FPGA 処理を ROS コンポーネント化する自動設計環境
Automatic Design Environment for Componentization of an FPGA Processing in ROS

宇都宮大学大学院工学研究科　○ 山科和史，木村仁美，大川猛，大津金光，横田隆史
Graduate School of Engineering, Utsunomiya University
Kazushi Yamashina, Hitomi Kimura, Takeshi Ohkawa, Kanemitsu Ootsu and Takashi Yokota

Abstract An autonomous mobile robot which behaves according to its environment requires high perfomance information processing. However, it is difficult for sequential software processing to satisfy the performance requirement due to battery operataion. We focus on an FPGA which can realize optimized palallel processing at low power. On the other hand, FPGAs can not be employed in robot development projects since the development of an optimized circuit on an FPGA is difficut and time consuming. So, we have proposed ROS-compliant FPGA component for easy integration of an FPGA into a robot system. In this paper, we propose an automatic design environment named cReComp for componentization of an FPGA processing in ROS. An experimental result shows that a software developer without an experience of an FPGA can implement an ROS-compliant FPGA component within an hour.

1　はじめに

災害救助ロボットや無人ドローンなどロボットへの要求として，周囲の状況を判断し，自律的に行動できることがあげられる．ロボットの自律制御には画像処理や膨大なセンサ入力に対する計算処理など非常に高性能な処理が求められる．その反面，自律制御を行うロボットは無線通信かつバッテリ駆動が望ましく駆動時間を確保する必要があるため，消費電力の大きい高性能なプロセッサを搭載できない．それゆえに，ロボット上の限られた電力において高性能な処理を実現する必要がある．

こうした要求に応えるため，電力あたりの処理性能が高い FPGA(Field Programmable Gate Array) を用いることで，電力制約下においても高性能な処理を実現可能である [1]．FPGA は任意のデジタル回路をプログラミングによって実現可能な LSI である．ソフトウェアの逐次的な処理に対して，FPGA では目的に特化した並列処理や制御処理をハードウェアの処理として実現することができるため，画像処理，ネットワーク処理や検索サーバなどに利用されている．しかし，一般的に FPGA の開発はハードウェア記述言語を用いた RTL(Register Transfer Level) による設計が必要であり，ソフトウェア開発に比べて開発にかかる時間が膨大である．そこで FPGA 上における開発コストの削減のために，高性能な処理が求められる部分のみをハードウェア化することによって，必要最低限の開発コストでありながらシステム全体の性能向上が図ることができるハードウェア/ソフトウェア協調設計が有効である．一方，ロボット開発はハードウェア/ソフトウェアに加えて，電気回路や機構制御など非常に多岐にわたる専門分野の知識が必要となるため，各開発者の能力において開発が可能な範囲を遥かに凌駕する [2] ため，開発コストが大きい．

そこで我々は FPGA をロボット開発へ容易に導入可能とするために ROS 準拠 FPGA コンポーネントを提案している [3]．ロボットの開発コスト削減の開発手法として，コンポーネント指向開発 [2] が強く認識されており，ROS(Robot Operating System) は現在，コンポーネント指向なロボットアプリケーション開発のためのソフトウェアプラットフォームとして世界的に普及しつつある．ROS はロボットアプリケーション同士の通信ライブラリとアプリケーション開発のためのビルドシステムを提供する．ROS 準拠 FPGA コンポーネントでは FPGA による処理とソフトウェアによる処理からなる協調システムを ROS のコンポーネントとして提供することによって，FPGA をロボットへ容易に導入可能とし，電力制約がある中でも高速な処理を実現する．

本稿では，ROS 準拠 FPGA コンポーネントの従来の開発フロー，問題点に着目し，設計生産性の向上のために FPGA 処理のコンポーネント化のための自動設計環境 cReComp(creator for Reconfigurable Comoponent) を提案する．また，cReComp を用いて ROS 準拠の FPGA コンポーネントを設計・開発を行った際，コンポーネントの生産性と処理性能を評価について示す．

