YOLOv3-tinyベースCNNによる低消費電力物体認識システムのFPGA実装

松田 将朋[1,a)]　荒木 康利[1,b)]　眞邉 泰斗[1,c)]　石塚 洋一[1,d)]　柴田 裕一郎[1,e)]

概要：FPGA は並列性の高さと低消費電力性から，組み込み指向の CNN アクセラレータとして用いられることが増えているが，厳しい電力制約下においては高性能と低消費電力性の両立の困難さが問題となっている．本論文では YOLOv3-tiny をベースに Depthwise Separable Convolution を取り入れた物体検出用ネットワークを提案し，FPGA に実装した上で提案手法の処理速度，検出精度，消費電力の面での有効性を調査する．結果として，既存実装と比較して約 30 % のレイテンシと約 20 % の消費電力を削減でき，Separable Convolution が消費電力の削減や処理性能の向上に対して有効であることがわかった．

キーワード：FPGA，低消費電力アーキテクチャ，機械学習，CNN

An FPGA implementation of object recognition system with low power consumption using YOLOv3-tiny-based CNN

Masatomo Matsuda[1,a)]　Yasutoshi Araki[1,b)]　Taito Manabe[1,c)]　Yoichi Ishizuka[1,d)]　Yuichiro Shibata[1,e)]

Abstract: Although the use of FPGAs for embedded-oriented CNN accelerators has been spreading owing to a high degree of parallelism and low power consumption, it is still difficult to achive high performance with FPGAs under strict power constraint. In this paper, we propose a YOLOv3-tiny-based new network model for object detection which uses Depthwise Separable Convolution, and evaluate the effectiveness of the proposal in terms of processing speed, detection accuracy, and power consumption. As a result, we reduced 30% of total latency and 20% of total power. The results showed that Depthwise Separable Convolution is effective for reducing power consumption and improving performance.

Keywords: FPGA, Low-power consumption architecture, Machine learning, CNN

1. はじめに

近年さまざまな産業分野で，従来人手で行っていた作業を機械で自動化するという動きが活発になっている．そのような試みの一つに，人手不足に悩まされる交通誘導作業を，AI システムによって自動化することを目指す取り組みがある．システムを実現できれば，作業員の負担が軽減され，人手不足の解消が期待できるが，道路や工事現場といった場所では使用可能な電力に制限があるほか，現場での持ち運びやすさも考慮しなければならないため，可搬性と電力効率に優れたシステムが求められる．このような場面において，FPGA (Field Programmable Gate Array) は，その並列性の高さと低消費電力性から，組み込みシステムに用いられることが増えている．

交通誘導を自動で行うためには，車両の検出機能が必須である．車両の検出には画像をベースとするものや，ミリ波レーダを使うものなど様々な方法があるが，画像を用いた物体認識においては，CNN (Convolutional Neural Network) が多くの分野で使用されている．CNN ベースの物体検出

[1] 長崎大学
Nagasaki University，Bunkyo-machi，852-8521，Japan
[a)] matsuda@pca.cis.nagasaki-u.ac.jp
[b)] yaraki@pca.cis.nagasaki-u.ac.jp
[c)] tmanabe@nagasaki-u.ac.jp
[d)] isy2@nagasaki-u.ac.jp
[e)] shibata@cis.nagasaki-u.ac.jp

モデルである YOLOv3 は精度・速度ともに優れた性能が報告されている [1]．また，軽量モデルである YOLOv3-tiny は YOLOv3 より精度は劣るものの，約 6.3 倍のフレームレートを達成しており，組み込みシステム向けのモデルと言える [2]．現在我々が共同で開発を進めている交通誘導システムではカメラからの車両検出システムとして，リアルタイム処理，低消費電力性，可搬性の観点から FPGA を用いているが，カメラなどを接続した FPGA ボード全体に供給可能な電力は 4 W 程度と厳しく制約されている．

本研究では，車両検出に向けた YOLOv3-tiny をベースとした新たなネットワークモデルを提案し，組み込みシステム向けの FPGA SoC である Zynq に推論処理を実装した．その上で，提案手法が処理性能や消費電力の面でどのような効果があるかを調査した．

本論文の構成は以下の通りである．第 2 章で本論文に関連する研究を紹介する．第 3 章では提案ネットワークについて述べ，第 4 章で推論処理の FPGA 設計・実装について述べる．第 5 章では設計したシステムの評価・考察を行い，第 6 章にて結論を述べる．

