Assignment 4: CNN Accelerator

Due Date

- **Report & Code Submission – July 15 (Sun), 11:59 PM**
- **Schedule for Design Review & Final Presentation will be announced later.**

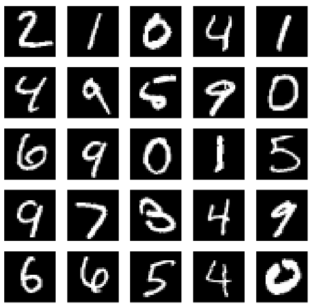
Notice

- Design the image filtering IP (PL), and build a system that includes this IP together with the PS. Test the complete system using the provided Jupyter code.
- **Design a *CNN accelerator IP* module using Verilog or SystemVerilog. Additionally, create a *programming system (block design)* to control the module, and develop the corresponding *control code written in Python*.**
- Please submit the report individually, and only one person per team should submit the design project.
- Compress your Vivado project folder and PS codes (.ipynb) into a .zip file for submission. The compressed folder should include not only the top design but also all project folders for the sub-IP blocks. Additionally, make sure that design source files such as .v and .xdc are also included.

The goal of this project is to design a CNN accelerator that classifies 10,000 images from the MNIST dataset using the provided CNN weight parameters, with a focus on optimizing power consumption, resource utilization, and latency. Additional credit will be awarded for efforts to design optimized hardware that utilizes even lower bit precision than the given parameters. It is recommended to develop an optimal hardware architecture that meets the specified requirements. All computations and data alignment required for CNN acceleration must be performed within the PL (Programmable Logic). The PS (Processing System) may only be used to sequentially write image data to memory accessible by the PL, monitor the status of the PL, and read back the results. For further details, please refer to the main guidelines below.

MNIST Dataset

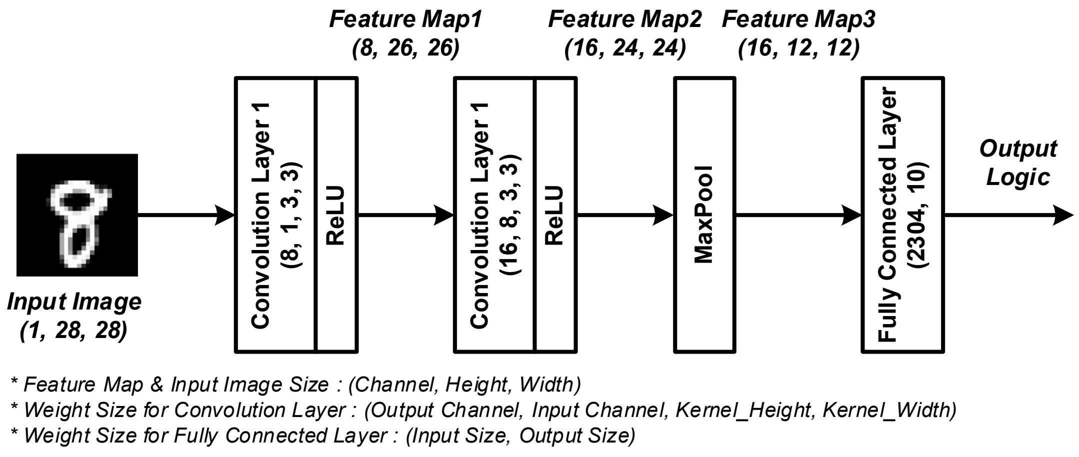
The MNIST dataset is a representative image dataset consisting of handwritten digits (from 0 to 9). It is one of the most widely used benchmarks in machine learning, especially in the fields of image classification and deep learning. Each image is a grayscale image with a size of 28×28 pixels, and each pixel is represented as an 8-bit unsigned integer with a value ranging from 0 to 255. The MNIST dataset provides a simple yet effective environment for experimenting with handwritten digit recognition problems, making it frequently used for comparing the performance of various algorithms or testing new neural network architectures.



