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

信学技報
TECHNICAL REPORT OF IEICE.

FPGA処理をソフトウェアコンポーネント化する設計ツール cReCompの高機能化の検討

山科　和史†　大川　猛†　大津　金光†　横田　隆史†

† 宇都宮大学大学院工学研究科　〒321–8585 栃木県宇都宮市陽東 7–1–2
E-mail: †kazushi@virgo.is.utsunomiya-u.ac.jp

あらまし　我々は，開発コストの高い FPGA をロボットシステムへ容易に導入するために，ROS 準拠 FPGA コンポーネントを提案している．ROS はロボットのソフトウェアプラットフォームとして，近年主流になりつつあり，ロボット向けのアプリケーションのビルドシステムと，通信フレームワークを提供する．本稿では，簡易な設定記述を行うのみで FPGA と CPU を接続し，ROS に準拠した FPGA のソフトウェアコンポーネント生成が可能な，設計支援ツールの cReComp について述べる．さらに，cReComp のこれまでの開発経緯について述べ，さらなる高機能化について検討する．

キーワード　FPGA，ROS，ROS 準拠 FPGA コンポーネント，コンポーネント指向設計

Functional Improvement of cReComp Design Tool for Software-Component Generation of FPGA Processing

Kazushi YAMASHINA†, Takeshi OHKAWA†, Kanemitsu OOTSU†, and Takashi YOKOTA†

† Graduate School of Engineering, Utsunomiya University
Yoto 7–1–2, Utsunomiya, Tochigi, 321–8585 Japan
E-mail: †kazushi@virgo.is.utsunomiya-u.ac.jp

Key words　FPGA, ROS, ROS-compliant FPGA component, Component-oriented design

1. はじめに

近年，自律移動ロボットの高機能化に伴い，ロボットのための処理（自己位置推定・地図作成，各種センサを用いた計算処理など）をするためのハードウェアプラットフォームは高性能であることが求められる．一般的に自律移動ロボットは屋内，屋外問わず様々な環境における使用が想定される [1]．それゆえに自律移動ロボットは様々な環境において自由に移動するために無線通信かつバッテリ駆動が求められ，ロボットの消費電力は可能な限り小さいことが求められる．したがって，低消費電力のために，ロボットへの高性能なプロセッサを搭載せず，高性能な処理を実現する必要がある．

我々はロボットに電力制約がある中でも，高性能な処理を実現するためのハードウェアプラットフォームとして，FPGA（Field Programmable Gate Array）に注目している．FPGA は一般的な汎用マイクロプロセッサに比べ電力効率が良いため，電力制約がある中でも高性能な処理が可能である [2]．一般的に，FPGA は HDL（Hardware Description Language）を用いた RTL（Register Transfer Level）での開発となるので，ソフトウェアに比べて難しく開発コストが高い．一方，ロボットを構成する要素技術として情報処理，機械制御，電気回路など非常に多くの技術が挙げられる．それゆえにロボット開発において，全ての専門分野を把握するのは現実的ではなく，莫大な開発コストがかかる [3]．

現代のロボットは制御，認識，行動計画等の様々なレベルのソフトウェアにより構成されている．そのため，FPGA をロボットのハードウェアプラットフォームとして導入するためには，HW/SW 協調設計が必要不可欠である．HW/SW 協調設計では，システム全体において高性能な処理が求められる部分のみをハードウェア化（FPGA の回路化）し，他の部分はソフトウェアの処理とする．このような設計手法をとることで，必要最小限の開発コストにおいてシステム全体の処理性能の向上を図ることが可能である．それゆえに，FPGA の開発生産性を上げることが，ロボットへの FPGA 導入の容易性向上に大きく関わる．一方，ロボット開発においてはコンポーネント指向設計が，開発生産性を向上する手法として強く認識されてい

