# ハードウェア開発手法を用いない FPGA 実装による画像分類処理の高速化

佐藤　貴一　　（指導教員　西村　俊二）

平成30年1月26日

## The problem in using FPGAs without using hardware development method

TAKAKAZU SATO　　(ACADEMIC ADVISOR　SHUNJI NISHIMURA )

**概要**：近年，半導体集積回路に関するムーアの法則が終焉を迎えることが懸念されており，その対策として， 3次元集積技術や新しい構造のトランジスタの採用など，様々な技術開発が行われている．しかし，そのような対策に起因するコストの上昇が半導体業界の経済的限界をもたらしている．その一方で，単位時間当たりの計算量が多い処理を行うハイパフォーマンスコンピューティング分野では，データ発生源から絶え間なく到来するストリームデータ処理や，ディープラーニングなどの計算負荷が高い処理を高速化する演算器が求められている．このような状況下で，プログラマブル素子であるField Programmable Gate Array（FPGA）が注目されており，Hardware Description Language（HDL）などのハードウェア記述言語を使用して内部の回路を作りこむのが主流となっている．この手法では，ハードウェアに精通していないユーザは簡単にプログラミングすることができず，このことがハイパフォーマンスコンピューティング分野でリコンフィギャラブルシステムが活用されない場合の大きな要因となっている．そこで本研究では，ハードウェア記述言語や，回路設計などのハードウェアの知識を使用せずにFPGAを利用することで，学習済みニューラルネットワークを利用した画像分類アプリケーションの高速化を実現した．CPUによる処理と，AFIを用いたFPGAによる処理で370枚の物体画像を分類するプログラムの実行速度を比較した結果，FPGAでの処理を行った場合，CPUによる処理を行った場合に対して，約7.6倍の高速化効果が得られた．

キーワード: FPGA，画像分類処理，AWS

## 1. はじめに

### 1.1 背景と目的

「半導体の集積率は 1 年半~2 年で倍になる」というムーアの法則に従い，半導体の半導体技術開発はその予測どおりに約 50 年間発展してきたが，近年，終焉を迎えることが懸念されている [1]. 技術的には, 半導体特性ばらつきの急増による設計困難度の増大，消費電力の急増などが，問題としてあげられる．ムーアの法則を維持するために，3 次元集積技術，新しい構造のトランジスタの採用など，これらの問題点の解決へ向けた様々な技術開発が行われている [2]. しかし，そのような対策に起因するコストの上昇も含めた微細化によるプロセスコスト，マスク製造コストの急増，製造装置の価格高騰が，経済的限界をもたらし，これらのコストを気にすることのないような付加価値と需要の見込めるチップを除くと，今や半導体開発そのものが経済的に成り立たないような状況になりつつある．

その一方で，データ発生源から絶え間なく到来するストリームデータを逐次的に処理していく処理方法の拡大，ディープラーニングなどの計算負荷が高い処理を高速に処理できる演算器が求められている．そのような処理を行う主な演算器として，汎用プロセッサである CPU と, 大規模な並列処理を行い処理の高速化を実現する Graphics Processing Unit（GPU）があげられる．しかし，両演算器において高い演算性能と低消費電力の両立が課題であった [3].

このような状況下で， Field Programmable Gate Array（FPGA）をはじめとするプログラマブルな素子の重要性が一層増大している．FPGA は，クロック周波数が CPU に比べて大幅に小さいので, 低電力で高性能な処理を行うことができる [4].

ストリーム処理や, 計算負荷の高いアルゴリズムは, リコンフィギャラブルシステムにより高速化を実現することが可能である．リコンフィギャラブルシステムは，FPGA をはじめとする書き換え可能なデバイス上に対象とするアルゴリズムを直接ハードウェア化して実行することにより，ハードウェアの速度性能とソフトウェアの柔軟性を共に実現することを可能にするシステムであり，近年の FPGA の急速な発展とともにその応用分野を広げている [5].

現在のハイ・パフォーマンス・コンピューティング用のリコンフィギャラブルシステムの開発は，Hardware Description Language（HDL）や very high speed

integrated circuits HDL（VHDL）などのハードウェア記述言語を使用して，内部の回路を作り込むのが主流となっている．この手法は，ハードウェアに精通していないユーザが，簡単にプログラミングすることは困難である，という点がリコンフィギャラブルシステムをハイパフォーマンスコンピューティング分野で用いる場合の大きな妨げとなっている．

本研究では，クラウドサーバ上で扱える FPGA を用いて，ハードウェア記述言語や回路設計などのハードウェアの知識を使用せずに，一般物体認識用の学習済みニューラルネットワークの推論を利用した，画像分類の処理を行うソフトウェアアプリケーションのハードウェアアクセラレーションを行い，ハードウェアに精通していないユーザでもリコンフィギャラブルシステムの実装が可能であるか考察する．

### 1.2 本論文の構成

本論文の構成は以下の通りである. 2 章では FPGA の概要について述べ，3 章では FPGA の論理設計の開発手法について述べる．4 章では FPGA によって高速化する学習済みニューラルネットワークの推論を用いた画像分類アプリケーションの実装方法および処理速度の評価について述べる．5 章で実験結果について考察し，6 章で本論文の結論をまとめる.