< Fig. 1 Example of 2D Convolution >

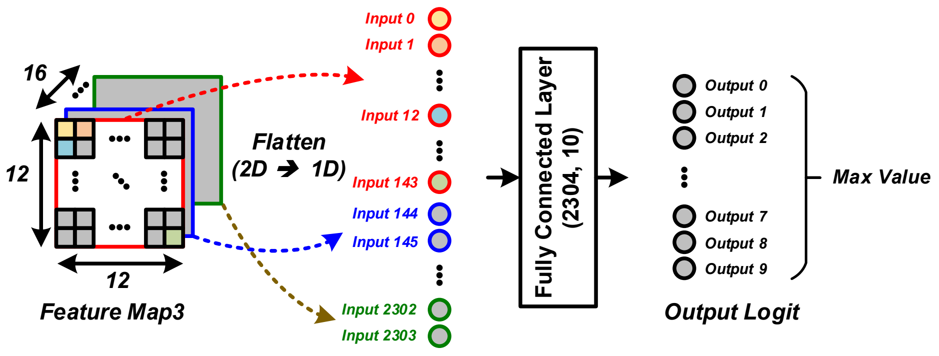
CNN Architecture

Fig.2 shows the CNN targeted for acceleration in this project, which consists of two convolution layers, one MaxPool layer, and one fully connected layer. Each convolution layer uses a 3×3 kernel with a stride of 1, and zero padding is not applied, so the width and height of the output feature map decrease by 2 compared to the input. The first convolution layer has 8 output channels, and the second layer has 16 output channels. A ReLU function is applied after each convolution layer. The MaxPool layer uses a 2×2 filter, resulting in an output of (16, 12, 12) in the order of (channel, height, width).


Feature Map1 (8, 26, 26)
Feature Map2 (16, 24, 24)
Feature Map3 (16, 12, 12)
Input Image (1, 28, 28)
Convolution Layer 1 (8, 1, 3, 3)
ReLU
Convolution Layer 1 (16, 8, 3, 3)
ReLU
MaxPool
Fully Connected Layer (2304, 10)
Output Logic
* Feature Map & Input Image Size : (Channel, Height, Width)
* Weight Size for Convolution Layer : (Output Channel, Input Channel, Kernel_Height, Kernel_Width)
* Weight Size for Fully Connected Layer : (Input Size, Output Size)

< Fig. 2 Target CNN Network Architecture >

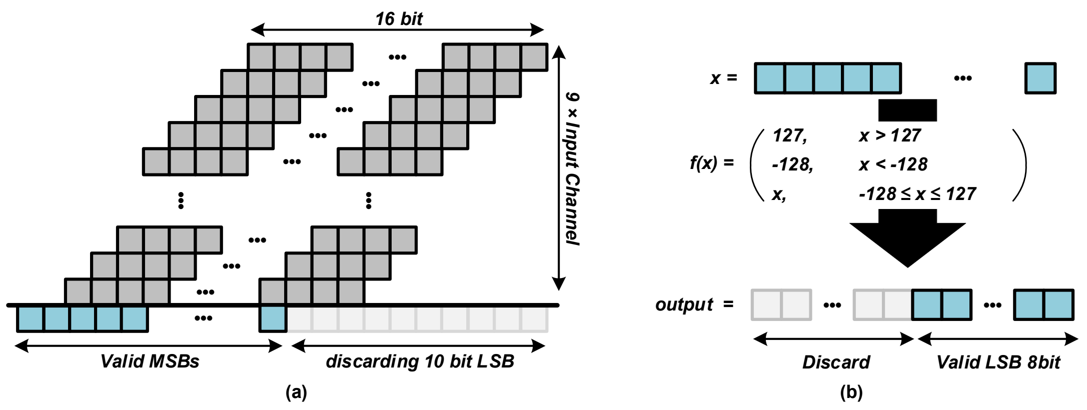
To use the output of the MaxPool layer as the input to the fully connected layer, it must be converted into a 1D vector. In this process, the data is arranged contiguously in the order of width, height, and channel as show in Fig. 3. As a result, the input to the fully connected layer is a 1D vector with 2,304 dimensions. The fully connected layer outputs a 1D vector with 10 dimensions (logit values for each class), and the index of the largest value among these is selected as the final result (predicted digit).



