ロボットにおける高速計算を実現するための FPGA/ASIC/GPU開発～実装ソリューション

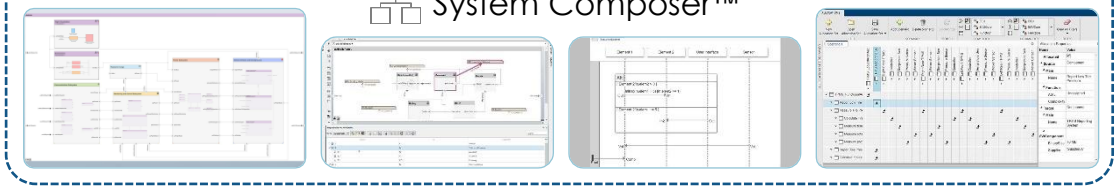
MathWorks Japan
アプリケーションエンジニア部

自律ロボティクス開発ワークフロー

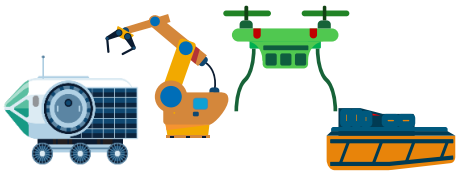
MATLAB® Simulink®

システムエンジニアリング

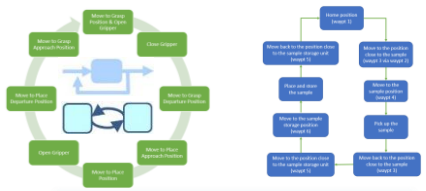
System Composer™



プラットフォーム設計

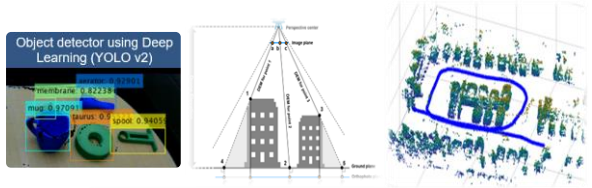


ロボットモデル

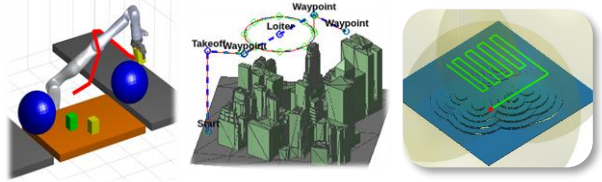


制御

AI・自律系設計

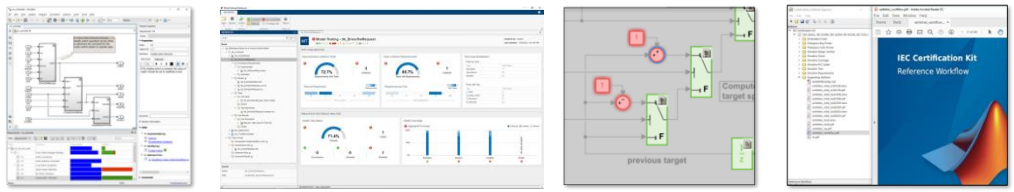


認識・自己位置推定



モーションプランニング

検証と妥当性確認



連成

テスト

実装

3Dシミュレーション

シーン・環境　　オブジェクト・アクター

ロボットとの接続

ハードウェアへの実装

Raspberry Pi™

ASIC, FPGA

PLC

PX4 Autopilot

NVIDIA® Jetson®

ROS Nodes

Speedgoat

アルゴリズムがシミュレーション上では動くことは分かったけど…

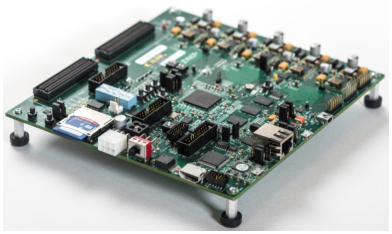
GPU、FPGA、ASICは非常に優れたデバイスですが、実装にはハードルも･･･

準備のハードル
- ハードウェアのアクセスが大変

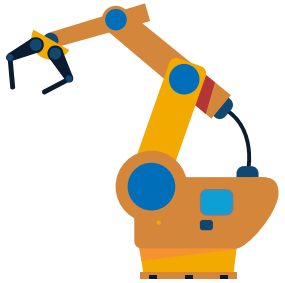
実装のハードル
- ハードウェアやインターフェイス周りの知識が必要
- ハンドコーディング時にバグが混入

検証のハードル
- アルゴリズム開発環境と検証環境／言語が異なり、性能検証するのが難しい

GPU



FPGA/ASIC



MATLAB／Simulinkを使うことでこれらのハードルを大きく下げ、“本質”に集中してGPU＆FPGAを開発しましょう！

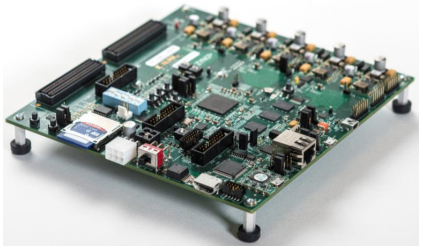
Agenda

・前半１５分：
GPU開発のハードルを下げる

準備
実装
検証

を活用

・後半１５分：
FPGA/ASIC開発のハードルを下げる



準備
実装
検証

を活用

Agenda

・前半１５分：
GPU開発のハードルを下げる

準備
実装
検証

MATLAB®
を活用

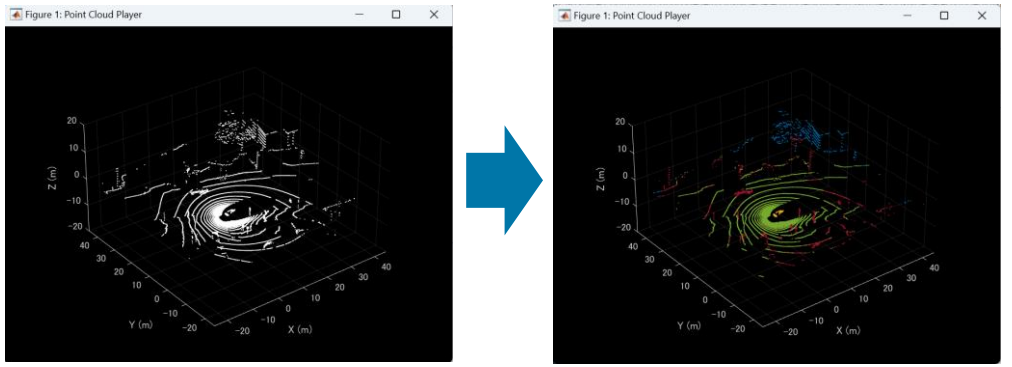
・後半１５分：
FPGA/ASIC開発のハードルを下げる

準備
実装
検証

SIMULINK®
を活用

デモ：点群処理での地面検出アルゴリズムをGPUへ実装

- ゴール：
  - 点群処理での地面検出のアルゴリズムを GPU（NVIDIA Jetson Orin™）へ実装



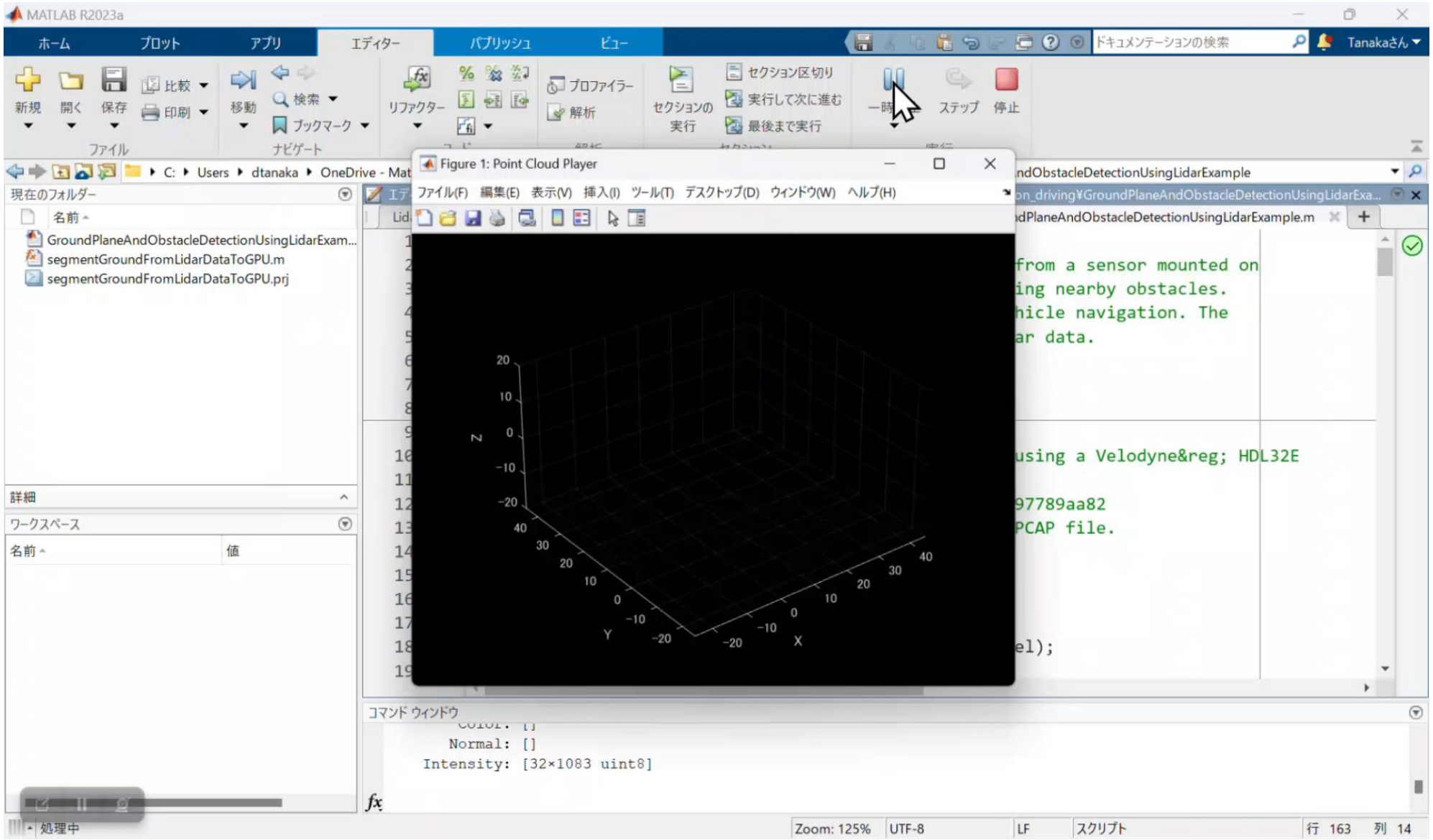
- ハードル：既存の開発方法

準備	ハードウェアへのアクセス
実装	ハンドコーディング
検証	開発環境と検証環境の相違

- MATLABを活用する方法
  - サポートパッケージでJetsonと連携
  - 開発したアルゴリズムと等価な GPUコードを自動生成
  - 同一環境上で性能検証

