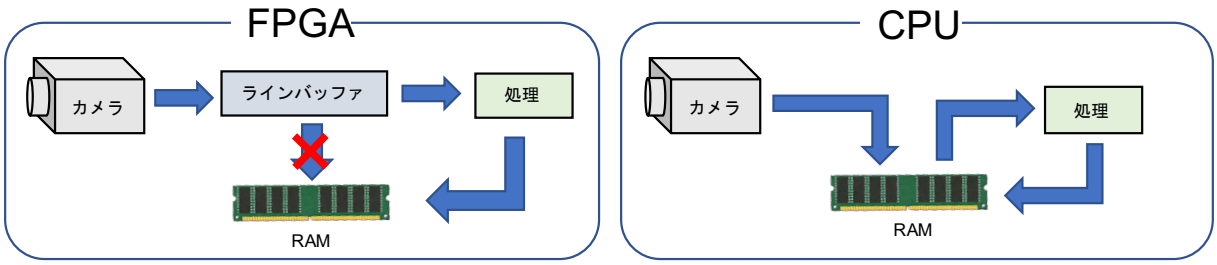


図 2.1: CPU と FPGA の処理方法の違い [31]

次に，GPU の処理方法について述べる．図 2.2 に GPU を用いた処理方法の概要図を示す．GPU を用いた演算の場合も同様に，入力画像を縮小処理した上で，GPU 上のビデオメモリ（Video RAM: VRAM）を確保（Malloc）してニューロンや畳み込みのパラメータ群と縮小した画像データを展開してから演算を行う．この縮小，メモリ確保，パラメータの展開に一般的に 1〜100ms の時間を必要とする．また，メモリへ完全に画像データを置くためには，カメラの速度に依存するが，一般的に 30〜50ms の時間を必要とする．

つまり，CPU や GPU 自体の動作速度が高速であっても，メモリアクセスがボトルネックとなり処理性能と落としている．本研究では，FPGA を用いて，この CPU・GPU のメモリアクセス不要のアーキテクチャを設計し，CPU・GPU のメモリアクセスを処理時間に割り当てることで，画像処理の高速化を目指す．

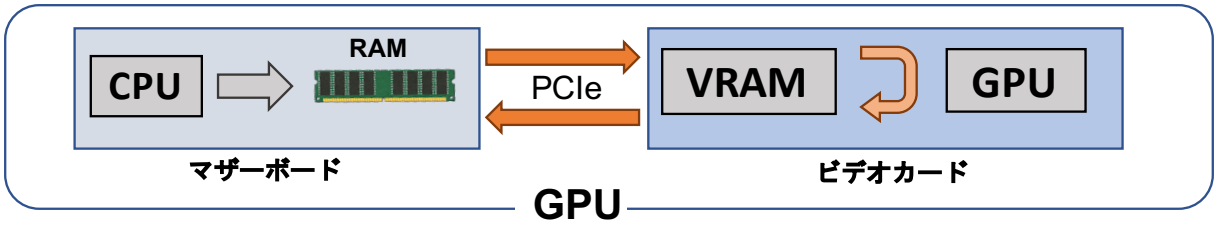


図 2.2: GPU の処理方法

2.2. 本研究におけるカメラデバイスを用いた場合のリアルタイム処理の定義

次に，本研究におけるカメラデバイスを用いた場合のリアルタイム処理について述べる．図 2.3 に本研究におけるリアルタイム処理について示す．本研究の目的は，第 1.2 章でも述べた通り，カメラデバイスを用いたリアルタイム AI 画像処理である．そこで，本研究では，カメラが N 番目の画像 1 フレームの転送が終了する前に，N-1 番目の画像処理を終えていることをリアルタイム処理と定義する．このように定義することで，カメラのフレームレートを守りながらフレーム落ち無しに処理を行うことが実現している状態になる．つまり，カメラのフレーム落ちがないため，図 2.4 のように，畳み込み処理後の結果をカメラデータとして扱うことができ，畳み込みカメラシステムを構築できる．そこで，本研究では，図 2.5 に示すようにカメラの取り込み終了時から，処理が完了するまでの時間（実用的遅れ時間）をレイテンシーと定義し計測を行った．これを実現するための手法として，本研究ではパイプライン処理を用いた．これを用いることで，カメラデータを取り込みながら画像処理を行うことが可能である．

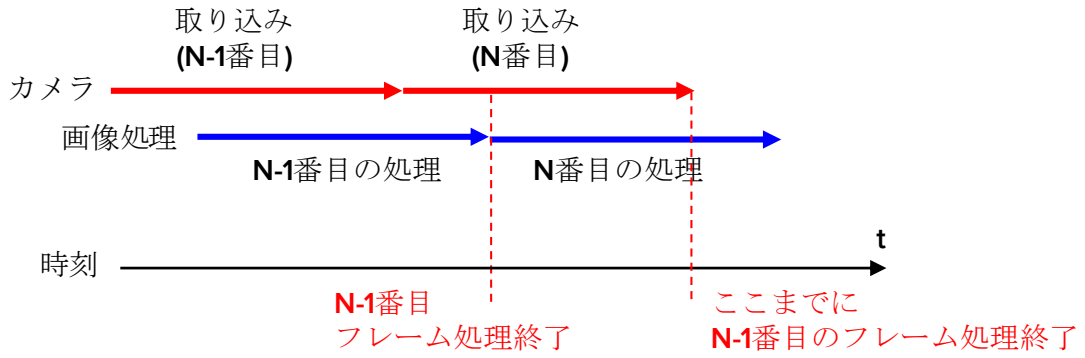


図 2.3: リアルタイム処理の定義

2.3. パイプライン処理とラインバッファ構造

本章では，第 2.2 章で述べたリアルタイム処理を実現するために用いたパイプライン処理について述べる．図 2.6 に代表的な処理方式の概要を示す．時系列で処理を一つずつ実行する方式を逐次処理と呼ぶのに対して，並列処理は同時刻に複数の処理を実行する方式

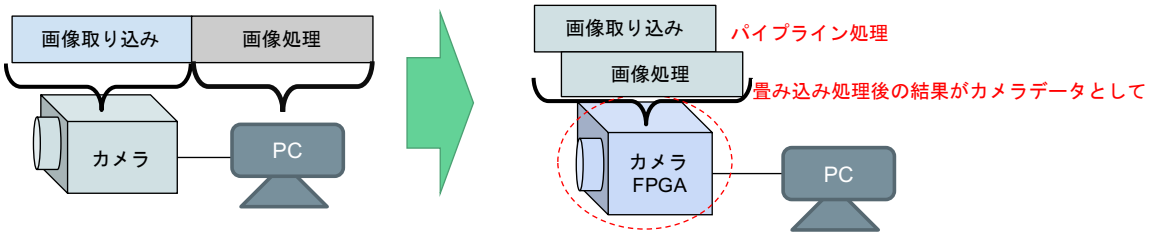


図 2.4: リアルタイム画像処理カメラシステムの概要

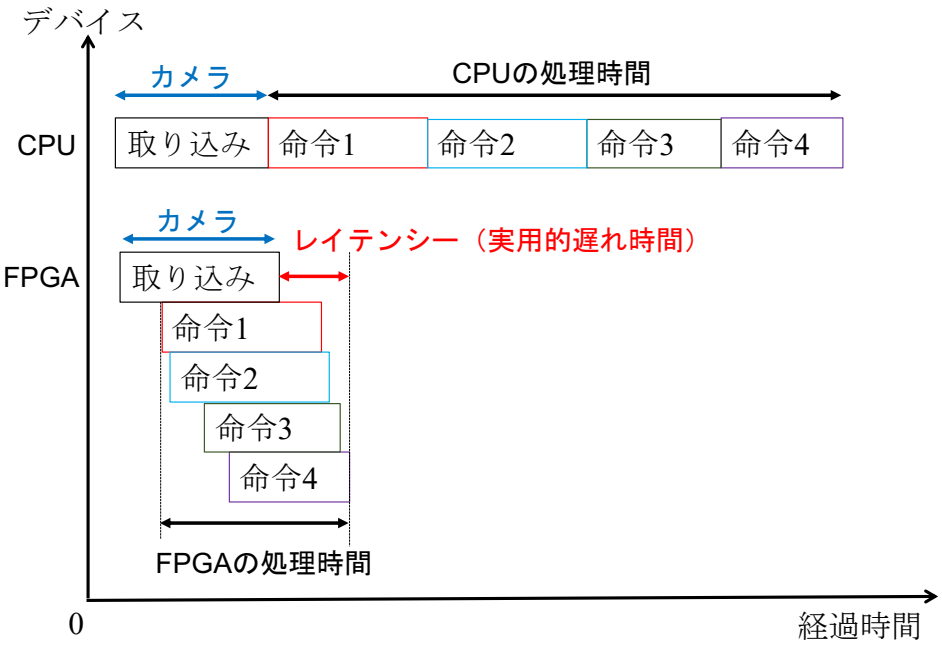


図 2.5: 本研究におけるレイテンシーの定義

である．パイプライン処理とは例えば命令１の処理結果が命令２の入力データであった場合，全てのデータに対して命令１を実行し終えるのを待つのではなく命令１の処理が終わったデータに対して順次命令２を実行する方式である．

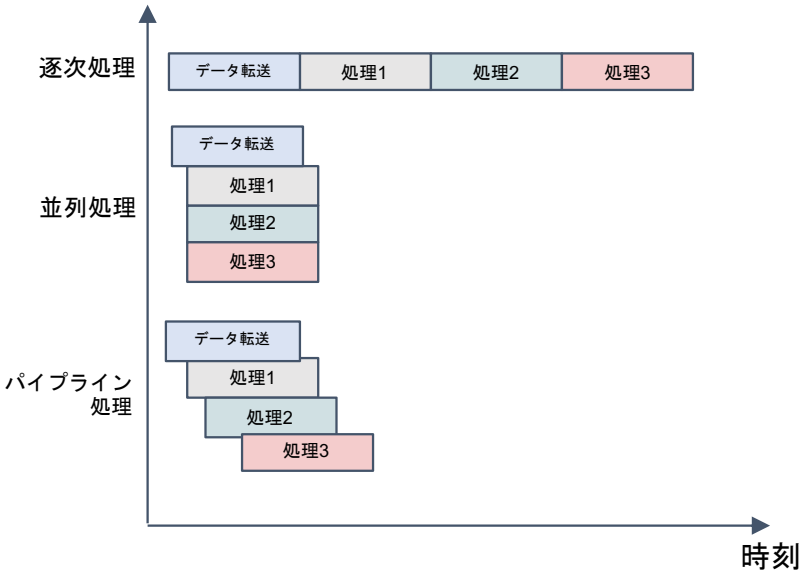


図 2.6: 代表的な処理方式

パイプライン処理を画像処理フィルターで実現するために，本研究ではラインバッファ構造を使用した．ラインバッファとは横 1 ライン分の画素を格納する先入れ先出し（Fisrt In First Out）型のシフトレジスタで，画像処理を行うフィルターが 3×3 の場合，ラインバッファを 3 本用意する必要がある．また，3 本用意した場合，ウィンドウサイズが 1×1 から 3×3 のフィルター処理に対応することができる．そのため，あらかじめ想定されるフィルターの最大サイズ分の本数だけラインバッファを用意することで画像処理システムの変更にも対応することができる．また，ウィンドウサイズが 3×3 のフィルター処理の場合，ラインバッファに 3 本分のデータを格納してから処理が始まるため，開始後，画像 3 ライン分の画素データ格納時間の遅れは必ず生じる．

次に，ラインバッファ構造の動作についての詳細を述べる．図 2.7 に画像の横サイズ 640 で，ウィンドウサイズが 3×3 画像処理フィルター処理を行う場合のラインバッファの動作概要を示す．ウィンドウサイズが 3×3 のフィルターで，画像の横サイズ（1 ライン）が 640 であるため，640 ワードの画素データを格納できるラインバッファを 3 本用意している．カメラから転送されてくる 1 画素分のデータを Buf0 に格納する．それと同時に，Buf0 の 0 番地に格納されていた画素データを Buf0 の 1 番地にシフトさせる．同様に Buf0 の 1 番地に格納されていた画素データを Buf0 の 2 番地にシフトさせる．この隣の番地に画素データをシフトさせる処理を全番地に対して行う．カメラから 1 画素分のデータが転送されてくるたびに破線枠で囲った部分の画素データに対してフィルター処理を実行する．このフィルター処理は，ラインバッファの回路と独立して並列動作するため，バッファの転送を待つ必要がない．ラインバッファに格納された画素データ群は，カメラクロックに同期してシフトしていくため，フィルター窓を移動させる必要はない．フレーム単位ではなくピクセル単位でフィルター処理を実行でき，処理の高速化を図っている．このように，カメラから 1 画素分のデータが転送されて来るタイミングで行うパイプライ

ン処理構造にすることで，カメラ 1 画素ずつの速度に合わせて処理後のデータを 1 画素ずつ出力する構造にできる．したがって，カメラからデータを取り込みながら画像処理を行うことができる．

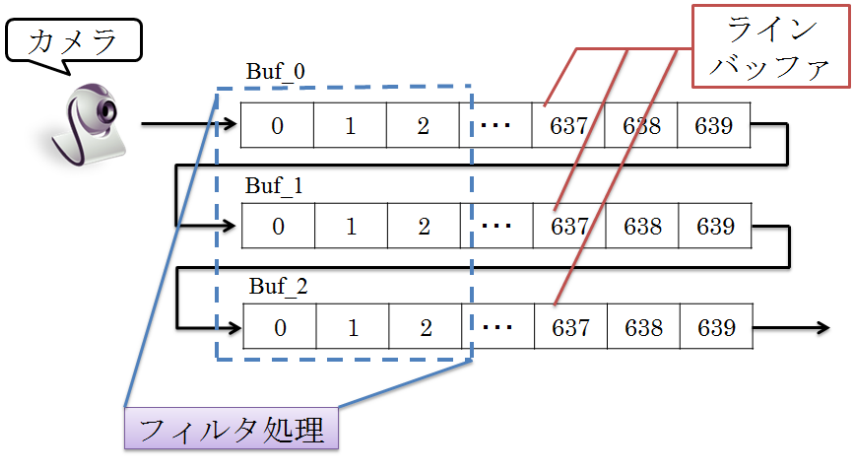


図 2.7: ラインバッファ構造 [32]

2.4. SoC FPGA を用いた画像処理システム

2.4.1. SoC FPGA

本研究では，SoC FPGA という LSI を用いた．SoC FPGA とは ARM プロセッサー，ペリフェラル，メモリー・インターフェイスで構成される HPS（Hard Processor System）ファブリックと FPGA ファブリックが同じチップに内包された LSI である．汎用ハードコアを多く兼ね備え，1 チップでシステム全体を構築可能で，省スペース・省電力である．FPGA の設計手法として，直接ハードウェア記述言語（HDL：Hardware Description Language）で設計する方法と C 言語等からの高位合成（High Level Synthesis）による設計方法がある．高位合成は設計者が C 言語や Python 言語での処理を実現するという点では設計がしやすいが，FPGA の性能を最大限に発揮するハードウェアを生成することはできないという報告がある [33, 34]．本研究では，高位合成による設計は行わず，HDL で設計する手法を前提として行った．

本研究で用いた SoC FPGA は図 2.8 に示す Intel 社（旧：Altera 社）の Cyclone V SoC (5CSEMA5F31C6)搭載の DE1-SoC である．Cyclone V SoC の HPS 部は Dual-Core ARM-A9 プロセッサーや 1GB の DDR3 SDRAM，Ethernet 等が搭載されており，Linux OS を FPGA とは独立し単独で動作させることが可能である．そのため，オープンソース画像処理ライブラリである OpenCV を使用することが可能である．つまり，FPGA の出力結果を HPS 部で OS 上の優れた GUI や OpenCV 画像処理ライブラリを使用して確認可能なシステムの構築が可能である．これにより，画像処理システムを機能モジュール単位に分割し，各機能モジュールをソフトウェア・ハードウェアに分割するハードウェア・ソフトウェア協調設計を行うことできる．レイテンシーのボトルネック部分をハードウェア化することでシステム性能の向上が期待できる．通常，ソフトウェア上で行う画像処理システ

ムにおいては，ハードウェアベースのシステムに比べシステムのデバッグが容易である．本研究でターゲットとしている畳み込みニューラルネットワークにおいては，畳み込みフィルターの係数・プーリング方式の変更・全結合層のニューロン等の重みパラメータの変更をソフトウェア側に割り振ることでより使いやすい画像処理システムの構築が可能である．HPS 部と FPGA 部の間は HPS Bridge と呼ばれるバスで接続されており，双方向でデータを自由にやりとりすることが可能である．

