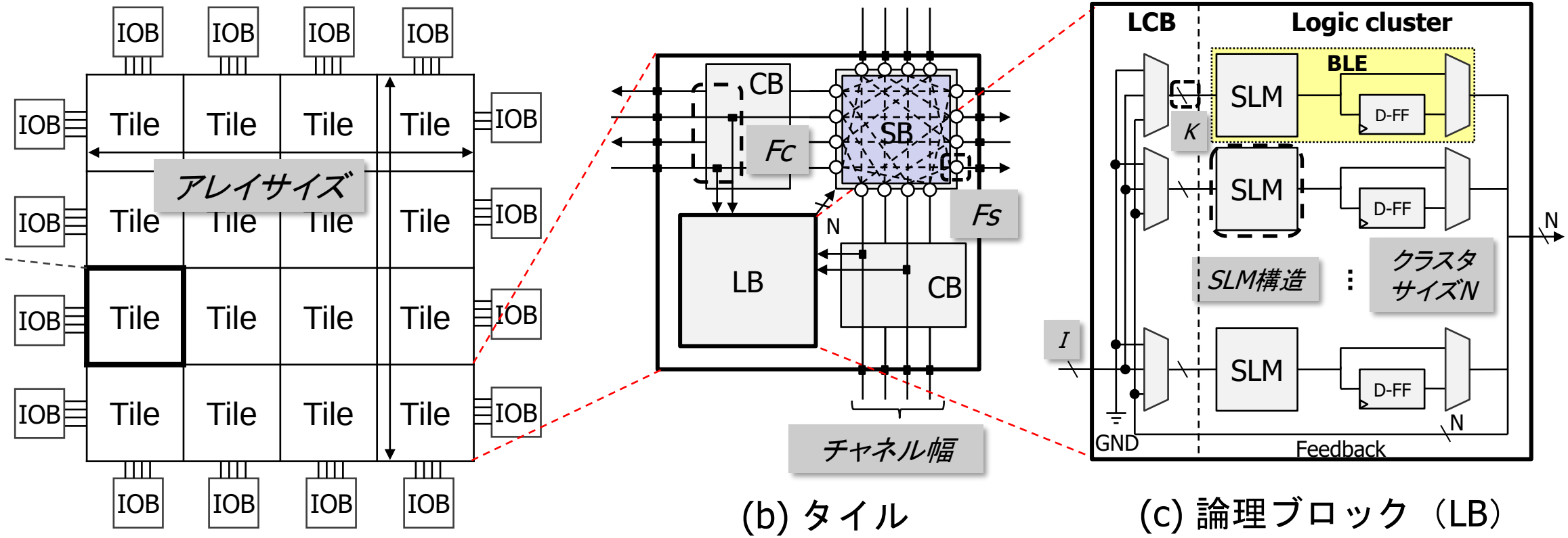
CRESTコンピューティング基盤領域

試作チップ（TEG1）のデモンストレーション

熊本大学　飯田グループ

TEG1　FPGA-IPアーキテクチャ

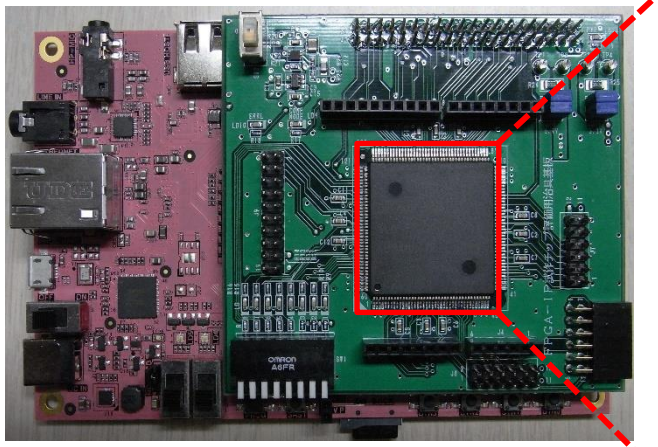
- １種類のタイルをアレイ状に配置
- 周囲のタイルに入出力のためのIOBを配置



(a) FPGAコア　(b) タイル　(c) 論理ブロック（LB）

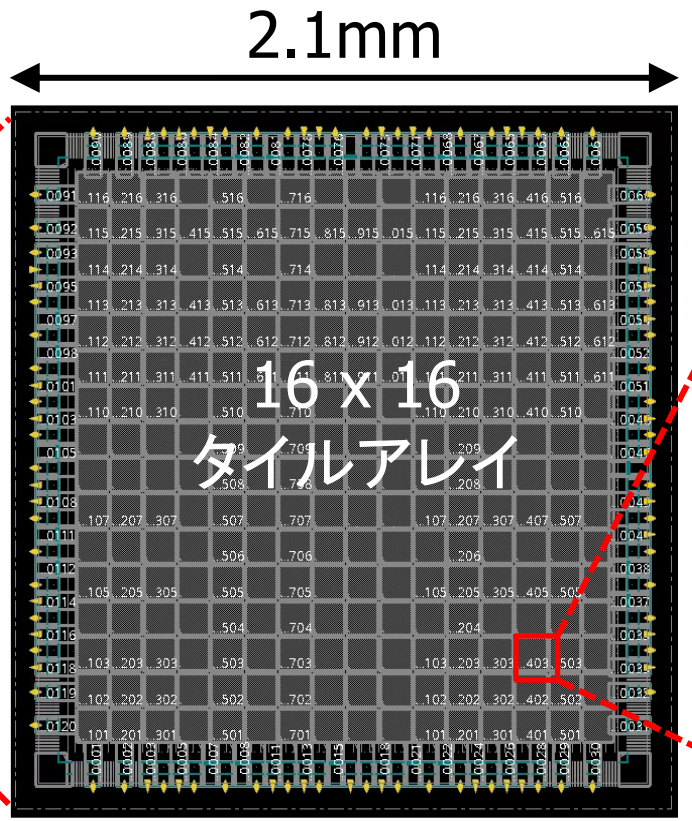
LCB：Local Connection Block　CB：Connection Block　SB：Switch Block
BLE：Basic Logic Element　LB：Logic Block　IOB：I/O Block
SLM：Scalable Logic Module