る [3]．コンポーネント指向設計では，ロボットの機能をソフトウェアコンポーネント化し，コードやライブラリの再利用性を上げることで，開発生産性の向上を行う．

ロボットへの FPGA の導入を容易にするためには，FPGA 上において HW/SW 協調設計をしたシステムを．ソフトウェアコンポーネントとする技術が必要である．そこで我々は ROS 準拠 FPGA コンポーネントを提案している [5]．ROS（Robot Operating System）はコンポーネント指向設計の一種であり，ロボット開発のソフトウェアプラットフォームとして世界的に主流となりつつある．ROS 準拠 FPGA コンポーネントでは，FPGA の処理とソフトウェアの処理からなる協調システムを，ROS のソフトウェアコンポーネントとして提供することにより，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，任意の FPGA の回路に応じて，最適な ROS 準拠 FPGA コンポーネント化を行うための設計支援ツール **cReComp** について述べる．また，cReComp のこれまでの開発過程を示し，さらなる高機能化のための検討を行う．

2. ROS 準拠 FPGA コンポーネント

2.1 コンポーネント指向型プラットフォーム：ROS

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ロボットのためのソフトウェアプラットフォームである [6]．ROS はコンポーネント指向開発の一種であり，ロボット開発のためのビルドシステムと，アプリケーション同士の通信フレームワークを提供する．また，ROS では画像処理や，カメラ入力処理，モータ制御などの機能が，それぞれ 1 つのソフトウェアパッケージ（ROS コンポーネント）としてオープンソースで公開されている．開発者が利用可能なソフトウェア資産は非常に多く存在し，2016 年 7 月現在，3000 以上のコンポーネントが登録されている [7]．それゆえに，ROS はロボット向けのソフトウェアプラットフォームとして，世界的に主流となりつつある．

ROS における通信方法は Publish：配信/Subscribe：購読（Pub/Sub）メッセージングと Service という 2 種類がある．ROS の通信モデルを図 1 に示す．Service は一般的なサーバ・クライアント方式（同期通信）である．これに対して，Pub/Sub メッセージングは非同期通信の一種である．ROS における Pub/Sub メッセージングでは，publisher（配信者）が，topic という通信チャネルへ message をキューイングする．message とは，開発者が任意のデータ構造を用いることが可能な ROS のデータ形式である．また，subscriber（購読者）は任意の topic に対して購読登録をして，topic が更新されるたびにメッセージを受信し，メッセージに応じた処理を行う．この際，publisher と subscriber は node と呼び，node は ROS におけるコンポーネントに任意の数を持たせることが可能である．それゆえに，各コンポーネントは publisher と subscriber のどちらにもなりうる．

ROS のシステムの特徴として，各コンポーネント同士が通信をする際は，通信相手のことを知る必要はなく，疎結合であることがあげられる．それゆえに各コンポーネントの単体デバッグが容易に行うことが可能であり，システムへのコンポーネントの参入・脱退も容易に行うことができる．

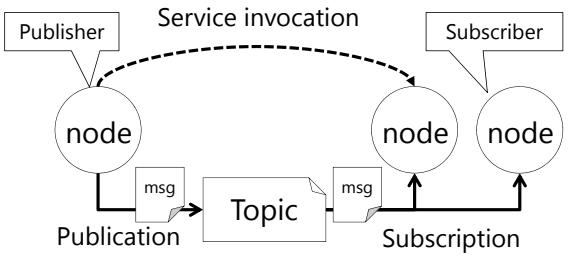

図 1　ROS の通信モデル

2.2 ROS 準拠 FPGA コンポーネントの概要

ROS 準拠 FPGA コンポーネントは，FPGA の処理を含む ROS コンポーネントである．本研究における FPGA の ROS システムへの導入とは，FPGA の回路をソフトウェアから操作できるということである．図 2 に ROS 準拠 FPGA コンポーネントのシステムモデルを示す．ROS 準拠 FPGA コンポーネントは，FPGA の処理回路と ROS のコンポーネントとの通信をするソフトウェア，また HW-SW 間の通信をするインターフェイス（I/F）によって構成する．HW/SW 間の通信を担う I/F は 2 種類からなり，一方は CPU と FPGA 間の通信路制御を行う回路（ハードウェア I/F）である．もう一方は，CPU より FPGA 上のハードウェア I/F へアクセスするためのソフトウェア I/F である．図 2 のようなシステムモデルとすることで，ROS 準拠 FPGA コンポーネント（FPGA 上の回路）は ROS における Pub/Sub メッセージングに対応し，ROS コンポーネントとデータ通信が可能である．それゆえに他の ROS コンポーネントの挙動によって，FPGA 内部の処理のアルゴリズムを切り替えたり，FPGA において処理した結果のフィードバックをしたりすることが容易であるため，ソフトウェアからハードウェアの操作ができる．また，ROS 準拠 FPGA コンポーネント（FPGA の処理）は他の ROS コンポーネントと同等に参入したり，脱退したりできるため，FPGA の導入も任意に行える．

