Vitis Vision Libraryを用いた画像処理のハードウェア実装と性能評価

指導教員

甲斐 博 准教授

高橋 寛 教授

令和 5 年 2 月 9 日提出

愛媛大学工学部工学科

応用情報工学コース

計算機/ソフトウェアシステム研究室

学籍番号：9520315M

西川　竜矢

目 次

第 1 章　序論 **1**

1.1　研究背景 1

1.2　論文の構成 2

第 2 章　準備 **3**

2.1　画像処理 3

2.1.1　物体検出 3

2.1.2　物体検出の詳細 3

2.1.3　OpenCV 4

2.2　FPGA 4

2.2.1　FPGA 概要 4

2.2.2　Ultra96v2 5

2.3　ハードウェアアクセラレーション 5

第 3 章　エッジ AI プラットフォーム開発 **6**

3.1　Vitis プラットフォーム概要 6

3.2　Vitis プラットフォーム開発環境とツール 6

3.3　Vitis プラットフォーム作成 7

3.3.1　HW(Hardware) Component 7

3.3.2　SW(Software) Component 10

3.3.3　Vitis Target Platform 12

3.4　Vitis Vision Library 13

第 4 章　アプリケーションの実装 **14**

4.1　アプリケーション作成 (PS) 14

4.2　アプリケーション作成 (PS+PL) 15

第 5 章　評価実験 **18**

5.1　実験準備 18

5.1.1　データセット 18

5.1.2　SD_Card の作成 19

5.1.3　実験環境の構築 20

5.2　アプリケーションの実行 20

5.2.1　アプリケーションの実行 (PS) 20

5.2.2　アプリケーションの実行 (PS+PL) 21

5.3　実験結果 22

第 6 章　考察 **24**

第 7 章　あとがき **25**

謝辞 **26**

参考文献 **26**

第1章　序論

1.1　研究背景

近年，日常生活を送るうえで多くの場面で IoT が活用されている．また，エッジ AI の登場により，IoT 機器における処理全体に要する時間の短縮に成功している．その中でも，自動車の歩行者検知や製造業における外観検査などに使われている技術に物体検出がある．物体検出には高いリアルタイム性が求められているが，IoT 機器におけるエッジデバイスは推論モデルの学習環境に用いられる PC と比較すると CPU 性能やメモリ容量といったリソースが劣る [1]．こうした現状から，限られたリソースの中で，処理速度やリソース使用量などのパフォーマンスをどれだけ向上させられるかが課題となっている．

物体検出の流れとしてはじめに入力画像に対する前処理を行い，ここで整形された画像データを用いて推論処理を行う．これらの工程において，入力画像に対する前処理よりも，推論処理の方が処理に要する時間が大きくなる．そのため，前処理に対する高速化の研究よりも推論処理に対する高速化の研究の方が盛んである．推論処理に対する高速化手法は，主流な方法である量子化をはじめ，レイヤフュージョン，デバイス最適化，マルチスレッド化などさまざまである [2]．今後さらに推論処理の高速化が進んでくると，高速化された推論処理に対して，その前に処理速度の遅い前処理の工程が入ってくるため，折角高速化した推論処理のパフォーマンスを最大限に発揮できなくなってしまう恐れがある [3]．

加えて，エッジコンピューティングにおいて注目すべきはそのエッジデバイスである．FPGA（Field Programmable Gate Array）は，コンピュータをはじめ通信機器やプリンタなど多くのエッジデバイスに搭載され用いられている．その大きな特徴として現場で論理回路の構成を書き換え可能である点が挙げられる．また，FPGA は大量のデータを高速に処理し，さらに消費電力が低いという利点がある [4]．

以上の点から，本研究では，物体検出におけるリアルタイム性を課題として取り上げ，入力画像に対する前処理を高速化することを目的とする．目的達成のために画像の前処理に対して FPGA を用いたハードウェアアクセラレーションを実装することを目標とする．高速化した前処理と，そうでない前処理に対して処理速度の計測を行い，ハードウェアアクセラ

レーションによりどの程度処理時間を短縮できたかを検証する．

1.2 論文の構成

本論文では，第 1 章では研究背景について，第 2 章では本論文を読むにあたって必要となる基本的な知識を述べる．第 3 章ではエッジ AI で動作するアプリケーションのためのプラットフォームの概要並びに作成手順について述べる．第 4 章ではエッジ AI で動作するアプリケーションの開発について述べ，第 5 章では実験評価と題して，作成したアプリケーションを実際に動作させる環境ならびに実行結果について述べる．第 6 章では第 5 章の実験結果に対して考察を行い，第 7 章では本研究全体を通したあとがきについて述べる．

第2章　準備

本章では，この論文を読むにあたって理解しておく必要のある予備知識について述べる．

2.1　画像処理

2.1.1　物体検出

機械学習を活用して画像に映る特定の物体を検出する技術を物体検出 [5] と呼ぶ．物体検出における大まかな流れを図 2.1 に示す．

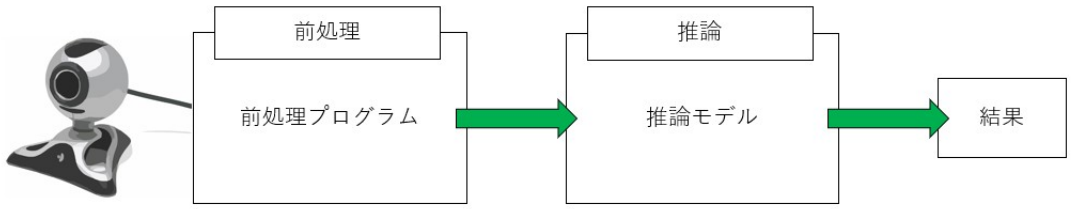

図 2.1: 物体検出の流れ

物体検出の流れとして，はじめにカメラから入力画像を取得する．この画像に対して画像前処理を行う．この処理は画像データの整形を行い，整形したデータを推論モデルに渡す．整形された画像データを受け取った推論モデルは推論処理を行い，推論結果を出力する．こうした流れで物体検出技術は実現されており，製造業や自動車分野，さらには医療分野など幅広い分野で利用されている．

2.1.2　物体検出の詳細

本研究における物体検出の詳細について示す．本研究では推論モデルに，YOLOv3 tiny および YOLOv4 [6] を想定しており，入力画像に対する前処理はリサイズを行う．