アイデアを具現化する“本質”に集中できる

コード生成・実行デモ


MATLAB R2023a
ホーム
プロット
アプリ
エディター
パブリッシュ
ビュー
Figure 1: Point Cloud Player
ファイル(F) 編集(E) 表示(V) 挿入(I) ツール(T) デスクトップ(D) ウィンドウ(W) ヘルプ(H)
現在のフォルダー
GroundPlaneAndObstacleDetectionUsingLidarExam...
segmentGroundFromLidarDataToGPU.m
segmentGroundFromLidarDataToGPU.prj
ワークスペース
コマンド ウィンドウ
Normal: []
Intensity: [32×1083 uint8]
処理中

Agenda

・前半１５分：
GPU開発のハードルを下げる

準備
実装
検証

– サポートパッケージでJetsonと連携

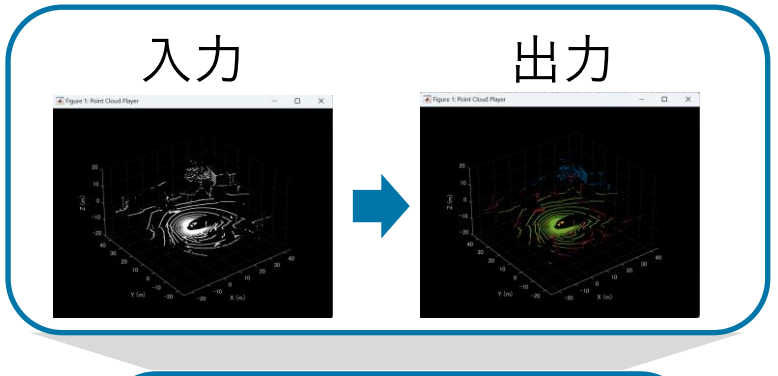
・後半１５分：
FPGA/ASIC開発のハードルを下げる

準備
実装
検証

サポートパッケージでJetsonと連携

お困りごと：ハードウェアへのアクセスが難しい ➡ サポートパッケージで連携環境構築支援

サポートパッケージ



開発用PC

NVIDIA Jetson Orin™

ディスプレイ

サポートしているハードウェア

NVIDIA Jetson Orin™、NVIDIA Xavier™ NX、NVIDIA Jetson AGX Xavier
NVIDIA Jetson Nano、NVIDIA Jetson TX2、NVIDIA Jetson TX1、NVIDIA DRIVE PX2

簡単にハードウェアにアクセスでき、素早くラピットプロトタイピングに移れる

Agenda

・前半１５分：
GPU開発のハードルを下げる

準備
実装
検証

– サポートパッケージでJetsonと連携
– 開発したアルゴリズムと等価な
GPUコードを自動生成

・後半１５分：
FPGA/ASIC開発のハードルを下げる

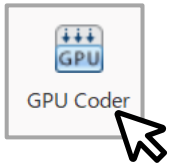
準備
実装
検証

開発したアルゴリズムと等価なGPUコードを自動生成

GPUコード生成対象選択

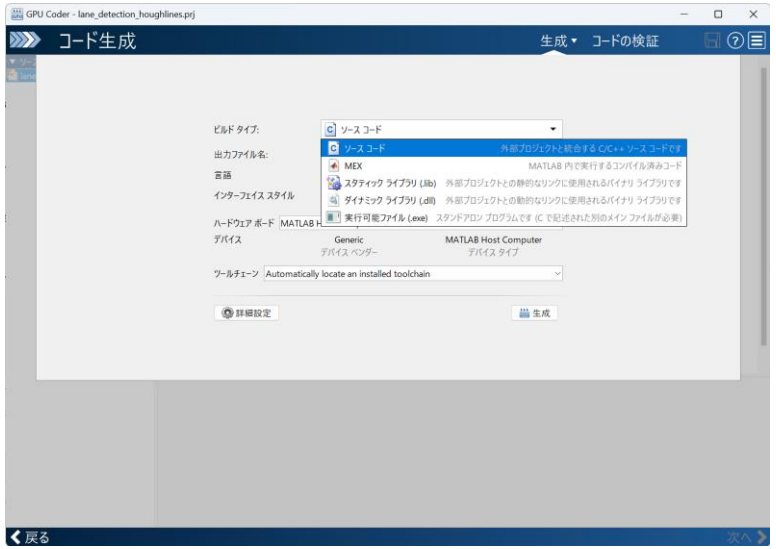
①起動方法

>> gpucoder もしくは



②GPUコード化する関数を選択

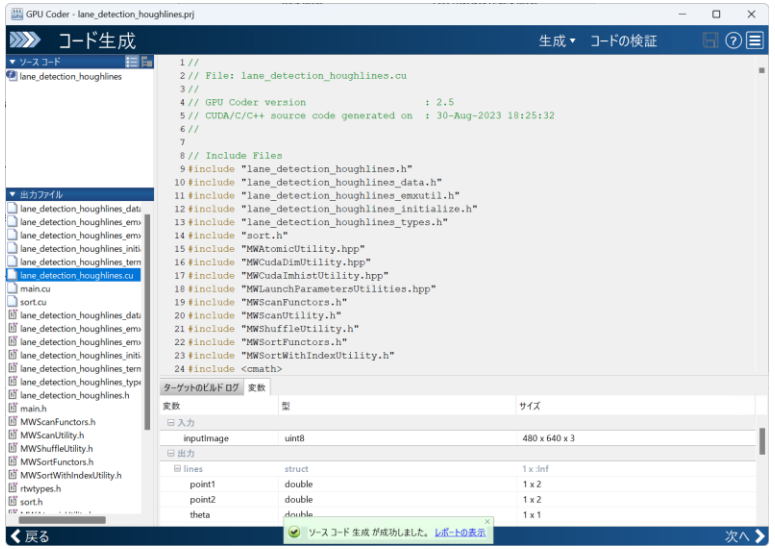
ビルドタイプ選択



③生成先を選択

- ソースコード
- MEX(MATLABで実行するコード)
- スタティックライブラリ(.lib)
- ダイナミックライブラリ(.dll)
- 実行可能ファイル(.exe)

生成したコード確認



④生成したソースコードを確認

GUIベースのアプリで等価なGPUコード生成可能

開発したアルゴリズムと等価なGPUコードを自動生成

MATLAB環境で開発することでハードを意識せず、抽象度高く記述

```
% Stop processing the frame after specified time.
stopTime = veloReader.EndTime;

i = 1;
isPlayerOpen = true;
while hasFrame(veloReader) && veloReader.CurrentTime < stopTime && isPlayerOpen

    % Grab the next lidar scan
    ptCloud = readFrame(veloReader);

    % Segment points belonging to the ego vehicle
    [egoPoints,groundPoints,obstaclePoints] = segmentGroundAndObstacles_pil(ptCloud.Location);

    i = i+1;
    closePlayer = ~hasFrame(veloReader);

    % Update lidar display
    points = struct('EgoPoints',egoPoints, 'GroundPoints',groundPoints, ...
        'ObstaclePoints',obstaclePoints);
    isPlayerOpen = helperUpdateView(lidarViewer, ptCloud, points, colors, closePlayer);
    pause(0.1);
end
```

MATLABコード

生成

```
// Arguments    : const float cpu_ptCloudLocation_data[]
//                const int ptCloudLocation_size[3]
//                bool cpu_egoPoints_data[]
//                int egoPoints_size[2]
//                emxArray_boolean_T *cpu_groundPoints
//                emxArray_uint32_T *cpu_obstaclePoints
// Return Type  : void
//
void segmentGroundAndObstacles(const float cpu_ptCloudLocation_data[],
                               const int ptCloudLocation_size[3],
                               bool cpu_egoPoints_data[], int egoPoints_size[2],
                               emxArray_boolean_T *cpu_groundPoints,
                               emxArray_uint32_T *cpu_obstaclePoints)
{
  coder::pointCloud ptCloud;
  coder::pointCloud ptCloudSegmented;
  dim3 block;
  dim3 grid;
  emxArray_boolean_T gpu_groundPoints;
  emxArray_real32_T gpu_dists;
  emxArray_real32_T *cpu_dists;
  emxArray_uint32_T gpu_obstaclePoints;
  double count;
  float *gpu_ptCloudLocation_data;
  int nonEgoGroundPoints_size[2];
  int i;
  bool cpu_nonEgoGroundPoints_data[35200];
  bool(*gpu_nonEgoGroundPoints_data)[35200];
```

GPUコード(CUDA)