チップレイアウト外観・評価ボード

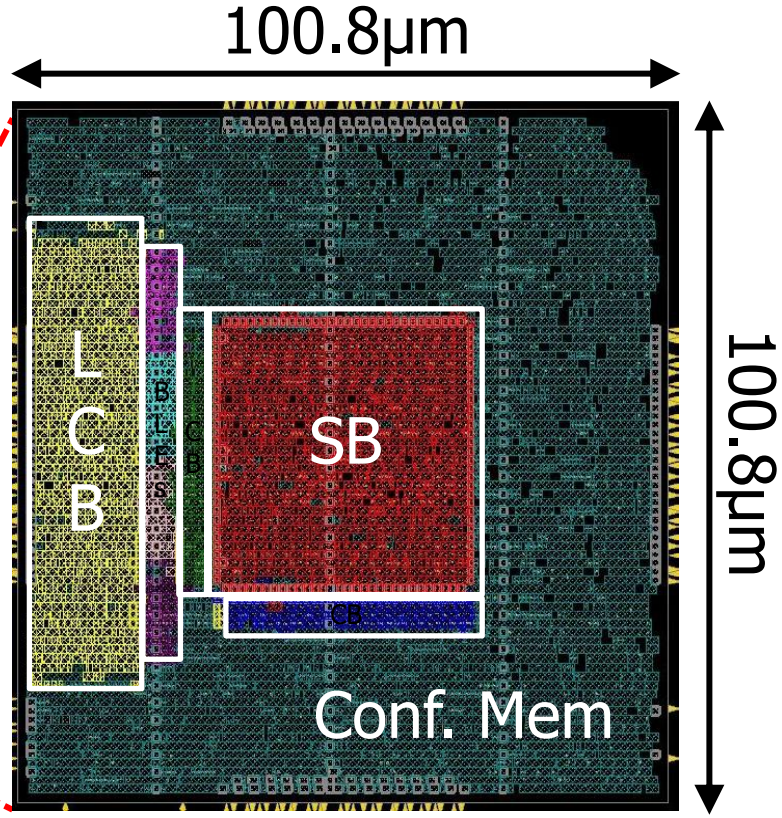


ホスト：PYNQ-Z2

(a) 評価ボード



(b) チップ

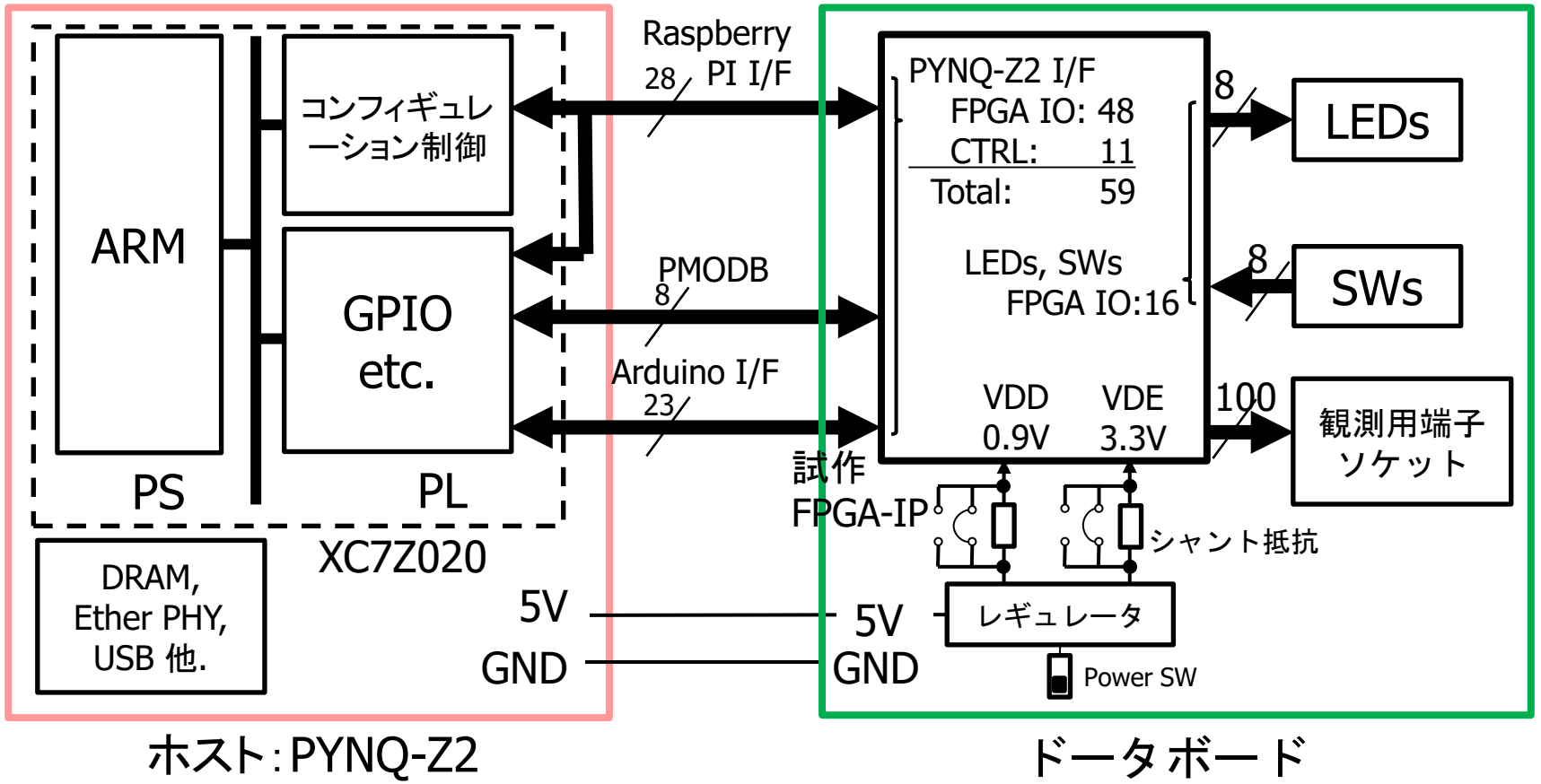


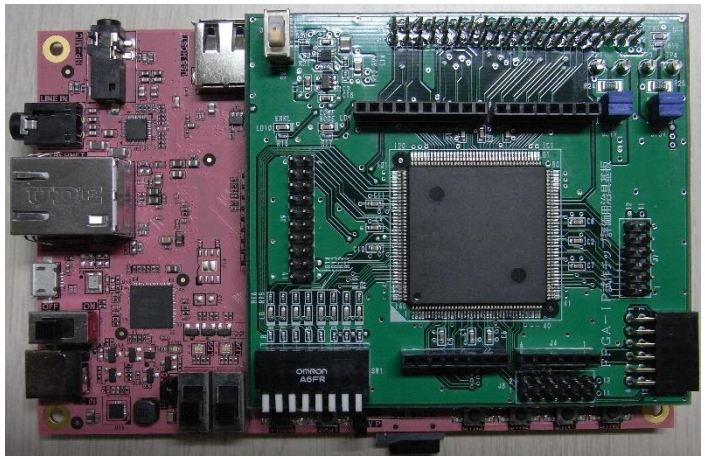
(c) タイル

評価ボード

設計方針：評価チップの制御信号およびFPGAの入出力を柔軟に制御可能に

- PYNQ-Z2ボード(Xilinx Zynq XC7Z020)のドータボードとして設計
- PYNQからPythonより制御
- FPGA-IP制御信号11本，入出力48本の計59本をPL部に接続（I/O電圧3.3Vを採用）





アプリケーション実装評価

1. 32ビットバイナリカウンタ
   - 基本動作の確認
   - 動作周波数，資源使用率，消費電力
2. ベクトル乗算
   - 16ビットデータ内にパックされた2つの8ビット符号なし数値を乗算し，16ビット符号なし数値を得る
3. 移動平均
   - 連続する8個の8ビット符号なしデータに対し単純移動平均を求める

1. バイナリカウンタ

- 24ビットカウンタの場合
  - コンフィギュレーション後，問題なく動作していることを確認

    資源利用量：　FF：24/1024(3%)，BLE：38/1024(4%)

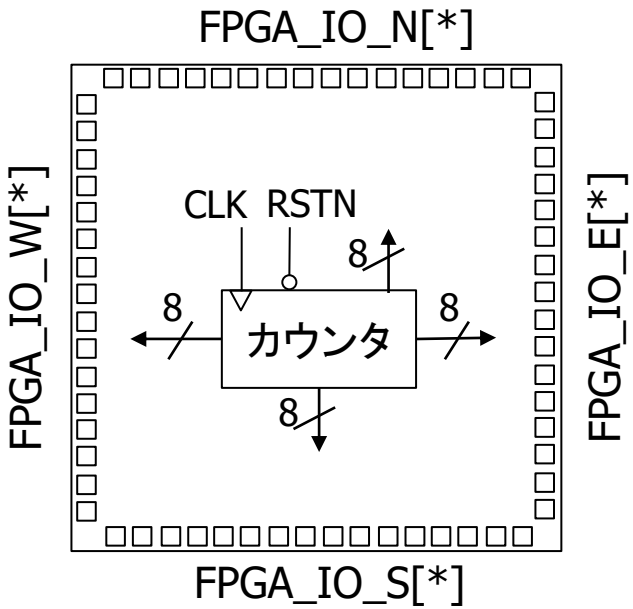
    STA解析結果：　FF-to-FF　5.658ns（176.7MHz）

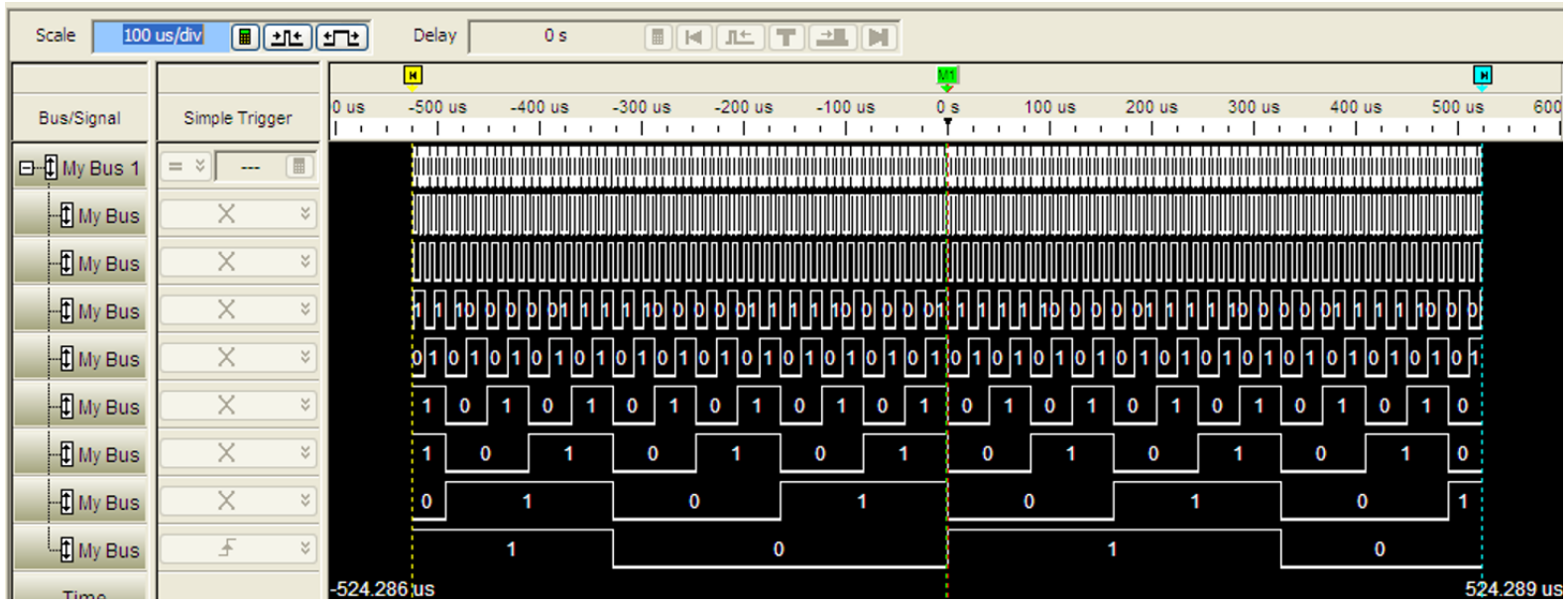
    動作周波数：(実測)

    内部：約117MHz(FF-to-FF：8.5ns)，　外部：約 83MHz(CLK→IO：12.03ns)

