Vol.2022-ARC-248 No.57
Vol.2022-SLDM-198 No.57
Vol.2022-EMB-59 No.57
2022/3/11

FPGA を用いた遠隔ロボットハンドの制御の高速化

王 魯迪† 大川 猛‡

†東海大学大学院 情報通信学研究科 情報通信学専攻 〒108-8619 東京都港区高輪 2-3-23

‡東海大学 情報通信学部 組込みソフトウェア工学科 〒108-8619 東京都港区高輪 2-3-23

E-mail: †ludiwangjp@gmail.com, ‡ohkawa.takeshi@tokai.ac.jp

あらまし 座標データの処理を FPGA で高速化することで、離れた場所からロボットハンドを操作することを可能とする高速制御方式を提案する．FPGA の使用により、ロボットハンドの遅延の短縮、座標算出の性能向上、消費電力の低減を効果的に実現することを目指す．具体的には、手の座標情報をモーションキャプチャ装置によって収集し、FPGA による処理で座標・角度・加速度計算を高速化し、更にロボットハンドを ROS2(Robot Operating System version 2)を使って統合することで、遠隔ロボットハンド制御の高速化が可能となると考えられる．座標計算処理時間の初期評価を行ったところ、指の関節間角度の計算時間が ARM プロセッサで 5ms、AMD Ryzen で 3ms に対し、FPGA を用いた場合には 2ms という結果が得られた．一方、座標計算処理 ROS2 ノードの通信遅延時間は 21ms であり、遅延要求の 24ms には収まるものの、通信遅延の削減が課題である．

1.序論

近年、CPU の性能向上が鈍化していることなどにより，FPGA（Field Programmable Gate Array） に代表される製造後にユーザーが構成をカスタマイズすることができる再構成可能な高速処理・低消費電力デバイスが普及している[1]．例えば、画像処理や画像認識、センサ値からの姿勢推定・位置推定、3 次元座標の変換、地図作成と自己位置推定モータ出力制御、開発のためのモニタ表示、テストデータ入力、ログデータ記録などに FPGA デバイスを利用することでの処理の高速化が報告されている．上記のようなデータ処理は CPU デバイスでも可能であるが、消費電力の増加が問題である．消費電力が増加すると、バッテリーの持ちが悪くなるため、ロボットに大きな重いバッテリーを搭載しないといけなくなる．FPGA は、ハードウェアの並列処理によって、ソフトウェアでは実現できない高性能な処理を、低消費電力かつ省スペースで実現できる可能性がある[2]．そこで CPU デバイスに代わって、より多くのインテリジェントな FPGA デバイスが私たちの生活に入り込んできており、多くの利便性をもたらしてくれる．例えば、離れた場所にいる患者さんを手術する必要がある医師．危険な化学試薬を正確に混合する必要がある科学者．など、多くの人たちの悩みを遠隔ロボットハンドで解決することが可能であると考えられる．

現在一般的なロボットハンドの制御方法は、「ユーザによる仮想コンソールでのコマンド入力、もしくは物理的なジョイスティックによる操作」である．

本研究では、FPGA で制御される遠隔ロボットハンドの実現を目指して検討を行う．モーションキャプチャ装置で収集したユーザーの手の座標情報を使って、手の動きを復元できる．図 1 のように、取得した座標データは ROS2 (Robot Operating System version 2)が Node として公開し、PYNQ [3]は Subscriber により必要なデータを取得する．

