Vol.2018-ARC-230 No.3
Vol.2018-SLDM-183 No.3
Vol.2018-EMB-47 No.3
2018/3/7

高位合成を用いた画像処理ハードウェアのROS準拠FPGAコンポーネント設計フローの検討

菅田 悠平[1,a)]　大川 猛[1,b)]　大津 金光[1,c)]　横田 隆史[1,d)]

概要：ロボットビジョンにおいて，画像処理に FPGA (Field Programmable Gate Array) を用いることで，高速化・消費電力の削減が期待される．しかし，FPGA を用いたシステム設計は，開発生産性が低いことが課題であった．この課題に対して，我々はロボット開発に広く使われている ROS (Robot Operating System) に準拠した，FPGA コンポーネント技術を提案した．本稿では，FAST 特徴点検出を題材として，高位合成を用いた画像処理ハードウェアをコンポーネント化する際の設計フローの検討結果について述べる．

キーワード：FPGA，ROS，ロボット，高位合成，ハードウェア TCP/IP

Study of Design Flow of ROS Compliant FPGA Component of Image Processing Hardware by High-Level Synthesis

Sugata Yuhei[1,a)]　Ohkawa Takeshi[1,b)]　Ootsu Kanemitsu[1,c)]　Yokota Takashi[1,d)]

Abstract: In robot vision, using FPGA (Field Programmable Gate Array) for image processing is expected to speed up and reduce power consumption. However, system design using FPGA has a problem of low development productivity. In response to this problem, we proposed FPGA component technology compliant with ROS (Robot Operating System) widely used for robot development. In this paper, the design flow for componentizing the image processing hardware using high level synthesis is discussed, using FAST feature point detection as a case study.

Keywords: FPGA, ROS, Robot, High-Level Synthesis, Hardware TCP/IP,

1. はじめに

災害現場や日常生活の支援において，自律型ロボットの活躍が期待されている [1]．自律型ロボットは，画像処理やアクチュエータ駆動などの様々な処理・制御を行うソフトウェアから構成される．特に，カメラからの入力画像を基とした物体認識などの画像処理は，自律型ロボットにおけるビジョンシステムを実現するには欠かせない．一方，自律型ロボットの多くはバッテリ駆動であることから，消費電力の制約が大きい．そのため，高速なマイクロプロセッサや GPU (Graphics Processing Unit) をロボット本体に搭載し，画像処理を行うことは難しい [2]．

このような消費電力の制約がある中での処理基盤として，FPGA (Field Programmable Gate Array) が注目されている [2]．FPGA とは，ユーザがディジタル論理回路を自由にプログラミングすることが可能な LSI チップである．ハードウェア並列処理を FPGA に実装することで，ソフトウェア処理と比較して，高速化・消費電力の削減が期待される [3]．しかし，FPGA を用いたシステム設計は，開発生産性が低いことが課題である．FPGA に回路を設計する際，一般的にはクロック単位でレジスタに書き込む論理演算を行う RTL (Register Transfer Level) の設計を行わなければならない．近年は，C 言語，Java，Python などのソフトウェア言語からハードウェアを合成する高位合成ツールが研究開発されている [3][4]．しかし，FPGA の性能を引き出すには，FPGA の構造を把握した上での並列化指示子の適切な挿入などを行う必要がある．また，FPGA への

[1] 宇都宮大学大学院工学研究科
Graduate School of Engineering Utsunomiya University
7-1-2 Yoto, Utsunomiya, Tochigi, 321-8585, Japan
[a)] sugata@virgo.is.utsunomiya-u.ac.jp
[b)] ohkawa@is.utsunomiya-u.ac.jp
[c)] kim@is.utsunomiya-u.ac.jp
[d)] yokota@is.utsunomiya-u.ac.jp

Vol.2018-ARC-230 No.3
Vol.2018-SLDM-183 No.3
Vol.2018-EMB-47 No.3
2018/3/7

回路設計に加えて，回路を制御するためのソフトウェアと組み合わせる際にもハードウェア・ソフトウェア両方の知識が要求される．そのため，様々なソフトウェアから構成されるロボットシステムにおいて，FPGA を導入することは容易ではない．

