第4回 AI エッジコンテスト
（実装コンテスト②）

TFlite delegate を用いた実装

山下　伸逸
ハードウェアエンジニア
主にアナログ回路設計

概要

- TensorFlow Lite の delegate 機構を用い、FPGA にアクセラレータを実装した
- 実装は、主に SystemVerilog を用いた RTL 記述で行った
- 推論ネットワークは deeplabv3 mobilenetv3 を用いた
- アプリケーションは TFlite の python インターフェースを用いて開発した

TensorFlow Lite は Google の提供する Mobile / Edge Device 用の軽量な推論プラットフォーム
TensorFlow や Keras のネットワークから軽量な FlatBuffer 形式の graph に変換する converter とその graph で推論を実行する、各種 Mobile デバイスに対応した Interpreter が提供されている
delegate 機構によって演算を外部アクセラレータに委譲することができる

推論ネットワーク 1

- ネットワーク
  - TensorFlow DeepLab Model Zoo
    mobilenetv3_small_cityscapes_trainfine
    1025x2049 の入力画像サイズを 513x1025 に縮小
    （メモリオーバーで学習できず）
- 19 class → 評価対象 4 + bg の 5 class で転移学習
  - クラス数を減らして推論実行速度を上げる
  - 信号、歩行者の精度が上がらないのでラベルの重みを調整した
- 転移学習後 quantization-aware training で 8bit 量子化
  → TFlite FlatBuffer 形式の graph に変換
- 結果：mIOU = 0.51 (float)、0.47(uint8) 基準満たせず