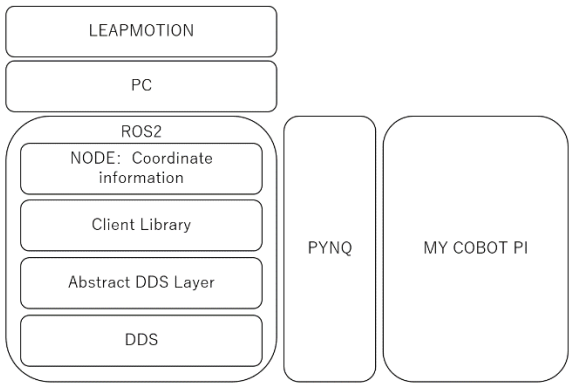

図 1 提案する遠隔ロボットハンド制御システム

FPGA の座標計算を高速化するために、ARM と FPGA 直接接続された通信バスがある PYNQ （ARM+FPGA）を選択した[3]．これによりデータ通信が高速になり、ロボットハンドの制御系がシンプルかつ効率的になり、データ処理による遅延が少なくなる．FPGA デバイスを使ったデータ処理の目標は、1 回の計算で 0.002 秒以下、1 万回の計算で 3 秒以下を実現する．

本稿の構成は以下の通りである．第 2 節では一般的な遠隔ロボットハンドの機能について紹介する．第 3 節では Leap Motion [4]と PYNQ を組み合わせたロボットハンドの提案について述べる．第 4 節では FPGA デバイスで行った性能評価と、他のデバイス(CPU と ARM)と比較した処理時間について述べる．最後に第 5 節で結論を述べる．

Vol.2022-ARC-248 No.57
Vol.2022-SLDM-198 No.57
Vol.2022-EMB-59 No.57
2022/3/11

2. ロボットハンドと FPGA

現在、FPGA のロボットハンドの研究が行われている.例えば、FPGA を活用した視覚神経模倣ロボットビジョンシステム[5]、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[6]などの研究がある. また、車両の座標を計算するために FPGA ロボットを用いた研究も行われている. また、ROS2 と FPGA を組み合わせて開発する研究も盛んになってきている.

一般的に、遠隔ロボットハンドは、ユーザーの手によって操作される. ユニークな点はロボットの指関節はそれぞれ個別に動くことができ、その動きはユーザーの手の座標と指関節の角度によって制御される. 人間の視覚や触覚は、平均して 0.024～0.044 秒の遅れを知覚することができる[7].そこでユーザーに遅延を感じさせないためには、モーションキャプチャ、ROS2 データ通信、データ処理、モータ制御の合計遅延時間は 0.024 秒（24ms）以内が要求される.

3. 提案

本節では、デザインをソフトウェア部分とハードウェア部分と ROS2 部分に分けて述べる. 3.1 では FPGA を用いて処理を行う座標計算のアルゴリズムを説明する. 3.2 ではソフトウェアの紹介を行う. 主に Python で設計したプログラムのフローチャートと、プログラムの組み方を述べる. 3.3 では ハードウェアの紹介を行う. IP コアとブロック図の作成を述べる. 3.4 では ROS2 の紹介. ROS2 のシステム構成と役割を述べる.

3.1. 座標計算のアルゴリズム

遠隔ロボットハンドは、操作者（Master）の手の状態をもとにして、遠隔ロボットハンド（Slave）の各関節をモータにより目標の角度に制御することで、操作者と同じ手の状態を実現する。操作者の手の状態を Leap Motion で取得すると、各指を構成する 4 本の骨（Bone）の両端の座標が得られる。これらの骨の座標から、骨と骨を連結する関節（Joint）の角度を計算し、遠隔ロボットハンドの関節のモータの制御目標角度を算出する必要がある。この計算には三角関数を多用するため、FPGA を用いて並列計算することで効率よく処理することが期待される。

座標の計算式は以下のとおりである。

$$a = (a_1, a_2, a_3)$$
$$b = (b_1, b_2, b_3)$$
$$c = (c_1, c_2, c_3)$$

$$x_1, y_1, z_1 = (a_1 - b_1), (a_2 - b_2), (a_3 - b_3)$$
$$x_2, y_2, z_2 = (c_1 - b_1), (c_2 - b_2), (c_3 - b_3)$$
$$\cos\_b = \frac{(x_1 * x_2 + y_1 * y_2 + z_1 * z_2)}{\sqrt{x_1^2 + y_1^2 + z_1^2} * \sqrt{x_2^2 + y_2^2 + z_2^2}}$$
$$Angle = \cos^{-1}(\cos b * (\frac{180}{\pi}))$$

3.2. ソフトウェアの紹介

本研究の遠隔ロボットハンドシステムの、Python を用いたソフトウェア開発について紹介する. 図 2 に、Leap Motion を接続した PC と、FPGA を搭載した PYNQ から構成される、遠隔ロボットハンドシステムの処理フローを示す.

