Vol.2019-ARC-235 No.13
Vol.2019-SLDM-187 No.13
Vol.2019-EMB-50 No.13
2019/3/17

プログラマブルSoCを活用した自動運転ロボットにおける交通信号検出タスクの設計空間探索

新田 泰大[1,2]　田村 爽[1,2]　高瀬 英希[1,2]　高木 一義[1]　高木 直史[1]

概要：プログラマブル SoC における SW/HW 協調システムの設計では構成可能なシステム構成の候補から，目的を満たすシステムを選択する必要がある．本稿では自動運転ロボットの交通信号検出タスクにおける設計空間探索を行い，リソース使用量および性能評価を行った．探索の結果，リソース使用量に対する処理性能の向上の観点では，HOG 特徴量計算を HW 化することで最も FPGA を活用できることが明らかになった．

キーワード：プログラマブル SoC，FPGA，協調設計環境

Design space exploration for traffic signal detection task in automatic driving robot utilizing programmable SoC

Yasuhiro Nitta[1,2]　Sou Tamura[1,2]　Hideki Takase[1,2]　Kazuyoshi Takagi[1]　Naofumi Takagi[1]

Abstract: In the SW/HW cooperative design system in the programmable SoC, it is necessary to select a system that satisfies the objective from the configurable system candidates. In this paper, we conducted a design space exploration in the traffic signal detection task of the autonomous driving robot, and evaluated resource utilization and performance. As a result of the exploration, from the viewpoint of improving the processing performance with respect to the resource utilization, it became clear that FPGA can be utilized most effectively by offloading HOG feature calculation to hardware.

Keywords: Programmable SoCs，FPGA，Codesign Environment

1. はじめに

近年，様々な状況で自律移動ロボットの需要が高まっており，研究や技術開発が盛んに行われている．無人ドローンや AI ロボットなどは，周囲の状況を分析することによって，人間の指示を介在せずに自身を制御することが求められる．制御のための複雑なソフトウェアを実行するためには高性能かつ高消費電力なプロセッサを必要とする．しかし，ロボットの小型化や長時間稼働のためには，より低消費電力な演算処理装置が必要とされる．

組込みシステムで高度な処理を低消費電力で実現する方法として，プロセッサに加えて FPGA（Field Programmable Gate Array）を組み合わせたヘテロジニアス構成を取るデバイスであるプログラマブル SoC の採用が挙げられる．プログラマブル SoC を用いたシステムでは，負荷が大きい処理を FPGA 上のハードウェア（HW）で実行し，柔軟性の求められる処理をプロセッサ上のソフトウェア（SW）で実行することができる．これにより，組込みシステムの高性能化と低消費電力化が実現できる．

プログラマブル SoC における SW/HW 協調システムにおいては，構成可能なシステムの候補が多数存在する．システム構成の候補としては，HW 化する処理の選択，HW 化される処理の最適化方式および SW/HW 間の通信方式の選択などがある．プログラマブル SoC を活用した協調システムにおいてプロセッサと FPGA のそれぞれの機能

[1] 京都大学 大学院情報学研究科
Graduate School of Informatics, Kyoto University
[2] emb@lab3.kuis.kyoto-u.ac.jp

Vol.2019-ARC-235 No.13
Vol.2019-SLDM-187 No.13
Vol.2019-EMB-50 No.13
2019/3/17

と利点を活用するためには，設計空間の中から性能の高いシステムを探索する必要がある．性能評価の指標には処理速度に加えて FPGA のリソース使用量および消費電力量などが挙げられる．

これまでに設計空間探索に関する様々な研究が行われている．文献 [1] では，ランダムフォレスト法を用いた合成結果予測モデルの作成によって，低コストな設計空間探索を実現する手法を提案している．設計空間探索においては SWORDS フレームワーク [2] を活用している．本フレームワークは，SW によるアルゴリズム記述とシステム構成情報記述から HW 化される実行モジュールおよび SW/HW 間の通信インタフェースを自動生成する．文献 [3] では，機械学習の観点から設計空間探索に取り組み，8 種類のアルゴリズムを比較している．これによれば，ランダムフォレスト法が高位合成ツール CyberWorkbench における使用リソースおよび実行時間の両方の指標において予測モデル生成に適していることが示されている．

