# ソフトウェアエンジニアでもできる、ハードウェアをやわらかく使う方法

満田 賢一郎
（株）システム計画研究所／ISP
2017/08/24, 25
SWEST 19
於 下呂温泉 水明館

# 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
    - SWEST18 s5a：弊社　上島による講演
      「技術者が知っておきたいDeep Learningの基礎と組込みでの利用」

# 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）


FPGAでアルゴリズム(画像処理、DNN)を加速（アクセラレート）

# アクセラレータ開発の必要性

# 組込み系でも、アルゴリズムの加速は必要か？

## 組込み系への需要の変化

- IoT化：エッジデバイスで取得したり扱うデータが増加
  - カメラなどビジョン系デバイスによる情報収集
  - 収集した情報の加工
    - データクレンジング
    - プライバシー保護処理
    - データ圧縮

# 組込み系でも、アルゴリズムの加速は必要か？

## 組込み系への需要の変化

- IoT化：エッジデバイスで取得したり扱うデータが増加
- AI化：エッジデバイスでの高度な判断処理の要請
  - 取得したデータからの特徴量抽出
  - 複雑なアルゴリズムによる判断処理
    - 機械学習
    - ディープラーニング


特に、リアルタイム性を求められる組込み系では影響大

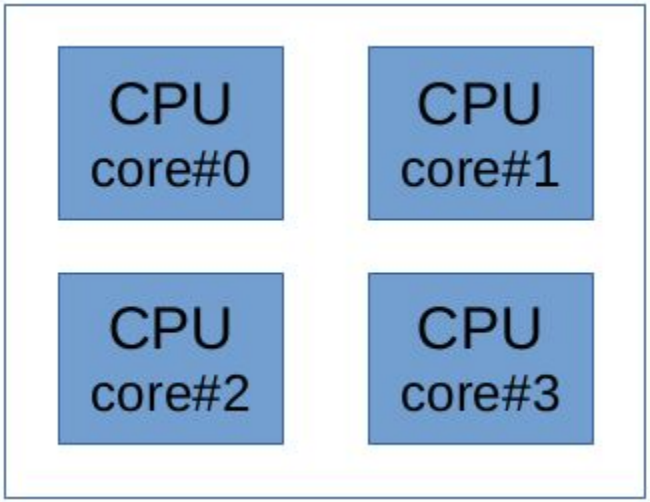
# 処理量の増加への対策……HWの強化

HWを強化して重いSWベースアルゴリズムを処理可能に

- 対策１：CPUコア数を増やす
- 対策２：CPU＋GPUなど専用HWを追加したSoC
- 対策３：CPU＋再構成可能なHWを追加したSoC

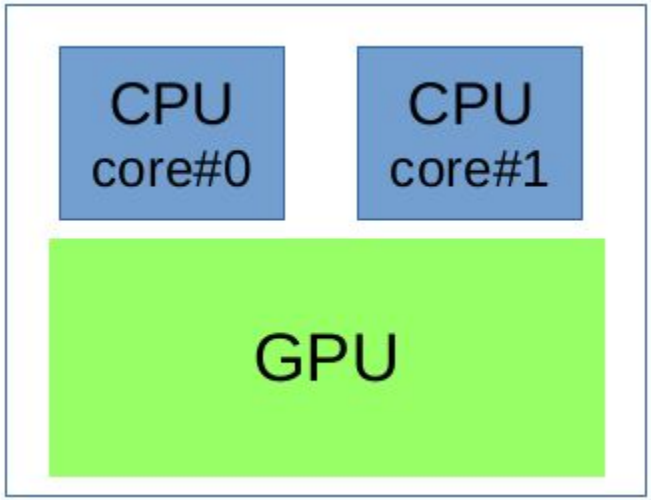
# 対策1：CPUコアを増やす

- マルチコアCPU
  - CPUコアを2個以上搭載
    - 近年ではわりと一般的なHWの高性能化手法
    - 各CPUコアに処理を振り分け、並列実行
    - ただし、並列処理での効率化には限界（アムダールの法則）



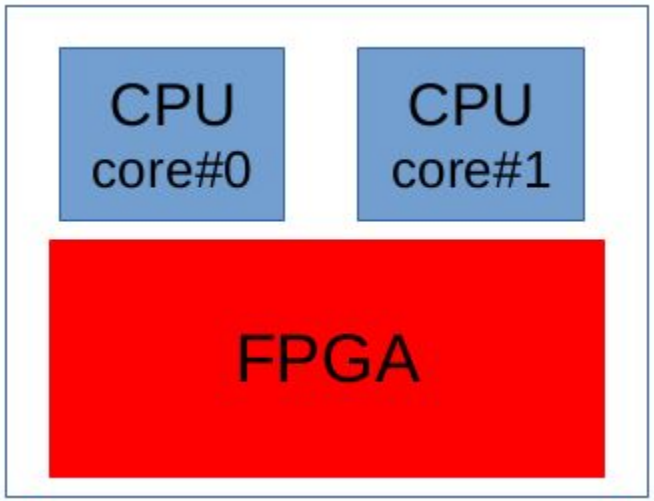
# 対策2：専用HWを追加したSoC

- SoC (System on a chip)
  - CPUコアと応用目的の専用機能を統合
    - 近年、GPU（GPGPU対応）を搭載した製品が登場
    - 数十～数百個のGPUコアを搭載し、SIMD／SIMTによる並列計算
    - CPUに比べて汎用性に欠ける



# 対策３：再構成可能HWを追加したSoC

- SoC (System on a chip)
  - CPUコアと、機能に応じて再構成可能なHWを統合
    - 近年、FPGA（field-programmable gate array）を搭載した製品が登場
    - 対象アルゴリズムを論理回路に変換しFPGA部に配置
    - SWエンジニアが対象アルゴリズムを論理回路として設計？



# 強化されたHWにアルゴリズムを実装

これらにアルゴリズムを実装するのはSWエンジニアの仕事

- マルチコアCPU
  - 一般的な開発言語で実装可能
  - OSやライブラリによりサポートあり

# 強化されたHWにアルゴリズムを実装

これらにアルゴリズムを実装するのはSWエンジニアの仕事

- マルチコアCPU
- CPU＋GPUなど専用HWを追加したSoC
  - 一般的な開発言語に近い言語で実装可能
  - ライブラリによるサポートあり

# 強化されたHWにアルゴリズムを実装

これらにアルゴリズムを実装するのはSWエンジニアの仕事

- マルチコアCPU
- CPU＋GPUなど専用HWを追加したSoC
- CPU＋再構成可能なHWを追加したSoC
  - FPGAの開発言語？ライブラリ？
  - そもそもFPGAはハードウェアなのでは……？