2. 関連研究

本章では，YOLO の FPGA 実装に関する研究を紹介する．YOLOv3-tiny モデル向けの FPGA アーキテクチャを提案している文献 [3] では，資源量が比較的少ないローエンドの FPGA にも実装可能なアーキテクチャを設計している．ボードの消費電力も 3.36 W と比較的小さく，本プロジェクトが求める可搬性と低消費電力性を兼ね備えた実装である．一方文献 [3] は，精度をできるだけ保ちながらローエンド FPGA に実装しているため，処理速度は 1.88 FPS に留まり，工事現場における車両の流れを把握するための車両検出システムとしては，リアルタイム性に欠ける．

Yap らは YOLOv2 モデルを OpenCL を使って Cyclone V FPGA に実装している．この研究では，Convolution と Batch Normalization の統合により冗長性を減らし，さらに 16bit 固定小数点数を使うことでパフォーマンスを向上させている [4].

Duy らは YOLOv2 モデルの推論を FPGA でアクセラレーションしている．重みを 1bit，アクティベーションを 3bit から 6bit で表現することですべてのパラメータを FPGA の内部メモリに格納している．中間層の出力も FPGA 内部メモリに保持しているため，外部メモリへのアクセスを最小限に抑えることで消費電力の削減とスループットの向上を図ることができたと報告している [5]．しかし，[5] ではオンチップの電力だけでも 18.29 W を消費しており，本プロジェクトの厳しい電力制約を考慮すると，さらに小さな消費電力で動作する物体検出システムを開発する必要がある．また，[5] で用いられている FPGA は Virtex-7 XC7VX485T という比較的大規模な FPGA であり，低消費電力と可搬性が求められる本システムには向いていない．

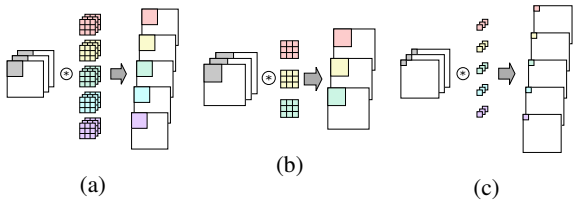

図 1: (a) 通常の畳み込み，(b) Depthwise Convolution，(c) Pointwise Convolution

Ning らは YOLOv2 モデルを Zynq xc7z035 FPGA に実装し，検出精度や電力の評価を行っている [6]．同文献ではレイヤの結合や，8 bit 固定小数点数への量子化などを行うことでハードウェアに適したネットワークにしているほか，PS-DDR メモリ と PL-DDR メモリ を使用してパラメータや中間層の出力を格納している．すべてのレイヤを並列に処理することで高いスループットを達成するとともに，消費電力が限られる場面に適用可能なほど低消費電力であると報告されているが，オンチップ電力は 5.96 W であり，本プロジェクトでの厳しい電力制約を考慮すると，本システムへの適用は難しい．

3. 提案ネットワーク

3.1 Depthwise Separable Convolution

Depthwise Separable Convolution は，MobileNet [7] で提案された手法であり，通常のカーネルサイズ 3×3 の畳み込み (図 1a) を空間方向への畳み込み (Depthwise Convolution, 図 1b) とチャネル方向への畳み込み (Pointwise Convolution, 図 1c) に分割して行うことで，畳み込み処理を近似しつつ，大幅なパラメータ削減効果が期待できる．入力チャネル数を N，カーネルサイズを K，出力チャネル数 (フィルタ数) を M とすると，通常の畳み込みのパラメータ数は式 1，Depthwise Separable Convolution のパラメータ数は式 2 のように表すことができる．

$$MK^2N \tag{1}$$

$$(M + K^2)N \tag{2}$$

$$\frac{1}{M} + \frac{1}{K^2} \tag{3}$$

これらの比は式 3 のように表されるため，フィルタ数が大きいほど，その効果は大きいと言える．本研究では YOLOv3-tiny をベースに Depthwise Separable Convolution を導入し，分類クラスを 1 クラス (Car) のみにしたモデルを提案する．

3.2 提案ネットワーク: YOLOv3-tiny-improved

本研究において提案するネットワークを YOLOv3-tiny-improved と呼ぶこととする．Depthwise Separable Convolution を利用したモデルは，第 1 層のみ通常の畳み込み層を用いることが多い [7][8][9]．しかし，表 1 に示すように，1

