

ソフトウェア技術者から見た FPGAの魅力と可能性

満田 賢一郎
(株)システム計画研究所／ISP

2018/01/19
第20回 東京FPGAカンファレンス2018with
プログラマブルデバイスプラザ
於 東京都立産業技術研究センター本部

はじめに:ISPの紹介

- 所属:(株)システム計画研究所／ISP
 - 1977年創業の独立系研究開発型のソフトウェア開発会社です。

はじめに:ISPの紹介

- 所属:(株)システム計画研究所／ISP
 - 1977年創業の独立系研究開発型のソフトウェア開発会社です。
- ISPの事業分野
 - 医療情報・Webアプリケーションシステム事業
 - 通信・ネットワーク・制御・宇宙システム事業
 - AIシステム事業
 - 画像処理システム事業
 - 独自の画像処理アルゴリズム開発:ROBUSKEY等(GPU/CUDA)

はじめに:ISPの紹介

- 所属:(株)システム計画研究所／ISP
 - 1977年創業の独立系研究開発型のソフトウェア開発会社です。
- ISPの事業分野
 - 医療情報・Webアプリケーションシステム事業
 - 通信・ネットワーク・制御・宇宙システム事業
 - AIシステム事業
 - 画像処理システム事業
 - 独自の画像処理アルゴリズム開発:ROBUSKEY等
 - アトラクションシステム開発:赤外線通信ボードの製作(FPGA込み)

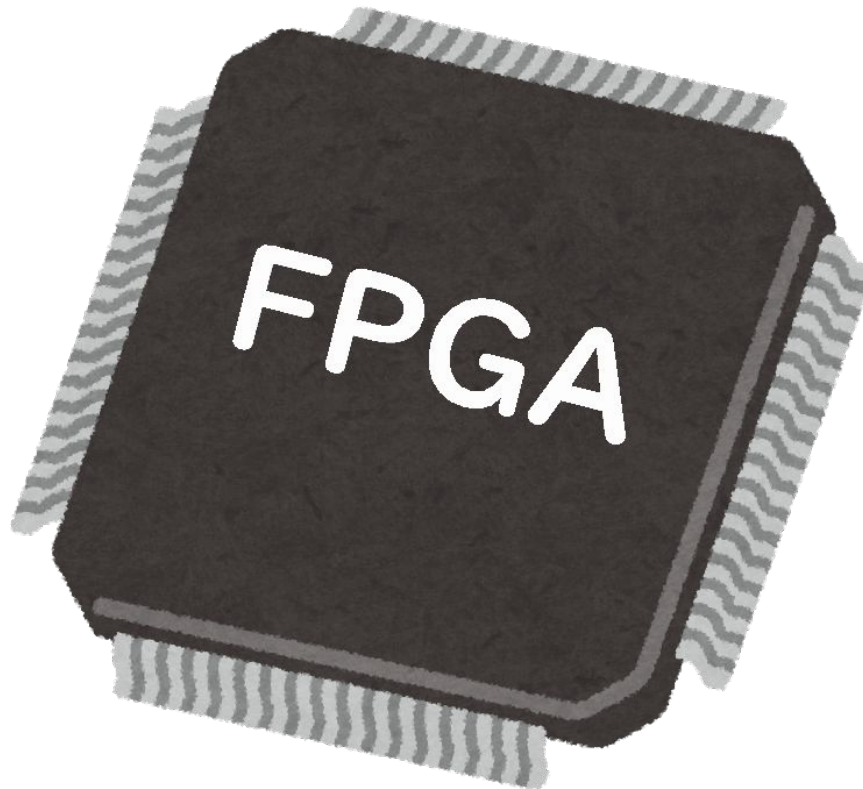
関心

FPGAでアルゴリズム(画像処理、DNN)を加速(アクセラレート)

ソフトウェア技術者から見たFPGA

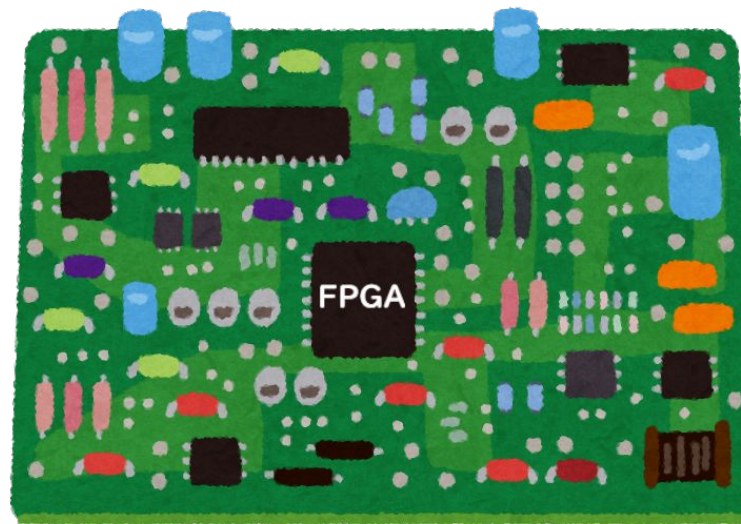
ソフトウェア技術者から見たFPGA(～2010年頃)

- 「動作を書き換えられる石でしょ？」
 - IC部品的一种、HW担当者の領分



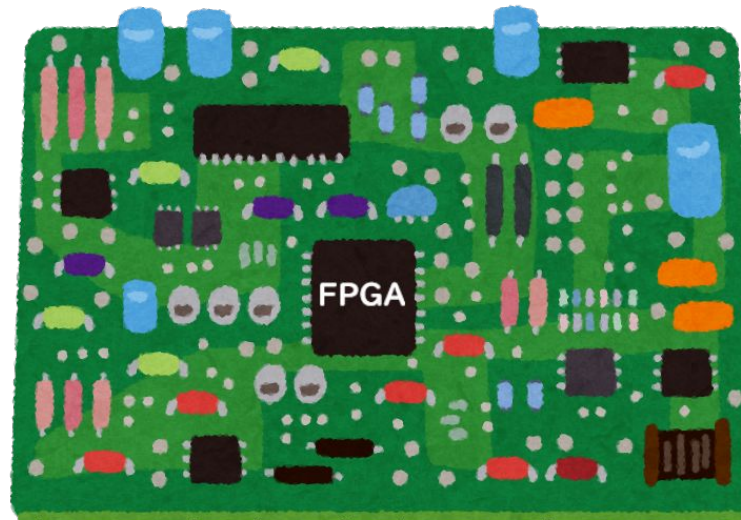
ソフトウェア技術者から見たFPGA(～2010年頃)

- 「動作を書き換えられる石でしょ？」
 - IC部品的一种、HW担当者の領分
- 「HWの試作品、評価ボードに載っているやつ」
 - HW担当者には便利な石らしい



ソフトウェア技術者から見たFPGA(～2010年頃)

- 「動作を書き換えられる石でしょ？」
 - IC部品的一种、HW担当者の領分
- 「HWの試作品、評価ボードに載っているやつ」
 - HW担当者には便利な石らしい



