一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
VLD2018-79,CPSY2018-89,RECONF2018-53(2019-01)

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

田村　爽†　新田　泰大†　高瀬　英希†　高木　一義†　高木　直史†

† 京都大学大学院情報学研究科，京都府京都市左京区吉田本町
E-mail: †emb@lab3.kuis.kyoto-u.ac.jp

あらまし　自律移動型のロボットに高度なデータ処理が可能である高性能なプロセッサを組み込むことは，コストだけでなく消費電力の大きさから難しい場合がある．このときにFPGAを活用することで，処理性能向上や消費電力削減が期待できる．しかし，FPGAにおけるハードウェアの設計コストは高く，ロボットシステムに組み込むことの敷居も高くなっている．そこで本稿では，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としてZytleBotを提案する．FPGAの活用方式として，センサのデータをFPGAを介して取得する方式およびCPUからFPGAにデータを送る方式の2種類を提案する．負荷の大きい処理をFPGA上にオフロードすることにより，低消費電力かつ高性能な自律移動ロボットを実現することができる．またROSを使用することで，多岐にわたる専門知識を要求されるロボット開発のコストを軽減することも可能となる．さらに本稿では，提案するプラットフォームの適用事例を示し，その有用性を議論する．

キーワード　プログラマブルSoC，FPGA，ROS

Sou TAMURA†, Yasuhiro NITTA†, Hideki TAKASE†, Kazuyoshi TAKAGI†, and Naofumi TAKAGI†

† Graduate School of Informatics, Kyoto University, Kyoto Sakyoku Yoshida-Honmachi
E-mail: †emb@lab3.kuis.kyoto-u.ac.jp

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，技術開発および研究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，人間の判断および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．

組込みシステムにおいて高度な処理を低消費電力で実現する方法として，プロセッサとFPGAを同一チップ上に集積したデバイスであるプログラマブルSoCを用いることが挙げられる．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，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で実行するように設計することで，高性能化と低消費電力化の両立を図ることが可能となる．しかし，プログラマブルSoCの利点を活かしたシステムの設計には，ハードウェアとソフトウェアのそれぞれの深い知識が必要となる．

ロボット開発においても多岐にわたる専門知識が要求される．ロボット開発においては生産性を向上させるために様々なフレームワークやミドルウェアが提供されている．ROS（Robot Operating System）[1] はロボット用ソフトウェアの設計生産性の向上に資する開発支援フレームワークである．ROSでは，ソフトウェア部品はノードとして表現され，ノード間の通信レイヤのフレームワークが提案されている．また，様々なシミュレータやデバッグツールによるロボット開発がサポートされる．

本研究では，ROSを用いたロボットシステムへのFPGAの導入を議論する．これを実現するためには，ROSおよびFPGAの深い理解が必要であり，実現するためには多大な開発コストが必要となる．そこで，我々はプログラマブルSoC上で動作するROSベースの自律移動ロボットプラットフォームであるZytleBotを提案する．提案するプラットフォームはFPGAおよびROSそれぞれの利点を活用できるよう設計されている．FPGAはROSのノードの中の一部の機能としてラップされて

おり，ユーザーは FPGA を利用するための固有の設計を考えることなくハードウェアアクセラレートを行うことができる．また，ROS から提供されている機能は全て用いることができるため，ロボット開発を効率的に行うことが可能である．

ARM プロセッサを介した FPGA の活用方法について，提案するプラットフォームでは 2 種類の方式を提供する．1 つ目はセンサからのデータを直接 FPGA で受け取り，CPU に送る前に前処理を行う方式である．これにより，画像データの歪曲補正，二値化およびエッジ検出などの処理のオフロードを可能とし，CPU だけでは困難な処理機能を実現できる．2 つ目は，CPU で処理しているデータの一部を FPGA に送り，その結果を FPGA から受け取る方式である．これは，CPU と FPGA の間での通信時間の遅延が大きいため，リアルタイム性を損ねる可能性があるが，CPU で処理した前データを FPGA で処理することができる．

本稿では，プラットフォームを利用した適用事例として，FPT2018 FPGA Design Competition に参加するために開発したロボットを紹介する．搭載されている組込み向け CPU では処理しきれない計算を FPGA にオフロードすることによってリアルタイム性を確保し，安定した走行を実現した．また，FPGA を含むノードであっても ROS の機能である rosbag, catkin_make などが利用できることを検証することで，ROS を用いる有用性が保たれていることを示す．