本研究の目的は，実践的なロボットシステムにおけるプログラマブル SoC の有用性を検証することである．本研究では，プログラマブル SoC を活用した自動運転ロボットの交通信号検出タスクに対する設計空間探索を行う．探索により得られた複数の設計を比較することにより，システム構成におけるそれぞれの選択が性能に及ぼす影響について考察する．

本稿の構成は以下の通りである．まず第 2 章にて準備として，開発中の自動運転ロボットのシステムおよびロボットに搭載されるプログラマブル SoC について述べる．第 3 章にて設計空間探索の対象とする交通信号検出タスクの概要を説明し，本研究での探索対象とする設計空間を定義する．第 4 章にて設計空間探索を実施し，各システム構成の有用性について議論する．最後に第 5 章にて結論と今後の展望を述べる．

2. ZytleBot

本章では，自動運転ロボットのシステムおよびロボットに搭載されるプログラマブル SoC の概要について述べる．

我々は FPT2018 FPGA Design Competition[7] に参加するために，自動運転ロボットである ZytleBot を開発した [8]．ZytleBot という名前は，Zynq と TurtleBot[9] を統合した開発プラットフォームであることに由来する．ZytleBot の外観を図 1 に，システム構成を図 2 に示す．ZytleBot では実製品のプログラマブル SoC の 1 つである Xilinx 社の Zynq-7000 All Programmable SoC[4]（以下，Zynq）を採用している．ロボットには Zynq を搭載した Digilent 社製 Zybo Z7-20 開発ボード（以下 Zybo）が搭載されている．車体は TurtleBot3 Burger を元に構成している．TurtleBot3 は教育用のロボットであり，ROS（Robot Operating System）[10] の標準プラットフォームとなっている．

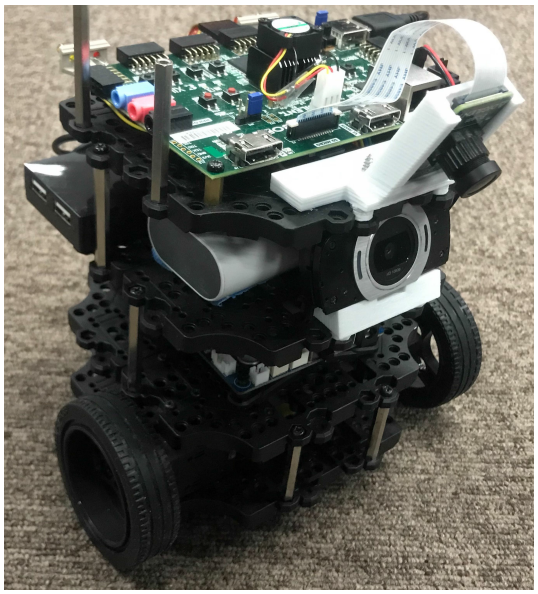

図 1 自動運転ロボット ZytleBot の外観 [8]

システムの構成としては，Zynq のプロセッサ上では Ubuntu 16.04 LTS が動作しており，Ubuntu 上で ROS Kinetic Kame を利用した自動運転システムが実行される．カメラは 2 台搭載しており，1 つは路面認識のためのカメラで，Digilent 社製 PCam 5C が Zybo に MIPI 接続されている．PCam 5C のカメラ情報は FPGA 側に直接接続される．もう 1 つのカメラは USB Web カメラであり，交通信号検出に使用する必要がある．このため，FPGA による画像処理の高速化および省電力化は必須の開発課題となる．

開発したロボットは以下のような機能を持つ．

- 路面を認識し，はみ出さずに走る
- 交差点を認識し，指示通りの道順で道路を走る
- 障害物の認識および回避を行う
- 信号機の認識および停止を行う

なお，参加したコンテストの規約上，カメラ以外のセンサは使用していない．また，ロボット外部との通信も禁止されており，走行に必要な判断および演算をすべて Zybo 上で処理する．