< Fig. 3 Example of Data Alignment for Fully Connected Layer and Output Logit >

Bit Precision

The target network in this project is trained using quantization to signed 8-bit integers. The input image data is preprocessed as signed 8-bit integers, converted from the original unsigned 8-bit integers (all input data is provided as .npy files). During inference, all feature maps must be concatenated to signed 8-bit integers immediately after each convolution or fully connected operation, as illustrated in Fig. 4. During layer operations, bit extension is required to minimize error in repeated addition and multiplication processes. When concatenating the bit-extended data back to signed 8-bit integers, the following method is used: first, data excluding the least significant 10 bits (LSB 10 bits) is concatenated. If the resulting value exceeds +127, it is saturated to 127; if it is less than -128, it is saturated to -128. Finally, only the least significant 8 bits are used as the output.



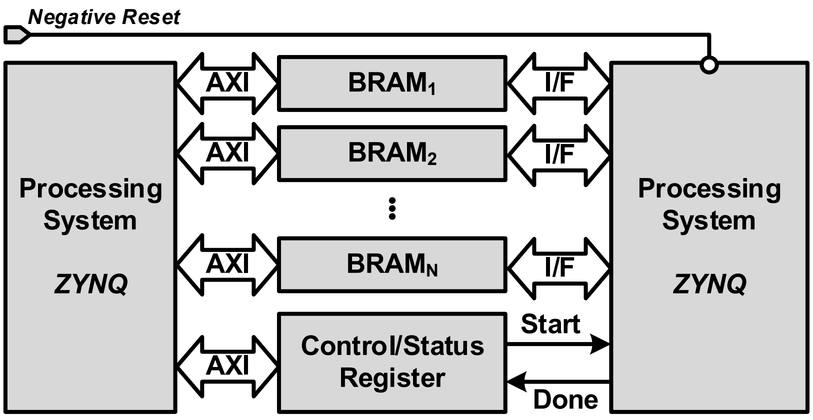
< Fig. 4 Example of Data Alignment for Fully Connected Layer and Output Logit >

When carrying out the assignment, you should first design an accelerator that supports the provided signed 8-bit integer weights and bit precision. Additional credit will be awarded if you also implement lower-bit quantization and design a dedicated architecture. In both the design review and the final presentation, you must specify whether you have implemented lower-bit quantization and design a dedicated architecture, providing details about your implementation. In this case, both the accelerator based on signed 8-bit integers and the additionally designed accelerator must be verified and submitted, and performance improvements must be reported in your presentation.

Design Constraints

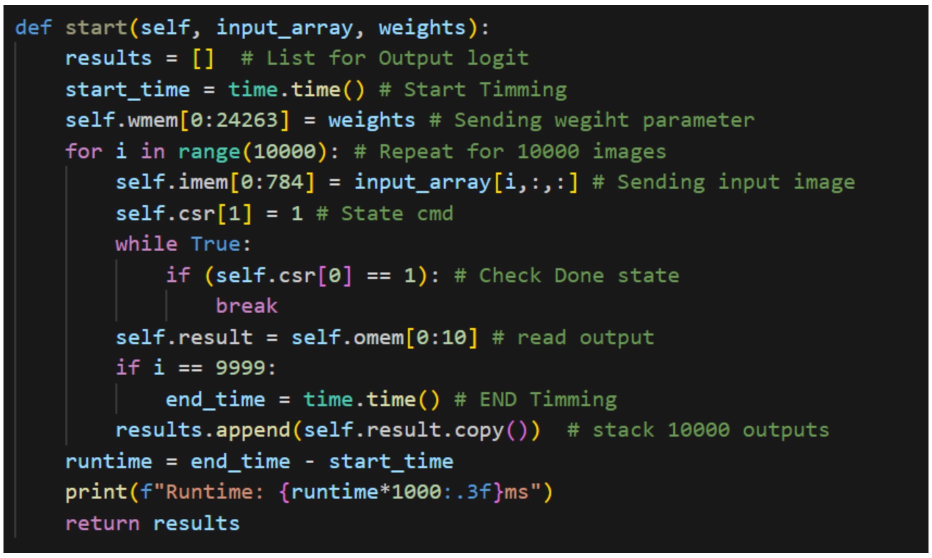
The objective of this assignment is to design a CNN accelerator with optimized hardware architecture. Therefore, performing CNN acceleration operations using the PS (Processing System) is strictly prohibited. The PS must only perform the following tasks: reading from and writing to memory connected to the PL (Programmable Logic) in address order, and providing a start signal to the PL or checking the done signal via control/status registers. Any other operations, such as rearranging data for convolution or fully connected computations, or performing specific operations like MaxPool or ReLU on the PS side, will not be given any points. Additionally, it is prohibited to convert image or weight data into .coe format and initialize BRAM with it. All initial image and weight data must be transferred from the PS.