2. ROS（Robot Operating System）

2.1 概要

ROS は，WillowGarage とスタンフォード大学が共同で開発したロボットソフトウェアの開発フレームワークである．OSRF（Open Source Robotics Foundation）[1] によってオープンソースコミュニティで開発が進められている．ロボットソフトウェアの再利用性を高めることを目標として開発された．

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ハードウェアの制御を行うノードを分離することが可能となる．ノード間の通信は出版購読通信と呼ばれる方式で実現される．トピックとしてデータの種類を分別し，パブリッシャがトピックに対して出版したデータを各サブスクライバノードが自身のタイミングで購読する一対他の通信モデルである．

図 1 にトピックによるノード間での出版購読通信のモデルを表す．扱うトピック名が同一であれば，ROS ロボットシステムにおいて，ハードウェアとそのノードを他のハードウェアと対応するノードに容易に変更できる．プログラム実行時には，ノードを管理する ROS マスタとなる roscore を立ち上げることにより，それぞれのノードの名前空間や変数空間を管理する．

2.2 ROS システムにおける通信プロトコル

ROS システムでは，ノード間およびノードと ROS マスタ間との通信では以下のプロトコルが使用されている．

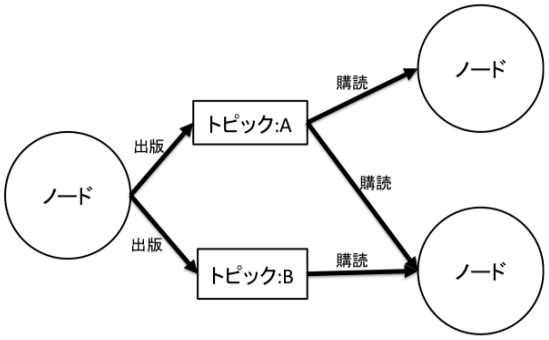

図 1: ROS におけるノード間の出版購読通信モデル

- XML-RPC 通信

XML-RPC は，通信データを XML でエンコードし，HTTP で転送することで手続き呼び出しを行う RPC プロトコル [2] である．ROS システムでは XML-RPC を用いてノードと ROS マスタ間およびノード間での手続き呼び出しを行う．

- TCPROS データプロトコル

TCPROS は ROS のデータ転送のためのトランスポートレイヤであり，pub/sub 通信およびサーバクライアント通信のデータ転送で使用される．TCPROS データプロトコルでは，データ内容を識別するためにノードの設定に対応して特有のフィールドが規定されている．それぞれのフィールドに対応するデータに対して，長さを示すビット列を 4B 分付与したものを 1 つのメッセージとして転送する．接続が確立した際に，一度だけノード情報をもつコネクションヘッダを送受信しその後データ転送を行う．

ROS における一般的な通信プロトコルは以上の通りであるが，組み込みシステムにおいて画像等の大量データ転送では TCPROS によるオーバーヘッドが問題となる．ノード間で画像データをやり取りする場合，一般的には ros::bridge と呼ばれる ROS の通信用の型に変形して，他ノードに対し publish する．通信量を減らしたい場合は，画像データの圧縮処理を行う．しかし，画像データの圧縮処理自体が CPU に負荷をかけることとなる．そのため，大量の通信による遅延もしくは圧縮処理がボトルネックとなる可能性がある．

この問題を解決する方法として，ROS の機能である Nodelet を利用することが挙げられる．従来の ROS のノードはそれぞれが別プロセスで起動しており，前述のとおり他ノードに対してデータを送信する場合は TCPROS プロトコルによるデータコピーを必要とする．しかし，Nodelet は各ノードを同プロセスのスレッドとして起動することで，メモリ空間を共有することができるため，データコピーを必要としない．これによるメリットとして，

- 通信時間の削減によるリアルタイム性の確保
- 通信時間を考慮しないで済むため，大量のデータを非圧縮で他ノードが参照可能

といった点が挙げられる．そのため，Nodelet は通信にかかるコストを大幅に削減し，ROS のデメリットを解消する．

2.3 関連研究

[3] では，ROS システムに FPGA を導入可能とする技術として，ROS 準拠 FPGA コンポーネント技術が提案された．この論文では，(1) プログラマブル SoC を用いて構築する手法および (2) FPGA 回路のみで ROS ノードを実現する手法が提案された．(1) では Xillinux および Xillybus￥citexillybus を用いており，通常のソフトウェア ROS ノードが ARM プロセッサ上で動作するため，ソフトウェア ROS システムと高い親和性，相互運用性を持つ．しかし，ROS のノード間通信のための TCP/IP 通信の遅延が問題とされている．(2) では通信遅延削減のためのハードウェア TCP/IP スタックを用いて FPGA 回路のみで ROS ノードを実現し，ROS の通信における大幅な性能向上が可能であることを示した．一方，現状では ROS メッセージ解釈・生成のハードウェア実装がは一部のメッセージ型にしか対応していないため，通信に用いることができる型が限定されている．