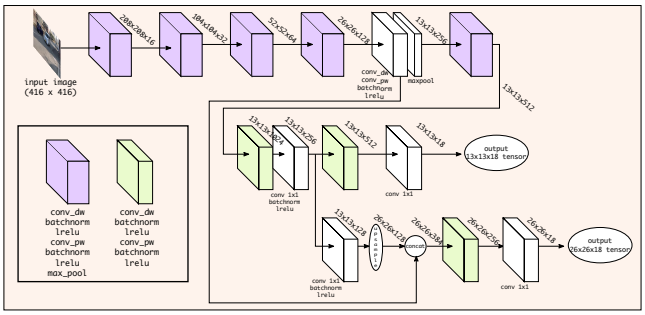

図 2: YOLOv3-tiny-improved のネットワークアーキテクチャ

表 1: test データセットにおける精度評価

Model	AP (%)	Model size (MB)
YOLOv3-tiny (1 class)	56.5	33.0
YOLOv3-tiny-improved	53.7	5.4
YOLOv3-tiny-improved′	54.0	5.4

層目が通常の畳み込み層である場合と Depthwise Separable Convolution である場合とを比較すると，AP (Average Precision) の低下は 0.3 ポイントであった．ソフトウェアにおいて，1 層目を Depthwise Separable Convolution にする場合の影響は，わずかな計算量削減とわずかな AP の低下でありメリットが小さいが，本研究における FPGA 実装では，1 種類の層につき 1 つのモジュールを実装するため，1 度しか使用されないモジュールは，ハードウェアの利用効率を低下させる．このため，YOLOv3-tiny-improved における 3×3 の畳み込み層は，第 1 層も含め全て Depthwise Separable Convolution で構成されており，ネットワーク構成は図 2 に示すとおりである．

3.3 ネットワークの学習

YOLOv3-tiny，YOLOv3-tiny-improved，YOLOv3-tiny-improved の第 1 層を通常の畳み込み層で構成したモデル (YOLOv3-tiny-improved′) を PyTorch を用いて実装し，学習を行った．なお，YOLOv3-tiny の分類クラスは 1 とした．学習には，Google Open Images Datasets [10] の物体検出用画像の内，Car，Truck，Bus，Motorcycle のラベル付けがされた画像をすべて Car クラスとして再構築した車両画像 31420 枚をデータセットとして用い，21994 枚を train，6284 枚を valid，3142 枚を test として使用した．学習率や学習アルゴリズムの決定には，[11] の実装を参考にした．

表 1 は, 学習した 3 つのモデルを test データセットによって評価した際の精度を示しており，YOLOv3-tiny-improved は YOLOv3-tiny と比べてモデルサイズが約 15 % ほどであるにも関わらず，AP の低下は約 2.8 ポイント に抑えられている．

表 2: 量子化による検出精度の比較

モデル	演算精度	AP (%)	モデルサイズ (MB)
YOLOv3-tiny-improved	Float (32 bit)	53.7	5.4
YOLOv3-tiny-improved	Fixed (16 bit)	52.4	2.7

3.4 モデルの量子化

CNN の演算は一般的に浮動小数点数で行われるが, FPGA 実装を考慮すると浮動小数点数演算よりもハードウェア量が少ない，固定小数点数演算の使用が望ましい．CNN の量子化に関する多くの研究では，小数部を 8bit とする 16bit 固定小数点数で浮動小数点数の場合と比べてほとんど精度を落とすことなく量子化できることが報告されている [3][4][12]．そのため，YOLOv3-tiny-improved においても同様に精度が保たれるのかを検証するため，16bit 固定小数点数で量子化した場合の検出精度の評価を行った．浮動小数点数から固定小数点数への変換は，式 4 に示すようにシフトと丸めを用いて行った．なお x_q，x_f はそれぞれ量子化後と量子化前の値を表し，round は四捨五入による整数への丸めを表す．

$$x_q = \mathrm{round}(x_f \times 2^8) \tag{4}$$

評価には Python を使用したが，Python では固定小数点数演算を直接サポートしていないため，整数をシフトして浮動小数点数として扱うことにより，擬似的に固定小数点数演算をシミュレートした．表 2 に，浮動小数点数と 16bit 固定小数点数での検出精度の比較を示す．

