# The Design Document for CSEE4840 Embedded System Design
# CNN Accelerator for Gesture Recognition

Yangfan Wang (yw4415)
Fengze Zhong (fz2393)
Xincheng Yu (xy2654)

Spring 2025

## Contents

# 1 Introduction

This project aims to implement an accelerator system that performs gesture recognition. The FPGA platform offers substantial computing power with the versatility to communicate with the software. Also, as the CNN relies on the fixed function unit to perform certain tasks, it is suitable to deploy it on FPGA with only accelerating the computation part.

As indicated in Figure 1, the dataset we choose contains 10 classifications of gestures, and this ensures that we can accomplish the recognition with a lightweight neural network.


Figure 1: Recognition Classes [1]

# 2 System Overview

The system we build contains the software control and the hardware acceleration. The software side will be running on the HPS Linux mainly for file operations and CNN flow control. It defines the CNN layers and all its parameters. The hardware side will be purely for computation including the kernel convolution, max pooling and ReLU activation.

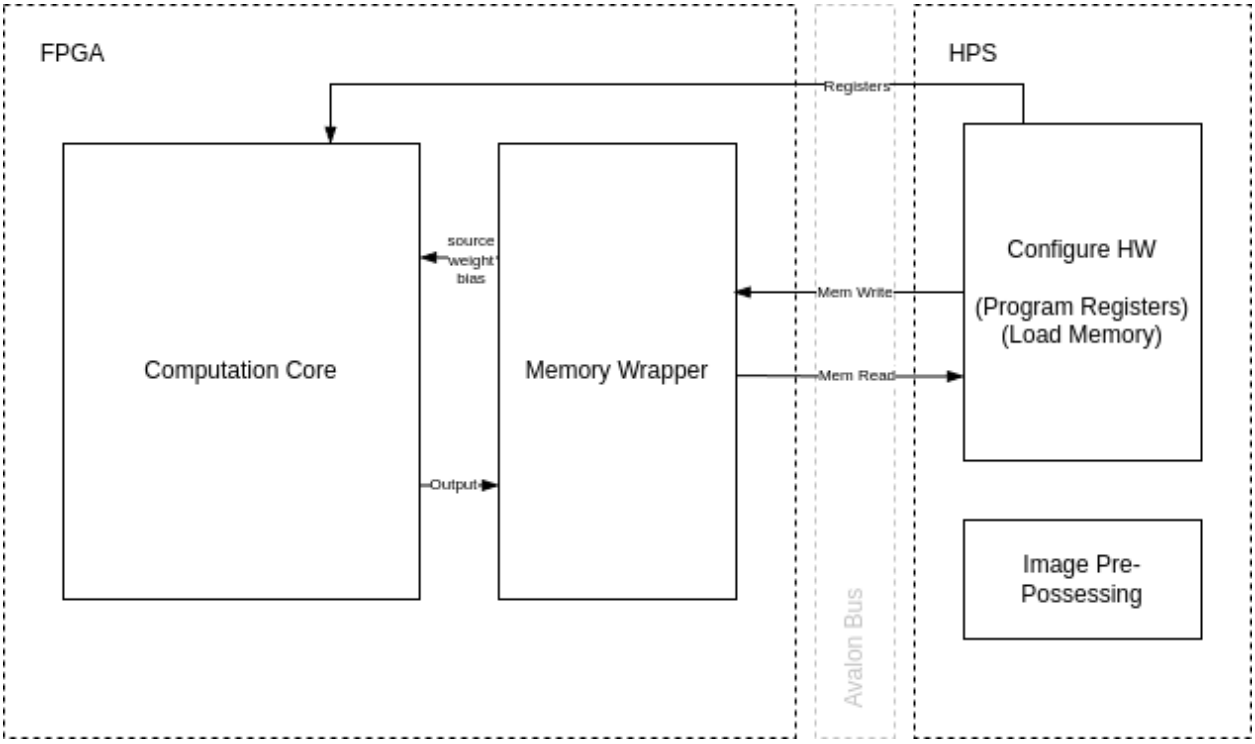


Figure 2: Block Diagram

## 2.1 Software

With an image pending for recognition, the software will firstly downsample and resize it to 32x32. Then it will follow the structure of the CNN trained and read weights and biases from files and load to the hardware according to the current layer.

