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

指導教員

高橋 寛 教授

甲斐 博 准教授

王森レイ 講師

令和 5 年 1 月 20 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

西川　竜矢

目 次

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．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．

また，物体検出に関する研究では，推論処理における高速化が盛んである．その高速化手法 [1] 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．しかし，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の工程があり，この工程における高速化の研究は少ない．実際に物体検出の前処理と推論処理では，推論処理に要する処理時間の方が前処理に要する処理時間より大きくなるのは周知の事実ではあるが，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．こうした点から本研究では推論処理の前処理の工程に対して高速化を進めていく．

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）は集積回路の一種であり，様々な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，こうした特徴から，FPGA は CPU と比較しても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．以上の点から，リアルタイム性を重要視しているエッジデバイスでの画像処理に対して FPGA を用いることは，先に述べた課題に対する有効的な解決策となるといえる．

1.2 論文の構成

※全体が完成したら書きます.

第2章　予備知識

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記述する．

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．

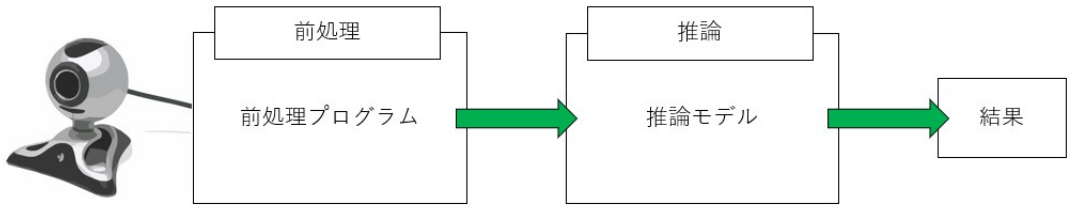

図 2.1: 物体検出の流れ

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 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．

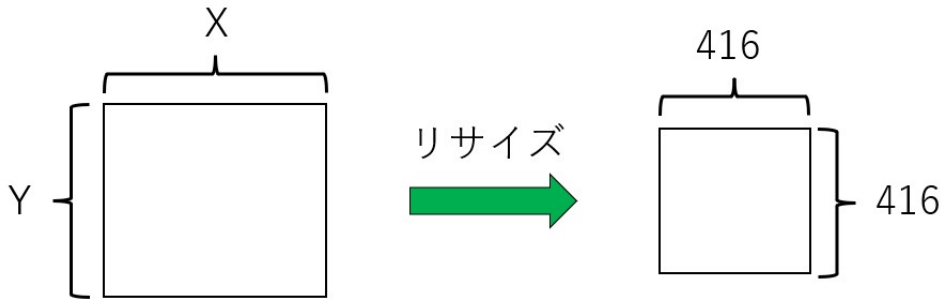

図 2.2: リサイズ

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 [2] 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

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．ここで言う「ゲートアレイ」とは，ASIC（Application Specific Integrated Circuit）の設計・製造手法のひとつである．この手法では，ウェハー上に標準 NAND ゲートや NOR ゲート等の論理回路，単体のトランジスタ，抵抗器などの受動素子といった部品を決まった形で配置し，その上に配線を加えることで各部品を配線し半

導体回路を完成させる．昨今では専用 LSI である ASIC の開発に数千万円から数億円の初期コストがかかることや，一度製造してしまった LSI の構成は製造後には変更できないなどの問題がある．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，ユーザの手元でロジックや配線を変更できるというコンセプトを基に開発される．現在開発されている FPGA の規模は様々だが，一昔前の専用 LSI を超える規模の回路を簡単に構成できるようになってきている．また，FPGA と ASIC などの専用 LSI との大きな違いとして「製造のための初期コスト」が不要なことも挙げられる．

2.2.2 Ultra96v2

本研究で扱う FPGA ボード．Ultra96v2 に搭載されている Zynq UltraScale+ MPSoC [3] は，プロセッサと FPGA を 1 チップに搭載しており，機能を以下に述べる．

- 高性能かつ大容量プログラムロジックを利用した高帯域な信号・画像処理
- Cortex-A53 アプリケーションプロセッサで余裕のあるシステム（OS/GUI）プロセス処理実行
- Cortex-R5 でタイミングクリティカルなリアルタイム処理をプログラムロジックと連携処理