2 ROS 準拠 FPGA コンポーネント

2.1 ROS の概要

ROS は OSRF (Open Source Robotics Foundation) によってオープンソースで公開されているコンポーネント指向のロボット向けソフトウェアプラットフォームである．ROS ではロボットアプリケーションの通信フレームワークと，アプリケーション開発のためのビルドシステムを提供する．ROS は主に UNIX OS において動作し，Ubuntu が公式にサポートされている．

ROS におけるアプリケーション同士の通信モデルは Publish(配信)/Subscribe(購読) メッセージング (図 1) である．各アプリケーションは `node` と称し，`Topic` という通信チャネルを介してメッセージ (データ) を送受信することによってデータ通信を行なう．

ROS のシステムの特徴として，`node` 同士は通信をする相手の状態を知る必要がなく，互いに疎であることがあげられる．それゆえに，システムへのアプリケーションの追加や脱退が容易に行うことができるため各アプリケーションの設計，デバッグが容易である．ROS 準拠 FPGA コンポーネントは ROS の通信モデルに準拠することで，FPGA のロボットへの容易な導入を図る．

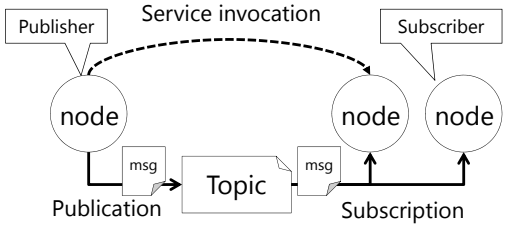

図 1: Publish/Subscribe メッセージング

2.2 ROS 準拠 FPGA コンポーネント

ROS 準拠 FPGA コンポーネントのシステム構成例を図 2 に示す．ハードウェア/ソフトウェア協調設計を実現するために，コンポーネントを実装するハードウェアプラットフォームとして Programmable SoC(ARM プロセッサ+FPGA) を用いる．各処理・制御をするハードウェアモジュールは FPGA/プロセッサ間のデータ通信が可能なソフトウェアインターフェイスを持ち，インターフェイスは Publish/Subscribe メッセージングが可能である．ハードウェアによる処理を ROS に準拠してソフトウェアでラッピングすることで他の `node` と同等に通信ができるため，ROS 準拠 FPGA コンポーネントは ROS で構築したシステムへ容易に導入ができる．

コンポーネントの粒度は ROS における「取り替えたい機能」に相当する．ROS において 1 つのソフトウェア群の単位は 1 種類の機能と同等の粒度で実装がされており，使用できるソフトウェア資源が非常に多く存在する．その反面ハードウェアは開発された回路の実装方法や入出力信号の規格統一化がなされていないため再利用性が低い．それゆえに，コンポーネントの粒度を 1 種類の機能としコンポーネント化することで，ハードウェア資源の再利用性の向上が可能である．一方，各ハードウェアモジュールに対してコンポーネント化を行うと，ソフトウェア/ハードウェア間の通信にかかる遅延時間が発生する．ハードウェアモジュール間において低遅延なデータ通信が必要な場合はコンポーネントを拡張し，ハードウェアモジュール間で直接ハードワイヤな接続を行うことで FPGA のナノ～マイクロ秒オーダーの処理を実現する．したがって，図 3 に示すように，それぞれの制御・処理を適切に配置し，要求されるオーダーの切り分けが重要である．要求性能が高い処理はハードウェア (FPGA) へ配置し，ソフトウェアでは，他の `node` との通信機能，ハードウェアへの制御コマンド，デバッグ出力などの機能を配置する．このような処理配置を行うことで，FPGA の高速かつ低遅延な処理とソフトウェアの扱いやすさを両立させる．

我々は，ROS の公式 Wiki において，すでに 2 つ ROS 準拠 FPGA コンポーネントを公開している [4].