# FPGA SoCの紹介

# 身近なFPGA SoCデバイスの紹介

FPGA SoCはToppersプロジェクトでもサポート

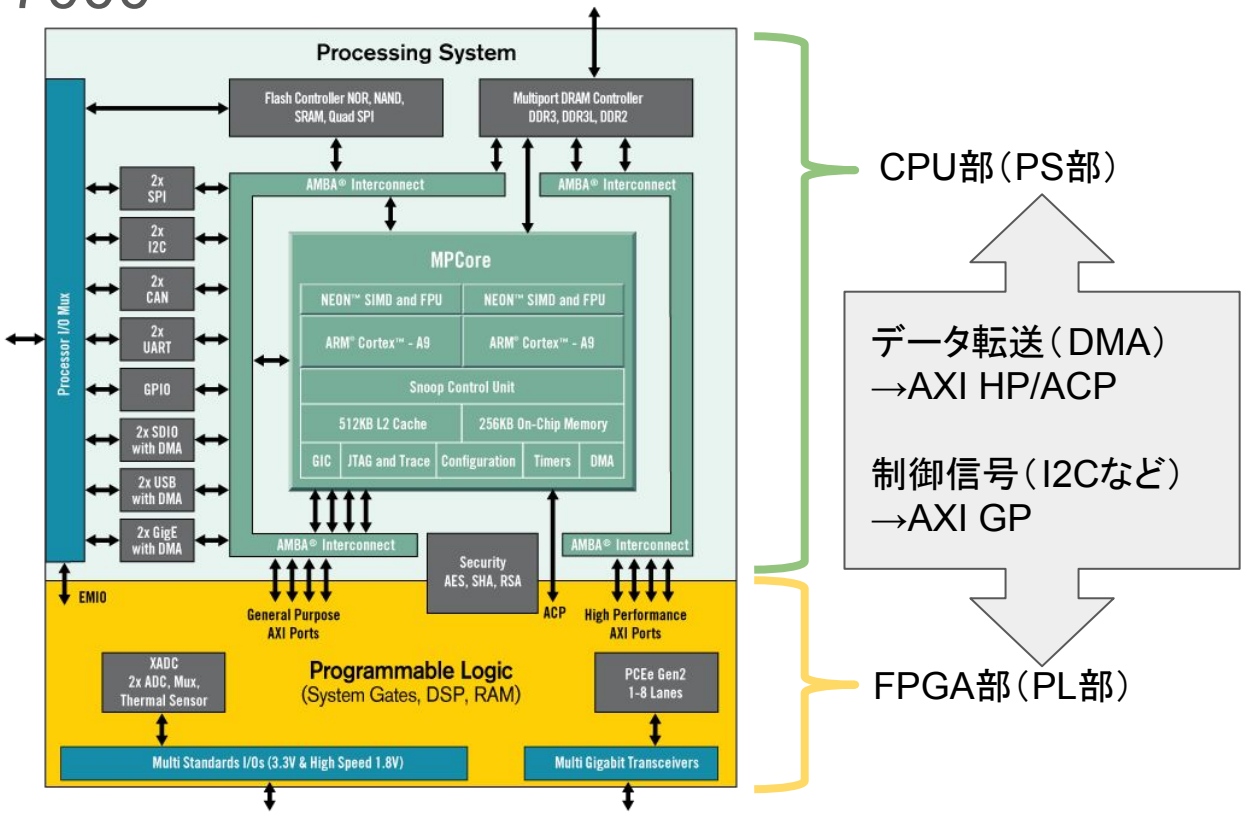
- TOPPERS/ASP3
  - Xilinx ZYNQ-7000 (Cortex-A9+FPGA)
- TOPPERS/FMPカーネル
  - Xilinx ZYNQ Ultrascale+ MPSoC (Cortex-A53&R5+FPGA)
  - Xilinx ZYNQ-7000 (Cortex-A9+FPGA)
  - Intel（旧Altera） Cyclone V SoC (Cortex-A9+FPGA)
- SafeG
  - Intel（旧Altera） Cyclone V SoC開発キット
  - Avnet ZedBoard (Xilinx ZYNQ-7000)

# FPGA SoCとは

- 例）Xilinx ZYNQ-7000

ひとつのチップの中に、
・CPU（Proccessing System）
・FPGA（Programmable Logic）
を配置。

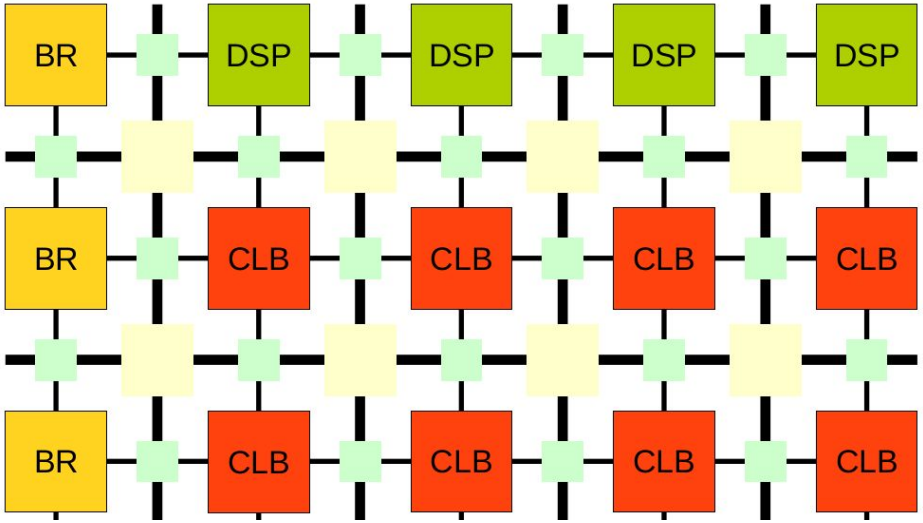
CPU部とFPGA部の接続
・32bit幅のAXI GPポート
・64bit幅のAXI HPポート
・64bit幅のAXI ACP（アクセラレータコヒーレンシー）ポート
・割り込みポート




引用https://japan.xilinx.com/content/dam/xilinx/imgs/block-diagrams/zynq-mp-core-dual.png