2.1 ROS

ロボットの自動運転システムを支援する ROS について説明する．ROS は，ロボットソフトウェアの効率的な開発を支援することを目標としたプラットフォームである．ROS では，プログラム部品をノードとして表現し，複数のノードを組み合わせることでロボットシステムを構築する．ノード間の通信層は出版購読型通信に基づくミドルウェアとして提供されており，これによって複雑なアルゴリズムを実行するアプリケーションノードと車体制御を行うノードを分離することが可能となる．本研究において開発する自動運転システムにおいても，あらかじめ SW で開

Vol.2019-ARC-235 No.13
Vol.2019-SLDM-187 No.13
Vol.2019-EMB-50 No.13
2019/3/17

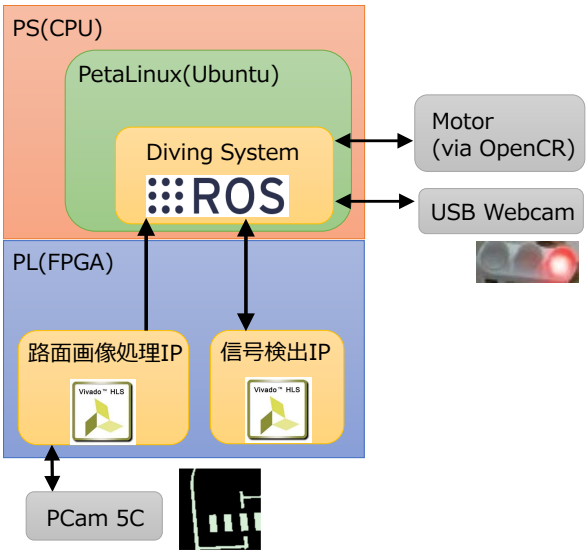

図 2　ZytleBot のシステム構成

発しておいたノードを，FPGA を利用するノードに差し替えることで他のノードに影響を及ぼすことなく HW へ移行することができる．

2.2　Zynq

Zynq は主にプロセッシングシステム（PS）とプログラマブルロジック（PL）で構成されている．PS としてはデュアルコアの ARM Cortex-A9 が搭載され，外部メモリインタフェース，DMA コントローラおよび入出力ペリフェラルなどを含む．PL としては FPGA を中心として RAM や DSP などが搭載されている．

Zynq において SW/HW 協調システム設計を実現する開発環境が Xilinx 社より提供されている．FPGA 上に実装する回路の設計，論理合成，配置配線およびビットリーム生成のためのツールとして Vivado Design Suite（以降 Vivado）[5] が提供されている．Vivado 上では Verilog HDL や VHDL などのハードウェア記述言語（HDL: Hardware Description Language）で回路を記述するか，IP（Intellectual Property）を組み合わせてシステムを構築する．HDL での複雑なアルゴリズムの設計はソフトウェアでの設計よりもコストが高いため，C/C++といった SW 向けの高級言語の記述から HDL を生成する技術である高位合成（HLS: High Level Synthesis）が注目されている．Xilinx 社からは高位合成のツールとして Vivado HLS が提供されている．本研究では交通信号検出タスクの HW 設計に Vivado HLS を採用している．

Zynq の PS 内の ARM CPU 上では Linux を動作させることができる．Vivado で合成した回路を Linux アプリケーションから駆動させるためにはブートローダおよび Linux カーネルをカスタマイズする必要があり，このためのツールとして Xilinx 社から Petalinux ツール [6] が提供

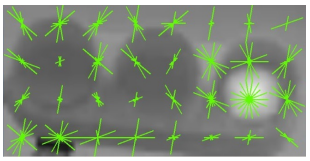

図 3　HOG 特徴量の可視化表現

されている．Petalinux ではカスタム BSP（Board Support Package）生成，Linux コンフィギュレーションをサポートしている．カスタム Linux BSP の生成により，回路に含まれる IP およびそのインタフェースが参照するメモリアドレスの情報を含むデバイスツリーが生成される．Linux の持つインタフェースの 1 つである UIO（Userspace I/O）を使用するようにデバイスツリー内に記述することで，Linux 上で各 IP は UIO デバイスとして認識される．これにより IP 固有のデバイスドライバの開発をすることなく，ユーザ空間から IP 回路の制御ができるようになる．