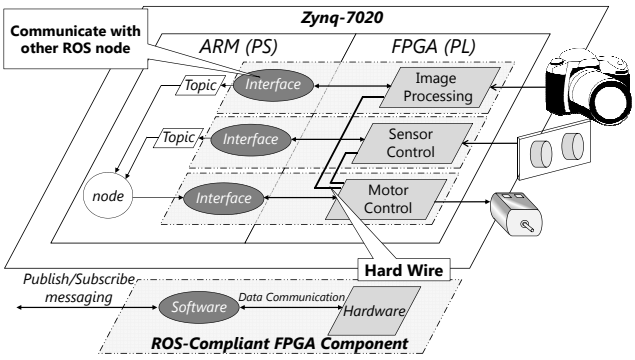

図 2: ROS 準拠 FPGA コンポーネントのシステム構成例

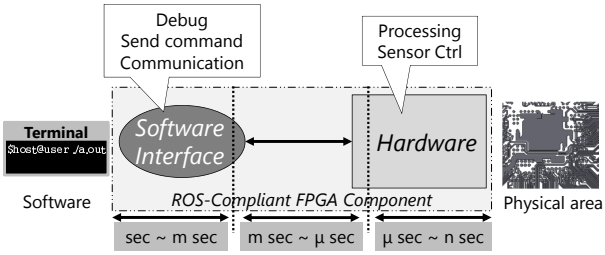

図 3: 要求されるオーダーに応じた処理の配置

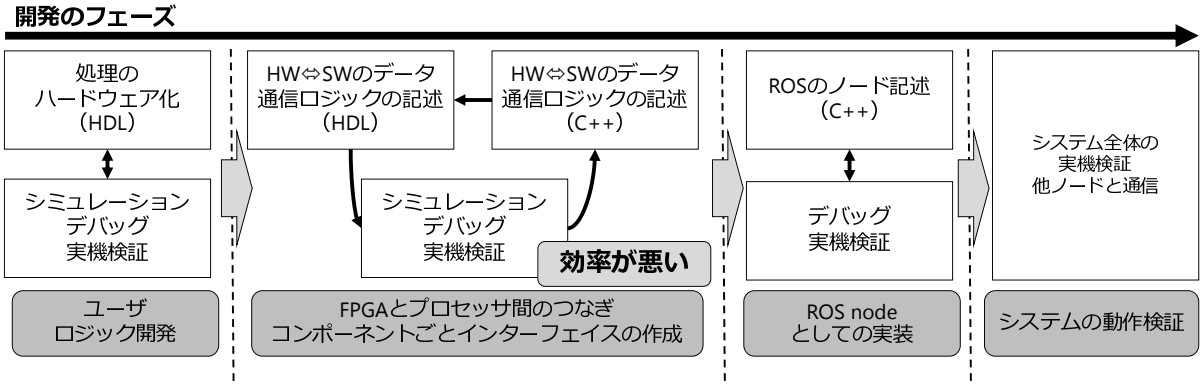

図 4: HW/SW 協調設計における従来のコンポーネント開発フロー

2.3 従来のコンポーネント開発における問題

HW/SW 協調設計における従来のコンポーネント開発フローを図 4 に示す．FPGA 処理のコンポーネント化では，始めにハードウェア化する任意の処理対象を決め，ハードウェア化を行う．この際の任意の処理ハードウェアを本稿ではユーザロジックとする．ユーザロジックへのデータ入力やデータ出力を使用する場合はプロセッサと FPGA が通信できるインターフェイスを作成する必要がある．インターフェイス作成には，FPGA におけるデータ受け渡しのための通信路の設計，通信トランザクションの考慮，回路の実装が伴う．ソフトウェア側では，FPGA 側に実装した通信路へのデータ通信の読み書きの回数やバイト数を考慮する必要がある．インターフェイスを作成し，FPGA 処理のコンポーネント化が完了したあとに，ROS の通信モデルに準拠した宣言を追加することで 1 つの ROS 準拠 FPGA コンポーネントが完成する．

FPGA 上における開発であるため，ユーザロジックの開発には HW/SW 間の通信のためのデバッグや動作検証が伴い，回路開発のコストは大きい．そのため，ユーザロジックをシステムへの導入のためのコンポーネント化にかかる開発コストも大きく，非常に効率が悪いという問題があった．