# FPGAとは？

- FPGA内部構造：各リソースを格子状に配置




CLB：Configurable Logic Block
- LUT（Look up table）とFF（flip-flop）からなる論理演算素子

BR:Block RAM, 36/18Kbit単位で扱えるオンチップRAM

DSP:Digital Signal Processer, 積和演算専用回路

# FPGAを使ったアクセラレータ開発の実際

# 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

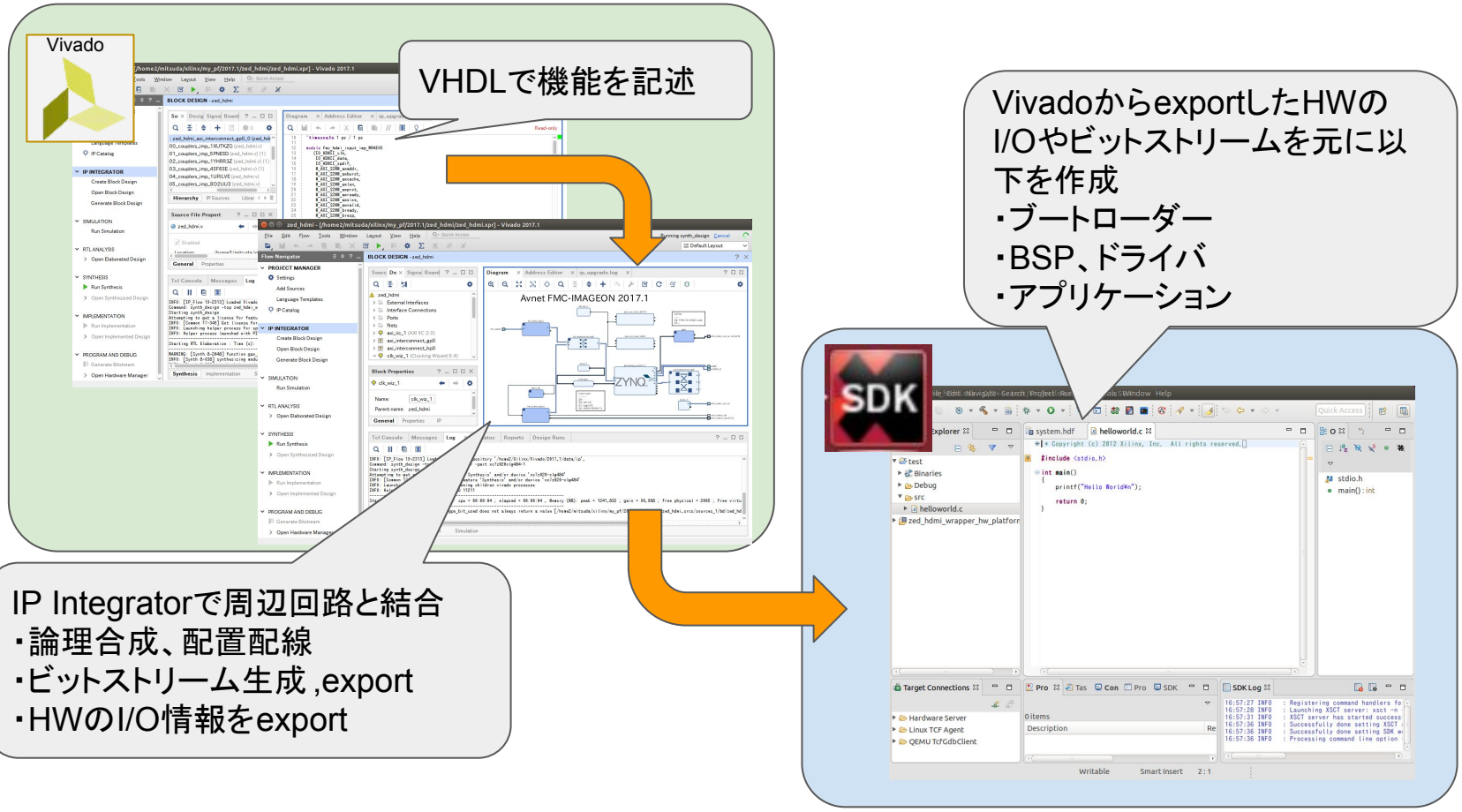
SWエンジニアでRTL設計は頑張ればなんとかなる？
⇒とりあえず挑戦

# 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：チュートリアルの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：チュートリアルのデザインを改造してLチカ


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」

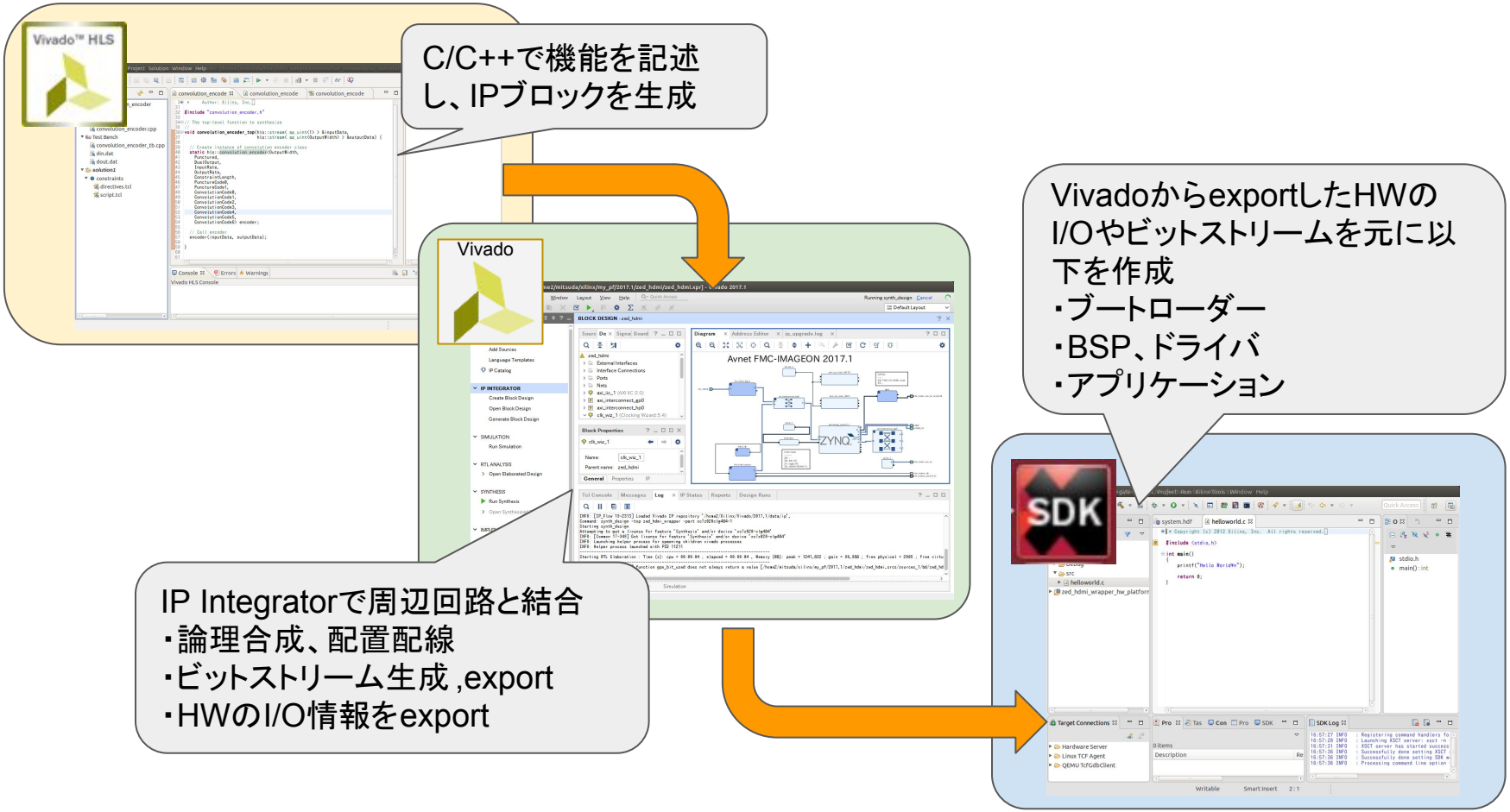
- FPGA版　FHD 60fps リアルタイムROBUSKEY
- CUDA版　FHD 30fps リアルタイムROBUSKEY
- Tegra K1で動作するROBUSKEY
- プラグイン製品 ROBUSKEY for Video