また, [4] では，FPGA ボードとラップトップを搭載し，ラップトップ上の ROS ノードから FPGA ボード上の資源を扱う手法を提案している．PC の ROS ノードと FPGA ボードの CPU の間のインタフェース，および FPGA ボードの PS/PL 間のインタフェースを自動生成することによって，ROS システムへのハードウェアアクセラレート導入を実現している．

3. FPGA 統合開発プラットフォーム

プラットフォームのシステム構成を図 2 に，ハードウェア構成を表 1 に示す．プラットフォームの外観は図 3 である．

本研究の提案である ZytleBot は，Zynq と TurtleBot3 の統合であるという由来から名前付けている．Zybo は Zynq-7020 APSoC [5] を搭載した開発ボードで，マルチメディアや多数の周辺機器接続をサポートしている．車体には自律移動ロボットアプリケーションの研究・開発のためのロボットキットである TurtleBot3 [6] を採用した．Zynq 上に Ubuntu16.04, ROS のディストリビューションは Kinetic Kame である．センサ類については Zybo に搭載されている MIPI，PMOD および USB ポートからの入力を想定している．また，TurtleBot3 のキットに含まれる Arduino 準拠のマイクロコントローラである openCR を用いる．ROS ノードから openCR を介してモータを制御するパッケージが公開されているので，これを利用することでロボットを制御する．

Zybo 上で Linux を動作させるためにはブートローダーおよび 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 社から提供されている Petalinux [7] を利用している．Petalinux ではカスタム BSP 生成，Linux コンフィギュレーション，およびソフトウェア開発ツールをサポートしている.Linux カーネルは 3 つのレベル（システムコールインターフェース，カーネルコード，および BSP）に分割できる．BSP は Board Support Package の略で，特定のアーキテクチャに対してプロセッサおよびプラットフォームに固有のパッケージとして機能する．PetaLinux ツールは，作成した回路および開発ボードのアーキテクチャに対応したカスタム Linux BSP を自動生成することができる．

表 1: プラットフォームのハードウェア構成

Body	TurtleBot3
Main board	Zybo Z7-20
OS	Ubuntu 16.04
ROS	Kinetic Kame

3.1 ROS システムにおける FPGA 活用方法

提案するプラットフォームでは 2 種類の FPGA の活用方式を提案する．それぞれの ROS ノードの概要を次に記す．

（1） センサのデータを FPGA を介して取得する方式

ROS では，ROS クライアントライブラリとしてアプリケーションを記述するための API がユーザに提供されている．この API を利用することによって，FPGA からデータを取得し送信する ROS ノードを実装している．具体的には，ros::NodeHandle::advertise() によってあらかじめ ROS のマスタへの登録を行う．FPGA からのデータ取得はポーリングで行い，ポーリングの実行毎に ros::NodeHandle::publish() を呼び出すことによってデータの出版を行う．

（2） CPU から FPGA にデータを送る方式

初期化関数の中であらかじめパラメータを設定し，同時に ros::NodeHandle::subscribe() によりサブスクライバーとして ROS のマスタへの登録を行う．CPU で取得したデータを FPGA に送る処理は，購読先として登録しているトピックに出版があった場合のコールバック関数として実装されている．

3.2 Nodelet 対応

我々の提案するプラットフォームでは，Nodelet の利用を想定している．通信によるオーバーヘッドを抑えることで，FPGA をより活用することが可能となる．また，それぞれのプログラムがスレッド単位になることで，スケジューリングを可能とする．重要な機能を優先的にスケジューリングすることによって，リアルタイム性が求められる処理を重視しつつ他の機能を実行することが可能となる．

