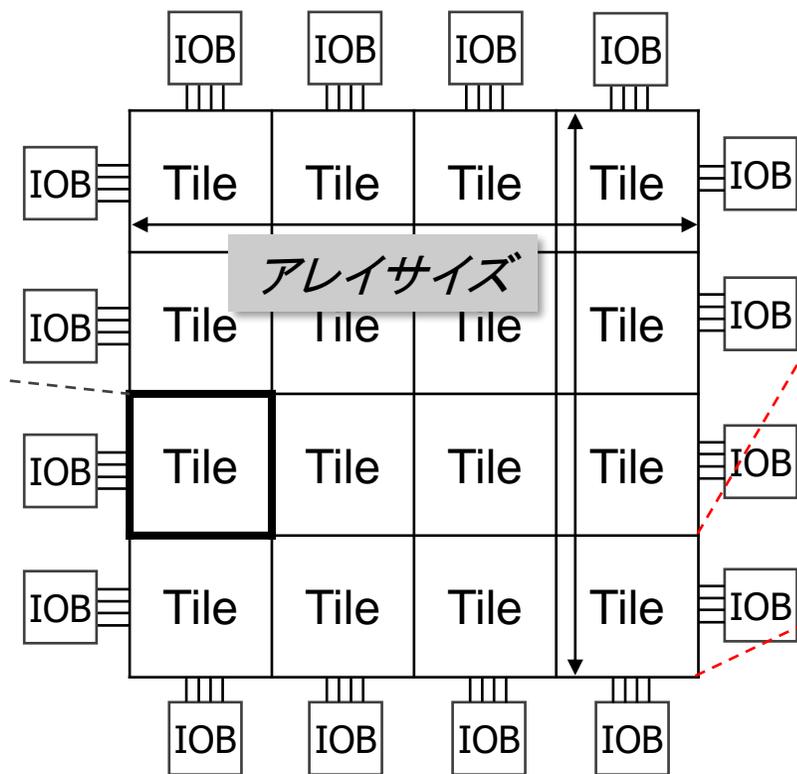


CRESTコンピューティング基盤領域

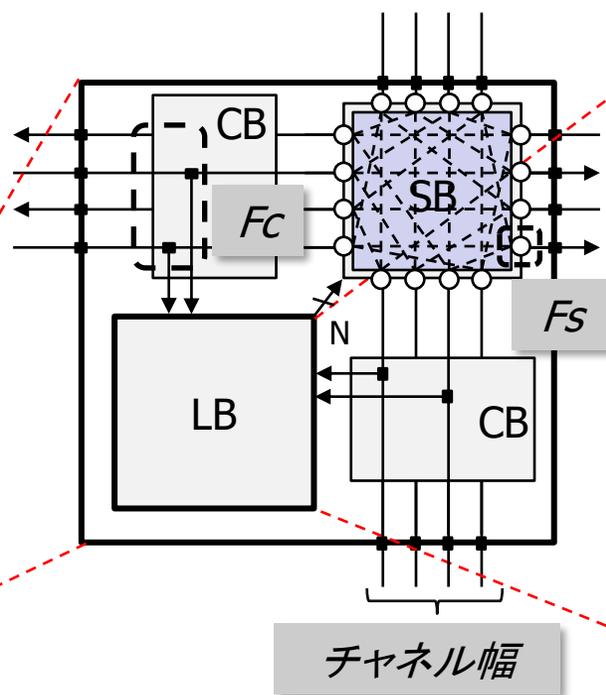
試作チップ(TEG1)のデモンストレーション
熊本大学 飯田グループ

TEG1 FPGA-IPアーキテクチャ

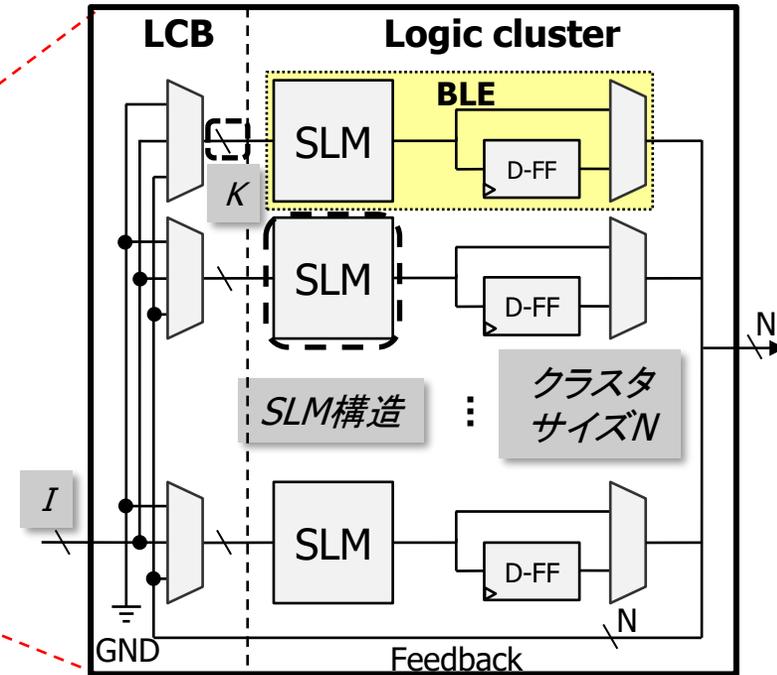
- 1種類のタイルをアレイ状に配置
- 周囲のタイルに入出力のためのIOBを配置