FPGAはソフトウェア技術者にとって「他人事」だった

ソフトウェア技術者から見たFPGA 2013年頃～

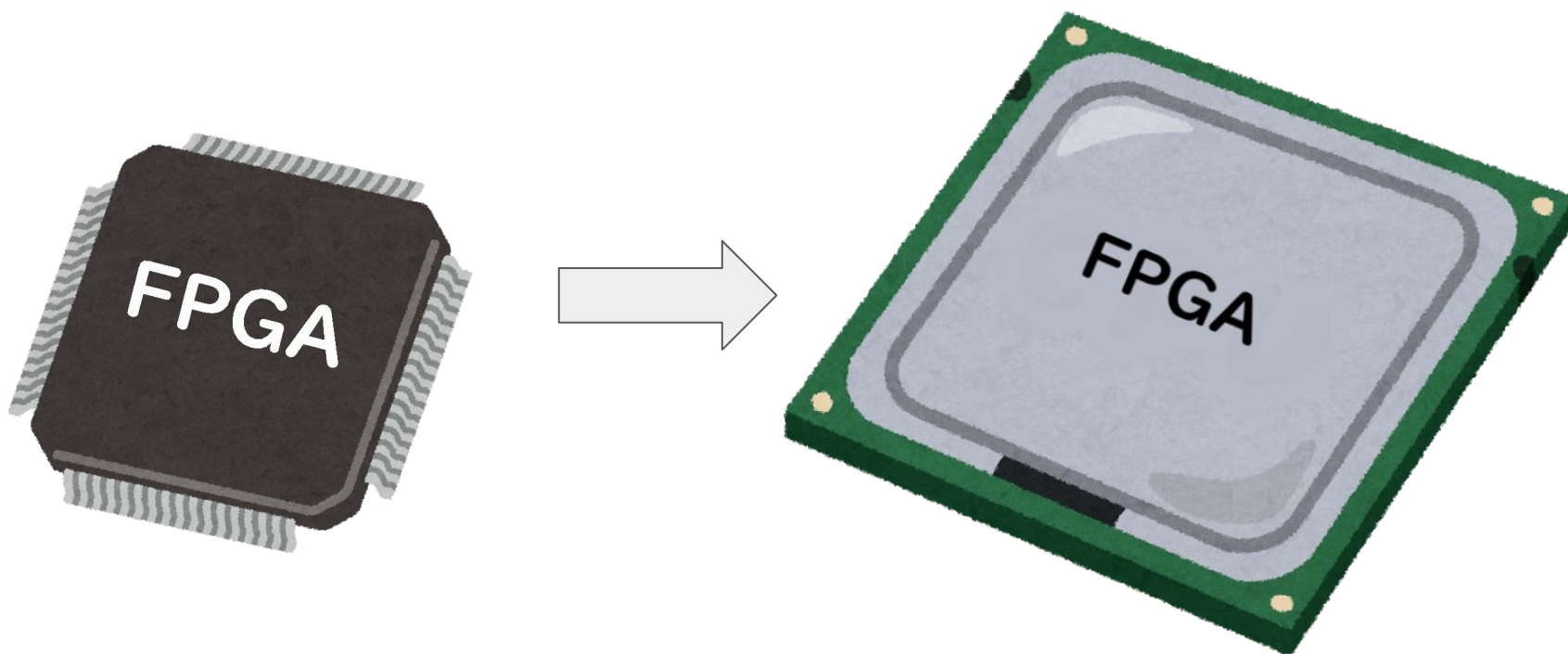
ソフトウェア技術者らもFPGAに注目

ソフトウェア技術者がFPGAに注目する3つの理由

ソフトウェア技術者らもFPGAに注目

ソフトウェア技術者がFPGAに注目する3つの理由

1. FPGAの進化(大容量化、FPGA SoCの登場)



ソフトウェア技術者らもFPGAに注目

ソフトウェア技術者がFPGAに注目する3つの理由

1. FPGAの進化（大容量化、FPGA SoCの登場）
2. 開発ツールの進化

ソフトウェア技術者らもFPGAに注目

ソフトウェア技術者がFPGAに注目する3つの理由

1. FPGAの進化(大容量化、FPGA SoCの登場)
2. 開発ツールの進化
3. 技術者を取り巻く環境の変化

FPGAに注目する3つの理由

FPGAの進化（大容量化、FPGA SoCの登場）

- 半導体プロセスルールの進化により、大容量化
 - より複雑な処理をFPGAにオフロードすることが可能に
 - DSPやメモリブロックなどハードマクロの充実
 - FPGA SoCの登場

FPGAに注目する3つの理由

FPGAの進化(大容量化、FPGA SoCの登場)

- 半導体プロセスルールの進化により、大容量化
 - より複雑な処理をFPGAにオフロードすることが可能に
 - DSPやメモリブロックなどハードマクロの充実
 - FPGA SoCの登場

FPGAが「アクセラレータ」として使用可能に

FPGAに注目する3つの理由

開発ツールの進化

- ビジュアルな環境・アシスト機能をもつ回路設計ツール
 - 直観的な操作で回路設計
- 高位合成(HLS)の実用化
 - 複雑なアルゴリズムを高い抽象度で記述可能に

FPGAに注目する3つの理由

開発ツールの進化

- ビジュアルな環境・アシスト機能をもつ回路設計ツール
 - 直観的な操作で回路設計
- 高位合成(HLS)の実用化
 - 複雑なアルゴリズムを高い抽象度で記述可能に

HDLが書けないアルゴリズム設計者やソフトウェア技術者でもFPGAの利用が可能に

FPGAに注目する3つの理由

技術者を取り巻く環境の変化

- 2014年～2017年のできごと
 - FPGA評価ボードの低価格化、開発ツールの無償化
 - MicrosoftやBaidu等クラウド企業がデータセンターにFPGA採用
 - ドワンゴによる、HW/FPGA技術者の募集
 - IntelによるAltera買収(XeonにFPGAが入る?)
 - AI系アルゴリズムのFPGA・HW実装への期待
 - AmazonによるFPGAインスタンスサービス開始

FPGAに注目する3つの理由

技術者を取り巻く環境の変化

- 2014年～2017年のできごと
 - FPGA評価ボードの低価格化、開発ツールの無償化
 - MicrosoftやBaidu等クラウド企業がデータセンターにFPGA採用
 - ドワンゴによる、HW/FPGA技術者の募集
 - IntelによるAltera買収(XeonにFPGAが入る?)
 - AI系アルゴリズムのFPGA・HW実装への期待
 - AmazonによるFPGAインスタンスサービス開始

FPGAはソフトウェア技術者にとっても「我が事」に

ソフトウェア技術者によるFPGA開発 (私の場合)

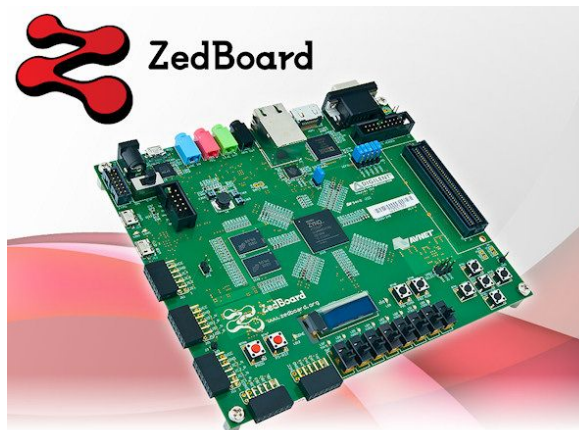
講演者の略歴

- 通信・ネットワーク・制御・宇宙システム事業の部署に配属
主に通信システム開発に従事
 - 無線通信装置の制御ファームウェア開発
 - Wi-Fiから4Gまで通信関連の規格調査を経験
 - 各種ドキュメントからの規格の調査を経験



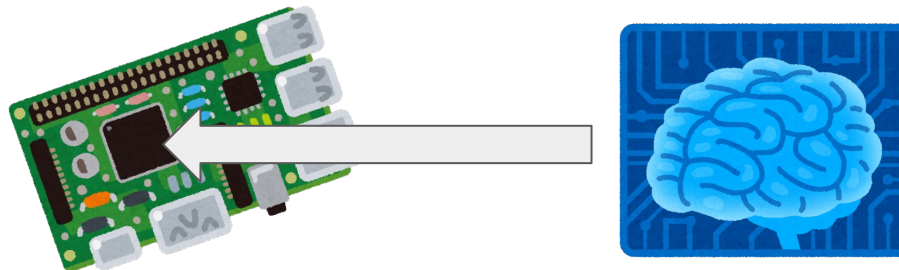
講演者の略歴

- 通信・ネットワーク・制御・宇宙システム事業の部署に配属
主に通信システム開発に従事
 - 無線通信装置の制御ファームウェア開発
 - Wi-Fiから4Gまで通信関連の規格調査を経験
- 2013年11月に画像処理システム事業の部署へ異動
 - 目的: FPGAで自社の画像処理アルゴリズムの高速化
 - AVNET製ZYNQ搭載ボードZedBoardを渡され、手探りで着手



講演者の略歴

- 通信・ネットワーク・制御・宇宙システム事業の部署に配属
主に通信システム開発に従事
 - 無線通信装置の制御ファームウェア開発
 - Wi-Fiから4Gまで通信関連の規格調査を経験
- 2013年11月に画像処理システム事業の部署へ異動
 - 目的:FPGAで自社の画像処理アルゴリズムの高速化
- 近年は画像処理システム事業、AIシステム事業に携わる
 - 主に、組み込みシステムの応用に従事



ZedBoardを手にした当時の私の状況

- HW/FPGAについては全くの初心者
 - RTL設計 (VHDL/VerilogHDL) 未経験
 - 組み込みLinux/ARM cpuは多少知っている程度



ZedBoardを手にした当時の私の状況

- HW/FPGAについては全くの初心者
- とりあえずZedBoardを使ってみよう！
 - USBにキーボード、マウスを接続
 - HDMIにディスプレイを接続
 - 付属のSDカードを挿入
 - 電源ON

画面に2羽のペンギンが映るだけでなにも起こらない...



ZedBoardを手にした当時の私の状況

- HW/FPGAについては全くの初心者
 - RTL設計 (VHDL/VerilogHDL) 未経験
 - 組み込みLinux/ARM cpuは多少知っている程度
- とりあえずZedBoardを使ってみよう！
 - USBにキーボード、マウスを接続
 - HDMIにディスプレイを接続
 - 付属のSDカードを挿入
 - 電源ON

画面に2羽のペンギンが映るだけでなにも起こらない...

