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 CNN Architecture

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 2.

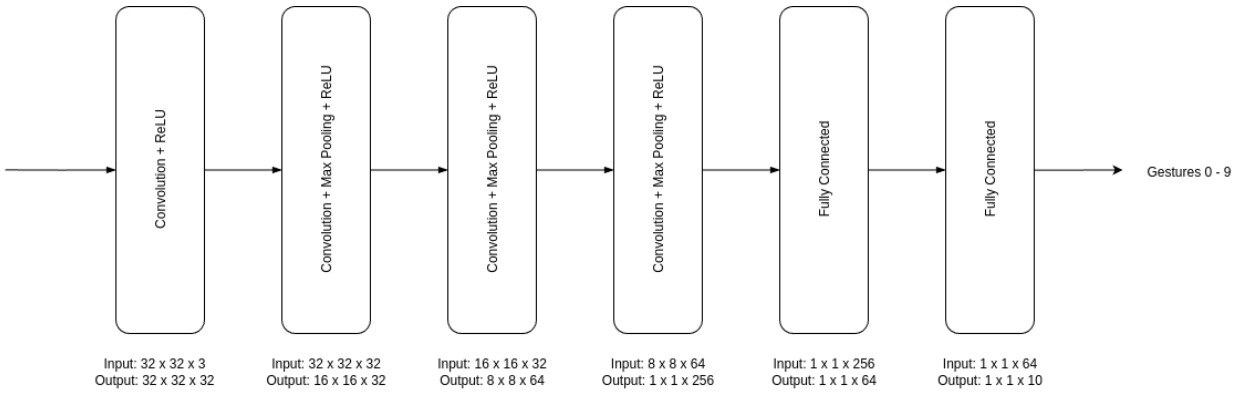

Figure 2: Convolution Neural Network Architecture

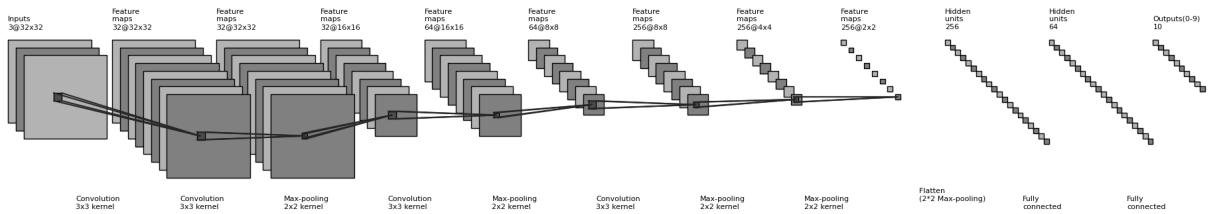

Figure 3: Detailed Convolution Neural Network Architecture

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.

3 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.