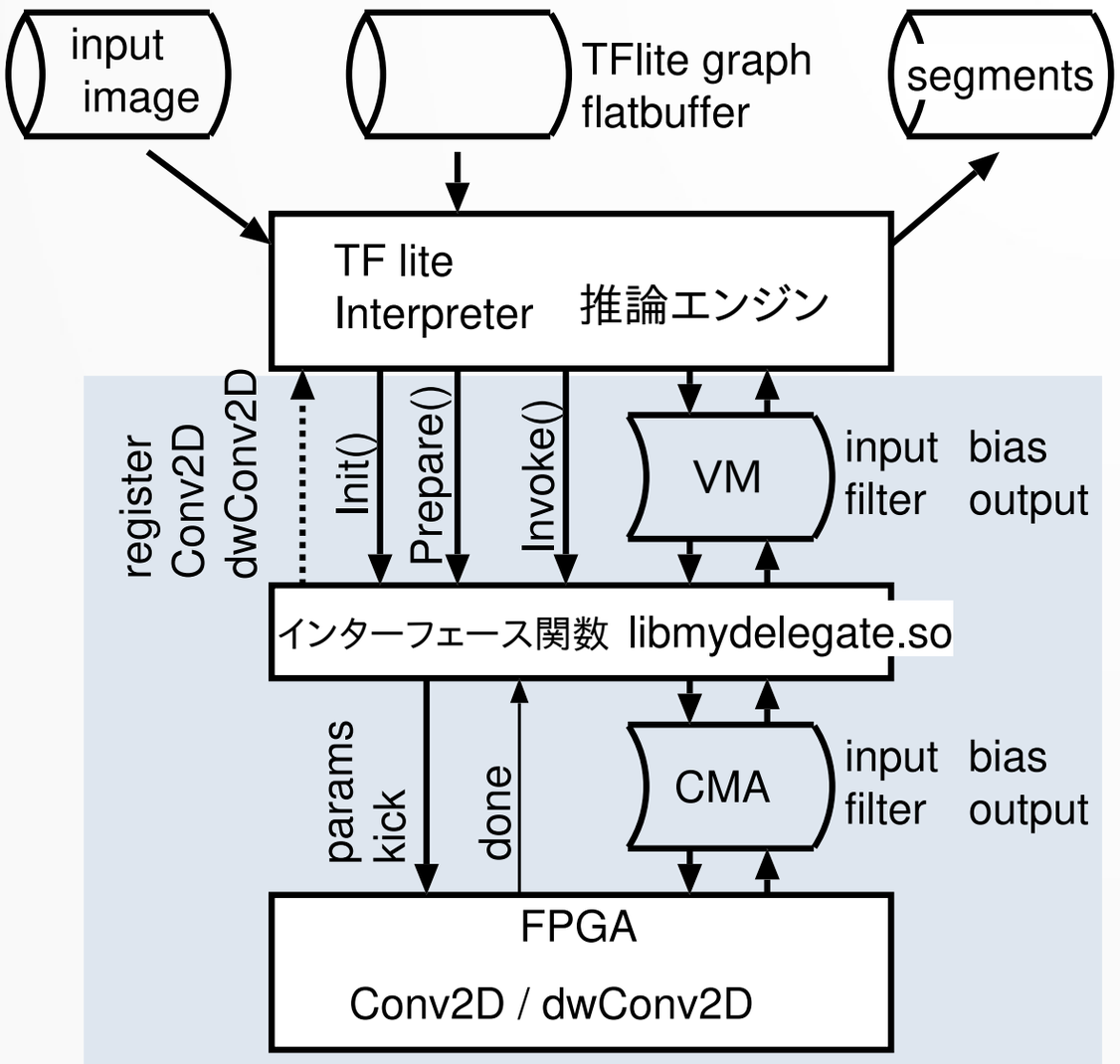
推論ネットワーク 2

mobilenetv3_small_cityscape 5 class 8bit 量子化ネット
TFlite benchmark aarch64 cpu 実行結果

```
Number of nodes executed: 128  aarch64
=========== Summary by node type ================
         [Node type]  [count]   [avg ms]   [avg %]
              CONV_2D        45    285.908   45.176%
    DEPTHWISE_CONV_2D        11    159.624   25.222%
      RESIZE_BILINEAR         5     80.827   12.771%
             ARG_MAX         1     55.767    8.812%
                 MUL        20     20.289    3.206%
          HARD_SWISH        18     13.774    2.176%
                MEAN         9      9.709    1.534%
                 ADD        17      6.371    1.007%
      AVERAGE_POOL_2D         1      0.589    0.093%
    Misc Runtime Ops         1      0.019    0.003%
                      count=52  avg=632.942
```