まずはドキュメントを読んで、チュートリアルで勉強開始

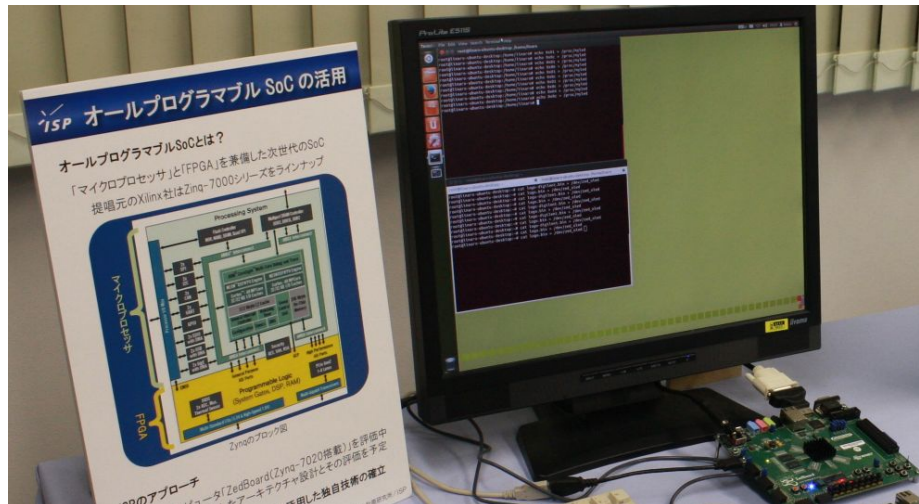
一般的なFPGA開発手法

- ハードウェア記述言語(HDL)を用いたRTL設計
 - RTL(register transfer level)設計:
回路の動作をレジスタ間のデータ転送とそれに対する論理演算の組合せで記述する設計手法
 - レジスタ:ここでは、任意の機能をもつ順序回路(記憶素子を含む論理回路)をブラックボックス化したもの
 - 順序回路:FPGAでは一般に入力や内部状態がクロック信号に同期して一斉に変化する同期式順序回路を指す
 - 記述に用いられる主なHDL: VHDL, Verilog HDL
 - HDLで記述したRTLを、ツールでFPGAのリソースにマッピング

ソフトウェア技術者でもRTL設計は頑張ればなんとかなる？
⇒とりあえず挑戦

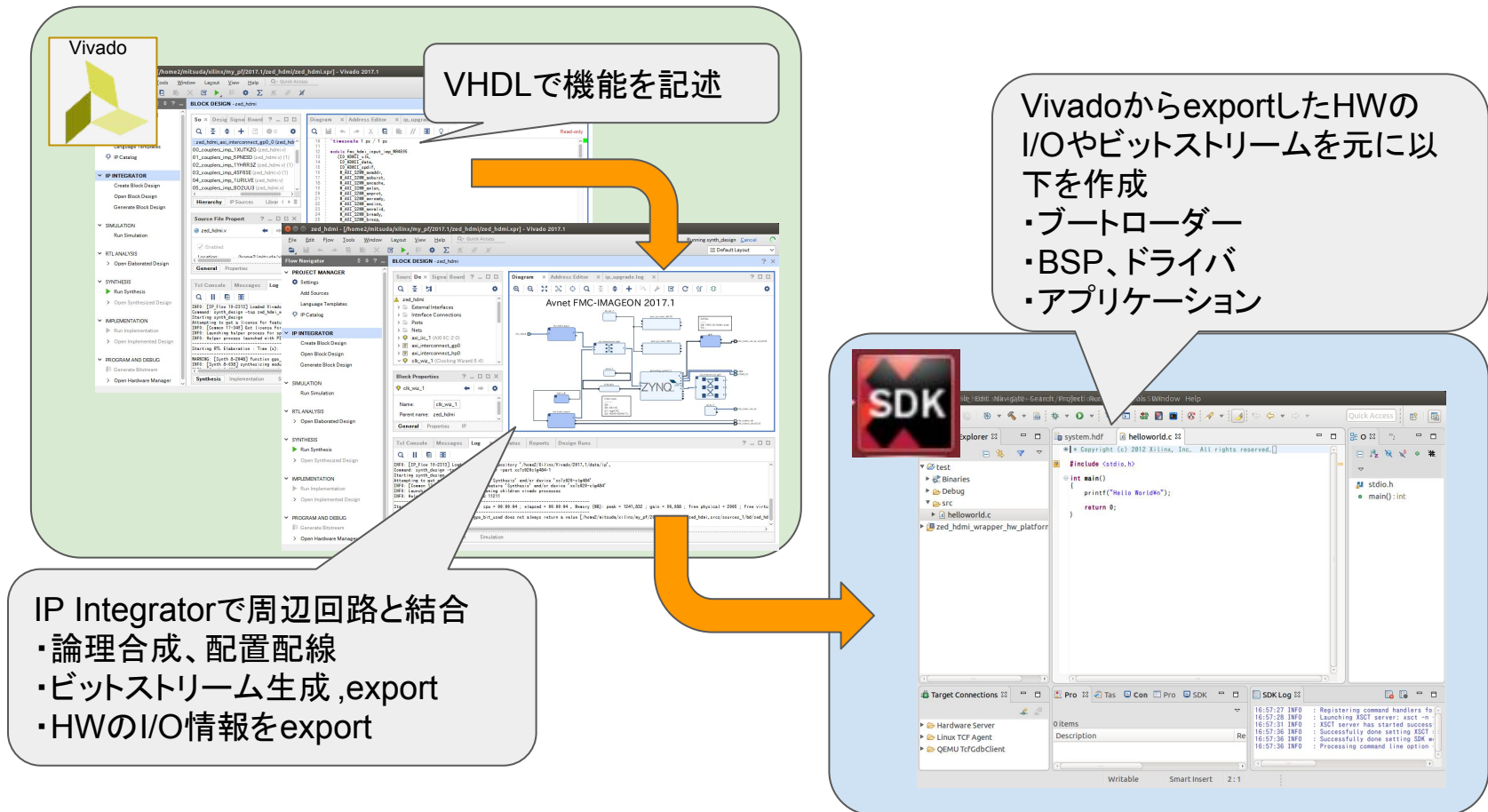
RTL設計をやってみた(自主学習プロジェクト)

- ターゲットボード: ZedBoard (ZYNQ-7000)
- VHDLの教科書とボードのマニュアルを読みながら実施
- 開発期間: 約一ヶ月(一人で)
- 成果: チュートリアルデザインを改造してLチカ
 - LED点滅(PWM)周期をubuntu LinuxのコンソールのGPIOから設定
 - OLEDに弊社ロゴを表示



RTL設計での開発フロー(概略)

■ XilinxのVivadoでHWを作成し、SDKで制御SWを作成



RTL設計をやってみて得た結論

- ターゲットボード: ZedBoard (ZYNQ-7000)
- VHDLの教科書とボードのマニュアルを読みながら実施
- 開発期間: 約一ヶ月(一人で)
- 成果: チュートリアルデザインを改造してLチカ



画像処理のアルゴリズム
実装までのギャップ
が埋められない.....

ソフトウェア技術者が

SWベースのアルゴリズムからRTL設計をするのは非現実的

高位合成(HLS)によるFPGA開発

■ 高位合成(HLS)を用いた設計

- 主なHLS言語: C/C++, OpenCL等SW開発言語ベース
 - Xilinx社 (Vivado HLS/SDx), Intel社 (FPGA SDK for OpenCL*)
 - Java, Python, Fortran、独自言語での開発ツールも存在
- HLS (High-Level synthesis) 設計:
HLS言語で記述されたアルゴリズムツールでRTLに変換
 - アルゴリズムでよく使われる機能はライブラリ提供 (数学関数など)
 - FPGA側でよく使われる機能もライブラリ提供 (固定小数点型など)
 - RTLに必要な「クロック」や「リセット」などの信号は完全に隠蔽
 - FPGAでの処理並列化やリソース割当は独自の「指示子」で指定

SWでの実装と同様、「ソフト」にFPGA開発ができる？

⇒とりあえず挑戦

HLSでROBUSKEYアルゴリズムのFPGA化

ROBUSKEY:ISP独自の高品位クロマキー合成アルゴリズム

- 2014/8/E～10/Eで開発
 - 担当:私 & HW技術者
- InterBEEに参考出品
- 実装した機能
 - グリーンバック対応
 - HDMI入出力(1080p)
 - 入力はビデオカメラ
 - 出力はPCディスプレイ
- Xilinx社製 ZC706で動作
 - 弊社「技ラボ」にて報告

Inter BEE 2014出展報告

Posted by: Tadaharu INOUE in お知らせ © 2014/11/27 ☎ 868 Views

ISPは、11/19日(水)～11/21日(金)の3日間、幕張メッセにて開催された「Inter BEE 2014」(国際放送機器展)に出展いたしました。たくさんの方のご来場、誠にありがとうございました。