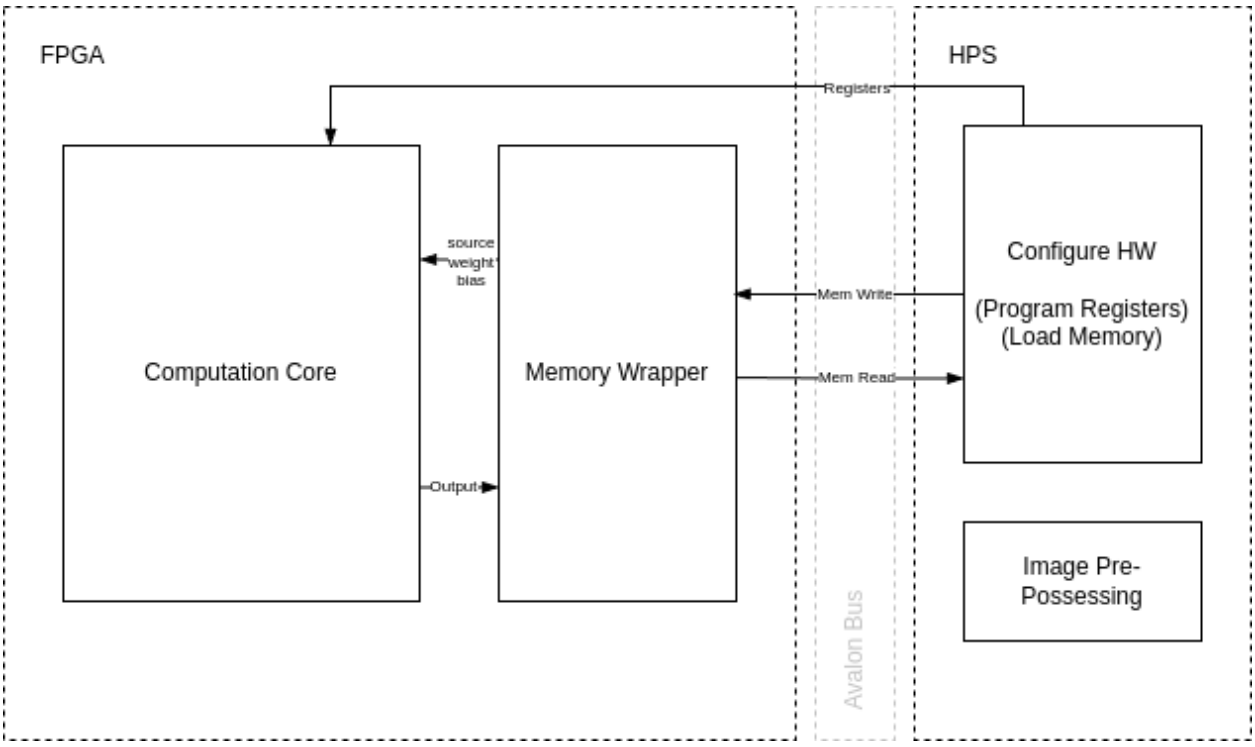

Figure 4: Block Diagram

3.1 Software

The software written in C is mainly composed of four parts:

- `Ioctl` functions, which communicate with the hardware.
- `JPEG_parsing` functions, which convert JPEG images into inputs for the neural network.
- `Network` functions, which load the parameters and build the network structure, including convolutional and fully connected layers.
- The `main` function, which controls the overall execution flow of the program.

We also have Python software that trains the Dwarf model using the sign language dataset and writes the model's parameters and configuration to a text file. This file is then parsed by the C program to construct the network.

Additionally, we have a program that generates valid inputs and weights for a convolutional or fully connected layer of a specific shape. It writes them into a file formatted as a Verilog testbench. This program also generates the corresponding golden output, which is used to compare against the hardware output.

As the C main function starts running, it first detects the CNN hardware, then loads the weights and biases of the four convolutional layers and two fully connected layers into corresponding structures and combines them into a complete network structure.

After that, it parses the image pending for recognition. The software first downsamples and resizes the image to 32×32, and then sorts the pixels into three channels as the input to the network.

Next, it adds padding, sends the input and the weights of the first convolutional layer to the hardware using ioctl, repeatedly polls for a finish signal from the hardware, and retrieves the result back into user space to serve as the input for the next layer. This procedure is repeated layer by layer until the final output vector is produced.

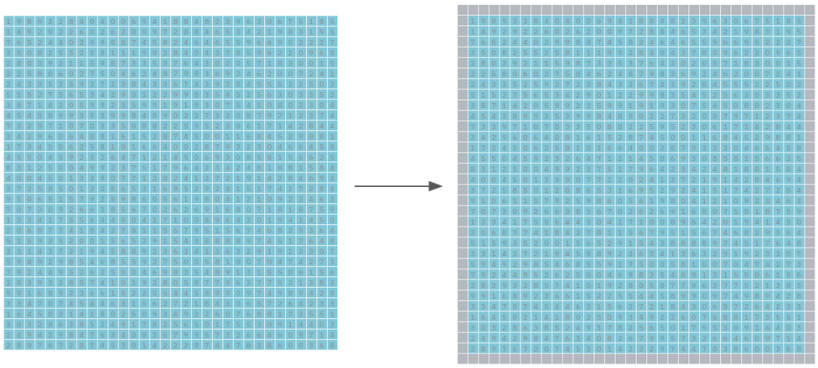

Figure 5: Illustration of zero padding a 32 by 32 matrix input

3.2 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 weights, and bias from the memory and loop through the entire source and destination channel pairs. After each kernel computation completed, the temporary output will write to the output memory. 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. Most of the continuous memory access is implemented in a pipelined manner.

3.2.1 Control FSM

The FSM has three main paths for computation. The default one is the Convolution that starts from 9 weight loads and then the kernel loop. Inside the kernel loop, it will fetch the adjacent 9 data from the memory and each one will be multiplied with corresponding weight value and accumulated with the previous result. The programmable register will decide if max pooling is needed after the convolution for each destination channel is ready. If Max Pooling path is taken, it will loop through all the pooling kernel and fetch the internal pool_size * pool_size. The pool rd address is updated based on the kernel position and also the internal position. The third path is the fully connected layer, the FSM will fetch data and weight one by one across the source channel and accumulate to one destination channel. The control FSM is responsible for computing all the memory access address for weight, data, and bias. It will also decides the intermittent output write location. The FSM follows the following diagram.