データ転送、メモリの確保等、インターフェイス周りやハードウェア周りを
あまり意識せずに記述することが可能

アルゴリズム開発という本質部分に集中できる

Agenda

・前半１５分：
GPU開発のハードルを下げる

準備

実装

検証

– サポートパッケージでJetsonと連携
– 開発したアルゴリズムと等価な
GPUコードを自動生成
– 同一環境上で性能検証

・後半１５分：
FPGA/ASIC開発のハードルを下げる

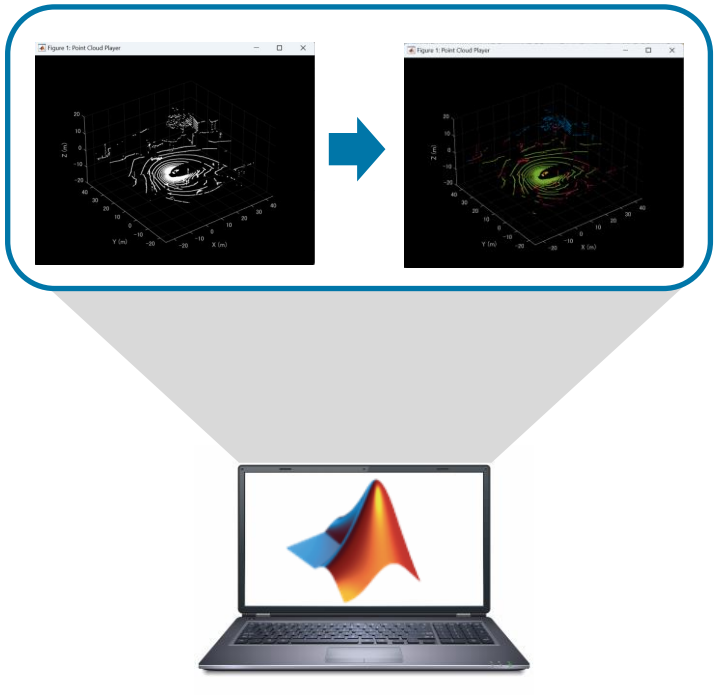
準備

実装

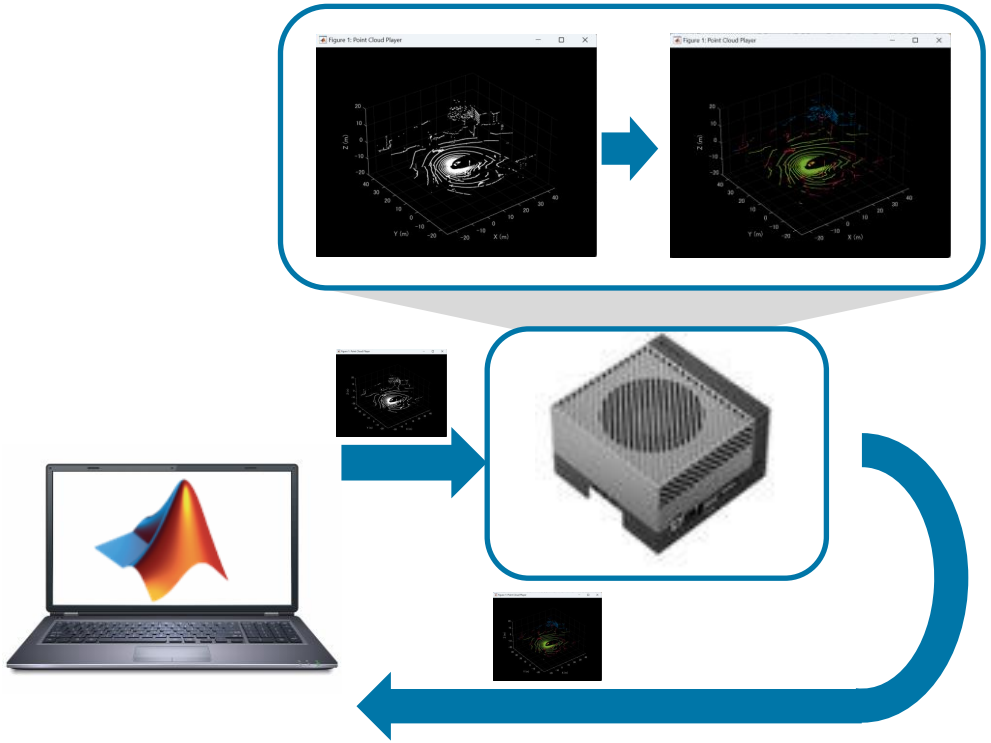
検証

Processor-in-the-Loop機能を用いたJetsonへの実装

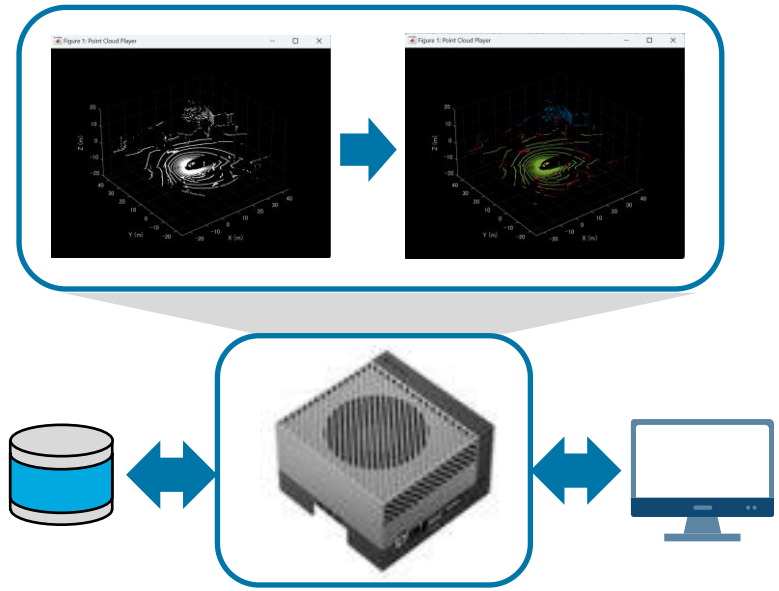
Software-in-the-Loop → **Processor-in-the-Loop** → **実機検証**



開発用PC上でコード実行



GPUコード化対象部分のみ実行

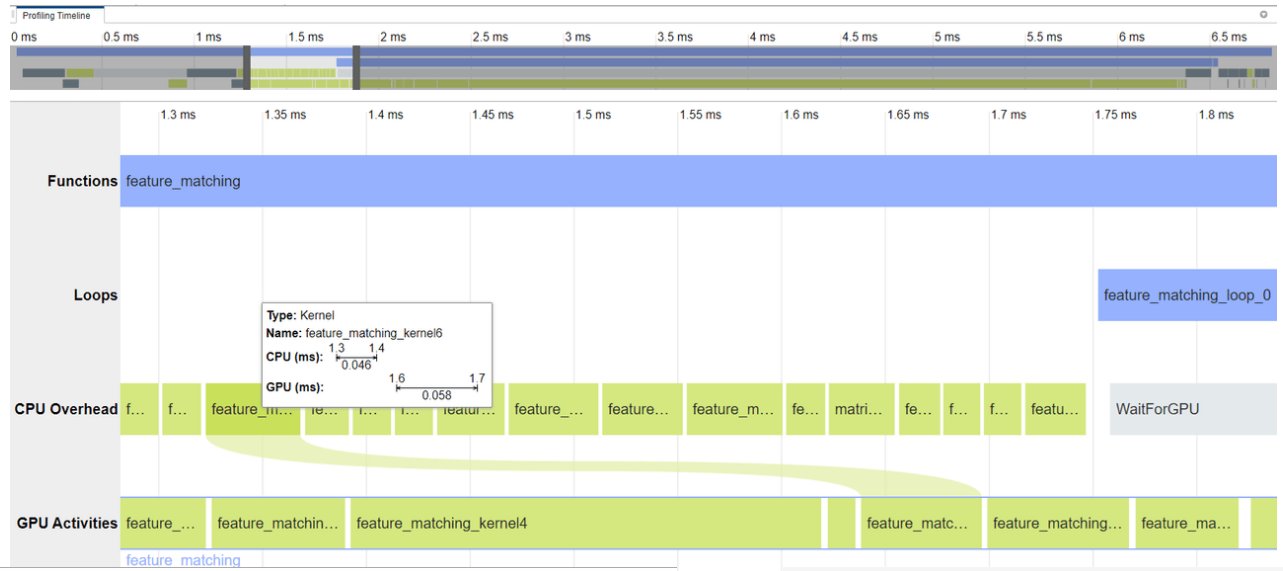


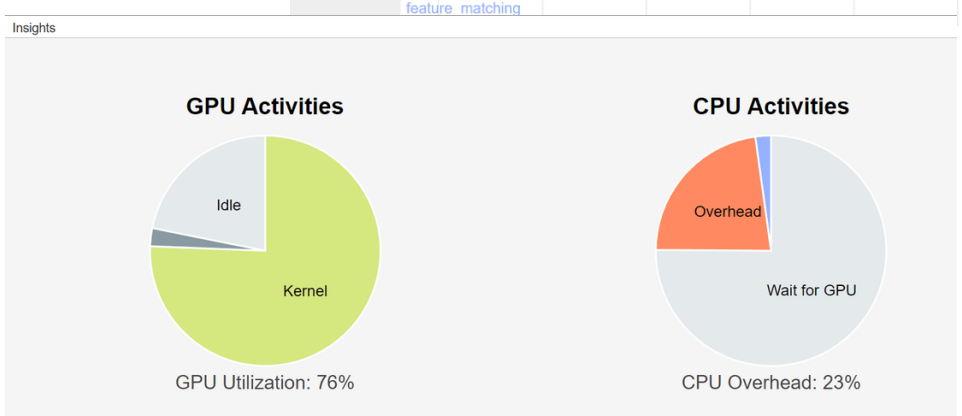
実機上でコード実行

MATLAB環境上で効率的に動作検証

GPU Performance Analyzerを用いた性能検証と最適化

- コード実行割合の可視化と最適化部分の特定によるチューニング
  - GPUとCPUの時系列の実行内訳を可視化し、時系列順に表示
  - 生成されたGPUコードのパフォーマンスを分析し最適化可能
  - MATLAB環境上からGUIで確認





Event Statistics

Type	Kernel
Name	feature_matching_kernel2
Start time	1.209344 ms
End time	1.253440 ms
Duration	0.044096 ms
Launch Params	
Grid size	[261, 1, 1]
Block size	[512, 1, 1]
Total threads	133.63 K
Shared memory	0 Byte
Registers per thread	16

MATLAB環境上で性能検証と最適化が可能

武蔵精密工業、自動車部品の外観検査にディープラーニングを活用

武蔵精密工業株式会社

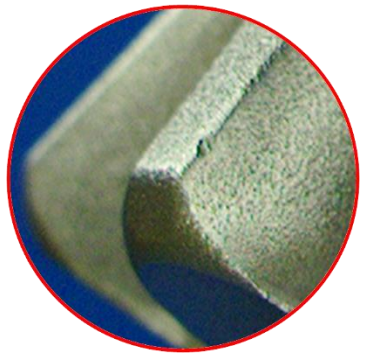
武蔵精密工業は、自動車部品ベベルギアの検査工程へのディープラーニングを用いた自動化を目指し、MATLAB®を用いたプロトタイピングを行いました。精度/速度の検証を経て、2018年5月からは製造現場での実証実験に取り組んでいます。今後、130万個/月の目視検査によって生じている作業負荷とコストの低減が期待されます。

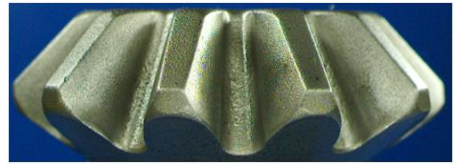
本プロジェクトではMathWorksのコンサルタントと協力し、画像の撮影手法の検討から前処理、App Designerを用いたアノテーションツールの作成、モデルの精度改善に取り組み、成果を上げました。出来上がったモデルはGPU Coder™ の自動コード生成機能を用いNVIDIA® Jetson™に実装し、ディープラーニングの判定結果をPLCと連携させています。

MATLAB利用のメリット:

- 画像撮影から組み込みGPUへの実装まで一貫した開発ワークフロー
- Class Activation Mappingによる判定要因の可視化
- ラベリング作業効率化のためのGUIツール作成
- MATLABの機能を効率よく活用するためのコンサルティングサービス

> “カメラ接続や前処理、様々な**学習済みモデルを簡単に利用**でき、ワークフロー全体に取り組むことができました。コンサルティングとの議論を通じ、**課題解決に多くのヒントを得られ、担当エンジニアの成長**にもつながりました。”



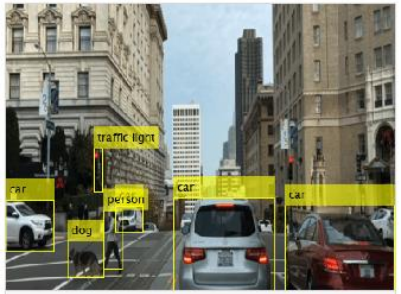


MATLABで始めるディープラーニング

ディープラーニング：製造現場で使える画像による異常検知

ビデオを見る (4:33)

