卒業研究発表会
2017年2月11日

FPGAを使った モルフォロジー処理の実装

Implementation of Morphological Operation Using FPGA

171401

Sakamoto Yoshiki

目次

- 研究動機
- FPGA
- 高位合成
- モルフォロジー
- 研究環境
- 実装
- 実験
- まとめ

研究動機

- FPGAを上手く扱うための技術の研究に興味があるため。

  例えば高位合成技術

- FPGA上で回路を設計し実装する経験を積むため。

  FPGA上でのアプリケーション開発は、SW開発と異なる点が多い。

- モルフォロジー処理を実装内容として選択した理由

  一要素あたりの処理が単純でスケールアップがしやすいと推測したから。

FPGAについて

- FPGA　(Field-programable Gate Array)

  プログラムすることによって後から、回路を実装可能なデバイスである。

- 現在FPGAはほぼ2社によって製造されている。

  Xilinx, Intel(Altera)

- FPGAはハードウェア記述言語によってプログラムする。

  ハードウェア記述言語(HDL)は、Verilog HDL, VHDLが使用される。

- FPGAは多くの分野で応用されている。

  ASIC(Application Specific Integrated Circuit)の代替デバイスとてのみならず、スーパーコンピュータ、ネットワーク、ビッグデータ処理、ゲノム科学、金融市場、人工知能、画像処理といった様々な分野で用いられている。

高位合成

- 高位合成(High-Level Synthesis)