2.1 項で述べた通り，ROS では開発者が利用可能なソフトウェア資産が非常に多く存在する．そのため，ROS の通信モデルに対応することで，ロボット開発の生産性が飛躍的に向上する．その反面，FPGA の回路は，実装方法や入出力インターフェイスの統一規格が明確にあるわけではなく，再利用性に乏しい．それゆえに，FPGA の回路を ROS 準拠 FPGA コンポーネント化することによって，FPGA の回路自体の再利用性と，ロボットシステムへの導入の容易性向上に大きく貢献する．

2.3 HW/SW 協調設計のためのプラットフォーム

ROS 準拠 FPGA コンポーネントにおいて，FPGA を用いる利点として一般の GPU や CPU を用いるよりも消費電力対性能比が高いことがあげられる [8]．FPGA の利点と開発生産性を共存させたまま，ROS 準拠 FPGA コンポーネントを実現するためには，1. 節で述べた通り，HW/SW 協調設計が有

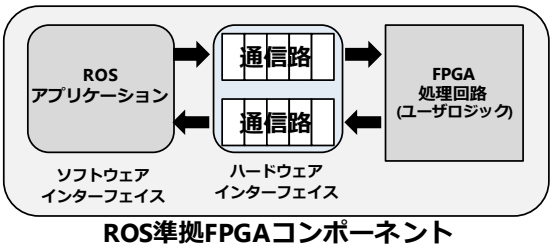

図 2　ROS 準拠 FPGA コンポーネントのシステムモデル

効である．そのため，ROS 準拠 FPGA コンポーネントを実現するハードウェアプラットフォームとして，組み込み向け CPU と FPGA を 1 枚のチップに搭載した，**Programmable SoC** を用いることとした．Programmable SoC の例としては，Xilinx 社の Zynq-7000，Altera 社の Cyclone V があげられる．Programmable SoC を用いる利点として，CPU と FPGA を同時に搭載しているがゆえに，FPGA の回路とソフトウェアをオンチップ上で扱うことができ，さらには，HW-SW 間のデータ通信が高速かつ低消費電力において可能である点があげられる．先行研究では，Sadari らが Xilinx 社の Zynq-7020 を用いて，画像のフィルタ処理の消費電力について評価した [9]．Sadari らの検証結果では，対象の処理において消費した電力は，チップ全体において 2W 以下にとどまり，この結果は一般的な高性能プロセッサに比べ，低消費電力であることがわかる．したがって，Programmable SoC は ROS 準拠 FPGA コンポーネントを実現するためのハードウェアプラットフォームとして適している．

Programmable SoC を用いた HW/SW 協調設計により，従来の RTL のみによる FPGA 開発と比べ，開発コストは相対的には減少する．しかし，このようなデバイス上において，HW/SW 協調設計を行う場合，CPU のバスシステムに関する深い理解や，プロセッサと FPGA のデータ通信の設計などが必要となり，依然として開発に対するハードルは高い．したがって，Programmable SoC 上に，アプリケーションレベルにおける設計のみで，HW/SW 協調システム，すなわち ROS 準拠 FPGA コンポーネントを実現するための設計環境が必要である．