すぐにお試しいただける製品例題



YOLO v4 深層学習を使用したオブジェクト検出用のコード生成

カスタム層を使用した You Only Look Once (YOLO) v4 オブジェクト検出器用のスタンドアロン CUDA® 実行可能ファイルを生成す

GPU Coder により最適化された車線検出

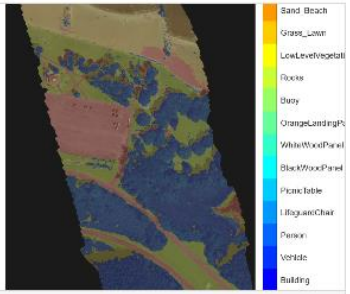
NVIDIA GPU で実行される深層学習車線検出アプリケーションを開発する。

交通標識の検出と認識

深層学習を使用する交通標識の検出および認識用途の CUDA MEX を生成する。

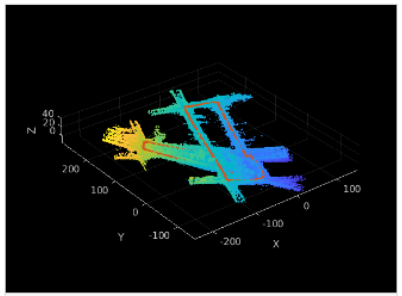
ロゴ認識ネットワーク

コードを生成し、入力イメージを 32 個のロゴ カテゴリに分類する。

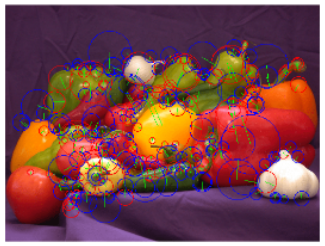


U-net を使用するセマンティック セグメンテーション ネットワークのコード…

イメージ セグメンテーション用の U-Net 深層学習ネットワークの CUDA コードを生成する。



GPU での SLAM の使用による LiDAR データからのマップ作成



SURF を使用した特徴抽出

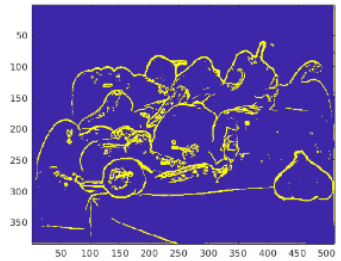
Speeded-Up Robust Features (SURF) を使用したオブジェクト認識は 3 つのステップで構成されます。特徴抽出、特徴記述、および特

特徴のマッチング

この例では、MATLAB® コードから CUDA® MEX を生成し、2 つのイメージ間で特徴マッチングを実行する方法を説明します。この例では、

関数 houghlines を使用した GPU での車線検出

この例では、イメージの車線マーカー境界を検出して出力する MATLAB® 関数の CUDA® MEX を生成する方法を説明します。この例



半精度でのソーベル法によるエッジ検出

この例では、MATLAB® 関数から生成された CUDA® MEX 関数を使用したイメージ内のエッジ検出について説明します。エッジ検出アルゴリ

GPUパートまとめ

・前半１５分：
GPU開発のハードルを下げる

準備
実装
検証

– サポートパッケージでJetsonと連携
– 開発したアルゴリズムと等価な
GPUコードを自動生成
– 同一環境上で性能検証

・後半１５分：
FPGA/ASIC開発のハードルを下げる

準備
実装
検証

SIMULINK®
を活用

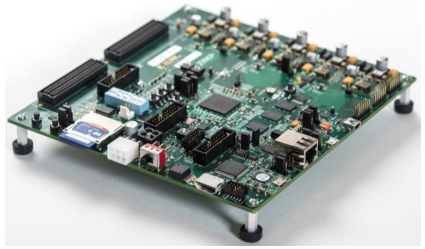
Agenda

・前半１５分：
GPU開発のハードルを下げる

準備
実装
検証

MATLAB®
を活用

・後半１５分：
FPGA/ASIC開発のハードルを下げる

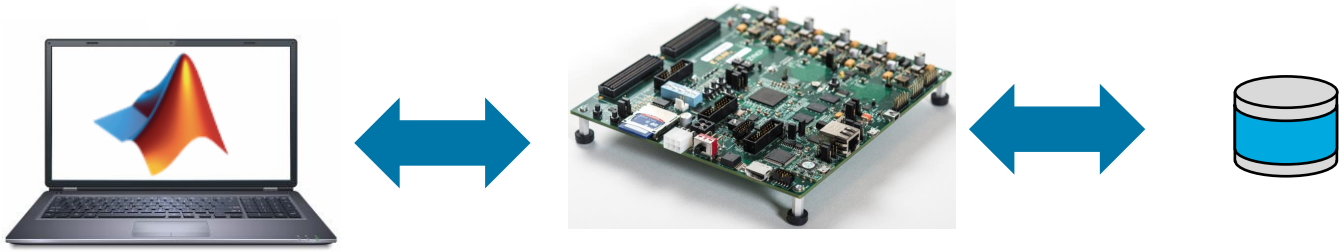


準備
実装
検証

SIMULINK®
を活用

デモ：簡単な点群処理アルゴリズムをFPGAへ実装

- ゴール：
  - 簡単な点群処理アルゴリズムをFPGA（Xilinx Zynq®）へ実装



開発用PC　　Xilinx Zynq ZC702　　LiDAR(Velodyne VLP-16)

- ハードル：既存の開発方法

準備　ハードウェアへのアクセス

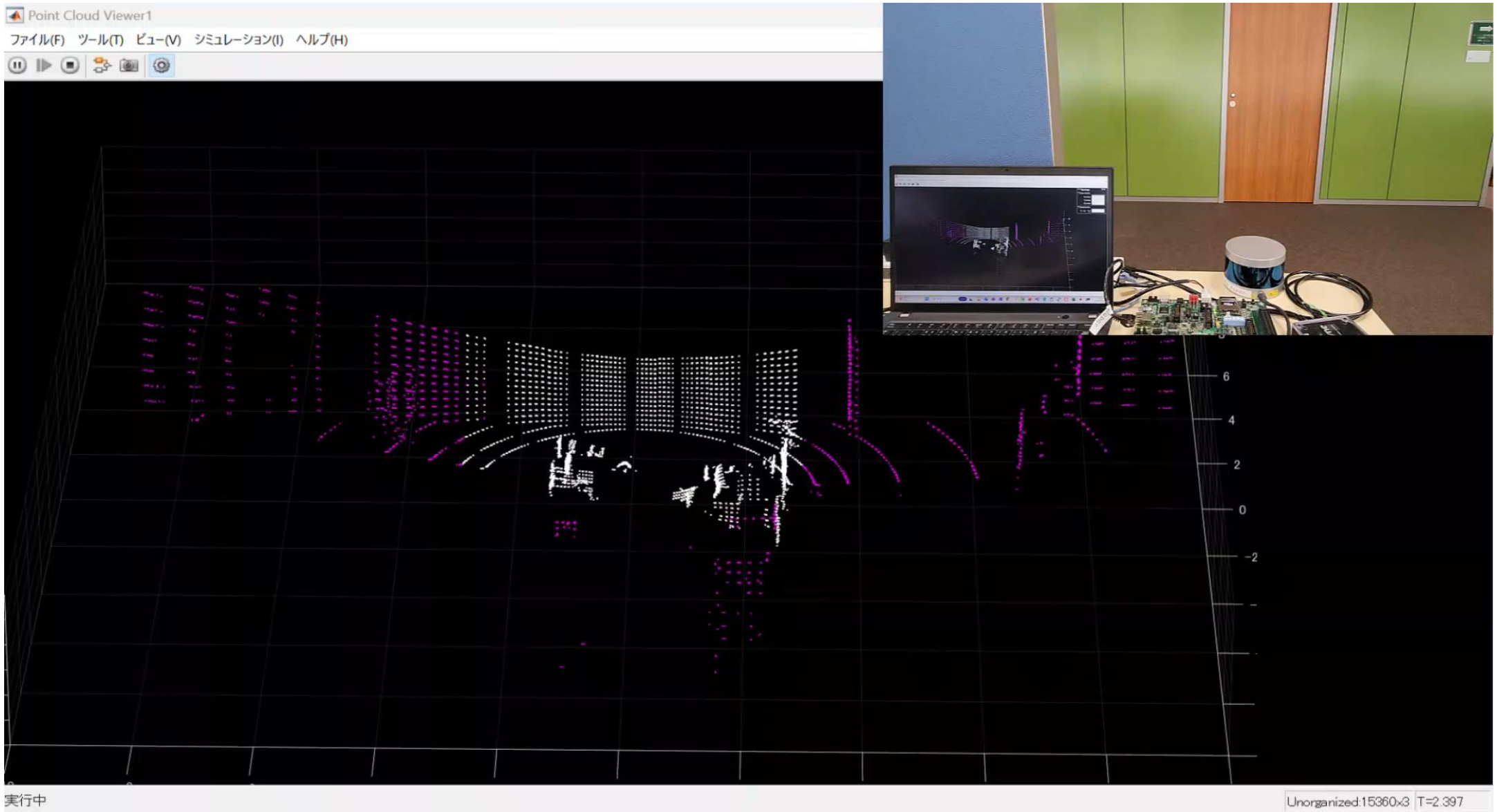
実装　ハンドコーディング

検証　開発環境と検証環境の相違

- Simulinkを活用する方法
  - サポートパッケージでXilinx Zynq開発環境構築
  - 開発したアルゴリズムと等価な HDLコードを自動生成
  - 同一環境上で性能検証

アルゴリズム開発という“本質”に集中することが出来る

FPGAへのLiDAR実装デモ



Agenda

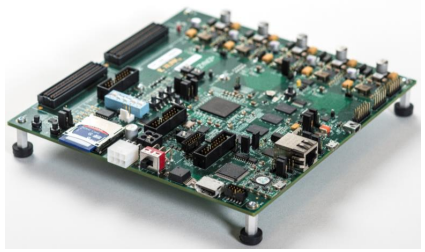
- 前半１５分：
  GPU開発のハードルを下げる

準備

実装

検証

- 後半１５分：
  FPGA/ASIC開発のハードルを下げる



準備

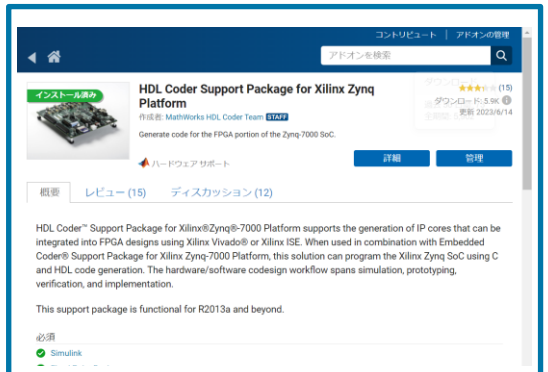
実装

検証

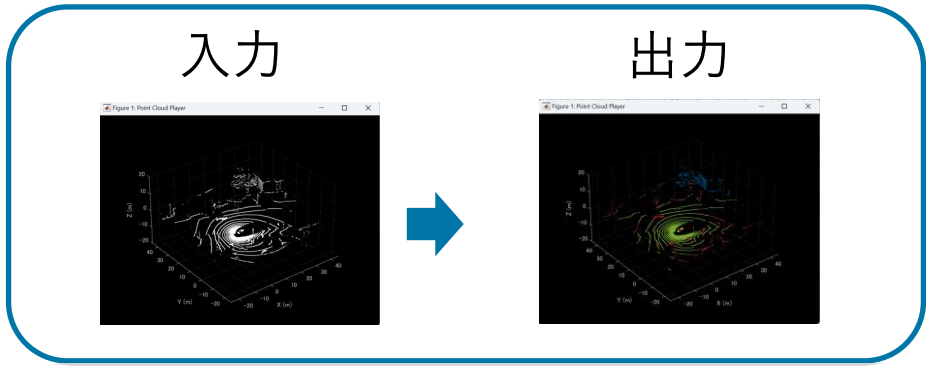
– サポートパッケージでXilinx Zynq開発環境構築

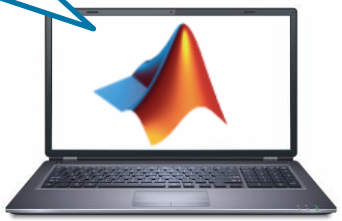
サポートパッケージでXilinx Zynq開発環境構築

お困りごと：ハードウェアへのアクセスが難しい・・・➡サポートパッケージで環境構築支援

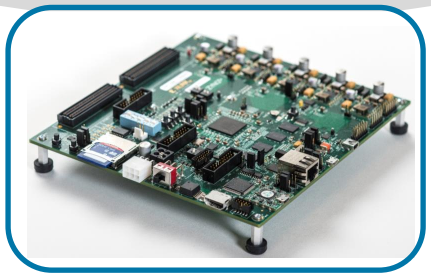


サポートパッケージ





開発用PC



Xilinx Zynq ZC702

LiDAR(Velodyne VLP-16)