## 2. FPGA の概要

本章では，2.1 で一般的な FPGA の仕組みと特徴について簡単に述べ, 2.2 で本研究で用いた，クラウドで使用できる FPGA について述べる.

### 2.1 FPGA の仕組みと特徴

ムーアの法則の終焉が囁かれ始めたこの数年，汎用プロセッサである CPU の代わりに, 用途に特化したプロセッサを使用しようとする動きが盛んになってきている．代表的な例が，本研究で使用する Field Programmable Gate Array(FPGA)である.

FPGA とは, ユーザが必要に応じて機能を書き換えることが可能な LSI である．製品出荷後でも，論理設計の再構成が可能なため，製品のアップデートや新たなプロトコル規格への対応を素早く行うことができる.

FPGA の中身は回路を構成するためのロジックや Digital Signal Processor（DSP），Block RAM（BRAM）や専用のハードウェアマクロなどがあり，これらを組み合わせて専用の回路を作ることができる．BRAM は FPGA 上に均等に配置され，必要に応じて RAM として利用することができる. ロジックブロックは書き換え可能な Look up Table（LUT）とフリップフロップから構成されている．FPGA の内部構成を図 1 に示し [6],

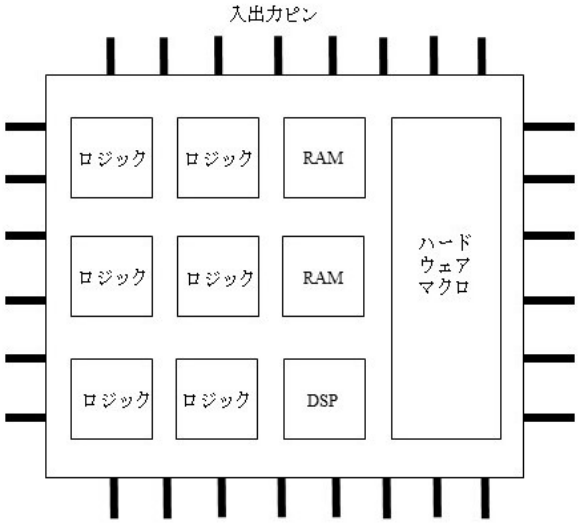


図 1　FPGA の構成

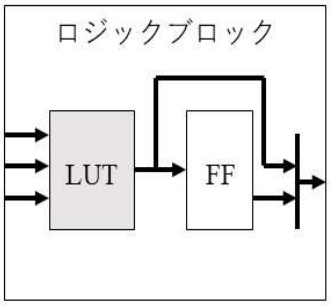


図 2　ロジックブロック内部の構成

ロジックブロックの内部構成を図 2 に示す.

また, FPGA では処理を並列して行うことができる．CPU ではプログラミングされた命令を逐次実行していくが，FPGA では同時にいくつでも並列して処理する回路を設計することができる．例えば，リスト 1 のように for ループ文で 100 回の乗算をするようなソースコードの場合，一般的に CPU では 100 回の乗算を繰り返し行うが，FPGA ではリスト 2 のように 100 個の乗算器を並べて配置し一度に並列処理を行うことが可能である [6].

リスト 1　for ループでの演算

```
for( i = 0; i < 100 ; ++i) {
   z [ i ] = x [ i ] * y [ i ];
}
```

リスト 2　並列に並べられた演算

```
z [0] = x [0] * y [0];
z [1] = x [1] * y [1];
z [2] = x [2] * y [2];
. . .
z [98] = x [98] * y [98];
z [99] = x [99] * y [99];
```

### 2.1.1 CPU と FPGA の違い

CPU でプログラムを実行する場合，記述されている命令順に CPU コアの命令処理回路で処理が行われる．

プログラミングされた命令はメモリから読みだされ，デコード，実行，メモリに書き戻す一連の動作を繰り返す．CPU は，OS のように複雑なプログラムを実行することを念頭において作られているため汎用的な処理が可能だが，演算器間でメモリを共有する仕組みであるため，コアの増加に伴い性能が頭打ちになってしまう．CPU で高速に処理するには，クロック周波数を向上させ，また，プログラムを改良するなどの措置をとらなければならない．