ISPとしては初の単独出展で、以下の出展を行いました。

高精度クロマキー合成エンジン「ROBUSKEY」

- FPGA版 FHD 60fps リアルタイムROBUSKEY
- CUDA版 FHD 30fps [リアルタイムROBUSKEY](#)
- Tegra K1で動作するROBUSKEY
- プラグイン製品 [ROBUSKEY for Video](#)

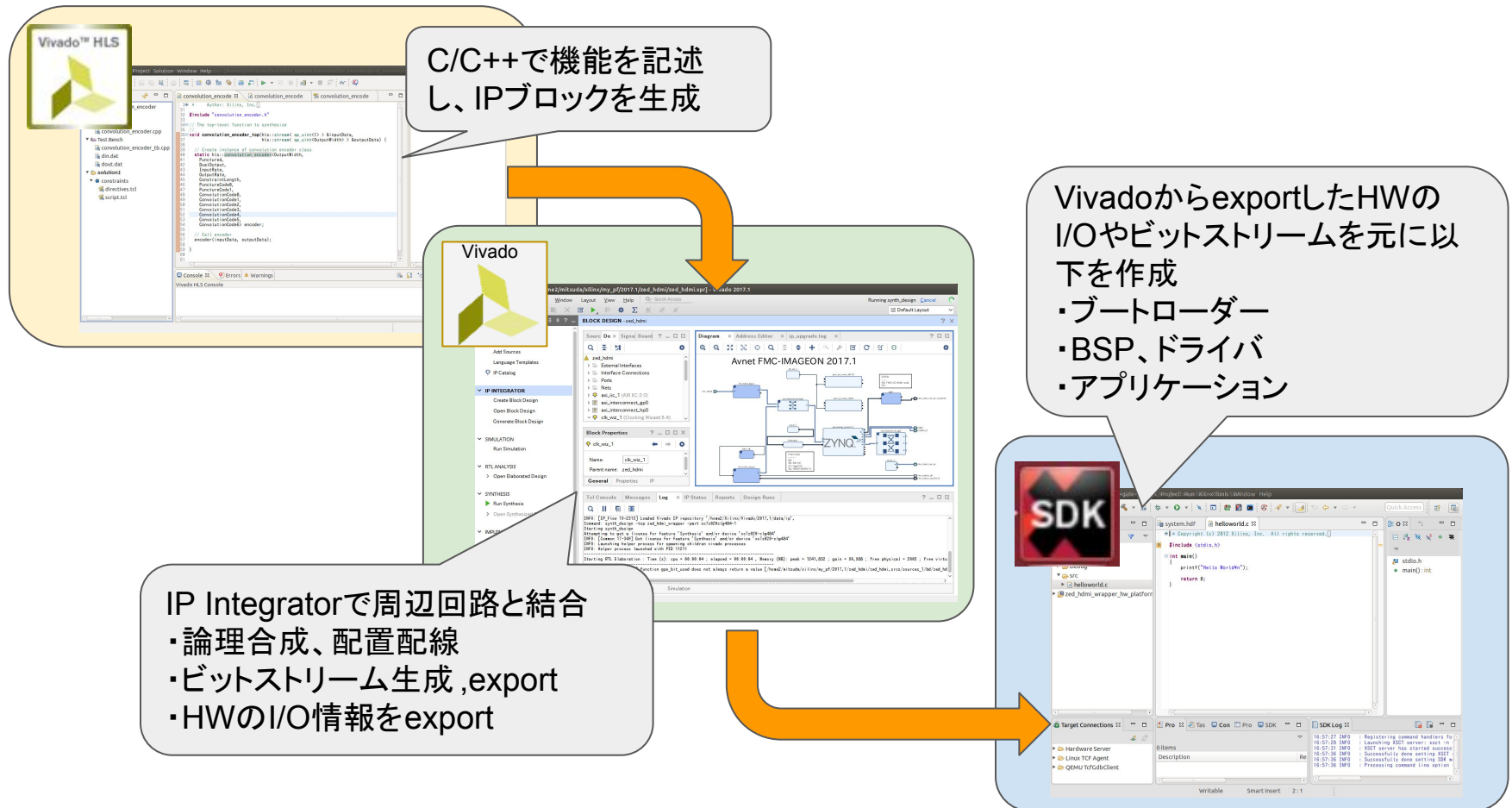
肌理(キメ)が残る肌補正処理エンジン「珠肌」

- CUDA版 FHD 30fps [リアルタイム珠肌](#)
- 珠肌プラグイン プロトタイプ



HLSを使った場合の開発フロー(概略)

■ Vivado_HLSとVivadoでHWを作成し、SDKで制御SWを作成



HLS可能なC/C++には様々なお約束が

- HLS可能なC/C++には以下のような制限がある
 - サポートする文法の制限
 - OSがサポートするシステムコールなどは未サポート(例外: memcpy())
 - 関数ポインタは使用不可
 - 再帰関数は使用不可
 - 動的なリソース操作(確保や削除)は未サポート(mallocやnew)
 - HLSで特別な意味を持つ記述
 - HLSはC言語の「関数」単位で実行(C++のクラス定義は不可)
 - HLS対象の関数では引数がIPのインターフェースとなる
 - memcpy()はメモリブロックからのバースト転送となる
 - 配列はBlock RAMに割り当て
 - 各種「指示子(ディレクティブ)」で動作やリソース割り当てを指定

HLSでROBUSKEYデモ作成して得た結論

- HLSはFPGA開発を「ソフト」にするツール
 - 特にソフトウェアの機能をHW化(IP化)する場合に有効
 - ただしリソースや速度面で良い設計になるとは限らない
 - ソフトウェア技術者がFPGAを使うことへの障壁を引き下げた



ソフトウェア技術者が

HLSでアクセラレータを作ることは比較的現実的

高位合成があればFPGA開発は簡単？

ROBUSKEYデモ作成で行った作業

■ 全体の作業フロー

- a. Vivado IPIで対象アルゴリズムを組み込むプラットフォームの設計
- b. PC上でSW実装のアルゴリズムから、HW化する処理を抽出
- c. Vivado HLS上で、抽出した処理の移植(処理の最適化も実施)
- d. Vivado HLSでアルゴリズムのcsim/CoSimで確認しIP化
- e. Vivado IPIで作成したIPを結合し合成、インプリメンテーション
- f. SDKでアプリケーションSW、BSP、Bootの作成

周辺回路やIPといった部品から作成し、それらを統合して最後にSWを書くという点で作業工程は「ボトムアップ型」

HLSでのアルゴリズムFPGA化は作業のごく一部

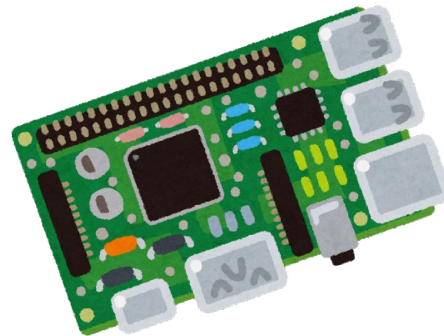
- 高位合成に関する作業はIP作成部分のみ
 - a. Vivado IPIで対象アルゴリズムを組み込むプラットフォームの設計
 - b. PC上でSW実装のアルゴリズムから、HW化する処理を抽出
 - c. Vivado HLS上で、抽出した処理の移植(処理の最適化も実施)
 - d. Vivado HLSでアルゴリズムのcsim/CoSimで確認しIP化
 - e. Vivado IPIで作成したIPを結合し合成、インプリメンテーション
 - f. SDKでアプリケーションSW、BSP、Bootの作成

作業的にはc,dで1ヶ月、その他a,b,e,fで計1ヶ月

そもそも、高位合成でのIPを作成以外にもソフトウェア技術者にとってはハードルの高い作業が沢山ある。

ソフトウェア技術者が感じる、高位合成以外の課題

- プラットフォームの扱いが難しい
 - 作成したIPを試用・評価するためのプラットフォームをどうするか？
 - ROBUSKEYのプラットフォーム作成にも2週間程度試行錯誤



ソフトウェア技術者が感じた高位合成以外の課題

- プラットフォームの扱いが難しい
 - 作成したIPを試用・評価するためのプラットフォームをどうするか？
- 複数のツールを使うなど、開発手順が煩雑
 - Vivado HLS→Vivado (IPI) →SDKと3種類のツールが必要
 - 各ツールごとにプロジェクトを管理する必要がある



ソフトウェア技術者が感じた高位合成以外の課題

- プラットフォームの扱いが難しい
 - 作成したIPを試用・評価するためのプラットフォームをどうするか？
- 複数のツールを使うなど、開発手順が煩雑
 - Vivado HLS→Vivado (IPI)→SDKと3種類のツールが必要
- 「ボトムアップ型」の開発手順に戸惑う
 - SWの設計ではシステム全体から詳細への「トップダウン型」