3. コンポーネント設計支援ツール cReComp

3.1 cReComp の概要と HW の抽象化

Programmable SoC 上における，ROS 準拠 FPGA コンポーネントの開発を支援するツールとして，我々は **cReComp（creator for Reconfigurable Component）** を提案している [10] [11]．cReComp はコンポーネント化対象の回路（ユーザロジック）を，アプリケーションレベルにおける設計のみで，FPGA の回路をコンポーネント化する設計支援ツールである．また，cReComp における全ての機能は Python によって記述した．

ROS 準拠 FPGA コンポーネントは，2.2 項において述べたように，FPGA を，ROS システムへ容易に導入可能とする，という要求が存在する．すなわち，ROS のソフトウェアコンポーネントから FPGA の回路を操作，またはデータ通信ができるように，ハードウェアの抽象化を行う必要がある．そこで，本稿では，ROS 準拠 FPGA コンポーネント設計におけるハードウェア抽象化方式を，他のモデルと比較し，明確化する．ROS 準拠 FPGA コンポーネントにおける抽象化方式はコンポーネント指向インターフェイスである．コンポーネント指向 I/F と，これまで FPGA におけるハードウェア抽象化方式の比較を，表 1 に示す．

表 1　FPGA における抽象化方式の比較

抽象化方式	アクセス単位	適応システム 例	通信路
メモリマップド I/O ＋割り込み	メモリ アクセス	プロセッサ シングルスレッド	チップ内バス or ボード内バス
デバイスドライバ インターフェイス	関数 呼び出し	プロセッサ マルチスレッド	チップ内バス or ボード内バス
コンポーネント指向 インターフェイス	Pub/Sub メッセージング	ROS システム	チップ内バス or ボード内バス or イーサネット or 無線通信等

FPGA を用いた HW/SW 協調設計において，用いられる HW 抽象化方式は a) メモリマップドマップド I/O ＋割り込み，b) デバイスドライバ I/F である [12]．ロボット開発へ FPGA を導入するという前提において，a) で問題となるのが，データ書き込み関する操作の粒度である．a) の場合，メモリへ一度に読み書きが可能な単位としては，バイト単位・ワード単位にとどまり，なおかつ一度のメモリアクセスのみにおいてデータ通信が完了する場合は殆どない．それゆえに，一連の操作を開発者が明示的に記述しなければならず，開発生産性向上の妨げとなる．一方，b) においては，ハードウェアをファイルとして扱うことが可能となり，ファイルに対する open，read，write，close という操作によってデータ通信が可能である．しかし，b) において問題となるのが，ROS システムへの導入を行う際に，ROS システム上で扱うデータ形式から，デバイスファイルへ書き込むためのデータ形式へ変換するための記述が必要となることである．ROS を用いたロボット開発においては，コンポーネントを統合してロボットシステムを組み上げることに集中したいが，その際に a)，b) のような低いレイヤーの開発を行う必要があると，設計生産性を低下させる要因となる．

a)，b) に対して，c) コンポーネント I/F は，Pub/Sub メッセージングによってデータ通信を行うことで，FPGA 回路への操作を可能とする．すなわち，ROS システムにおけるソフトウェアコンポーネントと同等に，publish/subscribe することが，FPGA へのアクセスとなる．cReComp では，Programmable SoC 上においてコンポーネント指向 I/F の実現のための設計支援を行う．次項では，cReComp において，アプリケーションレベルの設計のみで，どのようにコンポーネント化を実現するかを述べる．

3.2 コンポーネント生成のモデル

cReComp における，アプリケーションレベルによるコンポーネント化を実現するための，コンポーネント生成モデルを図 3 に示す．cReComp では 2 つのファイルを入力することでコンポーネントの生成をする．1 つ目はコンポーネント化対象