## 2.2 Hardware

On the hardware, the control FSM will loop through all the source and destination channels for this layer based on the configuration registers. It will first fetch the data of a 3x3 kernel size, weights, and bias from the memory. Then it will dispatch the computation to the MAC unit. After each kernel computation completed, the temporary output will write to the output memory. The registers act as the sliding window, therefore after the initial 9 reads at the beginning of each row, the subsequent reads will all be 3. After it loops through all the sources channels for all destination channels, it will assert a signal to indicate completion. The FSM will also deal the extra cycle delay from the memory access.

### 2.2.1 MAC

MAC units will have 9 parallel Q8.8 fixed-point multiplier. We will expand the number MAC units in parallel to expedite the computation, given 112 DSP blocks on the Cyclone V SE FPGA.

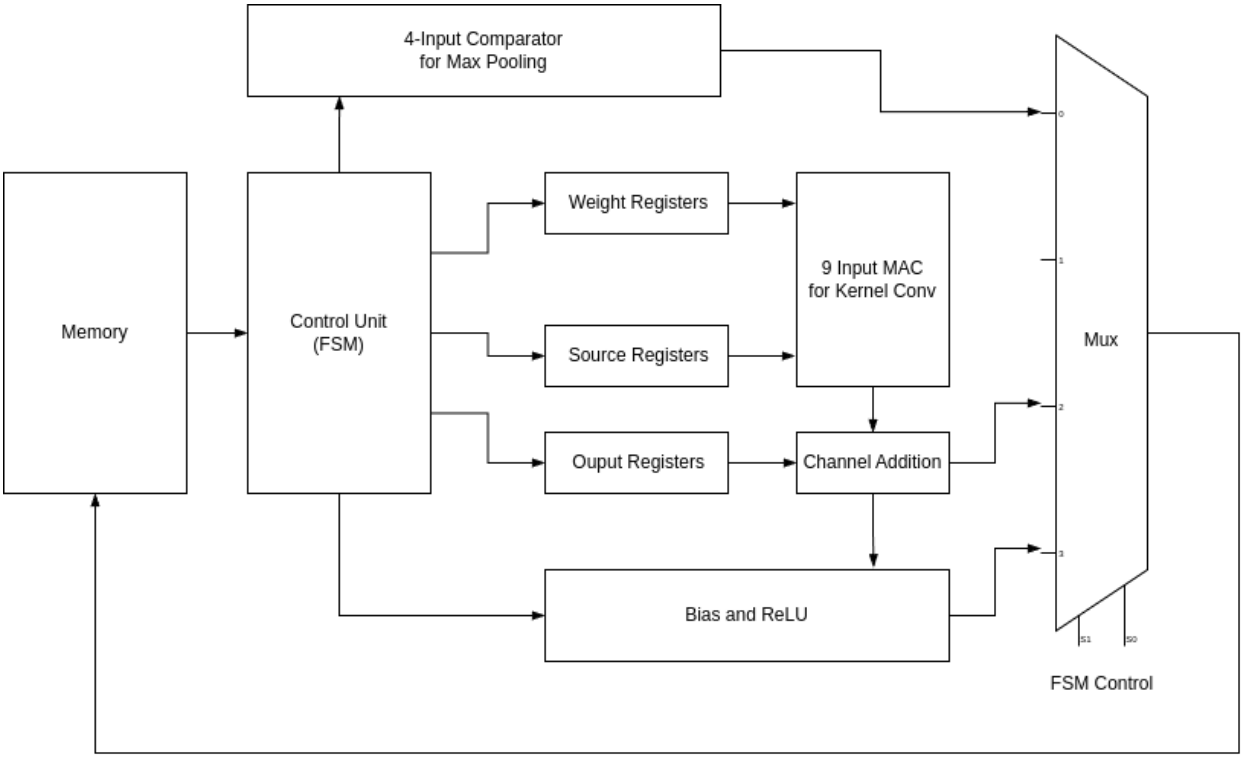


Figure 3: MAC Unit

# 3 CNN Structure

Due to the limited RAM on the FPGA, we chose to use a relatively lightweight network consisting of four convolutional layers and two fully connected layers, as shown in Figure 4.