- PC 側は、まず初期設定を済ませ、Leap Motion から手の座標を取得する.得られたデータは ROS2 に publish する.
- ·PYNQ 側は初期設定を完了し、ROS2 から手の座標情報を取得する. allocate メソッドを使用してメモリアドレスを割り当て、入力データと出力データを担当する 2 つの 配列を作成する.その後、bitstream のロード、角度計算の開始、制御コマンドの出力が行われる.

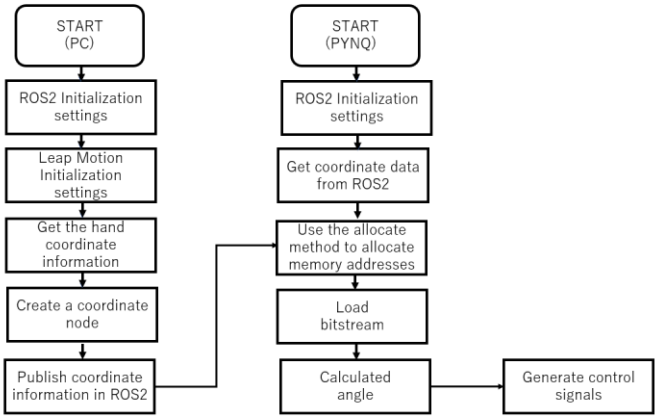

図 2 遠隔ロボットハンドシステムの処理フロー

3.3. ハードウェアの紹介

ハードウェア部分は、Vivado HLS を使用して、計算に必要な IP コアをカスタマイズしている. Vivado HLS とは Xilinx 社の提供する高位合成ツールの一つであり, 主に C, C++, System C などで記述したソースコードから Xilinx 社製 FPGA 用の HDL を自動で生成できる[8].

3.2.1. IP コア

MMIO（Memory-Mapped IO）（PS と PL はメモリアドレスを共有）により、配列 IN_ARR を読み込み、計算終了後に結果を OUT_ARR に入れて出力する.