HDL Coder Support Package for Xilinx Zynq Platformでサポートしているハードウェア・デバイスファミリ

ZedBoard、Xilinx Zynq ZC702、Xilinx Zynq ZC706、Xilinx Zynq UltraScale+、
MPSoC ZCU102、Xilinx Versal AI Core Series VCK190

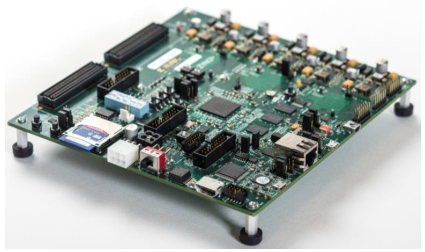
簡単にペリフェラル周りにアクセスし、素早くラピットプロトタイピングに移れる

Agenda

・前半１５分：
GPU開発のハードルを下げる

準備
実装
検証

・後半１５分：
FPGA/ASIC開発のハードルを下げる

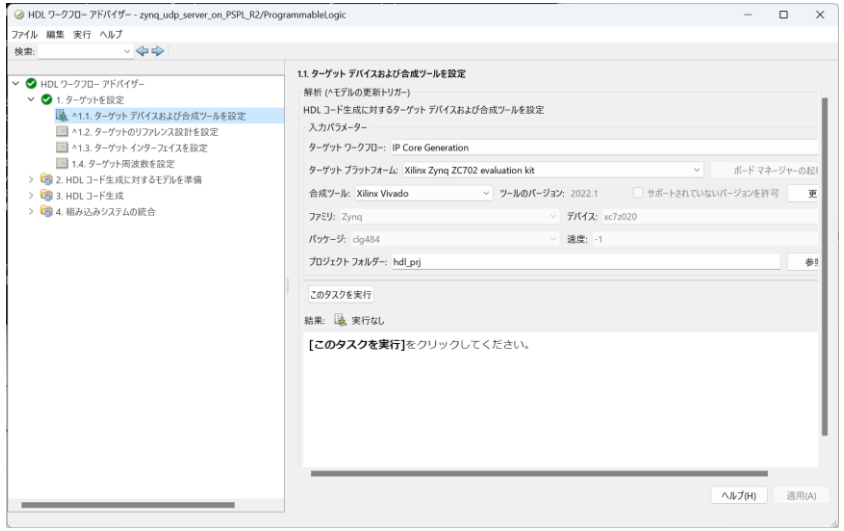


準備
実装
検証

– サポートパッケージでXilinx Zynq開発環境構築
– 開発したアルゴリズムと等価な
HDLコードを自動生成

開発したアルゴリズムと等価なHDLコードを自動生成

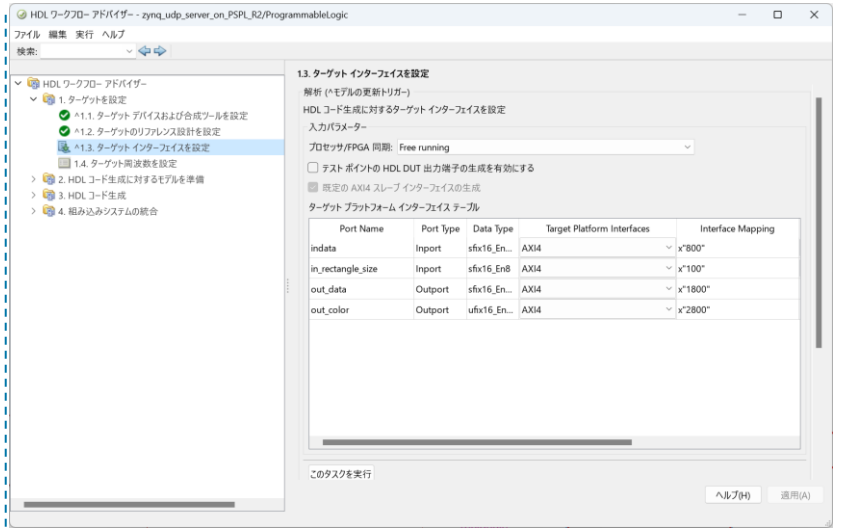
ターゲットFPGA選択



①生成先を選択
HDLコード生成のみ or Vivado/Quartus等の開発環境と連携するかを選択

②実装対象のボードを選択
サポートパッケージでサポートしているボードをデフォルトで用意

ハードへのマッピングを選択

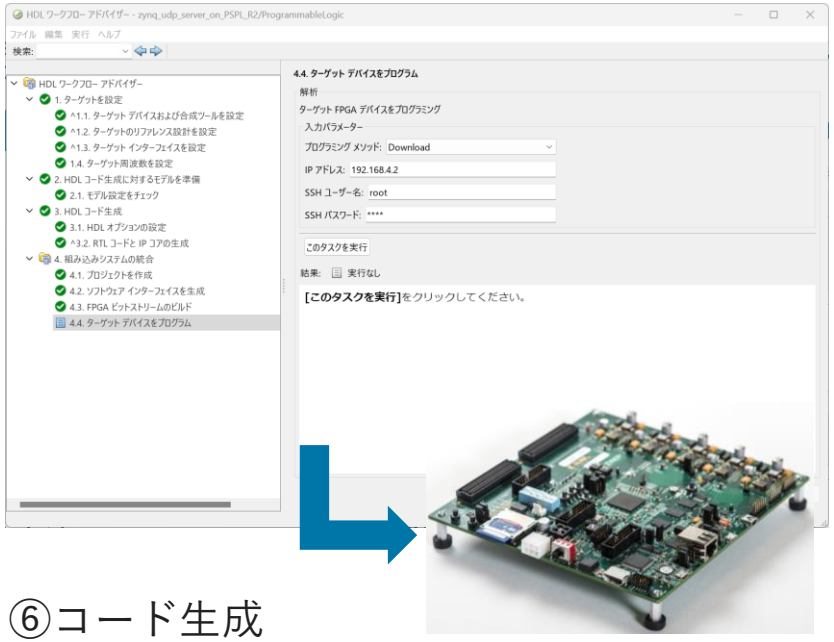


③インターフェイス指定
入出力ポート数、通信I/F(AXI等)、データサイズ・型を確認

④ターゲット周波数指定

⑤生成言語指定
VerilogHDL、VHDL

実装先選択



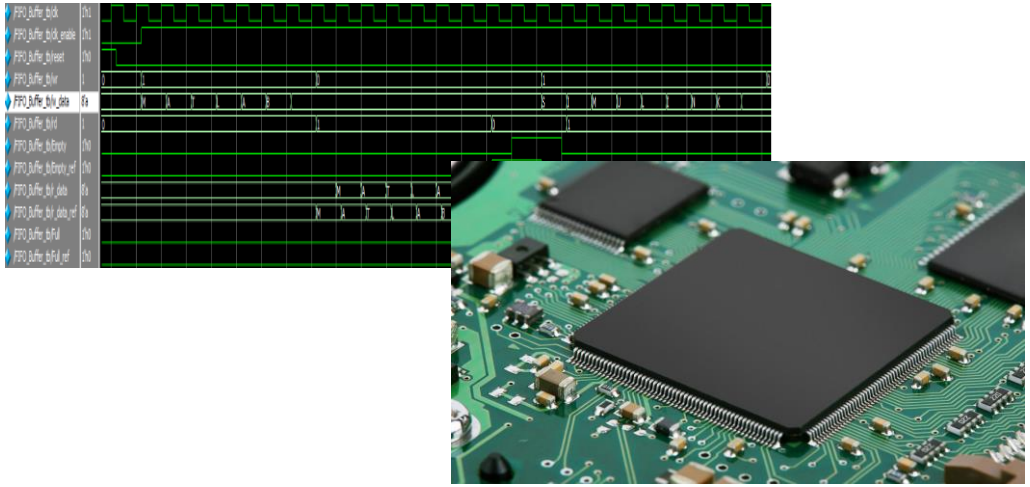
⑥コード生成
HDLコード、Vivado/Quartus等のプロジェクトファイル生成

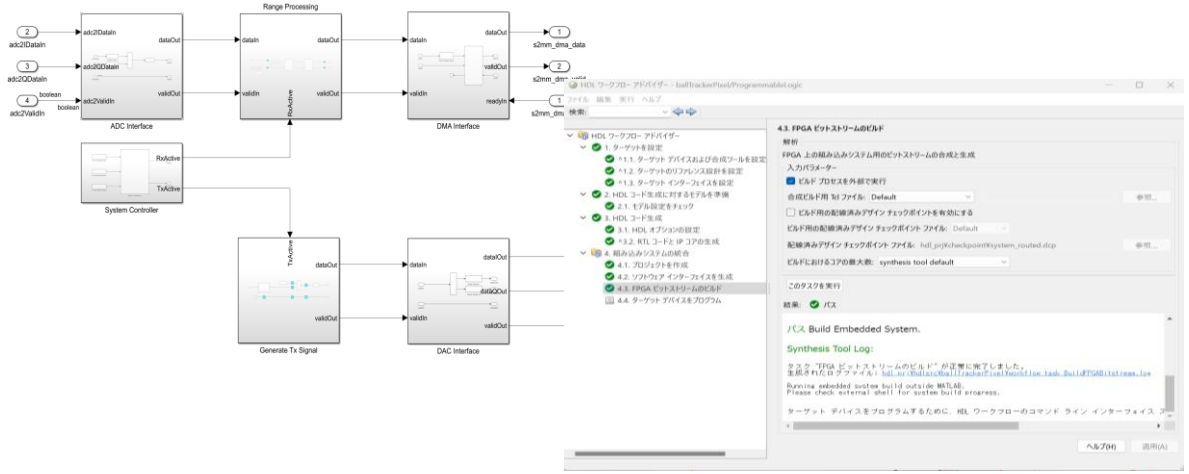
⑦ビットストリーム生成

⑧FPGAに書き込み

GUIベースのアプリでSoC FPGAシステム生成し、実装可能

階層構造やロジックにフォーカスした設計が可能





- ハンドコーディングでRTLの設計
  - 同期設計が難しい
  - I/Oピンのアサインが手間

- 遅延を意識せず階層構造やロジックにフォーカスした設計が可能
  - 高い抽象度での接続
  - タイミング意識せず、シミュレーションと等価な動作をするHDLを生成
  - SoCビルダー／ワークフローアドバイザー等による対話的なハードウェア割付