    IOバッファのディレイ大

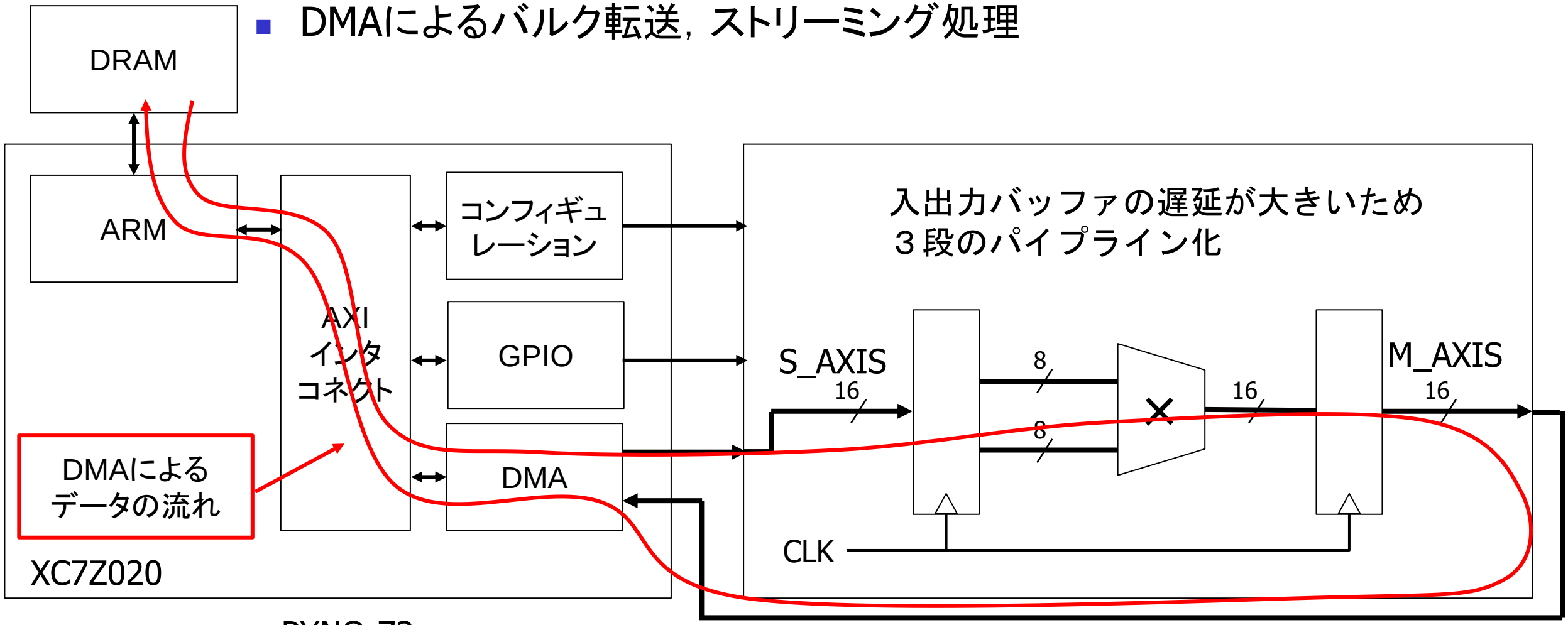
    消費電力：コア(0.9V)：162mW@117MHz





2. ベクトル乗算

- 16ビットデータ内にパックされた2つの8ビット符号なし数値を乗算し，16ビット符号なし数値を得る
- DMAによるバルク転送，ストリーミング処理

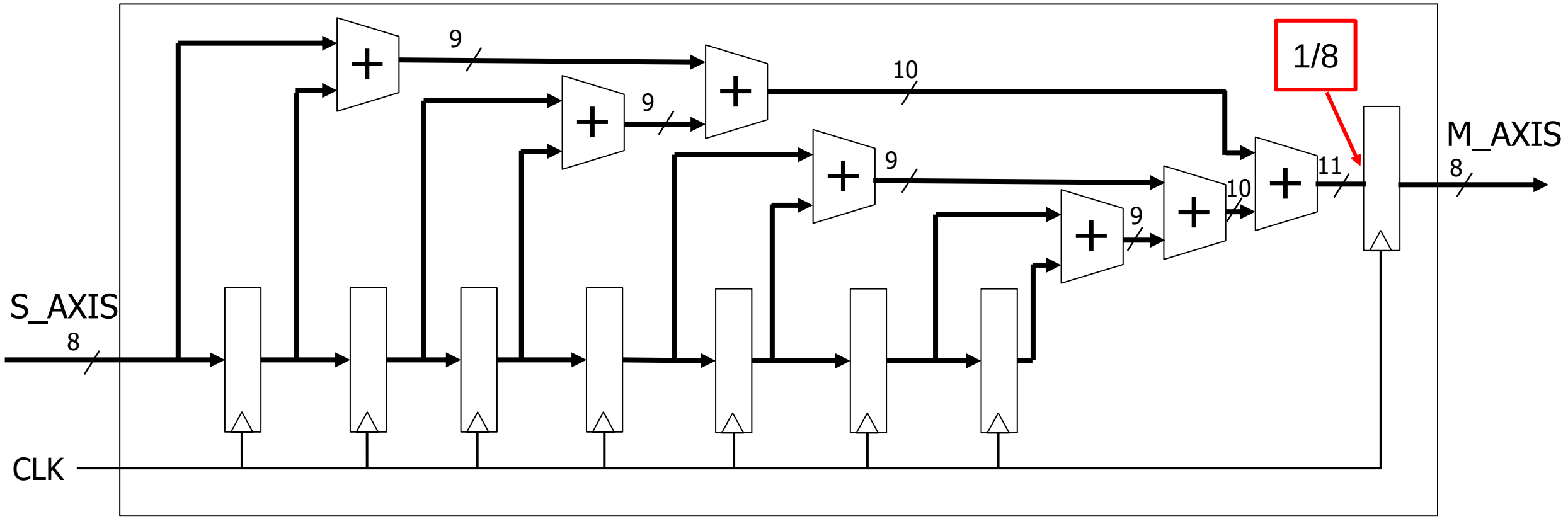


PYNQ-Z2

TEG1

3. 移動平均の計算

- 連続する8個の8ビット符号なしデータに対し単純移動平均を求める
- DMAによるバルク転送，ストリーミング処理
- セミシストリックアレイによる実装

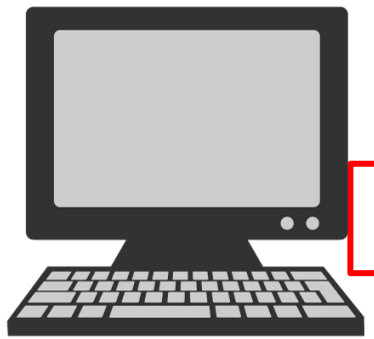


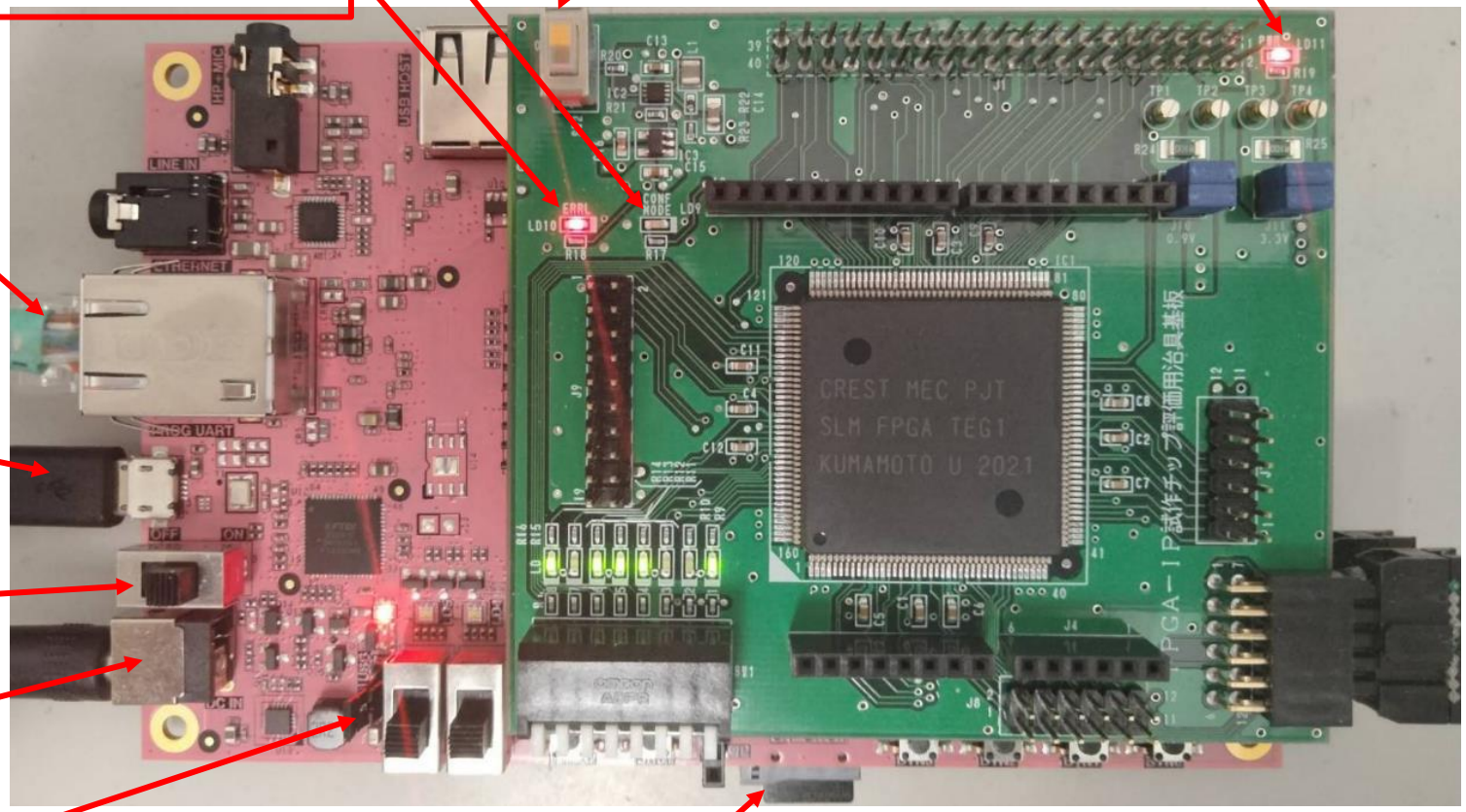
デモンストレーション　手順

32ビットバイナリカウンタの場合

9

セットアップ





CONF_MODE LED

ERRL LED（注：点灯時正常）

評価ボード
電源スイッチ（下OFF）

評価ボード電源LED

イーサネット

ホストPC

PYNQ-Z2
電源スイッチ（左OFF）

電源（12V2A）
芯線が＋

外部電源の時はショート
プラグを「REG」側にする

PYNQ-Z2

評価ボード

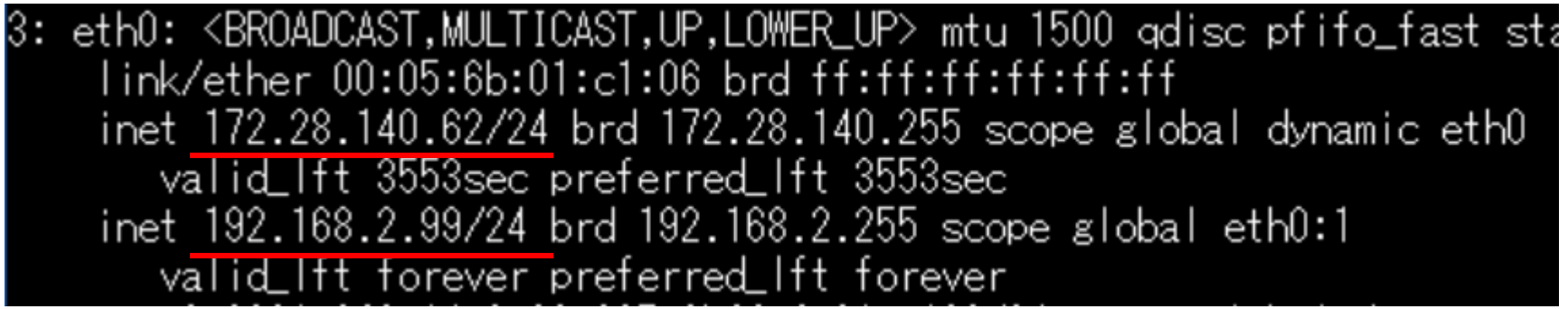
microSD
PYNQイメージ

セットアップ

1. PYNQ-Z2に以下を接続
   - 電源レギュレータ（12V2A）:
     - USB供給では容量不足で動作が不安定になることがある
   - イーサネット（UTP）
     - ホストPCからネットワーク経由でJupyter notebookを使うため
   - USBケーブル
     - ホストPCからTeratermなどでシリアル接続する
     - DHCPで割り当てられたIPアドレスを知る際に必要

セットアップ

2. セットアップ手順
   1. PYNQ-Z2の電源を入れてPYNQを動作させる
   2. Teratermを使ってホストPCからシリアルでログインする
      ID：xilinx，　PW:xilinx
      起動時は自動でログインするはずZ
   3. ip a コマンドでIPアドレスを調べる



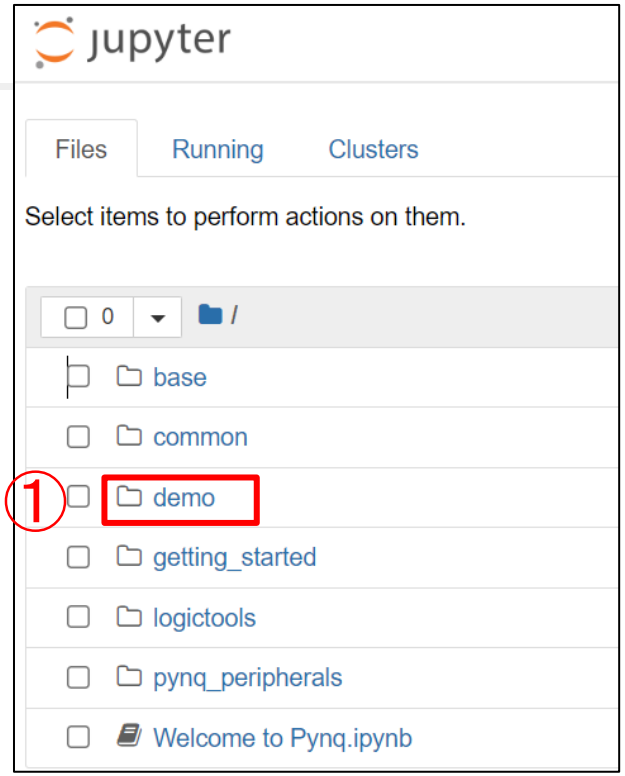
初期設定である 192.168.2.99 またはdhcpで割り当てられたIPアドレスを使う
（この例では，172.28.140.62）

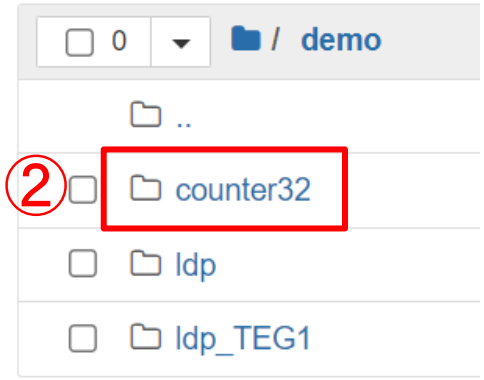
セットアップ

4. ホストPCのWebブラウザからPYNQ-Z2のJupyter notebookへ接続する
   http://調べたIPアドレス:9090/
   このときのPWも上記と同じ

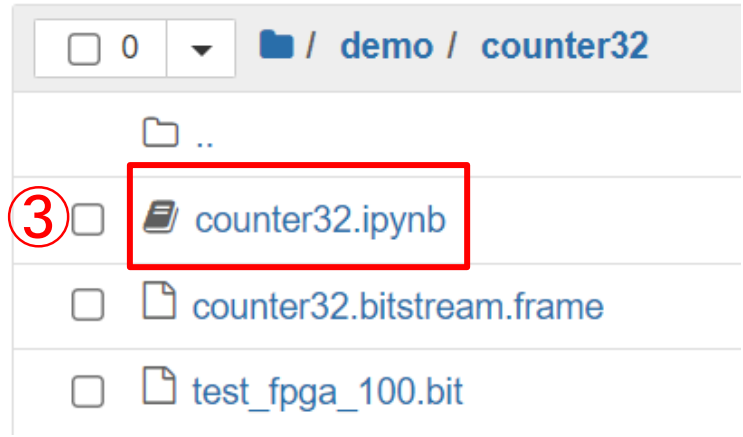
デモの実行方法

3. デモ（counter32）の実行順
   1. Jupyter notebookのFilesタブでdemoディレクトリに入る
   2. 実行するデモ（counter32）のディレクトリに入る