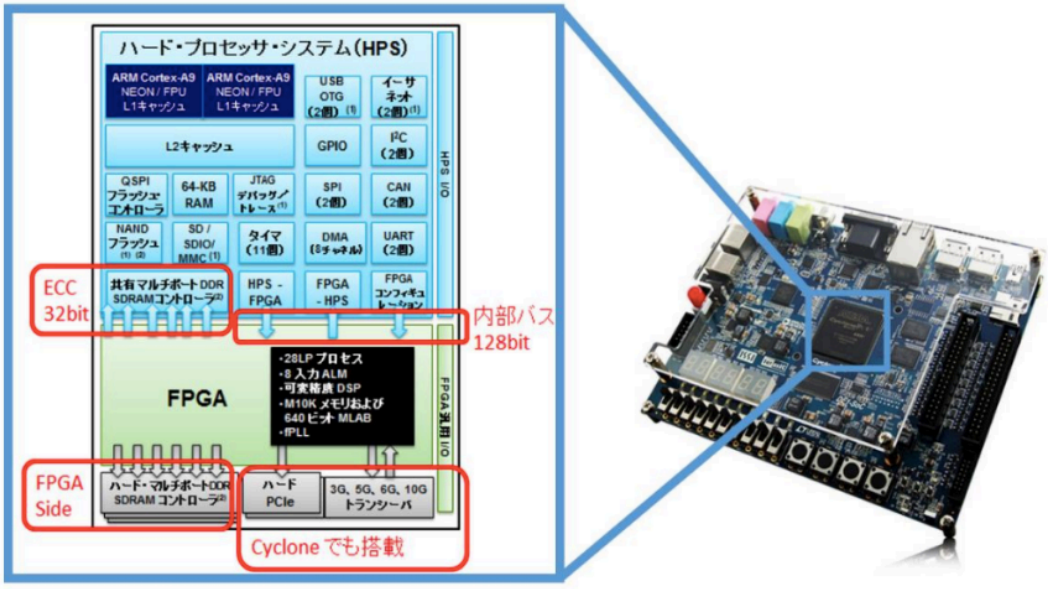


図 2.8: SoC FPGA の概要図

2.4.2. SoC FPGA の開発手法

図 2.9 に SoC FPGA の開発手順の概要を示す．まず，独自の IP コアの設計を行い，Qsys システムを用いて Intel 社が提供している IP コアとこれらを接続し統合システムを設計する．その後，Quartus を用いてこれら全てをまとめてコンフィグレーションを行い，回路情報である sof ファイルを生成する．Linux イメージは開発元の Intel が Ubuntu ベースの専用 OS を提供しており，Linux ファイルシステム・FPGA の回路情報が書き込まれた rbf ファイル・dtb(device tree blob) を SD カードに書き込みブートさせることで Linux が起動する．SD カードから，U-Boot システムでブートされ，rbf ファイルがロード・コンフィグレーションされ，設計した回路が FPGA 上に構成される．その後，Linux カーネルイメージに制御を移すことで，Linux が起動し，設計した IP コアのレジスタがメモリ上にマップされている．FPGA の回路情報は rbf ファイルに含まれているため，sof ファイルを rbf ファイルに変換を行い，再度コピーすることで，回路情報を書き換えることができる．この rbf ファイルが存在する SD カード内のパーティションを Linux 側でマウントしておくことで，ネットワーク経由で SoC FPGA の回路そのものを書き換えることも可能になる．

図 2.10 に Cyclone V SoC の概要図を示す．HPS 内部には FPGA 内部のロジックと共有されるマルチポート・メモリ・コントローラが構成されており，FPGA ファブリック

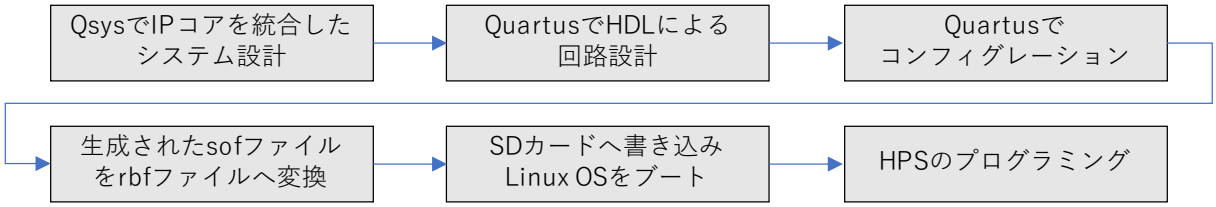
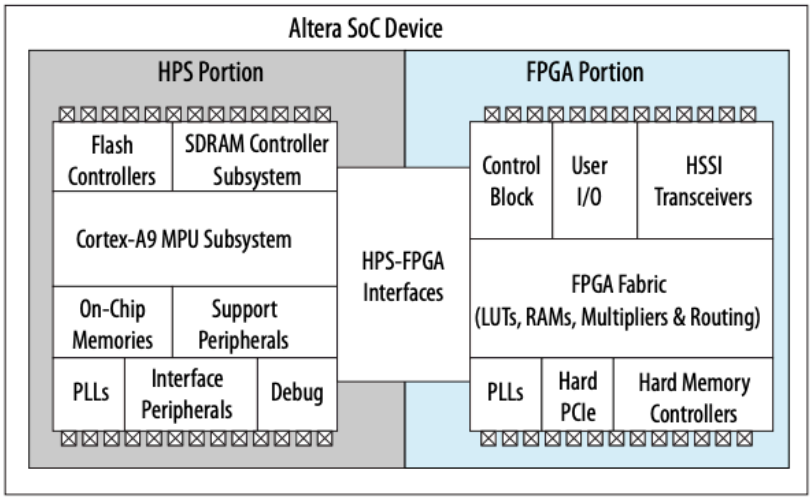


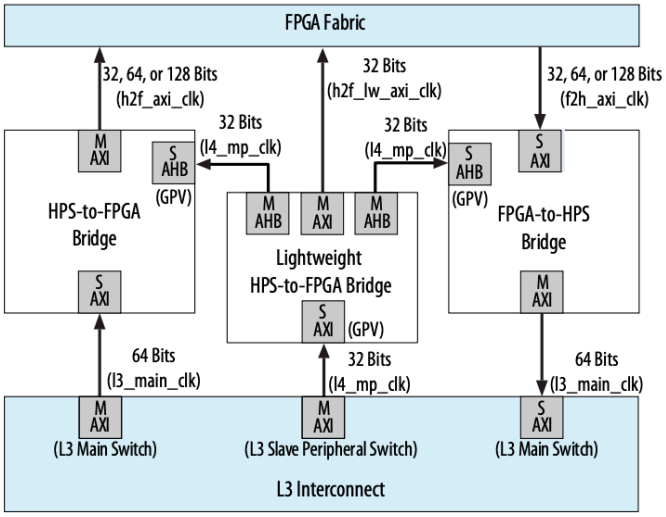
図 2.9: SoC FPGA の開発手順

との接続は，HPS-to-FPGA Bridge（H2F），FPGA-to-HPS Bridge（F2H），Lightweight HPS-to-FPGA Bridge（LWH2F），FPGA-to-HPS SDRAM Interface（F2S）で構成されている．FPGA-to-HPS SDRAM Interface 以外のインタフェースはそれぞれ，L3 Interconnect と FPGA ファブリックに AXI バスで接続されている．これらは，Qsys で設計することが可能で，FPGA 側から Qsys で設計するときは Avalon Memory-Mapped ブリッジとして扱える．Qsys システムとは容易な GUI インタフェースにより，IP 機能とサブシステムを素早く統合でき，インタコネクト・ロジックを自動生成するシステムである．これらはすべて，マルチポート・メモリ・コントローラでマッピングされているのでメモリマップド I/O 方式を用いてメモリ空間へのアクセスし，コントロールできる．メモリマップド I/O 方式とはアドレス空間にメモリと入出力機器が共存し CPU からメモリにアクセスするだけで入出力機器にアクセスできる方式である．アクセスの際，ARM 側から見た FPGA のコンポーネントアドレスは図 2.11 に示すように，FPGA-HPS ブリッジのベースアドレス + Qsys 上のオフセットアドレスで計算することができる．FPGA-HPS ブリッジのベースアドレスは Lightweight HPS-to-FPGA Bridge，HPS-to-FPGA Bridge（H2F），FPGA-to-HPS Bridge（F2H）でそれぞれ異なる．また，アクセスの際は，MPU ビュー，FPGA-to-SDRAM ビュー，L3 スイッチビューのアドレス空間があり，どこからアクセスするかによって見え方が異なる．ARM プロセッサー上からアクセスする場合は MPU 領域を使用し，Lightweight HPS-to-FPGA Bridge 等に接続された FPGA 側のマスタは L3 領域が適用され，FPGA-to-HPS SDRAM Interface で接続されたマスタから使用する場合は FPGA-to-SDRAM 領域を使用する．FPGA-to-SDRAM ビューにおいても，アドレス空間は 4GB 存在するが，直接 SDRAM へアクセスするため，アクセスできるのはメモリの実体があるエリアのみである．つまり，本研究の場合，DE1-SoC は 1GB の SDRAM を有しているため，1GB（0x4000_0000）以降にはアクセスしてはいけない．

本研究では，Lightweight HPS-to-FPGA Bridge 及び FPGA-to-HPS SDRAM Interface を用いた．Lightweight HPS-to-FPGA Bridge は HPS がバス・マスタとなり FPGA 側にアクセスするインタフェースで，FPGA 側のレジスタコントロールなど，データの信頼性が求められる用途に用いられる．本システムでは，畳み込みのフィルター係数・プーリング方式・ニューロンの重み等の変更に使用した．FPGA-to-HPS SDRAM Interface は FPGA がバス・マスタとなり HPS 側の SDRAM へ直接アクセスできるインタフェースである．HPS 側のメイン・スイッチを介さず直接 SDRAM へアクセスできるため，HPS 側の SDRAM への高速アクセスが可能である．FPGA 側からアクセスする際のアドレスは，DDR3 SDRAM の物理アドレスを直接指定することでアクセス可能である．これを用いることで，FPGA 側が SDRAM に読み書きし，HPS 側も SDRAM を読み書き可能になり SDRAM のデータを FPGA と HPS で共有することができる．本システムでは，FPGA 側からカメラの画像やフィルター処理後の画像を転送するのに用いた．



(a) CycloneV SoC のインターフェース



(b) CycloneV SoC の AXI バスの詳細

図 2.10: Cyclone V SoC の詳細 [35]

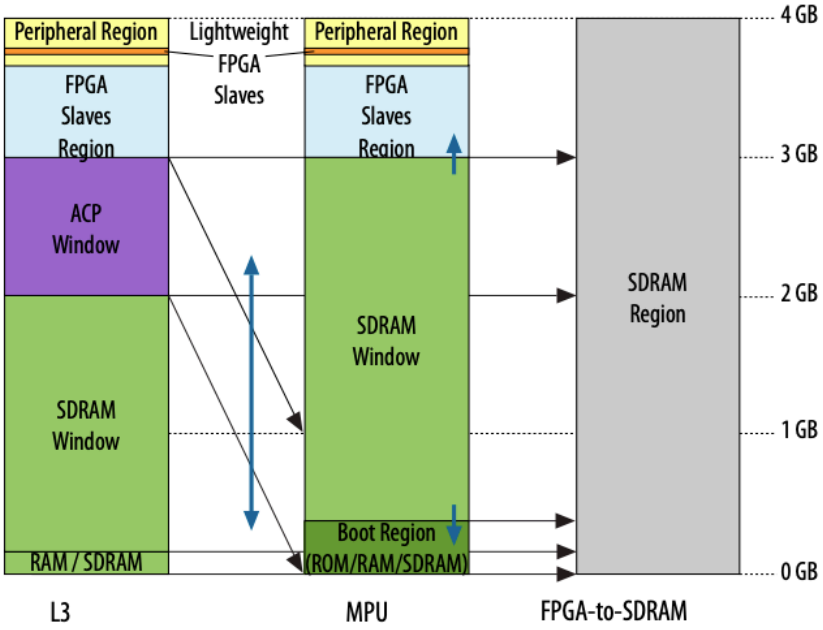


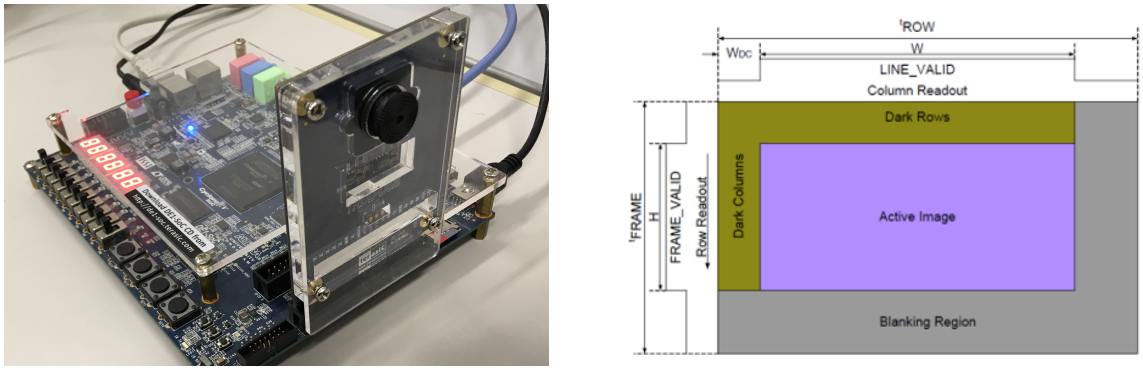
図 2.11: CycloneV SoC のアドレス空間 [35]

2.4.3. CMOS カメラ TRDB-D5M

本研究で使用した CMOS カメラの概要について述べる．図 2.12(a) に本研究で用いたカメラモジュールを示す．用いた CMOS カメラは Intel 社製 TRDB-D5M である．カメラの解像度，フレームレート等の動作モードは I2C 通信を用いてレジスタを制御することで設定可能である．本実験回路では FPGA 上に I2C 通信モジュールを設計しハードウェアベースで制御を行っている．解像度は最大 2,592×1,944 で，データは RGB ベイヤパターンで転送されてくる．フレームレートは最高解像度で最大 15fps，VGA 規格で最大 70fps である．本研究では，VGA 規格と SXVGA 規格でカメラ制御を行った．カメラモジュールの同期信号の詳細を図 2.12(b) に示す．垂直同期信号（VSync），水平同期信号（HSync），ピクセルクロック（PCLK）を用いて，データ 12bit を取得する．垂直同期信号と水平同期信号が High の時，ピクセルクロックの立ち上がりエッジに同期して画像データを取得する．ベイヤパターンを RGB カラー画像データとして扱うには，変換が必要であるが，本研究では，グレースケールで実験を想定し，ベイヤパターンの変換は行わず，取得された緑画素データ 12bit のうち，上位 8bit を RGB それぞれの画素値として扱うことでグレースケール画像とした．また，グレースケール画像の場合は 1 チャネルであるが，カラー画像を扱う場合には RGB の 3 チャネルが必要となり，リソースも 3 倍必要になる．

2.4.4. 画像処理カメラシステムの設計概要

本研究で構築したカメラシステムの概要について述べる．図 2.13 に構築したカメラシステムの概略図を示す [32, 36, 37]．カメラから転送されてくるデータは FIFO に取り込まれる．FIFO の動作概要を図 2.14 に示す．FIFO は First In First Out の略で先に入ったものから順番に先に取り出す出入りにおいて，順序が保存されるものである．これを用いることで書き込みと読み出しのタイミングの違いを吸収することができる．カメラデータ



(a) TRDB-D5M　　(b) 取り込みタイミング

図 2.12: カメラモジュール

はカメラクロックに同期して転送されてくるため，システム全体をカメラクロックで動作させた場合，システムの後方部になるにつれて，待ち時間が発生することが考えられる．そこで，カメラと FPGA 側で独立したクロックで動作させるため，書き込む側と読み出す側のクロック周波数が異なる DCFIFO を用いた．

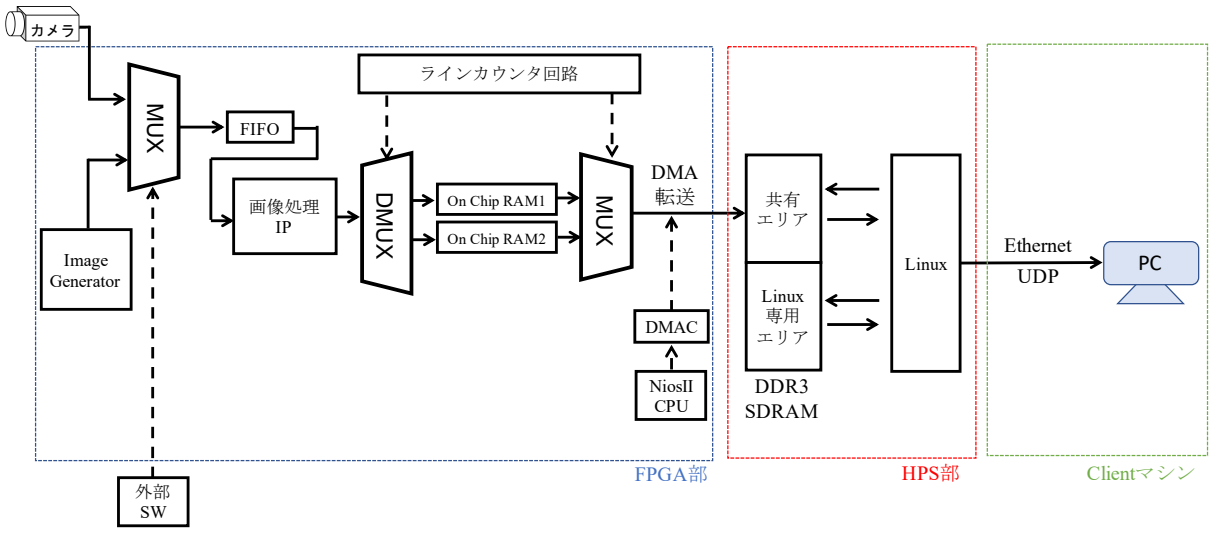


図 2.13: カメラシステムの概略図

FPGA 側では FIFO から取り出させたデータをラインバッファに格納し，その後画像処理 IP へ転送し，水平垂直画素に対してフィルター処理を行い，ダブルバッファに格納する．ダブルバッファには On Chip Memory（On Chip RAM）を用いた．On Chip Memory とは FPGA 上に埋め込まれている SRAM ベースの小容量・高速メモリのことで，制御が容易である．本研究では，FPGA の処理結果を一時的に格納するために用いた．1 ライン分の画素データを格納可能なサイズの On Chip Memory を用意し，1 ライン分のデータが格納させると，割り込み信号を発生させ，Nios CPU が，DMA Controller（DMAC）に転送命令を出し，HPS 経由で DDR3 SDRAM に転送させる．図 2.15 にダブルバッファ構造の概要を示す．ダブルバッファ方式を採用することで，片方のラインバッファが DMA 転送中にもう一方のラインバッファに処理結果を格納する事ができ，転送速度を考慮する事なく画像データを読み出す事が可能になる．DMA 転送とは，Direct Memory Access の略称