この課題を解決するために，これまでに我々は，FPGA 上に実装した回路を ROS に準拠したコンポーネントとして扱う ROS 準拠 FPGA コンポーネント化手法を提案してきた [5]．ROS (Robot Operating System) は，ソフトウェアをコンポーネントとして扱う，コンポーネント指向設計に基づいたロボット開発のためのソフトウェアプラットフォームである．これまでにプログラマブル SoC を対象とした設計手法の検討を行い，簡単な設定ファイルと HDL 記述を入力として ROS 準拠 FPGA コンポーネントの SW 実装と HW 実装を出力する自動生成ツール cReComp を提案した [6]．一方, プログラマブル SoC は組込み向けの ARM プロセッサコアを用いて TCP/IP および ROS の Publish/Subscribe 通信を行うため，通信遅延時間が大きいことが課題であり，画像データを受け渡すような通信量が多いアプリケーションには適していないことも明らかとなっている [5]．そこで我々は，通信部分をハードウェア TCP/IP スタックを用いて高速化するハードウェア ROS 準拠 FPGA コンポーネントを提案した [8]．評価の結果，ROS 標準の画像メッセージ形式においても大幅な通信時間の削減を明らかにした [9]．しかし，ハードウェア ROS 準拠 FPGA コンポーネントの実装は HDL を用いて手作業で行っているのが現状である．そのため，高効率な開発フローの確立が必要である．

一般的に高位合成は，生成された性能・効率が HDL の実装に比べて，十分に最適化されてるとはいえない．しかし近年，画像処理に用いられる OpenCV ライブラリが高位合成による開発環境から容易に使用できるようになり，ライブラリ化によって性能の良い回路が合成可能になってきている [12]．これらの状況を鑑み，高位合成によって生成された画像処理回路をハードウェア ROS 準拠 FPGA コンポーネントにする手順を確立することにより，ロボットシステムに必要とされる画像処理回路を含む ROS コンポーネントを瞬時に作り上げることが可能になると期待される．

本稿では，FAST 特徴点検出を題材として，高位合成を用いた画像処理ハードウェアをコンポーネント化する際の設計フローの検討結果について述べる．

2. ROS 準拠 FPGA コンポーネント

2.1 ROS の概要

ロボットシステムにおけるソフトウェアは，従来は単体のソフトウェアとして開発され，特にリアルタイム性や信頼性に重点が置かれていた．しかし，画像認識技術などの発展に伴い，高度に知的なソフトウェアと厳密な制御を要するソフトウェアが混在し，複雑化している．この状況に対して ROS が提案された [13]．ROS は，複数の独立したソフトウェアプロセスが互いにトピックと呼ばれる論理的な通信路を通じてメッセージをやり取りする，Publish/Subscribe モデルに基づくロボットソフトウェア開発環境である．ROS は OSRF(Open Source Robotics Foundation) によってオープンソースで公開されており，主に UNIX OS 上で動作し，Ubuntu が公式にサポートされている．ROS では，計算や処理を行うプロセスをノードといい，画像処理・アクチュエータ制御などの複数のノードを構成することで高度に知的なロボット制御を実現する．Publish/Subscribe 通信の概要を**図 1** に示す．この通信では，各ノードはトピックを介して，非同期にメッセージをやり取りする．トピックに対してメッセージの配信を行うノードを Publisher，トピックを購読してメッセージを受信するノードを Subscriber という．メッセージは構造化されたデータであり，ユーザはメッセージを独自に定義するか，ROS で予め定義されている形式 (座標情報，画像やセンサ情報等) を使用することができる．Publish/Subscribe 通信は，サーバクライアント通信などと比較して，ノード間の結びつきが疎であるため，ノードの追加・修正・削除などが容易である．