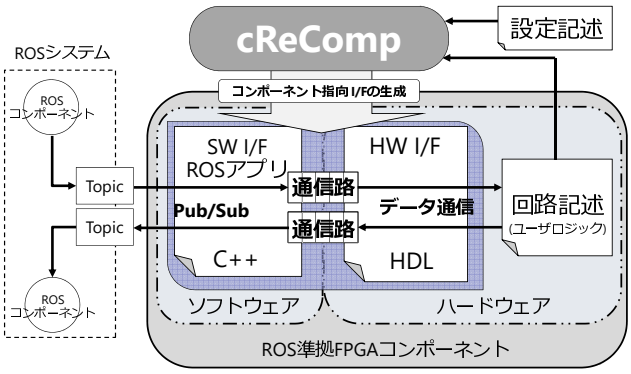

図 3　cReComp におけるコンポーネント生成モデル

となる FPGA の回路記述である．なお，cReComp ではコンポーネント化対象の回路記述はユーザロジックと呼ぶ．2 つ目はコンポーネント生成を行うための設定記述である．ROS 準拠 FPGA コンポーネントは Programmable SoC 上における HW/SW 協調システムであり，CPU（SW）と FPGA の回路（HW）の間にデータ通信を必要とする．cReComp における設定記述は，主に HW-SW 間の通信を行うためのルールに関する記述である．具体的には，適切なコンポーネント指向 I/F を生成するための記述であり，以下の要素である．

- 通信路のビット幅
- SW から HW へデータを受信する回数
- HW から SW へデータを送信する回数
- 受信と送信の切り替え条件
- 通信路とユーザロジックの接続

開発者が上記の通信に関するルールを記述することで，cReComp は設定内容に応じて最適なコンポーネント指向 I/F を生成する．したがって，開発者は，Programmable SoC 上における，プロセッサのバスシステムに関する知識や，データ通信のための回路設計を行わずとも，ハードウェアを抽象化できる．

コンポーネント指向 I/F 生成では主に 2 種類のファイルを出力する．1 つは通信路の制御ステートマシンを含む回路記述（HW I/F）である．ステートマシンは開発者が指定した通信のための設定記述に応じて動的に変化する．2 つ目は，HW I/F のデータを ROS の message 形式へ変換し，Pub/Sub メッセージングを行うソフトウェア（SW I/F）である．また，ROS では，ROS コンポーネントのビルドを行うための，ディレクトリ構造が規定されている．cReComp においても，生成した ROS 準拠 FPGA コンポーネントは同様のディレクトリ構造に準拠する．したがって，コンポーネントを利用する場合は即座にビルド可能であり，開発者自身のシステムへ容易に導入することが可能である．

4.　cReComp 高機能化の検討

4.1　開発初期：scrp を用いたコンポーネント化

cReComp ではコンポーネント化に際して，HW-SW 間通信のための設定記述が必要である．そこで我々は **scrp (specification for cReComp)** というオリジナルフォーマットを定義した．図 4 に開発初期の scrp の例を示す．開発初期の scrp

```
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 4　開発初期の scrp の例

の特徴としては，cReComp が提供する機能についての指定子を記述した後，それに付随するパラメータを設定するという点があげられる．例として，`r_cycle_32 1` という記述であれば，“SW からデータをリードする回数は 1 回” という意味となる．さらに，ユーザ定義の入出力信号を追加したい場合，scrp ファイル内によって宣言可能である．したがって，ユーザロジック内からの制御信号を，外部のセンサやモータに接続した場合でも柔軟に対応できる．

ユーザロジックに関しては，RTL（Verilog-HDL）によって記述されたファイルのみを対象とした．近年は，C 言語などプログラミング言語を用いて HDL を生成する技術である，HLS（High Level Synthesis）があり，HDL を用いらずとも回路記述が可能である．しかし，ロボットのモータ制御やセンサ制御などように，緻密な制御が求められる場合，HLS のような記述方法は適切ではなく，RTL を用いることが望ましい．したがって開発初期では，RTL 記述によるユーザロジックを優先すべき対象とした．

4.2　cReComp の評価実験

cReComp がコンポーネント開発の生産性にどのように影響するかを評価するため，6 人の被験者の協力を得て実験を行った（2016 年 2 月 15・16 日）[10]．この際の cReComp は前項で述べた開発初期の cReComp である．実験当時における被験者のハードウェア・ソフトウェア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

実験に用いた題材として超音波センサを取り上げ，センサの制御回路を cReComp によってコンポーネント化するという内容で行った．また，著者が記述した実験手順指導サイト [13] に即した順序で実験を進め，各手順を被験者が 5 段階 (5：易しい〜1：難しい) で，評価した．scrp を用いた設定記述に関する評価は 3.7 ポイントであり，被験者からは以下のような点について

指摘があった．

- 改行やインデントへの許容がない
- scrp の文法ミスが発見しづらい
- scrp の指定子の意味が分かりにくい

特に，“改行やインデントへの許容がない”，“scrp の文法ミスが発見しづらい” という指摘から，scrp の文法をチェックするパーサ機能の低さや，scrp の指定子における設定内容が不明瞭であるという問題が明らかとなった．また，scrp 記述（27 行，385 文字）において，被験者全体の平均所要時間として 17 分を要しており，設定記述の容易性や各設定項目の意味の明確化は，開発生産性に大きく関わっていることがわかる．

4.3 言語内 DSL への対応と scrp パーサの強化

cReComp における設定記述の容易性向上のため，主に 2 つの変更を加えた．1 つ目は Python を用いた言語内 DSL による設定記述への対応である．開発者は cReComp を使用する場合，設定記述方法として，scrp/Python の 2 種類から，任意のフォーマットを選択することが可能である．Python による設定記述は，cReComp が提供するライブラリを用いて行う．図 5 に Python を用いた言語内 DSL による設定記述例を示す．コンポーネント化に関する設定内容はライブラリで定義した，`Commponent` クラスのオブジェクトへ格納する．Python による記述の際は基本的に Component クラスのオブジェクトへ設定項目を，`add_*()` というメソッドによって追加する．例として，`add_input()` や `add_reg()` は，ユーザ定義の信号の追加を示している．このように，開発者はオブジェクトに対して，メソッドを呼び出すことで，コンポーネント化のための設定記述を容易に行うことができる．