Vol.2022-ARC-248 No.57
Vol.2022-SLDM-198 No.57
Vol.2022-EMB-59 No.57
2022/3/11

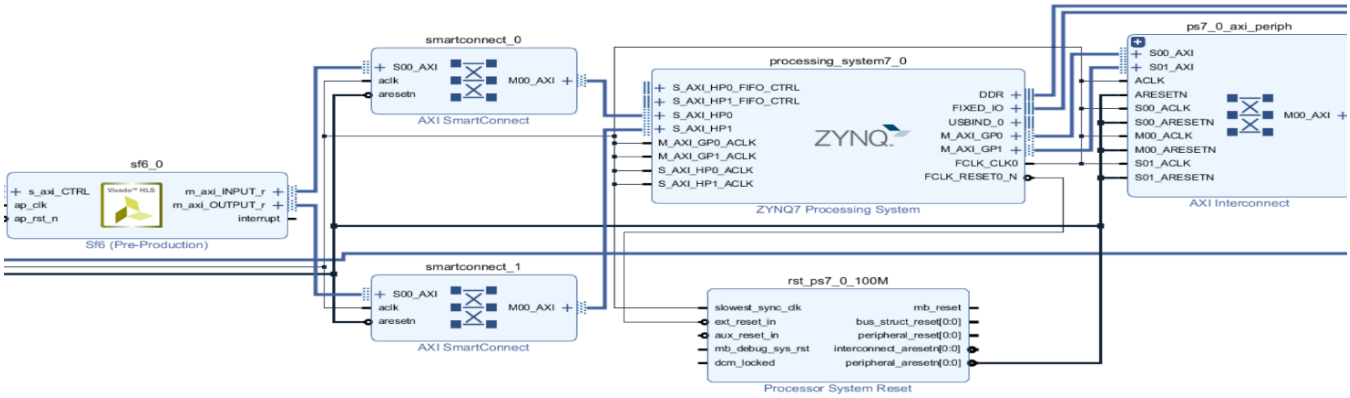

図 5 ブロック図

```
#include <ap_int.h>
#include <ap_fixed.h>
#include <hls_stream.h>
#include <math.h>
typedef ap_int<64> bit64;
typedef ap_uint<64> ubit64;

void sf6(bit64 in_arr[18], bit64 out_arr[18])
#pragma HLS INTERFACE m_axi depth=10 port=in_
#pragma HLS INTERFACE m_axi depth=10 port=out
#pragma HLS INTERFACE s_axilite register port
int one[12];
int i;

for(i=0;i<=14;i++){
    #pragma HLS PIPELINE II=2
    if (i<=2)
    {
    one[i]=in_arr[i]-in_arr[i+6];
    }
    if (3<=i<=5)
    {
    one[i]=in_arr[i]-in_arr[i+3];
}
    if (9<=i<=11)
    {
    one[i-3]=in_arr[i]-in_arr[i+6];
}
```

FPGAデバイスの初期設定

パイプラインの設定

データ分類

ラジアンの計算

角度計算

図 3 座標計算 IP コアのソースコード

```
// CTRL
// 0x00 : Control signals
//        bit 0  - ap_start (Read/Write/COH)
//        bit 1  - ap_done (Read/COR)
//        bit 2  - ap_idle (Read)
//        bit 3  - ap_ready (Read)
//        bit 7  - auto_restart (Read/Write)
//        others - reserved
// 0x04 : Global Interrupt Enable Register
//        bit 0  - Global Interrupt Enable (Read/Write)
//        others - reserved
// 0x08 : IP Interrupt Enable Register (Read/Write)
//        bit 0  - Channel 0 (ap_done)
//        bit 1  - Channel 1 (ap_ready)
//        others - reserved
// 0x0c : IP Interrupt Status Register (Read/TOW)
//        bit 0  - Channel 0 (ap_done)
//        bit 1  - Channel 1 (ap_ready)
//        others - reserved
// 0x10 : Data signal of in_arr_V
//        bit 31~0 - in_arr_V[31:0] (Read/Write)
// 0x14 : reserved
// 0x18 : Data signal of out_arr_V
//        bit 31~0 - out_arr_V[31:0] (Read/Write)
// 0x1c : reserved
// (SC = Self Clear, COR = Clear on Read, TOW = Toggle on Write, COH = Clear on Handshake)

#define XSF6_CTRL_ADDR_AP_CTRL       0x00
#define XSF6_CTRL_ADDR_GIE           0x04
#define XSF6_CTRL_ADDR_IER           0x08
#define XSF6_CTRL_ADDR_ISR           0x0c
#define XSF6_CTRL_ADDR_IN_ARR_V_DATA  0x10
#define XSF6_CTRL_BITS_IN_ARR_V_DATA  32
#define XSF6_CTRL_ADDR_OUT_ARR_V_DATA 0x18
#define XSF6_CTRL_BITS_OUT_ARR_V_DATA 32
```

図 4 メモリアドレス

図 3 のように、ホストプロトコルは m_axi(AXI マスタ)で、大量の座標データを送信するために使用される．1 本の指に対して骨が 4 個（関節が 3 個）あるため、6 つの 3 次元座標を用いて表現する．ソフトウェアから FPGA に渡すデータは 6 つの 3 次元座標を 64 ビット固定小数点数(ap_int<64>)で表したものである。ap_uint<64>すなわち 8 バイト×18 個の配列を用いて一本の指の情報（144 バイト）を受け渡すため、CPU は順次アドレスオフセットをインクリメントして座標データを IP コアに転送する．5 本の指の場合は、合計 20 個の関節があるため、合計 720 バイトの転送を行う．