Nodelet は，通常のノードで起動する場合と比較して一部の機能が使えない，複雑化により開発のコストがかかる，およびプログラムの疎結合性が失われるといったデメリットも存在する．また，それぞれのスレッドのスケジューリングを行う必要があるが，ノードの機能や実行にかかる負荷などを把握した上で調整する必要があるため，設計の理解が不可欠となる．しかし，我々の提案するプラットフォームではデータの送受信部分を Nodelet に対応させたものをパッケージ化しているため，ユーザーはアルゴリズムを変更するだけでよい．そのため，設計の困難さは緩和されている．

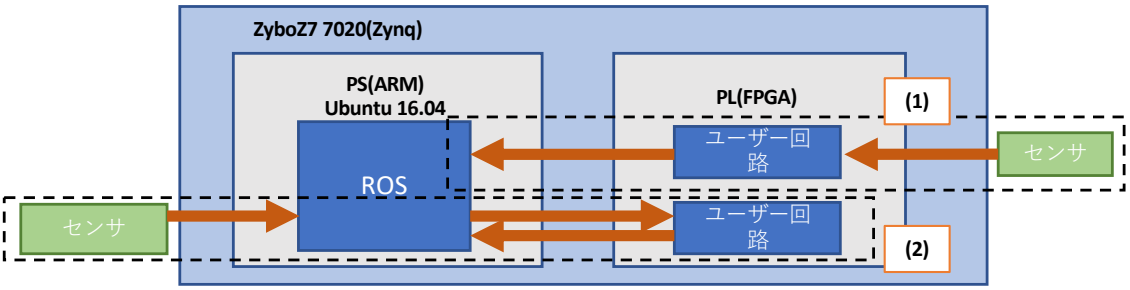

図 2: プラットフォームのシステム構成

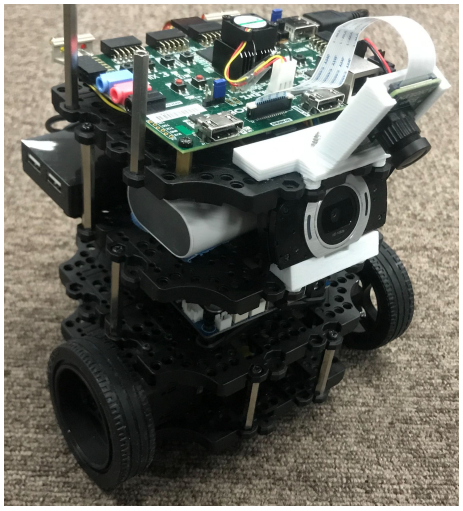

図 3: プラットフォームの外観

3.3 開発フロー

提案するプラットフォームで想定されている開発フローは図 4 ようになる．まず，各ノードおよび FPGA の外部仕様を決定する．ROS ノードの開発は，Gazebo [8] などのシミュレータや rviz といった可視化ツールを利用することで効率的に行うことができる．Gazebo は ROS によって動作するロボットのための 3D モデルの仮想シミュレータである．Gazebo 上のセンサを ROS システムに登録させることができるため，ROS トピックに擬似的な情報を流すことができる．また，Gazebo 上で作成した仮想のロボットを ROS システムに登録することで，開発した ROS ノードの出力によるロボットのふるまいを確認することができる．ロボット開発において実機での検証はコストがかかるため，このようなシミュレータを用いての開発が一般的である．

FPGA においては，通常の Linux で扱うことを想定した設計でよく，ROS を意識した設計手法を取る必要はない．Linux で FPGA を扱うためには，C や C++ によるインタフェースでの操作が一般的である．これらの C および C++ ファイルを ROS ビルドシステムに含めてコンパイルすることで，ROS ノード内で FPGA との通信インタフェースを利用することができる．ROS では CMake [9] と Python を組み合わせたビルドシステムである catkin_make が採用されており，CMakeLists というテキストファイルでターゲットやライブラリを指定する．ROS ノードおよび FPGA との通信インタフェースに必要なファイルをまとめてビルドするように CMakeLists に記述することで，ROS ノードにおいて FPGA との通信インタフェースの利用が可能となる．