  高級言語で書かれた動作記述から、
  ハードウェア記述言語を生成する技術である。

- 高位合成ツール

  Xilinx社の高位合成ツールである
  Vivado HLSはC, C++ベース。

  Altera社はOpenCLを使って高位合成が可能だ。

- 動作記述と回路記述の対応

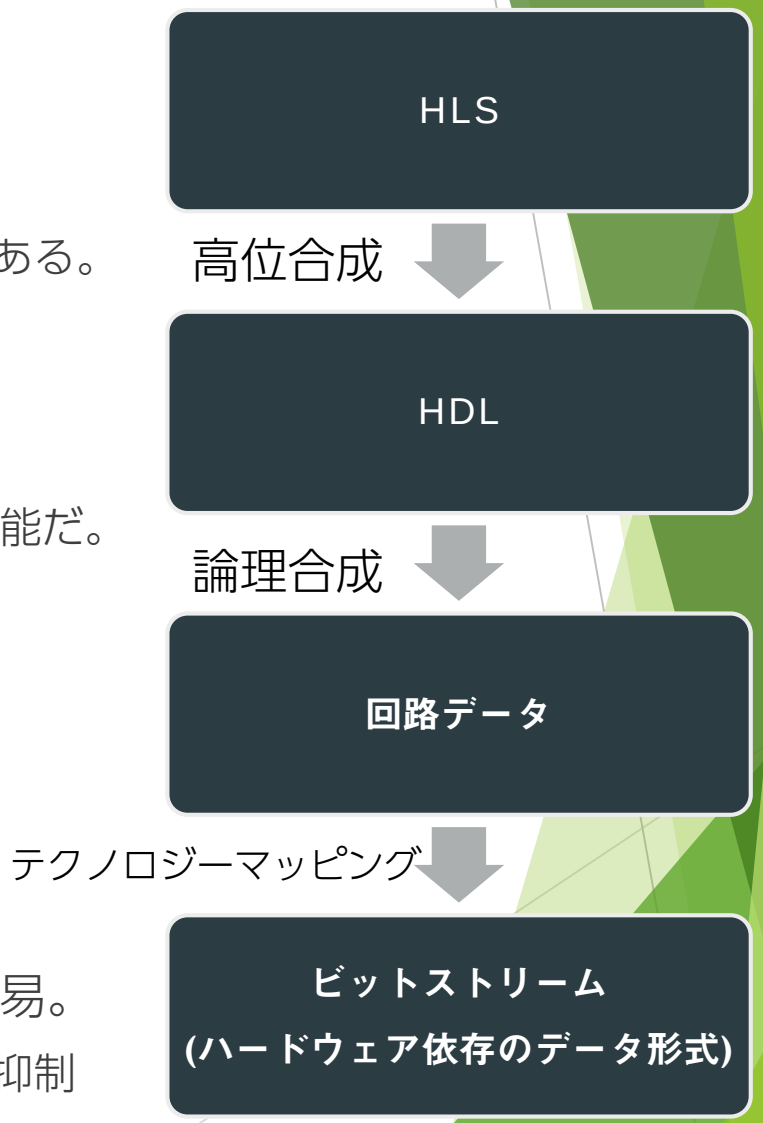
  変数→レジスタ、配列→メモリ、
  関数→回路モジュール

- 一部記述は高位合成不可能

  再呼び出し、動的ポインタ
  動的にインスタンスが生成出来ない為

- 高位合成はHDLを使った設計に比べ容易。

  納期の短縮、人為的ミスからなるバグの抑制



モルフォロジー

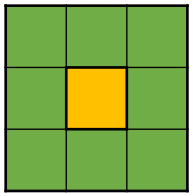
- モルフォロジー処理。
  - 画像に対して膨張・収縮処理を行い、画像の持つ構造を抽出する処理である。
- 応用例としては。
  - 画像の平滑化、孤立点除去、輪郭抽出、ノイズ除去等
- 膨張・収縮処理。
  - 注目した画素と、その注目した画素の周辺のあらかじめ指定した近傍の画素に対して論理演算を行う処理である。膨張処理の場合は論理和、収縮処理の場合は論理積を行う。

収縮処理一例

1	0	0	1	0	1	1	1	1	1
1	1	1	0	0	0	1	0	1	1
1	1	1	1	1	1	0	0	1	0
0	0	0	0	1	0	0	0	0	0
1	1	1	1	1	1	0	1	1	1
0	0	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	0	1	1
0	0	0	1	1	1	0	1	0	0
1	0	0	1	0	0	1	0	1	1

収縮処理

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	1	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