座標データの入力後は、1 本の指あたり 4 つの骨の両端の 3 次元座標から対応する ACOS の値を求め、最終的には関節の角度を求める計算を行う．

図 4 のように、最終的に制御信号(Control signals)、入力信号(Data signal of in arr_v)、出力信号(Data signal of out arr_v)が生成され、次の論理合成の部分で使用される．

また、IP コアへのデータの読み込みを高速化するために Vivado HLS の PIPELINE プラグマを使用した [9]．この方法は、関数またはループの Iteration Interval を短縮するものであり、FPGA の性能向上に効果的であり、座標データの計算のループにおいても有効であることを確認した．

3.2.2. ブロック図の作成と論理合成

図 5 に示すように、IP コアは一番左の sf6_0 である．HLS の入力ポート INPUT は AXI ポート m_axi_INPUT_r に合成され，出力ポート OUTPUT は m_axi_OUTPUT_r に合成され，制御ポート s_axi_CTRL に合成される．データの入力と出力のために、2 本の AXI SmartConnect を用いる必要がある．データ幅の異なるインターフェイス間のトランザクションは AXI SmartConnect によって自動で変換される[11]．ZYNQ Processing System とのデータ入力、データ出力を接続して PS と PL 間のロジック接続として機能する．

3.3. ROS2 の紹介

ROS2 は通信ミドルウェアとして DDS を用いることで，無線等通信リソースの限られた環境での複数ロボット動作，組込みマイコン環境での動作，リアルタイム制御等を目標とする[10]．

4. 性能評価

性能評価では、（1）開発した座標計算 FPGA IP コアの処理時間、（2）座標計算 FPGA IP コアを ROS2 ノード化した際の通信遅延時間、の評価を行うこととした。しかし、（2）は座標計算 FPGA IP コアの統合ができなかったため、座標データ ROS2 メッセージを用いた際の通信時間の評価のみ行った。

Vol.2022-ARC-248 No.57
Vol.2022-SLDM-198 No.57
Vol.2022-EMB-59 No.57
2022/3/11

4.1 座標計算 FPGA IP コアの処理時間

開発した座標計算 FPGA IP コアの評価のため、CPU と ARM を使って処理時間の比較を行った．3 つのデバイスが使用するボードを表 1 に示す．

表 1 評価環境

	型番
FPGA	PYNQ-Z2 (Xilinx XC7Z020)
小型 PC	Raspberry Pi 4B (Cortex-A72)
PC	ASUS ROG ZEPHYRUS G14 (AMD Ryzen 9 5900HS)

AMD Ryzen 9 5900HS は、コア数 8、スレッド数 16 の高性能 CPU である．Raspberry Pi 4B と PYNQ-Z2 は、消費電力、演算性能、GPIO の構成が似ている．対照実験として使用することができる．PC と Raspberry Pi 4B と PYNQ-Z2 の同じデータで計算した時間を記録した．

表 2 に、座標計算処理時間の測定結果を示す．PYNQ-Z2 は 1 回の座標計算を 0.002 秒で処理することが分かった．一方、ARM は 0.005 秒、Ryzen9 は 0.003 秒であった．表 2 および図 6 で、同様の座標計算処理を、6,000 回・10,000 回繰り返す時間を比較した。PYNQ-Z2 での座標計算処理時間は約 2 秒だった．Ryzen9 は演算能力が高い高性能な CPU デバイスであるが消費電力が大きく、それでも FPGA の方が高速に計算を完了することがわかる．同様の消費電力である ARM デバイスとの比較では、FPGA デバイスの方が高速であることが分かった．

表 2 座標計算処理時間(単位:s)

回数	ARM Cortex-A72	AMD Ryzen 9	Xilinx FPGA XC7Z020 (PYNQ-Z2)
1	0.005	0.003	0.002
100	0.038	0.031	0.025
6000	2.830	1.782	1.253
10000	4.156	2.947	2.092