以上の機能を持つことからも，自動車，医療，製造業，放送，通信などの幅広い分野で利用されている．

2.3 ハードウェアアクセラレーション

通常，CPU(PS:Processing System) によって処理されるプログラムが，処理速度や消費電力など課題を抱えている場合に，こうした処理のための専用ハードウェア (PL:Programmable Logic) を構成することで処理を援助し，高速化を図る手法のこと．

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

3.1　Vitis プラットフォーム概要

Vitis プラットフォームは，外部メモリインターフェイス，カスタム入出力インターフェイス，ソフトウェアランタイムなど，AMD ザイリンクスプラットフォームのハードウェアやソフトウェアのベースアーキテクチャおよびアプリケーションコンテキストを定義する．

Vitis プラットフォーム [4] を使用する設計書法は様々なメリットをもたらし，生産性の向上につながる．特徴を以下に列挙する．

- プラットフォームの再利用性：同じプラットフォーム上で異なるアクセラレーションアプリケーションを入れ替え可能
- アプリケーションの移植性：最小限の操作で異なるプラットフォーム間でアプリケーションを移植可能
- シミュレーション時間：カーネルを使用する協調シミュレーションの高速化
- ランタイム：オープンソースのランタイムを使用して PCIe® インターフェイスまたはエンベデッドインターフェイス経由でホストとデバイス間の通信を制御
- システムデバッグ：システム全体の協調シミュレーションによって，全ハードウェアコンパイル時間を短縮

3.2　Vitis プラットフォーム開発環境とツール

- Ubuntu18.04.3 LTS ･･･ LinuxOS の 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-v2020.1 ･･･ Xilinx が提供するハードウェア設計ツール．HDL からビットストリームの生成，FPGA への書き込みまでの開発における下位の部分を担当する．

- PetaLinux-v2020.1 ··· Xilinx が提供するツール．Xilinx のプロセッシングシステム上で組み込み Linux のソリューションをカスタマイズ，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．設計生産性の加速を目的とするこのソリューションは，Xilinx のハードウェア設計ツールと連動し，Versal，Zynq UltraScale+ MPSoP，Zynq-7000，および MicroBlaze 向けの Linux システムの開発を容易にする．
- Vitis-v2020.1 ··· Xilinx が提供する Vitis 統合ソフトウェアプラットフォーム．ツール内容を以下に述べる．
  - アクセラレーションアプリケーションをシームレスに構築するための包括的なコア開発キット
  - AMD ザイリンクスの FPGA および Versa I ACAP ハードウェアプラットフォーム向けに最適化された，ハードウェアアクセラレーション用の豊富なオープンソースライブラリ
  - 使い慣れた高レベルのフレームワークを利用して直接開発できる，プラグインタイプのドメイン特化開発環境
  - 今後さらに拡大する，ハードウェアアクセラレーションパートナー提供のライブラリおよび構築済みアプリケーションのエコシステム

3.3 Vitis プラットフォーム作成

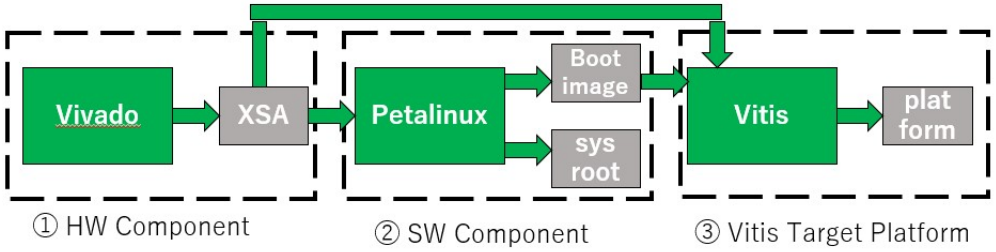

図 3.1: プラットフォーム作成フロー

プラットフォームの作成は図 3.1 プラットフォーム作成フローに従って行い，以下に示す工程は Xilinx 提供のユーザガイド [5] を参考にして実行する．また，Ultra96v2 向けプラッ

トフォームが avnet 社より提供されているので，このプラットフォームを用いても良いのだが，カーネル作成時にライブラリの追加などを行う必要があるため，avnet 社が提供しているプラットフォームの内，ハードウェア構成が構築されている XSA ファイルのみを引用する．そのため，1.HW Component の工程は省略する．図 3.2 に引用する XSA ファイルの BlockDesign を示す．