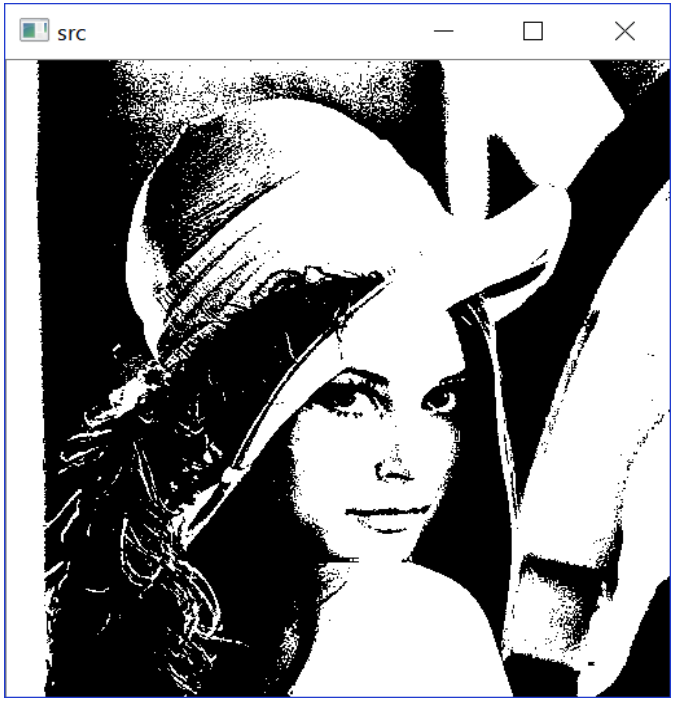


8近傍

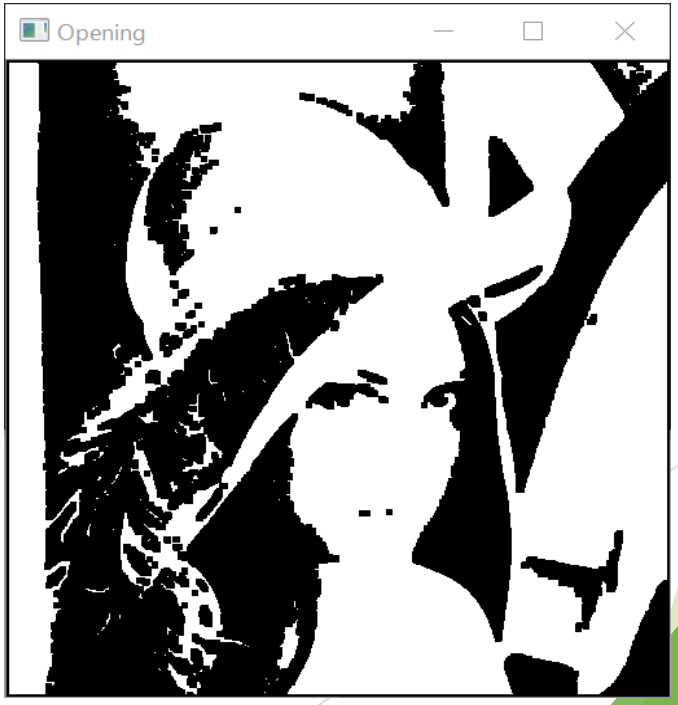
オープニング処理

　収縮を繰り返し行った後、同様の回数だけ膨張する処理をオープニングと呼ぶ。オープニング処理は、画像の突起部分の除去や結合部分の分離等に利用される。

元画像



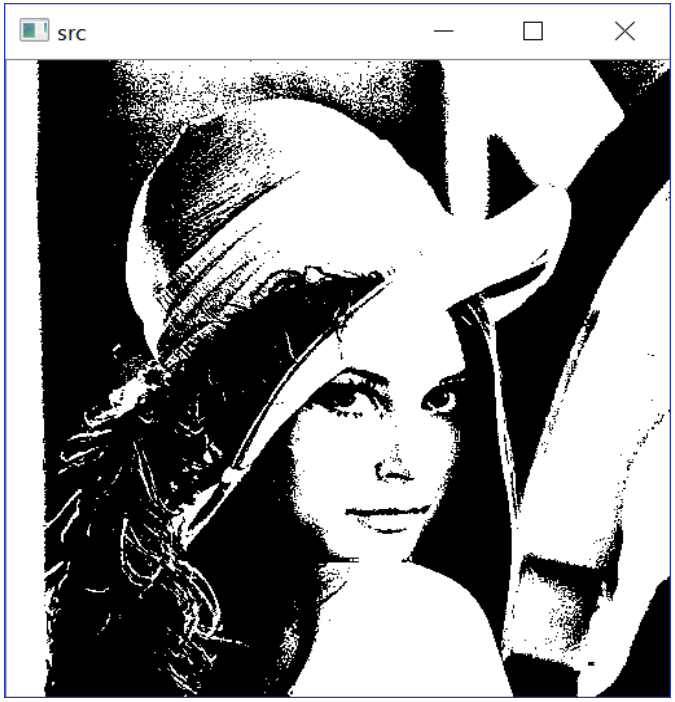
オープニング処理



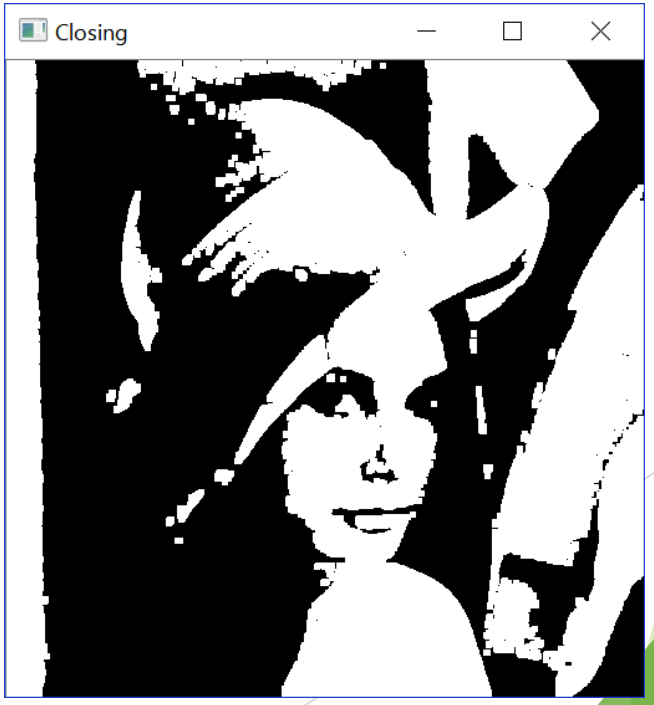
クロージング処理

　膨張を繰り返し行った後、同様の回数だけ収縮する処理をクロージングと呼ぶ。クロージング処理は、画像の穴埋めや切断部分の結合等に利用される。

元画像



クロージング処理



研究環境

- ZYBO
- Vivado 開発ツール群
  - Vivado統合開発ツール、VivadoHLS、VivadoSDK
- 開発・デバッグ用コンピュータ
  - CPU: Intel Core i7-4790k(4.00Ghz, 4Core, 8Thread)
  - Memory: DDR3 16.0GB
  - Storage: Samsung SSD 850 EVO 250GB
  - OS: Windows10 Home
  - コンパイラ、開発環境:
    Visual Studio Community 2015, MSYS2/GCC6.2.0
  - 使用ライブラリ: OpenCV 3.1.0