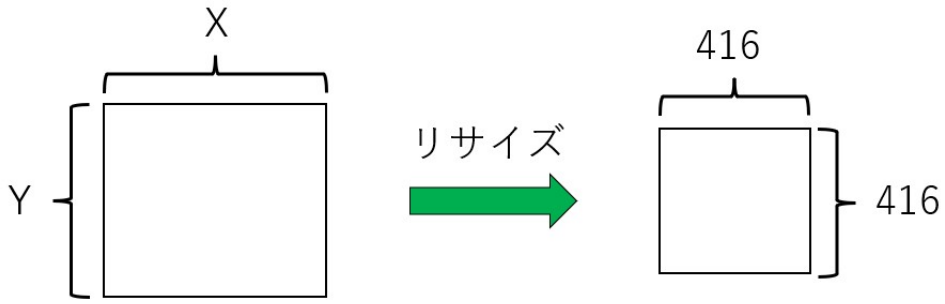

図 2.2: リサイズ

図 2.2 にリサイズの工程を示した．入力画像のサイズを横:X，縦:Y に対してリサイズ後のサイズは 416px の正方形である．

2.1.3 OpenCV

OpenCV とは Open Source Computer Vision Library の略称で，Intel が開発した画像・動画に関する処理機能をまとめたオープンソースのライブラリである．OpenCV の持つ機能 [7] を以下に列挙する．

- 画像の入出力
- 画像の前処理
- 画像内のオブジェクト検出
- 画像への描写処理
- Deep Learning

2.2 FPGA

本章では，FPGA [8] の概要と本研究で扱う FPGA ボードについて述べる．

2.2.1 FPGA 概要

FPGA は Field Programmable Gate Array の略称であり，日本語に直訳すると「現場で構成可能なゲートアレイ」となる．ここで，FPGA 上には論理ゲートやトランジスタなどのハードウェアの構成に必要なパーツが搭載されている．それらのパーツを必要に応じて配

線で繋ぎ合わせることで専用ハードウェアを構成することができる．このような専用ハードウェアの設計手法を「ゲートアレイ」と呼ぶ．また，FPGA の他にも専用ハードウェアを設計することのできるデバイスとして ASIC(Application Specific Integrated Circuit) がある．しかしこちらの ASIC では，一度製造してしまったハードウェアは，製造後にその構成を変更することができない．一方で FPGA は「現場で構成可能」という表現からもわかる通り，何度でもその構成を組み直すことが可能である．以上より，FPGA は，日々最適化されていくシステムの中で，そのハードウェア構成を常に更新し続けられる点から高い柔軟性を持っている．

2.2.2 Ultra96v2

本研究で扱う FPGA ボードであり Avnet 社から提供されている．Ultra96v2 に搭載されている Zynq UltraScale+ MPSoC [9] は，プロセッサと FPGA を 1 チップに搭載しており，高性能かつ大容量の PL(Programmable Logic) を有している．そのため，大量のデータを高速で処理する必要のある画像処理に用いられる．

2.3 ハードウェアアクセラレーション

ここでは，ハードウェアアクセラレーション [10] の概要を示す．CPU(PS:Processing System) は複雑な制御に優れているが，膨大なデータ処理を必要とするアプリケーションには最適とは限らない．こうした課題を抱えているアプリケーションに対して，アプリケーション内の大量の演算処理が必要な部分を切り離し，その処理のために構成した専用ハードウェア (PL:Programmable Logic) で処理を行うことで高速化を図る手法のことをハードウェアアクセラレーションと呼ぶ．

本研究では，物体検出における前処理に対してハードウェアアクセラレーションをすることで，処理の高速化を図る．

第3章　エッジAIプラットフォーム開発

本章ではハードウェアアクセラレーションアプリケーション開発を円滑に進めるためのプラットフォームについて述べる．

3.1　Vitis プラットフォーム概要

Vitis プラットフォーム [11] は，FPGA ボード上で動作するハードウェアやソフトウェアの基となるアーキテクチャを定義する．この Vitis プラットフォームは FPGA 上で PL(Programmable Logic) を用いて動作するアプリケーションのために作成する．また，一つの Vitis プラットフォーム上で複数のアプリケーションを作成することが可能である．そのため，FPGA のハードウェア構成や OS の起動イメージを一から作成する必要が無くなる．

本研究では，アプリケーション開発を円滑に進めるために Vitis プラットフォームを作成し，そのプラットフォーム上で画像処理用アプリケーションを作成する．

3.2　Vitis プラットフォーム開発環境とツール

ここでは Vitis プラットフォーム開発のための環境および使用するツールについて述べる．

- Ubuntu18.04.3 LTS ･･･ 開発用 PC の OS は Ubuntu を用いた．開発ツールである Vivado，PetaLinux，Vitis のバージョンに対応させて Ubuntu のバージョンは 18.04 を採用している．
- Vivado-v2020.2 ･･･ Xilinx 社が提供するハードウェア設計ツール．
- PetaLinux-v2020.2 ･･･ Xilinx 社が Yocto Project をベースとして Zynq シリーズ用にカスタマイズした Linux ディストリビューションの構築を行うためのフレームワーク．組み込みシステム向け Linux をカスタマイズ，ビルド，およびデプロイするために必要なモノを全て提供する．本研究で用いる Ultra96v2 ボードに搭載されている Zynq UltraScale+ MPSoC を含める多くのデバイスに対して Linux システムの開発を容易にする．

- Vitis-v2020.2･･･Xilinx 社が提供する Vitis 統合ソフトウェアプラットフォーム．FPGA 開発で用いるプラットフォームおよびアプリケーションの作成が可能．

3.3 Vitis プラットフォーム作成

ここでは，FPGA ボードで実行するアプリケーション開発の基盤となる Vitis プラットフォームの作成手順について述べる．

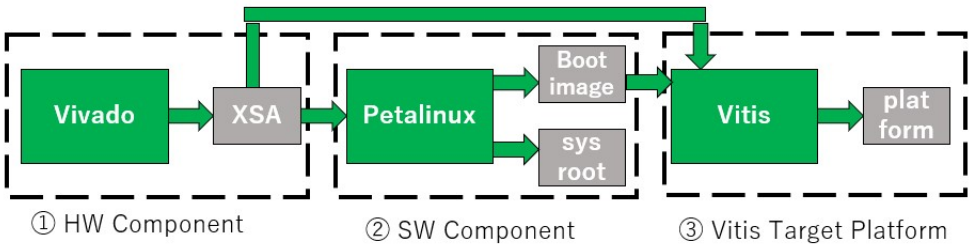

図 3.1: Vitis プラットフォーム作成フロー

図 3.1 はプラットフォーム作成フローを示している．はじめに Vivado を用いてハードウェアの構成を行い，XSA ファイルとして出力する．この工程を HW(Hardware) Component とする．Ultra96v2 上で LinuxOS を起動するためのファイル群を Boot image，ヘッダやライブラリが格納されている root ディレクトリを sysroot と表す．Petalinux を用いてこれらを作成する工程を SW(Software) Component とする．最後に HW Component と SW Component の成果物を Vitis で読み込みまとめることで Vitis プラットフォームを作成する．この工程を Vitis Target Platform とする．ここで示した開発フローを，Xilinx 提供のユーザガイド [12] を参考にして実行する．

