CNN Accelerator for Gesture Recognition on DE1-SoC FPGA

CSEE 4840 Embedded System

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

May 14, 2025

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Project Introduction

Target: Use FPGA to accelerate the inference process of a CNN model

Figure 1: 10 Gestures for recognition

CNN Architecture

The CNN contains 4 convolutional layers, 3 max pooling layers and 2 fully connected layers

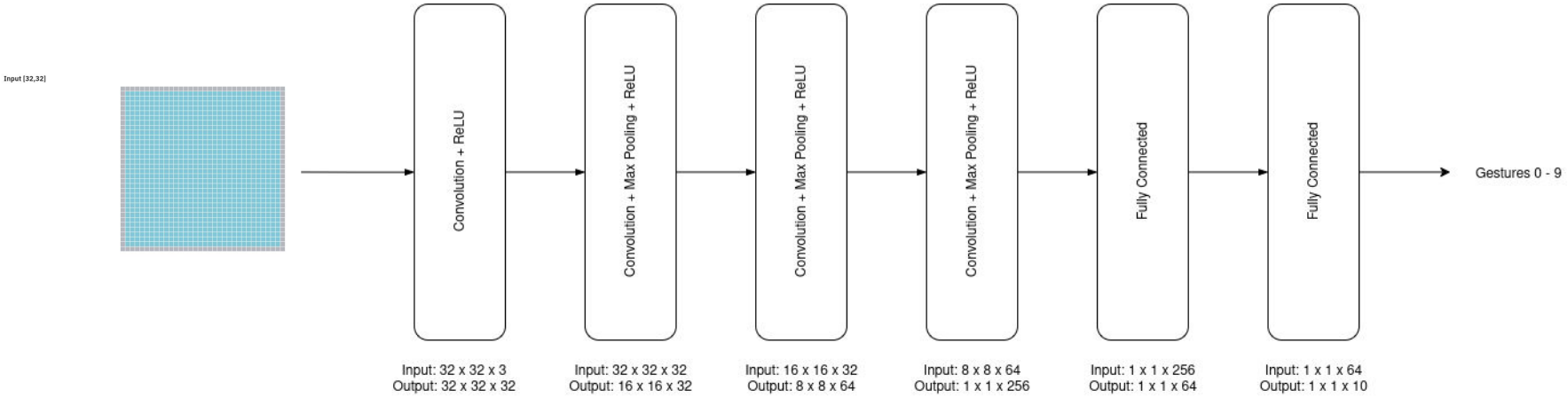


Figure 2: CNN Architecture

CNN Architecture

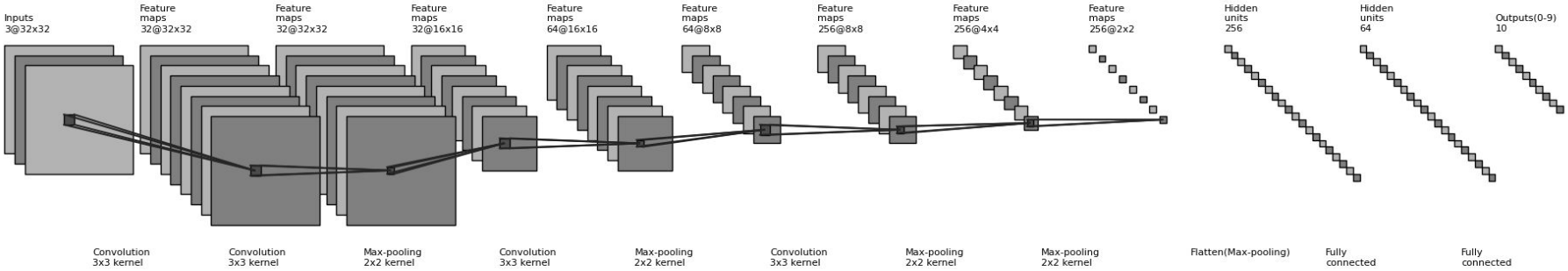


Figure 3: CNN Architecture (detailed)

Golden Model

Dataset:

Sign Language Digits Dataset, which contains 2,180 color images of hand gestures representing digits from 0 to 9.

Platform:

PyTorch

Data Augmentation Techniques:

Random rotation, Center cropping, and

Adjustments to brightness and contrast

Data Split:

80% for training and 20% for validation.