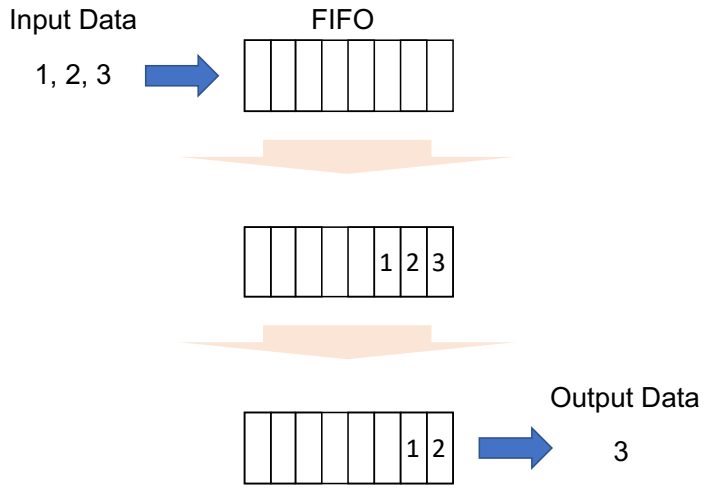


図 2.14: FIFO の動作概要

で CPU を経由せずにメモリのある区間を別のメモリ区間へ転送する技術である．図 2.16 に DMA 転送の概要を示す．一般的に CPU を経由する場合，メモリにアクセスし，データを読み込み，その後別のメモリエリアにデータを書き込む．このデータの読み書きには数クロックを必要とし，バス線も占有する．一方，DMA 転送の場合，図 2.16(a) に示すように CPU から，DMA Controller に転送容量・転送元アドレス・転送先アドレスを設定することで，図 2.16(b) のように DMA Controller は転送が終了するまで転送を続ける．本実験の場合，転送元が FPGA 上の On Chip Memory で，転送先は HPS 側の DDR3 SDRAM である．DDR3 SDRAM は F2S インタフェースで接続されているため DMA Controller の設定の際は，DDR3 SDRAM の物理アドレスを直接指定する．DMA Controller は Intel 社が公開している IP コアの DMA Controller をベアメタル方式で使用しており，HPS 側に内蔵されている DMA Controller は本システムでは用いていない．DMA Controller への転送命令を行う CPU は Nios ii/e プロセッサを用いた．Nios プロセッサとは，Intel 社の FPGA 上で動作する 32 ビットの組み込み用途マイコンである．

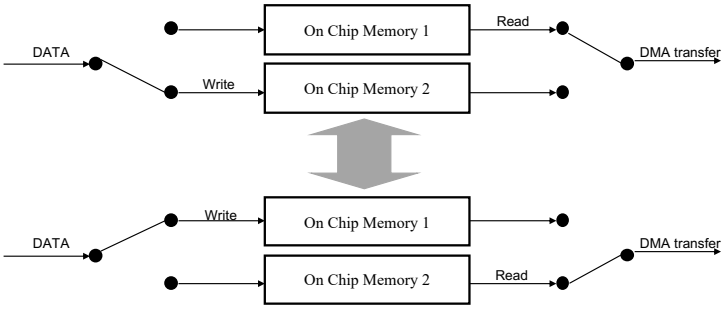
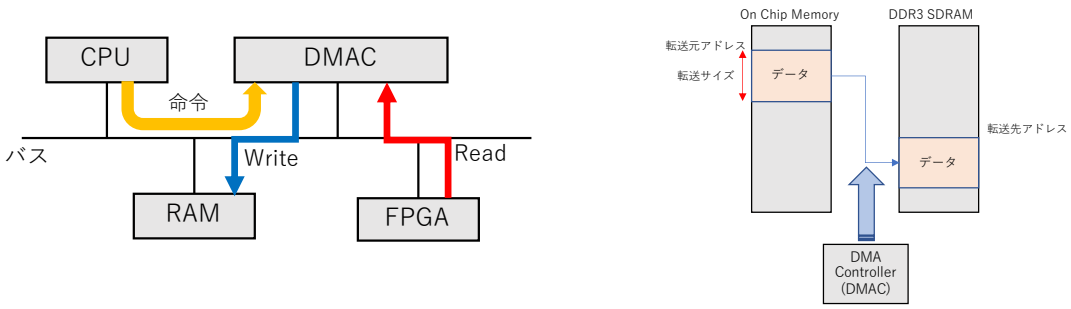


図 2.15: ダブルバッファ構造

DMA 転送を行う際の懸念点として，転送先の DDR3 SDRAM アドレスは Linux カーネルが使用するエリアと同じメモリ空間を利用してはならないことが挙げられる．使用するメモリエリアが被った場合，データが破壊され Linux がカーネルパニックを起こす可能性が考えられる．そのため，Linux が使用するメモリエリアを予め制限する必要がある．そこで，本研究では，1GB の DDR3 SDRAM の先頭 512MB は Linux 専用のエリアとし，後ろ 512MB を FPGA が使用する Linux との共有エリアとして扱うことにした．U-Boot



(a) DMA 転送の手順　　(b) DMA 転送

図 2.16: DMA 転送の概要

スクリプトを書き換えることで，U-Boot から Linux を起動する際に Linux に渡すメモリサイズの制限を行った．OS 側から FPGA が使用する DDR3 SDRAM のエリアにアクセスする場合，mmap を用いることでアクセス可能である．mmap とは，ファイルをメモリにマッピングする関数で，物理アドレスをマッピングして用いた．これにより，OpenCV を用いて，ブランク画像の画素値を転送されてくるメモリエリアのデータに差し替えることで FPGA の出力結果を確認することができる．

2.4.5. 画像処理カメラシステムの動作検証

第 2.4.4 章で設計した画像処理カメラシステムの動作検証について述べる．動作検証では，画素生成回路として，パターン生成回路を設計し，カメラ信号を用いて，カメラデータをパターン生成回路に差し替え，本カメラシステムの動作確認を行った．また，本動作確認の検証では，DMA 転送された画素データを元に生成された画像は Ethernet を経由して UDP 通信で手元の別マシン（クライアント PC）へ転送を行い，クライアント側で受信し，表示させることで確認を行った．画素生成回路とはカメラから転送されるカメラデータを擬似的に再現する回路のことである．これは，実際のカメラを用いてレイテンシー時間の計測を行った場合，カメラのブランクラインや画素のデータフォーマットなどカメラ依存のパラメータによりソフトウェア処理と同じ画素数で比較を行うことが難しいことが考えられる．そこで，任意の画素数を生成できる回路を設計することで実験を行いやすくした．本実験では 1 加算パターン画像を生成させることで，データ転送システム部のデータの順序が正しいかを確認することができる．また，本研究で使用した DE1-SoC の VGA 端子は FPGA 側にあるため，HPS 側の Linux の GUI・デスクトップ環境を表示させるには，FPGA 側で画面表示用のフレームバッファ回路を別途，設計する必要がある．実験回路以外で FPGA のリソースを消費しないように，本研究では，Linux 側で UDP 通信を用いて，別のマシンへ画像を送信する方式を採用した．TCP 通信ではなく，UDP 通信を用いることで，送信側・受信側がそれぞれ独立して動作させることができる．そのため，受信側の任意のタイミングでデバッグを行うことが可能になり，デバッグ環境がシンプルになる．

動作確認を行った結果について述べる．図 2.17 に，設計したカメラシステムにおけるデータ転送システム部の動作確認を行った結果を示す．動作の時系列順に図 2.17(a) から図 2.17(c) に示す．はじめに，図 2.17(a) が OpenCV で生成させたブランク画像である．

この時点では，まだ DDR3 SDRAM の FPGA が使用するメモリエリアのデータが適用されていない状態であるため，ブランク画像が表示されている．次に，mmap 関数を用いて，FPGA が使用するメモリエリアをマッピングした結果が図 2.17(b) で，DMA 転送が開始される前のため，電源投入時の不定の初期値が格納されている．その後，図 2.17(c) が回路コンフィグレーション後，DDR3 SDRAM のメモリエリアへ FPGA 側から DMA 転送開始後の結果で，図 2.17(d) が C 言語による 1 加算パターン生成回路のシミュレーション結果である．画像サイズは一般的な解像度規格 VGA（640×480）である．1 加算パターン生成回路の構造としては，カメラ信号において有効画素エリアでピクセルクロックに同期して 8bit のレジスタに 1 加算を行っている．画素値のとりうる範囲は 0 から 255 であるため，画像の途中で 8bit レジスタがオーバーフローを起こし，0 に戻るのを繰り返している．このレジスタの値をカメラの RGB それぞれ 8bit のデータ信号部分と切り替えることにより，グレースケールの 1 加算画像が生成される．両者を比較すると，FPGA 側でパターン生成回路とデータ転送システムが正しく動作していることが目視で確認することができる．本研究ではこのシステムをベースに各システムの設計を行った．

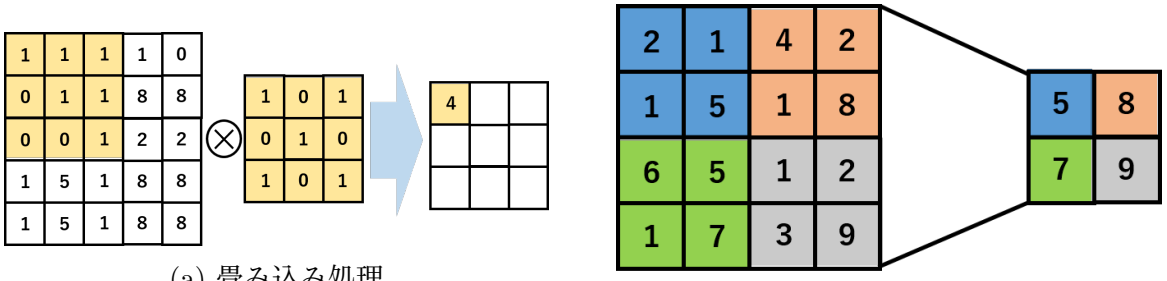
(a) ブランク画像　(b) 初期値　(c) FPGA　(d) C 言語

図 2.17: 転送システムの動作確認

第3章　畳み込みカメラシステムの設計とリアルタイム性の評価実験

3.1.　畳み込みニューラルネットワークと畳み込み層

ここでは，畳み込みニューラルネットワークと畳み込みニューラルネットワーク内における畳み込み処理とプーリング処理について述べる．畳み込みニューラルネットワークは，入力層，畳み込み層，プーリング層，全結合層層，出力層で構成されている．畳み込み層は入力画像の特徴抽出を行う層で，図 3.1(a) のように入力画像に対して重みフィルターの畳み込み演算を行う．プーリング層は図 3.1(b) のように画像の解像度を下げることで画像を荒くし，畳み込み層で抽出した特徴の縮小を行う．これにより，画像内での位置の違いや細かいノイズ等による影響を抑えることができる．プーリング処理には最大値を抽出する Max プーリングや，平均値を抽出する平均プーリングがある．今回の実験では，MAX プーリングを使用している．畳み込みニューラルネットワークにおける学習とは，この畳み込み層で適切な特徴量を抽出できるように適切なフィルターの値を調整し，求めることである．



(a) 畳み込み処理　　(b) MAX プーリング処理

図 3.1: 畳み込み層とプーリング層 [30]

本研究では，畳み込みニューラルネットワークの学習を FPGA 上で行うことは想定していない．学習自体は，オフライン環境で GPU を用いて学習を行い，学習済みの畳み込みフィルタ係数を FPGA 上に実装することを想定した．通常，畳み込みニューラルネットワークの畳み込みフィルタ係数や全結合層の重みニューロンは浮動小数点が用いられる．しかし，近年では，量子化技術により，ほぼ精度を落とさずパラメータのデータサイズを削減できることが数多く報告されている [38, 39]．通常 32bit 精度の単精度浮動小数点で表現される重みパラメータを 16bit 精度の浮動小数点または 8bit 精度の整数に量子化されている．さらには 1bit で表現をする二値化などがある．量子化を行うことで計算の高速化や省メモリ化を実現することが期待できる．FPGA 上での浮動小数点演算は DSP ブロックを多く消費し，FPGA のリソース不足が想定される．本研究においても，量子化

されたフィルター係数・重みパラメータを実装することを想定し，畳み込み層では浮動小数点表現は用いずに，整数型の畳み込み層の設計を行った．

3.2. カメラ信号を用いたリアルタイム検証実験

3.2.1. 実験内容

実験内容としては，図 3.2 に示すように，FPGA を用いて畳み込み層を設計し，実際のカメラ信号を用いて，畳み込み処理を行った．その後，レイテンシーの計測を行い，第 2.2 章で定義したリアルタイム性の検証を行った．また，CPU で同様の処理を行った場合の処理時間の計測も行い，処理速度の比較実験を行った．

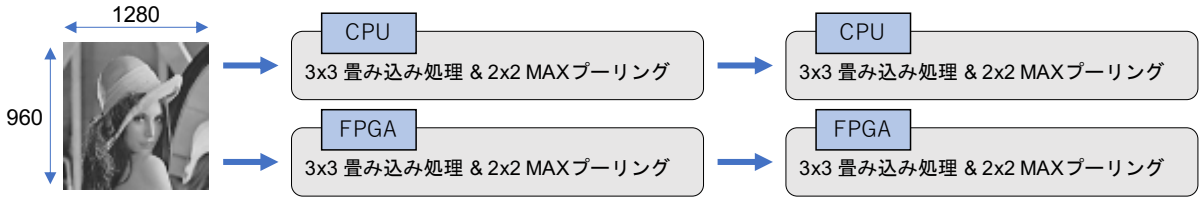


図 3.2: 実験概要

設計した畳み込み層の詳細について述べる．畳み込みフィルターサイズは 3×3 で，プーリング方式は 2×2 の MAX プーリングである．層の深さは 2 層の畳み込み層（3×3 の畳み込み + 2×2 の MAX プーリング + 3×3 の畳み込み + 2×2MAX プーリング）を設計した．本研究においては，以降 1 つの畳み込み層と 1 つプーリング層をまとめて 1 層の畳み込み層と呼ぶこととする．畳み込み後の活性化関数は，勾配消失問題に対応するため ReLU 関数（ランプ関数）が用いられることが多いため，本研究においても活性化関数には ReLU 関数を想定し，FPGA への実装を行った [40]．活性化関数とは，入力の総和に対して，どのような出力を行うかを決める関数である．ReLU 関数とは関数への入力が 0 以上の時はそのまま入力値を出力し，0 未満の場合は 0 を出力する関数である．活性化関数の他の例としては，ステップ関数，シグモイド関数等がある．

3.2.2. 実験方法

次に，FPGA，CPU のそれぞれの実験条件について述べる．本実験における FPGA のレイテンシーの定義を図 3.3 に示す．本実験では，カメラから畳み込み層に転送されてきた画像データに対して 1 フレーム分の画像データ取り込み終了時から畳み込み処理・MAX プーリング処理を行い，1 フレーム分の画像が HPS 側の DDR3 SDRAM へ DMA 転送されるまでの時間計測を行った．実験風景を図 3.4 に示す．カメラから取り込まれた画像が FPGA で処理され，HPS 側の Linux 側へ転送され，その後，別ノートパソコンをクライアントマシンとして UDP で DE1-SoC から Ethernet 経由で転送され確認している．本実験において，HPS 側は FPGA の処理結果の確認・デバッグ環境としてのみ用いている，そのため HPS 側の DDR3 SDRAM への DMA 転送終了後の処理はレイテンシーとして含めず，考慮していない．