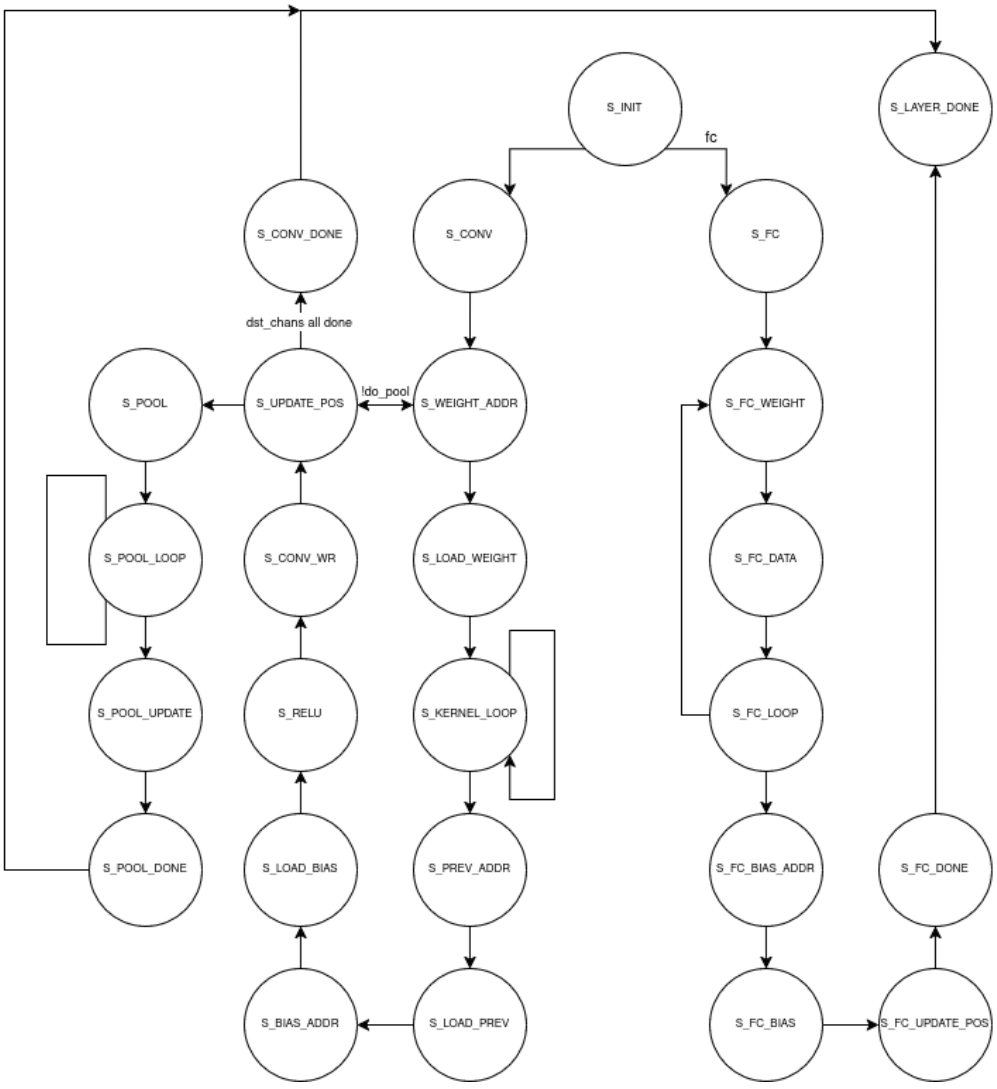

Figure 6: FSM

3.2.2 MAC

MAC units will have one input Q8.8 fixed-point multiplier and accumulator. Each time it will fetch memory data and multiply it with the weight loaded to the registers and accumulate it with the previous

sum. When input channels finish iterating, the result is accumulated in one output channel with a bias added, and iterate the output channel similarly until all the results are done. When it is doing fully connected layer, compute the data one by one while new data is being fetched from memory on the fly.

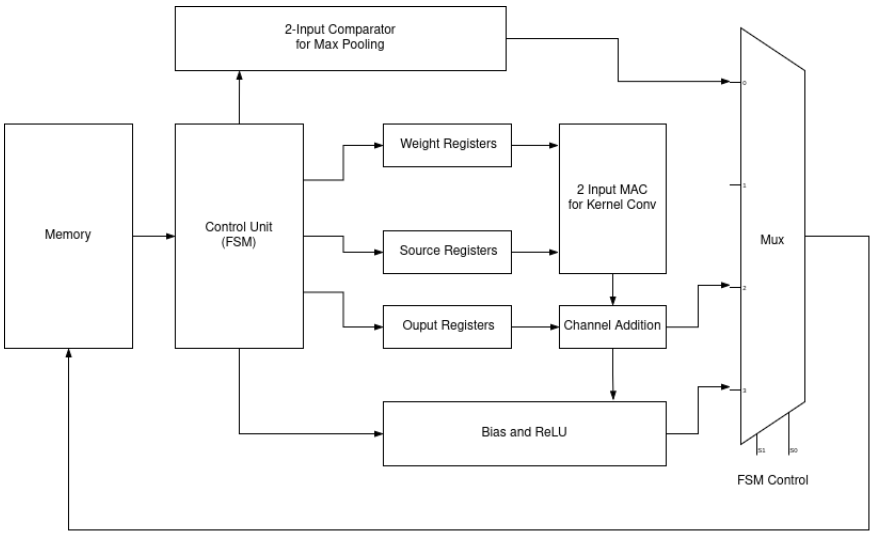

Figure 7: MAC Unit

3.3 Address Computation

The memory address computation is based on the FSM. The state decides which address will be issued to the memory and with what offset as its starting point.

- **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)
- **curr_pixel_pos:**
  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

- **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)

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

Division and multiplication with the pool_size is implemented with shift by 1 or 3 as pool_size is only configured to be 2 or 8.