3 コンポーネント自動設計環境 cReComp の提案

2.3 節において提起した問題に着目し，コンポーネント開発効率の向上のために**コンポーネント自動設計環境 cReComp(creator for Reconfigurable Comoponent)** を提案する．なお，提案する設計環境は GitHub においてオープンソースで公開中である [5]．各コンポーネントにおいて異なる，ユーザロジックとソフトウェア/ハードウェア間通信のトランザクション (読み書きの回数，データサイズ) を定義することで，ユーザロジックに応じて FPGA 処理を自動的にコンポーネント化する．

3.1 cReComp のシステム生成モデル

cReComp におけるシステム生成モデルを図 5 に示す．cReComp における自動コンポーネント化ではコンポーネント化対象のユーザロジックと通信トランザクションを定義するための **Scrp(Specifier for cReComp) ファイル**を入力とする．開発者が記述するのはユーザロジック (HDL 記述ファイル) と Scrp ファイルであり，コンポーネント化に必要となるソフトウェア/ハードウェア間の通信路の回路記述やソフトウェア記述は必要ない．入力ファイルに対して自動生成するのはハードウェア/ソフトウェア間通信の制御ロジックが記述されたハードウェアと (HDL 記述ファイル) と FIFO バッファへアクセスするためのソフトウェア (C++ファイル) の 2 種類である．これら 2 つを総称してコンポーネント化の際に生成されるインターフェイスとする．

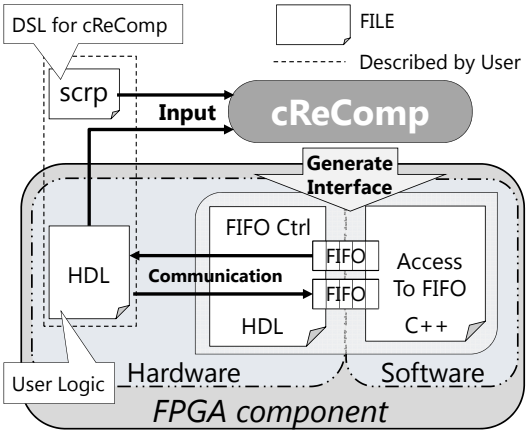

図 5: cReComp のシステム生成モデル

3.2 ARM-FPGA の通信方法

ROS 準拠 FPGA コンポーネントを Programmable SoC (ARM+FPGA) 上に実装するためには FPGA 上のユーザロジックに対して，プロセッサ上のソフトウェアからアクセスするための通信路が必要となる．具体的な ARM-FPGA の通信方法の検討のため，Xillybus 社が無償で公開している Programmable SoC の Zynq 向け開発プラットーフォームである Xillinux (試用版) を使用することとした．Xillinux は Zynq の ARM 上において動作する Linux OS(Ubuntu) であり，FPGA と ARM プロセッサ間の通信が可能な FIFO バッファに対してデバイスファイルとして read/write が可能である (図 6)．したがって，FIFO バッファに対してユーザロジックを接続し，任意の通信を行なう．FIFO バッファは 32bit と 8bit の 2 種類があり，データ転送用に 32bit，デバッグ用に 8bit を使用する．ROS 準拠 FPGA コンポーネントでは，FPGA 回路への Input/Output 用の 2 つの FIFO バッファを 32bit，8bit のそれぞれにおいて用いることで，ARM プロセッサと FPGA のそれぞれが独立に動作し，任意のタイミングで FIFO バッファへの read/write することを可能とした．

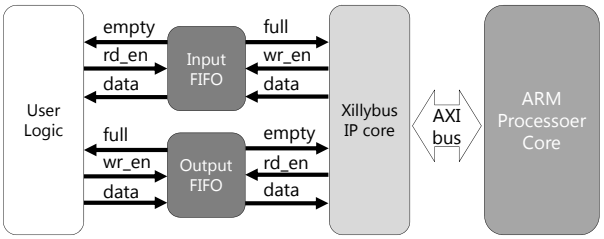

図 6: Xillinux における SW/HW 通信

3.3 コンポーネント化対象のユーザロジックモデルとステートマシン生成

cReComp においてコンポーネント化の対象となるユーザロジックモデルを図 7 に示す．cReComp においてコンポーネント化の対象となるユーザロジックは入力信号は組合せ論理，出力信号はレジスタ出力であるものを対象とする．そこで，対象のユーザロジックとプロセッサがデータのやり取りを行えるように cReComp は図 8 に示すステートマシンを生成する．プロセッサがユーザロジックへデータ入力をする際は `READY_RCV` ステートと `RCV_DATA` ステートにおいて Input 用の FIFO バッファからデータをレジスタに受け取り，ユーザロジックへデータを入力する．ユーザロジックがプロセッサへデータを出力する際は，`READY_SND` ステートと `SND_DATA` ステートにおいてユーザロジックのレジスタ出力の値を直