(a) FPGAコア



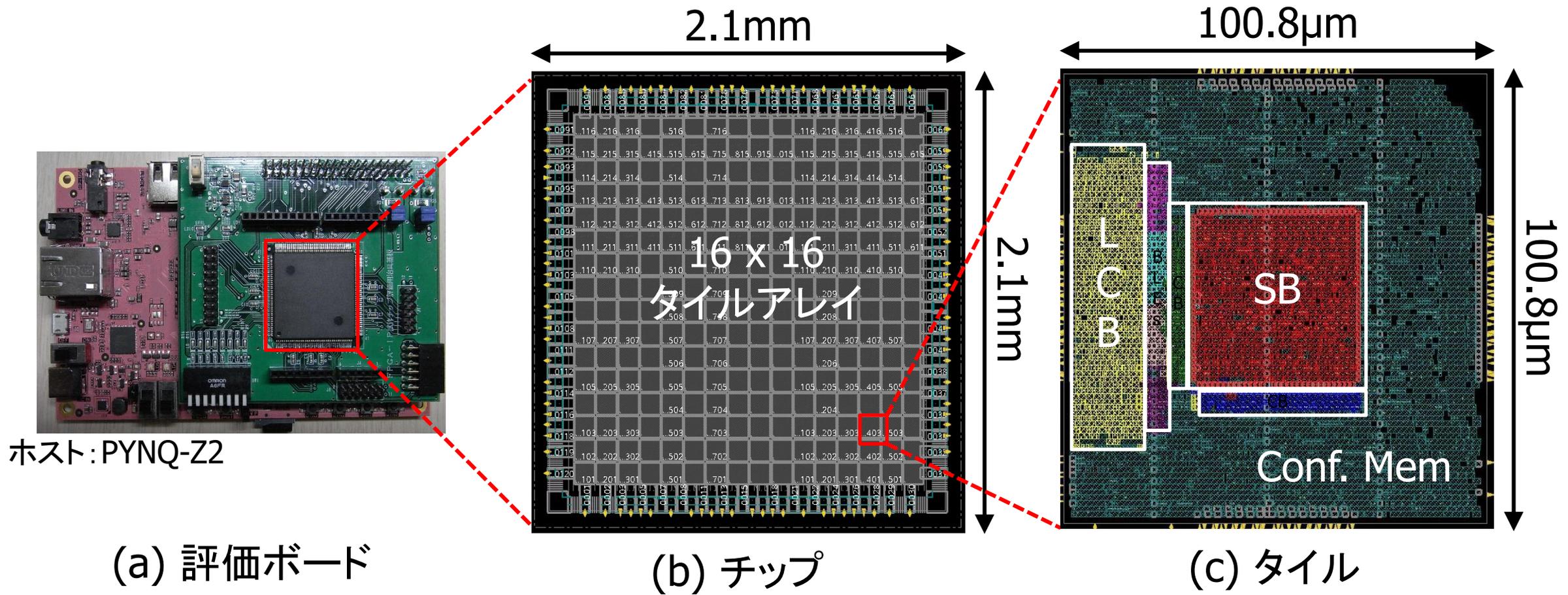
(b) タイル



(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

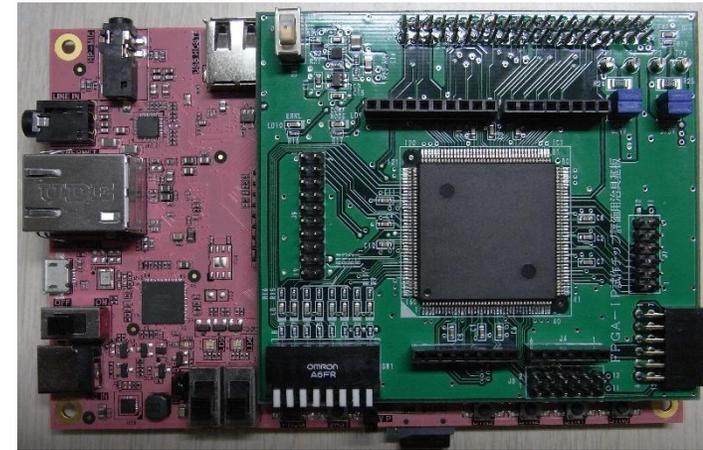
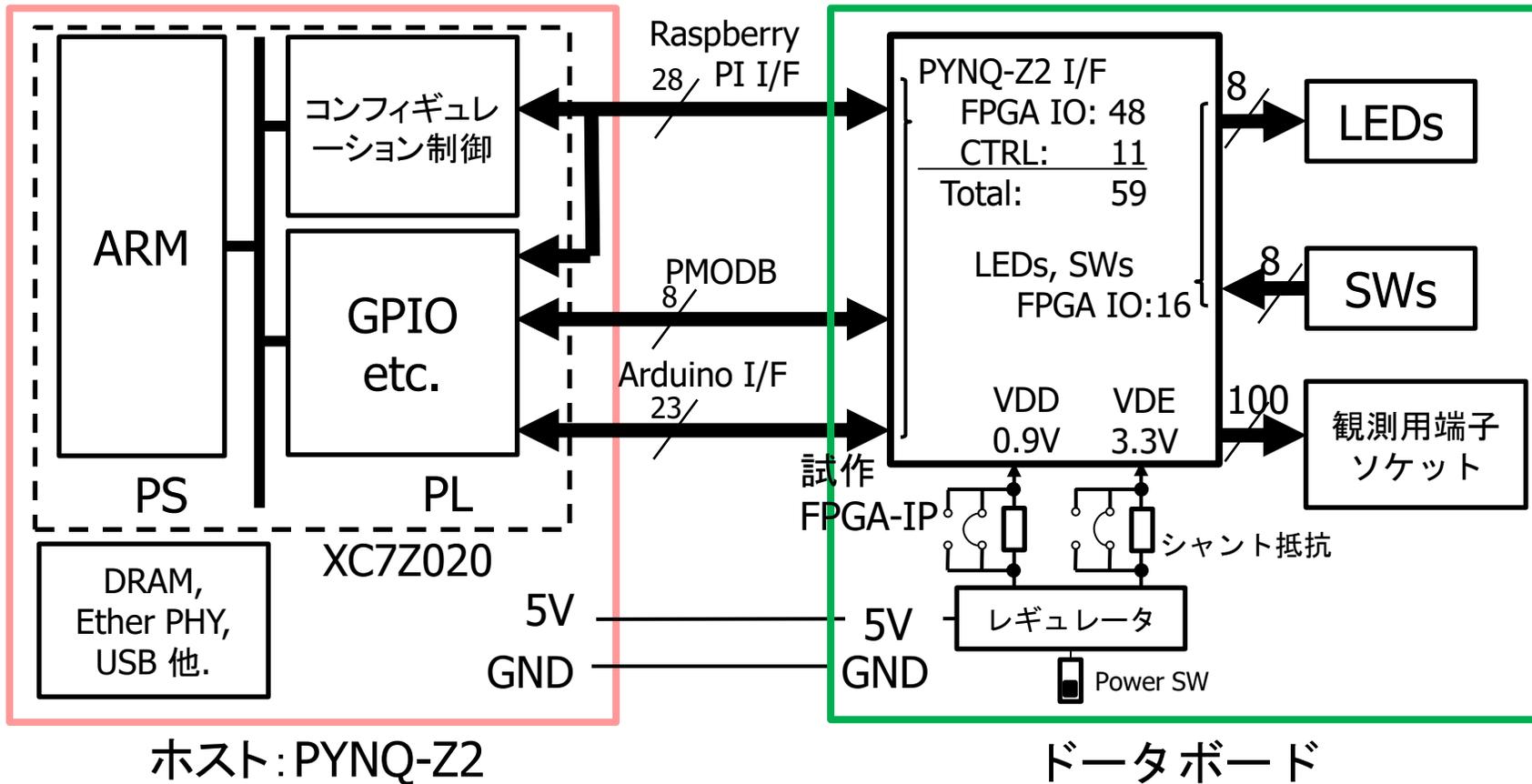
チップレイアウト外観・評価ボード



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）



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%)