3.4 Hardware-Software Interface

The software module will be responsible for filling the memory with input, weights, bias and configuring the accelerator, as well as reading the output from memory when the computation is done. The programmable registers are defined as in Table 1. The register file sub-addresses range from 0x00 to 0x20. Writing to the start field triggers the layer computation. For the starting addresses, there will be two writes issued with first time the lower 16 bits and second time upper 16 bits. As the registers access shares the same bus with the memory, when the write or the read goes to the register file, bit 19 of the address will be asserted and this bit will be used for the mux to select the data destination.

`num_src_chans`	number of source channels	uint16
`num_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
`fc`	fully connected layer	bool
`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	uint32
`weight_starting_address`	starting address of the weight	uint32
`bias_starting_address`	starting address of the bias	uint32
`output_starting_address`	starting address of the output	uint32
`start`	start layer computation	bool

Table 1: Variable Configuration

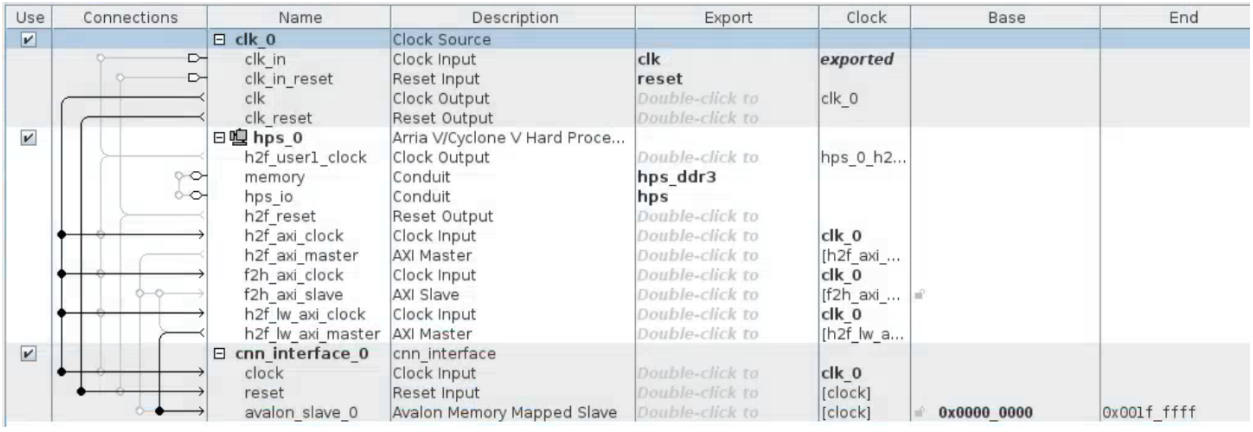

Figure 8: Qsys Diagram

After the software completes the configuration, it will keep reading the LAYER_DONE signal from the FPGA to proceed with reading all the output data.

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

4 Hardware Resource Utilization

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 Software following is the estimated memory usage for each layer, and given the 4450 Kbits FPGA memory, our scheme is: instantiate a BRAM (340 KB) which is bigger than the maximum memory usage of a layer (333KB), and the BRAM only store the data at certain layer .

Layer 1	72 KB
Layer 2	154 KB
Layer 3	88 KB
Layer 4	333 KB
Layer 5	33 KB
Layer 6	1.4 KB

Table 2: 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, comparisons and address computation part will rely on the DPS units, which we have sufficient on the FPGA.

```
+-------------------------------------------------------------------------+
; 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 % )                        ;
+---------------------------------+---------------------------------------+
```

Figure 9: Hardware Resource Utilization on Cyclone V FPGA

Resource	**Used**	**Total**	**Utilization**
ALMs (Logic)	1,498	32,070	5%
Registers	1,301	–	–
Block Memory Bits	2,785,424	4,065,280	69%
Pins	362	457	79%
DSP Blocks	21	87	24%

Table 3: FPGA Resource Utilization Summary

From the summary report of the hardware utilization, we can see

- Memory usage is significant (69%), indicating heavy BRAM requirements.
- DSP usage (24%) suggests MAC-intensive computation, it requires a lot of matrix and vector multiplication.
- Low logic (ALM) usage (5%) indicates plenty of remaining headroom for additional logic.

Due to the computation constraint, our design has a Fmax of 44MHz, thus, we have set the clock period to 25ns ensuring that there is no setup time violation from the FPGA at runtime.

5 Verification

We have designed the Python codes to simulate the behavior of the layers beside the PyTorch model. This Python model could generate random values as input and product expected output after convolution, pooling and fully connected for comparison. We have tested each convolution layer with and without the pooling and the fully connected layer using the randomly generated data and inspected the waveform at each stage to verify its correctness. The output from simulation is within 1-bit fixed point difference with the Python model using floating point for computation.

6 Improvement and Future

The design choice of using fixed point Q8.8 across all the input data, weight, and bias is not considered as the most optimal solution. As our model was trained in PyTorch with floating point, the conversion from floating to fixed will loss significant precision. This causes the computations to be very inaccurate

with respect to the original model. Also, a significant amount of our model weights are within the range of [-1, 1], the Q8.8 format actually wastes some bits and the multiplied data is approaching to zero eventually without providing any more meaningful information. We did not realize the issue until we have verified each individual layer with randomly generated data. It is crucial to have a matching reference model also using Q8.8 fixed point to simulate the behavior of the network.