FPGA の処理時間の計測方法として，GPIO へ信号をアサインし，直接ロジックアナライザ（ロジアナ）を用いて計測を行う方法と SignalTap を用いて計測を行う方法の大きく 2 つが挙げられる．SignalTap とは本システムの開発に使用している QuartusII の機能の一つで，SignalTap の IP を FPGA 上に一緒に実装することで，FPGA 上の信号を観測することができ，QuartusII の画面で観測データを表示・確認することができる機能である．しかし，この機能は FPGA のリソースを必要とするため，観測する信号の数や，観測時間が長くなると，膨大な FPGA リソースが必要となる．そのため，デバッグ回路が FPGA のリソースのほとんどを使用してしまうことが考えられる．以上の理由から，本研究において，FPGA の処理時間の計測には前者の GPIO へ信号をアサインし，直接ロジックアナライザを用いて計測を行う方法を採用した．

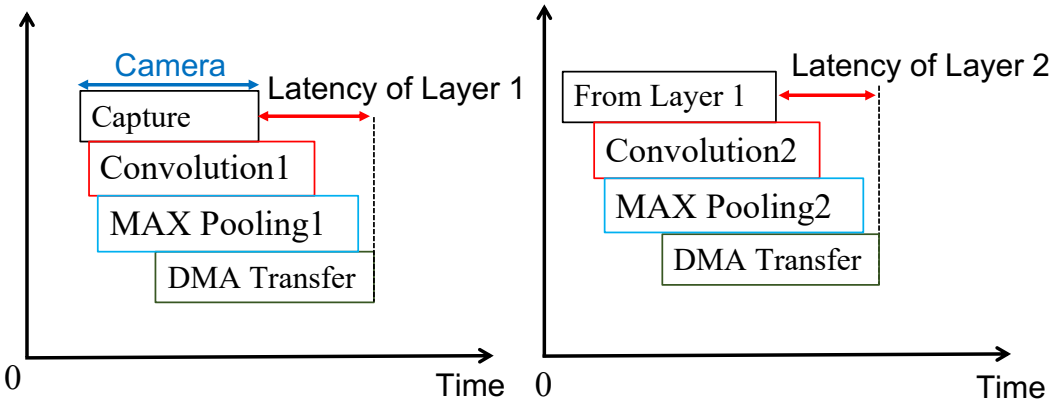
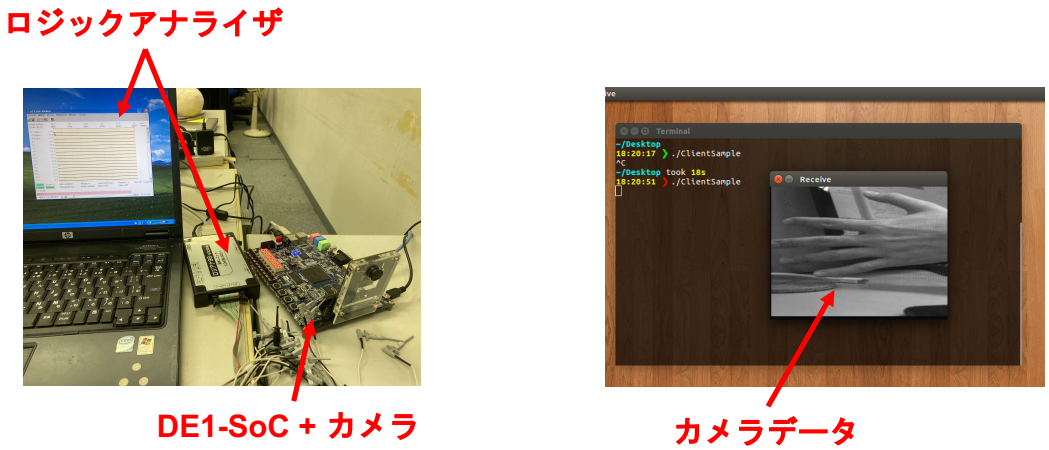


図 3.3: FPGA のレイテンシーの定義



(a) 計測風景　　(b) クライアントマシンでの確認画面

図 3.4: 実験風景

FPGA の実験環境を表 3.1 に示す．カメラ画像は一般的な解像度規格 SXVGA（1,280 × 960）を用いて実験を行った．また，実験に使用した FPGA ボード DE1-SoC のシステムクロックは 200MHz で，使用した CMOS カメラ（TRDB-D5M）のピクセルクロックは

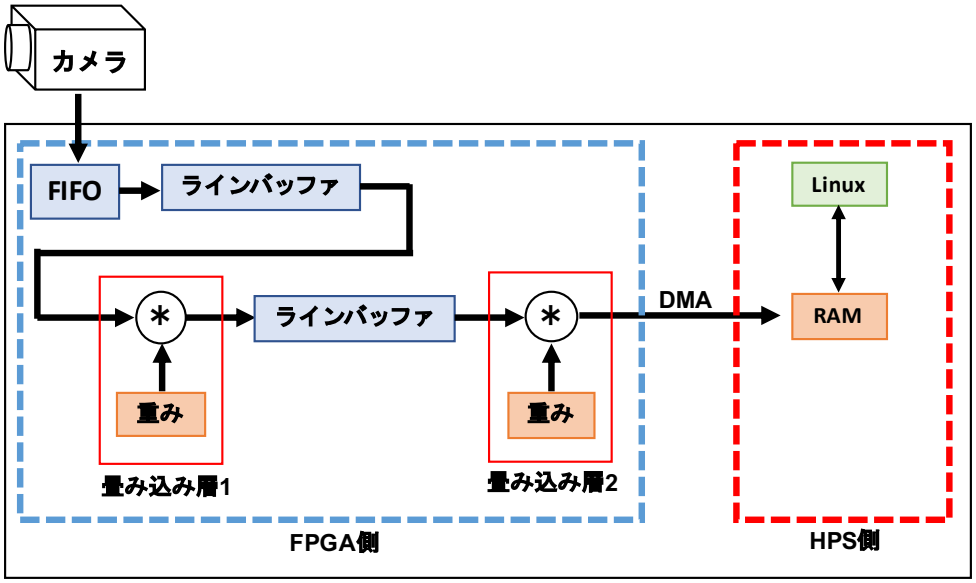
92MHz に設定をした．VerilogHDL 及び SystemVerilog を用いて設計を行い，QuartusII を用いて FPGA へ実装を行った．一般的にコンピュータ内部では 2 進数を扱う際，2 の補数表現が用いられる．しかし，本実験では，デバック作業・アルゴリズムの実装を容易にするため，絶対値表現を用いて設計を行った．特に，除算は計算時間を他の演算に比べ必要とするが，HDL はビット単位でデータアクセスできるため，上位数ビットを取り出す処理をすることで，ビットシフト演算や除算を表現可能になる．例えば，N ビットのデータの上位 N-1 ビットを取り出して，取り出したデータの最上位ビットにゼロを連接演算子で連接し，別モジュールにアサインすることで，2 で割った結果を渡しているのと同じになる．通常，1 つの畳み込み層では複数の特徴マップが生成される．SystemVerilog は多次元配列にサポートしているため，これを用いることで，for 文や多次元配列を用いて記述することが可能になり，VerilogHDL と同じ情報量で RTL 記述量を削減することができる．

表 3.1: FPGA の実験条件

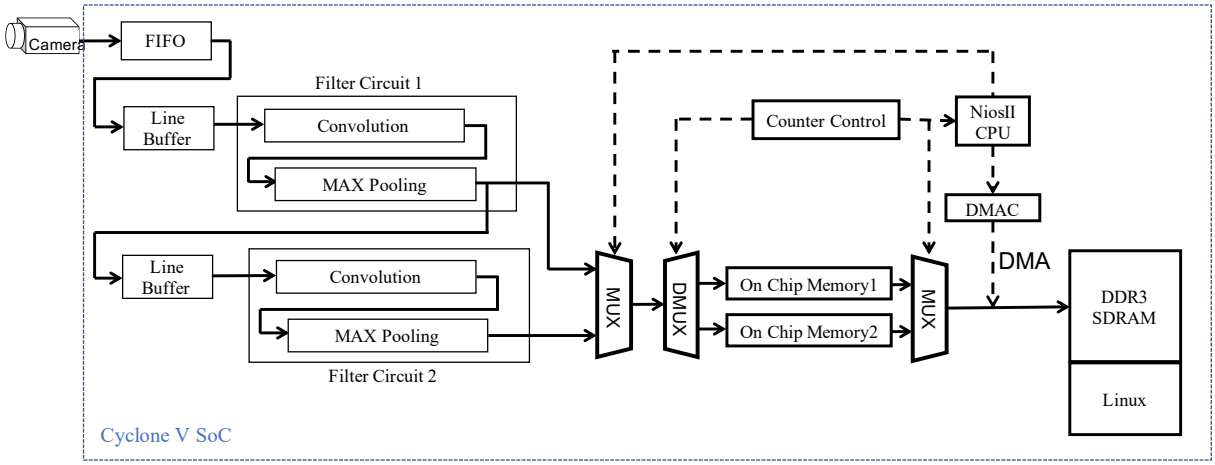
FPGA	開発言語	開発環境
Cyclone V	VerilogHDL SystemVerilog	Quartus ii 13.1 Web Edition

図 3.5 に本実験用に設計した回路図を示す．カメラと FPGA の動作クロックが異なるため，DCFIFO（Dual Clock FIFO）を用いる事で速度の違いを吸収している．カメラから取り込まれたグレースケール画像に対して，3×3 畳み込み処理と 2×2 の MAX プーリング処理が行われ，2 層目の回路へ転送される．2 層目転送後は 1 層目同様に 3×3 畳み込み処理と 2×2 の MAX プーリング処理が行われる．畳み込み層から出力された画像 1 ライン分のデータが On Chip Memory に格納されると，割り込み信号を発生させ，Nios CPU から DMA コントローラーに転送命令を出し，On Chip Memory から HPS 経由で DDR3 SDRAM に 1 ライン分の画像処理後のデータが DMA 転送される．図 3.5 中の MUX（マルチプレクサ）と DMUX（デマルチプレクサ）は信号切り替機である．これにより，1 ライン分をバッファとしておくダブルバッファ方式の On Chip Memory の制御を行っている．奇数ラインのデータを格納中は偶数ラインのデータが DMA 転送され，また，偶数ラインのデータを格納中は奇数ラインのデータが DMA 転送される．

次に，CPU の実験条件について述べる．表 3.2 に CPU の実験条件を示す．CPU は Intel(R) Core(TM) i5-2400S CPU(2.5GHz) を用いた．CPU 側のソフトウェア画像処理プログラムは C/C++言語を使用し，OpenCV 画像処理ライブラリを用いて作成した．コンパイルには，gcc を使用した．gcc でのコンパイル条件として，コンパイル時の最適化オプション等の指定はせず，並列化及び，マルチスレッド化などは一切行っていない．本実験で使用したカメラを CPU 環境で FPGA と同じ実験条件で検証比較する事は難しい．そのため，本実験での CPU 環境はあらかじめ用意しておいたグレースケール画像をメインメモリに読み込ませ，その画像に対してフィルター処理を行うこととした．図 3.6 に本実験における CPU のレイテンシーを示す．本実験では，CPU の処理時間には画像読み込み終了時から，フィルター処理が終了するまでの処理時間の計測を 10 回行い，計測された時間の平均時間を CPU のレイテンシー時間と設定した．FPGA・CPU それぞれ，画像取り込み終了時から畳み込み処理完了までの時間を計測し比較を行った．



(a) アーキテクチャの概要



(b) アーキテクチャの詳細

図 3.5: 畳み込みカメラシステムアーキテクチャ

表 3.2: CPU の実験条件

CPU	開発言語	開発環境
Intel(R)Core(TM) i5-2400S(2.4GHz)	C/C++	gcc

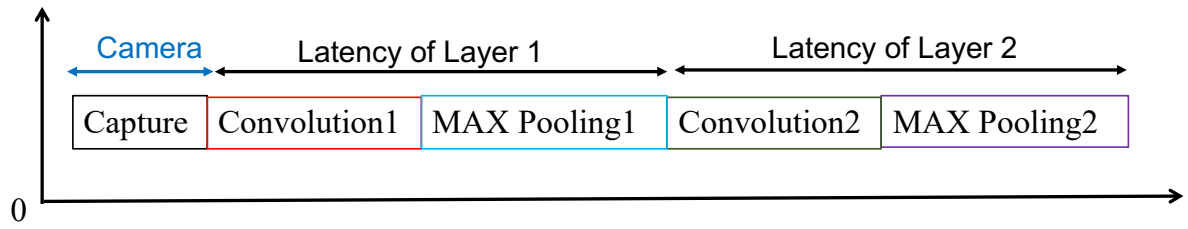


図 3.6: CPU のレイテンシーの定義

3.2.3. レイテンシーの計測結果

図 3.7 に 1 層目における FPGA と CPU によって処理された出力結果を示す．図 3.7(a) にグレースケール化前のカメラ画像を示す．図 3.7(b)，図 3.7(c) を比較すると，FPGA において正しく畳み込み処理が行われている事が目視で確認できる．各層における 1 フレーム分の処理結果を表 3.3 に示す．1 層目の 1,280×960 の画像に対する畳み込み処理・MAX プーリング処理の CPU のレイテンシーが 78,693.0us であったのに対して，FPGA のレイテンシーは 44.7us であった．2 層目の 640×480 においては，CPU のレイテンシーが 18,516.0us に対して，22.5us であった．CPU のレイテンシーが ms オーダーであるのに対して，FPGA は us オーダーのレイテンシーであり，FPGA が CPU に比べ優れた処理性能を示している．

FPGA のリソースの消費量としては，1 層・2 層合わせて 21%であった．畳み込み層の多層化を想定した場合，プーリング処理等で画像サイズが小さくなるため，リソースの消費量の増加は小さくなっていくと考えられる．

(a) カメラ画像　(b) FPGA　(c) CPU

図 3.7: 1 層目における出力結果

表 3.3: レイテンシーの比較

Device	Layer 1 [us]	Layer 2 [us]
FPGA	44.7	25.5
CPU	78,693.0	18,516.0

3.2.4. リアルタイム性の検証

ここでは，第 3.2.3 章で行った実験結果に対するリアルタイム性の検証を行った結果について述べる．FPGA において，カメラから取り込まれたデータは DCFIFO に入れることで，動作クロックの違いを吸収している．そのため，取り込み後のデータは FPGA の動作クロック 200MHz で動作している．つまり，1 クロックあたり，5ns で動作している．実験結果より，1 層目のレイテンシーが約 45us となり，約 9,000 クロック遅れとなっている．すなわち，次フレームの約 7 ライン遅れとなっている．1 層目の畳み込みフィルターが 3 × 3 で，MAX プーリングが 2 × 2 であるため，ラインバッファが合計 5 本分必要となり，少なくとも 5 ライン分の遅れが生じることになるため，遅れ時間が設計通りになっていることを確認した．2 層目のレイテンシーが 25us であり，約 5,000 クロック遅れとなっ

ており，1 層・2 層合わせて約 14,000 クロックの遅れとなる．1 層目と 2 層目合わせて合計 10 ラインの遅れとなり，次フレームの約 10 ラインが取り込み終了時に 2 層の畳み込み層での処理が行われていることになる．本実験では，フレームのサイズは $1,280 \times 960$ であり，960 ラインの内の 10 ラインは，全体 960 ラインの 1%程度の遅れとなっている．つまり，次フレームの取り込みが完了するまでに処理が完了していることになる．本研究では，リアルタイム処理の定義を次フレームの取り込み終了までに前の処理が完了していることと定義した．そのため，レジスタ転送レベルでのリアルタイム画像処理が実行できていることを確認できた．また，CPU に比べ，FPGA のレイテンシーのオーダーが小さいため，画像サイズを大きくした場合を想定しても，フィルター処理がボトルネックにならないと考えられる．2 層の畳み込み層におけるレイテンシーが次フレームの 1%程度であり，FPGA のリソース消費量の観点からも，さらなる多層化にも対応可能である．

第4章　ハードウェアベース多層ニューラルネットワークの設計と評価

4.1.　深層学習とニューラルネットワーク

深層学習とニューラルネットワークについての概要を述べる．深層学習とは，ニューラルネットワーク（Neural Network: NN）を多層にした機械学習のことである．ニューラルネットワークとは人間の脳の神経回路網を模倣した数学モデルのことで，図 4.1 に示すように，入力 x に対してそれぞれの重み w を掛け合わせ，総和を活性化関数 f で処理をし，y を出力する．本実験では活性化関数は ReLU 関数を用いた．ニューラルネットワークにおける学習とは，出力層が目標出力となるように重み w を調整することである．この学習により，学習済みモデルが生成され，これを用いて，入力画像に対して推論を行う．畳み込みニューラルネットワークにおける全結合層は多層ニューラルネットワークと構造は同じである．本章では，畳み込みニューラルネットワークの全結合層に向けて多層ニューラルネットワークを FPGA 上に実装し，学習済みモデルを用いて，推論時間の検証を行った実験について述べる．

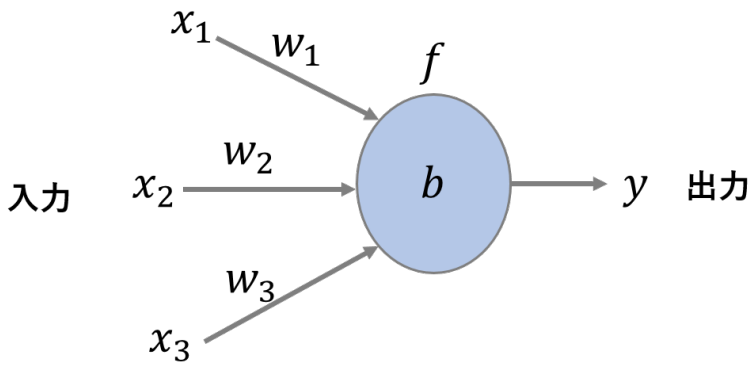


図 4.1: ニューラルネットワークの概要図 [30]