アルゴリズムのアクセラレーションだけに集中したいのに！

最新ツールで、もっと「ソフト」にFPGA開発

最新のツールによる課題解決

- アルゴリズムアクセラレータの開発に集中できない問題
 - プラットフォームの扱いが難しい
 - 複数のツールを使うなど、開発手順が煩雑
 - 「ボトムアップ型」の開発手順に戸惑う

これらの問題を解決するさまざまなツールが登場

ツールの例

- SDSoC™ , SDAccel™ (Xilinx)
- Visual System Integrator (SystemView)

今回はSDSoCを紹介

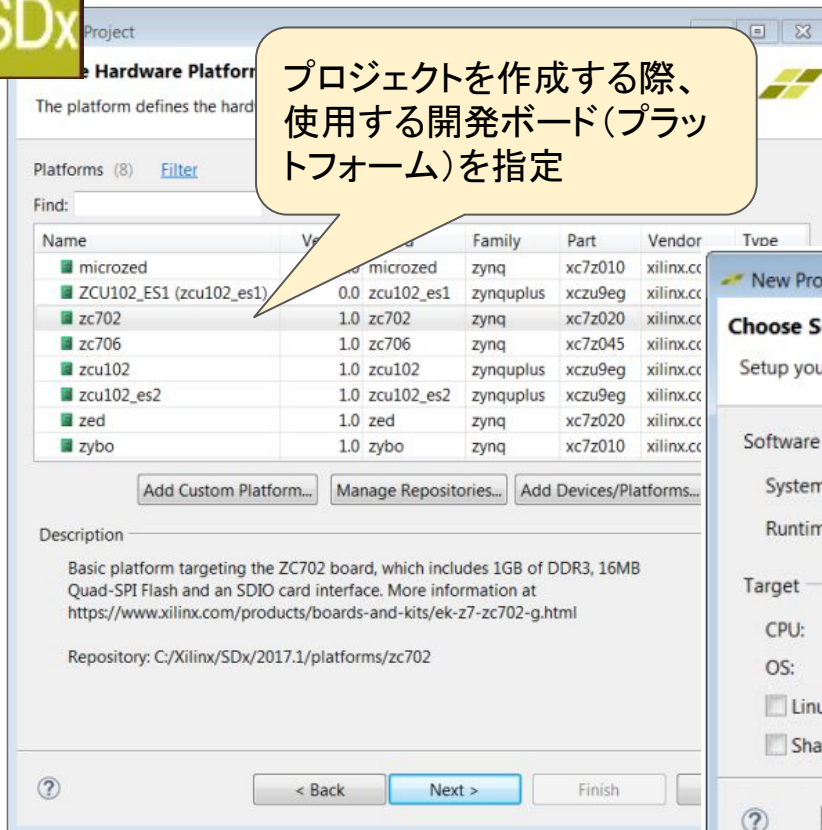


SDSoCの特徴

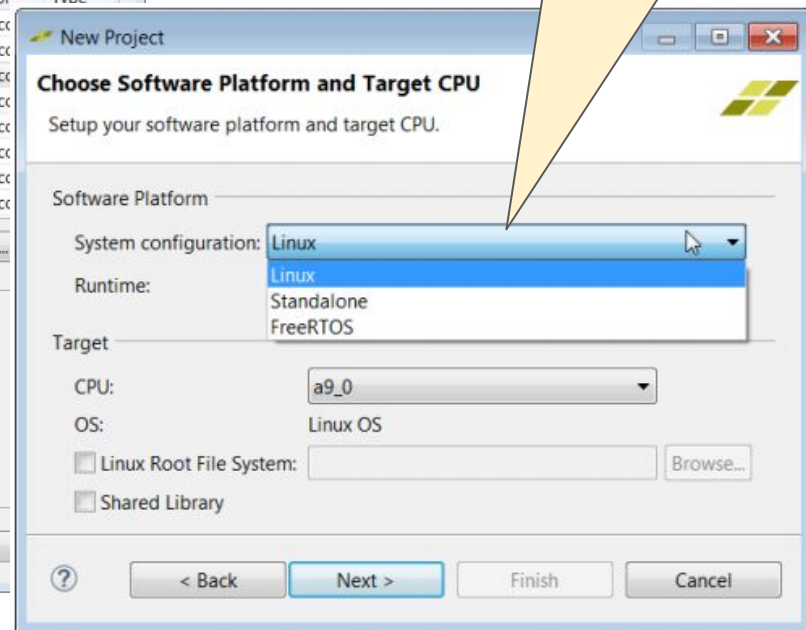
- あらかじめ提供される開発プラットフォーム
 - サードパーティも含め、ボードメーカーがSDSoCプラットフォームを提供
 - プラットフォームのカスタマイズは自由(マニュアル:ug1146)
- SW/HW(HLS)両方のソースコードを扱える統一環境
 - ソフトウェア技術者にも馴染みやすいEclipse IDEベースの開発環境
- アルゴリズムのアクセラレーションを意識した機能
 - ZYNQのSWアプリケーションの機能をHLSによりFPGA化
 - 内蔵プロファイラでアクセラレータ化関数を決定(HW関数)
 - SDSoC上でHW関数を指定⇒DMA転送回路やドライバを自動生成

SDSoCの特徴～プラットフォームの指定

■ あらかじめ提供される開発プラットフォーム



プロジェクトを作成する際、
使用する開発ボード(プラットフォーム)を指定



アプリケーションが動作する
OSを指定

SDSoCの特徴～SW/HW統合された開発環境

- SW/HW(HLS)両方のソースコードを扱える統一環境
 - Eclipse IDEベースの開発環境でアプリケーションをSWとして記述

SDx workspace - SDx - test2/src/mmult.cpp - Xilinx SDx

test2

- test2
- Binaries
- Archives
- Includes
- Release
- src
 - sd_card_prebuilt
 - madd.cpp
 - main.cpp
 - mmult.cpp
 - mmultadd.h
 - stdio.h
 - stdlib.h
 - mmult(float[], float[], float[]): void [H]
 - mmultadd.h
- project.sdx

```
2 * (c) Copyright 2013 - 2016 Xilinx, Inc. All rights reserved.
39
40 #include <stdio.h>
41 #include <stdlib.h>
42 #include "mmultadd.h"
43
44 //
45 *
46 * Design principles to achieve II = 1
47 * 1. Stream data into local RAM for inputs (multiple access required)
48 * 2. Partition local RAMs into M/2 sub-arrays for fully parallel access (dual-port read)
49 * 3. Pipeline the dot-product loop, to fully unroll it
50 * 4. Separate multiply-accumulate in inner loop to force two FP operators
51 *
52 //
53 void mmult (float A[N*N], float B[N*N], float C[N*N])
54 {
55     float Abuf[N][N], Bbuf[N][N];
56     #pragma HLS array_partition variable=Abuf block_factor=16 dim=2
57     #pragma HLS array_partition variable=Bbuf block_factor=16 dim=1
58
59     for(int i=0; i<N; i++) {
60         for(int j=0; j<N; j++) {
61             #pragma HLS PIPELINE
62             Abuf[i][j] = A[i * N + j];
63             Bbuf[j][i] = B[i * N + j];
64         }
65     }
66
67     for (int i = 0; i < N; i++) {
68         for (int j = 0; j < N; j++) {
69             #pragma HLS PIPELINE
70             float result = 0;
71             for (int k = 0; k < N; k++) {
72                 float term = Abuf[i][k] * Bbuf[k][j];
73                 result += term;
74             }
75             C[i * N + j] = result;
76         }
77     }
78 }
```

eclipseベースの統合環境
C/C++, OpenCLでシステムを
記述

- ・HW化する関数の選択
- ・アプリケーションのビルド
(SW部、HW部とも)

mmult

- Compilation Log (06 7 2017 03:08)
- HLS Report (06 7 2017 03:08)

madd

- Compilation Log (06 7 2017 03:07)
- HLS Report (06 7 2017 03:07)

Console

```
06:57:06 INFO : Registering command handlers for SDK TCF services
06:57:06 INFO : Launching XSCOT server: xscot -n -interactive /home2/mitsuda/xilinx/workspac
06:57:10 INFO : Validating SDSoC License...
06:57:13 INFO : XSCOT server has started successfully.
06:57:13 INFO : Successfully done setting XSCOT server connection channel
06:57:14 INFO : License available for SDSoC
06:57:14 INFO : Validating SDAccel License...
06:57:14 INFO : License for SDAccel is unavailable
06:57:26 INFO : Successfully done setting SDK workspace
```