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

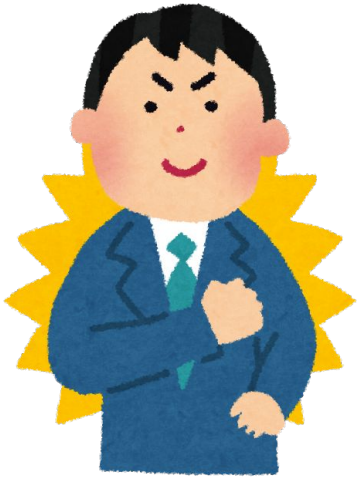


# HLS可能なC/C++には様々なお約束が

- 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
  - SWエンジニアがFPGAを使うことへの障壁を引き下げた



SWエンジニアが

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

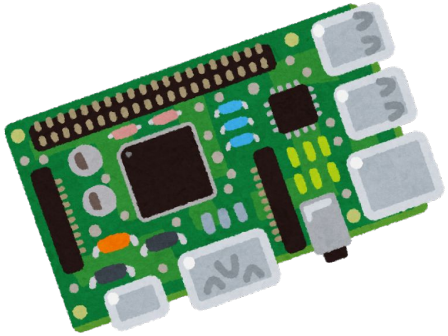
- 高位合成に関する作業はIP作成部分のみ
  a. Vivado IPIで対象アルゴリズムを組み込むプラットフォームの設計
  b. PC上でSW実装のアルゴリズムから、HW化する処理を抽出
  c. Vivado HLS上で、抽出した処理の移植（処理の最適化も実施）
  d. Vivado HLSでアルゴリズムのcsim/CoSimで確認しIP化
  e. Vivado IPIで作成したIPを結合し合成、インプリメンテーション
  f. SDKでアプリケーションSW、BSP、Bootの作成