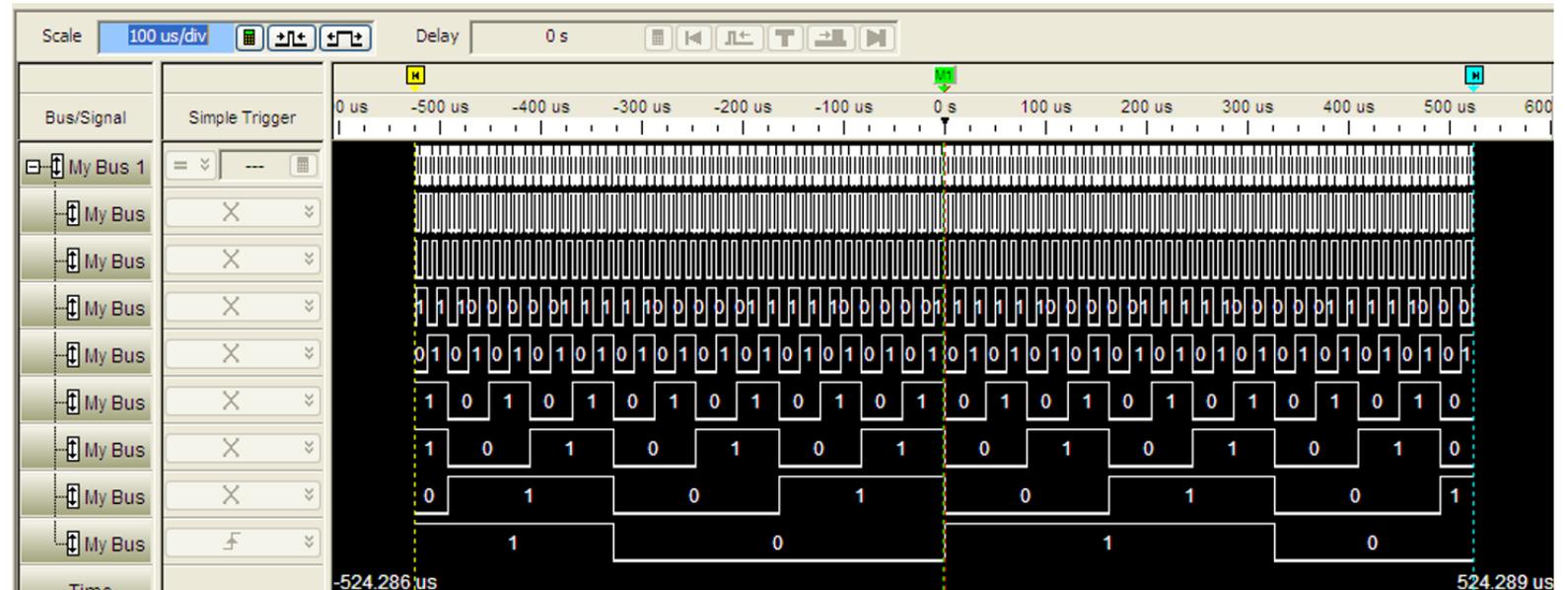
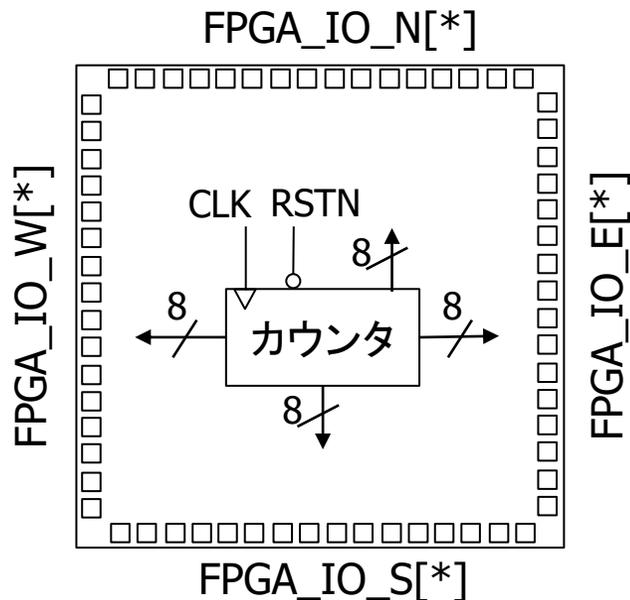
STA解析結果 : FF-to-FF 5.658ns (176.7MHz)

動作周波数 : (実測)