7 Group Contribution

Xincheng Yu:

- Designed the Python Model to train the network
- Designed the C software driver to process the data, weight, bias and configure the accelerator
- Designed the Python reference model for layers

Yangfan Wang:

- Designed the hardware accelerator that performs convolution, max pooling and fully connected layers
- Assisted in the software debugging and performed RTL verification for the individual layers

Fengze Zhong:

- Designed the hardware-software interface and memory/registers system that handles the Avalon requests
- Designed the initial version of the convolution and max pooling layer RTL code

8 Conclusion

We successfully designed and implemented a CNN accelerator on the Cyclone V FPGA for real-time gesture recognition. By offloading convolution, pooling, ReLU activation, and fully connected computations to dedicated hardware modules, we achieved efficient matrix/vetors computation and reduced latency.

- The system demonstrated strong modularity and hardware-software co-design. The software running on the ARM Cortex-A9 handled data preprocessing, control flow, and memory interfacing, while the hardware accelerator performed the intensive fixed-point MAC computations. With this architecture, we achieved a good accuracy using a lightweight CNN model trained in PyTorch, and effectively mapped the model to a resource-constrained embedded platform.
- Our hardware utilized only 5% of logic elements, 24% of DSP blocks, and 69% of BRAM resources, indicating good optimization and headroom for expansion. This demonstrates that FPGA-based CNN acceleration is both feasible and efficient for small-scale classification tasks like gesture recognition.

Overall, this project showcased the viability of hardware acceleration for edge AI applications, and provided hands-on experience in RTL design, SoC integration, memory-mapped interfaces, and Linux driver development.