ハードウェア実装の細かいノウハウ習得より先に、すぐにアイデアを実現性検討出来る

Agenda

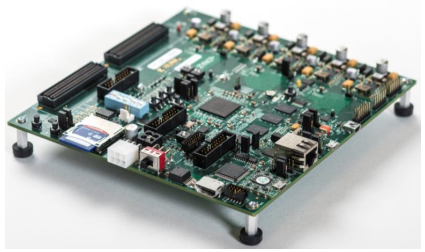
・前半１５分：
GPU開発のハードルを下げる

準備

実装

検証

・後半１５分：
FPGA/ASIC開発のハードルを下げる



準備

実装

検証

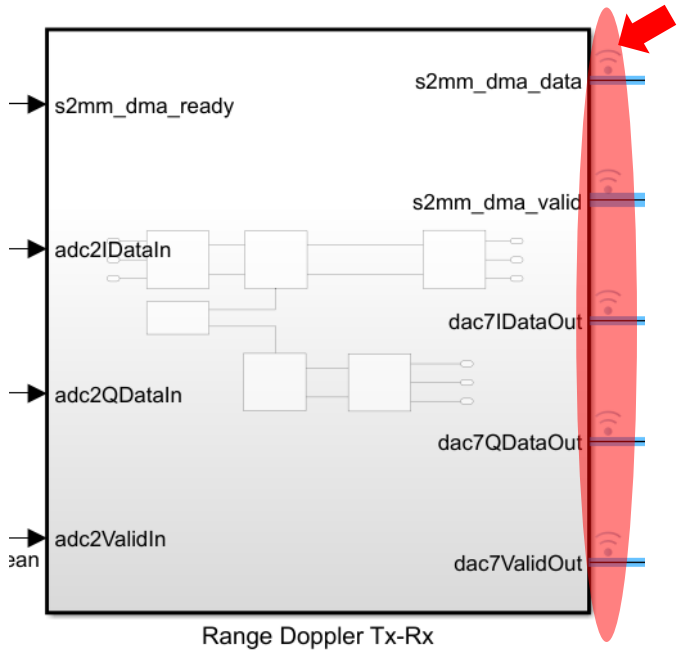
- サポートパッケージでXilinx Zynq開発環境構築
- 開発したアルゴリズムと等価な
  HDLコードを自動生成
- 同一環境上で性能検証

ロジックアナライザを用いたSimulink上からの検証

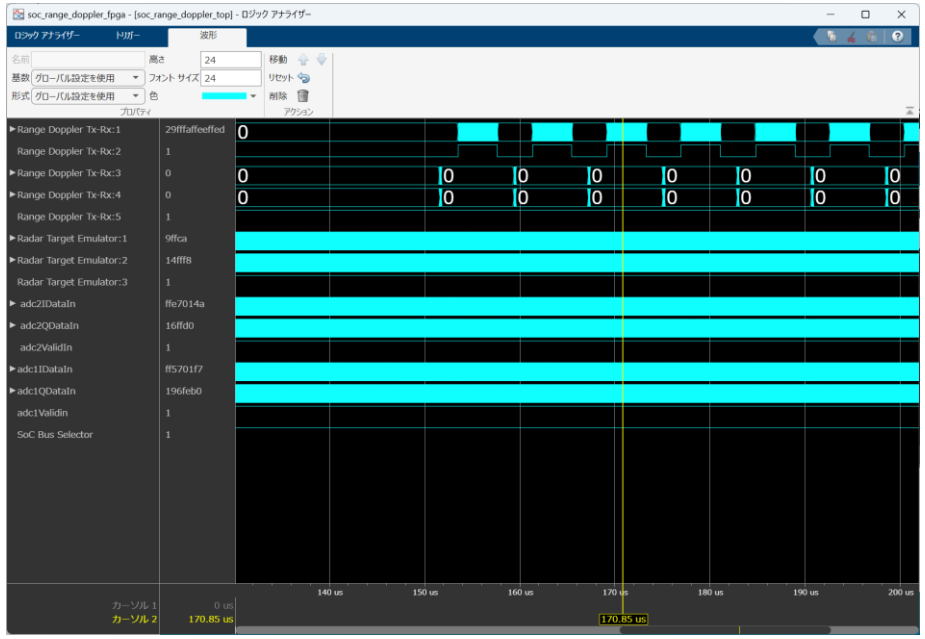
お困りごと：FPGAの性能改善・デバッグのためロジックの動作検証したい

➡ ロジックアナライザを用いて信号線の時系列データを確認

信号の取得箇所選択

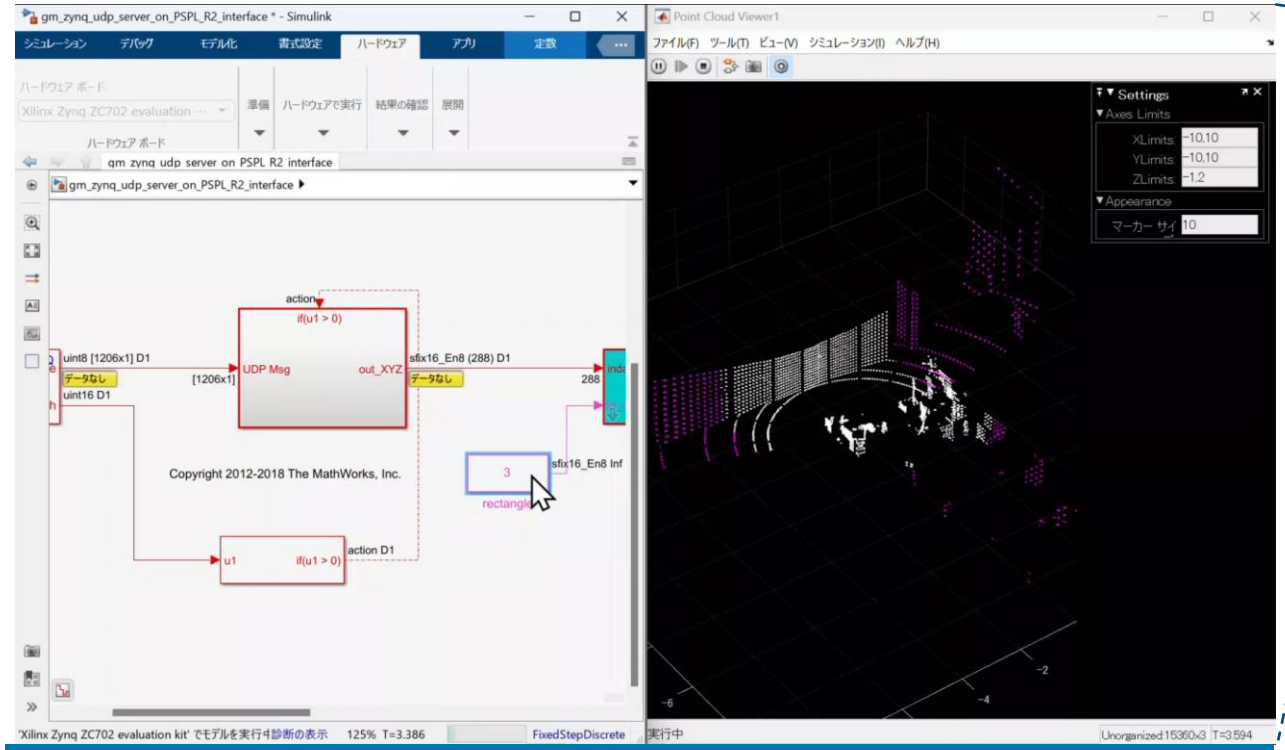


信号の値を確認

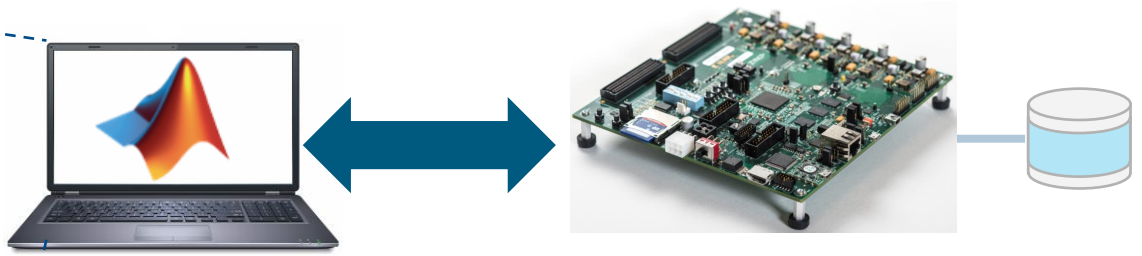


FPGAの動作をロジックアナライザにて検証可能

Simulinkと連携した動作中の実機パラメータ調整



例：FPGAと通信し、色範囲設定レジスタを更新



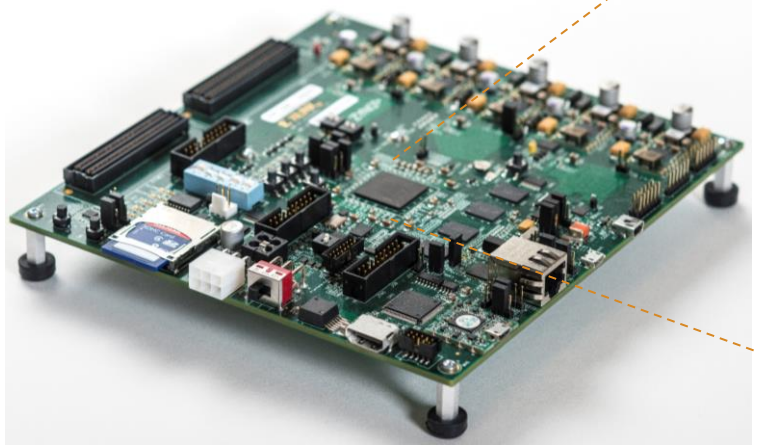
- 実機動作中にレジスタ値を調整可能
  - 実機でのデバッグ作業時間を削減
  - 実環境での調整時間を削減
  - 問題の切り分けが容易