3.　交通信号検出タスクとその設計空間

3.1　交通信号検出タスク

本研究で対象とする交通信号検出タスクの概要を図 4 に示す．USB Web カメラから取得した画像から一部領域（ウインドウ領域）を切り出して 32*64 ピクセルの大きさにスケールし，特徴量を抽出する．特徴量は RGB 画素値，HSV 画素値および HOG 特徴量を用いる．画素値の特徴量抽出には，ウインドウ領域の画像を 8*8 ピクセルにさらに縮小したものを使用する．HOG とは Histograms of Oriented Gradients の略であり，局所領域（セル）の輝度の勾配方向をヒストグラム化したものである．図 3 に HOG 特徴量を可視化したものを示す．HOG 特徴量の計算のための前処理としてグレースケール化が必要である．本研究では，セルのサイズを 8*8 ピクセル，ブロックサイズを 2*2 セル，ヒストラグラムの階級数は 9 とした．すなわち，HOG 特徴量の要素数は $\frac{32}{8} * \frac{64}{8} * 9 = 288$ となる．

抽出した特徴量から信号が赤であるか否かの 2 分類識別を行う．識別には機械学習アルゴリズムの 1 つであるランダムフォレスト法を用いた．ランダムフォレスト法は決定木による複数の弱学習器を統合させ汎化能力を向上させる，アンサンブル学習アルゴリズムの 1 つである．ランダムフォレストに含まれる決定木の葉ノードは，その葉ノードに到達する各分類ごとの学習画像の数を示す．入力画像が赤信号である確率は，葉ノードが示す学習画像の総数に対する赤信号の学習画像の枚数の割合とする．最終的に全ての決定木における確率の平均が，入力画像の赤信号である確率として決定される．なお，ウインドウ候補の列挙にはスライディングウインドウ法を使用し，候補の数は 360 とした．全てのウインドウ候補について識別を行い，赤信号である確率が閾値を超えるウインドウ候補が 1 つ以上あればカメラ画像に赤信号が存在すると判断する．ランダム

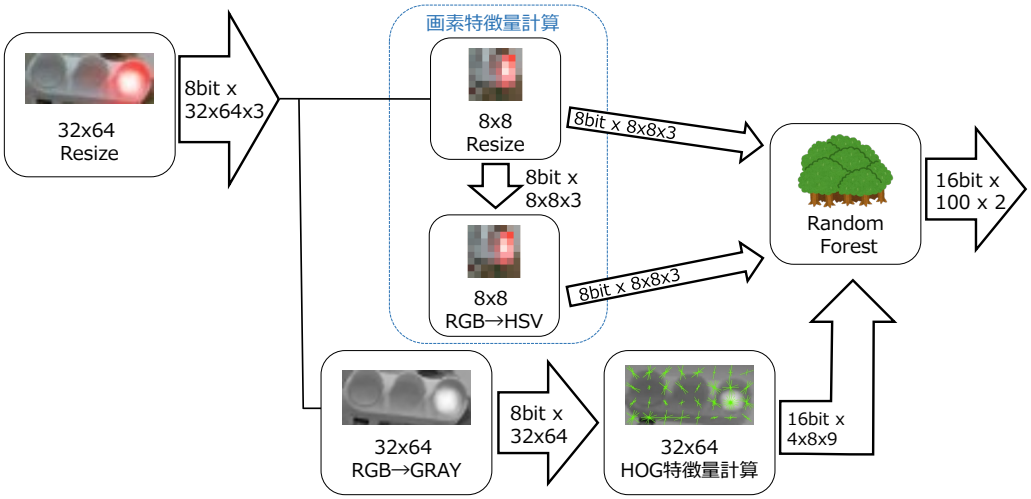

図 4　交通信号検知タスクの概要

フォレストに含まれる決定木の数は 100 とし，学習には各分類ごとに約 2000 枚の画像を用意して Python 向け機械学習ライブラリである scikit-learn[12] を使用した．抽出されたパラメータは，C++などの他言語実装に活用して推論することが可能である．