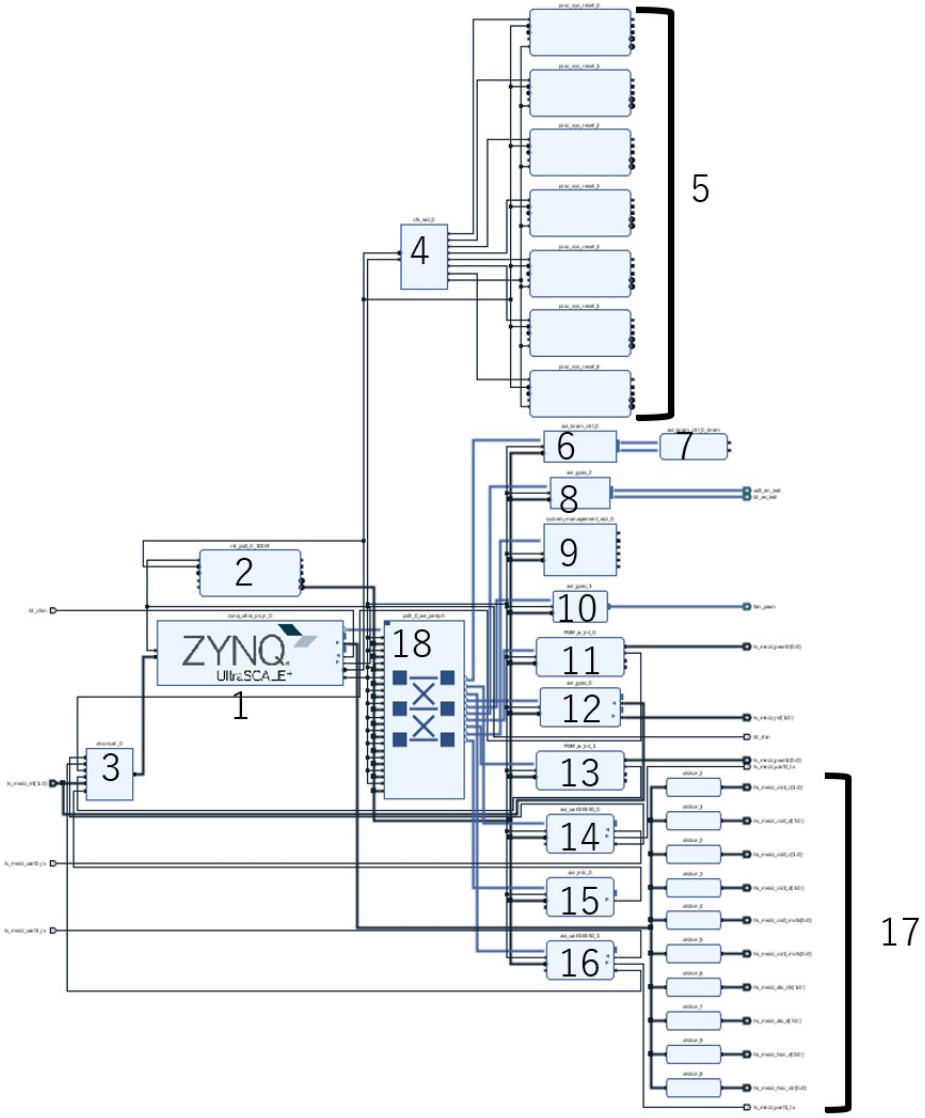

図 3.2: HW BlockDesign

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，FPGA 業界では，既に設計された設計資産，の意味で使用される．回路設計を行うユーザの視点からすると，既に設計された回路ライブラリを表す．こうした IP を利用することにより，容易な回路設計を実現している．図 3.2 に示した BlockDesign の各 IP を表 3.1 に列