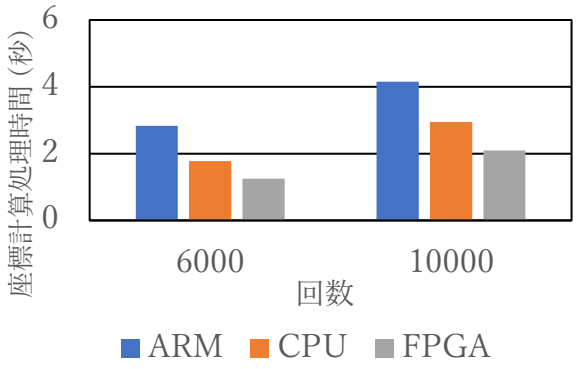

図 6 6000 回および 10000 回の座標計算処理時間(単位:s)

4.2 座標計算 ROS2 ノードの通信遅延時間

PC から PYNQ にデータを送る時間の評価のため、ROS2 通信測定環境を構築した．システムは PC (Ubuntu Linux 20.04) と PYNQ (Ubuntu Linux 18.04)、ルーターで構成されている．ROS2 のバージョンは、Foxy である．

図 7 に示すように、Leap Motion デバイスから取得した座標データは、1 本の指の情報（6 つの 3 次元座標）を 1 つの ROS2 メッセージとして、Publisher “Coordinate information” という Node が” Topic” に Publish する． PYNQ 上では、subscriber node “Display Device Status” が動作し、データを受信する．これにより、Leap Motion デバイスの状態を表示し、次に処理するデータを表示することができる．更に、PYNQ 上で subscriber node “Data Processing” が動作して座標データの処理を行うことを想定し、通信遅延時間の評価を行った。

表 3 に、PC 側の Publisher Node “Coordinate information” がデータを送信 (Publish) してから、PYNQ 側の Subscriber Node “Display Device Status” がデータを受信 (Callback) するまでの遅延時間を測定した．比較対象のために、TCP/IP のソケットによる通信の遅延時間も測定した．結果を表 3 に示す．メッセージ 30 回の通信の測定を行った結果、平均遅延時間は TCP/IP によるデータ送信で 29ms、ROS2 によるデータ送信で 21ms であった．

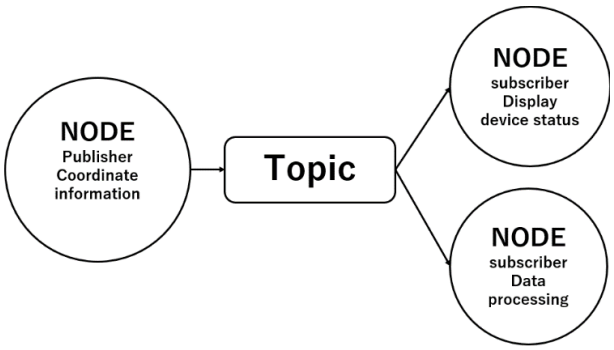

図 7 座標計算 ROS2 ノードの通信遅延時間評価環境

表 3 ROS2 および TCP/IP 通信遅延時間の測定結果

	遅延(単位: ms)
ROS2	21
TCP/IP	29

Vol.2022-ARC-248 No.57
Vol.2022-SLDM-198 No.57
Vol.2022-EMB-59 No.57
2022/3/11

4.3 考察

遠隔ロボットハンドの実現を想定した、4.1 節の座標計算 FPGA IP コアの処理時間評価では、高性能な AMD Ryzen 9 プロセッサと遜色ない処理性能（1 本の指 1 回あたりの処理時間=約 2ms）を示すことが分かった．一方、4.2 節の座標計算 ROS2 ノードの通信遅延時間の測定では、1 本の指の情報（６つの 3 次元座標）の通信に 21ms の遅延時間がかかることが分かった．すなわち，座標計算よりも通信に 10 倍程度の時間がかかっており、通信が性能・遅延時間のボトルネックになる可能性が高い．ただし TCP/IP ソケットであっても 29ms の遅延があるため、必ずしも ROS2 のプロトコルを用いることが遅延の原因ではない．以上のことから、FPGA IP コアおよび ROS2 を用いて遠隔ロボットハンドのシステムを構築する際には、通信遅延を削減する方法を検討する必要がある．