```
cp = cp.Component("sensor_ctl")

cp.add_input("clk",1)
cp.add_input("rst",1)
cp.add_inout("sig_out",1)
cp.add_wire("finish_flag",1)
cp.add_wire("busy_flag",1)

fifo_32 = com.Xillybus_fifo(1,1,"finish_flag && busy_flag != 0", 32)
cp.add_reg("dummy",31)
cp.add_reg("req_reg",1)
cp.add_wire("sensor_data",32)

fifo_32.assign("rcv","req_reg")
fifo_32.assign("rcv", "dummy")
fifo_32.assign("snd", "sensor_data")

sonic_sensor = Sonic_sensor("uut")
sonic_sensor.assign("clk","clk")
sonic_sensor.assign("rst","rst")
sonic_sensor.assign("req","req_reg")
sonic_sensor.assign("busy","busy_flag")
sonic_sensor.assign("sig","sig_out")
sonic_sensor.assign("finish","finish_flag")
sonic_sensor.assign("out_data","sensor_data")

cp.add_ul(sonic_sensor)
cp.add_com(fifo_32)

cp.componentize()
```

図 5　Python の言語内 DSL による設定記述例

```
r_cycle_32 1
w_cycle_32 1
rw_condition_32{
if(1)
}
```

変更前

```
communication Xillybus{
  rcv_cycle 1,
  snd_cycle 1,
  condition “1“,
  fifo_width 32,
  rcv = req_reg,
  snd = data_out
}
```

変更後

図 6　scrp の文法構造の比較

2 つ目は scrp のパーサ機能の強化である．開発初期の cReComp では，scrp ファイルを 1 行ごとに読み込み，各指定子に応じて処理を決定していた．それゆえに，ファイル中に，予期せぬ改行や，空白文字があると，正常にコンポーネント化することができず，開発生産性の大きな妨げになっていた．それゆえに，scrp 専用のコンパイラを開発することによって，この問題を解決した．またコンパイラ開発にあたり，scrp の指定子や文法構造を見直し，改善した．図 6 に scrp の文法構造の比較を示す．変更前は，指定子の後にパラメータを記述するのみであり，設定内容が不明瞭であった．変更後では，設定内容に関する指定子（`communication`）の後に`{}`を用いて，ブロック構造をとることによって，どの指定子がどんな機能に関する設定項目なのか，明確に表せるように改善を行った．

また，cReComp は各記述方法に対して，テンプレート生成機能を持つ．さらに，テンプレート生成時，ユーザロジックのファイルパスを指定することでユーザロジックへ信号配線するための記述が自動生成される．ユーザロジックの解析には高前田氏の veriloggen [14] を使用した．ユーザロジックの持つ入出力ポートの情報はテンプレート生成時に自動的に反映されるので，開発者の記述量は最小限となる．

4.4 今後対応するべき機能