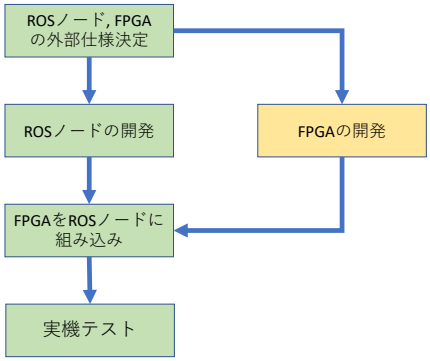

図 4: 開発フロー

ノードの開発が完了すれば，それぞれのノードを同時に動かすよう統合する．rosbag などのデバッグツールを用いることで，より効率的なテストが可能となる．統合するに当たっては，複数のノードを動作させても正しく動作するか，特にリアルタイム性が目標となる値を満たしているかが重要となる．問題が生じた場合は，それぞれのノードの CPU 使用率や処理にかかる時間などを計測し，ボトルネックを特定する．ノード間通信がボトルネックとなっている場合，Nodelet による問題解決が期待できる．

4. 適用事例：ミニチュアの道路を走る自律移動ロボット

FPT2018 FPGA Design Competition [10] に参加するためのロボットを，提案するプラットフォームを用いて開発した．

4.1 コンテスト概要

図 6 のようなミニチュアの道路上に FPGA により制御された自動車を走らせる．以下の課題をどれだけ達成できるたかに

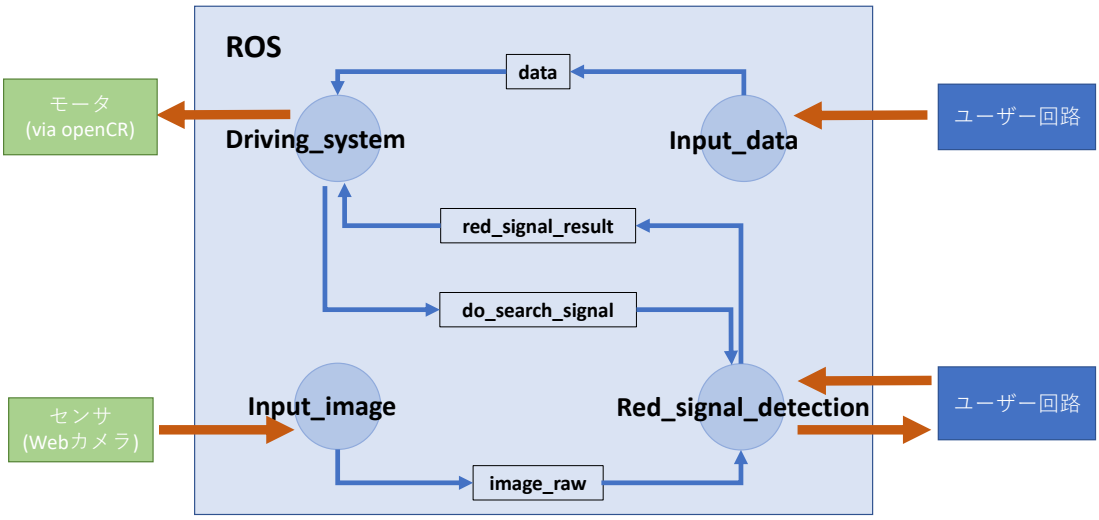

図 5: 作成したシステムの ROS ノードの構成

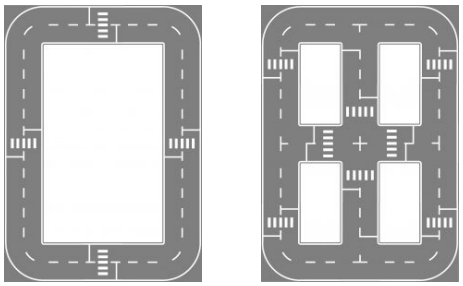

図 6: Competition courses

よって評価される．

（1） 道路をはみ出さずに走る
（2） 指示通りの道順で道路を走る
（3） 信号機の認識および停止
（4） 横断歩道の認識
（5） 障害物の認識および回避
（6） 人の認識および停止

自動車の開発条件として主に以下の制限がある．

- 外部との通信の禁止
- アルゴリズムは FPGA 内に実装
- 100Wh 未満のバッテリーでの動作
- CCD/CMOS カメラ以外のセンサ類の仕様は大幅減点

我々が開発した ZytleBot に実装した機能は以下の通りである．

（1） 道路を道なりに走行する
（2） 交差点であらかじめ設定した方角へ曲がる
（3） 信号機の認識および停止
（4） 障害物の認識および回避

競技規約を満たすために，処理はすべて FPGA ボード内の演算装置のみで行った．FPGA は Pcam から道路上の画像情報取得および信号機検出のための画像処理の一部に用いている．また開発においては Gazebo シミュレータ, catkin_make および rosbag といった ROS ツールを用い，FPGA を含むノードでもこれらが利用できることを確認した．