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

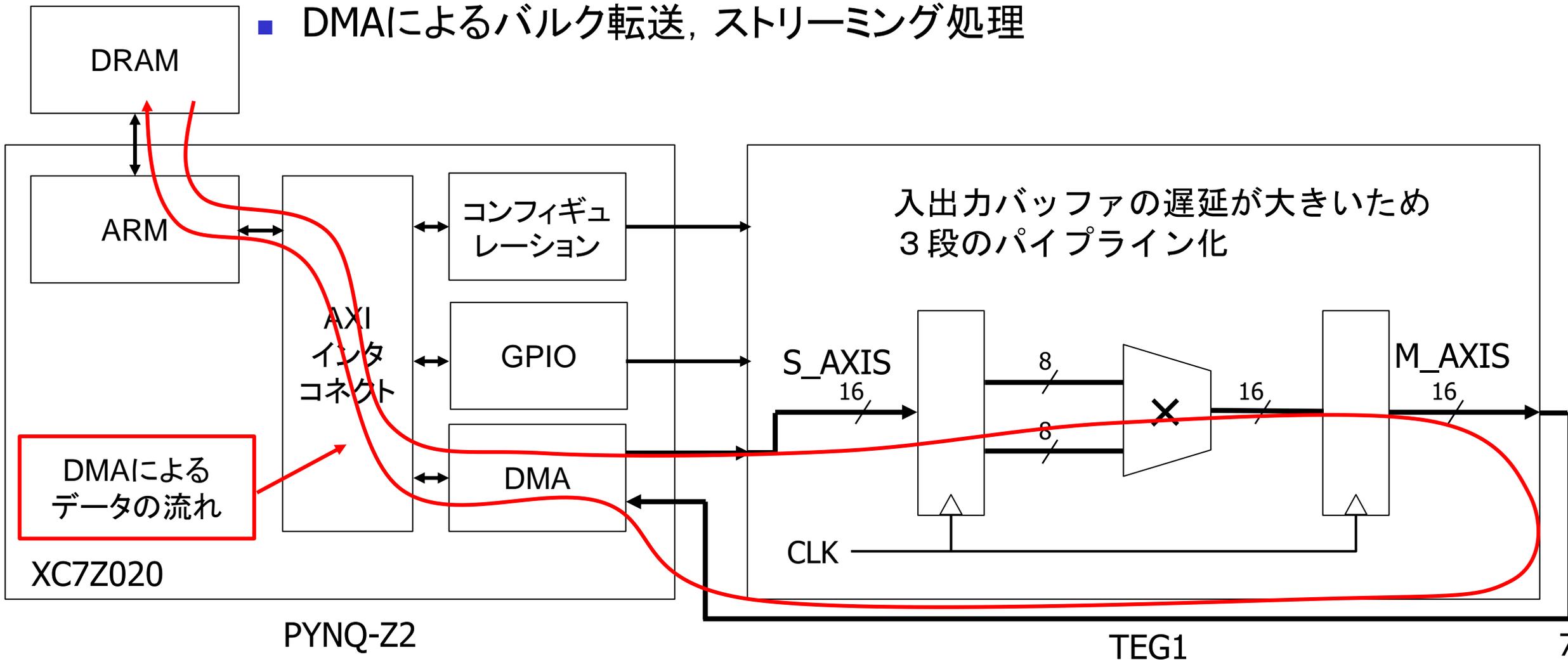
I/Oバッファの遅延大

消費電力 : コア (0.9V) : 162mW@117MHz



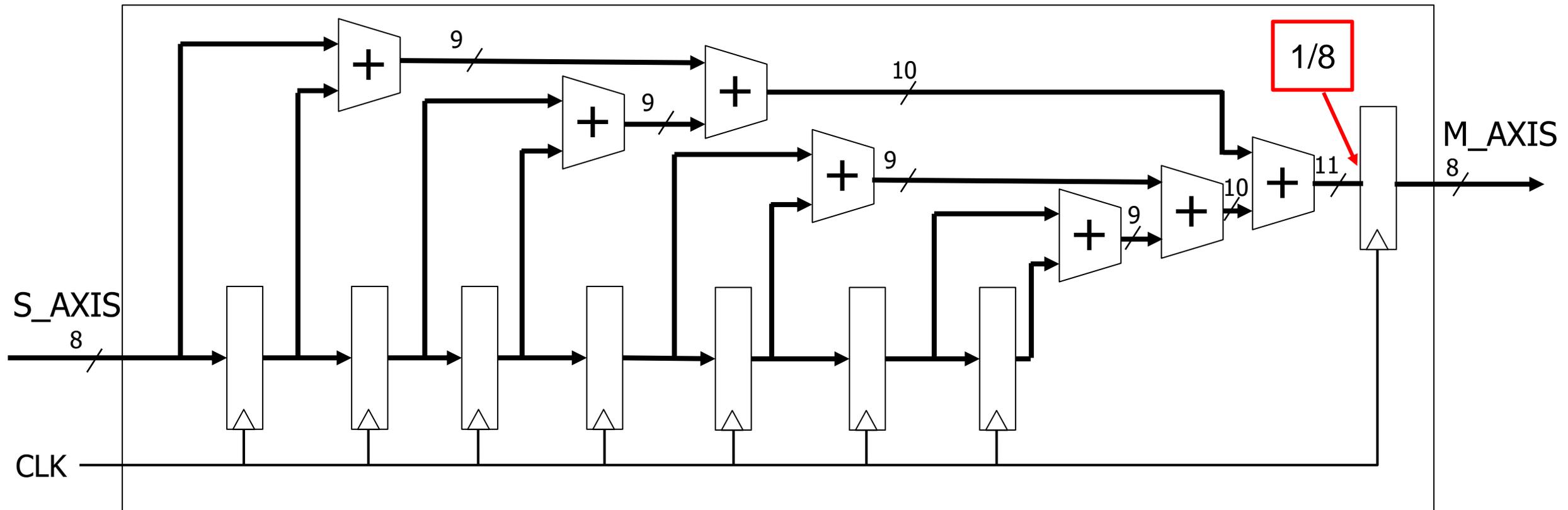
2. ベクトル乗算

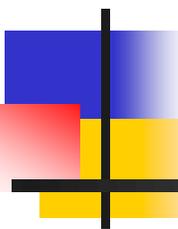
- 16ビットデータ内にパックされた2つの8ビット符号なし数値を乗算し、16ビット符号なし数値を得る
- DMAによるバルク転送, ストリーミング処理