9 Code Listing

9.1 Hardware

9.1.1 RTL Design

```
module cnn_interface (
    input  logic         clk,
    input  logic         reset,
    input  logic [15:0] writedata,
    input  logic         write,
    input  logic         chipselect,
    input  logic [19:0] address,
    output logic [15:0] readdata
);

    logic [15:0]  bram_wr_data;
    logic         bram_we;
  logic [18:0]  bram_addr;
  logic [15:0]  bram_rd_data;

  logic [18:0]  bram_rd_addr;
  logic [18:0]  bram_wr_addr;

  parameter BRAM_SIZE = 19'd174080;

    //assign bram_we = chipselect && write && (~address[18]);

  logic [15:0]  hps_mem_wr_data;
  logic     hps_mem_wr;
  logic [18:0]  hps_mem_wr_addr;
  logic     hps_mem_rd;
  logic [18:0]  hps_mem_rd_addr;
  logic [15:0]  hps_mem_rd_data;

  always @(*) begin
    hps_mem_wr    = chipselect && write && (~address[19]);
    hps_mem_wr_addr = (hps_mem_wr) ? address[18:0] : 'd0;
    hps_mem_wr_data = (hps_mem_wr) ? writedata : 'd0;
    hps_mem_rd    = chipselect && ~write && (~address[19]);
    hps_mem_rd_addr = (hps_mem_rd) ? address[18:0] : 'd0;
  end

    bram #(.ADDR_WIDTH(19), .ENTRY_NUMBER(BRAM_SIZE) ,.DATA_WIDTH(16))
        bram_inst (
        .clk    (clk),
        .we     (bram_we),
        .addr   (bram_addr),
        .wdata  (bram_wr_data),
        .rdata  (bram_rd_data)
    );

  logic [15:0]  conv_wr_data;
  logic       conv_wr;
  logic [18:0]  conv_wr_addr;
```

```
logic [15:0]  conv_rd_data;
logic         conv_rd;
logic [18:0]  conv_rd_addr;

logic         valid;

conv conv_inst (
  .clk(clk),
  .reset(reset),
  .mem_rd_data_i(bram_rd_data),
  .mem_rd_addr_o(conv_rd_addr),
  .mem_rd(conv_rd),
  .mem_wr_data_o(conv_wr_data),
  .mem_wr_addr_o(conv_wr_addr),
  .mem_wr(conv_wr),
  .config_addr_i(address[18:0]),
  .config_write_i(write && address[19]),
  .chipselect_i(chipselect),
  .config_data_i(writedata),
  .valid(valid)
);

always @(*) begin
  if (hps_mem_wr) begin
    bram_wr_data = hps_mem_wr_data;
    bram_we      = hps_mem_wr;
    bram_wr_addr = hps_mem_wr_addr;
  end else begin
    bram_wr_data = conv_wr_data;
    bram_we      = conv_wr;
    bram_wr_addr = conv_wr_addr;
  end
end

logic hps_mem_rd_delayed;

always @(posedge clk) begin
  hps_mem_rd_delayed <= hps_mem_rd;
end

always @(*) begin
  bram_rd_addr = 'd0;
  if (conv_rd) begin
    bram_rd_addr = conv_rd_addr;
  end else begin
    bram_rd_addr = hps_mem_rd_addr;
  end
end

always @(*) begin
  readdata = 0;
  if (hps_mem_rd_delayed) begin
    readdata = bram_rd_data;
```

```
    end else begin
      readdata = valid;
    end
  end

  assign conv_rd_data = bram_rd_data;

  assign bram_addr  = bram_we ? bram_wr_addr : bram_rd_addr;

endmodule

module bram #(
    parameter ADDR_WIDTH = 19,
    parameter ENTRY_NUMBER = 163840,
    parameter DATA_WIDTH = 16
)(
    input  logic                    clk,
    input  logic                    we,          // Write enable
    input  logic [ADDR_WIDTH-1:0]   addr,
    input  logic [DATA_WIDTH-1:0]   wdata,
    output logic [DATA_WIDTH-1:0]   rdata
);

    // Inferred BRAM block with Quartus hint
    (* ramstyle = "block" *)
    logic [DATA_WIDTH-1:0] mem [ENTRY_NUMBER-1:0];

    always_ff @(posedge clk) begin
        if (we) mem[addr] <= wdata;

        rdata <= mem[addr]; // Synchronous read (1-cycle delay)
    end

endmodule

module conv (
  input clk,
  input reset,
  input   [15:0]  mem_rd_data_i,
  output  [18:0]  mem_rd_addr_o,
  output      mem_rd,
  output  [15:0]  mem_wr_data_o,
  output  [18:0]  mem_wr_addr_o,
  output      mem_wr,
  input [4:0] config_addr_i,
  input     config_write_i,
  input     chipselect_i,
  input [15:0]  config_data_i,
  output logic  valid
);

  logic [15:0]  num_src_chans;
  logic [15:0]  num_dst_chans;
  logic [15:0]  num_cols;
```

```
logic [15:0]  num_rows;
logic     do_pool;
logic     fc;
logic [15:0]  pool_size;
logic [15:0]  pool_stride;
logic [31:0]  data_starting_addr;
logic [31:0]  weight_starting_addr;
logic [31:0]  bias_starting_addr;
logic [31:0]  output_starting_addr;
logic     start;

always_ff @(posedge clk) begin
  if (reset) begin
    num_src_chans   <= 'd0;
    num_dst_chans   <= 'd0;
    num_cols    <= 'd0;
    num_rows    <= 'd0;
    do_pool     <= 0;
    fc        <= 0;
    pool_size     <= 'd0;
    pool_stride   <= 'd0;
    data_starting_addr    <= 'd0;
    weight_starting_addr  <= 'd0;
    bias_starting_addr    <= 'd0;
    output_starting_addr  <= 'd0;
    start         <= 0;
  end else begin
    if (chipselect_i && config_write_i) begin
      case (config_addr_i)
        'd0: num_src_chans  <= config_data_i;
        'd1: num_dst_chans  <= config_data_i;