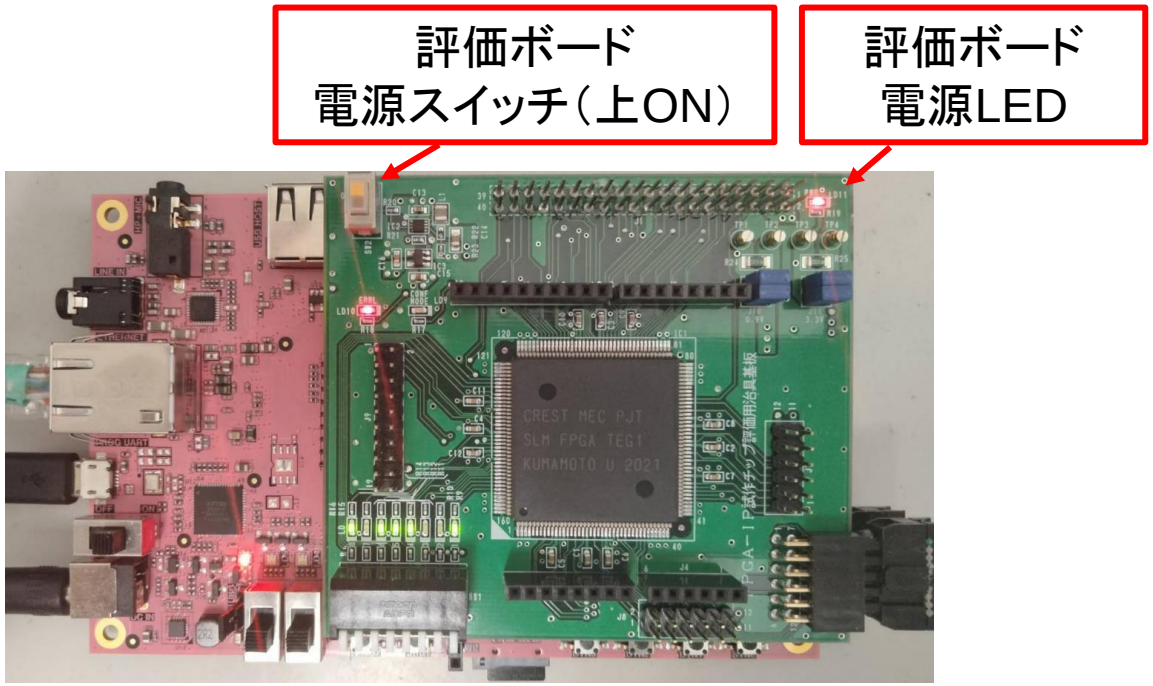


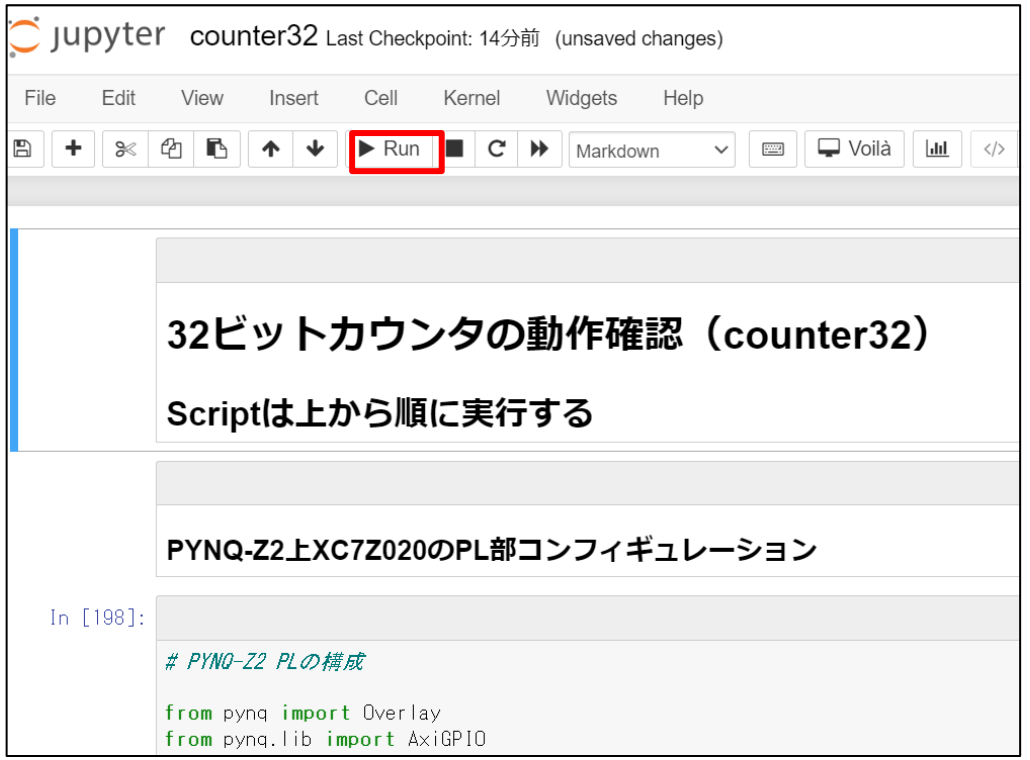
   3. “cocunter32.ipynb” をクリックしオープンする



デモの実行方法

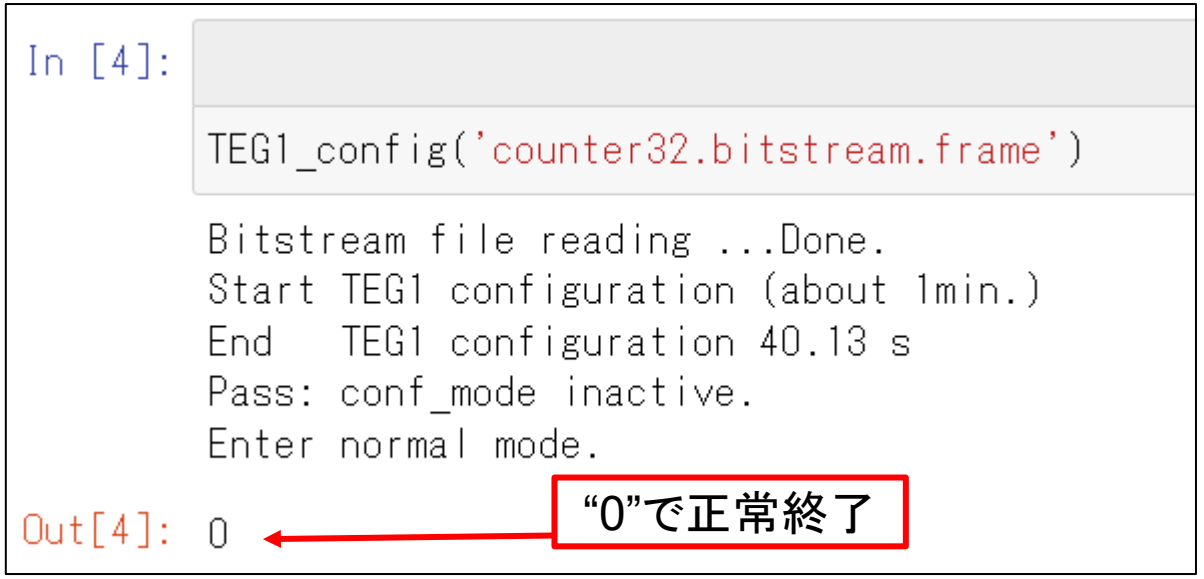
4. ページ内のpythonプログラムを上から順に実行する
   この際，「PYNQ-Z2上XC7Z020のPL部コンフィギュレーション」を実行し，
   PYNQ-Z2上のZynqのPL部をオーバーレイした後に，
   評価ボードの電源スイッチをON（上側）にする．
   （PYNQ-Z2の各IOには保護抵抗が入っているので壊れはしないが，
   安全のためにはこの手順が良い）





デモの実行方法

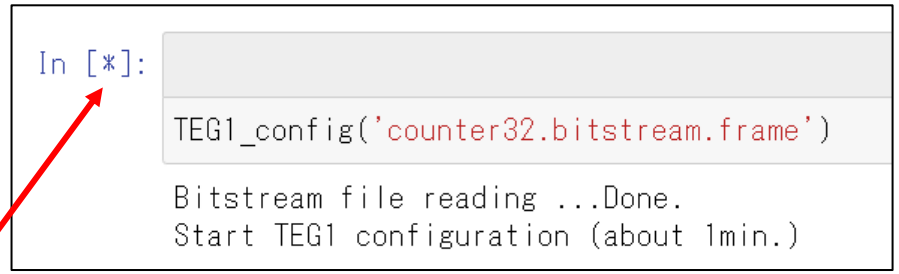
5. 「TEG1のコンフィギュレーション」でFPGA-IP（TEG1）のコンフィギュレーションを行う．プログラムでTCKを制御している関係から構成には約1分かかる．
構成が進んでいるかどうかは，評価ボード左上の緑LED（CONF_MODE）が点滅することで確認できる．
なお，CRCエラーが出た際は，評価ボード左上の赤LED（ERRL）が消灯する．



※電源を入れて最初の実行では，“Bitstream file reading”のメッセージが出るまで少し時間がかかります．
※初回はコンフィギュレーションに失敗することがあります．その際は次ページの手順で再実行します．

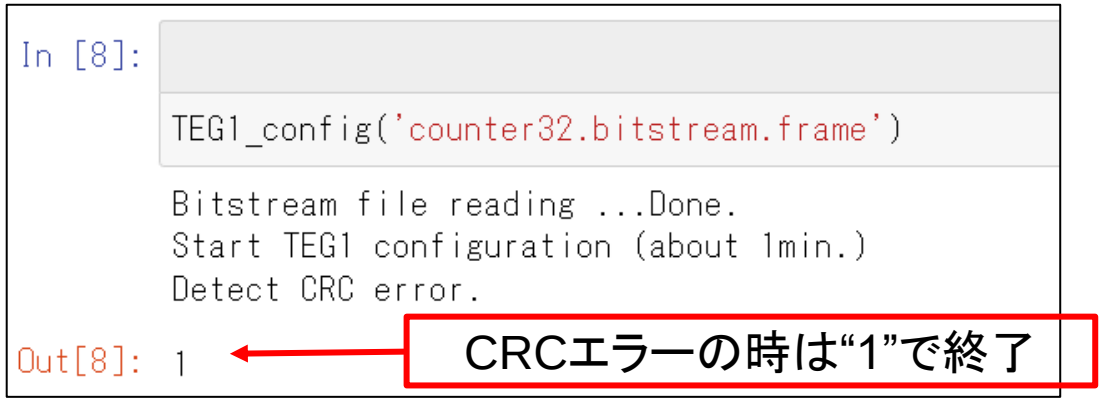
コンフィギュレーション時のトラブルへの対処

6. “Bitstraem file reading”のメッセージが出た後で，CONF_MODEのLEDが点滅していないときは，プロトコルエラーが発生している．この場合，プログラムが終了しない
また，ERRLのLEDが消灯したときはCRCエラーが発生している．
プログラムが終了していないとき（[*]の表示）の時は，Web画面上部の
■を押して終了させる．
その後，プログラムの先頭から再実行する．
（ボードの電源スイッチはONにしたままでよい）