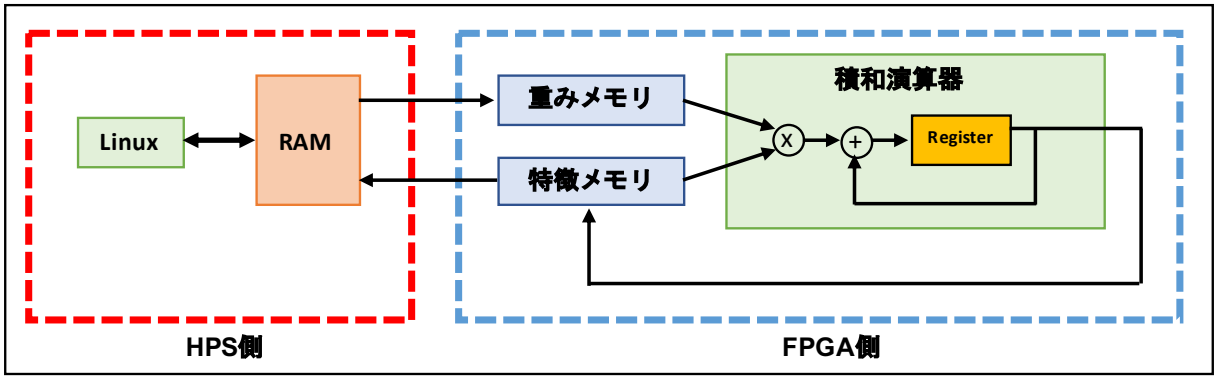
4.2. 畳み込みニューラルネットワークの全結合層に向けた多層ニューラルネットワークの実装実験

4.2.1. 実験内容

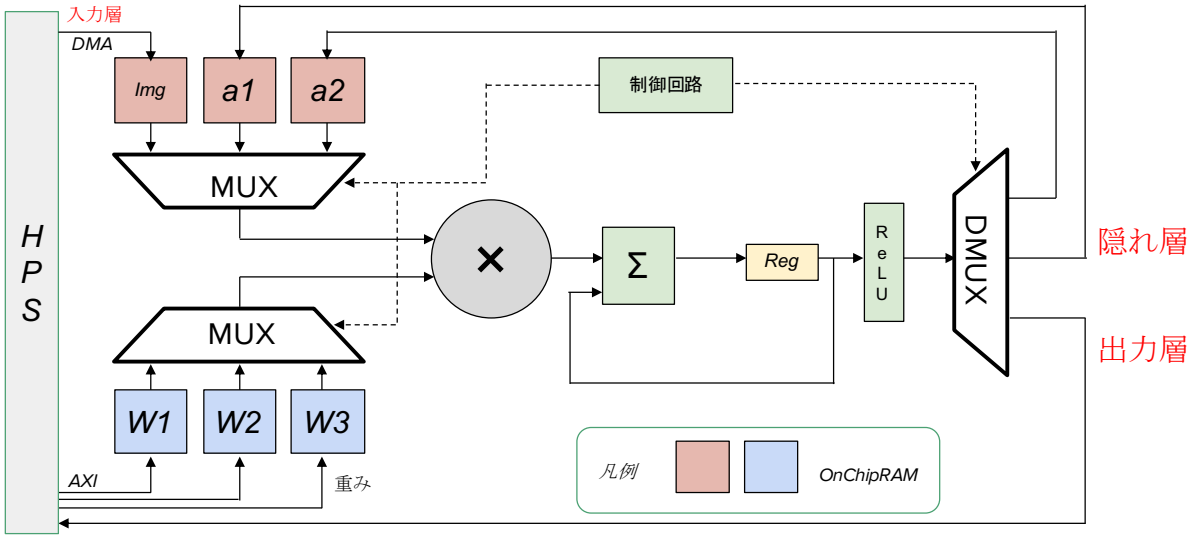
本研究の目的は，畳み込みニューラルネットワークを FPGA に実装することである．本章では，畳み込みニューラルネットワークの全結合層を想定した，多層ニューラルネットワークを FPGA 上及び CPU 上に実装し，予測時間の検証を行った．実験概要としては，まず，4 層の多層ニューラルネットワークモデルを既存のデータセットを用いてオフラインで学習を行い，FPGA・CPU 向けの学習済みモデルを作成した．その後，学習済みモデルを FPGA と CPU 上に実装し，1 フレームの画像の予測を行い，予測時間に対する比較・速度検証を行った．図 4.2 に FPGA 上に実装したニューラルネットワークのアーキテクチャを示す [41, 42]．構成としては，第 3 章の実験と同様の SoC FPGA を用いて，HPS と FPGA のメモリ共有エリアに重み情報と入力画像を用意し，FPGA 側の積和演算器でニューロン演算を行った．ニューロンの重み情報などは HPS 側で制御を行い，FPGA 側の On Chip RAM へ DMA 転送を行っている．演算結果をメモリ共有エリアに格納し，ループ型ステートマシンにより逐次処理でニューロン演算を行う．逐次処理でニューロン演算を行う場合，1 層目のループ回数が他の層と比較すると，極端多くなるため，1 層目のニューロン演算がボトルネックとなることが考えられる．そこで，1 層目のみ，2 並列でのニューロン演算でも実験を行い，推論時間の比較を行った．さらには，隠れ層 2 と出力層間を完全に並列化を行い，推論時間の検証を行った．

4.2.2. 実装方法

多層ニューラルネットワークの FPGA への実装フローを図 4.3 に示す．実装方法としては，まず事前に，Python で TensorFlow を用いて GPU 上で学習を行い，最終的に，整数型ベースの学習済みモデルを作成した．本研究では FPGA 上での学習は想定しておらず，学習済みモデルを FPGA 上に実装することを想定している．一般的に，機械学習で生成される学習済みモデルの重みは浮動小数点ベースである．TensorFlow を用いて作成した学習済みモデルの重みも 32bit の浮動小数点フォーマットである．第 3 章でも述べたが，FPGA 上での実装を考えたとき，浮動小数点演算は DSP を多く消費し，FPGA のリソース不足が考えられる [43, 44, 45]．そこで，本実験では TensorFlow Lite を使用して 32bit 浮動小数点から符号付き 8bit 整数への重みの量子化を行った．この量子化された重みパラメータを基に HDL を用いて FPGA 向けのモデルと C++を用いて CPU 向けのモデルの作成を行った．作成したこれら二つを用いて，CPU と FPGA での画像 1 枚あたりの推論時間の比較を行った．CPU の処理では，本実験で用いた SoC FPGA(Cyclone V SoC) の ARM プロセッサー上で行った．TensorFlow は Google が開発しオープンソースで公開している機械学習に向けのソフトウェアライブラリである．TensorFlow Lite はこの TensorFlow で作成されたモデルを組み込みデバイス・IoT デバイスで実行できるようにするためのツールでモデルサイズを抑えることが可能である．実装したニューラルネットワーク構造は，4 層のニューラルネットワークで，入力は 28×28，隠れ層 1 のニューロンの個数が 60 個，隠れ層 2 のニューロンの個数が 20 個，出力 2 の 2 クラス分類器であ



(a) FPGA ニューラルネットワークアーキテクチャの概要



(b) アーキテクチャの詳細

図 4.2: FPGA ニューラルネットワーク

る．本実験に用いたデータセットは，AI 搭載型害獣捕獲システムに用いたのと同様の猿（ラベル:0）・人（ラベル:1），2 つのラベル付けされたデータセット（以下：Monkey データセット）を用いた [30].

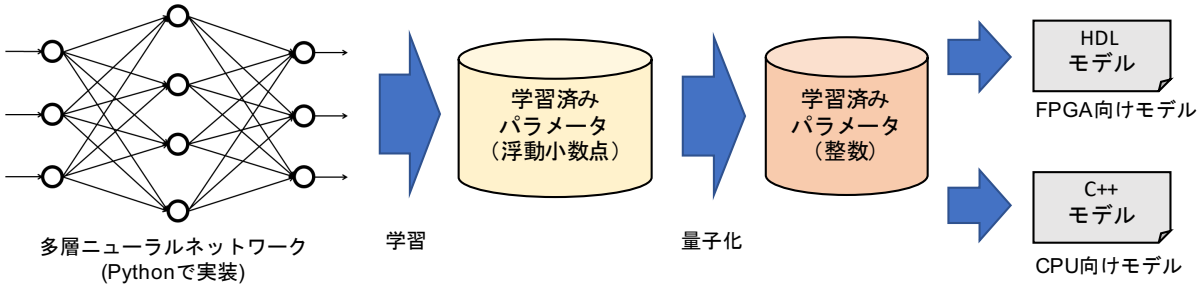


図 4.3: FPGA への実装手順

4.2.3. 実験結果と考察

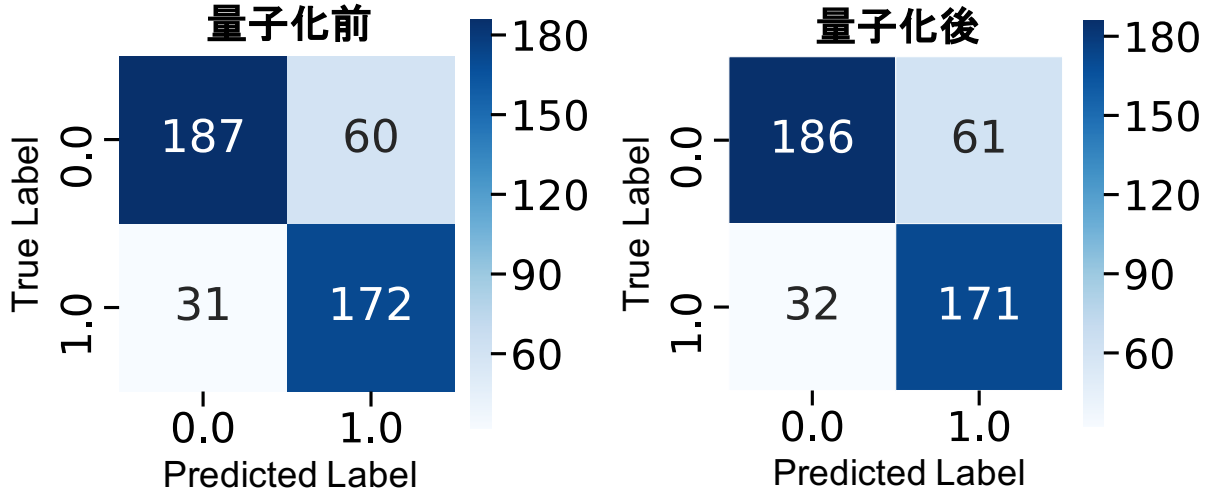
表 4.1 に TensorFlow で事前に作成した学習済みモデルの精度及び，TensorFlow Lite による量子化後の精度を示す．また，図 4.4 に量子化前後の混同行列を示す．量子化後のモデルは量子化前に比べ，猿（ラベル:0）・人（ラベル:1）それぞれ 1 枚ずつ多く誤認識している結果となり，認識精度の差は 1%未満に抑えており，量子化による精度の劣化についてはほとんど見られなかった．また，本害獣捕獲システムの重要な評価指標としている再現率 [30] においても誤差は 1%未満に抑えられている．

表 4.1: 量子化前後のモデルの精度比較

	量子化前 (float 型)	量子化後 (int 型)
精度 [%]	79.77	79.33
再現率 [%]	84.72	84.23

次に，FPGA・CPU での推論時間の結果について述べる．図 4.5(a) 及び，図 4.5(b) に計測を行った部分について示す．本稿では，入力層から隠れ層 1 までの演算時間を t_1，隠れ層 1 から隠れ層 2 までの演算時間を t_2，隠れ層 2 から出力層までの演算時間を t_3 としている．本実験では，カメラを用いずに予めメモリ上に用意したテストデータを用いた．カメラを用いていないので，メモリ展開時間等は含めずに，入力層から 2 つの隠れ層・出力層までの時間を処理時間の定義とした．表 4.2 は，FPGA における逐次処理及び並列処理での推論時間と各層における処理時間，また，設計上の理論値，ARM プロセッサー上で推論を行った推論時間の結果を示している．ニューロン演算を並列処理なしで行った場合，241.73us で，1 層目のニューロン演算のみを 2 並列で処理を行った場合，123.8us という結果が得られた．

逐次処理のみでの画像 1 枚当たりの推論時間が 241.73us であった点について，考察する．本実験での並列無しの逐次処理アーキテクチャでは全結合層の各層・各ニューロンが逐次処理のみで行われている．つまり，重みの個数分のクロックが計算に必要となる．t_1 の場合，入力が 28×28 の合計 784 個で隠れ層 1 のニューロンの個数が 60 個であるため，



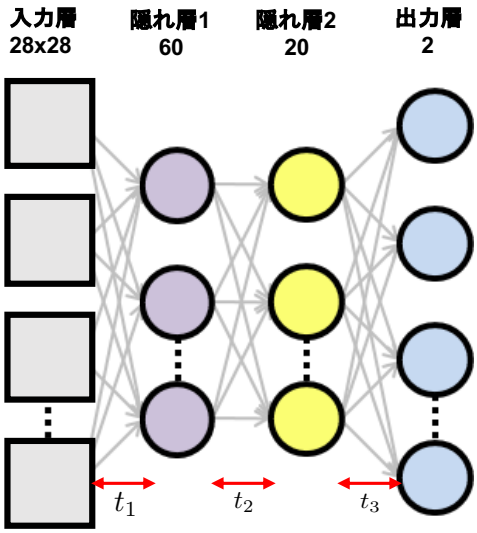
(a) 浮動小数点　　(b) 整数型

図 4.4: 量子化前後の混同行列

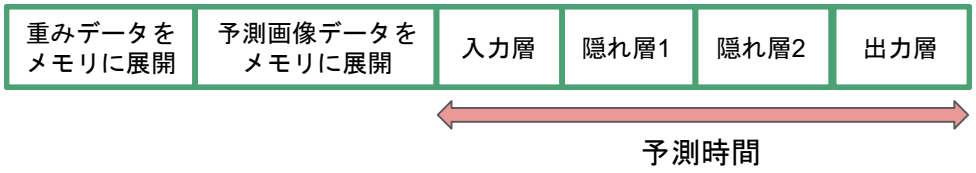
On Chip RAM のレイテンシーを考慮しなかった場合，$28 \times 28 \times 60$ クロック必要となり，合計約 47,040 クロック必要になる．FPGA 側の内部クロック数は 200MHz で，1 クロック $5ns$ となる．そのため，理論値は $47,040 \times 5 = 235,500ns$ となる．t_2，t_3 も t_1 と同様に理論値を算出することができる．よって，FPGA による画像 1 枚当たりの推論時間は理論値 $241.40us$ となり，本実験での実測値 $241.73us$ は同等の結果となっている．また，各層においてもそれぞれ実測値は On Chip RAM のレイテンシーを考慮しなかった場合の理論値とほぼ同じになっている．図 4.6(a) に逐次処理でのニューロン演算を行った際における各層の処理時間の比率をロジックアナライザで計測した結果を示す．それぞれの信号は全て同じサンプリング周波数で計測を行っている．各層の処理時間の比率を比較すると，1 層目の処理時間は，入力と隠れ層 1 のニューロン数が他の層に比べ多いため，極端に多くなっており，推論時間のほとんどを占めている．そのため，1 層目を 2 並列化することで，表 4.2 に示すように，推論時間が約半分になっている．したがって，FPGA の処理時間が設計通りになっていることが確認できた．

表 4.2: FPGA・CPU での予測時間と FPGA の理論値

			t_1 $[us]$	t_2 $[us]$	t_3 $[us]$	Total $[us]$
CPU	実測値	ARM での処理時間	-	-	-	2,560.00
FPGA	実測値	逐次処理のみ（並列無し）	235.50	6.00	0.23	241.73
		1 層目 2 並列化（並列あり）	117.75	6.00	0.23	123.98
	理論値	逐次処理のみ（並列無し）	235.20	6.00	0.20	241.40
		1 層目 2 並列化（並列あり）	117.60	6.00	0.20	123.80