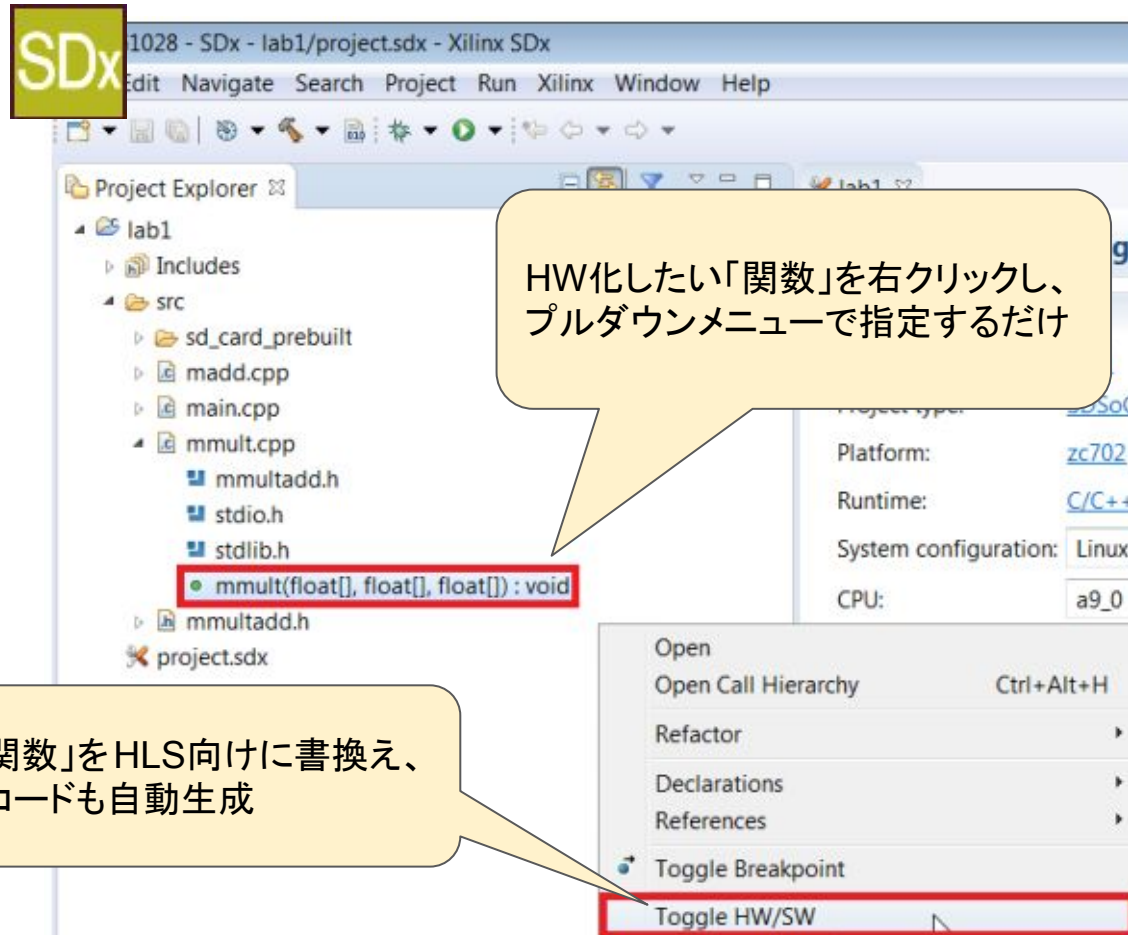
Target Connections

- Hardware Server
- Linux TCF Agent
- QEMU TcfGdbClient

Writable Smart Insert 53:11

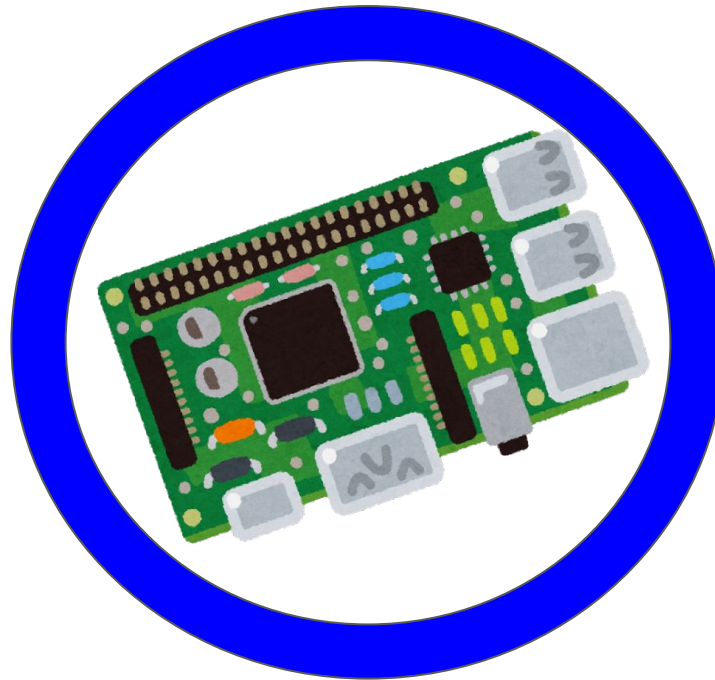
SDSoCの特徴～HW化対象の選択

■ アルゴリズムのアクセラレーションを意識した機能



SDSoC導入の効果

- プラットフォームの扱いは簡単になったか？
 - サポート済みボードzc702を使っている分にはあまり困らない！



SDSoC導入の効果

- プラットフォームの扱いは簡単になったか？
 - サポート済みボードZC702を使っている分にはあまり困らない！
- 開発手順の煩雑さは解消されたか？
 - 各ツールのレポートはSDSoCから参照可能
 - Vivado HLSはHW化対象の処理をDebugするために使用
 - ベースとなるSWの作成にはPC上のC/C++開発環境の方が便利



SDSoC導入の効果

- プラットフォームの扱いは簡単になったか？
 - サポート済みボードZC702を使っている分にはあまり困らない！
- 開発手順の煩雑さは解消されたか？
 - 各ツールのレポートはSDSoCから参照可能
 - Vivado HLSはHW化対象の処理をDebugするために使用
 - ベースとなるSWの作成にはPC上のC/C++開発環境の方が便利
- SWと同様の「トップダウン型」の開発フローでOKか？
 - 個人的には「トップダウン型」で行けていると思う