As shown in Fig. 5, the overall system configuration is similar to Assignment 3, but the internal BRAM structure can be freely designed. You may also define and use the Control/Status Registers according to your own design. However, you must write the .ipynb code to control these registers yourself, using the week 9 course materials as a reference. For debugging and testing convenience, please configure the reset signal for the PL separately using an external switch in this assignment as well.



< Fig. 5 System block diagram for the target design >

The latency of the accelerator is defined as the time from the moment data is first written to the BRAM to the moment when all outputs processed by the PL are read back by the PS. For example, suppose computations are performed sequentially for 10,000 images as shown in Fig. 6. The PS should start timing just before the weight parameters are sent. After all weight parameters have been transferred, the PS repeatedly performs the following steps in a loop: sending the input image, issuing the start command, checking the done state, and reading the output. Timing should stop when the output read for the last image is completed. The figure below provides an example of measuring latency using the time library; however, you are not required to control the PS exactly as shown in the example.

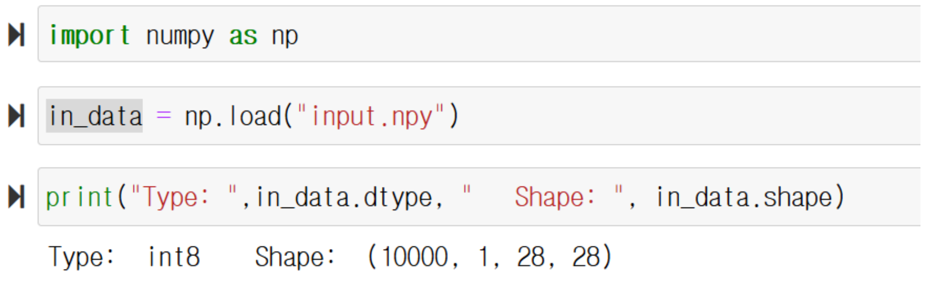


```python
def start(self, input_array, weights):
    results = []  # List for Output logit
    start_time = time.time() # Start Timming
    self.wmem[0:24263] = weights # Sending wegiht parameter
    for i in range(10000): # Repeat for 10000 images
        self.imem[0:784] = input_array[i,:,:] # Sending input image
        self.csr[1] = 1 # State cmd
        while True:
            if (self.csr[0] == 1): # Check Done state
                break
        self.result = self.omem[0:10] # read output
        if i == 9999:
            end_time = time.time() # END Timming
        results.append(self.result.copy())  # stack 10000 outputs
    runtime = end_time - start_time
    print(f"Runtime: {runtime*1000:.3f}ms")
    return results
```

< Fig. 6 Examples of using timing library and PS control code >

Skeleton Code

The skeleton code includes five .npy files. These files contain: the weights for the three layers that make up the CNN, 10,000 MNIST images, and the corresponding CNN output logits for those images. After uploading these files to the PYNQ board's Jupyter Lab, you can read them as NumPy arrays as shown in Fig 7. The format of this data is similar to the format you obtained in assignment 3 when reading images with cv2.imread. The input data has the shape (10000, 1, 28, 28), which corresponds to (number of images, channels, height, width). The other .npy files are also stored in the order illustrated in Fig. 2.



```python
import numpy as np
```

```python
in_data = np.load("input.npy")
```

```python
print("Type: ",in_data.dtype, "   Shape: ", in_data.shape)
```

```
Type:  int8    Shape:  (10000, 1, 28, 28)
```