Conv2D,dwConv2D で 70.4% を占める
→ この2つの演算を FPGA に delegate する

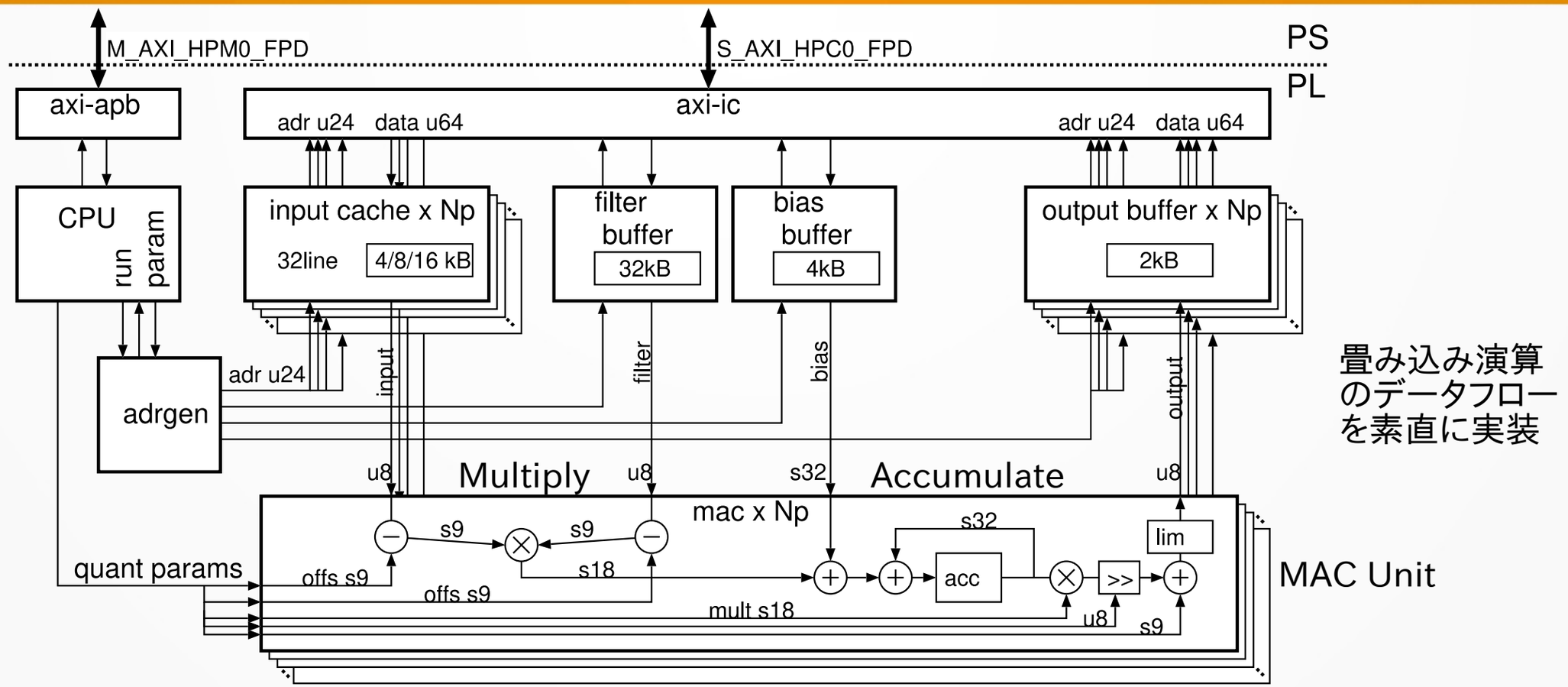
システム構成



- TFlite Interpreter の delegate API を用いて FPGA に演算を渡す

  Interpreter にインターフェース関数をリンクする
- delegate する演算種別 (Conv2D, depthwiseConv2D) を登録する
- Interpreter は FlatBuffer 形式の graph を実行し、登録した演算のみインターフェース関数に渡して実行を委譲
- インターフェース関数には Conv 演算パラメータと、input, filter, bias, output の 4 つの Tensor へのポインターが渡され、これを FPGA に渡して ハードウェアを kick し、演算終了を待つ
- データは CMA 領域を用いてやり取りする
  Linux の仮想記憶領域と CMA 領域の間で memcpy が発生する

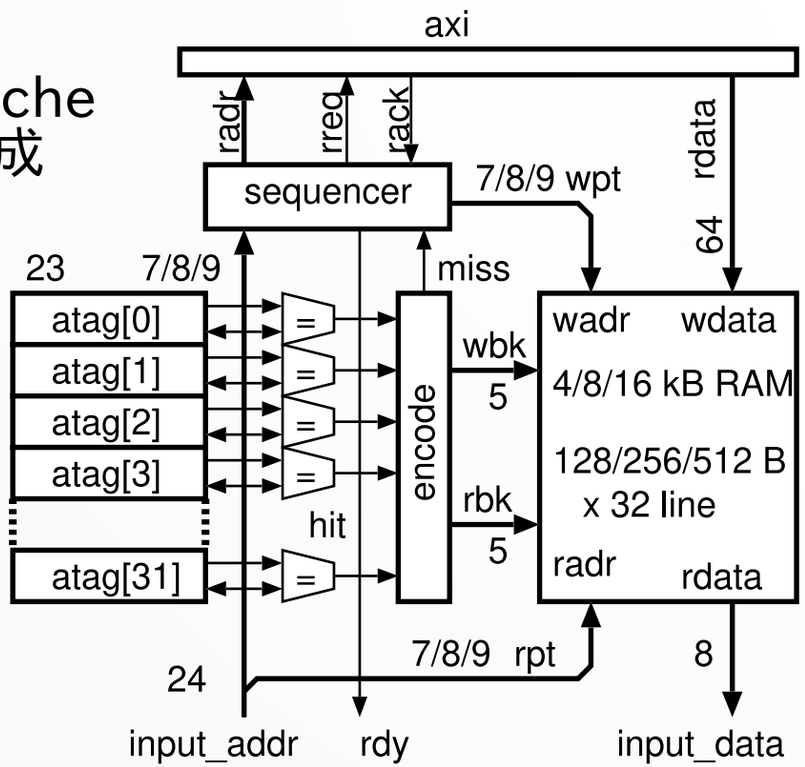
ハードウエア構成 1



- Conv 演算（MAC）は、Np 個並列に実装、Conv2D/dwConv2D は共通回路
- input, output のデータアクセスには Np 個並列にキャッシュメモリを設けた
- filter, bias は Conv 演算で共通、それぞれ 1 個のバッファメモリを設けた