3.3.1 HW(Hardware) Component

ここではハードウェアの構成を行う．Vivado を起動し，プロジェクト名を u96v2 とした．

はじめに，BlockDesign [13] と呼ばれる回路の設計を行う．図 3.2 に作成した BlockDesign を示す．

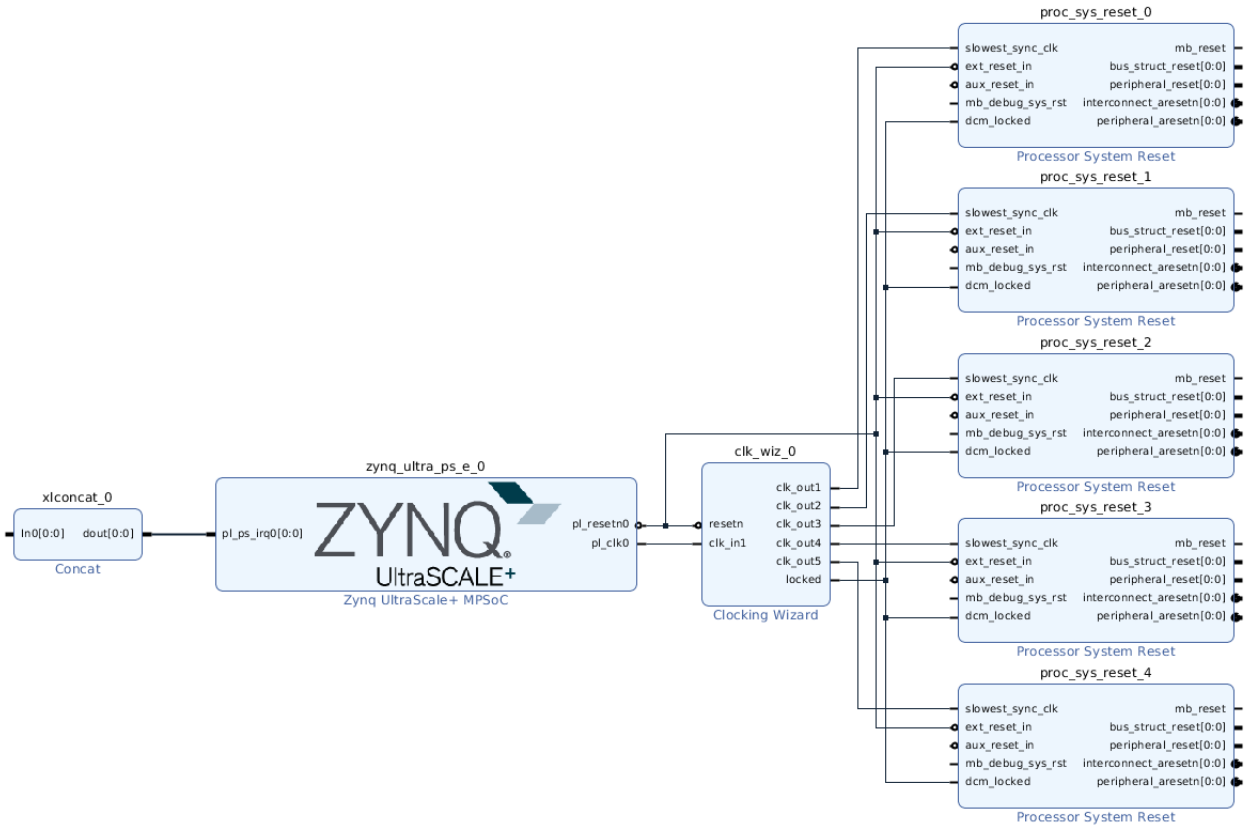

図 3.2: HW BlockDesign

図 3.2 中の配線によって繋がれているブロックの部分を IP(Intellectual Property) と呼び，このブロックは既に設計された回路モジュールを表している．Ultra96v2 ボードには Zynq UltraScale+ MPSoC と呼ばれる IP が搭載されている．この IP は PS(Processing System) と PL(Programmable Logic) の部分に分かれており，他の IP に対してクロックとリセットの供給を行う．作成した BlockDesign では，Zynq UltraScale+ MPSoC の出力クロックと出力リセットを Clking Wizard に供給する．Clocking Wizard は PL 部分にクロックの供給を行い，その出力周波数を指定することができる．Clocking Wizard のクロック出力数は Processor System Reset の数に合わせて 5 つに指定する．また Processor System Reset へのリセット入力は Zynq UltraScale+ MPSoC の出力リセットから供給され，その周波数は 100MHz に指定する．図 3.2 に示した BlockDesign における IP の種類，IP の名称，IP の入出力についてを表 3.1 に示す．

表 3.1: BlockDesign における IP

IP 種類	IP 名称	入力	出力
Zynq UltraScale+ MPSoC	zynq_ultra_ps_e_0	dout[0:0]	pl_resetn0
			pl_clk0
Concat	xlconcat_0		dout[0:0]
Clocking Wizard	clk_wiz_0	pl_resetn0	clk_out1
		pl_clk0	clk_out2
			clk_out3
			clk_out4
			clk_out5
			locked
Processor System Reset	proc_sys_reset_0	clk_out1	
		locked	
		pl_resetn0	
Processor System Reset	proc_sys_reset_1	clk_out2	
		locked	
		pl_resetn0	
Processor System Reset	proc_sys_reset_2	clk_out3	
		locked	
		pl_resetn0	
Processor System Reset	proc_sys_reset_3	clk_out4	
		locked	
		pl_resetn0	
Processor System Reset	proc_sys_reset_4	clk_out5	
		locked	
		pl_resetn0	

FPGA では FPGA 内部のロジックに供給するクロックを指定する必要がある．その機能を有している IP が図 3.2 中の Clocking Wizard であり，実際に出力されるクロックの詳細について表 3.2 に示す.