一方，FPGA は内蔵されたロジック，乗算器，RAM などを組み合わせて専用の回路を処理の順番に構成し，接続して動作させる．各専用回路は並行して動作し，入力信号が送られるとすぐに動作する．内在する並列性と規則性をハードウェア化することにより，効率よく高性能計算を実現することが可能である．

CPU と FPGA での処理の違いを具体例を用いて見てみる．リスト 3 に示す演算式を用いて違いを示す．

リスト 3　演算式の例

```
y = ( a * b ) + ( c * d )
```

CPU のような処理はリスト 4 のように CPU コアにあるレジスタ（r0-r2）にメモリから読み込んだ値を代入して演算回路で逐次実行する．

リスト 4　CPU での実行例

```
r0 = a
r1 = b
r2 = r0 * r1
r0 = c
r1 = d
r0 = r0 * r1
r2 = r2 + r1
y = r2
```

FPGA では，図 3 のような回路構成にすると，リスト 3 の演算を一度に完了させることが可能である．

また，ストリームデータを処理する場合は，図 4 のように算回路毎に分割し，クロックで同期させる回路を構成することも可能である．この場合は，乗算器および加算器は同じ時間軸で処理され，結果はそれぞれの演算器を順番に伝達される．

処理回路が多いほど結果が出力されるまでに時間がかかるため，一概にどの回路構成が良いかはシステムの設計思想で異なってくる．FPGA での開発では絶えずこのような並列性の処理を考えながら論理設計を行っていかなければならない．

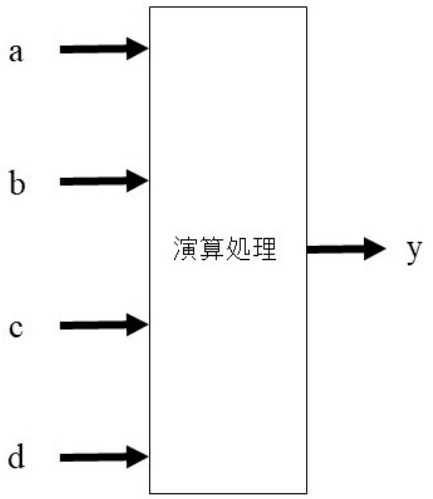


図 3　FPGA で演算を一括処理した場合

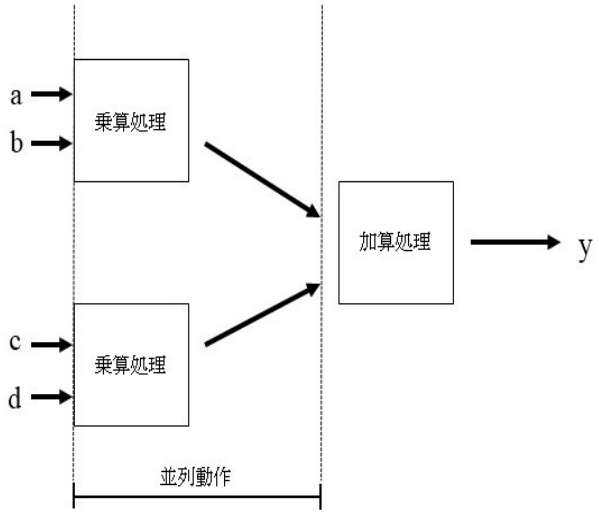


図 4　FPGA で演算処理ごとにブロックを分けた場合

また，CPU は汎用プロセッサであることから，使用していない回路も動作させたままになるが，FPGA は使用しない回路を実装する必要がないため，CPU コアのように無駄な回路を動作させておく必要が無くなる．さらに，同じスループットでも低い動作周波数であることから，消費電力を抑えることにつながる．

### 2.1.2 応用例

FPGA は，大規模の並列処理による高速動作に優れていることから，大規模整数演算，ハッシュ処理，特長量検出などのパターンマッチや暗号化/複合化，圧縮，展開処理，動画や音声のエンコーディング/デコーディング/トランスコーディング，フィルタリングなど，CPU が不得手なビットストリームに対するパイプライン処理が適している．加えて，ネットワーク処理や，ディープラーニングの一種である CNN（畳み込みニューラルネットワーク）などの処理を効率化しスループットの向上に貢献する．

## 2.2 AWS F1 インスタンス

F1 インスタンスとは，2016 年に Amazon Web Services(AWS)がリリースした，FPGA を搭載した仮想サーバのことである. F1 インスタンスは, 1 個の FPGA を搭載している F1.2xlarge と，8 個の FPGA を搭載している F1.16xlarge の 2 種類が存在する．F1 インスタンスには，16nm Xilinx UltraScale Plus FPGA の VU9P というデバイスが搭載されており，それぞれの FPGA は，64GiB DDR4 ECC メモリと，専用の PCIe x16 接続を備えている．また，それぞれの FPGA には，約 250 万個の論理素子と約 6800 個のデジタルシグナルプロセスエンジンが搭載されている．クラウドサービスを利用するため，使用したコンピューティング性能に対して時間単位の料金が生じる．

表 1 に 16nm Xilinx UltraScale Plus FPGA の詳細情報を示し，表 2 に F1 インスタンスの詳細情報を示す.