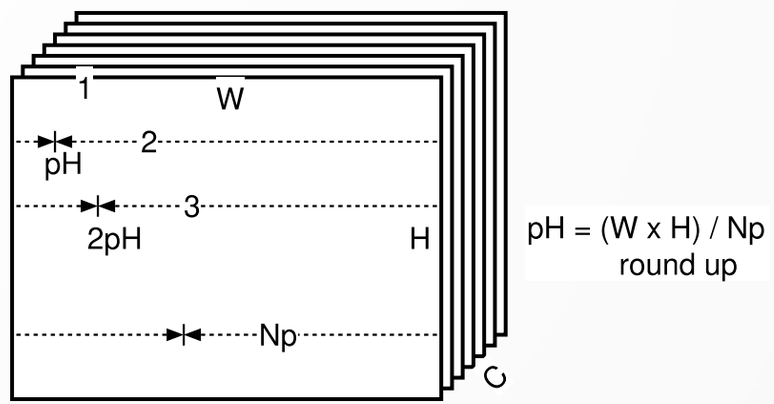
ハードウエア構成 2

input-cache
の構成



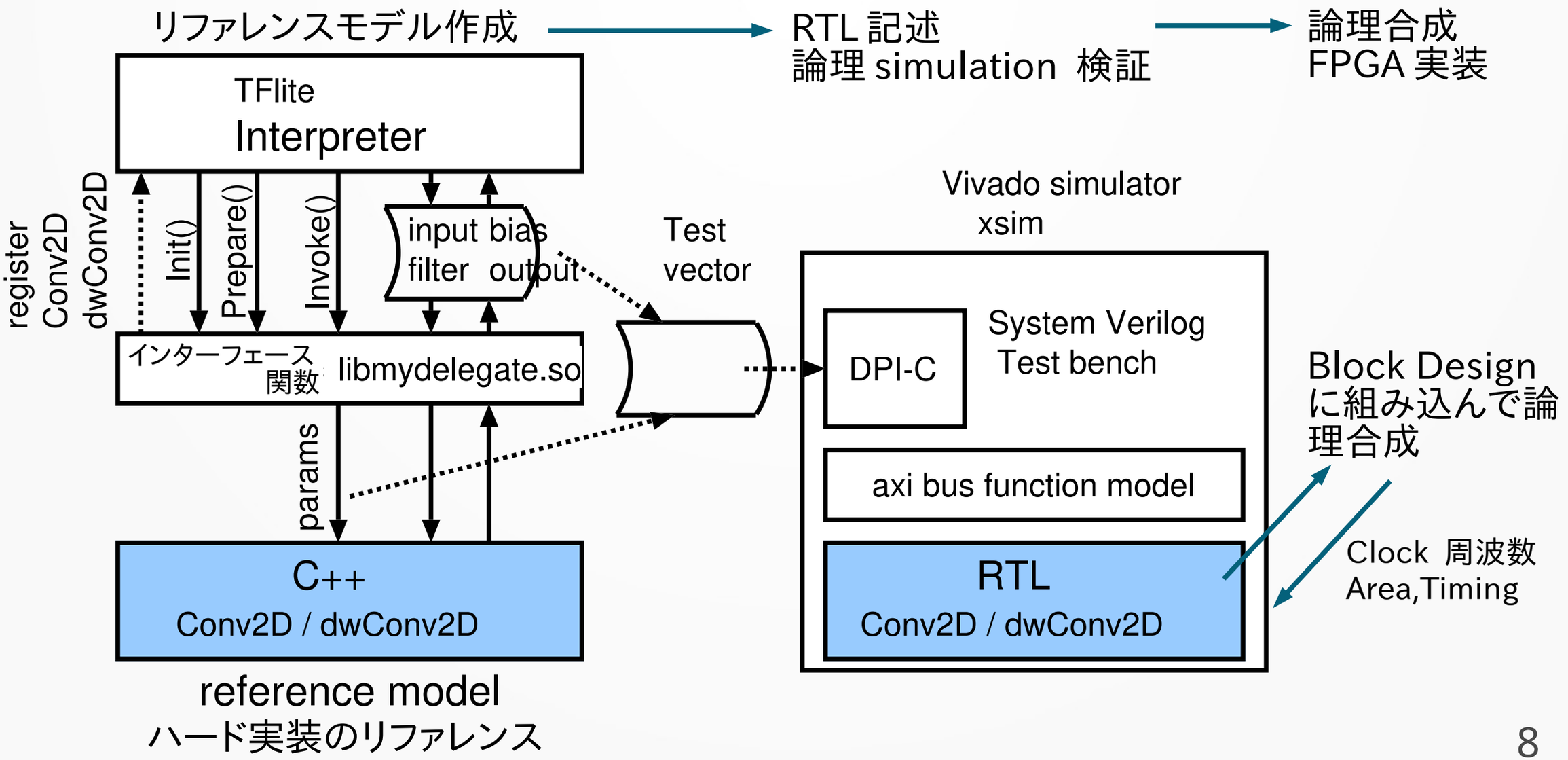
- input は畳み込みのタップで大きくアドレスが飛ぶので、32 ラインのキャッシュメモリ構造にした
  - movilenet-v3 では 5x5 の conv 演算がある
- 並列数 Np と cache メモリサイズは、RAM リソース、配線性、動作速度など様々なトレードオフになる