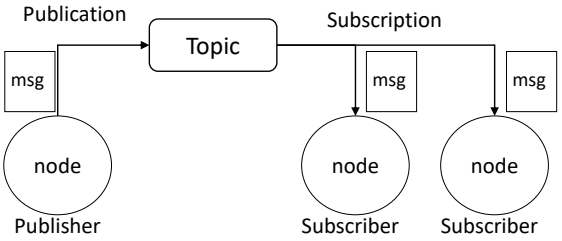

図 1 Publish/Subscribe メッセージング

このように ROS システムにおいて FPGA を導入することは多くのメリットを有すると考えられるが，一般に ROS システムへの FPGA 導入は進んでいないのが現状である．要因としては，FPGA 開発は通常 RTL 設計が必要であり，通常の手続き型ソフトウェア言語とは異なるため，一般のソフトウェアエンジニアが開発に参加することが困難であることが挙げられる．そのため，ROS システムにおいて FPGA を導入するための枠組みが必要であると考えられる．

2.2 ROS 準拠 FPGA コンポーネントの概要

我々は，FPGA の容易な ROS システムへの導入を目的として，ROS 準拠 FPGA コンポーネントを提案した [5]．ROS 準拠 FPGA コンポーネントは，入出力を行うための ROS ノード, FPGA に実装された処理回路から構成される．コンポーネントの入出力は，ROS の Publish/Subscribe 通信に準拠し, トピックへの購読・配信をすることで行う．そ

Vol.2018-ARC-230 No.3
Vol.2018-SLDM-183 No.3
Vol.2018-EMB-47 No.3
2018/3/7

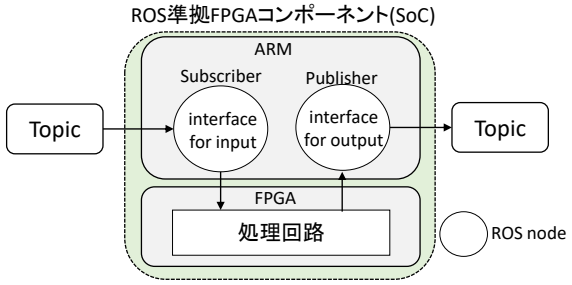

図 2 プログラマブル SoC 版の ROS 準拠 FPGA コンポーネント

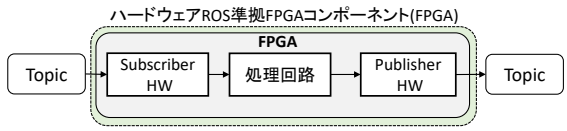

図 3 ハードウェア ROS 準拠 FPGA コンポーネント

のため，コンポーネント化された FPGA は，既存のソフトウェアシステムに変更を加えることなく ROS システムの一部として統合できる．我々は ROS 準拠 FPGA コンポーネントの実現形態として，二つの実装方式を提案した．2 つの実装方式を**図 2**，**図 3** に示す．

一つ目の提案 [5] では，コンポーネントを実装するプラットフォームとして，ARM プロセッサと FPGA を一つのチップに搭載した，プラグラマブル SoC (System On Chip) を使用した．この提案の目的は，ARM プロセッサ上で ROS ノードを動かし，高性能な回路部品 (IP) を FPGA 部分に実装することで，ワンチップで ROS ノードを実現することにある．チップ上で ARM プロセッサと FPGA が AXI-bus によって直結されていることで，高速な HW-SW 間の通信が可能である．ARM プロセッサ上において Linux OS (Ubuntu) および ROS が動作し，FPGA-ARM 間の通信は FIFO バッファを介して行う．

この提案においては，さらにコンポーネント生成自動化ツール cReComp の開発も行った [6]．このツールは，Verilog-HDL で書かれた FPGA 用のハードウェア記述と HW-SW 間の通信に関しての簡単な設定記述を入力とし，プログラマブル SoC 上で動作可能な ROS 準拠 FPGA コンポーネントのソフト・ハードのソースファイル一式を出力する．これにより，FPGA の経験が少ない開発者であっても，1 時間程度で ROS コンポーネントを開発できることが明らかとなった [6][7]．一方，この提案方式では通信性能が問題となった．先行研究では，FPGA 上に実装された画像処理回路をコンポーネント化し，ROS システム内でのコンポーネントの性能評価を行った [5]．この評価では，入力を行うノードが対象の ROS ノードに入力画像を送信してから，画像ラベリング処理を行い，出力ノードが処理結果を受信するまでの遅延時間を分析した．評価の結果，ラベリング処理の時間はソフトウェアと比較して 26 倍の速度向上を果たしたが，通信時間を含めたコンポーネント全体の処理時間は 1.7 倍の速度向上にとどまった．すなわち，ARM プロセッサで行う ROS ノードの Publish/Subscribe 通信がオーバーヘッドとなっていることが分かった．これは，ROS システム内で画像のような大きなメッセージをやり取りする場合に大きな問題となる．