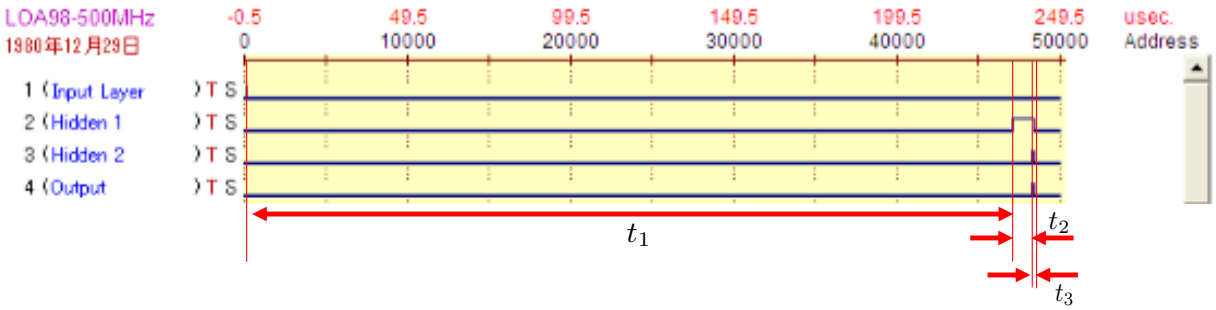


(a) ニューラルネットワークの構造

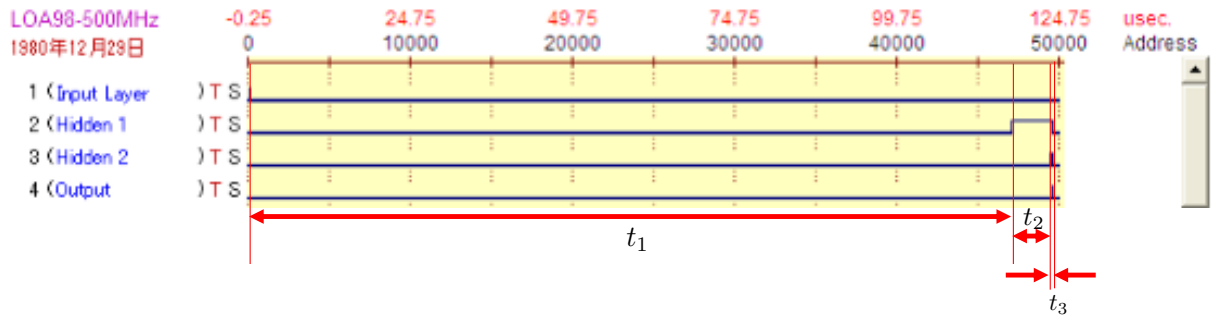


(b) レイテンシーの定義

図 4.5: ニューラルネットワークの構造とレイテンシーの定義



(a) 逐次処理における各層の処理時間の比率



(b) 1 層目を並列化した場合における各層の処理時間の比率

図 4.6: 各層の処理時間比率

本モデルでの FPGA でのメモリリソースの使用率は 45%程度である．CPU での予測時間の結果は 2.560ms となった．ARM プロセッサー上で行った結果と比較すると，大幅なレイテンシーの削減が行われている．本実験のニューロン演算は逐次処理では積和演算器 1 つ，並列処理では，2 つの積和演算器で繰り返し逐次処理を行っているため，図 4.7 に示すように，積和演算器を複数実装し，並列化をすることで，さらなる高速化を見込むことができる．理論上，リソースの問題を除けば，各層毎を並列に処理することができるため，層の数分のクロック時間で処理できることになる．したがって全結合層が 3 層である場合，理論的には 3 クロックで計算が終了することになるため，レイテンシー時間は 15ns 程度になると予想できる．

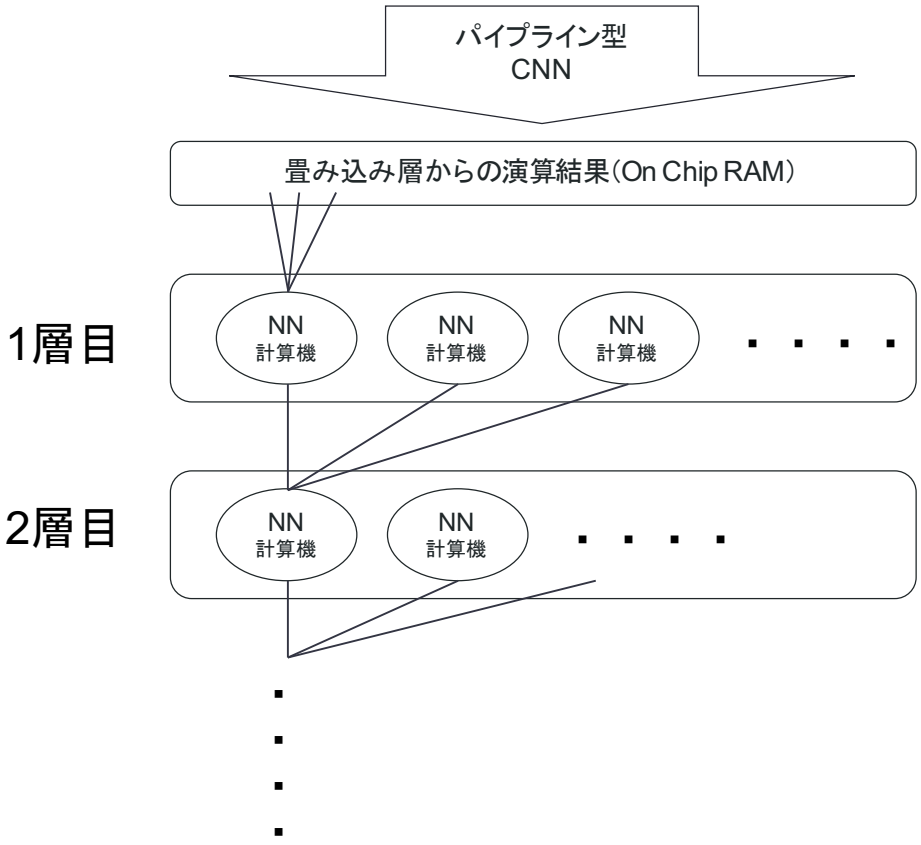


図 4.7: 並列型ニューラルネットワークアーキテクチャ

完全並列化した場合の検証を行った．ここでは，検証を簡易にするために，隠れ層 2 と出力層間の完全並列化を行った．図 4.5(a) に示す t_3 部分において，積和演算器を隠れ層 2 からの 20 個と出力層 2 の合計 40 個用意し，完全並列化を行った．動作周波数は 50MHz で 200MHz のサンプリングクロックで計測を行った．50MHz で従来通り，逐次処理で行った場合の処理時間を t'_3，完全並列化した場合の処理時間を t''_3 とした．

結果を図 4.8 に示す．t'_3 は 200MHz のサンプリングで 161 クロック遅れであるため，50MHz で約 40 クロック分となり，処理時間が理論値通りとなっている．また，t''_3 は 200MHz のサンプリングで 4 クロック遅れであるため，50MHz で 1 クロック分となり，1 層の処理時間が 1 クロックで行えていることを確認した．この結果より，完全並列化を行うことは，層の数分のクロック時間で処理できることが確認できたため，完全並列化は有効な手

段である．また，その際は第 3 章でも述べた System Verilog による設計が RTL 記述量を削減する点で有効であると考えられる．

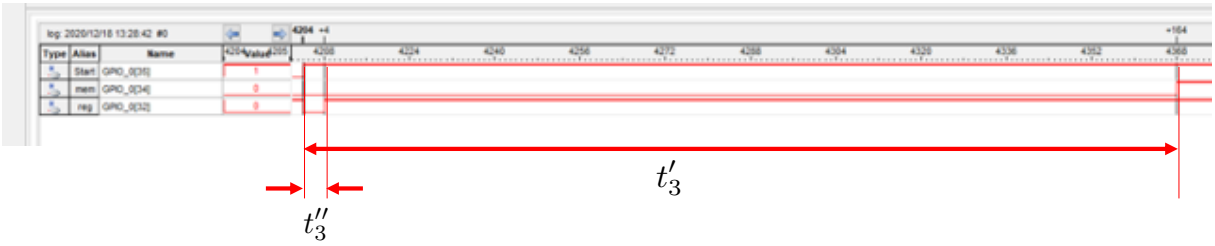


図 4.8: 完全並列化ニューラルネットワークとの比較

第5章　背景差分法による物体検出の実装と評価

5.1.　背景差分法

5.1.1.　背景差分法の概要

第3章，第4章で畳み込みニューラルネットワークのハードウェア化の有効性を示すことができた．本章では，背景差分法を用いて対象物体の場所を特定し，対象物体周辺を切り出して畳み込みニューラルネットワークの入力画像とする方法について述べる．

本研究でターゲットアプリケーションとしているAI搭載型害獣捕獲システムにおいて，背景差分法による物体検出[30]を前処理として行って，畳み込みニューラルネットワーク部に入力している．これは，カメラ画像中における分類したい対象物体の場所抽出することを目的としている．対象物体周辺を切り出すためには，対象部分と背景を二値値画像にすることで切り出し処理しやすくなる．カメラで撮影を行った場合，対象となる物体は背景画像上に写っているため，背景との差分が有効的である（図5.1(a)）．同様な処理に図5.1(b)に示すフレーム間差分がある．この手法は動画像に多く用いられ，特に物体の動きや速度を割り出すことに役立つが，本研究で扱う害獣捕獲では動画像解析の必要はないため，背景差分法を採用した．背景差分法とは，図5.1(a)に示すように，ターゲット画像と事前に取得しておいた背景画像の差をとることで，背景画像に存在しない対象物体を抽出する手法である．背景差分法では背景部分での画像変化が観測されないことが大前提である．提案されている背景差分法では，事前に照明変化にも対応可能な背景画像を生成する手法が示されている．本章では背景画像は事前に生成されていると仮定し，背景差分法による物体検出処理をFPGA上に実装を行った．以下に背景差分法を使った対象物体の切り出し手法とFPGAでの実装について述べる．

(a) 背景差分法　　(b) フレーム間差分法

図 5.1: 差分処理

図5.2にFPGA上で行う前処理の手順を示す．まず，背景差分法による二値化画像を

生成し，その二値化画像に対して，ランレングス圧縮を行う．その後，結合比較処理を行い，それぞれのラベルで同時に最大・最小の X 座標・Y 座標を取得することで，物体周辺のみの画像を取得できる．

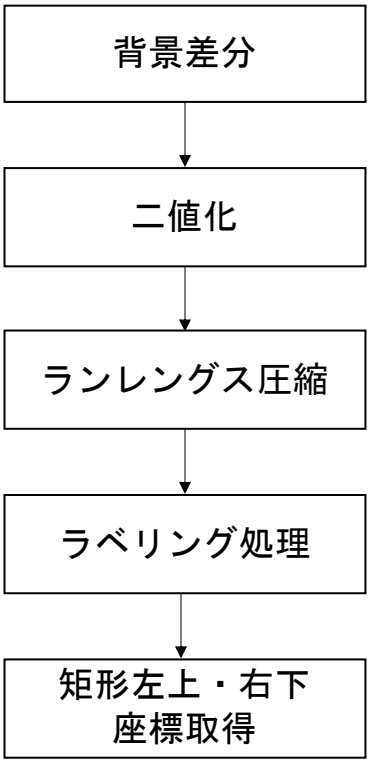


図 5.2: 前処理の概要

5.1.2. ハードウェアベース背景差分法の実装

背景差分法による二値化を行った実験について述べる．本実験でも，第 2 章で述べた画像処理システムをベースにし，差分画像を FPGA で生成し，HPS 側へ転送を行い，確認を行った．図 5.3 に実験回路を示す．背景メモリと差分画像を格納する On Chip RAM を用意し，カメラから 1 ピクセルずつ GPIO を介して取り込みながら，同タイミングで背景メモリから背景データ 1 ピクセルを取り出し，差をとり，On Chip RAM へ結果を格納する．これらの On Chip RAM は 1 ライン分のデータを格納できるようになっており，1 ラインの処理が終了すると，背景メモリの更新と処理結果の転送を行う．この結果を HPS 側で OpenCV を用いて表示を行った．画像サイズは一般的な解像度規格 VGA（640×480）で行った．レイテンシーの計測区間を図 5.4 に示す．1 フレームの取り込み終了時から差分をとり，二値化するまでの時間を計測した．二値化の閾値は手動で調整を行った．ピクセルクロックは 25MHz で処理は全てピクセルクロックに同期して行われるアーキテクチャになっている．

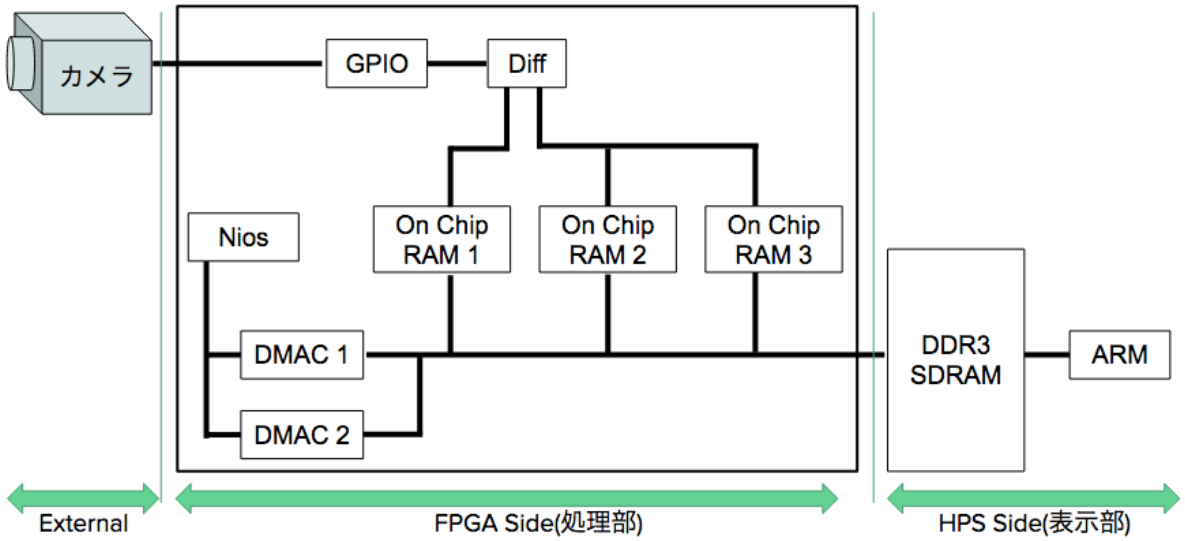


図 5.3: 背景差分回路のアーキテクチャ

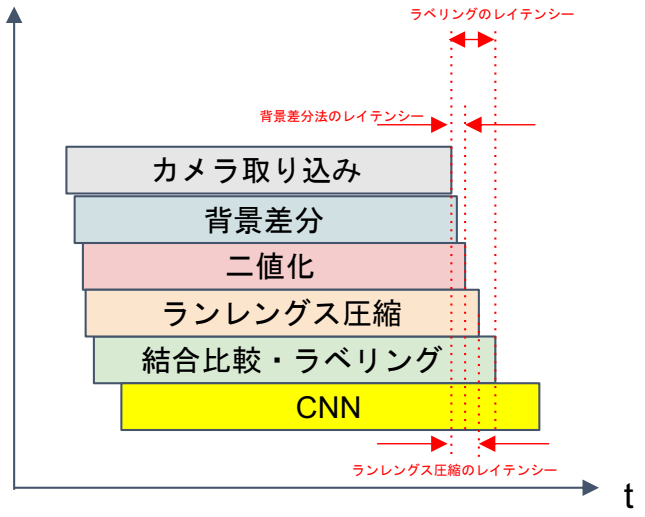
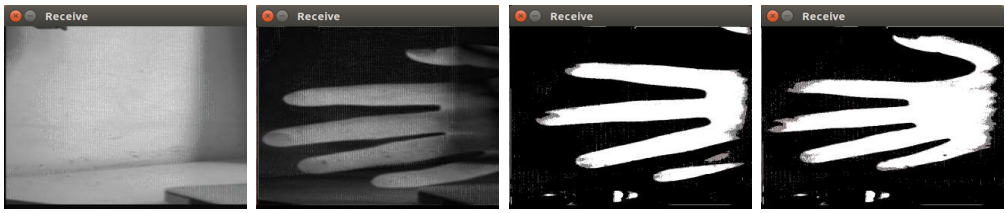


図 5.4: 背景差分法による物体検出のレイテンシーの定義

5.1.3. ハードウェアベース背景差分法のレイテンシーの計測結果

図 5.5 に処理結果を示す．図 5.5(a) が用意した背景画像である．図 5.5(b) から図 5.5(d) は 3 フレーム分の背景差分処理後の結果である．図 5.5(b) が差分画像の二値化を行っていない結果で，影の影響により，対象物体以外も現れている．図 5.5(c)，図 5.5(d) が二値化処理も行った差分画像である．ノイズ等の除去が行っていないため，ゴマ塩ノイズが見られるが，正しく差分画像が生成されいることが目視で確認することができる．計測結果を表 5.1 に示す．背景差分法はフィルター処理とは異なり，ラインバッファへの格納する必要が無く，ほぼレイテンシーをゼロで行うことができている．島崎らの報告によると，ノイズ除去の手法の 1 つであるメディアンフィルター処理（フィルターサイズ：9×9）を VGA 規格の画像に対して次フレームの 5 ライン分のレイテンシーで実現している．そのため，ノイズ除去を行った場合でも，リアルタイム性への影響はないと考えられる．