表 2 を見ると，16 bit 固定小数点数での検出精度は，単精度浮動小数点数のものと比べて，AP で 1.3 ポイントの低下が見られる程度である．このことから，YOLOv3-tiny-improved においても 16 bit 固定小数点数への量子化は有効だと言える．この検証結果から，16 bit 固定小数点数を用いた FPGA 実装を行うこととした．

4. FPGA 実装

本研究では，低消費電力性が報告されている文献 [3] の実装をベースに，FPGA 実装を行った．本実装では FPGA の資源量を考慮し，ネットワークを構成する一部の層のみを FPGA に実装する．図 3 に示すように，推論処理の各層ごとに，DMA を使用して DDR メモリから入力画像や中間層の出力をアクセラレータに転送し，積和処理などの出力結果を DDR メモリに書き戻す．本設計では 1 チップにプロセッサ部 (PS) とロジック部 (PL) が搭載された FPGA SoC と呼ばれるタイプの FPGA をターゲットとしており，各層の処理の開始時に，PS 部から AXI4-Lite 信号を利用して PL 部の各 IP のパラメータの設定を行う．

PL 部に実装された各モジュールは演算をチャネル方向に並列化し，パイプライン処理を行うが，並列に処理できる入力チャネル数は，FPGA の資源量やメモリバンド幅に

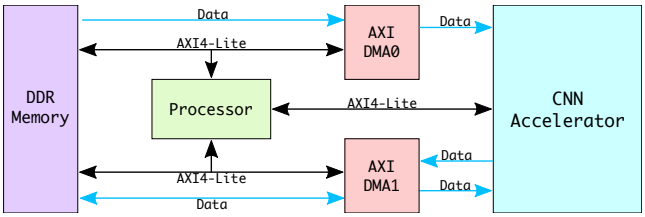

図 3: CNN アクセラレータとその周辺の構造

よって制限される．本実装では文献 [3] を参考に，並列処理するチャネル数の最大数 N_{max} を 32 とした．層の入力チャネル数 N_{in} と出力チャネル数 N_{out} が N_{max} より大きい場合は，1 層の処理を複数のサブレイヤに分ける必要があり，1 つのサブレイヤの処理ごとに DDR メモリとの通信が発生する．

FPGA の回路設計は高位合成を用いて行い，高位合成系には Vivado HLS 2019.1 を用いた．高位合成で作成した IP を用いて Vivado 2019.1 で回路を合成し，配置配線を行った．以下の節では，本研究で新たに設計した Pointwise Convolution 用モジュールと Depthwise Convolution 用モジュールの概要を説明する．

4.1 Pointwise Convolution モジュール

Pointwise Convolution モジュール (以下 Conv_pw モジュール) は，カーネルサイズ 1×1 の畳み込み演算を行う．図 4 に Conv_pw モジュールの概要を示す．通常，ストリーム入力される画素から畳み込み演算を行う場合，入力画素を保持しウィンドウを形成するためラインバッファが必要だが，Conv_pw モジュールではウィンドウを形成する必要がないため，ラインバッファが不要となる．Conv_pw モジュールでは 4 チャネル分のストリーム入力を受け取り，1 チャネル分の入力を最大 N_{max} 個にコピーし，weight buffer に格納されたそれぞれに対応する重みカーネルを乗ずる．4 チャネル分の加算を重みカーネル方向で独立して行い，$\frac{N_{max}}{4}$ チャネル分が積算されたタイミングで，出力バッファへ送る．Conv_pw モジュールの出力部は 4 チャネル分のポートをもつため，出力バッファの値を $\frac{N_{max}}{4}$ 回に分けてストリーム出力する．

4.2 Depthwise Convolution モジュール

Depthwise Convolution モジュール (以下 Conv_dw モジュール) は，チャネル間で独立したカーネルサイズ 3×3 の畳み込み演算を行う．図 4 に Conv_dw モジュールの概要を示す．Conv_dw モジュールは 4 つの Conv Kernel と，4 チャネル分のストリーム入力を保持するラインバッファを持つ．3×3 のウィンドウが揃うと，ウィンドウの切り出しを行い，weight buffer に格納された重みカーネルを使用して，Conv Kernel で畳み込み演算を行う．通常の畳み込みと比較してチャネル間での積算がないため，チャネル間での加算を行う回路が必要ない．