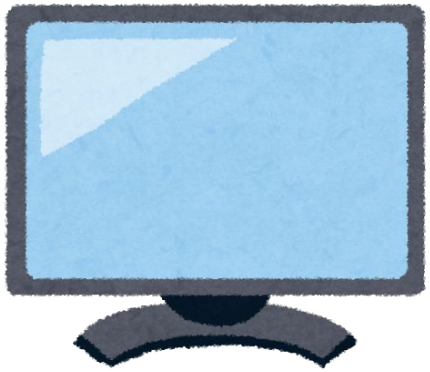
  作業的にはc,dで1ヶ月、その他a,b,e,fで計1ヶ月

そもそも、高位合成でのIPを作成以外にもSWエンジニアにとってはハードルの高い作業が沢山ある。

# SWエンジニアが感じる、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

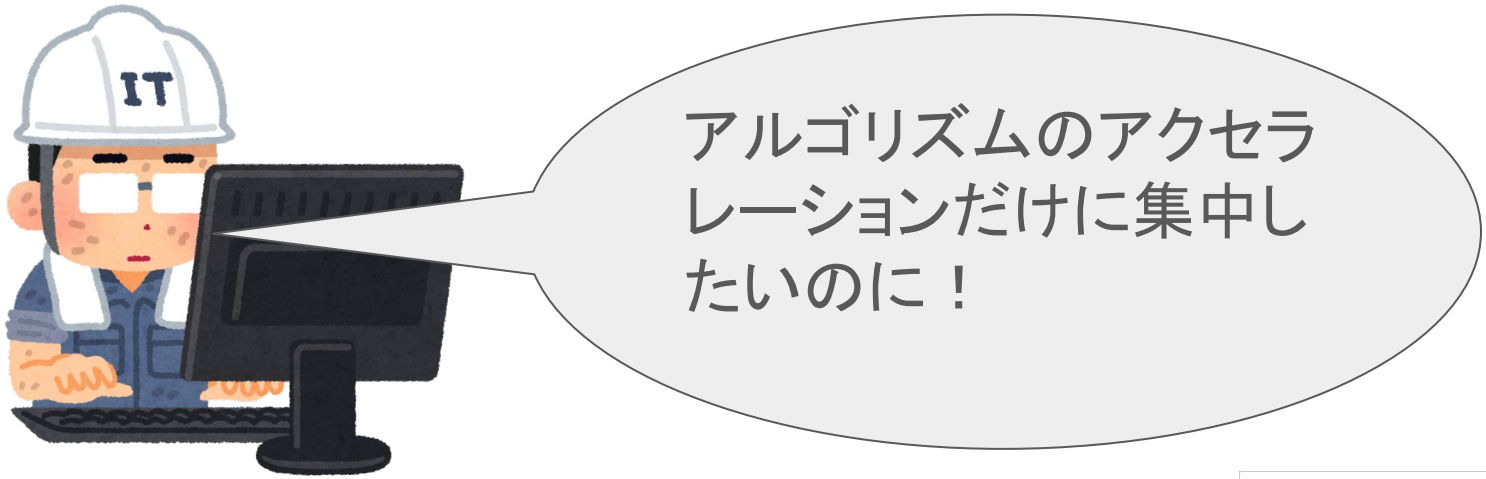


# SWエンジニアが感じた高位合成以外の課題

- 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

# SWエンジニア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」



# 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）

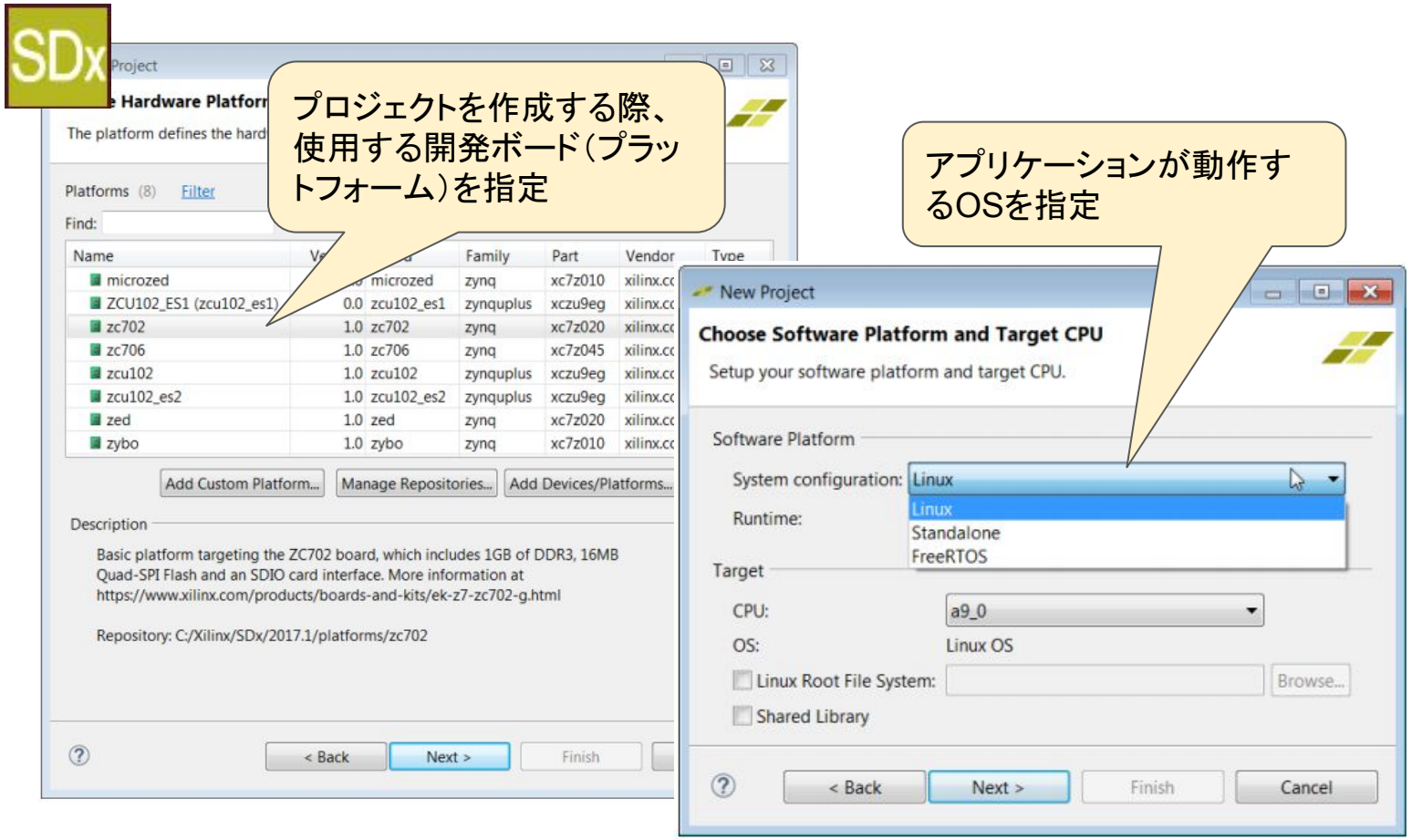
今回はSDSoCを紹介

# 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
  - SWエンジニアにも馴染みやすい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