表 3.2: Clocking Wizard による出力クロック

ポート名	出力周波数 (MHz)	
	要求	実際
clock_out1	100	100
clock_out2	150	150
clock_out3	200	200
clock_out4	300	300
clock_out5	400	400

以上の BlockDesign の作成を終了し，u96v2.xsa というファイル名で保存することでここでの工程は終了する．

3.3.2 SW(Software) Component

この工程では Petalinux を用いて作業を行う．Petalinux の保持しているツールの中で本研究で用いるのは，petalinux-create，petalinux-config，petalinux-build である．SW Component を作成する手順について図 3.3 に示す．

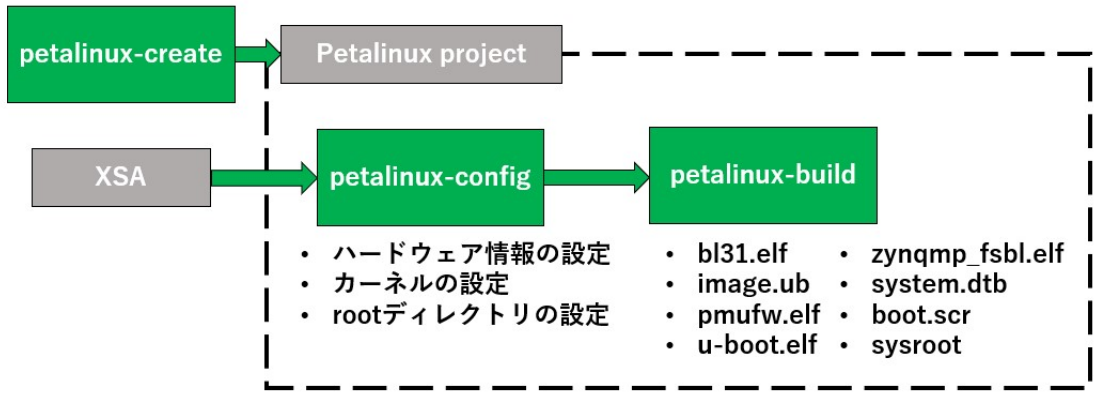

図 3.3: SW Component 作成フロー

はじめに，プロジェクト名を u96v2 として petalinux project の専用ディレクトリを作成する．プロジェクトの作成には petalinux-create ツールと BSP(Board Support Package) を用いる．このパッケージは Avnet 社から提供されており，様々な種類のある FPGA ボードに対して，それぞれのボードの基本的な設定を完了しているパッケージである．petalinux

project 内でなければ Petalinux の保持するツールが利用できないため，以降はこのディレクトリ内で SW Component の工程を行う．

作成したディレクトリに移動し，petalinux-config ツールを用いて u96v2.xsa からハードウェアコンポーネントの情報のアップデート，Linux カーネルの設定，root ディレクトリの設定の順で行う．

はじめに，ハードウェア情報のアップデートにおける作業を述べる．Ultra96v2 を起動するには，起動用データを SD カードに格納し，それを Ultra96v2 に挿入することで起動させる．ここで，root ディレクトリとして SD カードに格納されるファイルは EXT 型である必要があるため，Root filesystem type を EXT に設定する．

次に，Petalinux のベースは Yocto Project であるため，Yocto の設定でデバッグを有効にする．最後に UART の設定を行う．Ultra96v2 はシリアル通信における標準入出力を UART1 で行うため，psu_uart_1 を指定する．以上の変更を保存してハードウェア情報のアップデートは終了する．

次に Linux カーネルの設定について述べる，petalinux project の作成で BSP(Board Support Package) を用いたことから，既にカーネルに対する基本的な設定が完了していた．そのため，ここでのカーネル設定は特に変更せず次の工程へ移る．

加えて，Petalinux プロジェクトの中に user-rootfsconfig というファイルがある．これは root ファイルシステムに含めるパッケージを管理する．ここで，ホストとアクセラレータの通信を容易にするソフトウェアインターフェイスである XRT(Xilinx Runtime Library)，Linux カーネルドライバである zocl，複数の異なる計算用リソースを併用するために用いる OpenCL，画像処理ライブラリである OpenCV の項目を user-rootfsconfig に追加する．ここで追加した zocl ドライバをデバイスツリーに含める必要がある．デバイスツリーは system-user.dtsi というファイルに記述されているため，このファイルを編集して zocl を追加した．

最後に root ディレクトリの設定を行う．ここでは作成したアプリケーションを動かすために必要となるライブラリを有効にする．具体的には gcc ランタイムの C++，Python3，OpenCV，user package を有効にする．以上で Petalinux-config ツールを用いた設定を終了する．

最後に，設定した PetaLinux プロジェクトの内容を参照して Ultra96v2 上で動作する LinuxOS の起動イメージおよび sysroot の生成を行う．この作業は petalinux-build ツールを用いる．ここでは複数のファイルが生成されるが，その中に sdk.sh というファイルがある．こ

れは自己解凍ファイルであり，このファイルを実行し，自己解凍することで sysroot ディレクトリが生成される．ここまで，LinuxOS を起動するためのコンポーネントを作成してきた．これらのコンポーネント構造をまとめるファイルが必要である．そのファイル名を linux.bif として作成する．

以下に SW Component における成果物のうち，後の Vitis Target Platform の作成で用いるファイルおよびディレクトリを列挙する．

- bl31.elf
- image.ub
- pmufw.elf
- u-boot.elf
- zynqmp_fsbl.elf
- system.dtb
- rootfs.cpio
- boot.scr
- linux.bif
- sysroot

新たに boot という名前のディレクトリを作成し，上記の生成物の内，bl31.elf，image.ub，pmufw.elf，u-boot.elf，zynqmp_fsbl.elf，system.dtb，boot.scr を格納してまとめて管理する．

3.3.3 Vitis Target Platform

HW Component と SW Component の工程で得られた成果物と Vitis ツールを用いて Vitis Target Platform の作成を行う．設定項目と設定内容を表 3.3 に示す．

表 3.3: Vitis Target Platform 初期設定

項目	内容
XSA_file	u96v2.xsa
OS	Linux
Processor	psu_cortexa53
Bif file	linux.bif
Boot Component Directory	boot
Linux Image Directory	boot
Linux Rootfs	rootfs.cpio
Sysroot Directory	sysroot/aarch64-xilinx-linux

表3.3 の設定内容について記述する．Vitis を起動後，HW Component で示した XSA ファイルを選択して，Platform Project を作成する．Domein 設定を開き，OS を Linux，Processor を psu_cortexa53 に指定する．加えて Bif ファイルを linux.bif，Boot Components Directory および Linux Image Directory を SW Component の工程でまとめた boot ディレクトリ，Sysroot Directory に sysroots/aarch64-xilinx-linux ディレクトリを指定した．以上の設定後，Vitis ツール上でビルドボタンを押すことで，設定内容をまとめて Vitis Target Platform として出力される．ここで作成したプラットフォームが格納されているディレクトリ名を u96v2_pfm とした．ここで作成した Vitis Target Platform を用いて第 4 章のアプリケーション開発を行う．

3.4 Vitis Vision Library

FPGA 用の画像データ前処理アプリケーションは Vitis というツールで作成するため，Vitis ツール専用のライブラリを用いる必要がある．Vitis を用いた FPGA アプリケーション開発に対して Xilinx 社が提供しているライブラリに Vitis アクセラレーションライブラリがある．このライブラリの一種で，画像処理に対する機能を含んだものが Vitis Vision Library [14] であり，OpenCV の画像処理関数の多くを含んでいる．