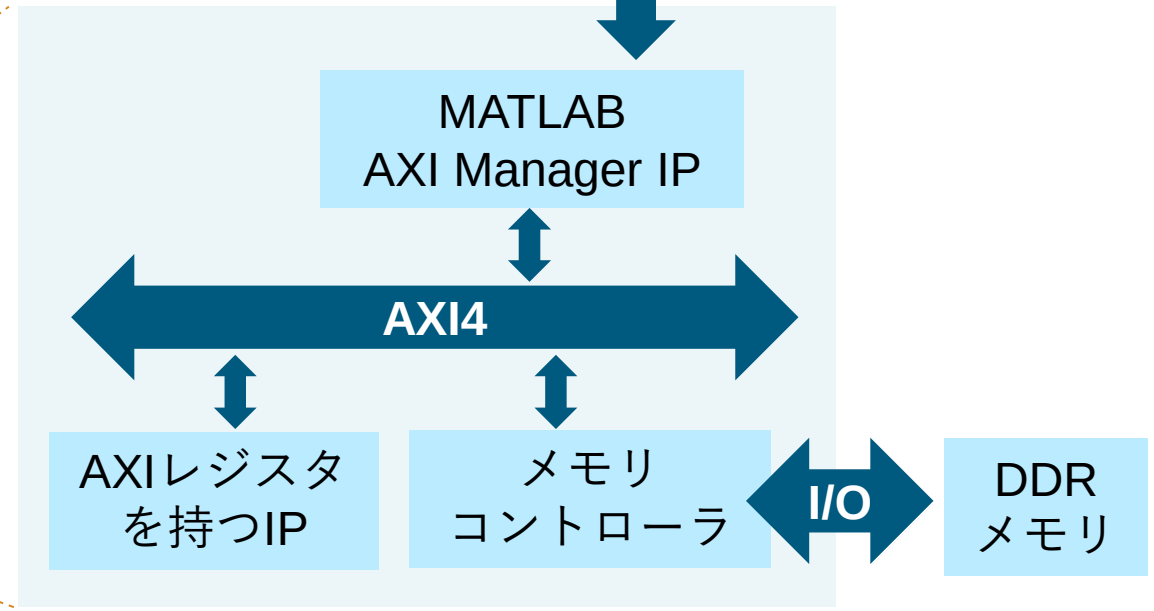
実機実装を加速

実機実装後の実環境へのリファインも容易

Simulink上からの実機実行時の動作検証

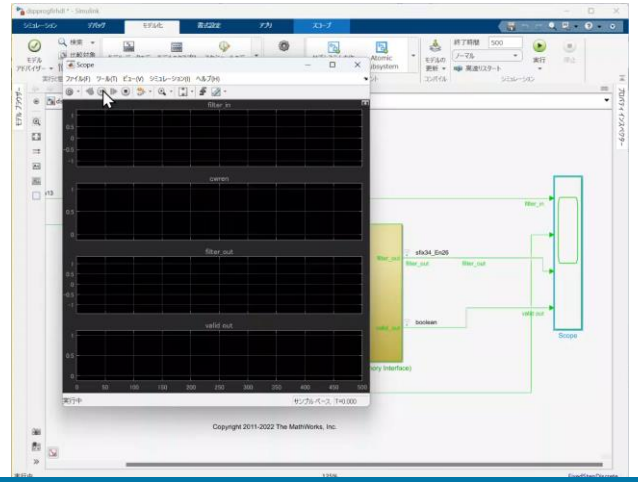
- AXI Managerを使った動作検証
  - MATLAB/Simulink環境から
    オンボードメモリへのアクセス
  - 動作時のメモリ／IPのレジスタの値を
    読み出し・書き込み



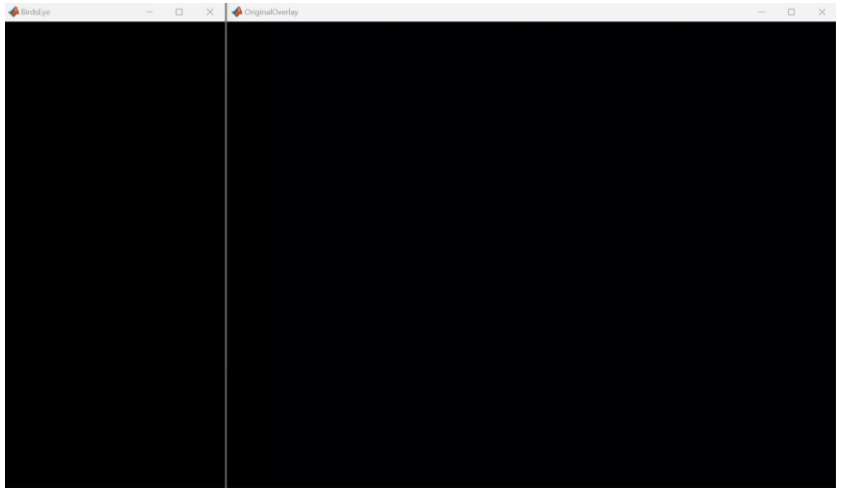


実装後の動作検証も強力にサポート

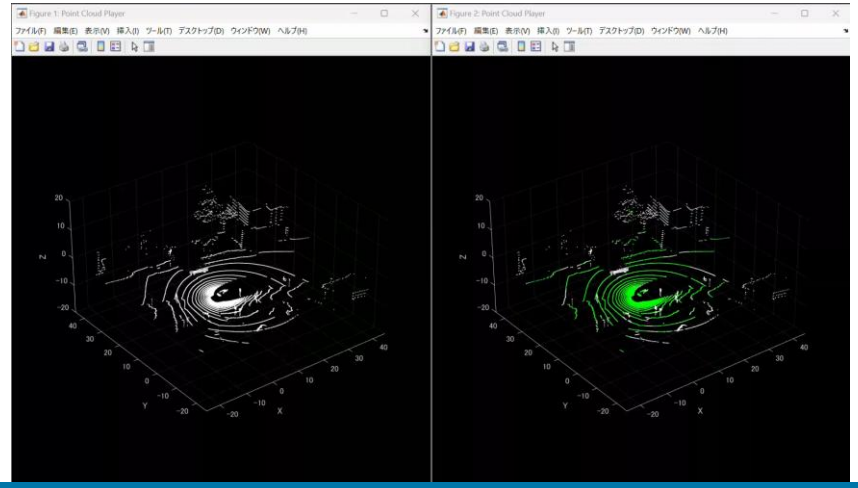
すぐにお試しいただける製品例題



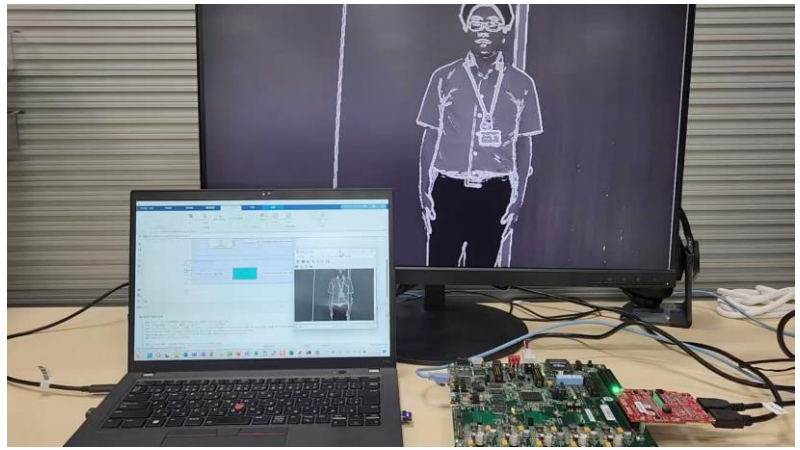
FPGA 用のプログラム可能な FIR フィルター



FPGAを用いたレーン検知



FPGAを用いたLiDARデータからの地面検知



ビデオカメラ入力のエッジ強調処理

FPGAパートまとめ

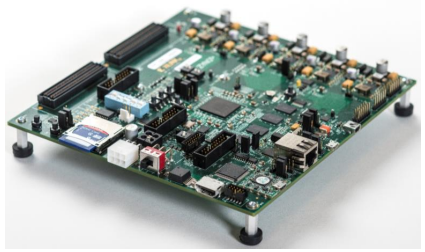
- 前半１５分：
  GPU開発のハードルを下げる

準備

実装

検証

- 後半１５分：
  FPGA/ASIC開発のハードルを下げる



準備

- サポートパッケージでXilinx Zynq開発環境構築

実装

- 開発したアルゴリズムと等価な
  HDLコードを自動生成

検証

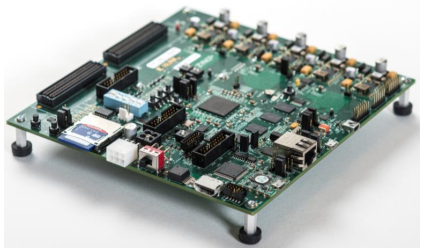
- 同一環境上で性能検証

全体のまとめ

MATLAB® SIMULINK® を活用することで

準備のハードル	・ハードウェアへのアクセスが大変	・サポートパッケージで Jetsonと連携／Xilinx Zynq開発環境構築
実装のハードル	・ハードウェアやインターフェイス周りの知識が必要 ・ハンドコーディング時にバグが混入	・開発したアルゴリズムと等価な GPU／HDLコードを自動生成
検証のハードル	・アルゴリズム開発環境と検証環境／言語が異なり、性能検証するのが難しい	・同一環境上で性能検証

GPU



FPGA/ASIC

MATLAB／Simulinkを用いることで“本質”に集中し、ラピットプロトタイピングが可能！

技術トレーニングコース（FPGA）

- MathWorks技術トレーニングコース
  - パブリックコース（東京、名古屋、大阪）、オンサイトコース、オンライン
- トレーニングコース受講のメリット
  - 短期間で豊富な機能を効率的に習得
  - 実用的な演習問題により、理解度向上
- 関連コース
  - HDL CoderによるHDLコード生成
  - MATLABとSimulinkによるXilinx Zynq SoCプログラミング（ZedBoard付き）

https://jp.mathworks.com/services/training/index.html

GPU／FPGA開発に向けた立ち上がりを強力にサポート

技術コンサルティングサービスのご紹介

- MathWorksの技術コンサルティングとは？
  - お客様の要求にあわせ、習熟度向上を目的としたアドバイザリーサービス

- 弊社製品をご利用頂く方の悩み（例：FPGA）
  - 手書きHDLでFPGA実装の経験はあるが、モデル記述に慣れない
  - Simulinkの経験はあるが、FPGAのことが良くわからない
  - SimulinkもFPGAも初めてで、何から始めればよいかわからない

- コンサルティングサービスの内容
  - お客様のスキルレベル、開発フェーズにあわせた効果的な技術習得の計画立案
  - 短期集中型トレーニングパッケージのご提供と、フォローアップサポート

プロジェクトの早期立ち上げが可能に!