プログラム実行中

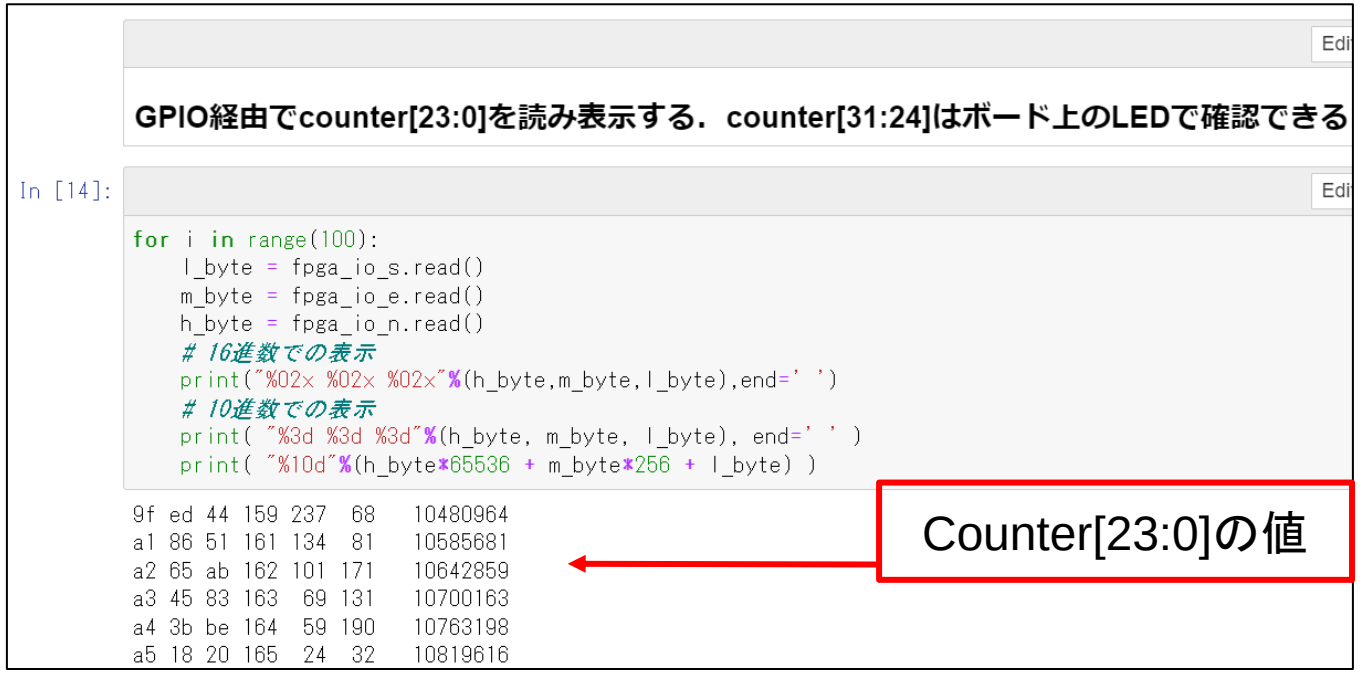
実行中のプログラムは■で止める



CRCエラーの時は“1”で終了

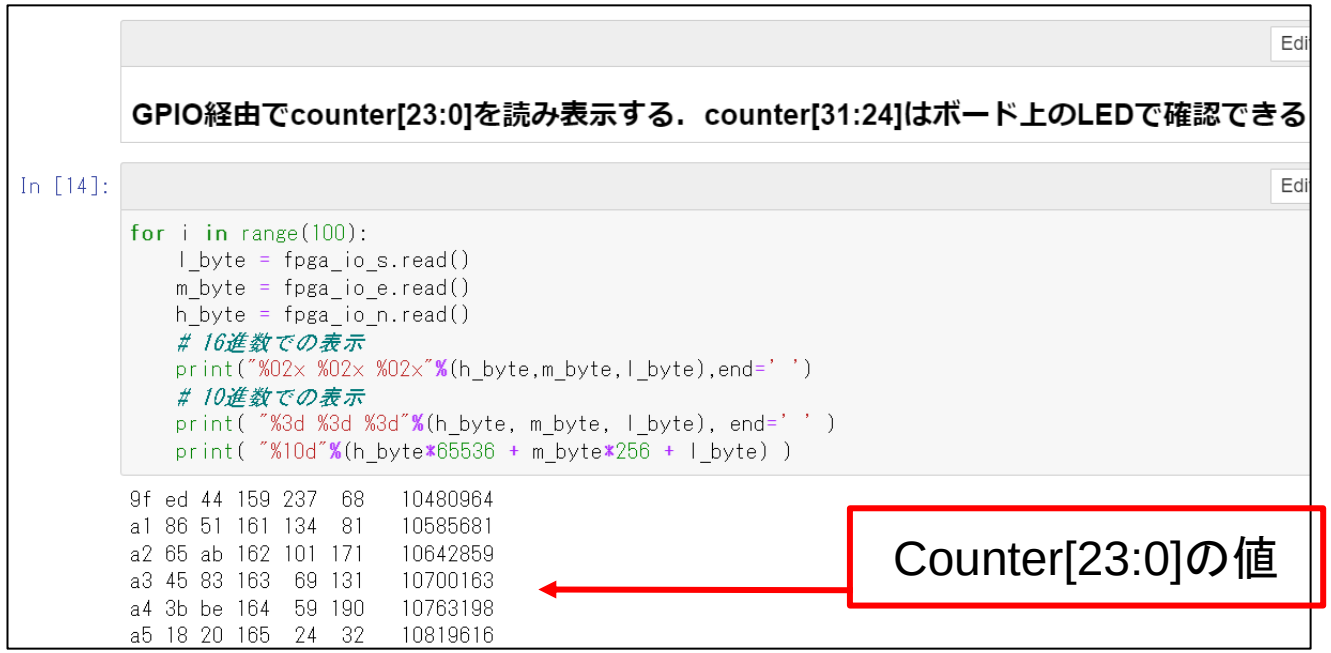
デモの実行方法

7. コンフィギュレーション後は，直ちにカウンタが動作する．LEDで確認できる．
8. カウンタのリセット（負論理）はDIPスイッチの第0ビット．
   スイッチを下にするとカウンタが0になり停止．
   スイッチを上にするとカウントアップを開始．
9. 「GPIO経由でcounter[23:0]を読み表示する」を実行すると
   カウンタのcounter[23:0]をGPIOを介して読み出し表示する．



デモの実行方法

10. カウンタのリセット（負論理）はDIPスイッチの第0ビット.
    スイッチを下にするとカウンタが0になり停止.
    スイッチを上にするとカウントアップを開始.
11. 「GPIO経由でcounter[23:0]を読み表示する」を実行すると
    カウンタのcounter[23:0]をGPIOを介して読み出し表示する.