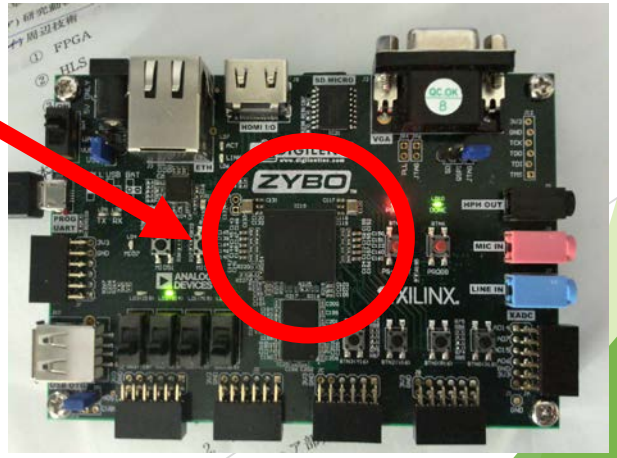
ZYBO

- 実装と実験にDegilent社のZYBOを使用した。
- ZyboはZynqチップを実装した安価な評価ボードである。

  参考:秋月電子で税込み24.700円

  Zyboに実装されたZynqチップは2つのARMコア(Coretex-A9)を内包している。
- 多くのI/Oがボードには実装されている。

ZYNQチップ
FPGA＋ARM



実装

- SW部とHW部に分けて実装した。

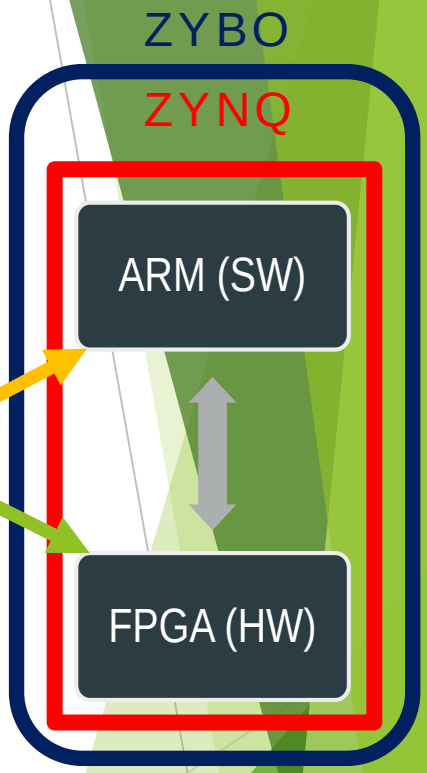
  ZYNQチップにはARMコアが組み込まれている。

- モルフォロジー処理そのものはHW部

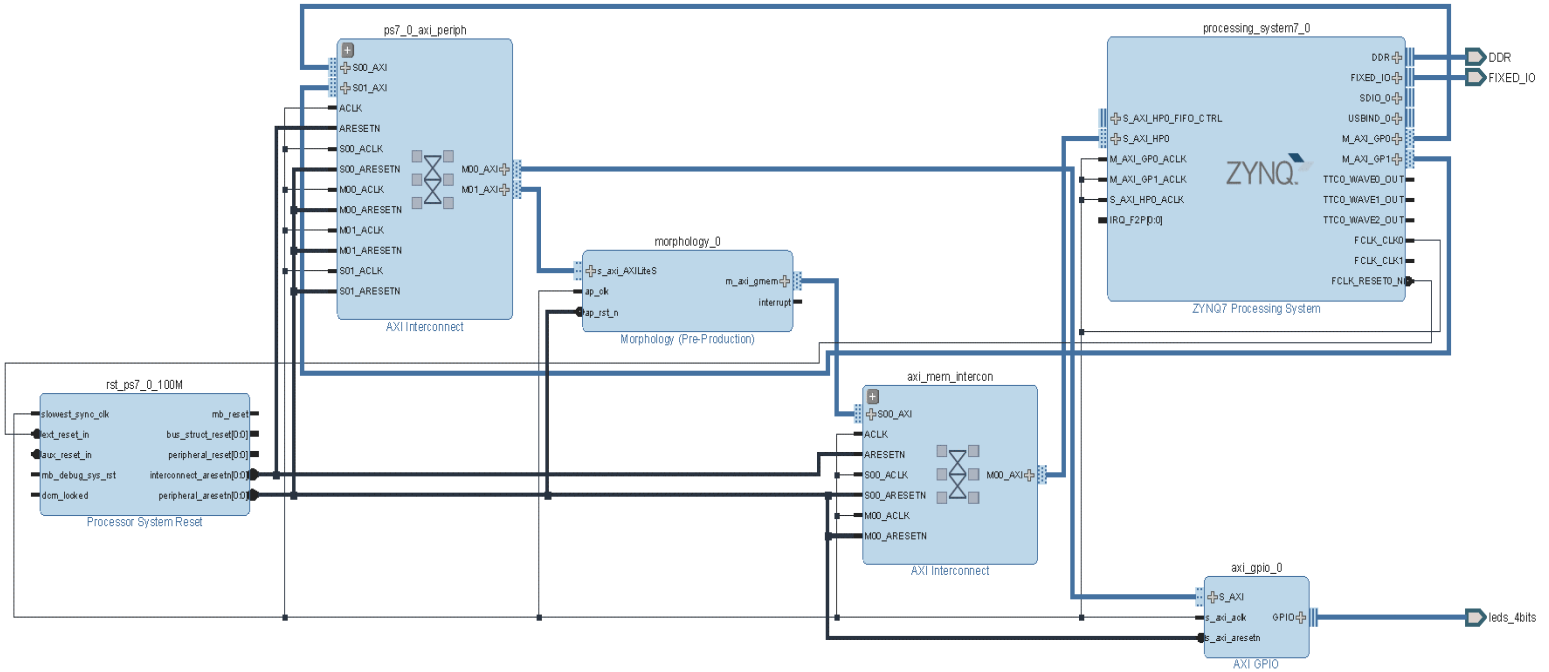
  そもそも、研究のテーマがモルフォロジー処理をFPGA、ハードウェアで実装すること。