第4章　アプリケーションの実装

本章では物体検出における画像前処理アプリケーションの作成について述べる．Ultra96v2 のような FPGA ボードには基本的なシステム全体の制御を行う PS(Processing System) 部分と特定の演算処理などに用いられる PL(Programmable Logic) 部分が存在する．本研究では，PS のみを利用して処理を行うアプリケーションと PS に加えて PL を用いて処理を行うアプリケーションの作成を行う．

4.1　アプリケーション作成 (PS)

実行時に PS(Processing System) のみを用いて画像の前処理を行うアプリケーションの作成について述べる．このアプリケーションのソースコードは Python で記述しており，ファイル名を prepro_PS.py とし以下に示す．

プログラム 4.1: prepro_PS.py

```
1  import glob
2  import sys
3  import time
4  import cv2
5  import numpy as np
6
7  length = 416
8  args = sys.argv
9
10 img = cv2.imread(args[1])
11
12 time_sta = time.perf_counter()
13
14 img_resize = cv2.resize(img, dsize=(length, length))
15
16 time_end = time.perf_counter()
17
18 t = time_end - time_sta
19
20 print(args[1] + " : " + str(t*1000) + "[ms]")
```

prepro_PS.py の概要を以下に述べる．1 行目から 5 行目までは必要なモジュールを利用するための記述である．7 行目の変数 length は入力画像に対してリサイズした後の画像サイズである 416px を定義している．

10 行目では opencv を利用した入力画像の取得を行う．

14 行目の img_resize には入力画像をリサイズした後の配列を格納する．opencv の関数である cv2.resize では，入力画像 img に対して length に指定した 416px のサイズにリサイズ処理を行う．

リサイズにかかる処理時間は変数 t に格納する．リサイズの処理を行う前後で時刻を取得し，その処理時間の計算を行う．1 枚当たりにかかった処理時間 t に対して 1000 を乗算することで出力時間の単位を ms に変更している．

4.2 アプリケーション作成 (PS+PL)

PS(Processing System) に加えて PL(Programmable Logic) を用いるアプリケーションの作成は，Vitis Target Platform と Vitis Vision Library を用いて Vitis ツールで開発する．ここで用いる Vitis Target Platform は第 3 章で作成した u96v2_pfm を用いる．また，Vitis Vision Library は Xilinx 社からリリースされている画像処理ライブラリで，FPGA を用いた画像処理アプリケーションの開発に利用される．github にこのライブラリがリリースされているので，利用する Vitis ツールのバージョンと揃えて Vitis Vision Library2020.2 をダウンロードする．

アプリケーションの作成は開発 PC の端末上で行う．

ダウンロードした Visit Vision Library のディレクトリの中には L1，L2，L3 のディレクトリが存在している．今回のアプリケーション開発では Vitis Library Kernel と呼ばれる L2 を用いる．L2 は Vitis ツールでビルドをするだけでアプリケーションとして実行することができる関数を保持している．この L2 の中には様々な種類の関数が格納されているディレクトリがある．その中から，本研究で目的とする画像処理である resize を選択してそのディレクトリに移動する．このディレクトリの中に makefile が存在しており，make を実行するのだが，ここで環境変数を指定する必要がある．指定した環境変数を表 4.1 に示す．

表 4.1: アプリケーション開発における環境変数の設定

環境変数名	説明	指定パス
DEVICE	利用するプラットフォーム プロジェクトファイル	/u96v2_pfm/u_96v2_pfm.xpfm
K_IMAGE	カーネルイメージ	/u96v2_pfm/sw/u96v2_pfm/linux_domain/image/image.ub
ROOTFS	ファイルシステム	/u96v2_pfm/sw/u96v2_pfm/linux_domain/rootfs/rootfs.cpio

環境変数の設定を完了したため，FPGA 上で実行する画像処理アプリケーション，アプリケーションを動かす上で用いるハードウェアカーネルならびに起動イメージを作成する．

はじめにアプリケーションを動かす上で用いるハードウェアカーネルの作成は以下のコマンドを端末上で実行した．

make host xclbin TARGET=hw HOST_ARCH=aarch64

このコマンドを実行することで，krnl_resize.xclbin というファイルが作成される．

次に，画像処理アプリケーションと起動イメージを作成する．作成には以下のコマンドを実行する．

make run TARGET=hw

このコマンドを実行することで，Ultra96v2 でアプリケーションを実行するのに必要なファイル群を出力してくれる．

以上でアプリケーションファイル，ハードウェアカーネルファイル，起動イメージファイルが生成された．生成されたファイル群を以下に示す．

- BOOT.bin
- bl31.elf
- image.ub
- pmufw.elf
- u-boot.elf
- zynqmp_fsbl.elf
- system.dtb
- emconfig.json
- run_script.sh
- xrt.ini
- resize

- krnl_resize.xclbin

ここで，アプリケーションを開発したことにより，第 3 章で作成した Vitis Target Platform のハードウェア設計が変更されている．Vitis Target Platform におけるハードウェア構成である BlockDesign は図 3.2 で示している．変更後の BlockDesign について図 4.1 に示す．

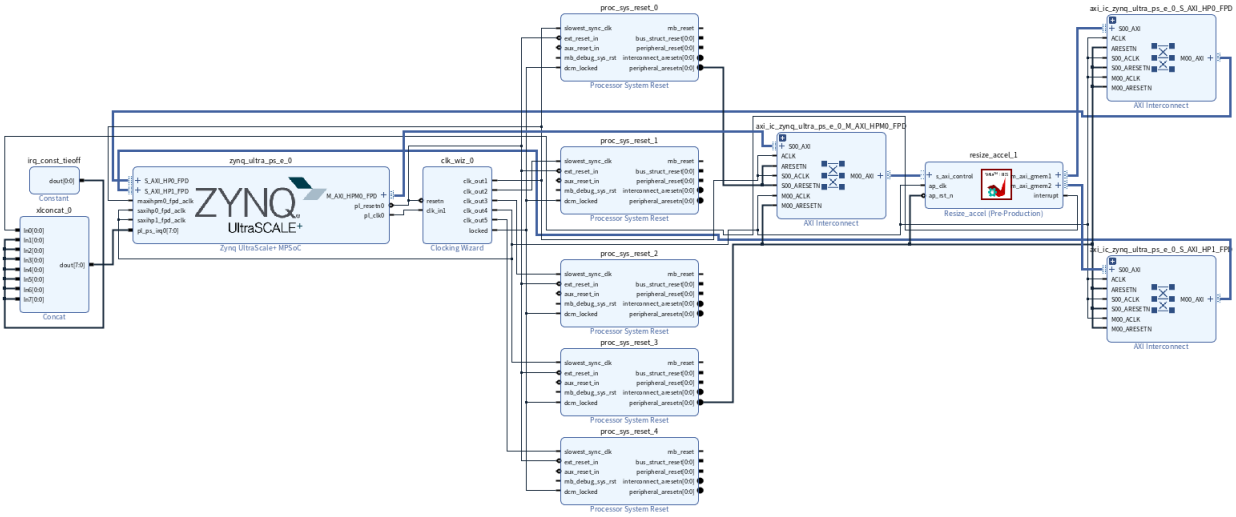

図 4.1: アプリケーション開発後 BlockDesign

図 3.2 と図 4.1 の BlockDesign を比較すると，図の右側に IP(Intellectual Property) が追加されている．IP(Intellectual Property) とは，BlockDesign 中にあるブロックの部分のことを表す設計済みの回路モジュールである．図 4.1 の BlockDesign では，Vitis Vision Library に含まれているリサイズ処理を高速化するための専用 IP(Intellectual Property) が追加されている．