(a) 背景画像（事前用意）　(b) 差分画像 1（グレースケール）　(c) 差分画像 2（二値化）　(d) 差分画像 3（二値化）

図 5.5: 背景差分法による二値化結果

5.2. ランレングス圧縮処理とラベリング処理

5.2.1. ランレングス圧縮処理とラベリング処理の概要

ランレングス処理とラベリング処理の概要について述べる．通常，逆 U 字型や X 型などの複雑な形に対するラベリングではラベリング処理をハードウェアで行う際には複数回の走査が必要になる．島崎らの研究 [46] では，ハードウェア・ソフトウェア協調設計により 1 回の走査で正しくラベリング処理が行われている．本研究におけるターゲットアプリケーションにおいては上記の例に挙げた複雑な形は想定されないため，ソフトウェア処理は用いずにハードウェア処理のみで単純な形のラベリングを行うモジュールの設計を行った．ソフトウェアを用いないことで，ハードウェア・ソフトウェア間のデータ転送時間を必要としないため，レイテンシーを小さくすることが期待できる．

ランレングス圧縮とは図 5.6 のように連続して現れる同じ数値データを数値情報として繰り返しの回数に置換しデータ量を削減する可逆な圧縮方式である．この方式は画像内に同じ数値が連続する区間が多い場合に有効で，同じ数値の繰り返しが少ないと元のデータ量よりも圧縮後のデータ量が大きくなるという欠点がある．本研究におけるターゲットアプリケーションにおいては画像内の一部に対象物体が集中すると考えられているため，ランレングス圧縮を用いることは有効であると考えられる．ランレングス圧縮を用いる理由について述べる．ランレングス圧縮を行うことで，ランレングス圧縮の段階で画像の同じ Y 座標を持つランレングスデータはラベリングされる．そのため，ラベリング処理は

上・下の X 座標のランレングスデータと繋がっているかを確認するだけのシンプルなアルゴリズムでラベリングを行うことができる．

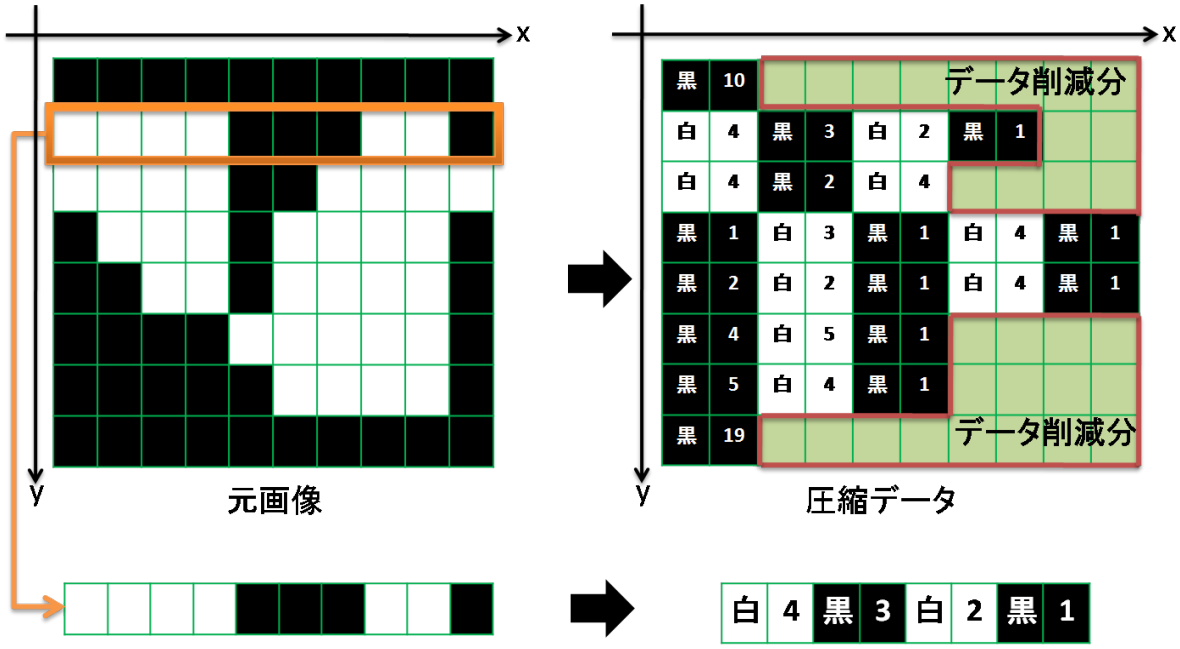


図 5.6: ランレングス圧縮の概要 [32]

図 5.7 にランレングス圧縮の処理手順を示す．入力データとしてカメラクロックに同期して二値化されたピクセルデータが転送されてくる．本研究では，入力データとして背景差分法により，二値化されたピクセルデータが転送されてくることを想定している．画素データが転送されてくるたびに，1 クロック前の入力データと画素値の比較を行い，同じかどうか比較を行う．比較を行った結果，もし値が同じであれば，ランレングスの長さのカウンタ（ランレングス長）を 1 加算する．値が異なる場合は新しい結合データとして出力する．もし，画素値が黒（0）の場合，結合データは生成されず，破棄される．これは，背景差分法により，二値化された時点で，黒の画素データは不要なデータとしているためである．白（255）の場合のみ結合データが生成される．また，Y 座標の更新があると，自動的に新しい結合データを生成する．

5.2.2. ハードウェアベースラベリングの実装

本研究で設計したラベリングモジュールのアーキテクチャを図 5.8 に示す．現在取り込み中の画素データを n 番目として，ランレングスモジュールから転送されてくるランレングスデータを On Chip RAM1 へ格納し，同時に FIFO へ格納する．この時，ランレングスのスタート位置の X 座標と Y 座標，ランレングス長，On Chip RAM1 へ格納するアドレスを On Chip RAM1，FIFO へ格納している．これにより，FIFO からデータを取り出すと，On Chip RAM1 の保存先が取得できる．このアドレスを用いて，読み出しコントローラーを制御している．FIFO から取り出されたデータを最新データとして，読み出しコントローラーは On Chip RAM1 から 1 ライン前のデータを読み出し結合モジュールで結合比較を行う．読み出しコントローラーは FIFO から取り出された On Chip RAM1

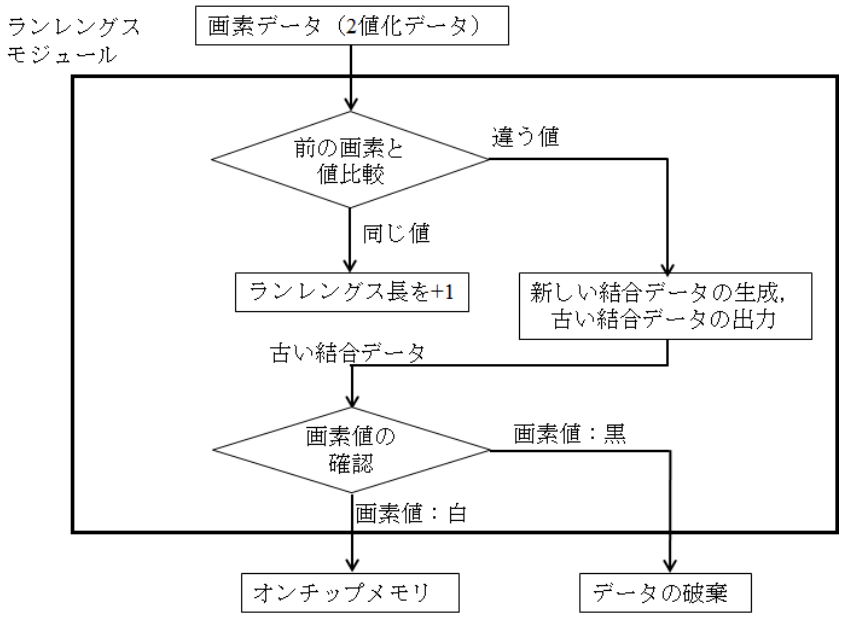


図 5.7: ランレングス圧縮のアルゴリズム [32]

の保存先のアドレスを基準として，Y 座標が 1 つ離れたデータを On Chip RAM1 から順番に読み出し，Y 座標が 2 つ以上離れるまで結合比較を繰り返す．結合比較アルゴリズムは図 5.9 に示す島崎らが用いた方法 [32] と同様の手法を用いた．二つの結合データにおけるスタート位置の X 座標を比較し，小さい方の X 座標にランレングス長を加算した結果が大きい方の X 座標より大きいかどうかを比較して結合比較を行っている．もし，小さい方の X 座標にランレングス長を加算した結果が大きい X 座標より大きい場合，これらの結合データは繋がりを持つ．小さい方の X 座標にランレングス長を加算した結果が大きい方の X 座標より小さい場合にはこれらの結合データは繋がりを持たないことになる．On Chip RAM2 はラベル格納用メモリで，初期値はアドレスの値が格納されており，On Chip RAM1 と対応している．同時に，各ラベルでの X・Y 座標の最大・最小値を比較することで，各ラベルの左上・右下の座標を取得することができる．このように，画像取り込みからラベリングまでをパイプライン処理構造で設計することで，ラベリングの切り出しサイズの影響を受けずに処理が可能になる．

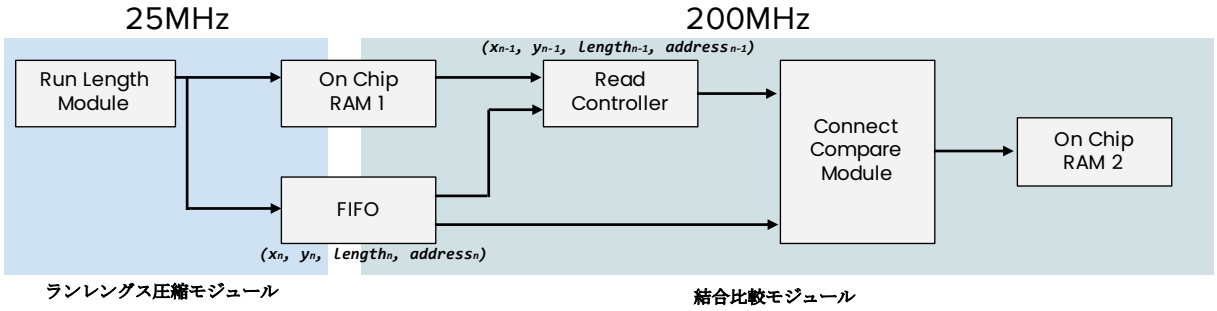


図 5.8: ラベリングモジュール

ラベリングモジュールのレイテンシー評価を行った実験について述べる．本実験では，単純な形のラベリングを行うことを想定しており，動作検証を容易にするため，カメラ

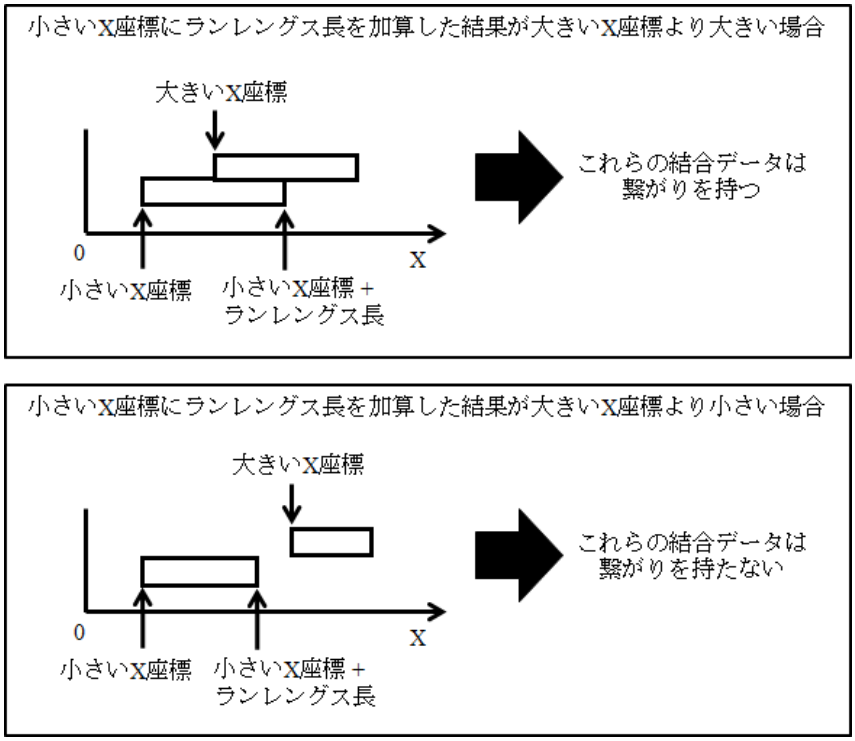


図 5.9: 結合比較条件 [32]

からの画素データは用いずにカメラ信号と，第 2 章で述べた画素生成回路を用いて実験を行った．実験概要としては，画素生成回路から生成されたデータに対してランレングス圧縮を行い，結合比較モジュールでラベル付を行うまでの時間をレイテンシーと定義し計測を行った．図 5.4 にランレングス圧縮，及びラベリング処理のレイテンシー計測部分を示す．ラベリング結果はラベル付けされたランレングスデータを FPGA 上に組み込んである NiosCPU を用いて，csv で取得し，Python で復元することで動作検証を行った．図 5.8 において，ランレングス圧縮を行い，FIFO・On Chip RAM2 への格納まではカメラクロックでの動作を想定し，25MHz で行い，読み出し以降を FPGA のシステムクロックの 200MHz で行った．画像サイズは一般的な解像度規格 VGA（640×480）で行った．

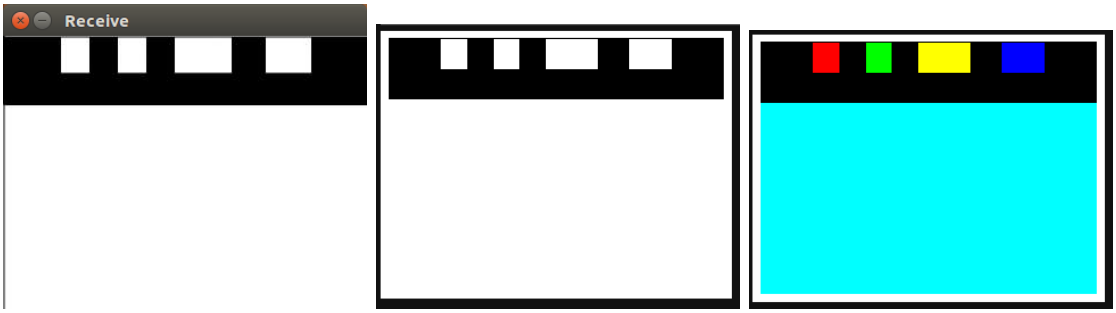
5.2.3. ハードウェアベースラベリングのレイテンシーの計測結果

レイテンシーの計測結果を表 5.1 に示す．ランレングス圧縮処理はラインバッファを用いる必要がなく，また処理自体にループ要素がないため，背景差分法同様にほぼレイテンシーをゼロで行うことができている．対して，ラベリング処理は On Chip RAM をループ処理で比較しているため，二値化処理やランレングス圧縮に比べ，レイテンシーが大きくなっている．しかし，単純な形のみのラベリングにのみ対応しているため，アルゴリズムが単純である．FIFO へ取り込み後は 200MHz で動作しているため，レイテンシー時間を 1,800 クロックに抑えている．この時点で左上と右下の X，Y 座標が取得できているということになり，この切り出し画像が畳み込みニューラルネットワークの入力画像となる．つまり，畳み込みニューラルネットワークモジュールは VGA 規格の画像から，切り抜き後の画像を数マイクロ秒オーダーで受け取ることが可能ということになる．また，図 5.10 に処理結果と FPGA の処理結果から復元した結果を示す．図 5.10(a) が今回使用

した画素生成回路で生成したテストデータである．図 5.10(b) がランレングス圧縮されたラベリング前のランレングスデータを復元した結果である．図 5.10(a) と比較すると，正しくランレングス圧縮されていることが確認できる．図 5.10(c) がラベル付きのランレングスデータを復元した結果であり，ラベル毎に色をつけている．これらから，ラベル付が正しく行われていることを目視で確認できた．

表 5.1: ラベリング処理のレイテンシー

Device	背景差分法 $[us]$	ランレングス圧縮 $[us]$	ラベリング処理 $[us]$
FPGA	0.01	0.18	9.20



(a) 画素生成回路で生成した画像　(b) ランレングス圧縮結果を復元した結果　(c) ラベル付きのデータを元に復元した結果

図 5.10: 転送システムの動作確認

第6章　おわりに

6.1.　研究成果のまとめ

本研究ではFPGAを用いてAIシステムを小型カメラデバイスに搭載し，リアルタイムAI処理の実現を目的としIoT向けAIカメラシステムの開発を行った．本稿では，AI処理の中でも，畳み込みニューラルネットワークのハードウェア化をターゲットとして，AI搭載型害獣捕獲システムへの適用を目的とした．深層学習は演算コストが大きく，高速化にはGPUが必須であるため，小型の組み込みデバイスへの実装は難しいといった点に焦点を当てた．はじめに，CPU・GPUのアーキテクチャについて比較を行い，メモリアクセスがリアルタイム性へのボトルネックとなっていることを示した．