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

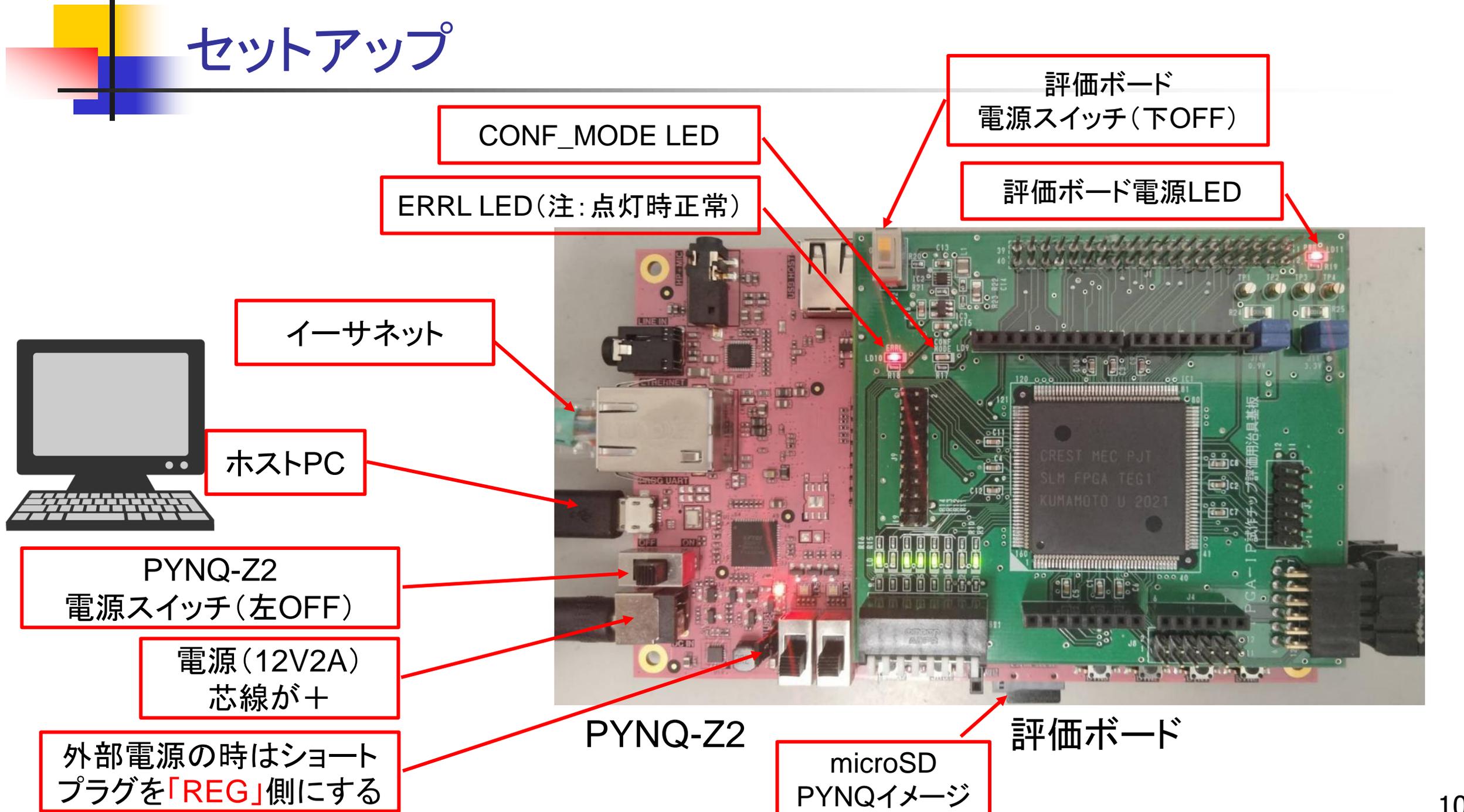




デモンストレーション 手順

32ビットバイナリカウンタの場合

セットアップ



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

Host 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

microSD
PYNQイメージ

評価ボード

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

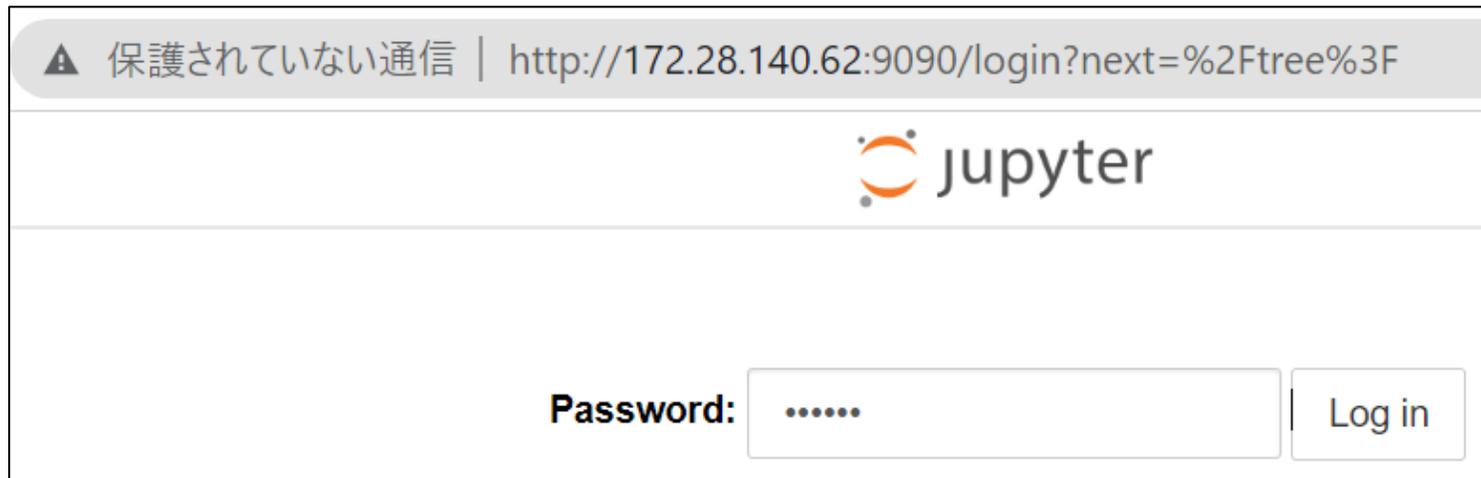
1. PYNQ-Z2の電源を入れてPYNQを動作させる
2. Teratermを使ってホストPCからシリアルでログインする
ID:xilinx, PW:xilinx
起動時は自動でログインするはずZ
3. `ip a` コマンドでIPアドレスを調べる

```
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast sta
link/ether 00:05:6b:01:c1:06 brd ff:ff:ff:ff:ff:ff
inet 172.28.140.62/24 brd 172.28.140.255 scope global dynamic eth0
    valid_lft 3553sec preferred_lft 3553sec
inet 192.168.2.99/24 brd 192.168.2.255 scope global eth0:1
    valid_lft forever preferred_lft forever
```

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



▲ 保護されていない通信 | http://172.28.140.62:9090/login?next=%2Ftree%3F

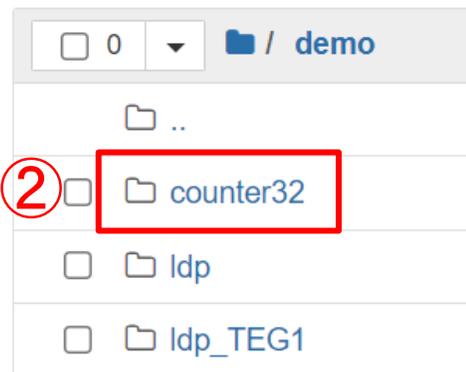
jupyter

Password:

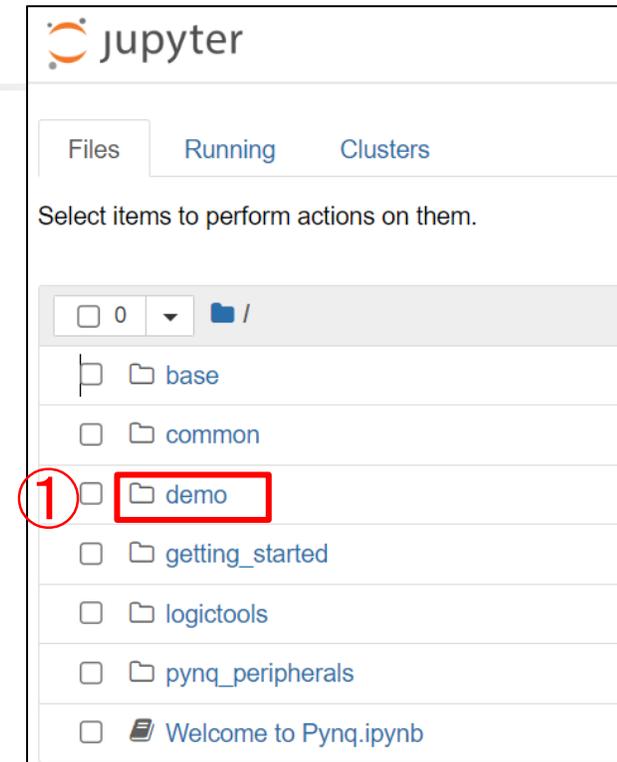
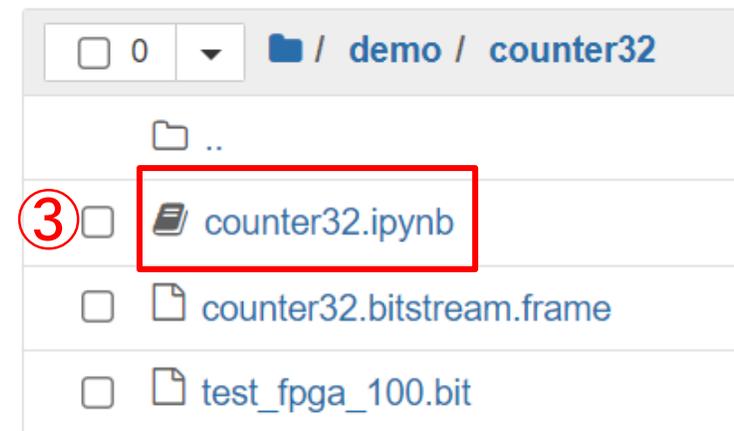
デモの実行方法

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

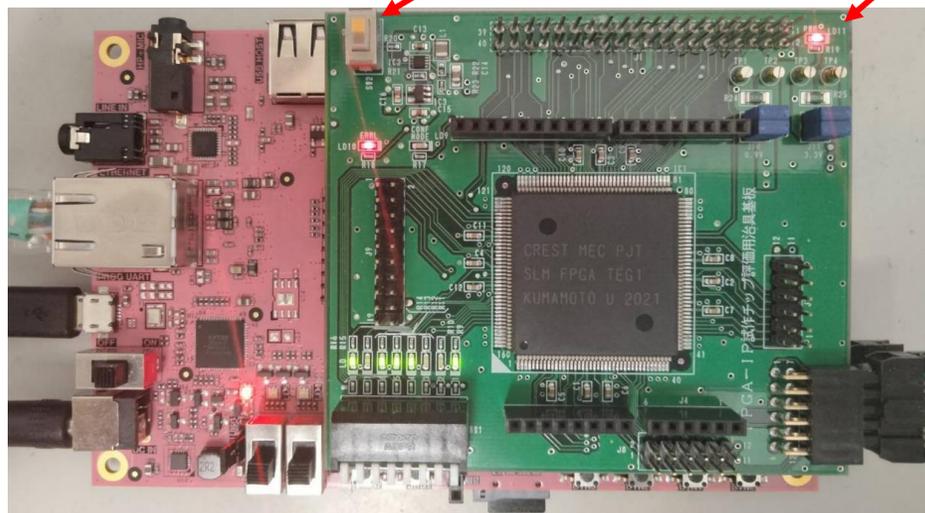


デモの実行方法

- 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)

評価ボード
電源スイッチ(上ON)

評価ボード
電源LED



jupyter counter32 Last Checkpoint: 14分前 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Run

32ビットカウンタの動作確認 (counter32)

Scriptは上から順に実行する

PYNQ-Z2上XC7Z020のPL部コンフィギュレーション

```
In [198]:  
  
# PYNQ-Z2 PLの構成  
from pynq import Overlay  
from pynq.lib import AxiGPIO
```