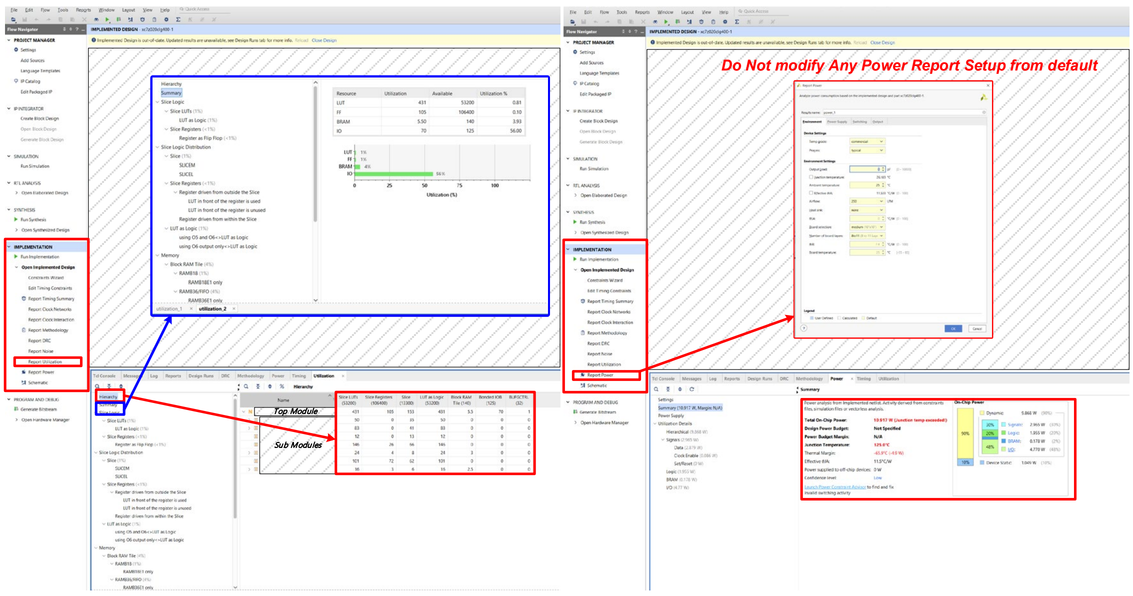
< Fig. 7 Examples of loading numpy array in Jupyter Lab >

Tips

After implementing your system in the VIVADO tool, you will receive reports on both the utilization of FPGA resources and the estimated power consumption of your design when implemented on the board. Refer to Fig. 8 for details. You are encouraged to check the utilization and power reports for your designed system and make efforts to optimize them.

The priorities for this project are as follows:

1. ***Maximize the utilization*** of the available FPGA resources to achieve the ***shortest possible latency***.
2. Design for ***minimal power consumption as long as it does not result in reduced throughput*** (or increased latency).
3. ***Every design must be accompanied by a logical explanation of your design choices***, and you should be able to justify them during the design review and final presentation.
4. The final grade for the project will be determined by a comprehensive evaluation, not just by simple quantitative metrics.



Resource	Utilization	Available	Utilization %
LUT	431	53200	0.81
FF	105	106400	0.10
BRAM	5.50	140	3.93
IO	70	125	56.00

Name	Slice LUTs (53200)	Slice Registers (106400)	Slice (13300)	LUT as Logic (53200)	Block RAM Tile (140)	Bonded IOB (125)	BUFGCTRL (32)
Top Module	431	105	153	431	5.5	70	1
Sub Modules	50	0	35	50	0	0	0
	83	0	41	83	0	0	0
	12	0	13	12	0	0	0
	146	26	66	146	0	0	0
	24	4	8	24	3	0	0
	101	72	62	101	0	0	0
	16	3	6	16	2.5	0	0

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	10.917 W (Junction temp exceeded!)
Design Power Budget:	Not Specified
Power Budget Margin:	N/A
Junction Temperature:	125.0°C
Thermal Margin:	-65.9°C (-4.9 W)
Effective θJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	9.868 W	(90%)
Signals:	2.965 W	(30%)
Logic:	1.955 W	(20%)
BRAM:	0.178 W	(2%)
I/O:	4.770 W	(48%)
Device Static:	1.049 W	(10%)

< Fig. 8 Examples of implementation report in VIVADO >

First, it is strongly recommended that you begin by implementing a module that can perform CNN computation for a single image. This module will serve as your baseline. You should then try applying various techniques covered in class to improve performance beyond this baseline. Be sure to compare the results and present these comparative metrics during your design review and final presentation.