この通信遅延の問題を解決するために，二つ目として，ハードウェア ROS 準拠 FPGA コンポーネントを提案した [8]．この提案手法では，ROS の Publish/Subscribe 通信をハードウェアで行うために，FPGA 上で利用可能なハードウェア TCP/IP スタックを用い，ROS プロトコルをハードウェアで解釈することで，遅延時間を最小化することを目指した．ハードウェア TCP/IP スタックとして SiTCP[10] を使用して，提案のハードウェア ROS 準拠 FPGA コンポーネントについて評価を行った [9]．評価のため，ギガビットイーサネットを経由して，PC と FPGA ボードの間で ROS の Publish/Subscribe 通信を行うシステムを構築した．評価の結果，ARM 上で動作させた通常のソフトウェアが画像 (RGB 1920x1080，6M バイト) を受信・送信するための遅延時間が 290ms であるのに対し，提案するハードウェア ROS 準拠 FPGA コンポーネントは 175ms であった．

すなわち，第 1 の提案のプログラマブル SoC を用いた ROS ノードと比較して，第 2 の提案のハードウェア通信による ROS ノードにおいては，Publish/Subscribe 通信の遅延時間を大幅に削減可能であることが分かった．しかし，ハードウェア ROS 準拠 FPGA コンポーネントの開発は，現在 Verilog-HDL を用いて行っており，開発の生産性が低い問題がある．そこで，cReComp と同様のコンポーネント自動ツールの実現のため，ハードウェア ROS 準拠 FPGA コンポーネントの開発フローの検討を行うこととした．

3. ハードウェア ROS 準拠 FPGA コンポーネントの開発フローの検討

3.1 コンポーネント化対象の画像処理回路の設計方法

FPGA における画像処理回路の設計において，有望視されているのが Vivado HLS 等の高位合成ツールである．そのため，ハードウェア ROS 準拠 FPGA コンポーネントがコンポーネント化の対象とする FPGA 処理 (ユーザロジック) は，高位合成による画像処理を前提とする．高位合成ツール等で作成した回路をそのまま ROS 準拠 FPGA コンポーネントにすることができれば，コンポーネントの生産性が大幅に向上することが期待できる．ここでは，例として Xilinx 社の Vivado HLS ツールを使用した場合の画像処理回路の設計について考える．

Vivado HLS では，Vivado HLS ビデオライブラリが用意されており，これを用いると OpenCV の関数を利用して画像処理回路を開発することが容易である [11][12]．OpenCV

Vol.2018-ARC-230 No.3
Vol.2018-SLDM-183 No.3
Vol.2018-EMB-47 No.3
2018/3/7

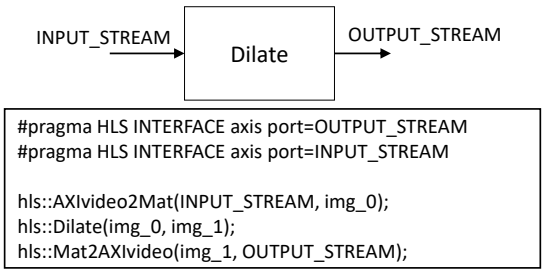

図 4 Vivado HLS ライブラリを使用した高位合成例

表 1 sensor_msg/image 型の定義

型	変数名
uint32	seq
time	stamp
string	frame_id
uint32	height
uint32	width
string	encoding
uint8	is_bigendian
uint32	step
uint8[]	data