サポートの紹介

- MathWorksのサポートエンジニア
  - お客様の環境への導入支援
  - 技術的な成立性確認をサポート
- 詰まりがちなポイントをサポート可能
  - お客様の開発環境構築
  - お客様で解決が難しいエラー
  - 弊社製品についての技術的ご相談

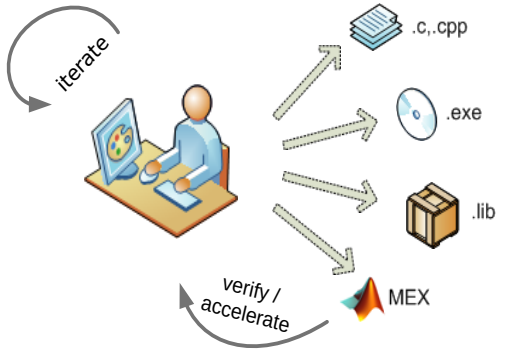
専門のエンジニアによる的確なサポート

GPU Coder関連製品

GPU Coderに
必須となります

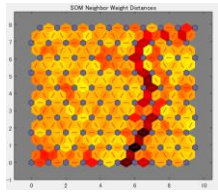
MATLAB Coder™

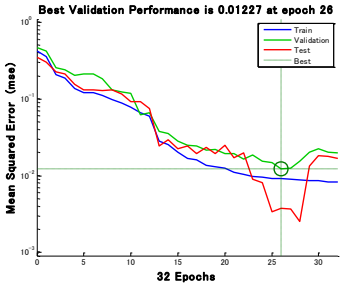
- MATLABプログラムからC/C++コードを生成
- MATLAB上で、
  アルゴリズム開発から
  実装までフローを統合

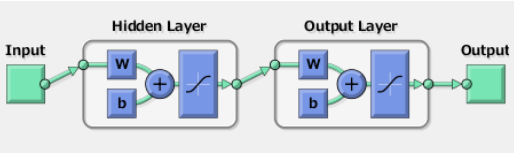


Deep Learning Toolbox

- ニューラルネットワークの構築、学習
- データフィッティング
- クラスタリング
- パターン認識
- 深層学習
- GPUによる計算の高速化

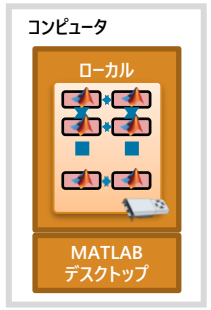






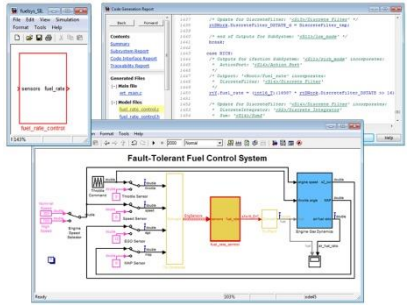
Parallel Computing Toolbox

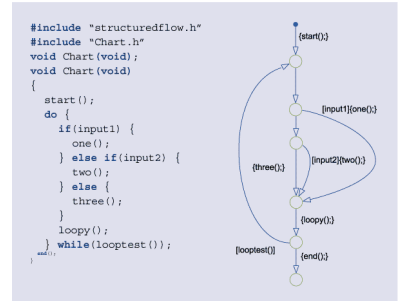
- MATLAB & Simulink と連携した並列処理
- 対話的な並列計算実行
- GPGPU による高速演算
- ジョブおよびタスクの制御



Embedded Coder®

- MATLABプログラム/Simulinkモデル
  から組込み用C/C++コードを自動生成

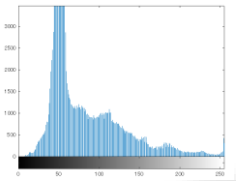




GPU Coder関連製品：画像処理・コンピュータービジョン

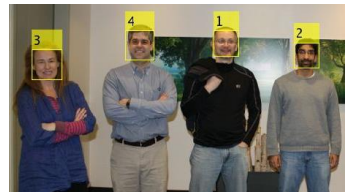
Image Processing Toolbox™

- コーナー、円検出
- 幾何学的変換
- 各種画像フィルタ処理
- レジストレーション（位置合せ）
- セグメンテーション（領域分割）
- 画像の領域の定量評価



Computer Vision System Toolbox™

- カメラキャリブレーション
- 特徴点・特徴量抽出
- 機械学習による物体認識
- 動画ストリーミング処理
- トラッキング
- ステレオビジョン・３D表示



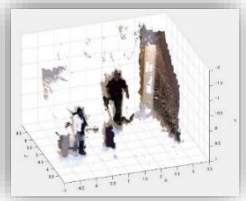
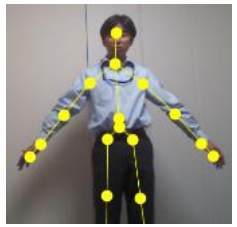


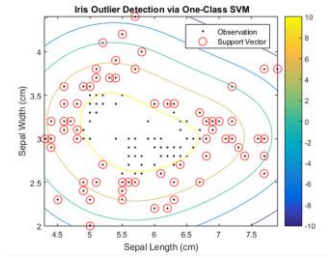
Image Acquisition Toolbox™

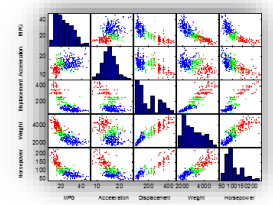
- デバイスから画像、動画直接取り込み
  - フレームグラバボード
  - DCAM, Camera Link®
  - GigE Vision®, Webカメラ
  - Microsoft® Kinect® for Windows®

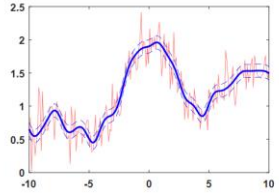


Statistics and Machine Learning Toolbox™

- 機械学習
- 多変量統計
- 確率分布
- 回帰と分散分析
- 実験計画
- 統計的工程管理







PILによる検証とプロファイリング

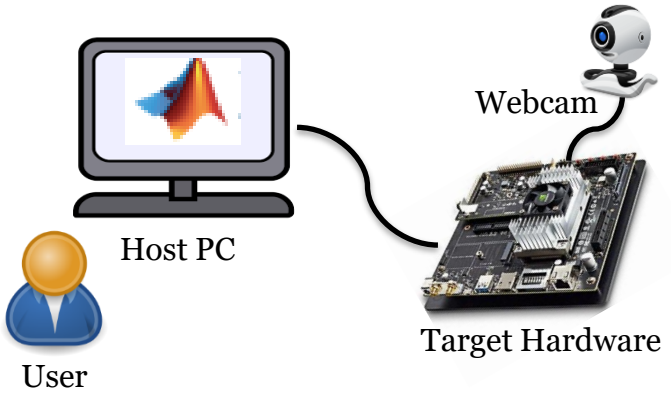
PIL実行には
Embedded Coderが必要です

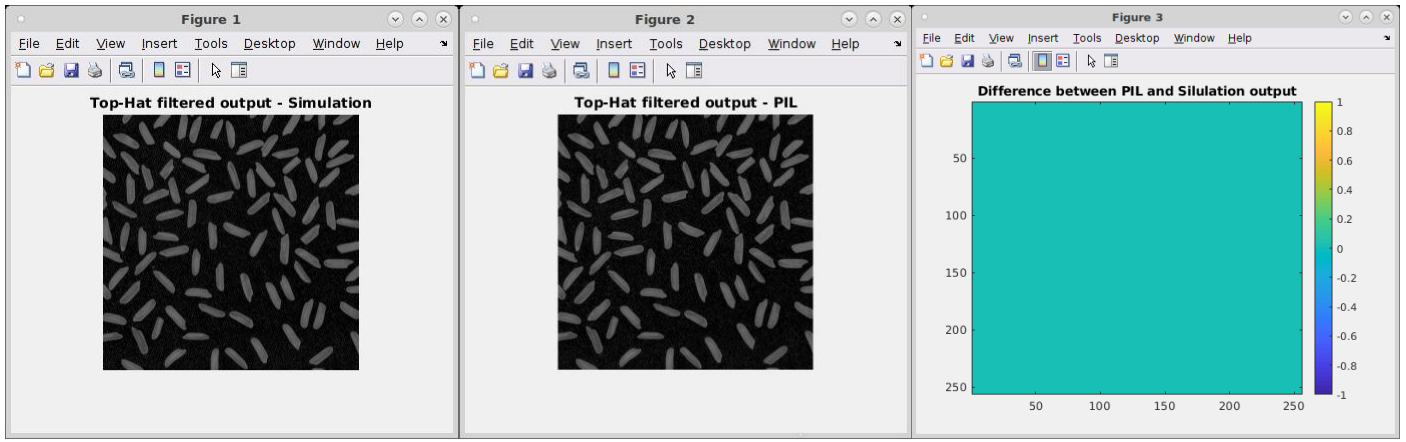
ハードウェア接続用オブジェクトの定義

コード生成用コンフィグレーション
(PIL利用時は静的ライブラリ選択)

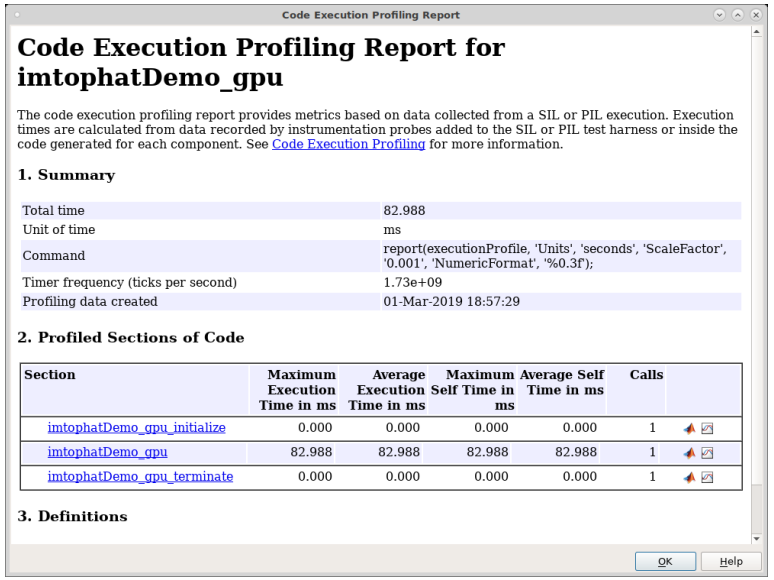
プロファイリングの有効化と
PILによる検証選択

```
hwobj = jetson('192.168.2.101', 'nvidia', 'nvidia');

cfg = coder.gpuConfig('lib', 'ecoder', true);


cfg.CodeExecutionProfiling = true;
cfg.VerificationMode = 'PIL';
```





ホスト上での実行結果　　ターゲット上(Jetson)での実行結果　　結果差分確認


Code Execution Profiling Report

Code Execution Profiling Report for imtophatDemo_gpu

The code execution profiling report provides metrics based on data collected from a SIL or PIL execution. Execution times are calculated from data recorded by instrumentation probes added to the SIL or PIL test harness or inside the code generated for each component. See Code Execution Profiling for more information.

1. Summary

Total time	82.988
Unit of time	ms
Command	report(executionProfile, 'Units', 'seconds', 'ScaleFactor', '0.001', 'NumericFormat', '%0.3f');
Timer frequency (ticks per second)	1.73e+09
Profiling data created	01-Mar-2019 18:57:29

2. Profiled Sections of Code

Section	Maximum Execution Time in ms	Average Execution Time in ms	Maximum Self Time in ms	Average Self Time in ms	Calls
imtophatDemo_gpu_initialize	0.000	0.000	0.000	0.000	1
imtophatDemo_gpu	82.988	82.988	82.988	82.988	1
imtophatDemo_gpu_terminate	0.000	0.000	0.000	0.000	1

3. Definitions

OK Help

プロファイリングレポートで実行時間を確認