以上でアプリケーションの実装と変更後のハードウェア構成の確認を終了する．

第5章　評価実験

本章ではこれまでに作成してきたアプリケーションの実行するまでの実験準備，実行手順，実験結果について述べる．

5.1　実験準備

ここでは Ultra96v2 を起動する前に行う必要のある準備をデータセット，SD_Card の作成の順に述べる．

5.1.1　データセット

実験に用いるサンプル画像は，Vitis-AI の Ultra96v2 上動作に関する記事 [15] に記載されているマスクを着用している人々の画像データセットである Mask Dataset [16] を利用した．このデータセットの内，50 枚の画像に対して前処理プログラムを実行する．サンプル画像の例を図 5.1 に示す．

図 5.1: Mask Dataset Sample Picture

5.1.2 SD_Card の作成

Ultra96v2 は OS の起動イメージと root ディレクトリを一つの SD カード内の異なるパーティションに格納する必要がある．そのため，MicroSD カードを 2 つのパーティションに分割する．分割の概要について表 5.1 に示す．

表 5.1: MicroSD のパーティション分割

項目	第 1 パーティション	第 2 パーティション
名称	boot	rootfs
ファイルシステム	FAT32	ext4
サイズ	4GB	28GB

表 5.1 のように，第 1 パーティションの名称を boot，ファイルシステムは FAT32，サイズは 4GB に指定した．続いて，第 2 パーティションの名称は rootfs，ファイルシステムは ext4，サイズは 28GB に指定した．名称はそれぞれの格納するファイルを表している．また，Ultra96v2 には Zynq UltraScale+ MPSoC と呼ばれるデバイスが搭載されている．このデバイスは FAT ファイルシステムにある起動用ファイルを呼び出すことで起動する．サイズは，格納するファイルのサイズに合わせて指定してある．次に，作成したデータの書き込みについて図 5.2 に示す．

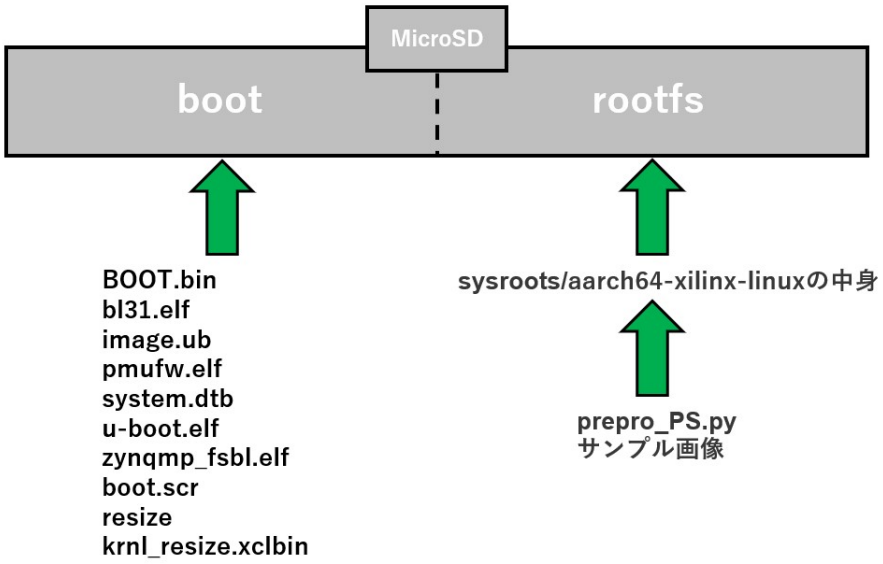

図 5.2: SD カードへのデータ格納

第 1 パーティションである boot には図 5.2 に示している 10 種類のファイルを書き込む．

第 2 パーティションである rootfs には Linux システムの root ファイルを書き込む．同時に，作成した Ultra96v2 上のデバイスのうち，PS(Processing System) のみで処理を行う画像処理アプリケーションと実験に用いるサンプル画像のデータセットもこの第 2 パーティションに書き込む．

5.1.3 実験環境の構築

ここでは Ultra96v2 を起動し，これまで作成してきたアプリケーションを実行する前にやっておく必要のある環境構築について述べる．

必要なファイルの書き込みが完了したので，MicroSD カードを Ultra96v2 に挿入し，PC とシリアル通信を行う．シリアル通信用ツールである gtkterm を起動し，設定画面を開き，ポートを/dev/ttyUSB1 に，ボーレートを 115200 に指定した．Ultra96v2 の起動ボタンを押すと，Linux カーネルが起動する．最初に，Ultra96v2 の Wi-Fi 設定を行う．この設定の完了後，PS(Processing System) のみを用いて画像処理を行うアプリケーションのために，Python 用の OpenCV をダウンロードする．Python3 で cv2 をインポートできることを確認した．また，PS(Processing System) と PL(Programmable Logic) の両方を用いて処理を行うアプリケーションでは，XRT(Xilinx Runtime Library) と呼ばれるホストとアクセラレータの通信を行うソフトウェアインターフェイスを用いるためこのライブラリもダウンロードする．

5.2 アプリケーションの実行

ここでは作成したアプリケーションを実際に Ultra96v2 上で実行した手順について述べる．

5.2.1 アプリケーションの実行 (PS)

作成した PS(Processing System) のみを用いて画像処理を行うアプリケーションを実行する．画像処理アプリケーションである prepro_PS.py とサンプル画像が格納されている mask_before ディレクトリのあるディレクトリで以下のコマンドを実行した．

python3 prepro_PS.py ./pict/Mask_1.jpg

このコマンドの第 1 引数の画像の名称を変更して 50 枚の画像に対して実行した．

5.2.2 アプリケーションの実行 (PS+PL)