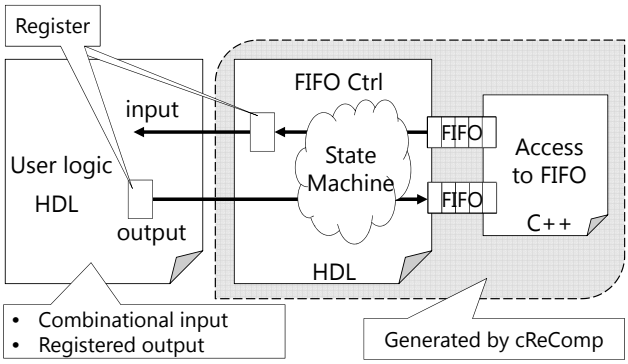

図 7: コンポーネント化対象のユーザロジックモデル

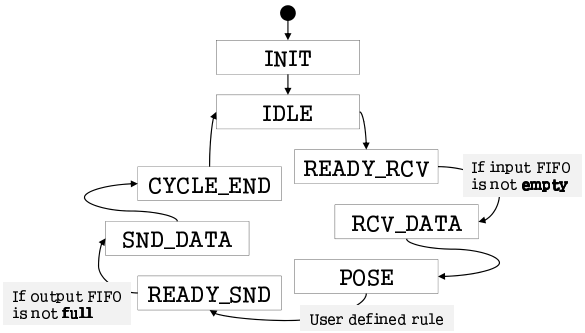

図 8: FIFO バッファ制御のためのステートマシン

接 Output 用の FIFO バッファへ出力する．

対象とするユーザロジックは現時点では直接的な HDL 記述によるものであるが，今後さまざまな形式に対応してく予定である．

3.4 Scrp ファイルによる SW/HW 通信トランザクションの定義

cReComp では Scrp ファイルを使用して，ソフトウェアとハードウェア間の通信のトランザクションを定義する．Scrp ファイルの例を図 9 に示す．テンプレートは cReComp を使用する際に生成することができる．各設定項目はフラグと称し，フラグに対応したそれぞれの定義を行うことによってコンポーネントの自動生成が可能である．各フラグについての定義可能な要素を表 1 に示す．なお，表 1 に示したフラグは 32bit FIFO 用のみのフラグであり，*が付加したフラグは 8bit FIFO にも対応するフラグがある．また，cReComp において Scrp ファイルテンプレートの生成時にユーザロジックの指定を行えば，対象の回路の入出力信号がテンプレート内に自動で記述される．したがって，ソフトウェア/ハードウェア間通信用の回路のために HDL 記述をしなくても，Scrp ファイル内において信号を自由に配線することが可能である．

```
module_name sensor_ctl
option_port{
            io,1,sig_out
}
use_fifo_32
make_32_alw{
            r,32,req_in
            w,32,sensor_data
}
r_cycle_32 1
rw_condition_32{
if(busy_flag && finish_flag)
}
w_cycle_32 1
wire_list{
            1,busy_flag
            1,finish_flag
}
sub_module_name sonic_sensor uut
assign_port sonic_sensor normal{
            req=req_in
            busy=busy_flag
            sig=sig_out
            finish=finish_flag
            out_data=sensor_data
}
end
```

図 9: Scrp ファイルの例

表 1: Scrp ファイルにおける各フラグの定義内容

フラグ	定義内容
module_name	HW インターフェイスの名前
option_port	任意のポート宣言
use_fifo_32*	FIFO を使用
make_32_alw*	ステート生成・関連信号宣言
r_cycle_32*	read する回数
rw_condition_32*	read/write の切り替え条件
w_cycle_32*	write する回数
reg_list	任意のレジスタ宣言
wire_list	任意の内部ワイヤ宣言

