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 4 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 **4**

1.1　研究背景 4

1.2　論文の構成 5

第 2 章　準備 **6**

2.1　画像処理 6

2.1.1　物体検出 6

2.1.2　前処理 6

2.1.3　OpenCV 6

2.2　FPGA 6

2.2.1　FPGA 概要 6

2.2.2　比較 7

第 3 章　アプリケーションの実装 **8**

3.1　開発環境 8

3.2　開発フロー 9

3.2.1　プラットフォーム作成 9

3.2.2　アプリケーション作成 (PS) 13

3.2.3　アプリケーション作成 (PS+PL) 15

第 4 章　評価実験 **16**

4.1　実験方法 16

4.2　実験結果 16

4.3　考察 16

第 5 章　まとめ **17**

謝辞 **18**

参考文献 **19**

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

本論文の構成について述べる．第 1 章では研究背景について述べる．第 2 章では本論文を読むにあたって必要となる予備知識について，画像処理および FPGA の観点から説明する．第 3 章では本研究で扱う画像処理アプリケーションの実装について，開発環境，開発フロー及び

第2章　準備

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.

2.1.2　前処理

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.

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.

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 比較

第3章　アプリケーションの実装

3.1　開発環境

- 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．
- OpenCV-v3.4.16 ･･･ 本研究で扱う Vitis-v2020.1 が用いる Vitis Vision Library に対応しているバージョンの OpenCV を用いる．
  - 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

- Ultra96v2 ･･･ 本研究で扱う FPGA ボード．Ultra96v2 に搭載されている Zynq UltraScale+ MPSoC 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．

3.2 開発フロー

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．アプリケーション作成の流れとしてプラットフォームの作成，PS のみを利用して処理を行うアプリケーションの作成，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．

3.2.1 プラットフォーム作成

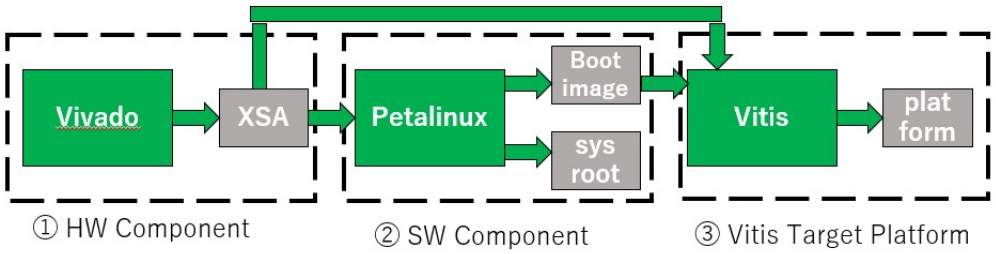

図 3.1: プラットフォーム作成フロー

プラットフォームの作成は図3.1 プロットフォーム作成フローに従って行う．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 ファイルの概要を示す．