挙する．

表 3.1: BlockDesign における IP

番号	IP 種類	IP 名称	出力端子
1	Zynq UltraScale++ MPSoC	zynq_ultra_ps_e_0	
2	Processor System Reset	rst_ps8_0_100M	
3	Concat	xlconcat_0	
4	Clocking Wizard	clk_wiz_0	
5	Processor System Reset	proc_sys_reset_0	
	Processor System Reset	proc_sys_reset_1	
	Processor System Reset	proc_sys_reset_2	
	Processor System Reset	proc_sys_reset_3	
	Processor System Reset	proc_sys_reset_4	
	Processor System Reset	proc_sys_reset_5	
	Processor System Reset	proc_sys_reset_6	
6	AXI BRAM Controller	axi_bram_ctrl_0	
7	Block Memory Generator	axi_bram_ctrl_0_bram	
8	AXI GPIO	axi_gpio_2	wifi_en_led, bt_en_led
9	System Management Wizard	system_management_wiz_0	
10	AXI GPIO	axi_gpio_1	fan_pwm
11		PWM_w_lnt_0	ls_mezz_pwm0[0:0]
12	AXI GPIO	axi_gpio_0	ls_mezz_rst[1:0]
13		PWM_w_lnt_1	ls_mezz_pwm1[0:0]
14	AXI UART166550	axi_uart166550_0	ls_mezz_uart0_tx
15	AXI Interrupt Controller	axi_intc_0	
16	AXI UART166550	axi_uart166550_1	ls_mezz_uart0_tx
17	Slice	xlslice_0	hs_mezz_csi0_c[1:0]
	Slice	xlslice_1	hs_mezz_csi0_d[7:0]
	Slice	xlslice_2	hs_mezz_csi1_c[1:0]
	Slice	xlslice_3	hs_mezz_csi1_d[3:0]
	Slice	xlslice_4	hs_mezz_csi0_c[1:0]
	Slice	xlslice_5	hs_mezz_csi1_d[3:0]
	Slice	xlslice_6	hs_mezz_dsi_mclk[0:0]
	Slice	xlslice_7	hs_mezz_dsi_clk[1:0]
	Slice	xlslice_8	hs_mezz_dsi_d[7:0]
	Slice	xlslice_9	hs_mezz_hsic_d[0:0]

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 4 番目の IP である 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	150	150
clock_out2	300	300
clock_out3	75	75
clock_out4	100	100
clock_out5	200	200
clock_out6	400	400
clock_out7	600	600

2.SW Component の工程に移る．この工程では Petalinux ツールを用いて作業を行う．はじめに，petalinux project を作成する．ここでは，後のカーネルおよび rootfs の設定を省略するため avnet 社の提供している Ultra96v2 専用の Board Support Package ファイルを用いてプロジェクトを立ち上げる．次に 1.HW Component で示した XSA ファイルからハードウェアコンポーネントの情報を取り込む．最後に rootfs の設定として，利用する user package，opencv，python のライブラリを有効にする．以上の設定を経て，PetaLinux プロジェクトのビルドを行う．ビルドを行うことで，設定した Linux カーネル並びに sdk.sh の生成を行う．生成した sdk.sh を自己解凍することで sysroot が生成される．以下に 2.SW Component における成果物のうち，後の Vitis Target Platform の作成と Vitis application 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
- fsbl.elf
- system.dtb
- linux.bif
- rootfs.ex4

- sysroot

上記の生成物の内, bl31.elf, image.ub, pmufw.elf, u-boot.elf, fsbl.elf, system.dtb, linux.bif を boot ディレクトリにまとめて管理する.

1.HW Component と 2.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. Vitis を起動後, 1.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_cortex53 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 boot ディレクトリ, rootfs file を rootfs.ex4, Sysroot Directory に sysroots/aarch64-xilinx-linux ディレクトリを指定した. 最後に Board Support Package の設定を行い, プラットフォームをビルドすることでプラットフォーム作成を完了する.

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. そのライブラリが, Xilinx から提供されている Vitis Vision Library であり, OpenCV の画像処理関数の多くを含んでいる. Vitis Vision Library の特徴を以下に列挙する.

- 色およびビット深度変換, ピクセルごとの算術演算, 幾何変換, 統計, フィルター, 特徴検出, 分類, 3D 再構成など性能に最適化された機能の提供
- カラー画像処理のネイティブサポート, マルチチャネルストリーミングのサポート
- オンチップメモリまたは外部メモリ間のデータ移動を効率的に管理して最大限の性能を達成
- ビジョンパイプラインに求められる演算能力をすばやく評価して最適なデバイスを選択
- ビジョンおよび画像処理アルゴリズムを高速化する方法を示すサンプルデザインを提供
- 関数パラメータを活用し, 1 クロックサイクルで複数ピクセルを処理してスループット要件を満たす

第4章　アプリケーションの実装

本研究では物体検出における画像前処理アプリケーションを作成する．PS のみを利用して処理を行うアプリケーションと PS と 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 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  re_length = 416
8  total = 0
9
10 for i in range(600):
11
12     img = cv2.imread("./mask_before/Mask_" + str(i+1) + ".jpg")
13
14     time_sta = time.time()
15
16     h, w = img.shape[:2]
17
18     re_h = re_w = re_length/max(h,w)
19
20     img_resize = cv2.resize(img, dsize=None, fx=re_h, fy=re_w)
21
22     time_end = time.time()
23     t = time_end - time_sta
24     total = total + t
25
```