4 cReComp を用いたコンポーネント開発

4.1 cReComp によるコンポーネント開発

cReComp を用いたコンポーネントの開発フローを図 10 に示す．従来のコンポーネント開発フローでは，各ユーザロジックにおいて異なるプロセッサー FPGA 間通信を考慮し，コンポーネントごとにインターフェイスを作成する必要があった．cReComp を用いた場合，Scrp ファイルの記述と，ユーザロジックへの配線定義を行うことでコンポーネントを自動生成することが可能なので，開発工程を大幅に減少することが可能である．Scrp ファイルの記述後は，コマンドラインにおいて Scrp ファイルを指定してコンポーネントを生成 (`./cReComp FileName.scrp`) する．

cReComp を用いたコンポーネント開発について，cReComp の設計生産性と生成した ROS 準拠 FPGA コンポーネントの性能について 2 種類の評価を行った．

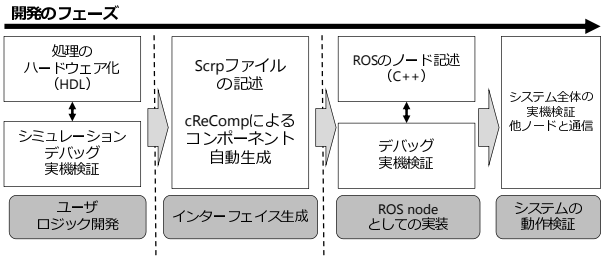

図 10: cReComp によるコンポーネント開発フロー

4.1.1 評価 1：cReComp の設計生産性の評価 超音波センサ制御

超音波センサの制御ハードウェアを cReComp によってコンポーネント化する被験者実験を行い，開発効率向上への影響について評価した．使用した超音波センサは Parallax 社製 PING Ultrasonic Distance Sensor である．超音波センサの制御回路を被験者へ配布し，著者が実験手順を記述した実験手順指導サイト [6] を利用し，1) cReComp によるユーザロジックのコンポーネント化，2) コンポーネントの動作検証，3) コンポーネントの ROS への導入・動作検証という順序で行なった．被験者は 6 人であり，被験者のハードウェア・ソフトウェアのに関する開発経験を以下に示す．

FPGA の開発経験 経験なし〜3 年

C++の開発経験 1〜6 年

Linux 使用経験 1〜3 年

実験において被験者全員が 3 時間以内に全工程を終えた．コンポーネント化に関する手順 1) において cReComp のインストール，Scrp ファイルの記述，コンポーネント化の大きく 3 種類の手順を行った．手順ごとの 5 段階難易度評価 (5 : 易しい，4 : やや易しい，3 : 普通，2 : やや難しい，1 : 難しい) と手順に要した時間の平均を図 11 に示す．ツールのインストール，コンポーネント化についての難易度はほぼ 5 という評価であり，手順に要した時間については 2 分程度で終了した．一方，Scrp ファイルの記述に関しては，難易度評価の平均値は 3.7(最高 5，最低 2) であり，手順に要した時間も 3 行程中最大の 17 分であった．コンポーネント化について，コマンドラインにおけるコマンド実行によってコンポーネントに関する一連のファイルが自動生成されるため，難易度は低いと考えられる．Scrp ファイルの記述においては評価値は 3.7 とやや易しいという結果にとどまった理由としては，Scrp ファイルのフラグの意味を把握するのに時間がかかったことが挙げられる．