関数の一部が高位合成に対応しており，手軽に高性能な画像処理を行うことができる．Vivado HLS を使用した画像処理回路の例を**図 4** に示す．AXIvideo2Mat, Mat2AXIvideo は，AXI4 ストリームと hls::Mat 型間の変換を行うインタフェース用のライブラリであり，ビデオライブラリを使用する際は基本的に AXI4 ストリーム形式で入出力を行う．hls::Mat 型を使用することで，OpenCV ライブラリに対応可能である．図 4 の hls::Dilate は，膨張処理を行う OpenCV の dilate 関数と対応する．

すなわち，コンポーネント化の対象となる画像処理回路の入出力は AXI4 ストリームの信号となる．ROS コンポーネントとして働くためには，外部から受信した ROS の画像メッセージを AXI4 ストリームの信号の形式に変換し，画像処理回路に入力する必要がある．また，画像処理回路の出力結果である AXI4 ストリームを ROS の画像メッセージ形式に変換する必要がある．

3.2 ROS メッセージと AXI4 ストリームの変換方法

ROS の画像メッセージ形式として，一般的に用いられるのが sensor_msg/image 型である．sensor_msg/image 型の定義を**表 1** に示す．このメッセージ型は，画像サイズ，フレーム番号，画像の形式などをもつメッセージ型であり，各ピクセルの値は data 配列に書かれる．

AXI4 ストリームプロトコルによるデータ転送は，TDATA，TVALID，TREADY の 3 つの信号で行われる．TVARID，TREADY が共に HIGH のときに，TDATA の値 (画像処理においてはピクセルデータ) が転送される．TDATA の値は画像のチャネル数 (グレイスケール，RGB) に応じて bit 幅が変化する．Vivado HLS ビデオライブラリを使用するときは，これらの基本信号に加え，TLAST，TUSER を使用する [11][12]．TLAST は画像の各行の最後のピクセルのときにアサートされ，TUSER はフレームの最初のピクセルの時にアサートされる．そのため，AXI4 ストリームプロトコルに変換を行うには，画像の縦横のサイズの値が必要となる．

以上のことから，ROS の画像メッセージ形式から AXI4 ストリーム形式への変換方法は，以下のシンプルな手順を踏めば良い．まず，ROS の画像メッセージ形式を解釈し，width，height の値，各ピクセルの値を記録する．次に，画像のピクセルデータのみを読み出し，width，height の値に応じて，適切に TLAST，TUSER 信号を使用し，AXI4 ストリームプロトコルに変換する．

一方，AXI4 ストリーム形式から ROS の画像メッセージ形式への変換は単純ではない．なぜならば，AXI4 ストリーム形式を基に，sensor_msg 型の変数全てを決めることができないからである．そのため，AXI4 ストリームから ROS の画像メッセージ形式を行うためには，予め encoding や is_bigendian 等のパラメータを設定する必要がある．

3.3 ハードウェア構造と開発フロー

これらの変換処理を C 言語を用いて記述し，高位合成によりハードウェア化するためのハードウェア構造と開発フローを検討した．ROS の画像メッセージ形式から AXI4 ストリーム形式に変換を行う Subscriber HW のブロック図を**図 5**，AXI4 ストリーム形式から ROS のメッセージ形式に変換を行う Publisher HW を**図 6** に示す．Subscriber HW において，TCP パケットとして受信する ROS のメッセージは，ハードウェア TCP/IP スタックである SiTCP[10] によって処理され，ROS メッセージのデータが FIFO に書き込まれる．Subscriber Logic は，ROS のメッセージを FIFO から読みだし，width,hegiht を記録し，ピクセルデータのみを後段の FIFO に書きこむ．FIFO2AXI は，FIFO からピクセルデータを読みだし，TLAST,TUSER 信号を width,hegiht に応じてアサートすることで，AXI4 ストリーム形式に変換する．Publisher HW において，AXI2FIFO は，AXI4 ストリーム形式で出力される処理結果を FIFO に書きこむ．次に Publisher Logic が FIFO に書き込まれたピクセルデータを ROS のメッセージ定義に従い変換する．