3.2　SW による処理性能

探索対象とする設計空間を定義するための足掛かりとして，全ての処理を SW のみで行った場合の実行時間を測定した．表 1 にこの測定結果を示す．なお図 4 における交通信号検知タスクの各処理の箱および矢印の相対的な大きさは，各処理の負荷および処理間の通信データ量にそれぞれ対応している．HOG 特徴量抽出における演算は，3.3 節で後述するとおり HW 設計向けに簡単化しているが，SW で処理する場合も同様の簡単化した演算を使用した．表の各処理時間は，360 枚全てのウインドウ候補における各処理の合計時間を表す．全ての処理を SW で行った場合，1 フレーム全体の処理時間の 50%以上を HOG 特徴量の計算が占めていることがわかる．次に画素特徴量の抽出，ランダムフォレストによる識別，の順に処理が大きいことがわかる．

3.3　各処理の FPGA 向け実装

図 4 に示すタスクの一部を FPGA に実装することで処理を高速化し，CPU 上の SW の処理負荷を軽減することを狙う．本節では，HOG 特徴量の計算およびランダムフォレストによる推論の HW 化に向けた実装について示す．

表 1　SW の処理性能

処理	実行時間 [ms]
画像のスケール (32*64)	12.8
画素特徴量抽出	30.6
グレイスケール化	3.4
HOG 特徴量計算	104.8
ランダムフォレストによる推論	17.7
1 フレーム全体の処理時間	192.0

HOG 特徴量の計算の HW 設計では文献 [11] に基づきラインバッファを用いた実装を行った．各画素における輝度勾配を求める際に周辺 4 画素の値が必要であり，1 画素につき 4 回のブロック RAM へのアクセスが必要になる．サイズ 3*64 のラインバッファに画素の情報を保存することで，最内ループにおけるブロック RAM へのアクセスは 1 回で済む．これにより，スループット性能が向上することが期待される．

また HOG 特徴量計算においてヒストグラムの階級を求めるには，輝度勾配に対する arctan 演算が必要になる．しかし arctan の値の計算には複雑かつ大規模な回路が生成されるため，階級の境界値と比較することで階級を決定する．具体的には，$180°/9 = 20°$ ごとの tan の値それぞれに対して，0 から 255 までの 256 個の整数の積を参照テーブルとしてブロック ROM に保存しておく．これにより，効率的にヒストグラムの階級を決定できる．

ランダムフォレストによる推論処理の HW 実装では，各決定木のパラメータをブロック ROM に保存しておき各決定木による推論を順に実行する．本研究における実装では各決定木による推論を並列実行せずに，時分割によりリソースを共有することでリソース使用量を下げている．また 100 本の決定木のパラメータをそれぞれ別のブロック ROM に割り当てた場合，ZYBO で利用可能なブロック ROM の数を超える．そのため，各 4 本の決定木のパラメータを後述する高位合成指示子 ARRAY MAP を用いて 1 つのブロック ROM に割り当てている．

3.4　設計空間の定義

本研究で探索対象とする設計空間の選択項目は，タスク

表 2　高位合成指示子の組み合わせの選択肢

HOG	(a)	指示子なし
	(b)	ARRAY PARTITION, ARRAY RESOURCE
	(c)	PIPELINE, ARRAY PARTITION, ARRAY RESOURCE
FEATURE	(d)	(a)+追加指示子なし
	(e)	(c)+追加指示子なし
	(f)	(c)+PIPELINE, ARRAY RESOURCE
	(g)	(c)+PIPELINE, ARRAY RESOURCE, ALLOCATION
ALL	(h)	(g)+ARRAY MAP