Accuracy:

93% on the original dataset

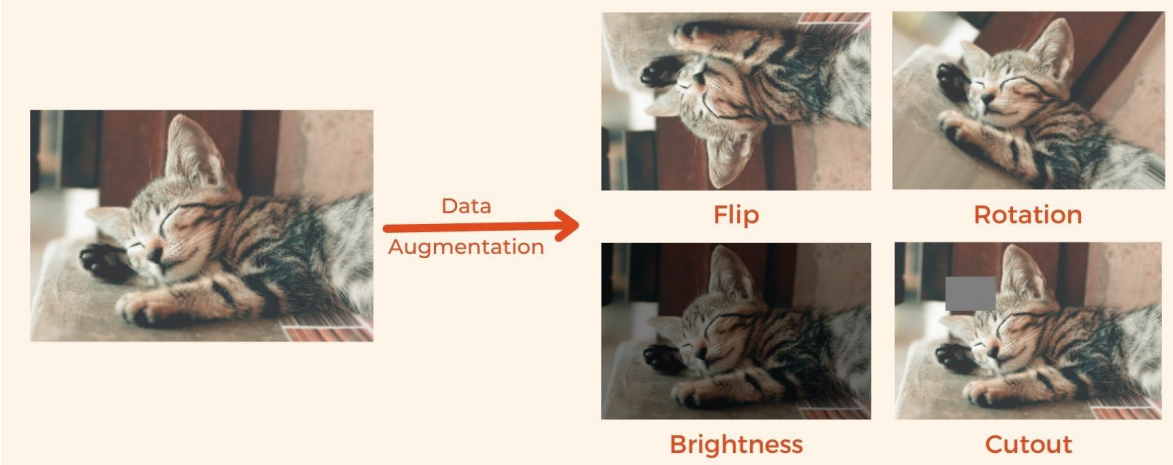
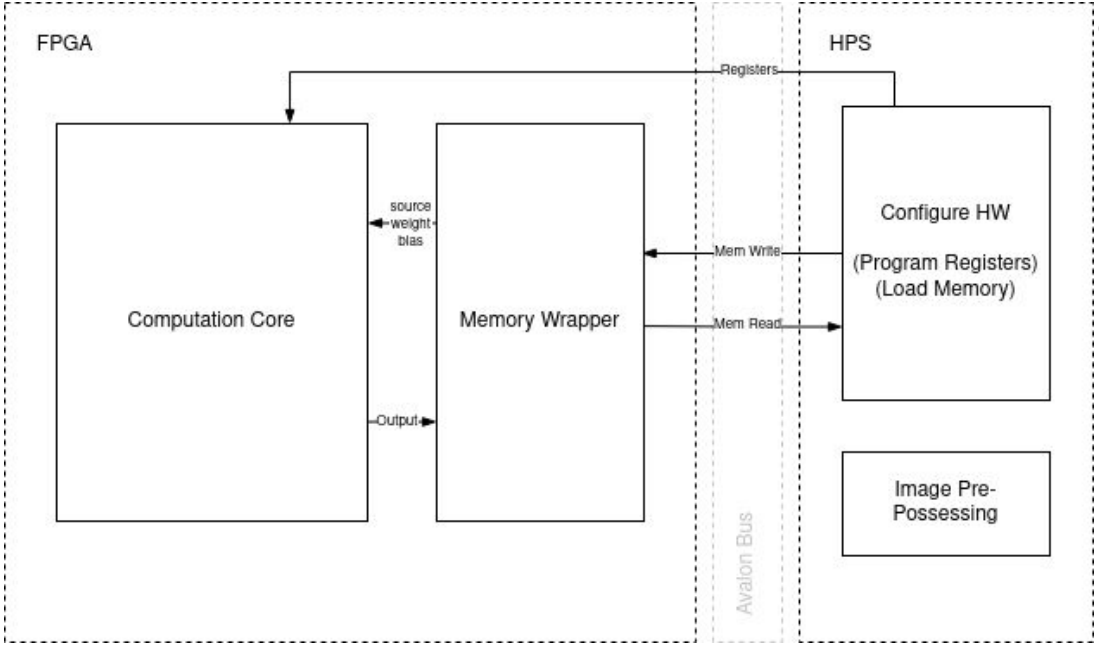


Figure 4: Data Augmentation Examples

System Architecture

The system we build contains the software control and the hardware acceleration.