5. 結論

本稿では、FPGA を用いた遠隔ロボットハンドの制御の高速化を目的とした座標計算 FPGA IP コアの処理性能、および座標計算 ROS2 ノードの通信遅延時間の評価を行った．座標計算処理を行う FPGA IP コアの論理回路を C 言語からの高位合成により作成した．更に、同 FPGA IP コアへのデータ入力および出力、制御信号の生成機能、および ROS2 データ通信を含むソフトウェアを作成した．FPGA を用いた遠隔ロボットハンドの制御の処理時間は、指の関節間角度の計算時間が ARM プロセッサで 5ms、AMD Ryzen で 3ms に対し、FPGA を用いた場合には 2ms という結果が得られた．FPGA を用いることで、従来の CPU より遠隔ロボットハンドの制御に必要な座標計算処理を高速化することができたと言える．

一方、座標計算処理 ROS2 ノード（ソフトウェア版）の通信遅延時間は 21ms であり、遅延要求の 24ms には収まるものの、通信遅延の削減が課題である．また、座標計算処理 FPGA IP コアを統合した ROS2 ノードの評価は、今後の課題である．

謝辞

本研究は,JST,CREST,JPMJCR19K1 の支援を受けたものです.

文 献

[1] M. Horowitz, “1.1 computing’s energy problem (and what we can do about it),” in Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International. IEEE, pp. 10-14, 2014.
DOI: 10.1109/ISSCC.2014.6757323
[2] “ロボットでも注目される FPGA”[Online] Available: https://www.acri.c.titech.ac.jp/wordpress/archives/141#toc1 アクセス日：2022/2/1
[3] “What is PYNQ” [Online] Available: http://www.pynq.io/home.html アクセス日：2022/1/16
[4] “Leap Motion Controller” [Online] Available: Ultraleap for Developers (leapmotion.com) アクセス日：2021/10/19
[5] 奥野弘嗣, “FPGA を活用した視覚神経模倣ロボットビジョンシステム,”電子情報通信学会 2020 年ソサイエティ大会 講演番号 CI-3-2, pp.1-2, 2020.
[6] 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,”情報処理学会 研究報告システム・アーキテクチャ (ARC) , Vol.2019-ARC-234, No.8, 6 pages, 2019.
[7] 門脇拓也, 丸山三智佳, 早川智彦, 松澤直熙, 岩崎健一郎, 石川正俊, “身体感覚と視覚情報にずれが生じる没入環境における映像遅延のユーザーパフォーマンスへの影響に関する研究,” 第 23 回日本バーチャルリアリティ学会大会 (VRSJ2018)(仙台, 2018.9.19)／論文集, 12B-1j, 2018.
[8] Ilya Ganusov, Henri Fraisse, Aaron Ng, Rafael Trapani Possignolo, Sabya Das, “Automated extra pipeline analysis of applications mapped to Xilinx UltraScale+ FPGAs,” in 2016 26th International Conference on Field Programmable Logic and Applications (FPL), 2016 IEEE International. 4 pages, 2016.
DOI: 10.1109/FPL.2016.7577344
[9] Xilinx, “Vivado HLS 最適化手法ガイド UG1270 (v2018.1),” 2018.
[10] 大川 猛, 菅田 悠平, 木戸 剛正, 若槻 泰迪, 大津 金光, 横田 隆史, “ROS2/DDS における FPGA を用いた Publish/Subscribe 通信処理の初期検討,” 情報処理学会研究報告 Vol.2018-EMB-48, No.3, 2 pages, 2018.
[11]“AXI SmartConnect”[Online]Available: https://japan.xilinx.com/products/intellectual-property/smartconnect.html アクセス日：2021/02/05