の SW/HW 切り分け，高位合成指示子による最適化，および高位合成 IP の並列数の 3 つとした．以下ではそれぞれの項目における選択肢について述べる．

1) タスクの SW/HW 切り分け
図 4 に示す処理の相対的な大きさおよび通信データ量から，タスクの切り分けの選択肢は以下の通りとした．
- 全て SW で処理
- HOG 特徴量計算のみ HW 化
- 特徴量計算全てを HW 化
- 特徴量計算および推論の全ての処理を HW 化

FPGA を使用する場合は各選択によって入出力の情報量，すなわち入出力のためのデータ転送にかかる時間が異なる．また，FPGA に実装される処理は，1 枚のウインドウに対する処理の一部であるため，1 フレームの処理ごとにウインドウの候補の数だけ HW を呼び出す必要がある．

2) 高位合成指示子による最適化
本研究ではタスクの HW 設計には高位合成ツールである Vivado HLS を使用する．Vivado HLS では高位合成対象の C++コードに pragma 指示子を挿入することにより生成する IP（HLS IP）の最適化ができる．本研究で使用する最適化指示子について説明する．
- PIPELINE 指示子
ループまたは関数内での演算を同時処理できるようにして開始間隔を削減することができる．下の階層にもループが含まれる場合，そのループは自動的に展開される．このため，使用されるリソース量が増大する．パイプラインの開始間隔についてもオプションで指定することができるが，アクセス競合により指定した開始間隔の HW を合成できないことがある．本研究ではこの指示子を使用する場合は，パイプラインの開始間隔は最も短くなるように設定した．
- ARRAY PARTITION 指示子
大きな配列を小さな配列に分割し，同時アクセスの競合のないブロック RAM に配置することができる．これにより，ブロック RAM アクセスの競合が原因となるボトルネックを解消することができる．
- ARRAY RESOURCE 指示子
変数を実装するのに使用するリソースを指定することができる．本研究ではこの指示子を使用する場合は，2 ポートブロック RAM を使用するように指定することで読み込みポートと書き込みポートを別々に使用できるようになる．これにより，BRAM アクセスの競合が原因となるボトルネックを解消することができる．
- ARRAY MAP 指示子
複数の小さな配列を 1 つの配列にまとめることができる．これにより使用されるブロック RAM の数が削減されるが，ブロック RAM アクセスの競合が発生する可能性がある．
- ALLOCATION 指示子
演算子，コア，または関数が使用される数を制限することができる．これにより使用されるリソース量を削減することができる．

3) 高位合成 IP の並列数
高位合成ツールにより生成された IP コアを複数配置することで，複数のウインドウ領域の処理を並列化することができる．並列化により HW タスクの総実行時間を短縮することができるが，多くのリソースを必要とする．また，CPU との通信速度は限られているため通信時間がボトルネックになる可能性がある．並列数の選択肢は 1，2 および 4 とした．

3.5　設計空間探索の対象とするシステム構成

本研究で探索対象とする設計について述べる．高位合成指示子の組み合わせの選択肢を表 2 に示す．(a) から (c) までは HOG 特徴量計算のみを HW 実行とする．(b) では ARRAY RESOURCE 指示子および ARRAY PARTITION 指示子をヒストグラムの値を保存する変数に適用する．さらに (c) では HOG 特徴量計算のループをパイプライン化する．いずれもスループットの向上を目的としている．(d) から (g) までは HOG を含む特徴量計算すべてを HW で処理する．(f) および (g) では 32*64 ピクセルの画像の縮小および RGB から HSV への色空間の変換におけるループをパイプライン化し，さらに画素特徴量を保存する変数を 2 ポートブロック RAM に割り当てる．色空間の変換の関数には除算が含まれており，リソースを多く使用する．(g) では ALLOCATION 指示子により色空間の変換

Vol.2019-ARC-235 No.13
Vol.2019-SLDM-187 No.13
Vol.2019-EMB-50 No.13
2019/3/17

表 3　HLS IP の評価

		HLS IP Latency (clock cycle)	
		min	max
HOG	(a)	32975	442575
	(b)	13715	13715
	(c)	7570	7570
FEATURE	(d)	38428	462876
	(e)	13027	15587
	(f)	12513	12513
	(g)	12578	12578
ALL	(h)	14234	25634