並列化のアドレス分割



- output Tensor の W x H を Np 分割した
- W x H < Np のときは並列実行数は W x H に制限される

システム開発手順


リファレンスモデル作成
RTL 記述
論理 simulation 検証
論理合成
FPGA 実装
TFlite
Interpreter
register
Conv2D
dwConv2D
Init()
Prepare()
Invoke()
input bias
filter output
インターフェース
関数
libmydelegate.so
params
C++
Conv2D / dwConv2D
reference model
ハード実装のリファレンス
Test
vector
Vivado simulator
xsim
DPI-C
System Verilog
Test bench
axi bus function model
RTL
Conv2D / dwConv2D
Block Design
に組み込んで論
理合成
Clock 周波数
Area,Timing

アプリケーション python interface

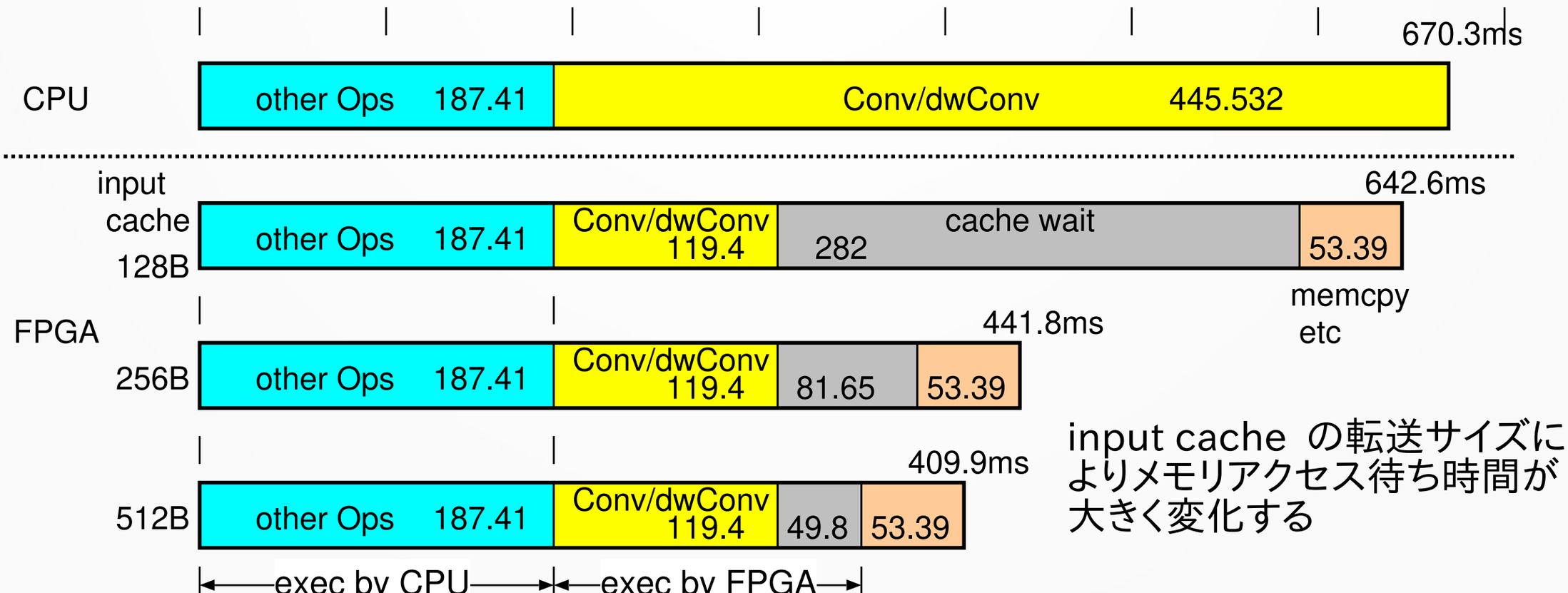
```
import tflite_runtime.interpreter as tflite

# Instantiate interpreter
interpreter = tflite.Interpreter(model_path=model,
      experimental_delegates=[tflite.load_delegate('{path-to}/libmydelegate.so.1')] )
interpreter.allocate_tensors()

# Get input and output tensors.
input_details  = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# set image
interpreter.set_tensor(input_details[0]['index'], np.uint8(image))

# Invoke
interpreter.invoke()

# Output inference result
segments = interpreter.get_tensor(output_details[0]['index'])[0]
```