- The hardware side will be purely for storage and computation including the kernel convolution, max pooling, ReLU activation and fully connected layer.



- The software side will be running on the HPS Linux mainly for input process and CNN flow control.

Figure 5: FPGA and HPS Block Diagram

Software

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

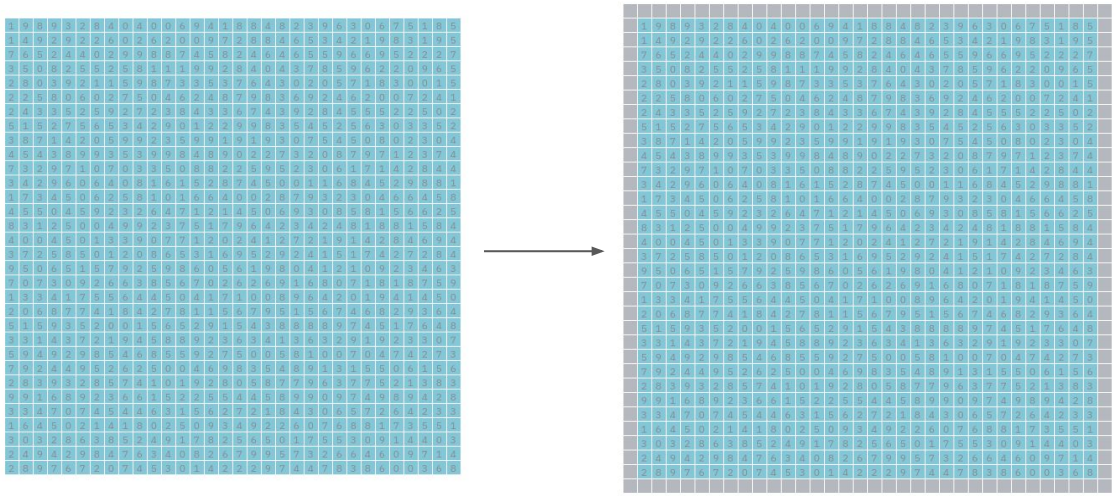
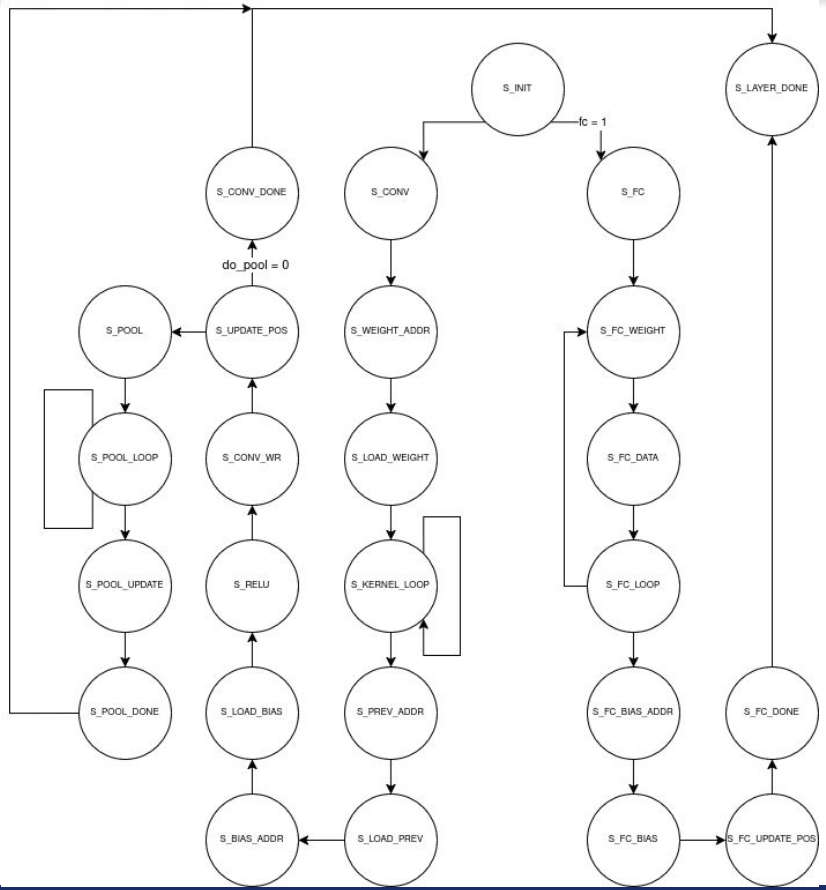