12. 再構成する際は最初から実行する

デモンストレーション　手順

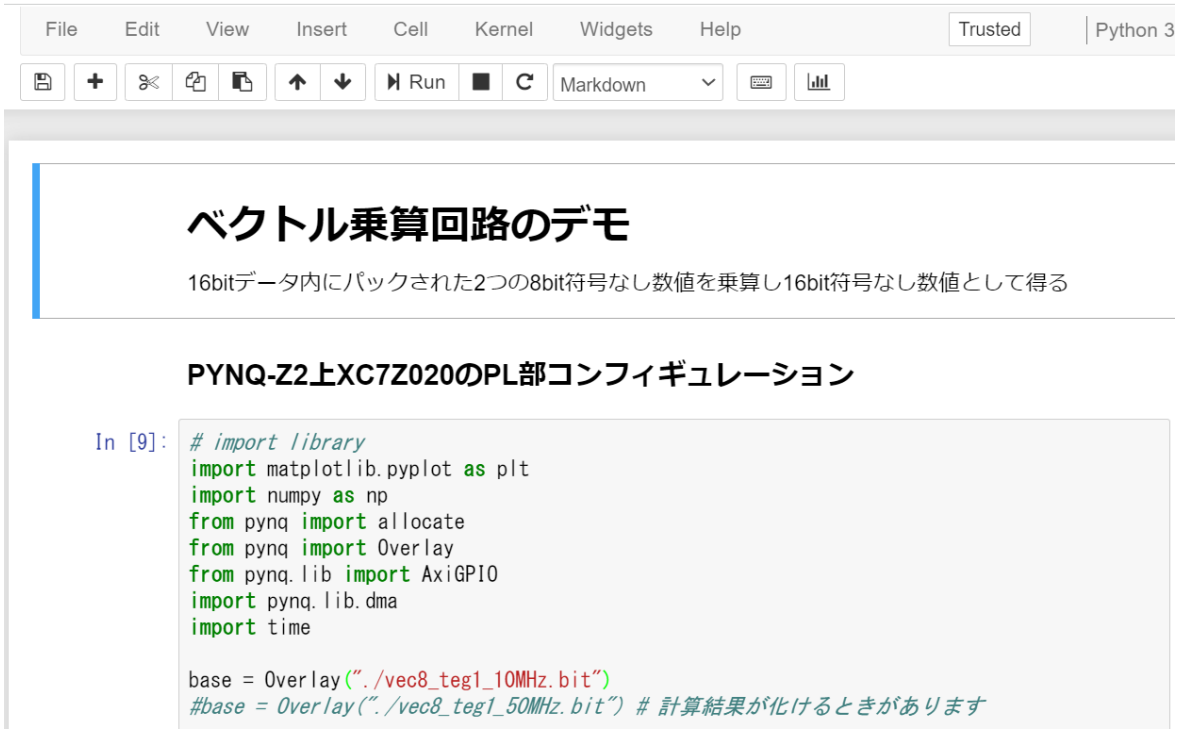
ベクトル乗算の場合

20

デモの実行方法

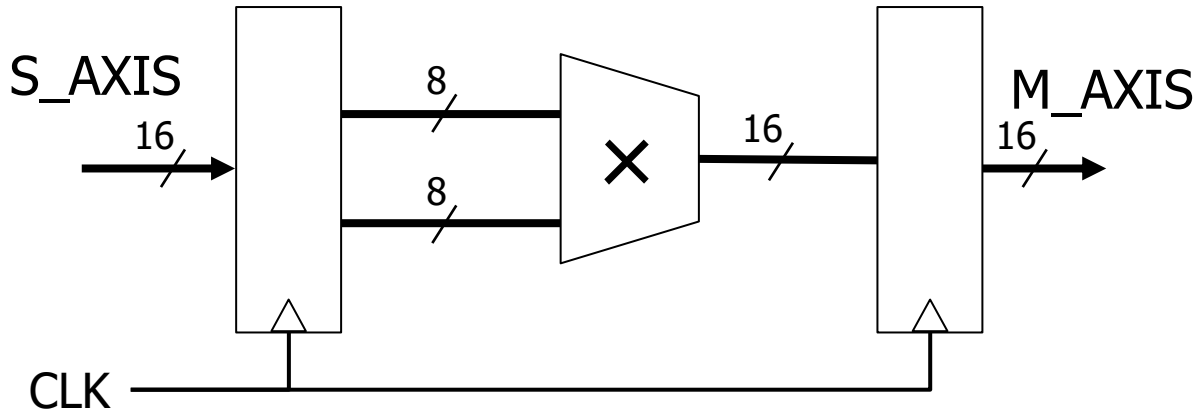
基本的な操作は，32ビットバイナリカウンタと同様

1. Jupyter notebookのFilesタブで “demo” ディレクトリに入る
2. 実行するデモ “vec8” のディレクトリに入る
3. “vec8.ipynb” をクリックしオープンする



デモの実行方法

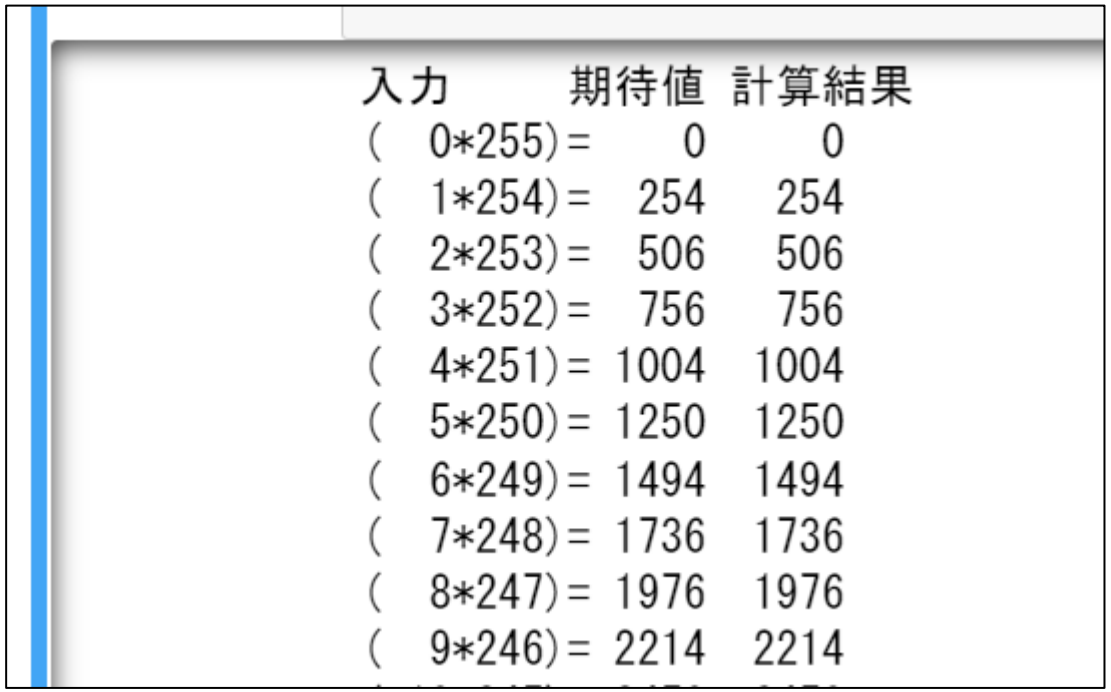
4. ページ内のpythonプログラムを上から順に実行する
   入力データ
   - 16ビットの上位バイトと下位バイトに，2つの8ビット符号なし数値をパッキング
   - 上位バイトと下位バイトを乗算後，乗算結果の16ビット符号なし数値をリターンする



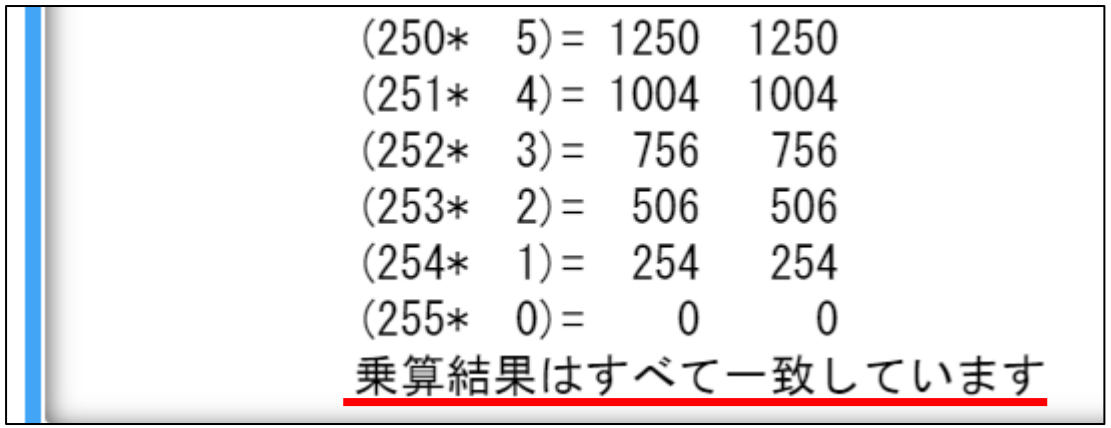

```
入力        期待値  計算結果
(  0*255)=      0       0
(  1*254)=    254     254
(  2*253)=    506     506
(  3*252)=    756     756
(  4*251)=   1004    1004
(  5*250)=   1250    1250
(  6*249)=   1494    1494
(  7*248)=   1736    1736
(  8*247)=   1976    1976
(  9*246)=   2214    2214
```

デモの実行方法

4. 「出力データの確認」の実行により，入力データと期待値，および実行結果を表示
5. 期待値と結果が一致していれば，最後に「乗算結果はすべて一致しています」と表示



```
入力        期待値  計算結果
(   0*255)=      0       0
(   1*254)=    254     254
(   2*253)=    506     506
(   3*252)=    756     756
(   4*251)=   1004    1004
(   5*250)=   1250    1250
(   6*249)=   1494    1494
(   7*248)=   1736    1736
(   8*247)=   1976    1976
(   9*246)=   2214    2214
```



```
(250*   5)=  1250    1250
(251*   4)=  1004    1004
(252*   3)=   756     756
(253*   2)=   506     506
(254*   1)=   254     254
(255*   0)=     0       0
乗算結果はすべて一致しています
```