        'd2: num_cols   <= config_data_i;
        'd3: num_rows   <= config_data_i;
        'd4: fc       <= config_data_i[0];
        'd5: do_pool    <= config_data_i[0];
        'd6: pool_size    <= config_data_i;
        'd7: pool_stride  <= config_data_i;
        'd8: data_starting_addr[15:0]   <= config_data_i;
        'd9: data_starting_addr[31:16]  <= config_data_i;
        'd10: weight_starting_addr[15:0]  <= config_data_i;
        'd11: weight_starting_addr[31:16] <= config_data_i;
        'd12: bias_starting_addr[15:0]    <= config_data_i;
        'd13: bias_starting_addr[31:16]   <= config_data_i;
        'd14: output_starting_addr[15:0]  <= config_data_i;
        'd15: output_starting_addr[31:16] <= config_data_i;
        'd16: start             <= 1;
      endcase
    end
    else if (start) begin
      start <= 0;
    end
  end
end
```

```
logic [15:0] curr_input_x;
logic [15:0] curr_input_y;

logic [15:0] curr_output_x;
logic [15:0] curr_output_y;

logic [15:0]  curr_src_chan;
logic [15:0]  curr_dst_chan;

logic mem_weight_rd;
logic [18:0] mem_weight_rd_addr;
logic [18:0] mem_weight_rd_addr_delayed;
logic mem_weight_valid;

logic signed [15:0] weight_data [8:0];

logic mem_data_rd;
logic [18:0] mem_data_rd_addr;
logic [18:0] mem_data_rd_addr_delayed;
logic mem_data_valid;

logic mem_prev_acc_rd;
logic [15:0] mem_prev_acc_rd_addr;

logic mem_bias_rd;
logic [15:0] mem_bias_rd_addr;

logic signed [15:0] bias_data;

/******** POOL ********/
logic [15:0] curr_pool_x;
logic [15:0] curr_pool_y;

logic [15:0] curr_pool_stride_x;
logic [15:0] curr_pool_stride_y;
logic [15:0] curr_pool_stride_x_delayed;
logic [15:0] curr_pool_stride_y_delayed;

logic [18:0] mem_pool_rd_addr;
logic    mem_pool_rd;
logic [18:0] mem_pool_rd_addr_delayed;
logic    mem_pool_valid;
logic [18:0] mem_pool_wr_addr;
logic [15:0] mem_pool_wr_data;
logic    mem_pool_wr;

logic [18:0] mem_fc_rd_addr;
logic    mem_fc_rd;
logic [18:0] mem_fc_rd_addr_delayed;
logic    mem_fc_valid;
logic [18:0] mem_fc_wr_addr;
logic [15:0] mem_fc_wr_data;
logic    mem_fc_wr;
```

```
logic signed [15:0] curr_pool_max;

logic [4:0]  pool_cycle_cnt;

logic       mem_output_wr;
logic [18:0]  mem_output_wr_addr;
logic [15:0]  mem_output_wr_data;

/******** FC ********/
logic [15:0] curr_fc_w_row;
logic [15:0] curr_fc_w_col;

localparam S_INIT        = 'd0;
localparam S_CONV        = 'd1;
localparam S_SET_WEIGHT_ADDR  = 'd2;
localparam S_LOAD_WEIGHT     = 'd3;
localparam S_COMPUTE_POS     = 'd4;
localparam S_KERNEL_LOOP     = 'd5;
localparam S_SET_PREV_ADDR     = 'd7;
localparam S_FETECH_PREV_SRC_CH = 'd8;
localparam S_CHECK_CH      = 'd9;
localparam S_SET_BIAS_ADDR     = 'd10;
localparam S_LOAD_BIAS       = 'd11;
localparam S_RELU        = 'd12;
localparam S_RESULT_WRITE_BACK  = 'd13;
localparam S_UPDATE_POS_AND_CH  = 'd14;
localparam S_CONV_DONE       = 'd15;
localparam S_POOL        = 'd16;
localparam S_POOL_FETCH_LOOP  = 'd17;
localparam S_POOL_FETCH_LOOP_Y  = 'd18;
localparam S_POOL_WRITE_BACK  = 'd19;
localparam S_POOL_UPDATE_POS  = 'd20;
localparam S_FC          = 'd21;
localparam S_FC_LOOP       = 'd22;
localparam S_FC_LOOP_WEIGHT    = 'd23;
localparam S_FC_LOOP_ACC     = 'd24;
localparam S_FC_SET_BIAS_ADDR = 'd25;
localparam S_FC_BIAS       = 'd26;
localparam S_FC_WRITE_BACK     = 'd27;
localparam S_FC_UPDATE_POS     = 'd28;
localparam S_POOL_DONE       = 'd29;
localparam S_FC_DONE       = 'd30;
localparam S_LAYER_DONE      = 'd128;

logic [7:0] curr_state;
logic [7:0] next_state;

always @(posedge clk) begin
  if (reset) begin
    curr_state <= S_INIT;
  end else begin
    curr_state <= next_state;
  end
```

```
end

logic [4:0] cycle_cnt;

always @(posedge clk) begin
  case (curr_state)
    S_LOAD_WEIGHT: cycle_cnt <= (mem_weight_rd_addr_delayed == 'd8) ? 'd0 :
        cycle_cnt + 'd1;
    S_KERNEL_LOOP: cycle_cnt <= (mem_data_rd_addr_delayed == 'd8) ? 'd0 :
        cycle_cnt + 'd1;
    default: cycle_cnt <= 'd0;
  endcase
end

always @(posedge clk) begin
  if (reset) begin
    valid <= 0;
  end else begin
    case (curr_state)
      S_LAYER_DONE:    valid <= 1;
      S_INIT:      valid <= start ? 0 : valid;
    endcase
  end
end


always @(*) begin
  case (curr_state)
    S_INIT:          next_state = (start) ? ((fc) ? S_FC : S_CONV) : S_INIT;
    S_CONV:          next_state = S_SET_WEIGHT_ADDR;
    S_SET_WEIGHT_ADDR:     next_state = S_LOAD_WEIGHT;
    S_LOAD_WEIGHT:       next_state = (mem_weight_rd_addr_delayed == 'd8) ?
        S_COMPUTE_POS : S_LOAD_WEIGHT;
    S_COMPUTE_POS:       next_state = S_SET_PREV_ADDR;
    S_SET_PREV_ADDR:     next_state = S_FETECH_PREV_SRC_CH;