Interpreter にインターフェース関数 (共有ライブラリ) を指定するだけ

実行結果

推論処理時間内訳

MAC 並列数　Np = 32
演算クロック周波数 = 150 MHz



input cache の転送サイズによりメモリアクセス待ち時間が大きく変化する

まとめ

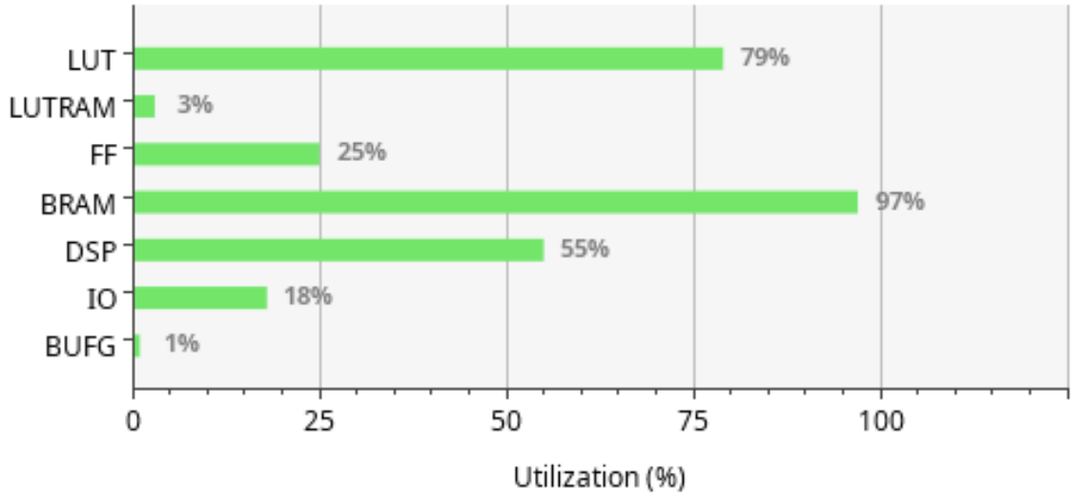
- TFlite の delegate 機構を用いた FPGA アクセラレータを開発し、動作を確認することができた
- FPGA の実行時間の内、input cache のメモリアクセス待ち時間が多くを占める
  - input data のアクセス効率を上げる工夫は有効である。画像アクセスの性質を利用し、プリフェッチするなどが考えられる
- 今回の方式は、TFlite 用に開発したネットワークを加工することなくそのまま実行することができる
  - 実装した Conv2D/dwConv2D の２つの演算のみ高速化される
- 演算の種類を増やすには、それぞれの演算に応じた回路を実装するか、より汎用性をもたせた演算回路を開発するか、回路規模と演算スピードのトレードオフになる