デモンストレーション　手順

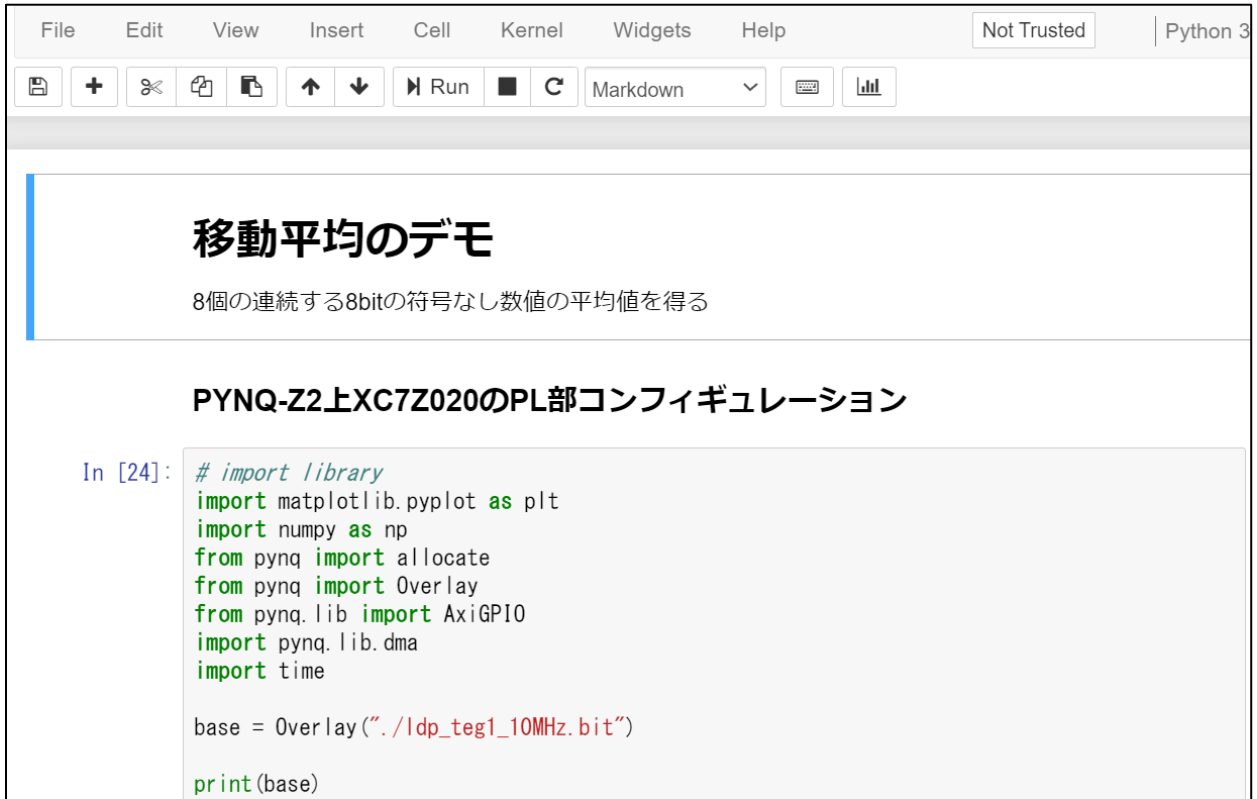
ベクトル乗算の場合

24

デモの実行方法

基本的な操作は，32ビットバイナリカウンタと同様

1. Jupyter notebookのFilesタブで “demo” ディレクトリに入る
2. 実行するデモ “sma8” のディレクトリに入る
3. “sma8.ipynb” をクリックしオープンする

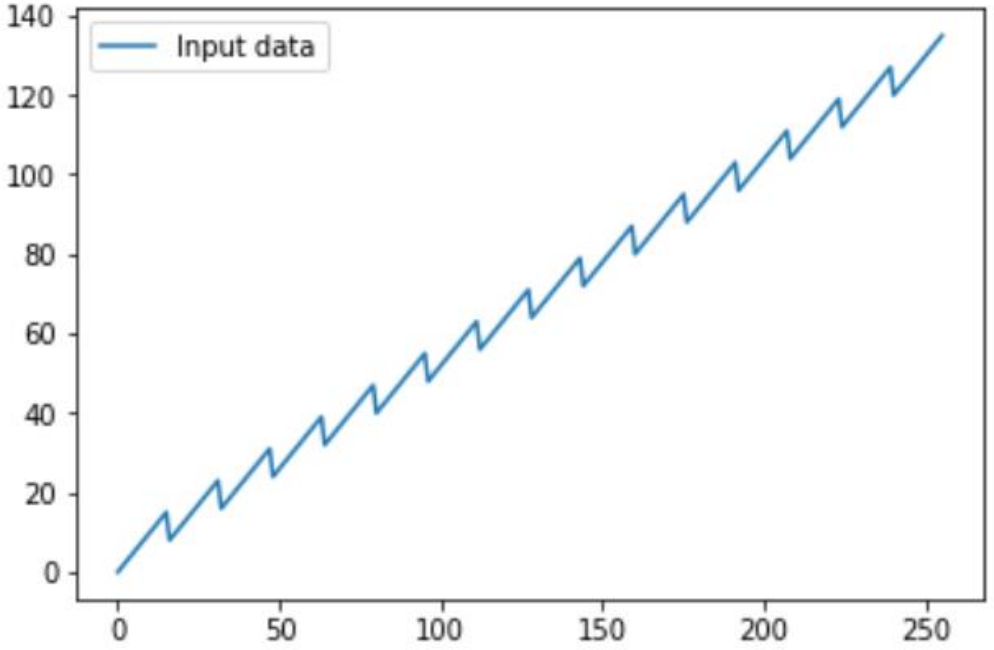


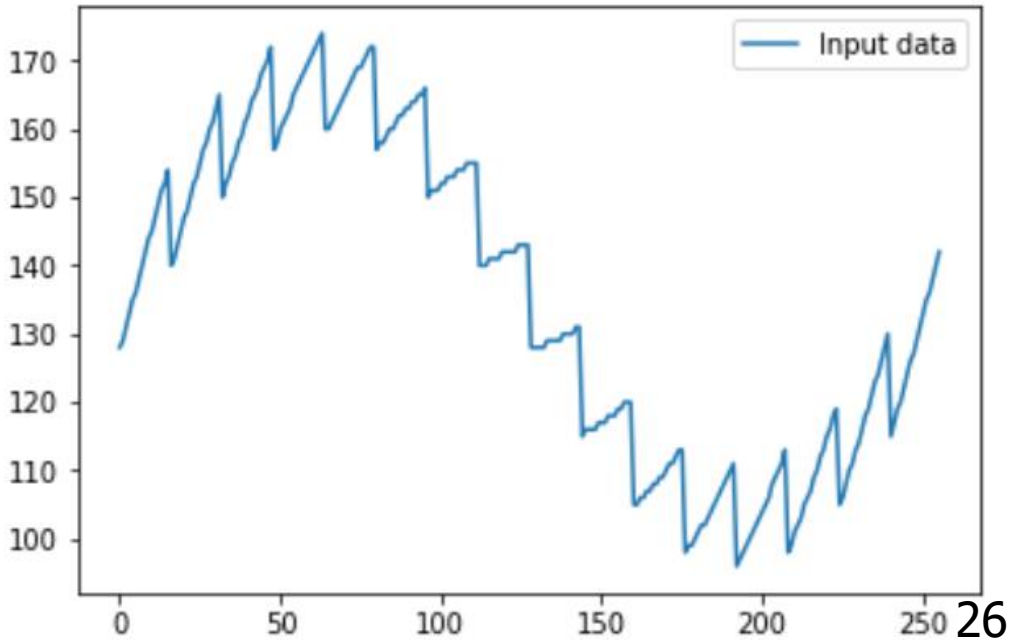
デモの実行方法

4. ページ内のpythonプログラムを上から順に実行する

入力データは以下の2種類を準備

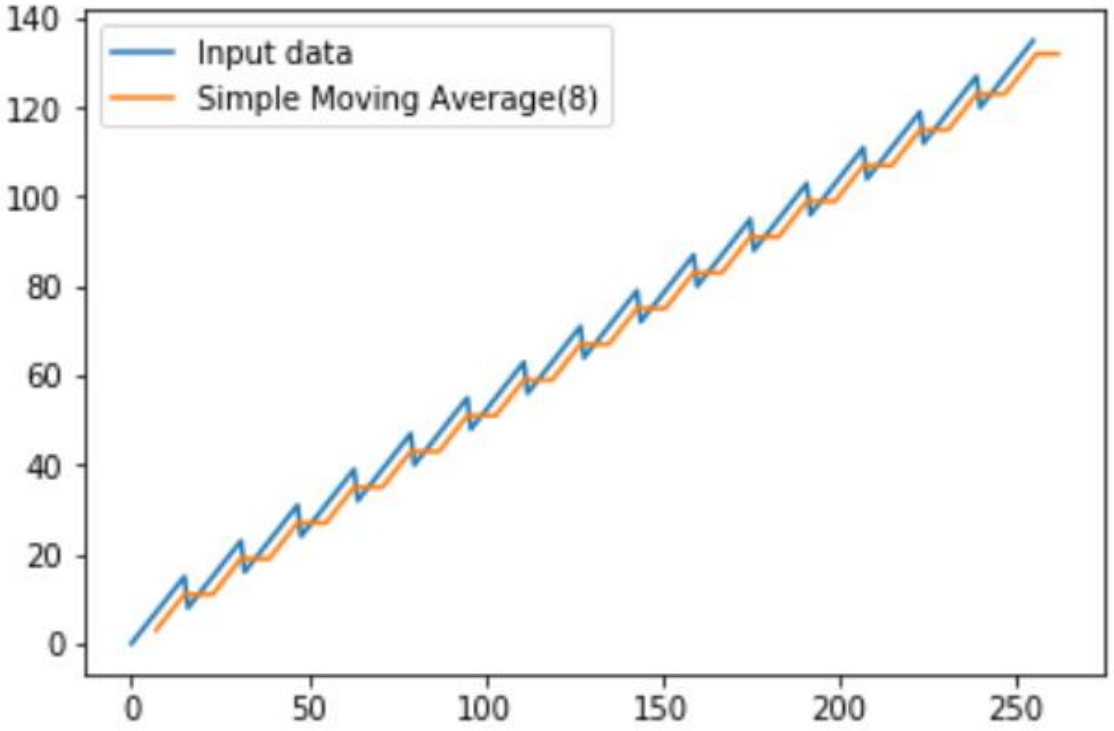
- 次第に大きくなる鋸波
  - $(i \% 16) + (i // 16) * 8$
- sin波に乗った鋸波
  - $(i \% 16) + 32 * \sin(2*\pi*i/N) + 128$



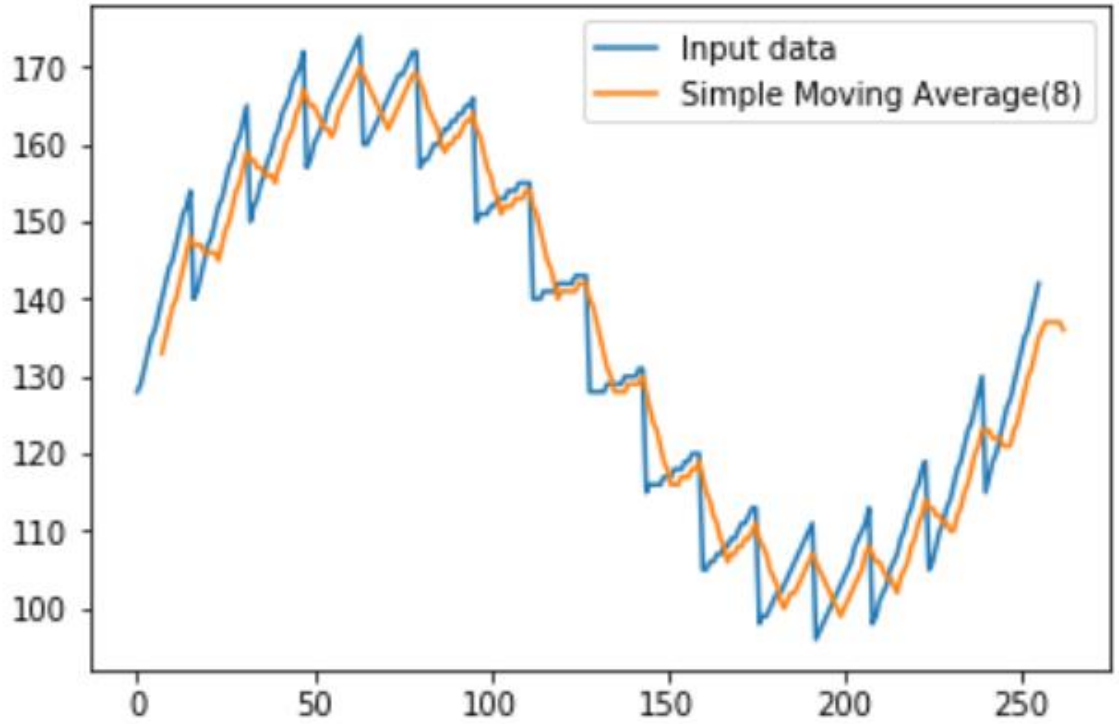


デモの実行方法

5. 実行結果



次第に大きくなる鋸波の結果



sin波に乗った鋸波の結果