このような HW 構造とした時，コンポーネント開発手順は**図 7** のようになる．まず，OpenCV を用いて画像処理を行うソフトウェアを作成し，Vivado HLS ビデオライブラリを使用して，高位合成を行う．次に画像処理に応じて encode, is_bigendian, 画像のチャネル数などのパラメータを設定する．Vivado HLS を使用し，パラメータに応じて，Publisher HW，Subscriber HW を合成する．つまり，OpenCV で作成した画像処理記述と ROS メッセー

Vol.2018-ARC-230 No.3
Vol.2018-SLDM-183 No.3
Vol.2018-EMB-47 No.3
2018/3/7

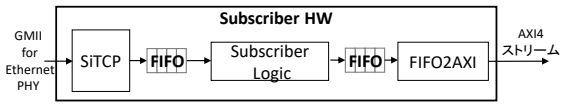

図 5 subscriber HW の概要

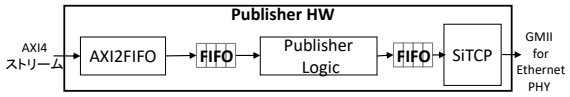

図 6 publisher HW の概要

図 7 コンポーネント開発手順

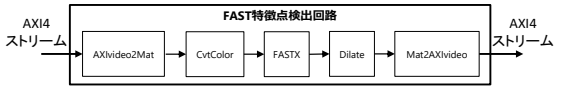

図 8 FAST 特徴点検出回路

```
void fifo2axi (    volatile int8_t *din_fifo2axi,
                   AXI_STREAM& IMAGE_STREAM)
{
   #pragma HLS INTERFACE axis  port=IMAGE_STREAM
   #pragma HLS INTERFACE ap_fifo  port=din_fifo2axi
   ap_axiu<32,1,1,1> lap;
   int height=480, width=640, i, j;
   int pixel[4]={0,0,0,0};

   for(i=0;i<height;i++){
     for(j=0;j<width;j++){
          pixel[0] = *din_fifo2axi;
          pixel[1] = *din_fifo2axi;
          pixel[2] = *din_fifo2axi;
          pixel[3] = *din_fifo2axi;
          lap.data=(0xff000000 &(pixel[3]<<24))
                   +(0x00ff0000 &(pixel[2]<<16))
                   +(0x0000ff00 &(pixel[1]<<8))
                   +(0x000000ff & pixel[0]);
          if(j ==width-1)      lap.last=1;
             else   lap.last=0;
          if(i==0 && j==0)     lap.user = 1;
             else   lap.user =0;
          IMAGE_STREAM << lap;
 }}}
```

図 9 FIFO2AXI のソースコード

ジ-AXI4 ストリームの変換記述から，高位合成のみで，コンポーネント生成が可能となる．次節では，FAST 特徴点検出を題材として，実際にコンポーネント開発を行った事例について述べる．

4. FAST 特徴点検出回路のコンポーネント開発

4.1 FAST 特徴点検出回路

Xilinx 社のリファレンスデザイン [12] を参考に，FAST 特徴点検出回路を実装した．FAST 特徴点検出は，物体のコーナ検出用の画像処理である．実装した FAST 特徴点検出回路のブロック図を**図 8** に示す．この回路は AXI ストリーム形式の入力画像を AXIVideo2Mat により，hls::Mat 型に変換する．次に CvtColor の BGR2GRAY 変換によって，二値化を行う．FASTX にて検出された特徴点を Dilate によって膨張した後に，hls::Mat 型から AXI4 ストリームに変換し，結果を出力する．また，実装にあたって画像の横幅，縦幅は固定値で実装し，画像ピクセルのチャネル数は 4 チャネルで実装した．そのため，AXI4 ストリームの TDATA の値は 32bit 幅となった．