- それ以外の制御・計測等をSW部

  制御部分(処理回路の開始・停止・完了確認）、時間計測をHWで実装すると設計が煩雑になり、研究テーマからも遠い為。



ZYNQチップ上の回路概観図



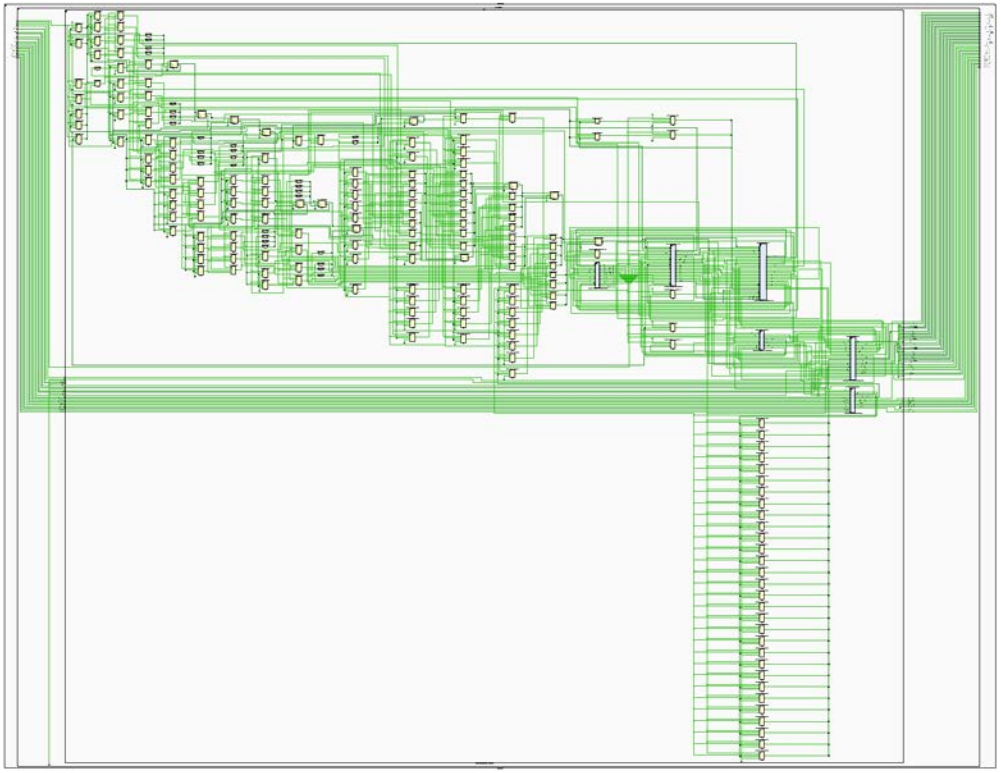
SW部

- 回路制御部
  - 回路を制御する為のAPIはVivado HLSが自動的に生成する為、それを使用した。
- 入出力部
  - 入出力は、Xilinx社が提供するxil_ioライブラリを使用した。
- 時間計測
  - Xilinx社が提供するxtimeライブラリ中のXTime_GetTimeを使用した。
    - この関数はクロック数を計測することで経過時間を計測することが可能で、ベアメタルでも時間を容易に計測できる。

HW部

- 高位合成を使いHW部の回路を実装した。
  - HDLを使った設計が困難だった為
- C言語を使った高位合成は、合成時間は1日にかかることもあった。
- 回路概観図

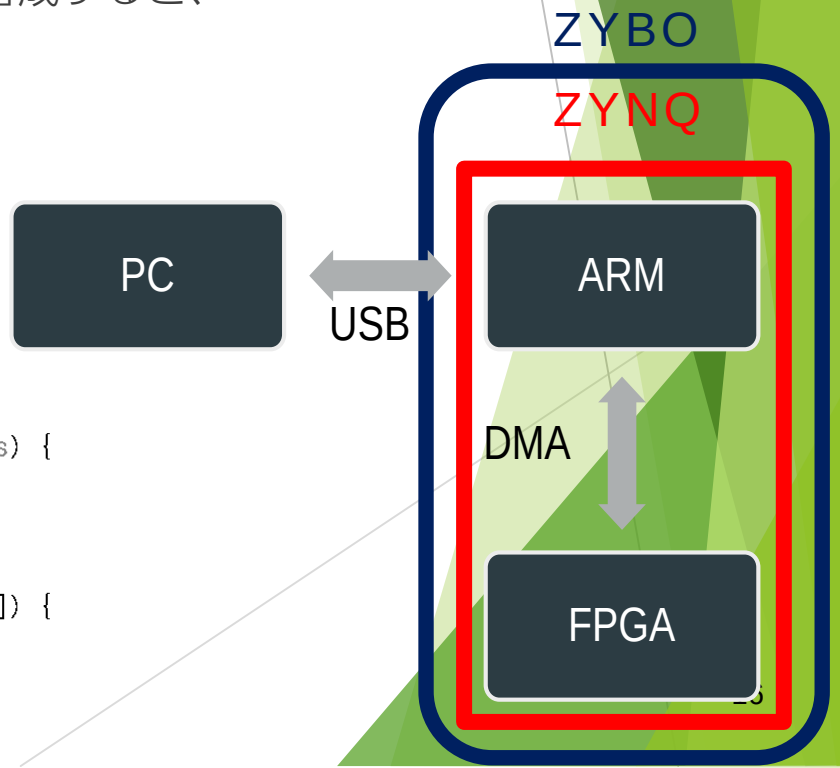
(論理合成結果)



HW部:DMA

- DMAとはDirect Memory Accessの略である。
- 大きな配列をARM⇔FPGA間で転送する為に使用した。
- 具体的な実装方法:

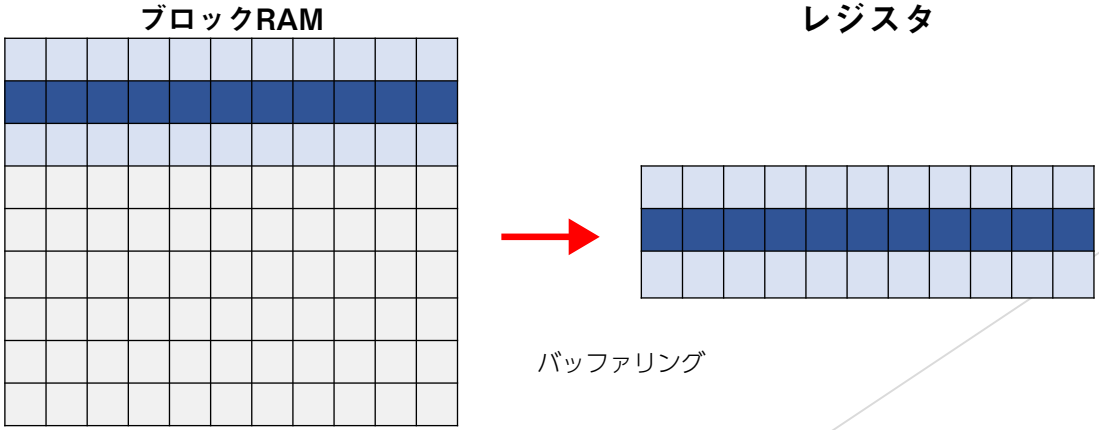
　forループ中にメモリと配列をコピーするよう記述、
もしくわmemcpyを使い、Vivado HLSで高位合成すると、
自動的にDMA回路が生成される。




```
1  void arm_to_fpga(int fpga_bram[], const int *memory_address) {
2      for (int i = 0; i<100; i++) {
3          fpga_bram[i] = memory_address[i];
4  }
5
6  void fpga_to_arm(int *memory_address, const int fpga_bram[]) {
7      for (int i = 0; i<100; i++) {
8          memory_address[i] = fpga_bram[i];
9  }
```

HW部: バッファリング

- SW向けのCソースコードで生成した回路が低速だった。
  - ブロックRAMへのアクセス数が多く、そこがボトルネックとなっていた。
    - ブロックRAMは、1クロックあたりに一定のデータ幅しかデータにアクセス出来ない。
- レジスタにバッファリングを行うようにした。
  - レジスタは任意のタイミングで任意のデータにアクセスすることが可能。しかし、回路面積を消費する。



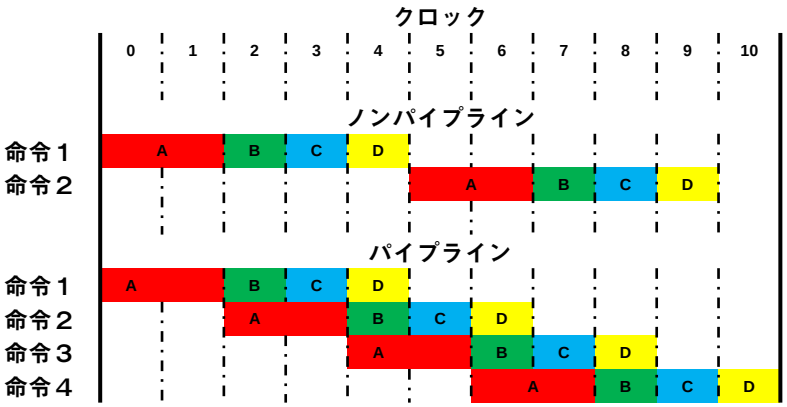
HW部: ディレクティブ最適化

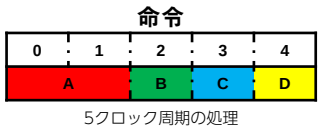
- Vivado HLSは生成する回路を指示句で指定することが可能。
- 今回、HLS_PIPLINEを膨張・収縮処理部に使用した。