表 1　16nm Xilinx UltraScale Plus FPGA(VU9P)の情報

| | |
|---|---|
| 論理セル(k) | 2,586 |
| デジタルシグナル プロセスエンジン | 6840 |
| フリップフロップ(k) | 2,364 |
| LUT(k) | 1,182 |
| 分散 RAM(Mb) | 36.1 |
| ブロック RAM(Mb) | 75.9 |
| UltraRAM(Mb) | 270.0 |
| Max.Single-Ended I/Os | 832 |

表 2　F1 インスタンスの情報

| インスタンスタイプ | F1.2xlarge | F1.16xlarge |
|---|---|---|
| FPGA ボード | 1 | 8 |
| vCPU | 8 | 64 |
| インスタンスメモリ (GiB) | 122 | 976 |
| SSD ストレージ (GB) | 470 | 4 x 940 |
| 1 時間当たりの料金 | $1.65 | $13.20 |

F1 インスタンスに搭載されている CPU は，Intel Broadcast E5 2686 v4 プロセッサである.

比較的規模の大きい FPGA を使用できるため，多くのハイ・パフォーマンス・コンピューティングアプリケーションにハードウェアアクセラレータとして活用される．従来のボード FPGA でも活用されていた，ネットワーク処理などに加え，金融分析やリアルタイム動画処理，ビッグデータ検索と分析，ディープラーニングなどの時間的に制約のあるアプリケーションに適している.

# 3. FPGA の開発手法

本章では，FPGA の論理設計の手法について説明する．3.1 で HDL を用いた開発手法について述べ，3.2 で高位合成を用いた開発手法について述べる．3.3 は，本研究で使用した，既存の AFI を用いた開発手法について述べる.

## 3.1 HDL を用いた開発

### 3.1.1 HDL を用いた開発の特徴

FPGA に所要の機能を実装する方法には，回路図を作成する方法や状態遷移図を定義する方法など様々なものがあるが，一般的にユーザは HDL（Hardware Description Language : ハードウェア記述言語）を用いて所要の機能を記述し，FPGA ベンダが提供するツールを用いて FPGA 上に実装する.

HDL は，電子回路やシステムの経時的ふるまいと，空間的構造を記述するためのプログラム言語である [7]．VHDL や Verilog-HDL を用いるのが主流である．HDL を用いることにより, 配置配線などの部品割り当て設計は後回しにし，論理設計に集中することができる.

### 3.1.2 開発フロー

FPGA 開発は一般的に HDL で論理回路を構成し, 論理合成と配置配線を行って FPGA の内部を構成するバイナリファイルを作成する.

論理合成は, HDL によるソースコードをハードウェア素子に変換することである．その後，LSI 向けに変換された回路を FPGA 内部で実際にどのように埋め込むかを決定し，配置配線の作業を行う．FPGA 開発ツールによっては論理合成と，次の工程である配置配線をひとまとめにして, コンパイルと呼ぶ. 図 5 に FPGA の開発フローを示す［6］.

また，FPGA 開発ツールは FPGA ツールベンダー各社が様々なものを提供している．主要なツールをまとめたものを表 4 に示す.

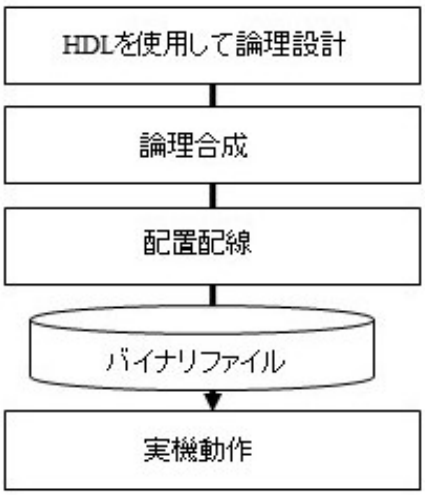


図 5　FPGA の開発フロー

表 4　開発ツール

| FPGA ツールベンダー | 開発ツール |
|---|---|
| ザイリンクス株式会社 | Vivado |
| インテル株式会社 | Quartus |

## 3.2 高位合成を用いた開発

### 3.2.1 高位合成の特徴

一般に FPGA の回路は，HDL を用いて機能を設計し，それを論理合成ツールに入力して生成する．しかし HDL を用いた設計には，ソフトウェア設計にはない難しさが存在する．処理をクロック単位で記述しなければならない点に加え，動作の周波数や信号が到達するタイミングを意識しなければならず，検証やデバッグが難しい [8].