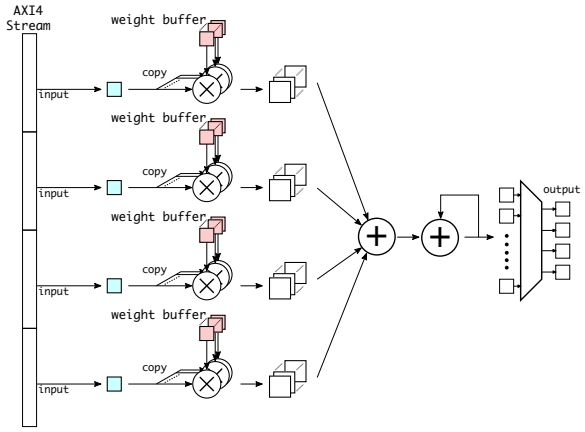

図 4: Conv_pw モジュールの概要

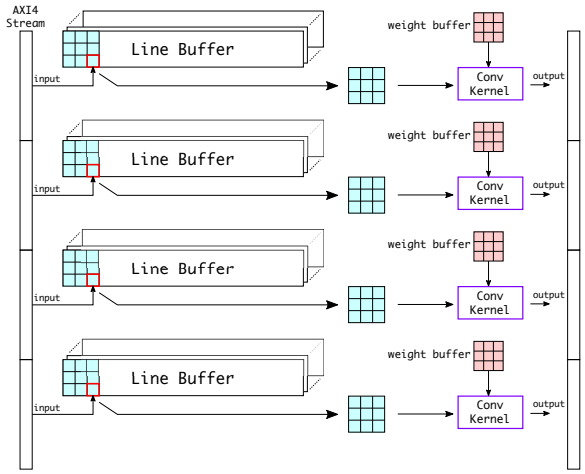

図 5: Conv_dw モジュールの概要

表 3: 資源使用量 (使用率)

Resource	Design A	Design B	Available
LUT	25830 (48.55 %)	18170 (34.15 %)	53200
LUTRAM	284 (1.63 %)	882 (5.07 %)	17400
FF	45286 (42.56 %)	19984 (18.78 %)	106400
BRAM	92 (65.71 %)	120 (85.71 %)	140
DSP	161 (73.18 %)	181 (82.27 %)	220

5. 評価

評価対象の FPGA には，Zynq-7020 (xc7z020) を用いた．本評価では，ベースとした実装と本実装との比較を行い，資源使用量，PL 部のレイテンシ，オンチップ電力の評価を行う．以下では，ベースとした実装を Design A，本実装を Design B とする．

5.1 資源使用量の評価

表 3 に Design A と Design B の資源使用量の比較を示す．表 3 を見ると，DSP や BRAM の使用率が上昇してい

表 4: PL 部における 1 回の推論のレイテンシ

Design	Latency (clock cycles)
Design A	57402524
Design B	39698222

ることがわかるが，これは Conv_dw モジュールや Conv_pw モジュールの実装により，元の実装と比べて乗算器が増えたことや，Conv_pw モジュールにおいてプラグマ ARRAY_PARTITION により重みバッファとして使用される BRAM の分割数を増やしたことによると考えられる．LUT と FF の減少は，元の実装では N_{max} 個存在した Conv Kernel が本実装では 4 つの Conv Kernel と N_{max} 個の乗算に分けられたことにより，加算などに使用されるロジックが減少したことによるものと考えられる．

5.2 レイテンシの評価

レイテンシの評価では，Vivado HLS での合成レポートを元に各層における各モジュールでの処理に要するクロックサイクル数を算出し，すべての層の処理に要するクロックサイクル数を合計することにより見積もりを行った．なお，各モジュール単体のレイテンシを合計しており，パイプライン処理による処理の重なりは考慮されていないため，実際のレイテンシは本評価での見積もりより多少小さくなると考えられる．なお，ベース実装では複数チャネルを 1 つのグループ (layer group) として処理することで DDR メモリとの通信回数を減らしており，本実装でもそれを参考にグループ化している．例えば，ベース実装では 最初の Convolution から Maxpooling までを layer group 0 としているため，本実装ではそれに相当する Depthwise Convolution から Maxpooling までを layer group 0 として比較を行っている．表 4 に推論 1 回あたりの PL 部におけるレイテンシの比較を示す．また，図 6 には layer group ごとのチャネル数の合計 ($N_{in} \times N_{out}$) と本実装によるレイテンシの削減率を示したグラフを示しており，左側の軸がチャネル数，右側の軸が削減率である．