ロボットにおいて，モータやセンサの制御のほかにも，画像処理やセンサ値からの計算処理（フィルタなど）に関しても，高い処理性能が求められる．画像処理や計算処理などのアルゴリズムが複雑な処理は，RTL によるハードウェア化をすると大変大きな開発コストがかかり，ロボット開発に導入するのは現実的ではない．したがって，アルゴリズムが複雑であり，スループットが重要とされる処理には，HLS によるハードウェア化が適している．しかし，現状の cReComp では RTL 記述によるユーザロジックのみに対応している．それゆえに，今後の FPGA を用いたロボット開発において，HLS によって生成したユーザロジックへの対応は必須と考えられる．したがって，本稿では，HLS によって生成したユーザロジックに対応するための初期検討として，現状の cReComp を用いてコンポーネント化を試みた．

ユーザロジックとして，Xilinx 社の HLS ツールである，Vivado HLS (14.04) のサンプルプロジェクト `fir_srrc` を用いた．`fir_srrc` は SSRC フィルタ（Square Root Raised Cosine) の回路記述であり，C++言語によってアルゴリズムを実現している．Vivado HLS では C++言語における関数の定義が，FPGA の回路のインターフェイス定義となる. すなわち，C++の関数の引数の型は，HLS によって生成される HW の入出力ポートに大きく影響する．図 7 に `fir_srrc` おける SRRC フィルタの関数定義と生成された HW の入出力ポートを示す．C++言語における引数の型（`din_t`，`dout_t`）は，Vivado HLS によって提供される独自の型であり，実態は入出力 FIFO である．

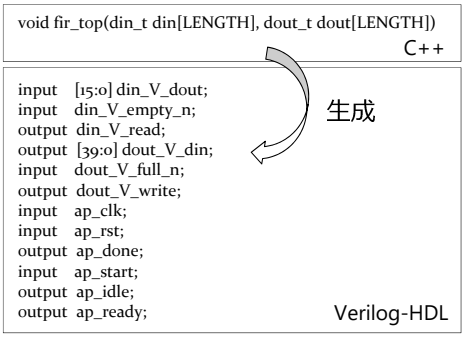

図 7　SRRC フィルタの関数定義と生成された HW の入出力ポート

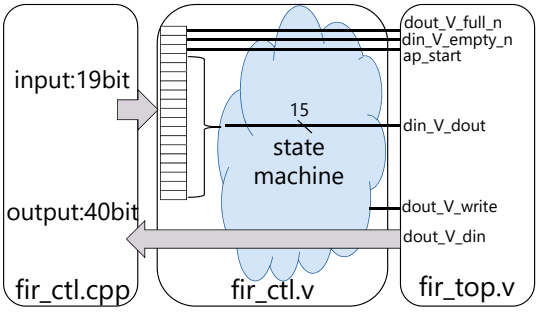

図 8　SRRC フィルタコンポーネントのシステム構成

FPGA 上の回路において，`din_V_empty_n` や `dout_V_full_n` を制御することによって，回路への入出力が可能である．また，`ap_start` や `ap_ready` などの信号は SSRC フィルタの処理に関する操作（処理の開始，処理の完了）信号である．以上の回路の機能を前提として，cReComp によってコンポーネント化を試みた．図 8 に生成したコンポーネントのシステム構成を示す．ユーザロジック HW-SW 通信のための通信路は，入力/出力それぞれ 19bit/40bit のビット幅である．特に入力に関しては，先頭から 16 ビット目までをデータポート，残り 3 ポートを制御用のポートとした．フィルタの処理は，データを決められた回数（10 回）ソフトウェアから入力すると処理がスタートし，処理が完了すと，ステートマシンによって自動的に SW へ処理結果が送信される．

生成したコンポーネントを Zynq-7020 を搭載した Avnet 社製 Zedboard を用いて動作確認を行った．cReComp によって生成したのは，`fir_ctl.v`（通信と制御のための HW）と `fir_ctl.cpp`（データ入出力のための SW）である．現状の cReComp によって生成したコンポーネントは，微小なコード修正を行う必要があったが，今後，cReComp によって生成するコードの改善により，HLS への対応も十分可能だと考えられる．また Zedboard 上において動作確認をした際のハードウェア量は，Flip-Flop：226（全 106400 中），LUT：323（全 53200 中）であった．

5. おわりに

本稿では，ロボット向けに FPGA の処理をコンポーネント化するツール，cReComp の高機能化について検討した．cReComp では，コンポーネント指向インターフェイスによってハードウェアを抽象化することで，アプリケーションレベルにおけるコンポーネント設計が可能であり，ロボットシステムにおける HW/SW 協調設計の生産性向上に貢献する．また，HLS によって生成したユーザロジックに対するコンポーネント化を行うため，cReComp におけるコード生成の再検討が今後の課題として挙げられる．なお，本稿にて述べた cReComp は，Github においてオープンソースソフトウェアとして公開中である [15].