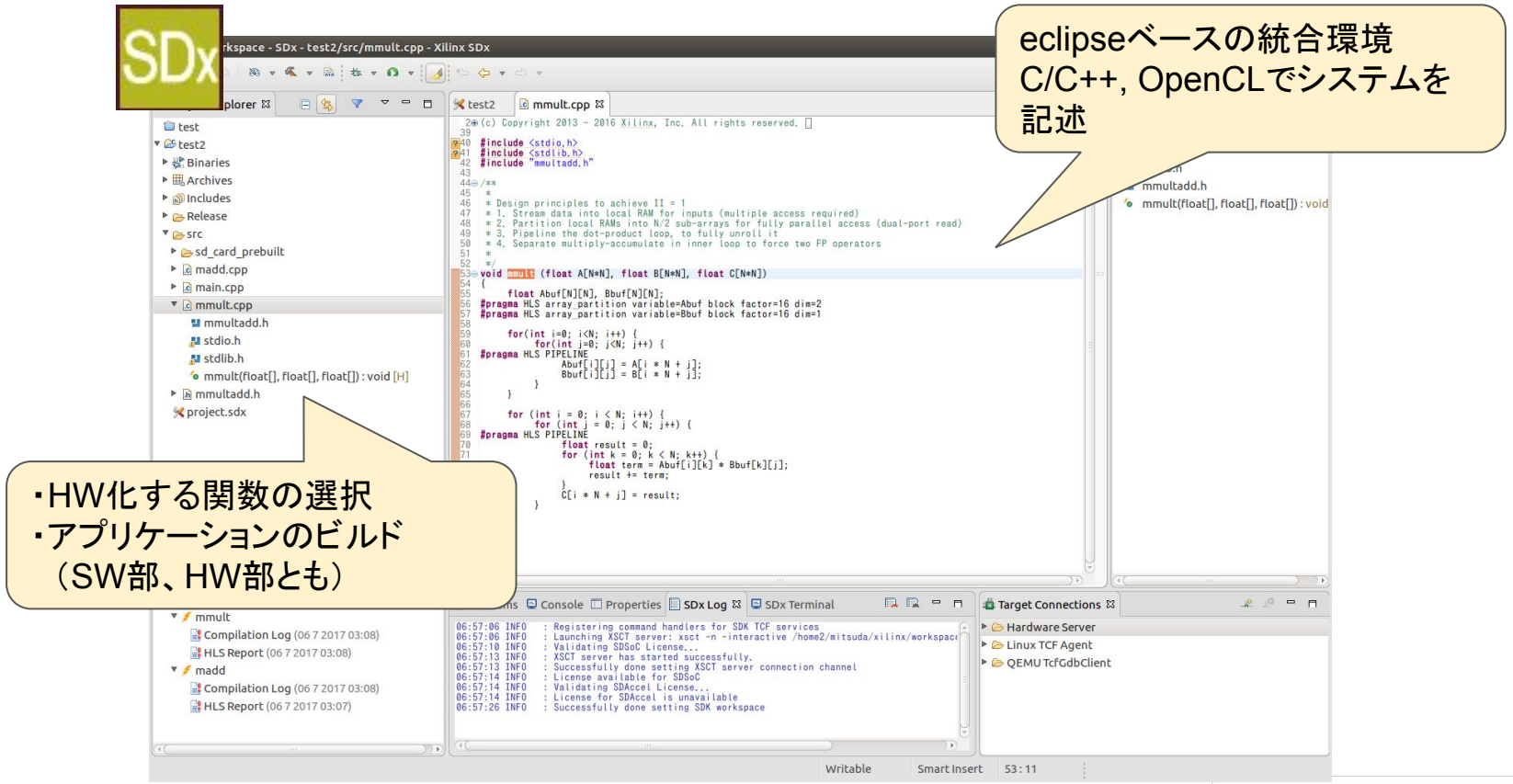
- あらかじめ提供される開発プラットフォーム




引用元:UG1028 SDSoC環境チュートリアル：入門

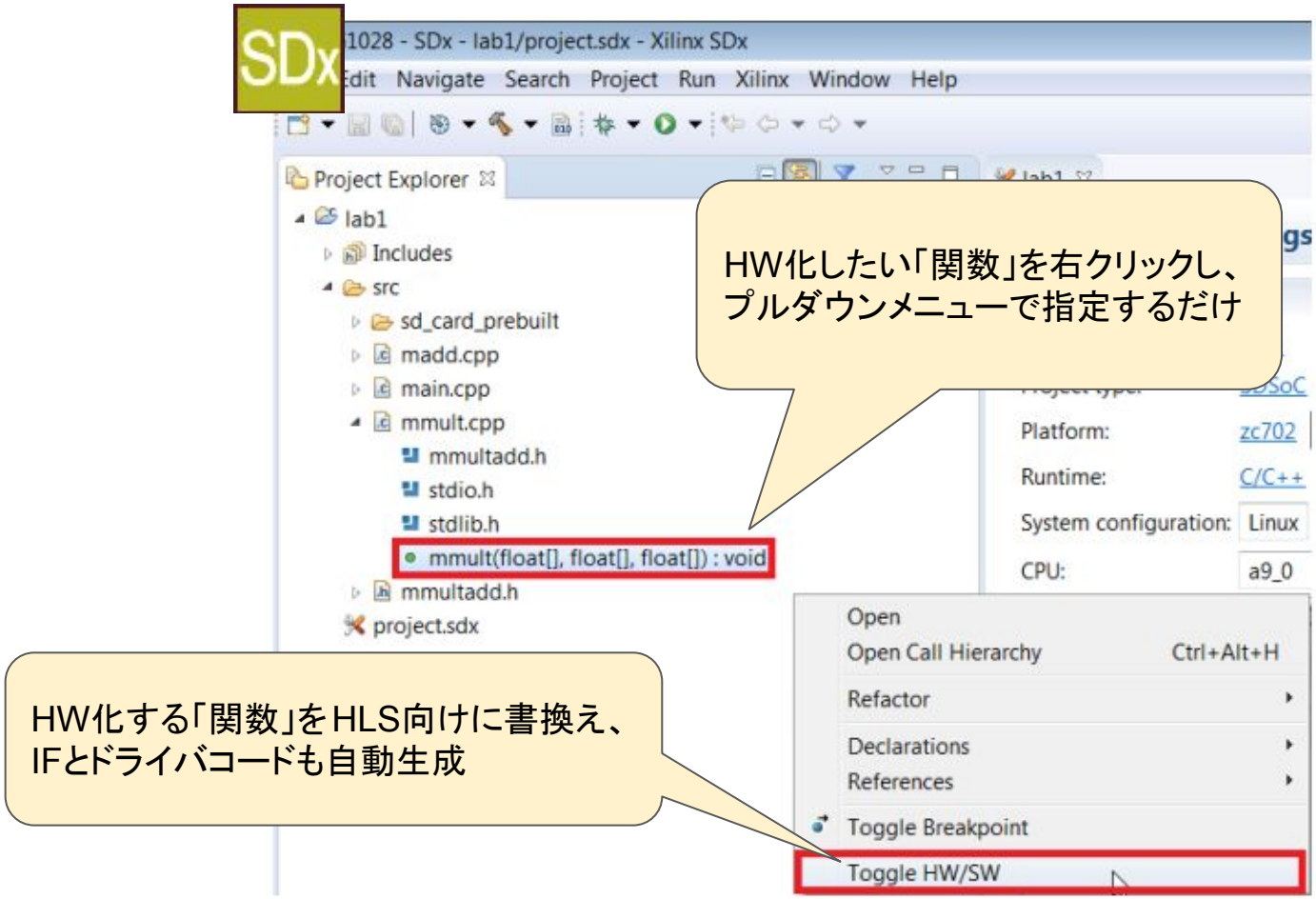
# SDSoCの特徴～SW/HW統合された開発環境

- 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



# SDSoCの特徴～HW化対象の選択

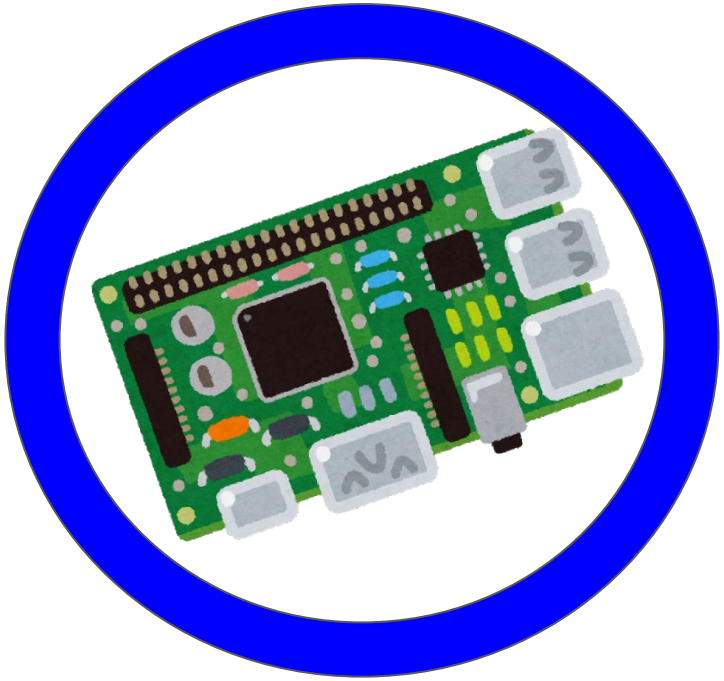
- アルゴリズムのアクセラレーションを意識した機能




引用元:UG1028 SDSoC環境チュートリアル：入門

# 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

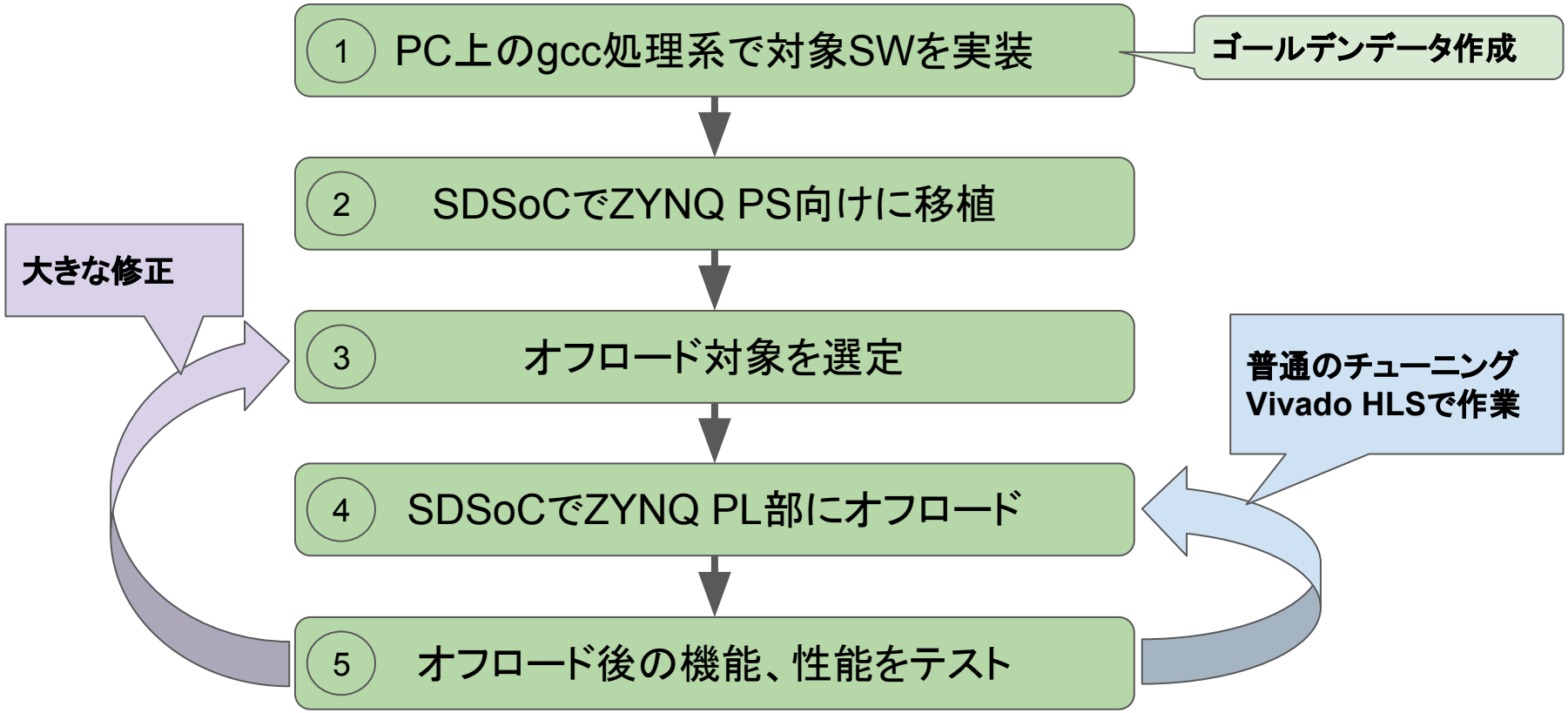
【率直な感想】

「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

# FPGAは「やわらかく」使えるのか？

# 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

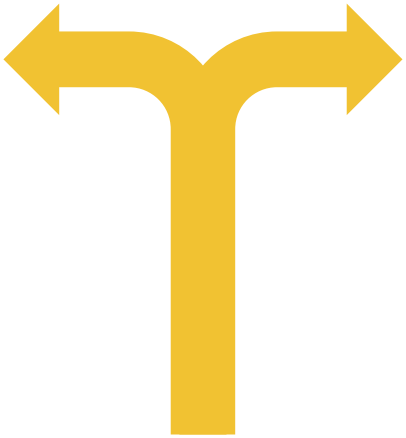
- 大前提：元のアルゴリズムの機能を担保すること
  - ただし、実現不可な機能や本質的でない機能は整理する
    - ダイナミックに確保されるメモリの必要量を決めて固定化
    - SWの柔軟性に資するパラメータなどは整理してなるべく固定化
- アクセラレータによる「目標性能」をどうするか
  - HW化によって達成すべき性能は決まっているか？
    - 決まっている場合、その目標は妥当か？
    - 決まっていない場合はどうするか？