PS(Processing System) と PL(Programmable Logic) を用いて画像処理をするアプリケーションの実行ファイルは，resize ファイルである．このファイルがあるディレクトリに移動する．ここで，環境変数を指定する．指定した環境変数を表 5.2 に示す．

表 5.2: アプリケーション実行における環境変数の設定

環境変数名	指定パス
LD_LIBRARY_PATH	/mnt:/tmp::/opt/xilinx/xrt/lib::/opt/xilinx/xrt/lib
XILINX_VITIS	/mnt
XILINX_XRT	/usr

実行ファイル名は resize であり，第 1 引数に入力画像，第 2 引数にリサイズ後の高さ，第 3 引数にリサイズ後の幅を取る．実行コマンドを以下に示す．

./resize /home/pict/Mask_1.jpg 416 416

サンプル画像は/home/pict ディレクトリに存在するため，第 1 引数は/home/pict/画像ファイル名となる．また，リサイズ後のサイズは 416px の正方形のため，第 2 引数と第 3 引数には 416 を指定している．

この実行コマンドを 50 枚のサンプル画像に対して実行する．

5.3 実験結果

ここでは実行した画像前処理アプリケーションの実行結果を表 5.3 に示す.

表 5.3: 画像処理実行結果

	処理時間 (ms)		画像サイズ (px)			処理時間 (ms)		画像サイズ (px)	
No	PS	PS+PL	Width	Height	No	PS	PS+PL	Width	Height
1	13.546	9.619	2000	1363	26	11.240	2.481	962	642
2	13.338	8.755	1920	1277	27	11.199	2.868	1200	600
3	11.493	2.756	1024	682	28	12.207	2.138	580	870
4	11.847	3.332	1240	698	29	12.224	2.098	580	870
5	13.489	9.784	2000	1388	30	12.365	2.160	580	871
6	9.619	1.256	620	434	31	10.738	2.418	1024	576
7	11.330	2.965	1200	630	32	12.797	6.405	1080	1620
8	9.080	1.219	610	350	33	12.992	7.092	1500	1319
9	13.237	5.269	1600	900	34	13.209	5.464	1500	1000
10	10.250	1.949	918	506	35	10.952	2.446	1100	550
11	10.781	2.389	1024	576	36	12.827	4.052	1200	900
12	11.419	2.704	1000	667	37	13.087	5.996	1535	1080
13	11.435	2.468	940	650	38	12.883	4.083	1200	900
14	15.076	49.505	4575	3021	39	12.732	3.537	1100	825
15	12.102	3.5171	1280	720	40	12.733	3.913	1199	868
16	14.346	18.234	3071	1727	41	10.672	2.108	950	534
17	12.296	2.072	552	886	42	9.903	1.704	800	480
18	12.437	2.098	552	901	43	10.644	2.399	1020	573
19	10.861	2.388	1024	576	44	12.618	3.495	1100	825
20	11.402	2.979	1200	630	45	12.709	3.676	1200	800
21	10.790	2.380	1024	576	46	13.168	6.118	1500	1125
22	12.377	3.203	1050	788	47	11.513	2.655	1000	667
23	10.477	2.367	1024	576	48	12.924	4.053	1200	900
24	10.900	1.992	800	600	49	12.636	3.648	1200	800
25	11.241	2.481	962	641	50	11.392	3.210	1200	640

表 5.3 では，画像処理を行う対象となる画像の番号を No で表しており，50 枚の画像に対して画像処理を行ったので No は 50 まで示している．次に，処理時間の項目は，第 4 章で作成した PS(Processing System) のみを用いて画像処理を行うアプリケーションと PS(Processing System) と PL(Programmable Logic) を用いて画像処理を行うアプリケーションを実行したときの処理時間を示したおり，処理時間の単位は ms としている．最後に画像サイズの項目は，画像処理対象である入力画像の横 (Width) と縦 (Height) のサイズを px 単位で示している．

ここでは，PS(Processing System) のみを用いて画像処理を行うアプリケーションを PS アプリケーション，PS(Processing System) と PL(Programmable Logic) を用いて画像処理を行うアプリケーションを PS+PL アプリケーションと呼ぶ．

表 5.3 の結果から，大半のリサイズ処理がハードウェアアクセラレーションにより高速化できている．ただ，No14 と No16 の画像に対するリサイズ処理では，PS+PL アプリケーションよりも PS アプリケーションの方が処理時間が短いことがわかる．この 2 枚の画像は，他の画像と比較すると明らかに画像サイズが大きい．このことから，入力画像のサイズとリサイズ後のサイズに大きな差があると今回用いたハードウェアアクセラレーションの方法では逆に処理時間が大きくなってしまうと推測する．

次に 3 つの場合に分けて実験結果を評価していく．

(a) 入力画像の高さと幅の両方がリサイズ後の画像サイズと大きく離れている場合，先にも述べた通りアクセラレーションによる高速化は見られない．

(b) 入力画像の高さと幅のどちらか一方がリサイズ後の画像サイズと近い場合，PS アプリケーションに比べ PS+PL アプリケーションの方が処理時間が大幅に小さくなっているためハードウェアアクセラレーションの効果が見られる．具体的には，No7，No15，No19，No21，No27 の画像に見られる．

(c) 入力画像の高さと幅の両方がリサイズ後の画像サイズと近い場合，こちらでもハードウェアアクセラレーションによる高速化の効果が見られる．具体的には，No6，No8，No42 の画像に見られる．処理時間だけを見ると，(b) の場合よりも短くなっている．しかし，PS アプリケーションと PS+PL アプリケーションの処理時間を比率で見た場合は，(b) の場合も (c) の場合も変わらない．

第6章　考察

本章では，ハードウェアアクセラレーションによる画像処理の高速化に対する考察と，本研究の今後の展望についての考察を述べる．

まず，ハードウェアアクセラレーションによる画像処理の高速化に対する考察を述べる．入力画像に対する画像処理のハードウェアアクセラレーションは物体検出の課題であるリアルタイム性の向上に対して効果的である．しかし，必ずしもそうである訳ではない．本研究における実験結果の中で，入力画像のサイズとリサイズ後の目標画像サイズに大きな差がある場合は，ハードウェアアクセラレーションによる恩恵は受けられないことが分かったからである．そのため，実際にエッジデバイス上で物体検出を行う際には，カメラから得られる入力画像のサイズと，推論モデルの受け付ける画像サイズに大きな差がない場合は，ハードウェアアクセラレーションを用いることが有効である．