FPGA の規模と動作クロックは年々向上しており，それに伴い 1 クロックで処理する内容やタイミングを考慮した設計が難しくなってきている．また，Register Transfer Level（RTL：レジスタ転送レベル）の検証やデバッグは，シミュレータと呼ばれる回路を模擬動作するツールを用いて行われるが，シミュレータは動作が非常に遅いため多数の入力パターンに検証に時間がかかってしまうことがある．更には，直感的で手軽なデバッグを行う機能も用意されていない．

そこで，これらの困難を克服し，ソフトウェア設計するように FPGA の回路を設計する手法として注目されているのが高位合成という技術である．

高位合成は，回路設計を HDL ではなく C 言語や C++ 言語，Python といった高級プログラミング言語を用いて記述する．それらの言語で記述したアプリケーションコードを高位合成ツール入力して，関数単位で HDL に変換し，出力させることで FPGA の回路を生成する．

高位合成には，プログラミング言語を用いて記述するという特徴から HDL よりも高い抽象度で設計を可能とし，プログラミング言語の段階で検証やデバッグを行うこと可能であるという利点が存在する [8].

プログラミング言語はクロックに関する記述を行う必要がないため，設計者はクロック周波数や信号が到達するタイミングを細かく意識せずに，回路の機能を設計することができる．どのクロックでどの処理を行うかは，高位合成ツールが自動的に判断する．

また，機能の検証やデバッグも変数の参照やステップ実行といった，ソフトウェアのデバッグ機能をそのまま用いることができるため，作業効率を高めることができる．

### 3.2.2 開発フロー

HDL を用いた開発手法と異なる点は，C 言語や C++ 言語，Python などの高級言語を用いて論理回路を記述することが可能である，という点である．高級言語を用いてソースコードを記述する段階で検証やデバッグが可能なため，HDL を用いた開発手法と比較して大幅な工数削減を実現することができる．HDL を用いた開発手法と高位合成を用いた開発手法も，論理設計より後ろのフローは変わらない．高位合成を用いた場合でも論理合成や配置配線は行わないといけないため，ハードウェアに精通していないユーザには難易度が高くなるという問題点が存在する．

図 6 に高位合成の開発フローを示す [6].

また，高位合成ツールは，FPGA ツールベンダー各社が様々なものを提供している．主要な高位合成ツールをまとめたものを表 5 に示す．

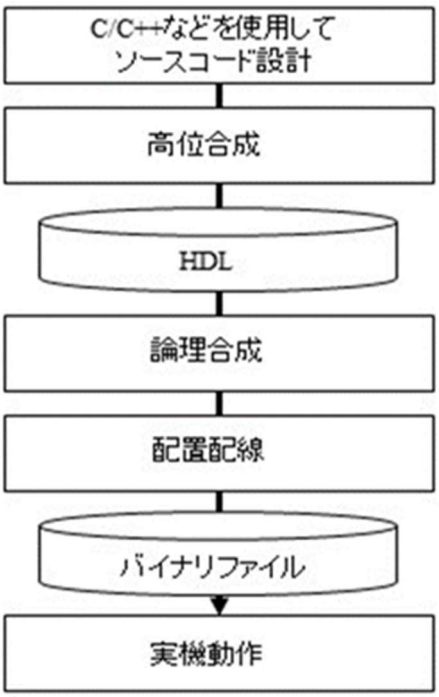


図 6　高位合成の開発フロー

表 5　高位合成ツール

| FPGA ツールベンダー | 高位合成ツール |
|---|---|
| ザイリンクス株式会社 | Vivado HLS<br>SDx（SDSoC , SDAccel） |
| 日本電気株式会社 | CyberWorkBench |
| メンター・グラフィックス・ジャパン株式会社 | Catapult C |
| 日本シノプシス合同会社 | Synphony C Compiler |
| 日本ケイデンス・デザイン・システムズ | C-to-Silicon Compiler |

### 3.3 AFI を用いた開発手法

F1 インスタンス上では，FPGA をプログラムするために作成する設計データを Amazon FPGA Image（AFI）と呼ぶ.

AFI は，AWS 独自のフォーマットであり，内部にはビットストリームデータを含んでいる. AFI を F1 インスタンスに搭載されている FPGA にロードすることで，ハードウェアアクセラレーションを実現することができる.