Figure 6: Example of 32x32 matrix zero padding at the input

Software

- `compute_network()`:
  - Main function that runs the network
- `run_conv()`:
  - Set convolution layer parameters
  - Allocate memory for output
- `run_fc()`:
  - Set fully connected layer parameters
  - Allocate memory for output
- `memory_ioctl()`
  - `CONV_WRITE & FC_WRITE`:
    - `copy_from_user`: configuration
    - `copy_from_user`: intput_data, weight, bias
    - `copy_to_user`: output_data

Hardware Top Level



Convolution

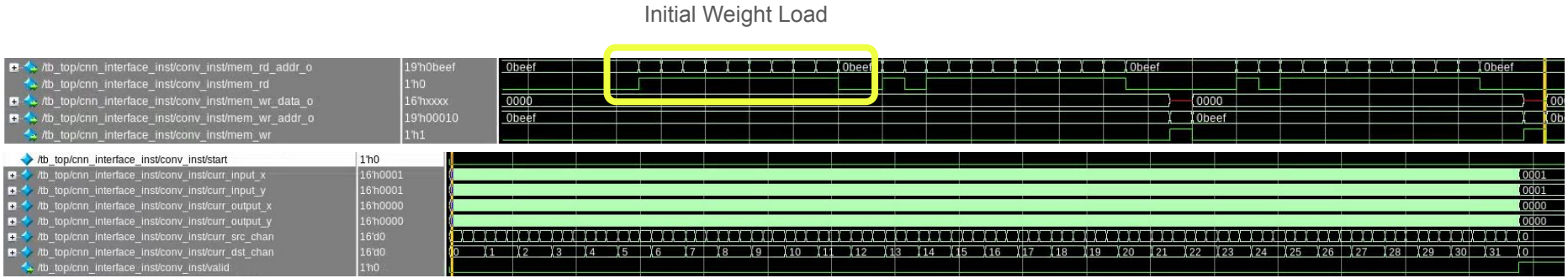
Max Pooling

Fully Connected

Figure 7: State machine Diagram

Hardware Top Level

Convolution



conv_weight_rd_addr:

9 * curr_src_chan + 9 * num_src_chans * curr_dst_chan + curr_kernel_id

conv_data_rd_addr:

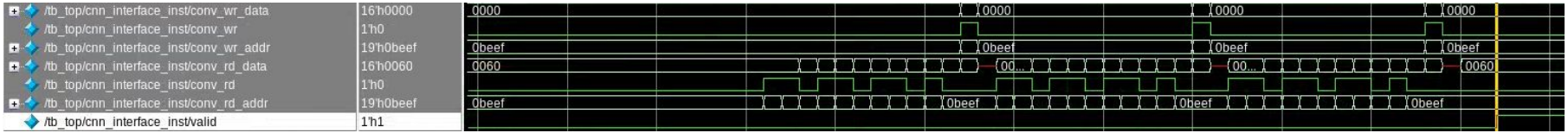
curr_pixel_pos + num_cols * num_rows * curr_src_chan

conv_output_wr_addr:

curr_output_y * (num_cols - 'd2) + curr_output_x + curr_dst_chan * (num_cols - 'd2) * (num_rows - 'd2)

Hardware Top Level

Fully Connected



fc_weight_rd_addr:
curr_fc_w_row * num_dst_chans + curr_fc_w_col

fc_data_rd_addr:
curr_fc_w_row

fc_wr_addr:
curr_fc_w_col

Hardware Component

Address Computation

Pixel Position:

curr_kernel_pos - num_cols - ‘d1
curr_kernel_pos - num_cols
curr_kernel_pos - num_cols + ‘d1
curr_kernel_pos - ‘d1
curr_kernel_pos
curr_kernel_pos + ‘d1
curr_kernel_pos + num_cols - ‘d1
curr_kernel_pos + num_cols
curr_kernel_pos + num_cols + ‘d1;

Hardware Component

Address Computation

pool_rd_addr:
(curr_pool_size * pool_size) + curr_pool_stride_y) * (num_cols - ‘d2) + (curr_pool_x * pool_size + curr_pool_stride_x + curr_dst_chan * (num_cols - ‘d2) * (num_rows - ‘d2)

pool_wr_addr:
(curr_pool_y) * (num_cols - ‘d2) / pool_size + (curr_pool_x)