謝辞　本研究開発は，総務省 SCOPE（受付番号 152103014）の委託を受けたものです．また，cReComp ワークショップの開催に，ご協力いただきました高瀬英希先生，veriloggen を使用するにあたり，助言をいただいた高前田伸也先生ありがとうございました．

文　献

[1] 金出武雄, “米国における自律移動ロボット研究の動向,” 日本ロボット学会誌, pp.376-383, 1987.
[2] 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, pp.19-24, 2009.
[3] 日本ロボット学会 (編) : “ロボットテクノロジー,” オーム社, 2011.
[4] Schirner, G., Gerstlauer, A., and DÃűmer, R., “System-level development of embedded software,” In Proceedings of the 2010 Asia and South Pacific Design Automation Conference (pp. 903-909). IEEE Press., 2010.
[5] Kazushi Yamashina, Takeshi Ohkawa, Kanemitsu Ootsu and Takashi Yokota, “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.
[6] http://www.osrfoundation.org/
[7] http://www.ros.org/is-ros-for-me/
[8] 谷田英生, 福井啓, 吉田浩章, 藤田昌宏 : “FPGA と GPGPU を利用した津波伝搬シミュレーションの高速化・高効率化”, 情報処理学会研究報告, Vol.2012-ARC-200, No.3, pp.1-8, 2012.
[9] Sadri, M., Weis, C., Wehn, N., and Benini, L : “Energy and performance exploration of accelerator coherency port using Xilinx ZYNQ”, In Proceedings of the 10th FPGAworld Conference, FPGAworld Conference 2013, pp.5-12, 2013.
[10] 山科 和史, 木村 仁美, 大川 猛, 大津 金光, 横田 隆史, “FPGA 処理を ROS コンポーネント化する自動設計環,” 第 60 回システム制御情報学会研究発表講演会予稿集, 2016.
[11] Kazushi Yamashina, Hitomi Kimura, Takeshi Ohkawa, Kanemitsu Ootsu and Takashi Yokota, “cReComp: Automated Design Tool for ROS-Compliant FPGA Component,” To appear in IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC-16), Sep.21-23, 2016.
[12] 大川猛, 高野創司, 植竹大地, 横田隆史, 大津金光, 馬場敬信, “ソフトウェアから FPGA を容易に扱うための分散オブジェクトプラットフォーム,” 第 54 回プログラミング・シンポジウム予稿集, pp.127-136, 2013.
[13] https://kazuyamashi.github.io/exp
[14] 高前田 伸也, “Python を用いた高水準ハードウェア設計環境の検討,” 電子情報通信学会研究会報告, RECONF2015-36, pp.21-26, 2015.
[15] https://github.com/kazuyamashi/cReComp.git