AFI は，クラウドサーバ上の Hardware Development Kit（HDK：ハードウェア開発キット）を用いて，開発することが可能である. HDK はザイリンクス株式会社の FPGA 開発ツール，コードサンプル，コンパイルスクリプト，デバッグインターフェースなどの F1 インスタンスの FPGA コードの開発に必要な多くのツールを含んでいるため，クラウドを用いない開発手法と開発ツールを変えることなく FPGA の論理設計を行うことができ，独自に開発した AFI は何度でも再利用ができる．加えて，ザイリンクス株式会社の FPGA 開発ツールを無料で使用できるライセンスも提供されているため，ライセンス購入，登録などの手間を省くことが可能である.

また，独自の設計データを AFI として登録する方法の他に，AWS の Marketplace で公開されている AFI を利用することができるため，一般的なハードウェアアクセラレーションをすばやく実装することができる．よって，従来の FPGA 開発と比較して，大幅な工数削減を実現することができる.

本研究では，Marketplace に公開されている学習済みニューラルネットワークを用いた画像分類アプリケーションの推論処理を高速化する AFI を用いる.

図 7 に AFI を用いる場合の開発フローを示す.

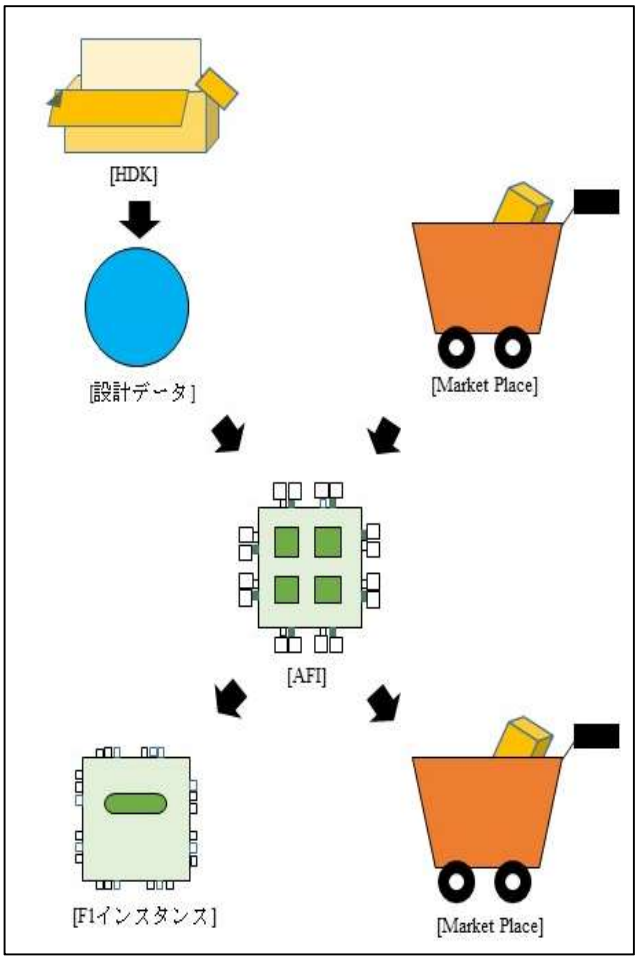


図 7　AFI を用いる場合の開発フロー

## 4.　実験内容

本章では, 4.1 ディープラーニングのフレームワークを用いた，画像分類アプリケーションの開発環境を述べ，4.2 でその実装方法を述べる．4.3 で，実装した画像分類アプリケーションに，AWS の marketplace で公開されている AFI を適用し, 推論処理の高速化を行う．4.4 で，CPU の場合と AFI を用いた場合の実行時間測定を行う.

### 4.1　開発環境

画像分類アプリケーションの実装や AFI のロード，プログラムの実行は AWS のクラウドサーバ上で行う．F1 インスタンスへのログインは, SSH クライアントソフトを用いて SSH 接続することでできる. 本研究で用いた SSH クライアントソフトは，Tera Term である．F1 インスタンス上で行う作業はすべて，Tera Term に表示される．また，本実験で使用しているローカル環境 PC の OS は Windows だが，F1 インスタンス上での環境は Linux となっている.