表 4 を見ると，本実装によってレイテンシが約 30.8 % 削減されたことが分かる．図 6 を見ると，前半の layer group では 3.1 で述べたように，レイヤのチャネル数が大きいほど Depthwise Separable Convolution による削減効果が大きいように見えるが．layer group 7 以降は，あまり相関がないように見受けられる．図 7 を見ると，layer group 0 や layer group 1 などの layer group は後処理が多くの割合を占めていることが分かる．これらの層のように畳み込み層の処理にかかるクロックサイクル数の割合が小さいと，畳み込み処理の高速化の影響は小さいため，Depthwise Separable Convolution による高速化のメリットは小さい．layer group 2 以降は Depthwise Separable Convolution によるレイテンシの削減効果が表れており，後半の層になり畳み込みに要するクロックサイクル数の割合が大きくなるほど，効果が高くなっていると言える．一方で，layer group 7 以降は Acc モジュールに要するクロックサイクル数の割合の変化が小さく，削減率があまり変化しなかった．これらの layer group では畳み込み処理に要するクロックサイクル数を削減するほど，Acc モジュールに要するクロックサイクル数が支配的になったためこのような結果になったと考えられる．

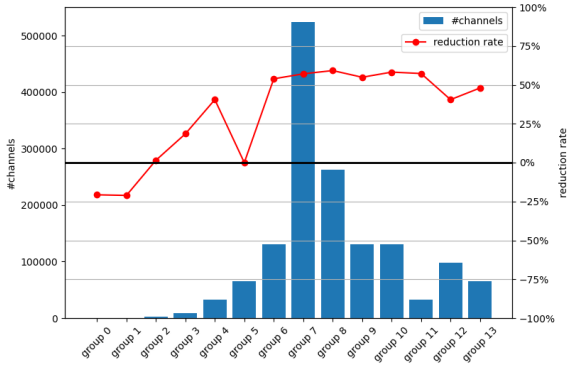

図 6: layer group 別のレイテンシの削減率

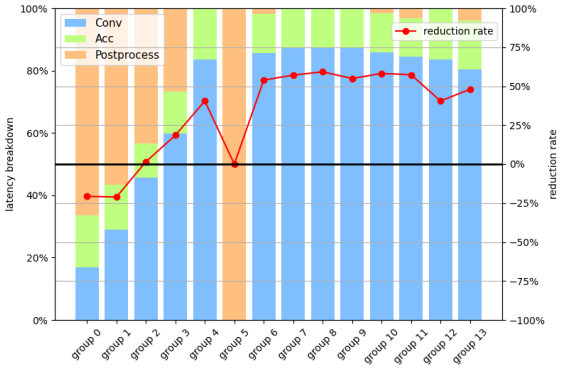

図 7: レイテンシの内訳

5.3 消費電力の評価

最後に消費電力についての評価を行う．本研究では，Vivado の配置配線実行後に Report Power コマンドを実行することにより得られる消費電力解析結果を元にベース実装との比較・評価を行った．表 5 にベース実装と本実装でのオンチップ電力の比較を示し，図 8 に動的消費電力の内訳の比較を示す．

表 5 を見ると，ベース実装に対して Depthwise Separable Convolution を取り入れた本実装ではオンチップ電力を約 0.5 W 削減できていることがわかる．また，図 8 を見ると，ベース実装の Conv モジュールが本実装の Conv_dw モジュールと Conv_pw モジュールに置き換わったことによ

表 5: ベース実装とのオンチップ電力の比較

Design	Static (W)	Dynamic (W)	Total (W)
Design A	0.180	2.325	2.505
Design B	0.165	1.838	2.003

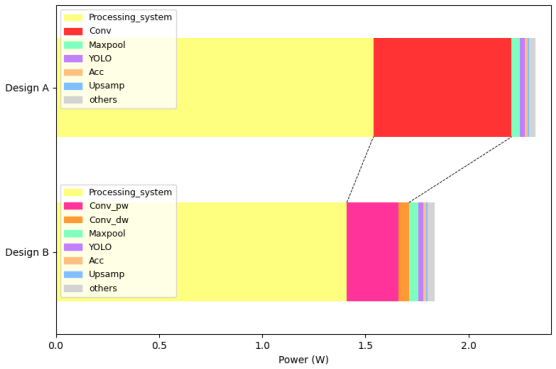

図 8: 動的電力の内訳

表 6: ベース実装と本実装におけるエネルギー効率

Design	Frequency (MHz)	Frame rate (FPS)	Energy efficiency (Frames / J)
Arm CPU	667	0.0942	–
Design A	100	1.742	0.695
Design B	100	2.519	1.258