Hardware Component

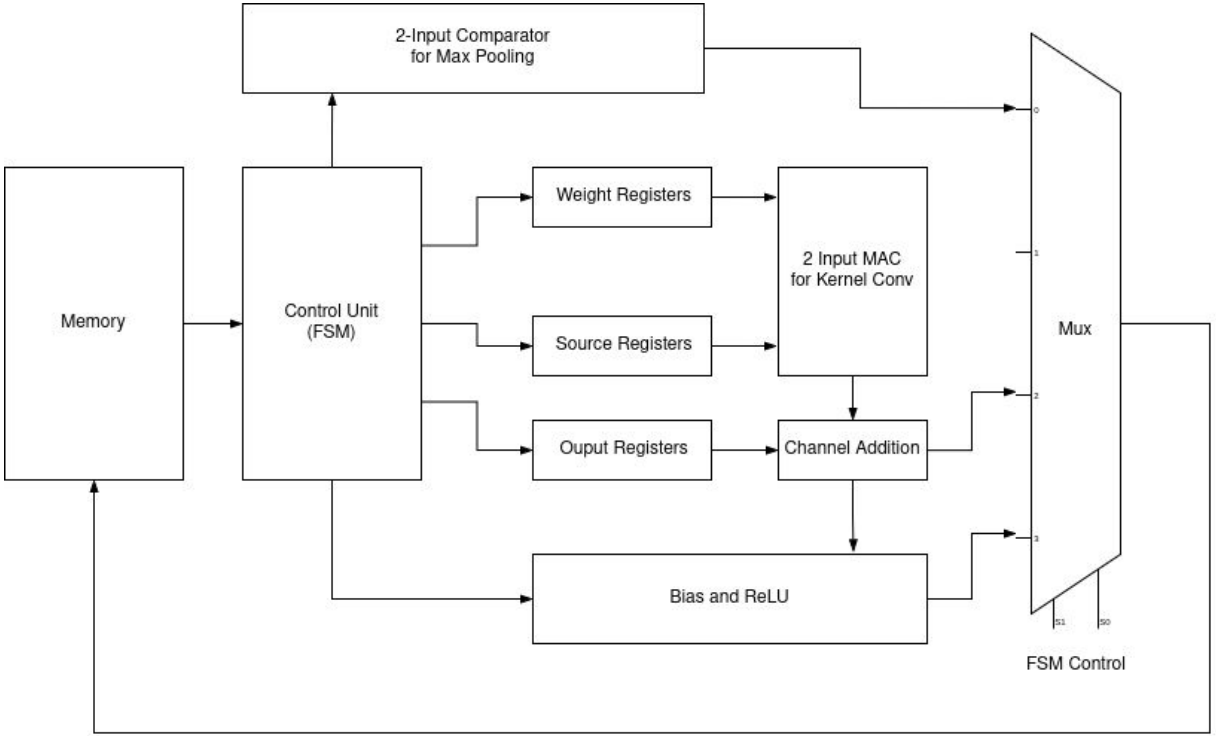


Figure 8: MAC Unit

HW/SW Interface

Programmable Registers

0	num_src_chans
1	num_dst_chans
2	num_cols
3	num_rows
4	fc
5	do_pool
6	pool_size
7	pool_stride
8	data_starting_addr[15:0]
9	data_starting_addr[31:16]
10	weight_starting_addr[15:0]
11	weight_starting_addr[31:16]
12	bias_starting_addr[15:0]
13	bias_starting_addr[31:16]
14	output_starting_addr[15:0]
15	output_starting_addr[31:16]
16	start

Read to any address will receive the LAYER_DONE signal from HW

This is how software tells hardware what to do and receive feedback from hardware

HW/SW Interface

Memory Space

Data
Weight
Bias
Output

Software does the zero padding for each layer

Verification

- Implemented Python Codes
  - Fully Connected Layer
  - Convolution
  - Pooling

  Generating Tests Case and Expected Output

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.