表 4　各設計の評価

			Block Design Utilization					Performance	
		HLS IP * #Parallel	LUT	LUTRAM	FF	BRAM	DSP	HW time (ms)	fps
HOG	1	a*1	9581	996	14035	4.5	5	153.3	3.8
	2	b*1	8229	1015	8969	5	5	52.6	6.3
	3	c*1	7938	1036	8328	8.5	5	30.2	7.3
	4	c*4	27659	3364	28357	34	20	8.9	9.6
FEATURE	5	d*1	15324	1927	18510	5.5	17	233.7	3.1
	6	e*1	13877	1970	12610	9.5	17	67.5	6.7
	7	f*1	15375	2196	14205	9.5	17	56.3	7.1
	8	f*2	29509	4145	26890	19	34	29.0	8.8
	9	f*4							
	10	g*1	13051	2078	12167	9.5	17	56.5	7.1
	11	g*4	47900	7530	43713	38	68	15.0	10.4
ALL	12	h*1	18731	2197	19927	121	17	66.4	8.5
Available			53200	17400	106400	140	220		

のインスタンス数を 1 つまでに制限する．これにより，リソース使用量の低下を見込むことができる．

(h) では特徴量計算およびランダムフォレストによる推論を行う．ARRAY MAP 演算子を用いて複数の決定木のパラメータを 1 つのブロック ROM に統合することで，リソース使用量を低減している．

表 2 に示す 8 種類の HLS IP に対して，並列化が可能でありそれによる高速化が見込まれるものは並列度も選択する．評価対象の設計空間は全 12 個となった．

なお PS-PL 間の通信については，全ての設計においてポートは HP ポートを使用し，プロトコルは IP コアの制御に AXI4-Lite を，入出力のデータ転送にはバースト転送可能な AXI4 を使用した．

4. 設計空間探索の評価

設計空間内の全 12 の設計を合成し，PS の CPU 上で動作する Ubuntu OS からそれぞれの設計の回路を利用して評価を行った．全ての設計において高位合成の際にクロックサイクルは 10ns を指定し，高位合成した IP に入力するクロックの周波数は 100MHz とした．

高位合成ツールによる HLS IP コアおよび全 12 の設計の評価の結果を表 3 および表 4 に示す．表 3 は Vivado HLS が示す IP コアのレイテンシを示し，表 4 は各設計の構成（IP コアの種類と並列数），IP コアを含む回路全体の論理合成後のリソース使用量，および SW/HW 協調信号検出タスクにより実測された処理性能値を表している．表の最下部には Zybo に搭載されている FPGA の利用可能な素子数を表す．処理性能評価のためにそれぞれの設計において 100 フレームの処理を行い，SW からの HW 呼び出し平均時間，および，ソフトウェアにおける処理を含めた 1 フレーム全体の平均処理時間を計測した．評価をする際は交通信号検出タスクのみ動作させ，自動運転システムに必要な他タスクは起動しない状態で行った．SW からの HW 呼び出し時間は PS-PL 間のデータ転送時間が含まれている．なお，設計 9 は LUT 使用率が 100%を超えており配置配線に失敗した．

1 秒あたりのフレーム処理能力 (fps) の観点から各設計の性能を比較した場合，設計 11 が 10.4fps と最も性能が高く，SW で処理した場合のおよそ 4 倍の高速化を実現した．HOG 特徴量のみ HW 化した設計の中では，HLS IP を 4 並列で実行する設計 4 が最も処理が高速であり，HOG 特徴量の計算は SW で処理した場合に比べて 11.7 倍高速となった．

タスクの SW/HW 切り分け選択について議論する．HOG 特徴量計算のみを HW 化した設計 3 に対して特徴量計算全てを HW 化した設計 7，10 のほうが処理性能が低い．そ