次に，再構成可能デバイスであるFPGAのパフォーマンス性能が，他のアクセラレータに比べ優れていることを確認し，これを用いてFPGA向けパイプライン処理設計を実装し，ボトルネックとなっているメモリアクセスを行わない流し込み演算処理システムの提案・設計を行った．ARMプロセッサーとFPGAが1チップになったSoC FPGAを用いて，ハードウェア・ソフトウェア協調設計の開発を行った．FPGAを用いたハードウェアベースの画像処理システムの開発とともに，FPGA-HPS間のAXIバスを用いて，ARMプロセッサー側ではFPGAのコンポーネントがメモリマップされるようにすることで，HPS側でFPGA側の処理結果を受け取ったりFPGA部に対して制御を行ったりすることを可能にする仕組みの設計を行った．FPGA部でハードウェア処理が実行された画像処理結果のディスプレイをする仕組みや，UDP通信により，外部からARM・FPGAに対して制御を行う仕組みを実現した．さらには，ソフトウェア及びハードウェアをネットワーク経由で遠隔で書き換える仕組みの設計を行った．また，カメラ信号に同期して動作する画素生成回路を設計し，ハードウェア・ソフトウェア協調設計と組み合わせることで，システムのデバッグ・検証を容易にする仕組みも開発した．

その後，カメラのフレーム落ち無しに処理できている状態を本研究におけるリアルタイム処理と定義し，カメラ信号を用いて畳み込み処理のリアルタイム性の検証を行い，許容時間以内に処理を完了できることを示した．1層目のハードウェアベースの畳み込み処理結果をソフトウェア部でディスプレイを行い，ソフトウェアによるシミュレーション結果と同じであることを確認し，ハードウェア側で畳み込み処理が正しく行われていることも確認した．また，CPUとの処理速度の比較をしても大幅な処理時間の削減が確認でき，畳み込み処理のハードウェア化の有効性が示された．このことから，FPGAのリソースの観点からも畳み込み層のさらなる多層化にも対応が可能であることが示された．

畳み込み処理のハードウェア化の有効性が示されたため，次に，畳み込みニューラルネットワークの全結合層のハードウェア化を想定し，4層の多層ニューラルネットワークの設計を行った．そして，CPU及びFPGA上に実装し，FPGAの推論時間が設計通りになることを確認した．また，CPUに比べ優れた処理性能を発揮することを確認した．ま

た，FPGA 上のニューラルネットワークにおいては，ボトルネックとなっている部分の並列化を行うことで，設計通り推論時間を削減することができた．特に，入力層に近いほど，特徴量が多く逐次処理方式では処理時間を必要とし，入力層部分のニューロンの演算を並列化することで，大幅な処理時間の削減をすることができた．また，最終層において，完全並列化を行うことで，最終層の処理時間を 1 クロックで動作することが確認でき，完全並列化することで，クロック時間で 1 つの層を処理できることを確認した．

さらには，畳み込みニューラルネットワークの入力の前処理として，背景差分法による物体検出の実装も行った．背景差分法は他のフィルター処理とは異なり，ラインバッファを必要としないため，ほぼレイテンシーがゼロで処理すること可能であることを確認した．また，先行研究で行われていたラベリング処理のハードウェア化を，本研究におけるターゲットアプリケーションへの適用を考慮し，単純な形のみ対応した，走査回数を削減したラベリングモジュールを再設計し，優れた処理性能が示された．ランレングス圧縮も背景差分法と同様に，ラインバッファを必要としないため，ほぼレイテンシーがゼロで処理すること可能であることを確認した．パイプライン処理構造のラベリングモジュールにより，畳み込みニューラルネットワークは切り抜き後の画像を数マイクロ秒オーダーで受け取ることが可能であることを確認した．これにより，ラベリングの切り出しサイズの影響を受けずに処理を可能にすることができた．

6.2. 今後の研究課題と展望

今後の取り組みと展望について述べる．本研究では，畳み込みニューラルネットワークの各部分での動作検証のみを行った．そのため，本稿で行った実験を全て統合し，検証を行う必要がある．さらには，本研究でターゲットしていた害獣捕獲システムへ適用し，動作検証を行う必要がある．また，畳み込み処理でのレイテンシーが CPU に比べ非常に高い処理性能を示したことから，カメラデータのサイズ別の検証を行うことで，人工知能向け計算機技術の新たな展開も期待できる．同時に，本実験で用いたハードウェア・ソフトウェア協調設計によるカメラシステムは外部 PC との通信も行えるため，現場にいなくても遠隔で FPGA システム全体を管理したりと，IoT 機器として他のシステムとの連携をしたりすることが期待できる．畳み込み層においては，さらなる多層化によるリソースの検証，フィルターサイズ・種類の違いによるレイテンシーの検証を行う必要がある．また，多層ニューラルネットワークの実験においては，処理時間が設計通りであることが確認できたため，リソースの可能な限り並列化を行う必要がある．背景差分法による物体検出実験におけるラベリングモジュールの検証では，画素生成回路を用いたシンプルな形に対する検証のみを行った．そのため，実際のカメラ信号を用いて，背景差分法からランレングス圧縮，ラベリングまでを結合し，検証を行う必要がある．さらには，本研究では，HDL による設計のみに焦点を当てたが，高位合成による設計との性能比較を行うことが挙げられる．

謝辞

本研究を遂行するにあたり，多忙にもかかわらず数々のご指導・助言を賜りました高知工科大学大学院電子・光システム工学教室 星野孝総准教授に心から深く 感謝致します，そして, 本研究に関して多忙にも関わらずご助言してくださいました高知工科大学大学院電子・光システム工学教室 綿森道夫准教授，密山幸男准教授に深く感謝申し上げます，そして，高知工科大学システム工学群 Soft Intelligent System on Chip 研究室の皆様には, 日頃から様々な意見を頂き，ありがとうございます. 最後になりましたが，学費及び生活費など様々な面で 4 年間大学生活および 2 年間の大学院生活を支え続けてくれた両親，家族一同に深く感謝いたします.

参考文献

[1] 山根達郎, 全邦釘. Deep learning による Semantic Segmentation を用いたコンクリート表面ひび割れの検出. 構造工学論文集 A, Vol.65, pp.130-138, 2019.

[2] 中村太郎, 浅海賢一, 小森望充. FPGA を用いた Deep Learning による自動車ナンバープレートの認識. ロボティクス・メカトロニクス講演会講演概要集, No.2A2-L18, 2018.

[3] 菊田遥平, 染谷悠一郎. クックパッドにおける Deep Learning を用いた料理画像判別の取り組み. 人工知能学会全国大会論文集 第31回全国大会, No.1A1OS05a1-1A1OS05a1, 2017.

[4] Shuochao Yao, Yiran Zhao, Aston Zhang, Shaohan Hu, Huajie Shao, Chao Zhang, Lu Su, and Tarek Abdelzaher. Deep learning for the internet of things. Computer, Vol.51, No.5, pp.32-41, 2018.

[5] 榊原伸介. 知能ロボットによる工場自動化と IoT,AI 活用について. システム/制御/情報, Vol.61, No.3, pp.101-106, 2017.

[6] 納谷太 池邉隆 古川茂人. 山田武士, 高橋敏. NTT グループにおける AI 研究の取り組みと方向性. NTT 技術ジャーナル, pp.8, 2016.

[7] Alex Krizhevsky, Ilya Sutskever, Geoffrey and E Hinton. Imagenet classification with deep convolutional neural networks. Communications of the ACM, Vol.60, No.6, pp.84-90, 2017.

[8] 植木一也. 映像検索におけるディープラーニング. 日本神経回路学会誌, Vol.24, No.1, pp.13-26, 2017.

[9] Simon Kornblith, Jonathon Shlens, Quoc and V Le. Do better imagenet models transfer better? Proceedings of the IEEE conference on computer vision and pattern recognition, pp.2661-2671, 2019.

[10] 中山英樹. 深層畳み込みニューラルネットワークによる画像特徴抽出と転移学習. 信学技報, Vol.115, No.146, pp.55-59, 2015.

[11] 高田広章. 組込みシステム開発技術の現状と展望. 情報処理学会論文誌, Vol.42, No.4, pp.930-938, 2001.

[12] 中本幸一, 高田広章, 田丸喜一郎. 組込みシステム開発の現状: 1. 組込みシステム技術の現状と動向. 情報処理, Vol.38, No.10, 1997.

[13] 上野聡, 高橋恒一, 中田秀基. 特集「AI 計算資源」にあたって. 人工知能, pp.33, 2018.

[14] 黒滝紘生, 松尾豊. Deep learning 利用法と知見の体系化. 人工知能学会全国大会論文集 第 28 回全国大会, 2014.

[15] 中山浩太郎, 岩澤有祐, 黒滝紘生, 松尾豊. Deep Learning の実装と現状. 情報処理, Vol.56, No.11, pp.1102-1109, 2015.

[16] 伊藤祐貴, 松宮遼, 遠藤敏夫. ooc_cudnnGPU 計算機のメモリ階層を利用した大規模深層学習ライブラリの開発. 研究報告ハイパフォーマンスコンピューティング (HPC), Vol.38, pp.1-10, 2017.

[17] Vassili Kovalev, Alexander Kalinovsky, and Sergey Kovalev. Deep learning with theano, torch, caffe, tensorflow, and deeplearning4j: Which one is the best in speed and accuracy? 2016.

[18] 鈴木宏明, 富永浩文, 佐藤一馬, 中村あすか, 前川仁孝. GPU を用いた格子ボルツマン法のループ展開を利用したメモリアクセスの局所性向上による高速化. 第 77 回全国大会講演論文集, Vol.2015, No.1, pp.71-72, 2015.

[19] Zhilu Chen, Jing Wang, Haibo He, and Xinming Huang. A fast deep learning system using GPU. 2014 IEEE International Symposium on Circuits and Systems(ISCAS), pp.1552-1555, IEEE, 2014.

[20] 関谷翠, 大沢和樹, 長沼大樹, 横田理央. 低ランク近似を用いた深層学習の行列積の高速化. 研究報告ハイパフォーマンスコンピューティング (HPC), Vol.2017, No.24, pp.1-7, 2017.

[21] Shuichi Asano, Tsutomu Maruyama, and Yoshiki Yamaguchi. Performance comparison of FPGA, GPU and CPU in image processing. 2009 international conference on field programmable logic and applications, pp.126-131, IEEE, 2009.

[22] Nicolas Gac, StéPhane Mancini, Michel Desvignes, and Dominique Houzet. High speed 3d tomography on CPU, GPU, and FPGA. 2008.

[23] 中村祐一. ASIC と FPGA どちらを使いますか? 電子情報通信学会誌 The journal of the Institute of Electronics,Information and Communication Engineers, Vol.96, No.2, pp.85-89, 2013.

[24] 長尾祥一, 喜多ちえ, 浜本隆之, 相澤清晴. スマートイメージセンサと FPGA を用いた高速動物体追跡と奥行き推定. 映像情報メディア学会誌, Vol.57, No.9, pp.1142-1148, 2003.

[25] 大明準治. 1,000fps 高速度画像処理のアクティブカメラへの応用:通常照明・複雑背景での非特定移動物体トラッキング. 日本ロボット学会誌 Journal of Robotics Society of Japan, Vol.23, No.3, pp.282-285, 2005.

[26] Alexandre Muñiz Garcia, Roberto Fernández Molanes, Juan J, Rodríguez-Andina, and José Fariña. 100fps camera-based ugv localization system using Cyclone V FPGA SoC. IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society, pp.2789-2794. IEEE, 2018.

[27] Jeremy Fowers, Greg Brown, Patrick Cooke, and Greg Stitt. A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications. Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays, pp.47-56, 2012.

[28] 星野孝総, 松田一晃, 山本達也. 小型カメラと SoC FPGA を用いた高速画像処理システムの開発とリアルタイム性の検討. 日本知能情報ファジィ学会ファジィシステムシンポジウム講演論文集 第 32 回ファジィシステムシンポジウム, pp.259-264. 日本知能情報ファジィ学会, 2016.

[29] Hidetoshi Onuma, Yuji Yaguchi, Hiroshi Miyaguchi, Tsuyoshi Akiyama, Katsumi, Kajiyama, Kenya Adachi, Atsushi Kikuchi, Takao Kojima, and Takashi Narumi. Svp: scan-line video processor-general purpose video processor. 1995 International Symposium on VLSI Technology, Systems, and Applications. Proceedings of Technical Papers, pp.196-200. IEEE, 1995.

[30] 亀阪亮紀. 畳み込みニューラルネットワークを用いた害獣捕獲システムの試作と検討. 高知工科大学 卒業研究報告書, 2019.

[31] トランジスタ技術 2018 年 11 月号. CQ 出版; 月刊版 (2018/11/1), 2018.

[32] 島崎仁宏. SoC-FPGA を用いた外観検査システムのソフトウェア・ハードウェア協調設計と性能検証. 高知工科大学 修士論文, 2017.

[33] 石原ひでみ. ソフトウェア技術者のための FPGA 入門 機械学習編, インプレス R&D, 2017.

[34] 三好健文. FPGA 向けの高位合成言語と処理系の研究動向, コンピュータ ソフトウェア, Vol.30, No.1, pp.76-84 , 日本ソフトウェア科学会, 2017.

[35] Altera Corporation, "Cyclone V Device Handbook, Volume 3: Hard Processor System Technical Reference Manual," 31 December 2020. [Online]. Available: http://www.altera.com/literature/hb/cyclone-v/cv_5v4.pdf.

[36] 松田一晃. カメラを用いたリアルタイム物体追跡システムのためのハードウェアプラットフォームの設計, 高知工科大学 卒業研究報告書, 2016.

[37] 山本達也. カメラと SoC FPGA を用いたリアルタイム物体追跡システムソフトウエアの開発. 高知工科大学 卒業研究報告書, 2016.

[38] 藤井智也, 佐藤真平, 中原啓貴. 2 値化畳込みニューラルネットワークのニューロン刈りによるメモリ量削減と FPGA 実現について. 電子情報通信学会技術研究報告 IEICE technical report: 信学技報, Vol.117, No.221, pp.25-30.

[39] 中原啓貴, 笹尾勤, 岩本久. Nested RNS の定数除算を用いた深層畳込みニューラルネットワークの FPGA 実現について. 研究報告システム・アーキテクチャ (ARC), Vol.2016, No.39, pp.1-6, 2016.

[40] 井手秀徳, 栗田多喜夫. CNN における ReLU 活性化関数に対するスパース正則化の適用と分析. 電子情報通信学会論文誌 D, Vol.101, No.8, pp.1110-1119, 2018

[41] Tuan Linh Dang and Yukinobu Hoshino. Hardware/software co-design for a neural network trained by particle swarm optimization algorithm. Neural Processing Letters, Vol.49, No.2, pp.481-505, 2019.

[42] 中原啓貴. 畳込みニューラルネットワークの FPGA 実装. 電子情報通信学会誌, Vol.103, No.5, pp.501-506, 2020.

[43] 李寧, 富岡洋一, 宮崎昭彦, 北澤仁志. FPGA に適した浮動小数点演算器の構成法. 第 76 回全国大会講演論文集, Vol.2014, No.1, pp.117-118, 2014.

[44] 橋本耕太郎, 林悠平, 田中康一郎, 佐藤寿倫. FPGA による DSP 向け多機能メモリの実現. 情報処理学会研究報告システム LSI 設計技術 (SLDM), Vol.105, pp.45-50, 2003.

[45] 田中愛久, 黒柳奨, 岩田彰. FPGA のためのニューラルネットワークのハードウェア化手法. 信学技報, NC2000-179, 2001.

[46] Masahiro Shimasaki and Yukinobu Hoshino. Pipeline labeling algorithm with parallel calculation of gravity center on FPGA. 2016 Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS), pp.86-91. IEEE, 2016.

研究業績

1. 亀阪亮紀，星野孝総．”FPGA に向けた CNN の設計と評価” 2019 IEEE SMC HIROSHIMA CHAPTER 若手研究会 in 広島. 2019.7. 口頭発表

2. 亀阪亮紀，星野孝総．”FPGA2 段接続による CNN の設計とレイテンシー評価” VISION ENGINEERING WORKSHOP 2019 ViEW2019 in 横浜. 2019.12. ポスター発表

3. 亀阪亮紀，星野孝総．”多段接続 FPGA による畳み込みニューラルネットワークの設計とレイテンシー評価” 令和元年度 日本知能情報ファジィ学会中国・四国支部大会 in 高知. 2019.12. 口頭発表

4. Ryoki KAMESAKA and Yukinobu HOSHINO．”Design of the convolution layer using HDL and evaluation of delay time using a camera signal” IEEE WORLD CONGRESS ON COMPUTATIONAL INTELLIGENCE 2020 WCCI2020 at Online. 2020.7. 口頭発表

5. 亀阪亮紀，星野孝総．”HDL を用いた畳み込み層の設計とカメラ信号を用いたレイテンシー評価” 第 36 回ファジィシステムシンポジウム FSS2020 at Online. 2020.9. 口頭発表

6. Ryoki KAMESAKA and Yukinobu HOSHINO．”Design of the convolution layer using SoC FPGA and evaluation of latency using a camera signal” INTERNATIONAL CONFERENCE ON SOFT COMPUTING AND INTELLIGENT SYSTEMS 2020 SCIS 2020 at Online. 2020.12. 口頭発表

7. 亀阪亮紀，星野孝総． ”HDL を用いた畳み込み層の設計とカメラ信号を用いたレイテンシー評価”. 知能と情報（日本知能情報ファジィ学会誌）, Vol. 33, No. 1, pp.515-519, 2021.