4.2 Subscriber HW および Publisher HW の高位合成による実装

Subscriber HW を構成する Subscriber Logic, FIFO2AXI は，前述の通り高位合成を使用して実装した．Subscriber Logic における ROS のメッセージ形式の解釈では，C 言語でステートマシンを記述し，ROS メッセージの内のピクセルデータのみを読みだすこととした．また，画像の width, height は定数 (640, 480) とした．Subscriber Logic の出力は FIFO を通じて FIFO2AXI に送られる．FIFO2AXI は，読みだしたデータを AXI4 ストリーム形式に変換を行う，FIFO2AXI の合成に用いたソースコードを**図 9** に示す．入出力のインタフェースは pragma を使用して指定し，FIFO 入力 (din_fifo2axi) は ap_fifo, AXI4 ストリーム出力 (IMAGE_STREAM) には axis を使用した．動作は画素数分を繰り返す二重の for ループで記述し，一回のイテレーションでは入力 FIFO から 4 つの 8bit データを読み出し，1 画素分の 32bit 幅のデータとして AXI4 ストリームに出力する．この際，3.2 節で説明した Vivado HLS ビデオライブラリにおける画像転送用の AXI4 ストリーム形式にするために，画像の行終わりの場合は last を 1 にし，フレームの始めのピクセルの場合は user を 1 にした．

Publisher HW を構成する Publisher Logic, AXI2FIFO も同様に高位合成を使用して実装した．AXI2FIFO は，FIFO2AXI と同様に実装した．一方，Publisher Logic の現状の実装では，ピクセルデータ以外は定数として，メッセージを生成することとした．なおこれらの Subscriber Logic, Publishr Logic は [8] に記載の方法で，ROS プロトコルの内の TCPROS 部分のみの通信を行い，XMLRPC 部分は外部の PC で行うこととした．

4.3 動作確認

コンポーネント化された FAST 特徴点検出回路の動作確認を行うため，**図 10** に示す ROS システムを構築した．この ROS システムでは，カメラを用いて入力を行い，

Vol.2018-ARC-230 No.3
Vol.2018-SLDM-183 No.3
Vol.2018-EMB-47 No.3
2018/3/7

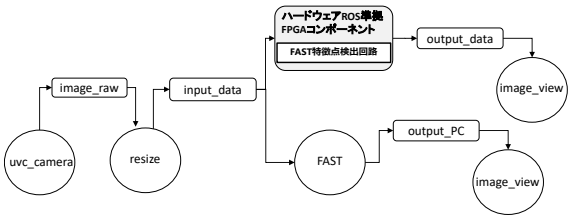

図 10 FAST 特徴点検出処理を行う ROS システム

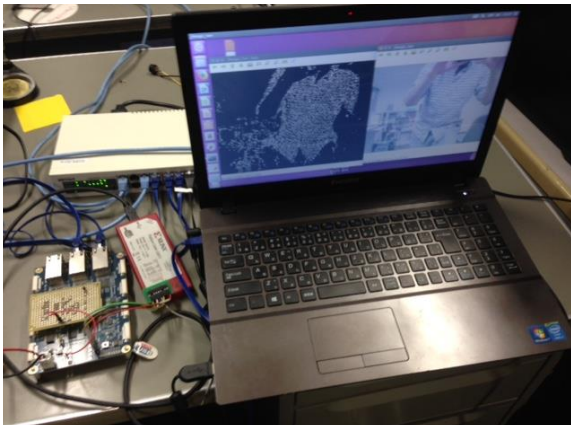

図 11 動作確認の様子

FAST 特徴点が検出された画像をディスプレイに表示する．uvc_camera ノード，image_view ノードは ROS の公式 wiki にて公開されている [13]．resize ノード，FAST ノードは OpenCV を使用して実装を行った，ソフトウェアである．uvc_camera ノードは，カメラ画像を image_raw トピックに配信する．resize ノードは，image_raw トピックから購読したデータをリサイズし，input_data トピックに配信する．FAST ノードは，input_data トピックから購読した画像データに FAST 特徴点検出処理を行い，output_PC トピックに処理結果を配信する．FAST 特徴点検出回路をコンポーネント化したハードウェア ROS 準拠 FPGA コンポーネントは，PC 上の FAST ノードと同様に, input_data トピックから購読し，処理結果を output_data トピックに配信する．image_view ノードは，sensor_msg/image 型の画像をディスプレイに表示を行うノードである．