4.2 ノード構成

作成したシステムのノード構成を図 5 に示す．
各ノードの構成は次の通りである．

- Input_data

FPGA 上の IP からデバイスドライバを介して画像データを取得する．取得はポーリングで行い，システム全体の CPU 使用割合が適切になるように周期を調整する．取得したデータは非圧縮で出版する．

- Input_image

Web カメラから画像データを取得する．取得はポーリングで行う．取得したデータは非圧縮で出版する．

- Red_signal_detection

Web カメラから取得した画像データを用いて，機械学習のアルゴリズムのひとつであるランダムフォレストによる赤信号検出を行う．ランダムフォレストでの検出では，あらかじめ学習させた枠にリサイズし，各枠に対して RGB 特徴量，HSV 特徴量およびグレースケール画像からの HOG 特徴量の計算を行い，学習結果と比較することでその枠内に赤信号があるかどうかを判別する．いずれかの枠で赤信号が検出された場合，赤信号が存在することを red_signal_detect トピックを介して driving_system に送信する．

本ケースでは，十分な精度を求めるため，おおよそ 300 枠前後計算する．それぞれの計算は独立しているため並列計算可能が可能である．必要な演算処理のうち，HOG の計算を FPGA に行わせることによって CPU の負荷を FPGA にオフロードすることができる．さらに，FPGA-CPU 間の通信にかかる時間を含めても，CPU のみで処理する場合よりも高速化された．

また，処理を軽減するために信号を検出する必要がある場合のみ実行するように，driving_system で赤信号検出が必要かを決定している．トピック:do_signal_search から命令を購読し，値が 0 である場合は画像処理を停止する．

- driving_system

data トピックから画像を購読し，購読処理を行った際にコールバックで処理を開始する．取得したデータおよび赤信号の有無から物体認識等の認知を行い，ロボットの次の動作のための判断を行う．また同時に，自己位置や車体の状態を更新する．判断の結果，信号を認識する必要がない場合は do_signal_search を介して red_signal_detection の処理を停止させるよう命令を送信する．

画像処理にはオープンソースライブラリである openCV [11] を用いる．判断結果によりロボットを制御するための命令を openCR へ送信する．openCR は Arduino 準拠のマイクロコントローラである．TurtleBot3 に標準搭載されており，openCR を介したモータ操作についてはコンポーネントとして公開されている．

5. おわりに

本稿では，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を提案し，有用性について述べた．FPGA と ROS を統合したプラットフォームを利用することによって，ロボット開発における ROS のメリットを享受しつつ，FPGA による低消費電力化および処理性能向上を実現することが可能である．

提案したプラットフォームの課題として，対応している FPGA ボートが限られている，現在対応している FPGA ボードの CPU の処理性能の不足により実用レベルで動作させられる ROS パッケージが少ないといった問題が挙げられる．今後の方針として，より多くの FPGA ボードに対応する，組み込みシステムに適した ROS パッケージの開発および公開されている ROS パッケージの実装による検証を行う．

謝辞

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

文　献

[1] Open Source Robotics Foundation, http://wiki.ros.org/
[2] Merrick, P., Allen, S. and Lapp, J.: XML remote procedure call (XML-RPC) (2006). US Patent 7,028,312.
[3] K. Yamashina, T. Ohkawa, K. Ootsu, T. Yokota, "Proposal of ROS-compliant FPGA Component for Low-Power Robotic Systems -case study on image processing application-", International Symposium on Foundations and Practice of Security The Computing Research Repository, 2015.
[4] Y.Ishida, T.Morie, H.TamukohA, "Hardware Accelerated Robot Middleware Package for Intelligent Processing on Robots", IEEE International Symposium on Circuits and Systems (ISCAS), 2018
[5] Xilinx 社，Zynq-7000 All Programmable SoC テクニカル リファレンス マニュアル
[6] ROBOTIS TurtleBot3, http://www.robotis.us/turtlebot-3/
[7] PetaLinux ツール資料: リファレンス ガイド (UG1144), Xilinx 社
[8] Gazebo, http://gazebosim.org/
[9] CMake, https://cmake.org/
[10] FPT2018 FPGA Design Competition, https://wwp.shizuoka.ac.jp/fpt-design-contest/
[11] OpenCV library, https://opencv.org/