れぞれを 4 並列にした設計 4，11 を比較しても，HOG 特徴量計算に加えて画素特徴量の計算を HW 化することによる処理性能の向上は乏しいと言える．この原因として，1 つのウインドウにつき，HOG 特徴量を計算する場合は入力が 1024Byte で出力が 576Byte であるのに対し，特徴量全てを計算する場合は入力が 3072Byte で出力が 1344Byte であり約 2.8 倍のデータを転送する必要があることが挙げられる．PS-PL 間の通信がボトルネックとなり，リソース使用量が増加しているにも関わらず処理性能の向上は乏しい．また，特徴量計算に加えてランダムフォレストによる推論までを HW 化した設計 12 は IP が 1 並列の設計の中では最も処理性能が高いが，ブロック RAM の使用率が高く IP の並列化ができない．

さらに高位合成指示子について議論する．設計 1 に対して設計 3 は高位合成指示子の違いだけで HW の実行時間が 5 倍高速である．パイプライン化およびブロック RAM 分割によるリソース競合回避の効果が確認できる．リソース使用量についても，パイプライン化を行ったにも関わらず，LUT と FF に関しては減少している．また設計 9 と 10 に関して，設計 9 が LUT 使用率が 100%を超え合成不能であるのに対し，設計 10 では ALLOCATION 指示子により除算回路のリソースを減らすことで IP の 4 並列化が実現できた．

今回の評価では交通信号検出タスクのみを動作させて fps を評価したが，実際の自動運転システムに組み込んだ場合，交通信号検出タスクにおける CPU 処理の速度が低下し，全体としてタスクの処理性能が低下する．より多くの処理を FPGA 化する場合は CPU の処理速度の低下の影響を受けにくいと考えられる．

5. おわりに

本研究では，プログラマブル SoC を活用した自動運転ロボットにおける交通信号検出タスクの設計空間探索を行った． 探索の結果，リソース使用量に対する処理性能の向上の観点では HOG 特徴量計算を HW 化することが最も FPGA を活用できていると言える．今後の展望として，設計空間の説明変数として PS-PL 間の通信ポートおよびプロトコルを加えることが考えられる．AXI4-Stream プロトコルによる転送によりさらに処理性能が向上することが期待される．

謝辞 本研究の一部は，2017 年度大川情報通信基金助成金および JST さきがけ JPMJPR18M8 の支援を受けたものである．

参考文献

[1] 谷祐輔，高瀬英希，高木一義，高木直史，協調設計環境 SWORDS フレームワークのツールチェーン実装および設計空間探索への適用, 情報処理学会研究報告, Vol. 2017-SLDM-179, No. 18, pp.1-6, 久米島, 2017
[2] 新田泰大，高瀬英希，高木一義，高木直史，SWORDS フレームワークにおける SW/HW 通信方式の自動選択に向けた検討, 電子情報通信学会技術研究報告, Vol. 118, No. 63, RECONF2018-8, pp. 39-44, 大崎, 2018 年 5 月.
[3] Liu, H.-Y. and Carloni, L. P.: On learning-based methods for designspace exploration with high-level synthesis, Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE, IEEE, pp.17 (2013).
[4] Xilinx 社，Zynq-7000 All Programmable SoC テクニカルリファレンスマニュアル https://japan.xilinx.com/products/silicon-devices/soc/zynq-7000.html.
[5] Xilinx 社，Vivado Design Suite https://japan.xilinx.com/products/design-tools/vivado.html
[6] PetaLinux ツール資料: リファレンスガイド (UG1144), Xilinx 社
[7] FPT2018 FPGA Design Competition, https://wwp.shizuoka.ac.jp/fpt-design-contest/
[8] 田村 爽, 新田泰大, 高瀬英希, 高木一義, 高木直史 ROS ベースの自律移動ロボットにおける FPGA 統合開発プラットフォーム，信学技報, vol. 118, no. 432, RECONF2018-53, pp. 43-48, 2019 年 1 月.
[9] ROBOTIS TurtleBot3, http://www.robotis.us/turtlebot-3/
[10] Open Source Robotics Foundation, http://wiki.ros.org/
[11] Vinh Ngo, Arnau Casadevall, Marc Codina, David Castells-Rufas, Jordi Carrabina, “A High-Performance HOG Extractor on FPGA” arXiv:1802.02187.
[12] scikit-learn, https://scikit-learn.org/stable/