この ROS システムにおいて，ソフトウェアである FAST ノードとコンポーネント化された FAST 特徴点検出回路の動作確認を行った．動作確認を行った際の実験環境の写真を**図 11** に示す．動作確認は Ubuntu16.04，ROS のバージョンは Kinetic で行い，使用した FPGA ボードは，株式会社 e-trees.Japan の exTri-CSI である [14]．ノート PC と FPGA ボードをギガビットイーサネットを用いて接続し，カメラ画像に対して PC 同様の FAST 特徴点検出処理が可能であることを確認した．

5. おわりに

本稿では，高位合成を使用した画像処理回路を ROS 準拠 FPGA コンポーネント化する際のハードウェア構造および設計フローについて検討した．ROS メッセージと AXI4 ストリームの変換処理を行う回路を高位合成で作成し，Vivado HLS のビデオライブラリ (OpenCV) を用いた FAST 特徴点検出回路のコンポーネント化をした．ROS のシステムを構築し動作確認をしたところ，FAST 特徴点検出を行えていることが分かった．

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．

参考文献

[1] 森岡博史, 長谷川修. ”生活支援人混みでも環境地図を学習して稼働する自律移動ロボットを開発”. 画像ラボ 2011 年 7 月号, 日本工業出版, pp.1-7, 2011.
[2] 石田 裕太郎，大川 猛，田向 権，“ FPGA のためのロボットミドルウェアインタフェースの基礎検討 ”，第 61 回 システム制御情報学会研究発表講演会，324-2, 2017.
[3] 三好健文, 船田悟史. ”FPGA 向け高位合成言語としての java の活用手法の検討.” 第 53 回プログラミング シンポジウム予稿集, pp. 59-68, 2012.
[4] 高前田伸也. ”PyCoRAM による Python を用いたポータブルな FPGA アクセラレータ開発.” 組込みシステムシンポジウム 2014 論文集, pp. 2-2, 2014
[5] K. Yamashina, T. Ohkawa, K. Ootsu and T. Yokota, “ Proposal of ROS-compliant FPGA Component for Low-Power Robotic Systems - case study on image processing application, ”Proc. of 2nd Intl. Workshop on FPGAs for Software Programmers, FSP2015, pp. 62-67, 2015.
[6] K. Yamashina, H. Kimura, T. Ohkawa, K. Ootsu, T. Yokota, “ cReComp: Automated Design Tool for ROS-Compliant FPGA Component, ” In proc. IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC-16), 2016.
[7] 山科 和史，松本 拓也，大川 猛，大津 金光，横田 隆史，“ FPGA を用いた処理のロボット向けコンポーネントと設計生産性評価 ”，電子情報通信学会 クラウドネットワークロボット研究会 (CNR)， 信学技報, vol.116, no.461, pp.23-28, 2017.
[8] Y.Sugata, T.Ohkawa, K.Ootsu, and T.Yokota. ”Acceleration of Publish/Subscribe Messaging in ROS-compliant FPGA Component.” In Proceedings of 8th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART 2017), 2017.
[9] 菅田 悠平，大川 猛，大津 金光，横田 隆史, “ ハードウェア TCP/IP スタックを使用した ROS 準拠 FPGA コンポーネントの通信性能評価 ”, 組込みシステムシンポジウム 2017 論文集, pp.62-69 , 2017.
[10] Tomohisa Uchida. ”Hardware-Based TCP Processor for Gigabit Ethernet,” IEEE Transactions on Nuclear Science, Vol.55, No.SIG 3, pp.1631-1637, 2008 .
[11] 小野 雅晃, ”AXI4 ストリームによるラプラシアン・フィルタ処理回路の実装,” CQ 出版社， FPGA マガジン,Vol.17, pp.80-98, 2017.
[12] Stephen Neuendorffer, Thomas Li, and Devin Wang. ”Accelerating opencv applications with zynq-7000 all programmable soc using vivado hls video libraries.” Xilinx Inc., XAPP1167(v3.0), 2015.
[13] ROS.org: http://wiki.ros.org/
[14] e-trees.Japan Inc.:http://e-trees.jp/