```
26    print("time:" + str(total) + "[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 re_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．ここでは入力画像の長辺を 416px にリサイズすることを想定しており，リサイズ後も長辺と短辺の大きさの比率は変わらない．10 行目から 20 行目までの画像処理の部分の説明をする．前処理を行うサンプル画像の枚数は 600 枚で行ったので 10 行目の range の中は 600 に指定している．12 行目では opencv を利用した入力画像の取得を行っており，続けて 16 行目で入力画像の縦のサイズ:h，横のサイズ:w を取得している．18 行目では入力画像を変換する倍率の計算を行っており，リサイズ後の長辺サイズである 416px を入力画像の縦，横のピクセルサイズのうち大きい方で除算することにより倍率を計算している．20 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 に対して入力画像 img と 18 行目で計算した倍率である re_h と re_w を引数として渡すことでリサイズ処理を行う．

リサイズにかかる処理時間は，変数 total 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を変数 total に加算していくことで，最終的に 600 枚当たりの処理時間の計測を行っている．

4.2 アプリケーション作成 (PS+PL)

PS と PL を用いるアプリケーションの作成は，3 章で作成したプラットフォームを利用して行う．Vitis を起動し，プラットフォームの選択を行い，その後サンプルから Vitis Vision Library の resize を選択する．作成したソースコードを以下に示す．

※完成したら書きます

アプリケーションの作成が完了すると新たに SD_Card.img が生成される．

第5章　評価実験

5.1　実験準備

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 [6] 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 [7] を利用した．このデータセットの内，600 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 で扱う MicroSD カードに，作成したデータを書き込む方法について図 5.2 に示す．

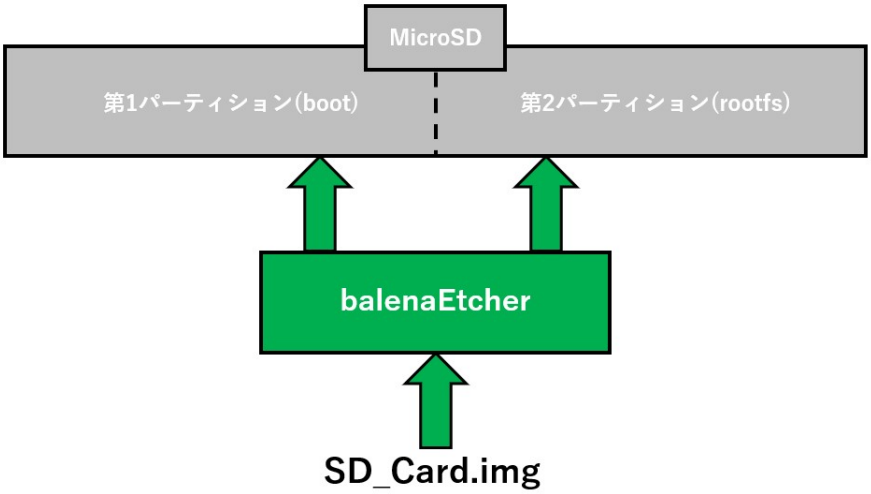

図 5.2: Mask Dataset Sample Picture

作成したデータの書き込みには balenaEtcher というツールを用いる．このツールを用いて SD __ Card.img を MicroSD カードに書き込むことで自動的に MicroSD カードのパーミッションが割り当てられる．MicroSD カードに書き込まれたデータを以下に列挙する．※実際に書き込んだら記述します.

5.1.3 実験環境の構築

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．加えて，はじめに PS のみを用いて処理を行うアプリケーションのために，python で用の OpenCV をダウンロードする.

5.2 アプリケーションの実行

※実行したら書きます.

5.3 実験結果

※結果出たら書きます.

第6章　考察

本章では，ハードウェア実装した前処理の性能に対する考察と，本研究の今後の展望についての考察を述べる．

まず，ハードウェア実装した前処理の性能に対する考察を述べる．※実験終わったら書きます

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，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．

本研究の結果※実験終わったら書きます

しかし，ハードウェア実装できた前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（）教授ならびに（）助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] 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，
2021,
”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”,
https://www.jstage.jst.go.jp/article/pjsai/JSAI2021/0/JSAI2021_3I4GS7a02/_pdf/-char/ja,
（参照 2023-01-19）

[2] 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）

[3] 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）

[4] 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）

[5] 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）

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

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