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）

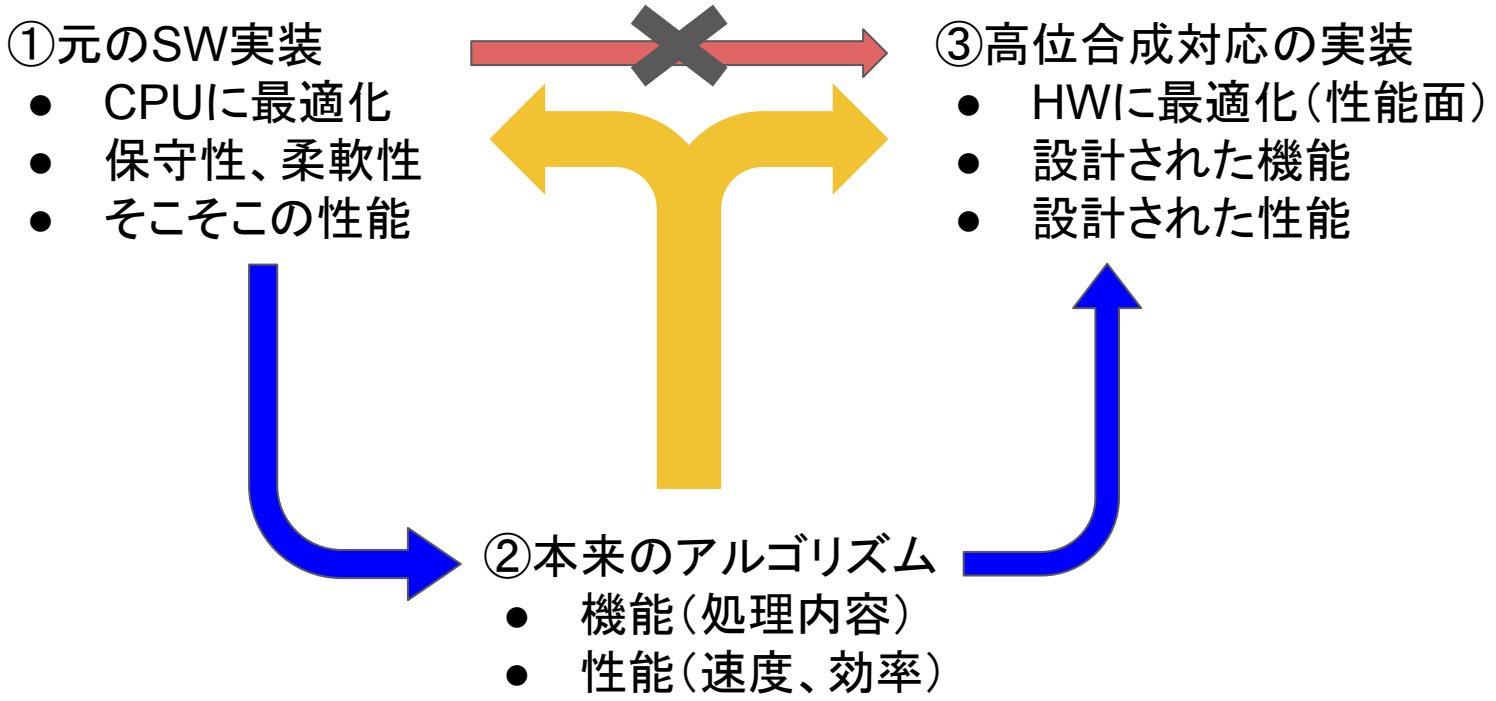
  *『回路設計では、達成すべき速度性能・クロック周波数・面積が明確な数値目標として決まっているのが普通で、回路設計者は機能だけでなく性能をいかに達成するかに腐心しています。』*

- 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



# SWエンジニアは高位合成を使いこなせるのか？

- SWエンジニアが高位合成/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

# 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])
    - 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）

# 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
  - 例2）階層化された関数は、どのように「配置されるのか」?

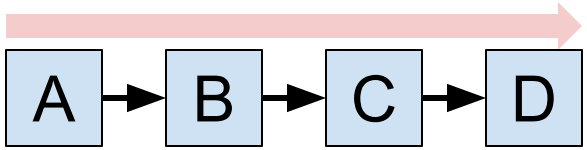
```
dut(…){
        A(…); //sub関数A
        B(…); //sub関数B
        C(…); //sub関数C
        D(…); //sub関数D
      …
}
```

# SWエンジニアに必要なHWの感覚

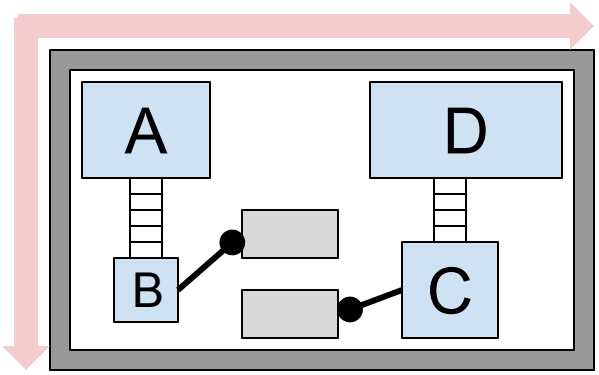
- SWには無い「物理リソース」の感覚
  - 例1）HLSで記述した関数の引数 W,X,Y,Zは「どんなIFにすべきか？」

    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
  - 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




「SWエンジニアは処理を時間軸で、HWエンジニア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SWエンジニアは時間軸で、HWエンジニアは空間に配置

**⇒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**

# SWエンジニア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
  - 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

# SWエンジニア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？」

# SWエンジニア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

# SWエンジニア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)

SWエンジニアは「おまじない」を受け入れる事には慣れている

# SWエンジニアに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

# SWエンジニアに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による高性能ディジタル回路設計』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：SWエンジニアでもFPGAを使うべき

- SWエンジニアもFPGAを使えるようになりたい
  - アクセラレートが必要なアルゴリズムがある
    - MSやBaiduの例でも明らか
  - 自分のアプリケーションが動くプラットフォームを増やしたい
    - 弊社のROBUSKEYがPC以外で動くことを熱望

# まとめ：SWエンジニアでもFPGAを使うべき

- SWエンジニアもFPGAを使えるようになりたい
  - アクセラレートが必要なアルゴリズムがある
  - 自分のアプリケーションが動くプラットフォームを増やしたい
- FPGAという選択肢を持つことによる技術の発展
  - これまで無理だと思っていたニーズに応えられる可能性
    - ニーズから新たな研究のシーズへ転換
  - これまで知らなかったシーズを元に、新たなニーズを創出