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

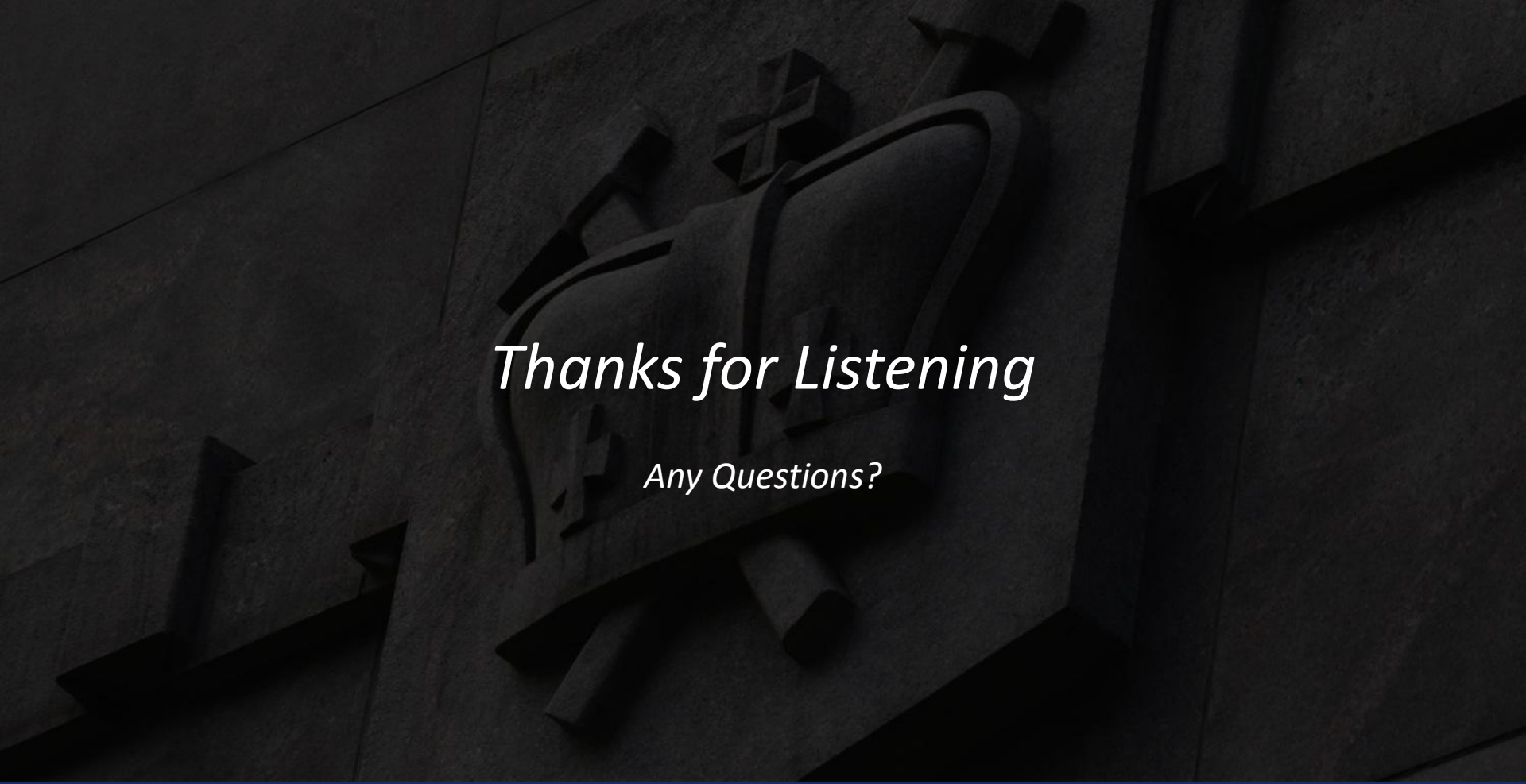
Table 1. memory requirement of each layer

- BRAM size must be larger than the maximum layer memory requirement
- BRAM is set to 340 KB

Resource Budgets

Flow Summary	
Flow Status	Successful - Wed May 14 14:40:42 2025
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	soc_system
Top-level Entity Name	soc_system_top
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	1,498 / 32,070 (5 %)
Total registers	1301
Total pins	362 / 457 (79 %)
Total virtual pins	0
Total block memory bits	2,785,424 / 4,065,280 (69 %)
Total DSP Blocks	21 / 87 (24 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	1 / 4 (25 %)

Fmax ~ 45MHz



Thanks for Listening

Any Questions?