る消費電力の削減割合が大きいことが分かる．この結果から，Depthwise Separable Convolution が FPGA の消費電力の観点からも有効であると言える．

また，参考程度ではあるが，ベース実装と同じ 100MHz で動作させたときの，レイテンシから換算した PL 部におけるフレームレートとエネルギーあたりの処理フレーム数 (フレームレート / 消費電力) は表 6 のようになる．表の 2 行目は，評価に用いた Zynq の Arm コア CPU (Arm Cortex-A9) のみを用いて，YOLOv3-tiny-improved モデルによる推論を実行した場合のフレームレートを示している．Arm コアでの推論処理は Python を用いて実装した．なお，Arm コアにおける消費電力はボード全体の消費電力となり，Design A，Design B とは基準が異なるため，表 6 における比較には載せていない．

レイテンシと消費電力の削減により，エネルギー (1 ジュール) あたりの処理フレーム数は，約 1.8 倍になっていることがわかり，提案手法によりエネルギー効率を向上させることができたと言える．また，Arm CPU を用いた場合と比較すると，本実装では 26.72 倍の高速化を達成している．

6. おわりに

本論文では，YOLOv3-tiny をベースとして Depthwise Separable Convolution を導入した車両検出モデルを提案し，FPGA に実装して性能面での効果を調査した．結果として，既存実装と比べてレイテンシを約 30 %，消費電力を約 20 % 削減できた．このことから，Depthwise Separable Convolution がハードウェア実装における速度・電力面でも効果があるという結論に至った．しかしながら，レイテンシから換算したフレームレートを見ると，まだリアルタイムでの処理とは言えないため，消費電力の低さを維持しつつ，さらなる高速化が必要である．

参考文献

[1] Joseph Redmon, Ali Farhadi: YOLOv3: An Incremental Improvement, `https://pjreddie.com/media/files/papers/YOLOv3.pdf` (2018).
[2] Joseph Chet Redmon: YOLO: Real-Time Object Detection, University of Washington (online), available from ⟨https://pjreddie.com/darknet/yolo/⟩ (accessed 2022-2-13).
[3] Zhewen Yu and Christos-Savvas Bouganis: A Parameterisable FPGA-Tailored Architecture for YOLOv3-tiny, *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, pp. 330–334 (2020).
[4] Yap June Wai, Zulkalnain bin Mohd Yussof, Sani Irwan bin Salim, Lim Kim Chuan: Fixed Point Implementation of Tiny-Yolo-v2 using OpenCL on FPGA, *International Journal of Advanced Computer Science and Applications (IJACSA)*, Vol. 9, No. 10, pp. 506–512 (2018).
[5] Duy Thanh Nguyen, Tuan Nghia Nguyen, Hyun Kim and Hyuk-Jae Lee: A High-Throughput and Power-Efficient FPGA Implementation of YOLO CNN for Object Detection, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 27, No. 8 (2019).
[6] Ning Zhang, Xin Wei, He Chen and Wenchao Liu: FPGA Implementation for CNN-Based Optical Remote Sensing Object Detection, *Electronics 2021*, Vol. 10, No. 3 (2021).
[7] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applicatoin, *arXiv: 1704.04861v1 [cs.CV]* (2017).
[8] Yadan Li, Zhenqi Han, Haoyu Xu, Lizhuang Liu, Xiaoqiang Li and Keke Zhang: YOLOv3-Lite: A Lightweight Crack Detection Network for Aircraft Structure Based on Depthwise Separable Convolutions, *Applied Sciences.*, Vol. 9, No. 18 (2019).
[9] Lanxue Dang, Peidone Pang and Jay Lee: Depth-Wise Separable Convolution Neural Network with Residual Connection for Hyperspectral Image Classification, *Remote Sens.*, Vol. 12, No. 20 (2020).
[10] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari: The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale, *IJCV* (2020).
[11] Erik Linder-Noren: PyTorch-YOLOv3, `https://github.com/eriklindernoren/PyTorch-YOLOv3`.
[12] Dallinger, D.: FPGA optimized dynamic post-training Quantization of Tiny-YoloV3, Technical report, TU WIEN, Christian Doppler Labratory (2021).