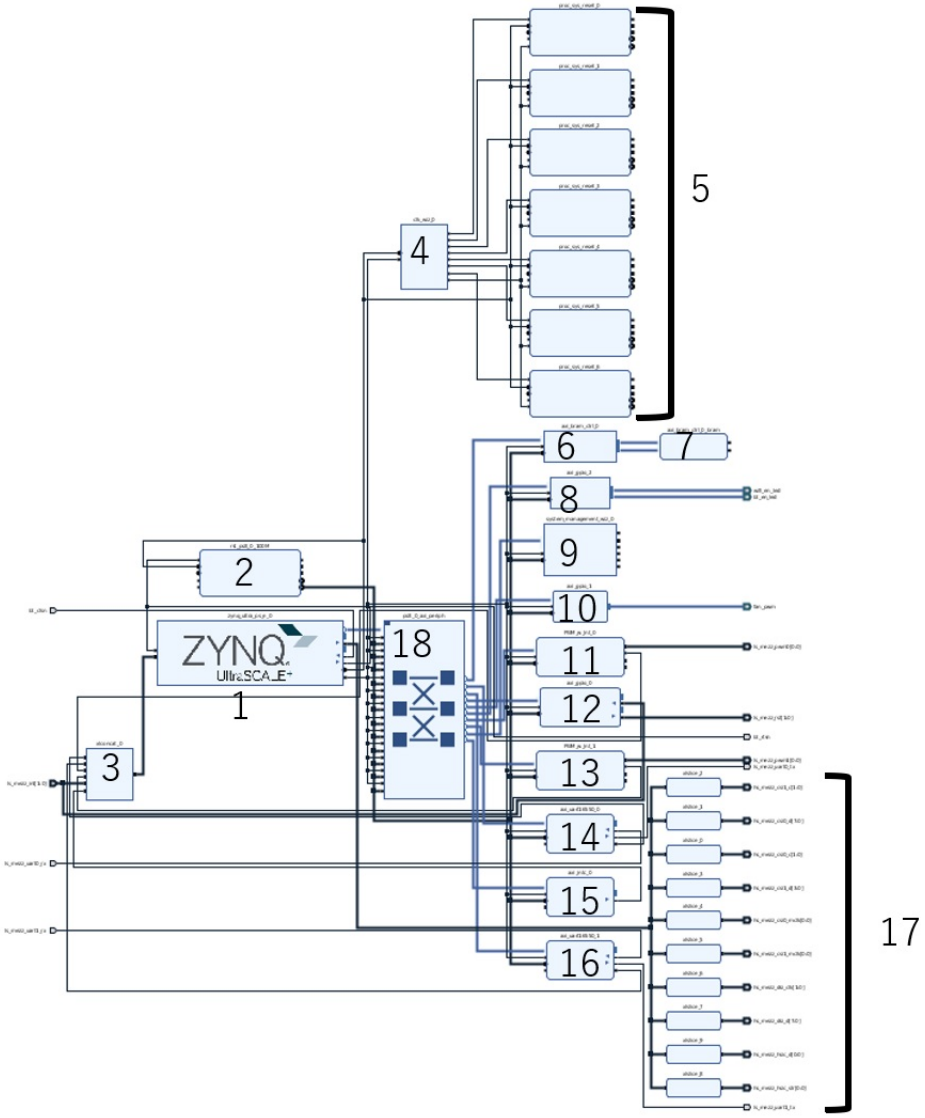

図 3.2: HW BlockDesign

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図 3.3 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

次に，2.SW Component の工程に移る．ここでは Petalinux を用いて Ultra96v2 上で起動するカーネルの作成を行う．

最後に，1.HW Component と 2.SW Component の工程で得られた成果物を用いて Vitis Target Platform の作成を行う．

3.2.2 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 3.1: 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
9  time_sta = time.time()
10
```

```
11  for i in range(600):
12
13      img = cv2.imread("./mask_before/Mask_" + str(i+1) + ".jpg")
14
15      h, w = img.shape[:2]
16
17      re_h = re_w = re_length/max(h,w)
18
19      img_resize = cv2.resize(img, dsize=None, fx=re_h, fy=re_w)
20
21      img_normal = img_resize/255
22
23  time_end = time.time()
24  t = time_end - time_sta
25
26  print(str(t))
```

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．9 行目の変数 time_sta はプログラムの実行開始時刻を取得しており，サンプル画像全てに対する処理が終了した時刻を 23 行目の変数 time_end で取得している．24 行目ではプログラムの終了時刻から開始時刻の値を減算することにより，処理時間の計測を行っている．最後に 11 行目から 21 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 11 行目の range の中は 600 に指定している．13 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 15 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．17 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．19 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 17 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．21 行目の img_normal は正規化後の画像配列を格納する．正規化とは，複数あるデータのうち，そのデータの取り得る最大値と最小値の差で各データを除算する．今回は，画像配列に対して正規化を行うため，その配列の要素の取り得る範囲は 0 から 255 である．そのため，21 行目ではリサイズ後の画像配列である img_resize を 255 で除算することにより正規化を行っている．以上の工程を経て，26 行目で処理時間の出力を行う．

3.2.3 アプリケーション作成 (PS+PL)

第4章　評価実験

4.1　実験方法

4.2　実験結果

4.3　考察

第5章　まとめ

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