デモの実行方法

- 「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。

```
In [4]: TEG1_config('counter32.bitstream.frame')

Bitstream file reading ...Done.
Start TEG1 configuration (about 1min.)
End   TEG1 configuration 40.13 s
Pass: conf_mode inactive.
Enter normal mode.

Out[4]: 0 ← “0”で正常終了
```

※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. “Bitstream 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)

```
In [*]:  
TEG1_config('counter32.bitstream.frame')  
Bitstream file reading ...Done.  
Start TEG1 configuration (about 1min.)
```



プログラム実行中

実行中のプログラムは■で止める

```
In [8]:  
TEG1_config('counter32.bitstream.frame')  
Bitstream file reading ...Done.  
Start TEG1 configuration (about 1min.)  
Detect CRC error.  
Out[8]: 1
```

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。

```
GPIO経由でcounter[23:0]を読み表示する。 counter[31:24]はボード上のLEDで確認できる

In [14]:
for i in range(100):
    l_byte = fpga_io_s.read()
    m_byte = fpga_io_e.read()
    h_byte = fpga_io_n.read()
    # 16進数での表示
    print("%02x %02x %02x"%(h_byte,m_byte,l_byte),end=' ')
    # 10進数での表示
    print(" %3d %3d %3d"%(h_byte, m_byte, l_byte), end=' ')
    print(" %10d"%(h_byte*65536 + m_byte*256 + l_byte) )

9f ed 44 159 237 68 10480964
a1 86 51 161 134 81 10585681
a2 65 ab 162 101 171 10642859
a3 45 83 163 69 131 10700163
a4 3b be 164 59 190 10763198
a5 18 20 165 24 32 10819616
```

Counter[23:0]の値

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.

```
GPIO経由でcounter[23:0]を読み表示する. counter[31:24]はボード上のLEDで確認できる
```

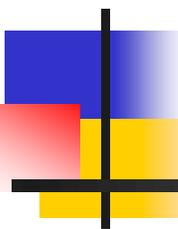
```
In [14]:
```

```
for i in range(100):  
    l_byte = fpga_io_s.read()  
    m_byte = fpga_io_e.read()  
    h_byte = fpga_io_n.read()  
    # 16進数での表示  
    print("%02x %02x %02x"%(h_byte,m_byte,l_byte),end=' ' )  
    # 10進数での表示  
    print(" %3d %3d %3d"%(h_byte, m_byte, l_byte), end=' ' )  
    print(" %10d"%(h_byte*65536 + m_byte*256 + l_byte) )
```

| | | | | | | |
|----|----|----|-----|-----|-----|----------|
| 9f | ed | 44 | 159 | 237 | 68 | 10480964 |
| a1 | 86 | 51 | 161 | 134 | 81 | 10585681 |
| a2 | 65 | ab | 162 | 101 | 171 | 10642859 |
| a3 | 45 | 83 | 163 | 69 | 131 | 10700163 |
| a4 | 3b | be | 164 | 59 | 190 | 10763198 |
| a5 | 18 | 20 | 165 | 24 | 32 | 10819616 |

Counter[23:0]の値

12. 再構成する際は最初から実行する



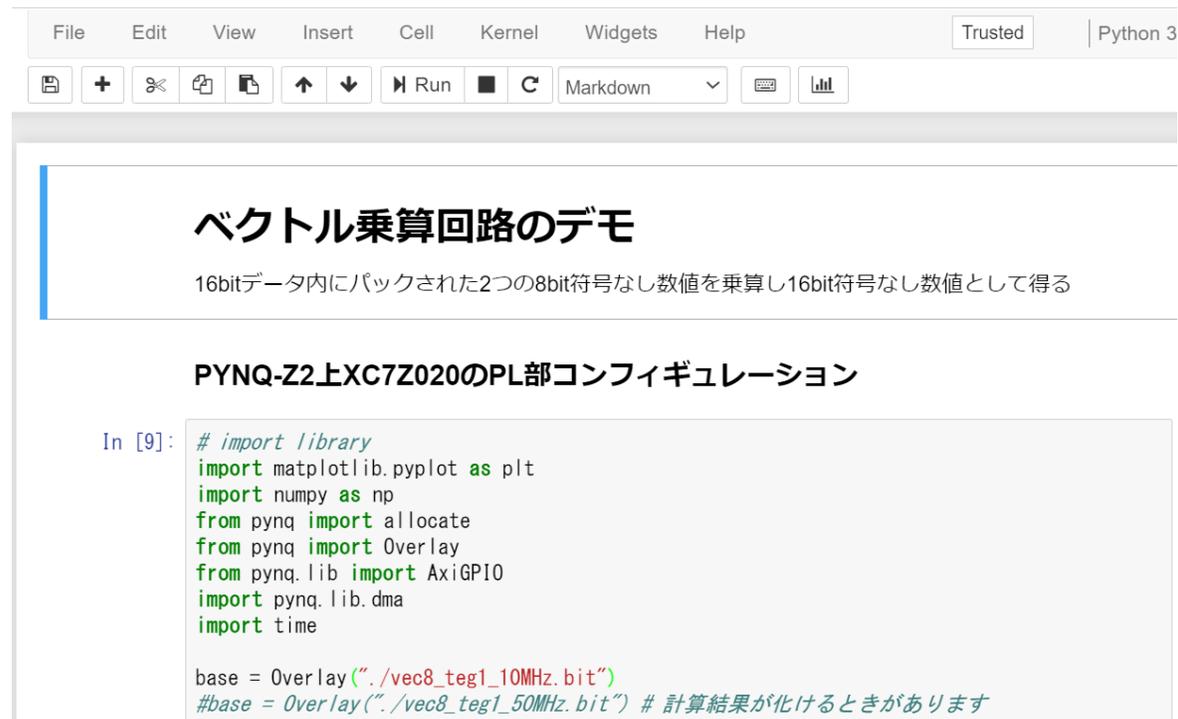
デモンストレーション 手順

ベクトル乗算の場合

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



The screenshot shows a Jupyter Notebook interface. The top menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. Below the menu bar is a toolbar with icons for file operations, navigation, and execution. The main content area displays the title 'ベクトル乗算回路のデモ' (Vector Multiplication Circuit Demo) and a subtitle '16bitデータ内にパックされた2つの8bit符号なし数値を乗算し16bit符号なし数値として得る' (Multiply two 8-bit unsigned integers packed in 16-bit data to get a 16-bit unsigned integer). Below this is a code cell with the following Python code:

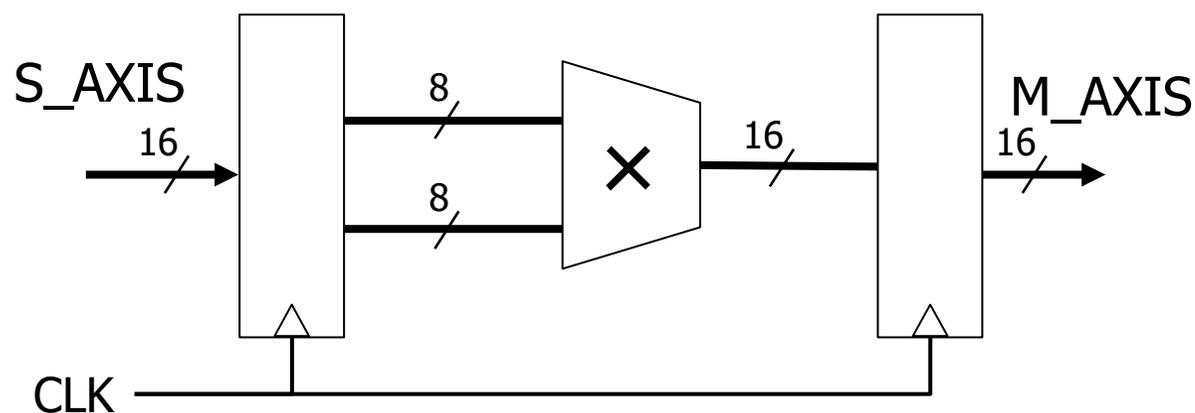
```
In [9]: # import library
import matplotlib.pyplot as plt
import numpy as np
from pynq import allocate
from pynq import Overlay
from pynq.lib import AxiGPIO
import pynq.lib.dma
import time

base = Overlay("./vec8_teg1_10MHz.bit")
#base = Overlay("./vec8_teg1_50MHz.bit") # 計算結果が化けるときがあります
```

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 |

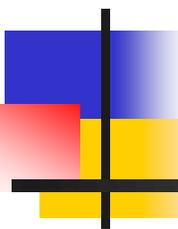
デモの実行方法

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



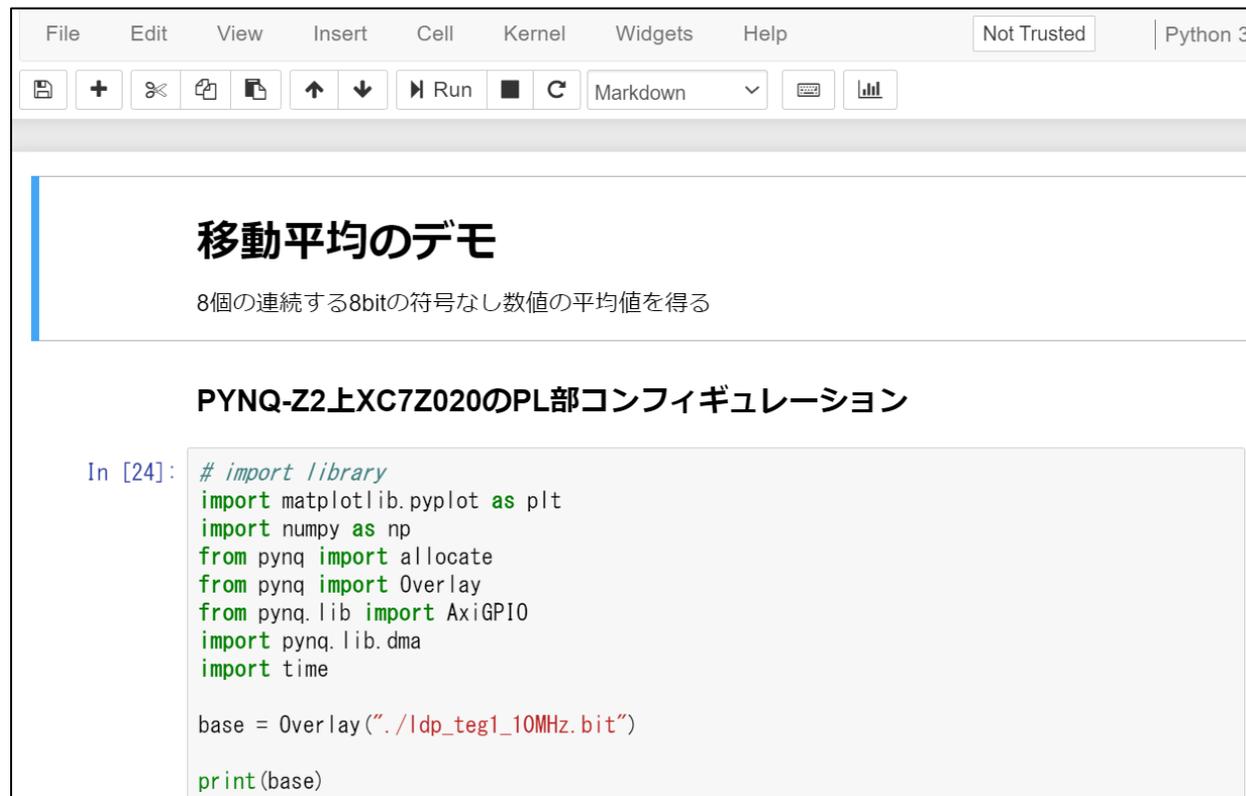
デモンストレーション 手順

ベクトル乗算の場合

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



The screenshot shows a Jupyter Notebook interface with a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations and execution. The notebook content includes a title "移動平均のデモ" (Moving Average Demo) with a subtitle "8個の連続する8bitの符号なし数値の平均値を得る" (Get the average value of 8 consecutive 8-bit unsigned integers). Below this is a section titled "PYNQ-Z2上XC7Z020のPL部コンフィギュレーション" (PL Configuration on PYNQ-Z2 XC7Z020). The code cell shows the following Python code:

```
In [24]: # import library
import matplotlib.pyplot as plt
import numpy as np
from pynq import allocate
from pynq import Overlay
from pynq.lib import AxiGPIO
import pynq.lib.dma
import time

base = Overlay("./ldp_teg1_10MHz.bit")

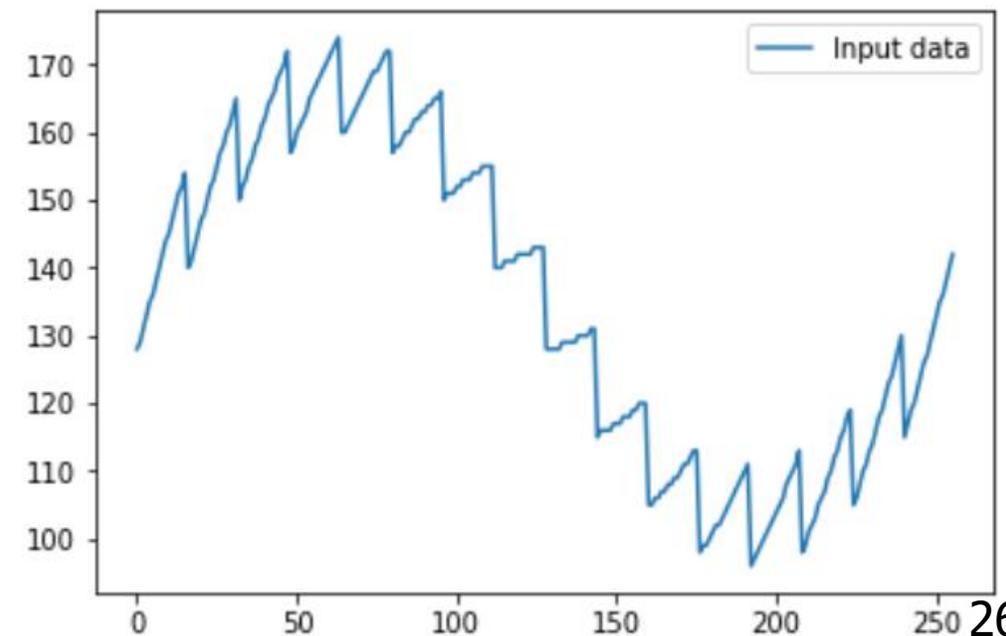
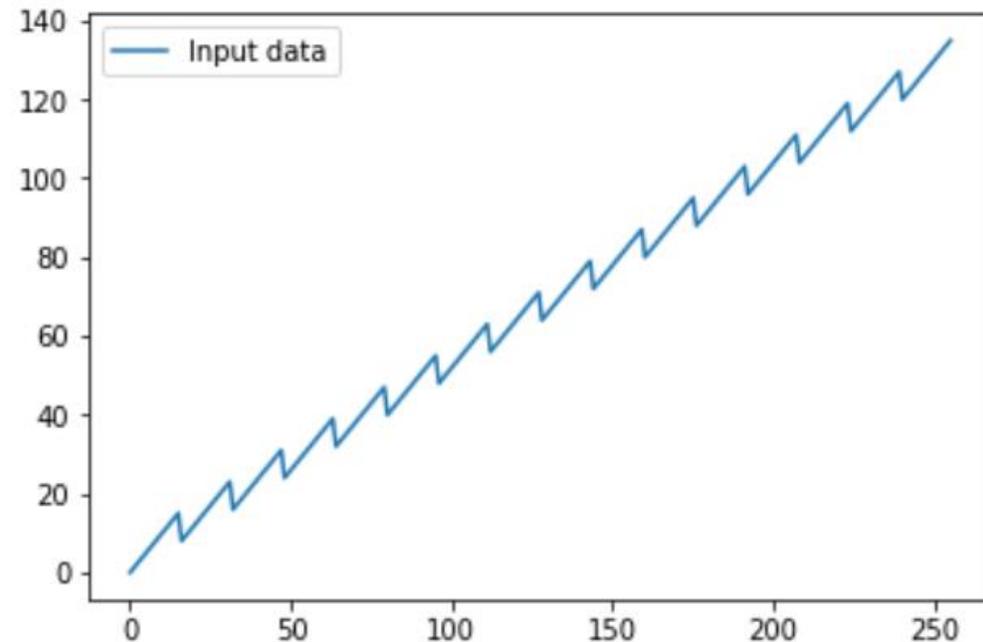
print(base)
```