### 4.2　画像分類アプリケーションの実装

本研究では，オープンソースのディープラーニング

フレームワークである Caffe を使用して，画像分類アプリケーションの実装を行う．

Caffe は, カリフォルニア大学バークレー校のコンピュータビジョンおよび機械学習の研究センターである Berkeley Vision Learning Center（BVLC）が中心となって開発している [9]．Caffe は C++言語や Python，Matlab から使用することが可能であり，OpenCV や OpenBLAS といった高機能ライブラリを駆使して他言語より高速に，画像認識に向けた深層学習を可能にしている機械学習ライブラリである．

本実験で実装する画像分類アプリケーションの画像分類処理や，一般物体認識用の学習済みニューラルネットワークのロードを行うプログラムは，C++言語で記述する．

#### 4.2.1 Caffe Net

本研究では，CaffeNet を用いる．Caffe net とは，ImageNet Large Scale Visual Recognition Challenge 2012 の分類タスクで優勝した AlexNet に，normalization と pooling の順番を変えるなどの変更を加えたものである [10]．ImageNet Large Scale Visual Recognition Challenge は, 数百ものオブジェクトカテゴリと数百万の画像に対するオブジェクトカテゴリの分類と検出のベンチマークである．ImageNet Large Scale Visual Recognition Challenge は，2010 年から現在まで毎年実施され，50 以上の機関が参加している [10].

AlexNet は，ImageNet のデータを元にした ImageNet Large Scale Visual Recognition Challenge のデータを使用しており，5 層の畳み込み層と 3 層の全結合層の 8 層からなる．表 6 に学習と検証，評価に用いられている画像データの枚数を示す [11].

表 6　画像データの枚数

| | |
|---|---|
| 学習（training） | 120 万画像 |
| 検証（validation） | 5 万画像 |
| 評価（test） | 15 万画像 |

画像の前処理として，以下の処理を行っている．

・解像度の正規化：
　畳み込みニューラルネットワークは，固定長ベクトルを入力とするため, 画像を 227×227 に変更する．短編を 227 ピクセルになるように縮尺を変え中心部分の 227×227 領域を切り出す.
・Per-pixel Mean Substraction: :
　ピクセル・チャンネルごとに平均を求め，入力から差し引く．

Caffe には，Transformer という前処理のためのクラスが付属している．

### 4.3 AFI の適用

本研究では，ハードウェア記述言語を使用せず，また，配置配線も行わずに FPGA を用いたリコンフィギャラブルシステムの実装を行うため，独自に開発した AFI は用いない. AWS の Market Place が提供している，ZEBRA on 1 FPGA image classification という，AFI を利用して，一般物体認識用の学習済みニューラルネットワークを用いた画像分類アプリケーションの分類処理を高速化する．

画像分類アプリケーションに，ZEBRA on 1 FPGA image classification をリンクさせ，実行することでハードウェアアクセラレーションを実現する．リンクする際に，ZEBRA on 1 FPGA image classification に含まれる， AFI をロードするための，C 言語で作成された共有ライブラリを使用する．

### 4.4 速度比較

実装した画像分類アプリケーションを使用して 370 枚の物体画像を分類し，CPU で実行する場合と，AFI を用いて FPGA で実行する場合との速度比較を行う．

本実験では，インテル® Xeon® プロセッサ E5-2695 v4 メモリが搭載されている CPU を用いた.

インテル® Xeon® プロセッサ E5-2695 v4 メモリのベース動作周波数は，2.10 GHz である．

CPU で処理した場合の実行時間の測定結果と AFI を適用した場合の実行時間の測定結果を表 7 に示す．表 7 の実行時間は，プログラム自体の処理時間であるため，プログラムの呼び出しと終了の時間は含まない．また，呼び出しと終了の時間を含んだ結果を表 8 に示す.

表 7　実行時間

| 計算器 | プログラムの実行時間[s] |
|---|---|
| CPU | 76.068 |
| FPGA | 10.045 |

表 8　呼び出し，終了の時間を含んだ実行時間

| 計算器 | 呼び出しと終了の時間を含んだプログラムの実行時間[s] |
|---|---|
| CPU | 76.321 |
| FPGA | 28.791 |

## 5. 考察

図 8 に示すように, CPU による処理と AFI を用いた FPGA による処理で 370 枚の物体画像を分類するプログラムの実行速度を比較した結果，FPGA で処理を行った場合が 76.068[s]で，CPU による処理を行った場合