以上

Utilization

Summary

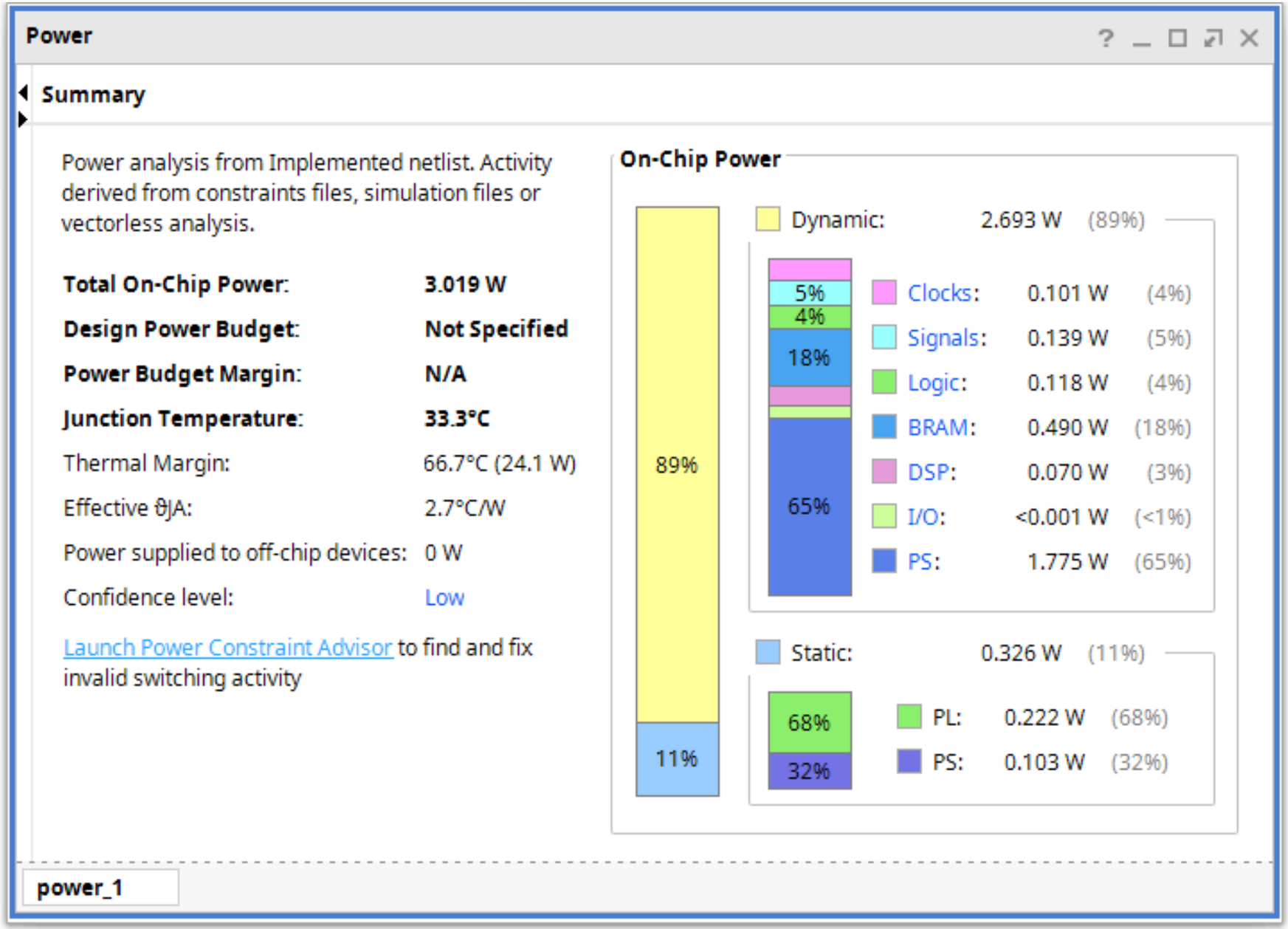
Resource	Utilization	Available	Utilization %
LUT	55612	70560	78.82
LUTRAM	963	28800	3.34
FF	35419	141120	25.10
BRAM	210	216	97.22
DSP	199	360	55.28
IO	15	82	18.29
BUFG	2	196	1.02



utilization_1

Utilization

Np : 32
clock: 150 MHz
input-cache: 512 B/line



Power

Np : 32
clock: 150 MHz
input-cache:
512 B/line

Block Design に２つの RTL ブロックを読み込んで接続