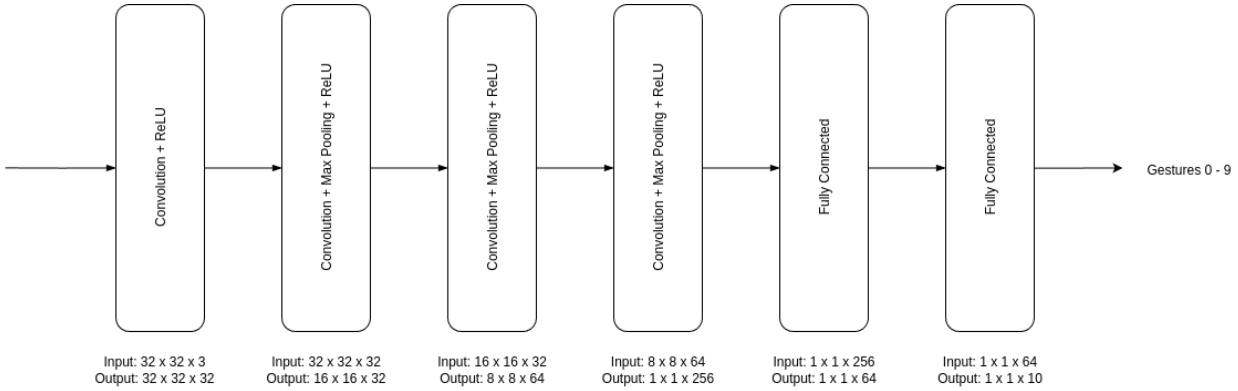


Figure 4: Convolution Neural Network Architecture

We used PyTorch to train our model on the Sign Language Digits Dataset, which contains 2,180 color images of hand gestures representing digits from 0 to 9. Since the number of samples per class is relatively small, we applied data augmentation techniques including random rotation, center cropping, and adjustments to brightness and contrast, in order to improve the model's generalization performance. We then split the dataset into 80% for training and 20% for validation. The current model achieved an accuracy of 93% on

the original dataset and is capable of distinguishing hand gesture images that resemble those in the training data with white background color.

# 4 Resource Budgets

The main constraint for this project is the BRAM needed to store the source data, weight, bias and output. All the data involved will be represented in Q8.8 fixed point using 2 bytes. The following is the estimated memory usage for each layer, and given the 4450 Kbits FPGA memory, we should be able to fit each layer in.

| | |
|---|---|
| Layer 1 | 72 KB |
| Layer 2 | 146 KB |
| Layer 3 | 68 KB |
| Layer 4 | 305 KB |
| Layer 5 | 33 KB |
| Layer 6 | 1.4 KB |

Table 1: Memory Allocation

Regarding to the LUTs and DSPs, as mentioned above, the FSM will simply loop through the source and destination channel therefore the logic will not consume much LUTs to implement. The MAC and comparisons part will rely on the DPS units, which we have sufficient on the FPGA.

# 5 The Hardware/Software Interface

The software module will be responsible for setting up the memory and configuring the accelerator. The programmable registers are defined as in Table 2. The write addresses range from 0x00 to 0x18. Writing to the start field triggers the layer computation.

| | | |
|---|---|---|
| src_chans | number of source channels | uint16 |
| dst_chans | number of destination channels | uint16 |
| num_cols | number of columns in the source | uint16 |
| num_rows | number of rows in the source | uint16 |
| do_pool | enable pooling for the layer | bool |
| pool_size | max pooling size | uint8 |
| pool_stride | max pooling stride | uint8 |
| data_starting_address | starting address of the source data | uint16 |
| weight_starting_address | starting address of the weight | uint16 |
| bias_starting_address | starting address of the bias | uint16 |
| output_starting_address | starting address of the output | uint16 |
| start | start layer computation | bool |

Table 2: Variable Configuration

After the hardware completes the computation, it will assert an interrupt signal to the software to indicate the readiness of the output value.

The software is also responsible for writing sources, weights, biases data and reading output from the local BRAM in the FPGA. The memory controller from the hardware side handles the data transactions. Considering the scale of the neural network, it is not feasible to load the weights for all the layers once to the BRAM. Therefore, after each layer is completed, the software will read from the memory and rewrite the memory structure.

Address map is defined as following:

| | |
|---|---|
| data_staring_address | data |
| weight_starting_address | weight |
| bias_starting_address | bias |
| output_starting_address | output |

# 6 Milestones

1. Complete the refinement of the neural network.
2. Implement the hardware component (MAC, Registers) in SystemVerilog.
3. Synthesize and resolve any timing and utilization problems.
4. Implement the C driver for dispatch layer tasks

# References

[1] Arda Mavi. Sign language digits dataset. `https://github.com/ardamavi/Sign-Language-Digits-Dataset`, 2017. Accessed: 2025-04-18.