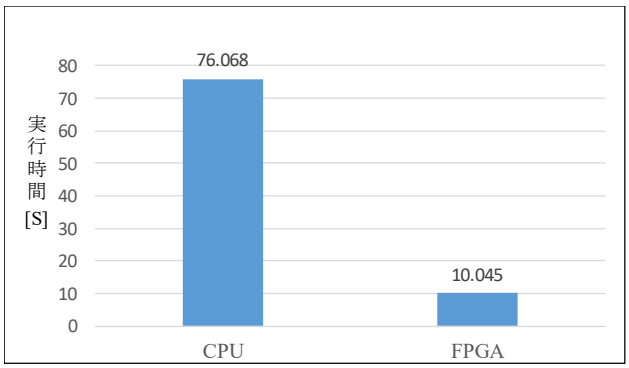


図 8　CPU と FPGA の実行時間比較

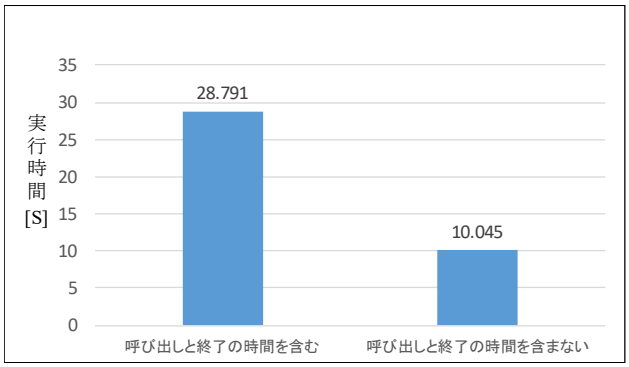


図 9　呼び出しと終了の時間を含む場合と含まない場合の実行時間比較(FPGA)

が 10.045[s]であるため，約 7.6 倍の高速化効果が得られた．

また，CPU の呼び出しと終了の時間を含んだ場合の実行時間と，含まない場合の実行時間に大きな差はないが，図 9 に示すように FPGA で実行した場合は 18.746[s]の差が発生している．この理由として，AFI を FPGA にロードしている時間であると考えられる．

## 6. おわりに

本稿では，ハードウェアに精通していないユーザがリコンフィギャラブルシステムのプログラミングを行うことは困難であるという問題に対して，HDL で記述する論理設計や配置配線などのハードウェア開発手法を用いずに FPGA の実装を行った．実装した FPGA を使って一般物体認識用の学習済みニューラルネットワークの推論を用いた画像分類アプリケーションの処理速度について，約7.6倍の高速化を行うことができた．

本研究の今後の課題について以下に述べる．

(1) 既存の AFI では適用するアプリケーションが限られるため，独自に開発した AFI を用いてハードウェアアクセラレーションを実現させる．
(2) ディープラーニングは，学習に膨大な時間を必要とするため，推論ではなく，学習の処理に適用できる AFI の開発を検討する．
(3) Caffe Net のような既存のニューラルネットワークだけではなく，独自のニューラルネットワークの推論の高速化を行う．

## 謝辞

本研究を進めるにあたり，様々なご指導を頂いた指導教員の西村俊二講師に深謝いたします．また，この研究の機会をくださった情報工学科の先生方，そして多くの知識やご指摘を下さいました西村研究室の皆様に厚く御礼申し上げます．

### 参考文献

[1] 福井正博，林憲一，“GPU による LSI 設計の高速化技術，” 電子情報通信学会 Fundamentals Review Vol.6 No.3 , 2013.

[2] 渡辺重佳，玉井翔人，佐藤匠，廣島佑 ，横田智広，“ムーアの法則以降の新しい半導体メモリとトランジスタの技術動向，” 湘南工科大学紀要 50(1), 2016-03.

[3] 立見駿介，張山昌論，伊藤康一，青木孝文，“OpenCL を用いた FPGA ベース画像処理プロセッサの設計，”公益社団法人 計測自動制御学会, 2015.

[4] S. Shimada and H. Kitajima, "Development of nonlinear analysis tools on FPGA," Nonlinear Theory and Its Applications, IEICE Vol.8, no.2,pp.173-179, 2016.

[5] Y. ATSUMARI, H. KUBO, Y. OGISHIMA, S. YOKOTA, H. TAMUKOH , M. SEKINE, “A Multidimensional Configurable Processor Array,” IEICE TRANS.INF.&SYST, Vol.E98-D,No.2,pp.313-324, 2015.

[6] 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]，” 第 1 編, 山城敬, 編, 株式会社インプレス R&D, 2017.

[7] J. Zhang, Q. Zhao, M. Kuga, M. Amagasaki, M. Iida , T. Sueyoshi, “A Comparison of Sorting Algorithms with FPGA Acceleration by High Level Synthesis,” 電気・情報関係学会九州支部連合大会講演論文集, 2014.

[8] 若林一敏，“ソフトウェアプログラムからハードウェア記述を合成する高位合成技術，” 電子情報通信学会，Fundamentals Review Vol.6, No.1, pp.37-50, 2012.

[9] 佐々木拓郎，林晋一郎，小西秀和 ，佐藤瞬，“一番大切な知識と技術が身につく Amazon Web Services パターン別構築・運用ガイド，” 第 2 編, SB クリエイティブ株式会社, 2017.

[10] 武井宏将, “初めてのディープラーニング, ” 松本昭彦, 編, 株式会社リックテレコム, 2017.

[11] A. Krizhevsky, I. Sutskever , G. E.Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” NIPS, 2012.

[12] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg , L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” 2015.