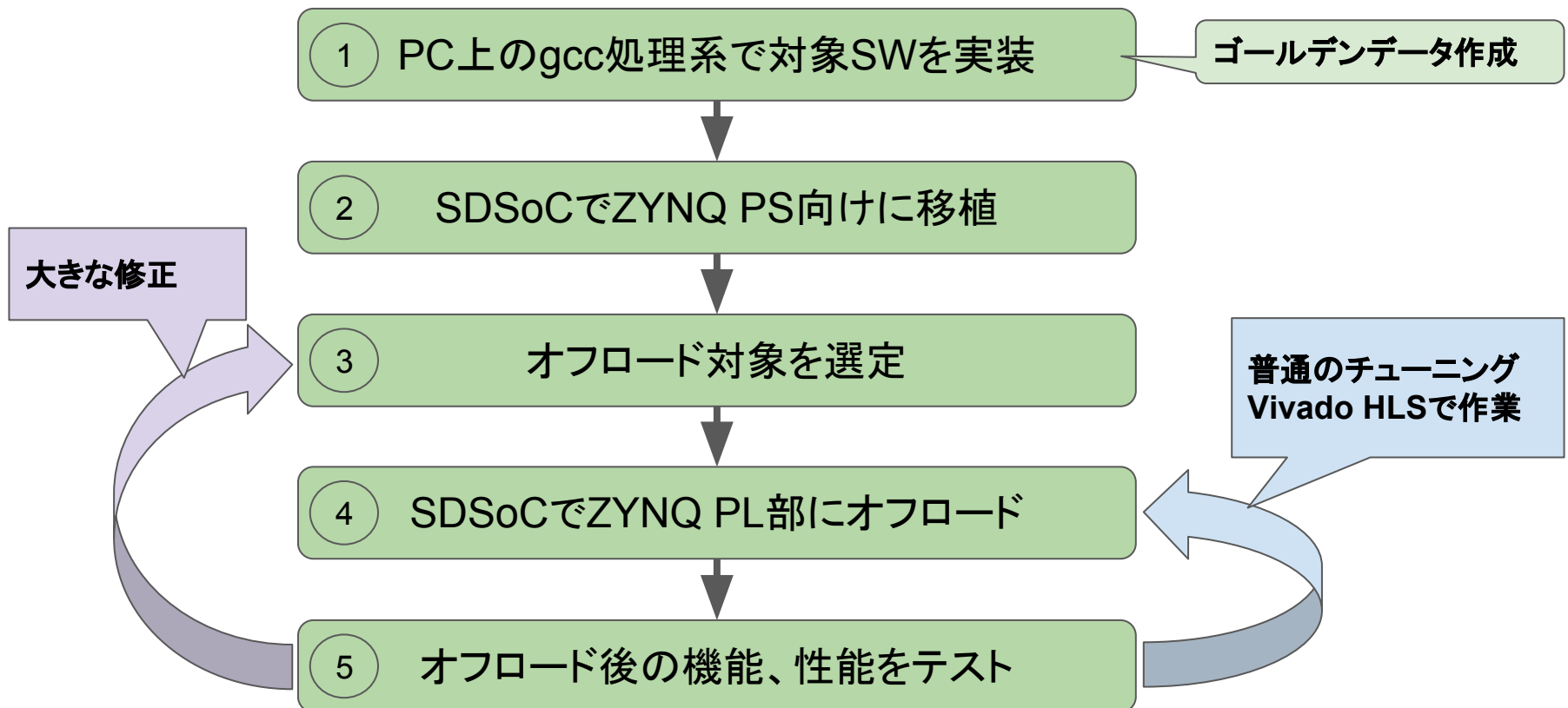
【率直な感想】

「SW実装されたアルゴリズムをHW化する」ための良いツール

ソフトウェア技術者にもFPGAは
使いこなせるのか？

SDSoCでの作業フローの実際

■ SW実装済みアルゴリズムをアクセラレーションする場合



SWからのアクセラレータ開発で考慮すること

- 大前提: 元のアルゴリズムの機能を担保すること
 - ただし、実現不可な機能や本質的でない機能は整理する
 - ダイナミックに確保されるメモリの必要量を決めて固定化
 - SWの柔軟性に資するパラメータなどは整理してなるべく固定化
- アクセラレータによる「目標性能」をどうするか
 - HW化によって達成すべき性能は決まっているか？
 - 決まっている場合、その目標は妥当か？
 - 決まっていない場合はどうするか？

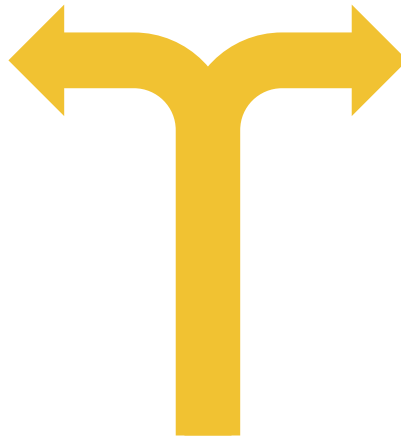
実際の作業としては、アルゴリズムの機能を常に確認しながら、性能のチューニングを行う

SWからHLSでアクセラレータ開発を行った例

- 元のSW:MNIST手書き文字認識(独自ネットワークを使用)
- SWから高位合成では①をベースに③を開発する作業

①元のSW実装

- VS2013で実装
- C++11
- IntelCPUに最適化
- 高い保守性、柔軟性



③高位合成対応の実装

- Vivado HLS2016.2
- C++0x
- HWに最適化(性能面)

②本来のアルゴリズム

- 画像をベースに数値処理

SWからHLSでアクセラレータ開発 Step1

- SW実装を高位合成が可能なソースコードに書き換える
 - 元のSW実装を「お約束」にしたがって修正
 - この例では、ここで浮動小数点⇒固定小数点化も実施
 - 高位合成の結果 (Vivado HLSの見積もり値)

	BRAM	DSP	FF	LUT
消費量	400	24	17350	28100
リソース	280	220	106400	53200

- BRAM消費量がオーバーのためFPGAで実行不可
- この実装でZynq PS部での動作速度: 約150fps
 - PSは667MHzで動作

SWからHLSでアクセラレータ開発 Step2

- Step1のコードを、元のアルゴリズムベースで還元
 - データとパラメータの保持方法などを再検討
 - 演算タスクの粒度を見直し、消費リソースを最小化

	BRAM	DSP	FF	LUT
消費量	400→145	24→6	17350→4450	28100→5377
リソース	280	220	106400	53200

- これでターゲットのリソースに収まった⇒FPGAで実行可能
- この実装でZynq PS部での動作速度: 約150fps→270fps
- この実装でZynq PL部での動作速度: 約150fps
 - PSは667MHz, PLは100MHzで動作
- アルゴリズムベースの実装に還元した結果、SWも高速化

SWからHLSでアクセラレータ開発 Step3

- Step2のコードから、並列化による速度向上
 - サイクル数をベースに各演算タスクの並列度と目標性能を設定
 - リソースの余裕をみて、微調整を繰り返す

	BRAM	DSP	FF	LUT
消費量	145→219	6→55	4450→9771	5377→14257
リソース	280	220	106400	53200

- 演算並列化の結果、DSPの消費の増加率が最も高い
- この実装でZynq PS部での動作速度: 約270fps→173fps
- この実装でZynq PL部での動作速度: 約150fps→7400fps
 - PSは667MHz, PLは100MHzで動作
- HW向けに最適化したコードは元のSW実装と全く別物

HLSで良いアクセラレータを開発するために必要な事

- 森岡澄夫氏の言葉を引用（FPGAマガジンNo.10 pp.9より）

『回路設計では、達成すべき速度性能・クロック周波数・面積が明確な数値目標として決まっているのが普通で、回路設計者は機能だけでなく性能をいかに達成するかに腐心しています。』

- HLSでは性能面のチューニングが直接的ではない
 - C/C++言語では「クロック」や「面積」「並列度」を記述できない
 - HLSでは指示子を用いて性能面のコントロールを行う
 - 指示子の意図が反映されるかは処理系次第

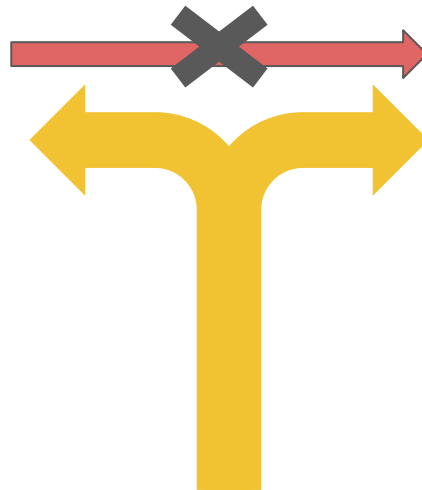
HWの知識を前提に、HLSのテクニックに熟知することが重要

SWから良いアクセラレータを開発を作る道筋

- SWから高位合成では①をベースに②に立ち返り、改めて③を開発する作業とすべき

①元のSW実装

- CPUに最適化
- 保守性、柔軟性
- そこそこの性能



③高位合成対応の実装

- HWに最適化(性能面)
- 設計された機能
- 設計された性能

②本来のアルゴリズム

- 機能(処理内容)
- 性能(速度、効率)

ソフトウェア技術者は高位合成を使いこなせるのか？

- ソフトウェア技術者が高位合成/FPGAを使う際の課題
 - よく「高位合成はHWを知らないと使えない」と言われる
 - ここでの「HW」とは、まずは以下の2つと考えられる
 1. FPGAおよびFPGAを構成する要素技術とその特性
 2. 非ノイマン型のアーキテクチャに関する理解
- なぜ、「HW」に関する2つの事を知る必要があるのか？

これらを理解しないと、HLSで良い設計ができないから

ソフトウェア技術者に必要なHWの感覚

■ SWには無い「物理リソース」の感覚

- 例1) HLSで記述した関数の引数 W,X,Y,Zは「どんなIFにすべきか？」

```
void dut(din_t W, din_t X, dout_t *Y, dout_t Z[5])
```

- データポート？
- メモリ(FIFO/BRAM)?
- プロトコルは？(ハンドシェークの有無)



ソフトウェア技術者に必要なHWの感覚

■ SWには無い「物理リソース」の感覚

- 例1) HLSで記述した関数の引数 W,X,Y,Zは「どんなIFにすべきか？」

```
void dut(din_t W, din_t X, dout_t *Y, dout_t Z[5])
```

■ FPGA内部での配置と面積(使用するリソース量)の感覚

- 例2) 階層化された関数は、どのように「配置されるのか」?

```
dut(...){  
    A(...); //sub関数A  
    B(...); //sub関数B  
    C(...); //sub関数C  
    D(...); //sub関数D  
    ...  
}
```

ソフトウェア技術者に必要なHWの感覚

■ SWには無い「物理リソース」の感覚

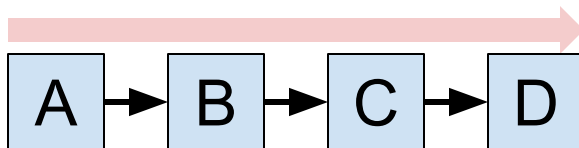
- 例1) HLSで記述した関数の引数 W,X,Y,Zは「どんなIFにすべきか？」

```
void dut(din_t W, din_t X, dout_t *Y, dout_t Z[5])
```

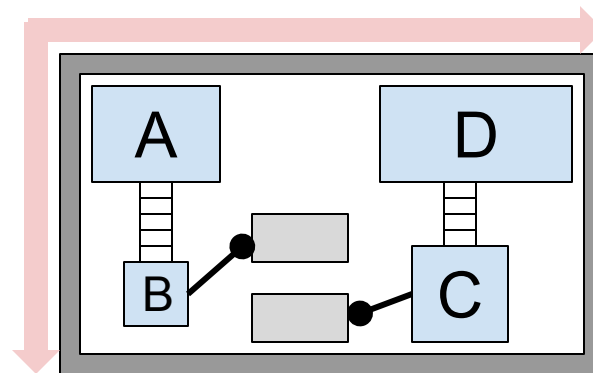
■ FPGA内部での配置と面積(使用するリソース量)の感覚

- 例2) 階層化された関数は、どのように「配置されるのか」?