cReComp を用いることで，FPGA 開発経験のない被

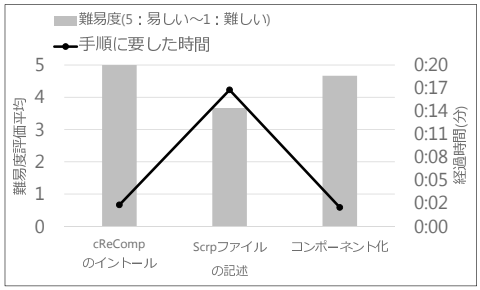

図 11: 難易度評価と手順に要した時間の平均

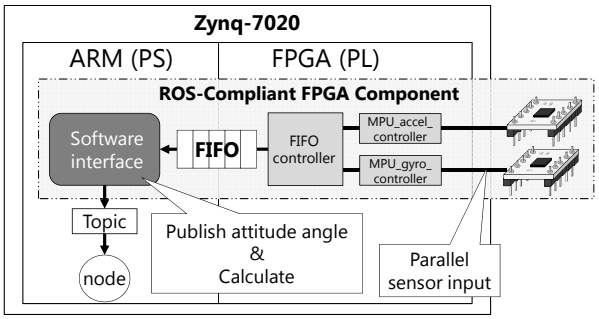

図 12: 姿勢角推定コンポーネント

験者においても 1 時間以内 (最大 42 分) にコンポーネント化を完了することができた．

4.1.2 評価 2：コンポーネントの性能評価 センサフュージョンによる姿勢角推定

センサフュージョンによる姿勢角推定を行うコンポーネントを開発し，コンポーネントの機能の妥当性と FPGA を用いることによる性能向上について評価した [7]．センサフュージョンとは複数かつ異なるセンサによって得られたデータに対して，各センサの測定における誤差の補完し合うことで，より信頼度の高い情報を得る手法である．

本事例における姿勢角推定ではジャイロセンサと加速度センサのセンサデータに対して相補フィルタを施すことによって信頼度の高い姿勢角を推定する．図 12 にコンポーネントのシステム像を示す．センサは，InvenSense 社製 MPU9250 を加速度センサ，ジャイロセンサとして 2 つもちいた．センサからの入力制御を行う回路である `MPU_accel_controller`，`MPU_gyro_controller` を cReComp を用いてコンポーネント化し，ソフトウェアインターフェイスを ROS における通信機能を実装しコンポーネント化した．FPGA において，2 つのセンサからの入力制御を並列に行うハードウェアによって，センサデータ取得の遅延を削減することによって高速化を図っており，1 つのセンサによって逐次的にソフトウェア処理する場合 (223 μ s) に比べ 1.85 倍の高速化 (120 μ s) を達成し，ROS における動作も確認した．

5 おわりに

本稿では，FPGA を容易にロボットシステムへ導入するための ROS 準拠 FPGA コンポーネントを紹介した．また，従来のコンポーネントの開発効率に着目し，コンポーネント自動設計環境 cReComp を提案した．評価 1 では cReComp を用いることで，FPGA 開発経験のない被験者においても 1 時間以内にコンポーネント化を完了できることが確認できた．さらに，評価 2 ではコンポーネントの導入をした際にソフトウェアのみのシステムと比較して，1.85 倍の高速化を達成した．

今後の予定としては，cReComp が現在対応するユーザロジックは直接的な HDL 記述によるものであるが，今後さまざまな形式に対応してくことがあげられる．

謝辞： 本研究は総務省 SCOPE(No.0159-0112) の支援により行われた．

参考文献

[1] 佐藤一輝，バートルスレンバルス，関根優年：”FPGA アレイを用いて TFlops を目指したボアソン方程式演算回路の実装と評価”，電子情報通信学会技術研究報告，VLD2008-94，CPSY2008-56，RECONF2008-58，2009．

[2] 日本ロボット学会 (編)：”ロボットテクノロジー”，オーム社, 2011.

[3] Kazushi Yamashina, et al. : ”Proposal of ROS-compliant FPGA Component for Low-Power Robotic Systems - case study on image processing application -”, Proceedings of 2nd International Workshop on FPGAs for Software Programmers, FSP2015, pp.62-67, 2015.

[4] OpenReroc : https://github.com/Kumikomi/OpenReroc

[5] cReComp : https://github.com/kazuyamashi/cReComp.git

[6] 実験手順書：http://kazuyamashi.github.io/exp/

[7] 木村 仁美，山科 和史，大川 猛，大津 金光，横田 隆史：“FPGA コンポーネントを用いた水中ロボット制御の高速化”，情報処理学会 第 78 回全国大会 講演論文集，講演番号 2H-03，2016