次に，本研究の今後の展望について考察を述べる．本研究では，物体検出における前処理の部分の中でも，特にリサイズの処理に対して焦点を当てた．しかし実際の前処理では，リサイズだけの処理をするわけではなく，正規化などの他の処理も行う，この前処理は推論モデルによってまちまちである．そのため，リサイズ以外の前処理についてもハードウェア実装をすることで高速化できると考える．また，エッジコンピューティングにおける速度以外の課題であるリソースの使用率や消費電力の項目についてもハードウェア化することでどのような変化をもたらすか検証する必要がある．加えて，ハードウェア実装した前処理を，物体検出に組み込み，前処理から推論処理までの物体検出全体の処理速度にどの程度影響を及ぼすのかも検証する必要がある．

第7章　あとがき

本研究では，物体検出におけるリアルタイム性を課題として取り上げ，入力画像に対する前処理を高速化することを目的とした．この目的を成し遂げるため，前処理の部分をハードウェア実装することを目標とした．本研究を進めるにあたって，はじめに開発ツールである Vivado，Petalinux，Vitis の使用方法について学習を行った．その後，プラットフォームの作成，アプリケーションの作成，FPGA ボードでの実験，性能評価の順で実行した．

実験結果として，Vitis Vision Library を用いたハードウェアアクセラレーションで画像処理を高速化できることがわかった．しかし，一概にこのハードウェアアクセラレーションが効果的であるとは言えない．なぜなら，入力画像のサイズとリサイズ後の目標サイズに大きな差がある場合は逆に処理時間が大きくなってしまったからである．ただ，画像サイズさえ注意すれば，画像処理に要する処理時間を大幅に短縮できるため，今後の物体検出に活用したい．

本研究でハードウェア実装できた物体検出における画像処理はリサイズ処理のみであり，さらに高速化を図るためには，その他の前処理についてもハードウェア実装する必要がある．加えてエッジデバイスにおけるリアルタイム性以外の課題であるリソース使用率や消費電力については検証できていないのため，これらの項目についても検証する必要がある．

以上を今後の課題とし，解決することでエッジデバイスにおける物体検出のリアルタイム性向上につながると考える．

謝辞

本研究，論文作成を進めるにあたり，御懇篤な御指導，御鞭撻を賜りました本学高橋寛教授ならびに甲斐博准教授，王森レイ講師に深く御礼申し上げます.

また，審査頂いた本学甲斐博准教授，遠藤慶一准教授ならびに宇戸寿幸准教授に深く御礼申し上げます.

最後に，多大な御協力と貴重な御助言を頂いた計算機/ソフトウェアシステム研究室の諸氏に厚く御礼申し上げます.

参考文献

[1] A. Wheeldon, R. Shafik, T. Rahman, J. Lei, A. Yakovlev, and O.-C. Granmo,
”Learning automata based energy-efficient AI hardware design for IoT applications,”
Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.,
vol. 378, no. 2182, p. 20190593, Oct. 2020. J. Misra and I. Saha,
“ Artificial neural networks in hardware: A survey of two decades of progress, ”
Neurocomputing, vol. 74, no. 1, pp. 239-255, 2010.

[2] 小西祥之, 国宗大介, 西田芳隆, ミラクシアエッジテクノロジー株式会社,
”物体検出におけるエッジ環境での推論高速化と精度上昇”,
2021,
https://www.jstage.jst.go.jp/article/pjsai/JSAI2021/0/JSAI2021_
3I4GS7a02/_pdf/-char/ja,
（参照 2023-01-19）

[3] 橋本恭佑, 伊藤雅博, 株式会社 日立製作所,
”機械学習における画像データ前処理の性能検証（後編）”,
2020-05-29,
https://qiita.com/kyosuke_hashimoto/items/90d7cd5dce0402da55ca,
（参照 2023-02-08）

[4] K. Guo, S. Zeng, J. Yu, Y. Wang, H. Yang,
“ A Survey of FPGA-based Neural Network Inference Accelerators, ”
J ACM Trans. Reconfigurable Technol. Syst.,
Vol. 12, No. 1, Article No.: 1, pp. 1-26, Apr. 2019.

[5] MACNICA,
”物体検出と Deep　Learning　入門から応用まで ”,

https://www.macnica.co.jp/business/ai/blog/142041/,
（参照 2023-02-08）

[6] ∞ ReNom,
”オブジェクト検出 YOLO”,
https://www.renom.jp/ja/notebooks/tutorial/image_processing/yolo/notebook.html,
（参照 2023-02-08）

[7] ケイエイブル株式会社,
”初めての Open CV（画像処理ライブラリ）ガイド”
https://www.klv.co.jp/corner/python-opencv-what-is-opencv.html,
（参照 2023-01-19）

[8] K. Guo, S. Zeng, J. Yu, Y. Wang, H. Yang,
”A Survey of FPGA-based Neural Network Inference Accelerators,”
J ACM Trans. Reconfigurable Technol. Syst.,
Vol. 12, No. 1, Article No.: 1, pp. 1-26, Apr. 2019.

[9] AVNET,
”Ultra96 開発ボードで Xilinx Zynq UltraScale+ MPSoC を手軽に評価”,
https://www.avnet.com/wps/portal/japan/products/product-highlights/ultra96/,
（参照 2023-01-19）

[10] Xilinx,
”アクセラレーションコンピューティングとは何か．そしてなぜ必要なのか”,
https://japan.xilinx.com/applications/adaptive-computing/what-is-accelerated-computing-and-why-is-it-important.html,
（参照 2023-01-29）

[11] Xilinx,
”Vitis 統合ソフトウェアプラットフォーム，プラットフォームベースフロー”,
https://japan.xilinx.com/products/design-tools/vitis/vitis-platform.

html,
（参照 2023-01-19）

[12] Xilinx,
”Vitis 統合ソフトウェアプラットフォームの資料：エンベデッドソフトウェア開発”,
2020-06-24,
https://docs.xilinx.com/r/2020.1-日本語/ug1400-vitis-embedded,
（参照 2023-01-18）

[13] Xilinx,
”Vivado Design Suite ユーザー ガイド: IP インテグレーターを使用した IP サブシステムの設計”
2020-01-04,
https://docs.xilinx.com/r/2020.2-%E6%97%A5%E6%9C%AC%E8%AA%9E/ug994-vivado-ip-subsystems/%E6%94%B9%E8%A8%82%E5%B1%A5%E6%AD%B4,
（参照 2023-02-08）

[14] Xilinx,
”Vitis　ビジョン ライブラリ”,
urlhttps://japan.xilinx.com/products/design-tools/vitis/vitis-libraries/vitis-vision.html,
（参照 2023-02-08）

[15] Qiita,
”Vitis-AI v1.4 on Ultra96v2”,
2021-11-06,
urlhttps://qiita.com/lp6m/items/4117a3bab185afedfd5f,
（参照 2023-01-19）

[16] Google Drive,
”Mask_Dataset”,
2020-02-26,
urlhttps://drive.google.com/drive/folders/1aAXDTl5kMPKAHE08WKGP2PifIdc21-ZG,
（参照 2023-01-19）