SWは時間軸で展開



HWは空間的に展開



「ソフトウェア技術者は処理を時間軸で、HW技術者は処理を空間に配置」

SWエンジニアに必要なHWの感覚

- SWには無い「物理リソース」の感覚

- 例1) HLSで記述した関数の引数 W,X,Y,Zは「どんなIFにすべきか？」

```
void dut(din_t W, din_t X, dout_t *Y, dout_t Z[5])
```

- FPGA内部での配置と面積(使用するリソース量)の感覚

- 処理をソフトウェア技術者は時間軸で、HW技術者は空間に配置

⇒高位合成は、新しいプログラミング言語を学ぶのとは違う

ソフトウェア技術者が高位合成を使うために学びたい事

- はじめて高位合成を経験したSW技術者へのヒアリング
 - 弊社の中堅SWエンジニア
 - 画像処理アプリケーションの一部をオフロードする
 - タスク分割やデータは事前に検討済み
 - SDSoC 2016.2を使用

ソフトウェア技術者が高位合成を使うために学びたい事

- はじめて高位合成を経験したエンジニアへのヒアリング
 - 最も困った点「自分が書いたものが、どうなっているか分からない」
 - ソースコードで意図したとおりに、HWが出来ているか判断できない
 - ビルド時のメッセージが大量で複雑
 - 各種レポートを見るのに慣れるまで時間がかかる
 - 高位合成の結果、リソース消費が問題となった場合の対策
 - 「要因は何か？」
 - 「どこをどう直すのか？」



ソフトウェア技術者が高位合成を使うために学びたい事

- はじめて高位合成を経験したエンジニアへのヒアリング
 - 最も困った点「自分が書いたものが、どうなっているか分からない」
 - ソースコードで意図したとおりに、HWが出来ているか判断できない
 - ビルド時のメッセージが大量で複雑
 - 各種レポートを見るのに慣れるまで時間がかかる
 - 高位合成の結果、リソース消費が問題となった場合の対策
 - 「要因は何か？」
 - 「どこをどう直すのか？」
- 事前に学んでおきたい点は以下に集約
 - FPGAを構成する要素技術や動作の特性
 - 非ノイマン型のアーキテクチャに関する理解

ソフトウェア技術者が高位合成を使うために学びたい事

- はじめて高位合成を経験したエンジニアへのヒアリング
 - 最も困った点「自分が書いたものが、どうなっているか分からない」
 - ソースコードで意図したとおりに、HWが出来ているか判断できない
 - 高位合成の結果、リソース消費が問題となった場合、
 - 「要因は何か?」「どこをどう直すのか?」
 - あまり気にならなかった点
 - ディレクティブを含む、HLS独特のコード記述法
 - 違和感があったところをリファクタリングしてみたら、エラーになった。
 - エラーになる理由は不明だが、「そういうものか」と自分を納得させた。
 - ツールのlook&feel(SDSoc)

ソフトウェア技術者は「おまじない」を書く事には慣れている

ソフトウェア技術者にHWを学ぶ場はあるか？

- 一般的に理解を進めるための理想的な環境
 - 基礎についての講義ができる先生がいる
 - 全く知らない概念を、本やwebだけ独学するのは厳しい
 - 先生は必ずしも身の回りに居なくても良い
 - 座学で学んだ知識を実践で体感できる
 - 自由に評価ボード、ツールを使用して体感することで技術が身につく
 - 体感したことを共感し確認できる仲間・メンターがいる
 - 思った通りうまく行ったことは自慢したい
 - うまく行かなかったことは相談し、解決したい
 - 身に着けた知識・技術を活用できること
 - 自分が先生になるのも良い
 - もちろん、仕事に活かせるのが一番良い



ソフトウェア技術者がHWを学ぶためのヒント

- Xilinx公式のトレーニングを受講する
 - 無料版のVivado_HLxやSDSoCにはTCが無い。でも行く価値あり！
- Xilinxが発信する一次情報を探す
 - ドキュメント、Webのデザインハブ、フォーラムの回答
- 本などで勉強
 - FPGAマガジン No.10, No.14, No.6など CQ出版
 - 天野英晴 編(2016)『FPGAの原理と構成』オーム社
 - 森岡澄夫(2002)『HDLによる高性能デジタル回路設計』CQ出版
 - 森岡澄夫(2012)『LSI/FPGAの回路アーキテクチャ設計法』CQ出版
- 各種イベント、技術者交流会での情報交換
 - 「FPGAエクストリームコンピューティング」など

まとめ：ソフトウェア技術者もFPGAを使いたい

FPGAの魅力

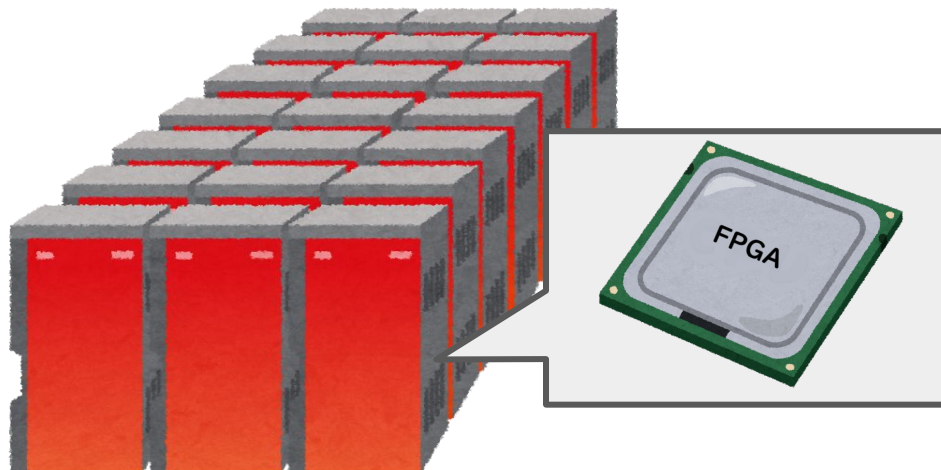
- アルゴリズムアクセラレータとしての活用
 - 画像処理（加工や特徴量抽出など）
 - AI（DNNや機械学習）処理の加速
 - エッジ側の処理（組み込み向けアクセラレータ）
 - クラウド側の処理（ビッグデータ処理）



まとめ：ソフトウェア技術者もFPGAを使いたい

FPGAの魅力

- アルゴリズムアクセラレータとしての活用
- クラウドサービスでの活用
 - Amazon等、複数のクラウドサービスがFPGAインスタンスを提供
 - クラウドサービス系のソフトウェア技術者が注目
 - 「モノづくり」が不要なため、ソフトウェア企業のみで参入可能



まとめ：より良いサービス／システムをつくるために

- エッジからクラウドまで、アルゴリズムアクセラレータが必要
 - エッジ側：リアルタイム性
 - クラウド側：大容量データ処理

FPGAの柔軟性を活かしたアクセラレータ開発に期待

まとめ：より良いサービス／システムをつくるために

- エッジからクラウドまで、アルゴリズムアクセラレータが必要
 - FPGAの柔軟性を活かしたアクセラレータ開発に期待
- 特に、高度な数理的アルゴリズムの実装
 - 画像処理(加工や特徴量抽出)
 - AI(DNNや機械学習)処理

HLSの高い抽象度での記述を活かした効率的な開発に期待

まとめ：より良いサービス／システムをつくるために

- エッジからクラウドまで、アルゴリズムアクセラレータが必要
 - FPGAの柔軟性を活かしたアクセラレータ開発に期待
- 高度な数理的アルゴリズムのアクセラレーションが必要
 - HLSの高い抽象度での記述を活かした効率的な開発に期待
- より良いサービス／システムをつくるために

アルゴリズム開発者(ソフトウェア技術者)とFPGA・ハードウェア技術者の協力に期待