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

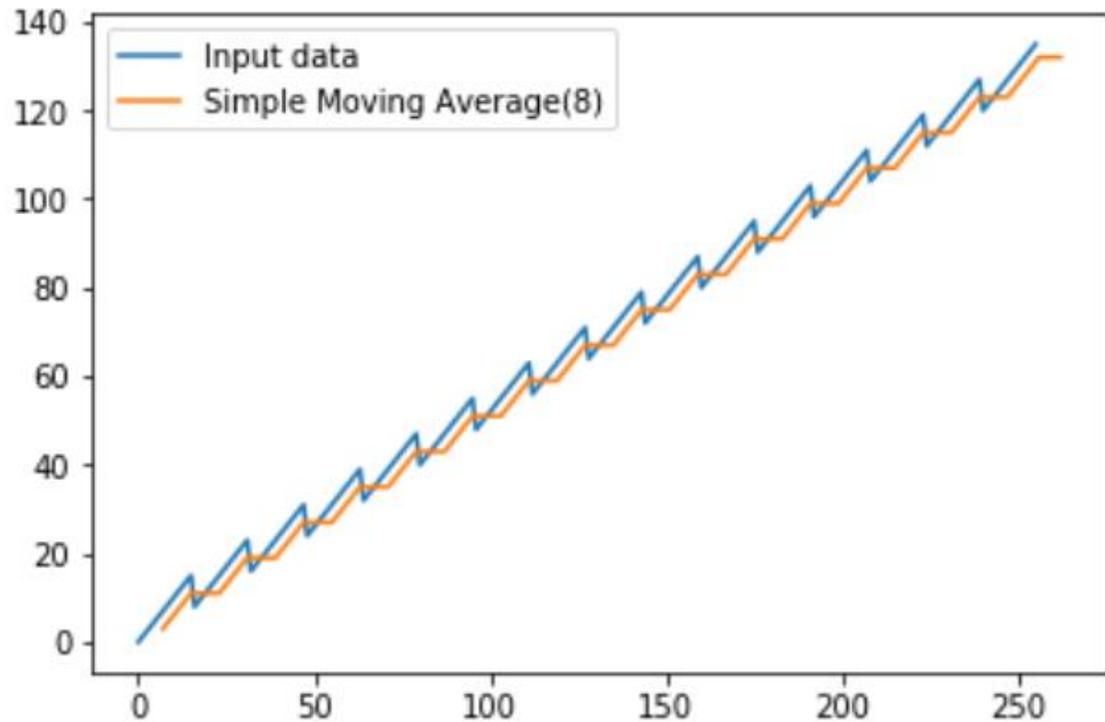
入力データは以下の2種類を準備

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

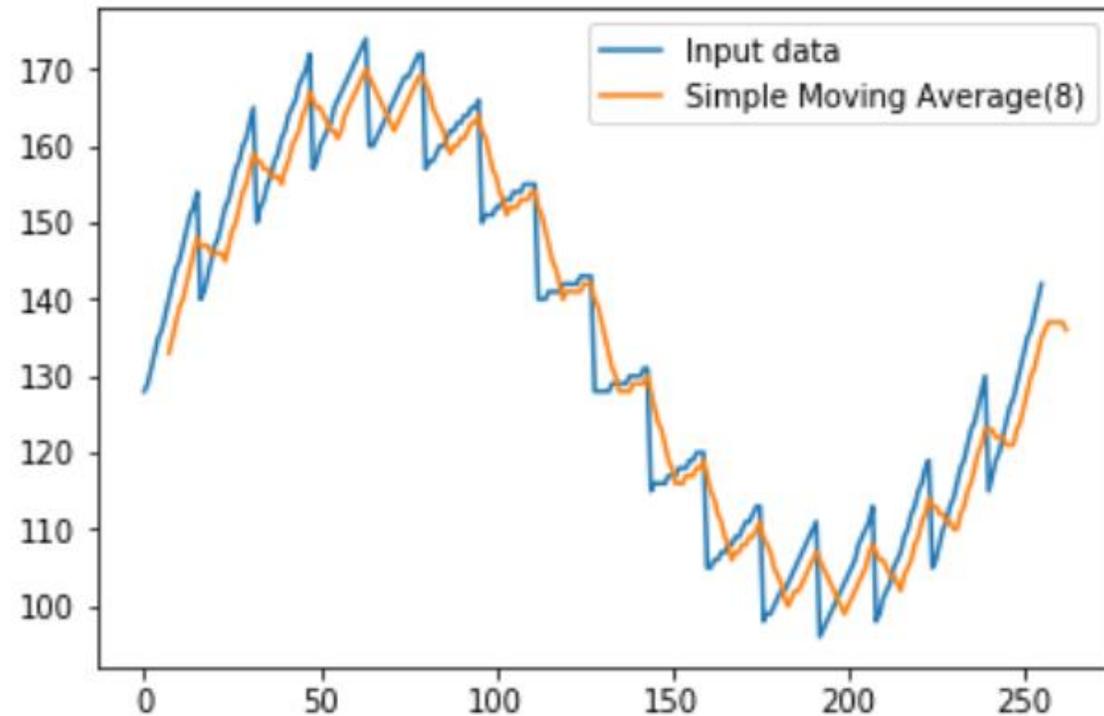


デモの実行方法

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