  HLS_PIPLINEは指定した処理をパイプライン化するディレクティブである。スループットが約4倍向上した。

- パイプライン処理とは

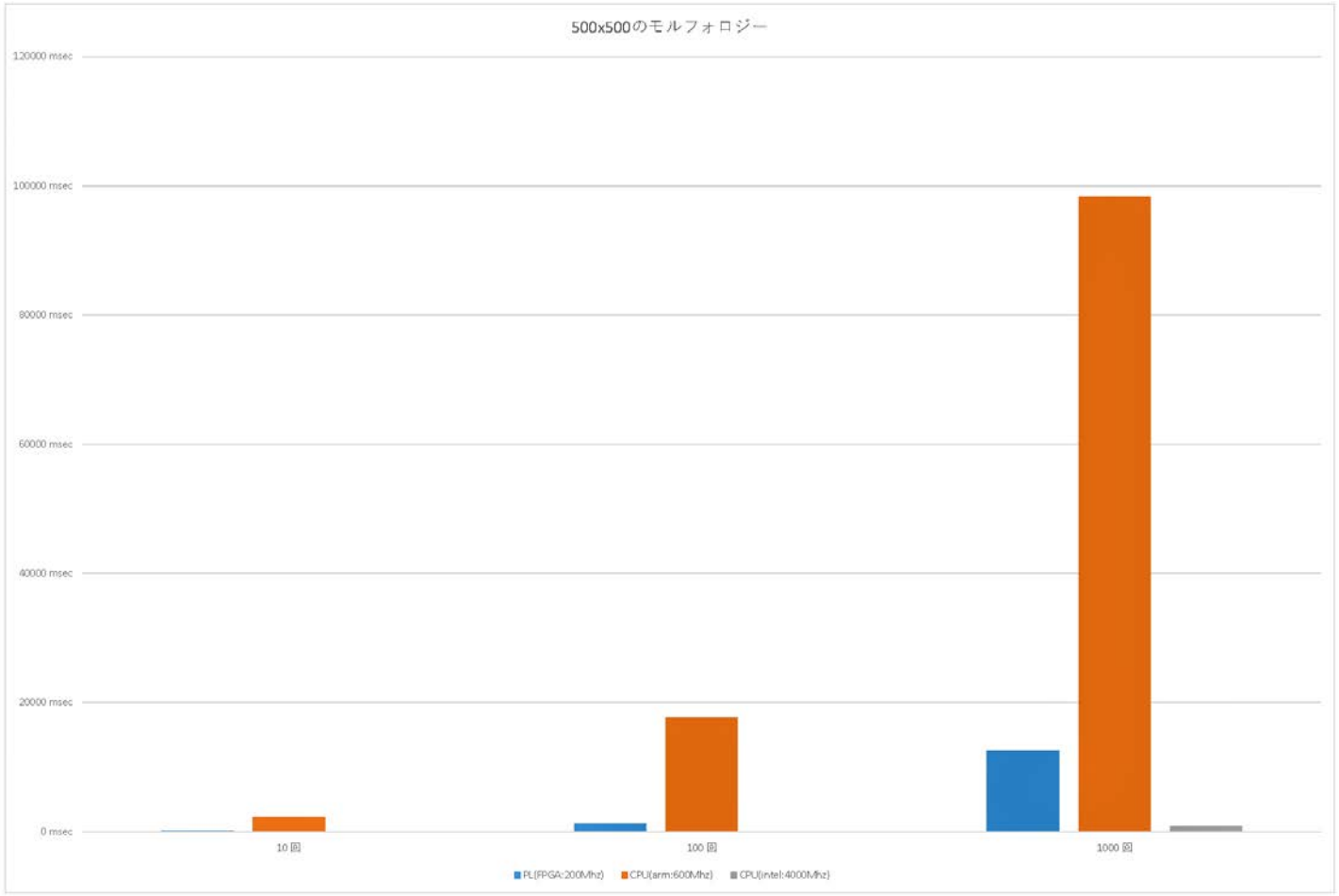
  逐次的な処理を多段階に分割し、先行する処理全体の完了を待たずに、段階ごとに処理を実行することによって スループットを向上させるものである。





5クロック周期の処理

結果



FPGA、FPGA上のARM、Intel Core i7-4790k で比較
Intel Core i7-4790kはWindows上のMSYS2/GCCで計測した。
それぞれ100回分の値の平均値を取った。

まとめ

- モルフォロジー処理回路はIntel CPUよりも低速だった。
  - OoO等のCPUの最適化機構、コンパイラの性能、クロックの高さで、Intel CPUは優れている。
- 消費電力は約1.5W
  - 消費電力としたはかなり低いものだと推測出来る。
- 高速な処理回路設計はやはり難しい。
  - 高速な処理回路の設計にはより多くのHW知識が必要。
- SW技術者が高速なFPGAアプリケーションを設計す為には、高位合成技術のさらなる発展が必要である。