    S_FETECH_PREV_SRC_CH: next_state = S_KERNEL_LOOP;
    S_KERNEL_LOOP:       next_state = (mem_data_rd_addr_delayed == 'd8) ?
        S_CHECK_CH : S_KERNEL_LOOP;
    S_CHECK_CH:        next_state = (curr_src_chan == num_src_chans - 'd1) ?
        S_SET_BIAS_ADDR : S_RESULT_WRITE_BACK;
    S_SET_BIAS_ADDR:     next_state = S_LOAD_BIAS;
    S_LOAD_BIAS:       next_state = S_RELU;
    S_RELU:          next_state = S_RESULT_WRITE_BACK;
    S_RESULT_WRITE_BACK:   next_state = S_UPDATE_POS_AND_CH;
    S_UPDATE_POS_AND_CH:   begin
      if ((curr_input_x == (num_cols - 'd2)) && (curr_input_y == (num_rows -
          'd2))) begin
        if ((curr_src_chan == num_src_chans - 'd1)) begin
          if (do_pool) begin
            next_state = S_POOL;
          end else begin
            if (curr_dst_chan == num_dst_chans - 'd1) begin
              next_state = S_CONV_DONE;
            end else begin
```

```
            next_state = S_SET_WEIGHT_ADDR;
          end
        end
      end else begin
        next_state = S_SET_WEIGHT_ADDR;
      end
    end else begin
      next_state = S_COMPUTE_POS;
    end
  end
  S_CONV_DONE:        next_state = (do_pool) ? S_POOL : S_LAYER_DONE;
  S_POOL:             next_state = S_POOL_FETCH_LOOP;
  S_POOL_FETCH_LOOP:  begin
    if ((curr_pool_stride_x_delayed == pool_size - 'd1) && (
        curr_pool_stride_y_delayed == pool_size - 'd1)) begin
      next_state = S_POOL_WRITE_BACK;
    end else begin
      next_state = S_POOL_FETCH_LOOP;
    end
  end
  S_POOL_WRITE_BACK:  next_state = S_POOL_UPDATE_POS;
  S_POOL_UPDATE_POS:  begin
    if (((curr_pool_x * pool_size) == (num_cols - pool_size - 'd2)) && ((
        curr_pool_y * pool_size) == (num_rows - pool_size - 'd2))) begin
      next_state = S_POOL_DONE;
    end else begin
      next_state = S_POOL_FETCH_LOOP;
    end
  end
  S_POOL_DONE:        begin
    if (curr_dst_chan == num_dst_chans - 'd1 && curr_src_chan ==
        num_src_chans - 'd1) begin
      next_state = S_LAYER_DONE;
    end else begin
      next_state = S_CONV;
    end
  end
  S_FC:               next_state = S_FC_LOOP;
  S_FC_LOOP:          next_state = S_FC_LOOP_WEIGHT;
  S_FC_LOOP_WEIGHT:   next_state = S_FC_LOOP_ACC;
  S_FC_LOOP_ACC:      begin
    if (curr_fc_w_row == num_src_chans - 'd1) begin
      next_state = S_FC_SET_BIAS_ADDR;
    end else begin
      next_state = S_FC_LOOP;
    end
  end
  S_FC_SET_BIAS_ADDR: next_state = S_FC_BIAS;
  S_FC_BIAS:          next_state = S_FC_WRITE_BACK;
  S_FC_WRITE_BACK:    next_state = S_FC_UPDATE_POS;
  S_FC_UPDATE_POS:    begin
    if (curr_fc_w_col == num_dst_chans - 'd1) begin
      next_state = S_FC_DONE;
    end else begin
```

```
        next_state = S_FC_LOOP;
      end
    end
    S_FC_DONE:          next_state = S_LAYER_DONE;
    S_LAYER_DONE:       next_state = S_INIT;
    default:          next_state = S_INIT;
  endcase
end

always @(posedge clk) begin
  if (reset) begin
    curr_input_x <= 'd1;
    curr_input_y <= 'd1;
  end else begin
    case (curr_state)
      S_LOAD_WEIGHT: begin
        curr_input_x <= 'd1;
        curr_input_y <= 'd1;
      end

      S_UPDATE_POS_AND_CH: begin
        curr_input_x <= (curr_input_x == (num_cols - 'd2)) ? 'd1 : (
            curr_input_x + 'd1);
        curr_input_y <= (curr_input_x == (num_cols - 'd2)) ? ((curr_input_y
            == (num_rows - 'd2)) ? 'd1 : (curr_input_y + 'd1)) :
            curr_input_y;
      end
    endcase
  end
end

logic [15:0] curr_kernel_pos;
logic [15:0] curr_pixel_pos;

always @(posedge clk) begin
  if (reset) begin
    curr_kernel_pos <= 'd0;
  end else begin
    if (curr_state == S_COMPUTE_POS) begin
      curr_kernel_pos <= curr_input_y * num_cols + curr_input_x;
    end
  end
end

always @(posedge clk) begin
  if (reset) begin
    curr_src_chan <= 'd0;
    curr_dst_chan <= 'd0;
  end else begin
    case (curr_state)
      S_UPDATE_POS_AND_CH: begin
        if (~do_pool) begin
          if ((curr_input_x == (num_cols - 'd2)) && (curr_input_y == (
              num_rows - 'd2))) begin
```

```
            curr_src_chan <= (curr_src_chan == num_src_chans - 'd1) ? 'd0 :
                (curr_src_chan + 'd1);
            curr_dst_chan <= (curr_src_chan == num_src_chans - 'd1) ? ((
                curr_dst_chan == num_dst_chans - 'd1) ? 'd0 : (curr_dst_chan
                 + 'd1)) : curr_dst_chan;
          end
        end else begin
          if (curr_src_chan != num_src_chans - 'd1) begin
            if ((curr_input_x == (num_cols - 'd2)) && (curr_input_y == (
                num_rows - 'd2))) begin
              curr_src_chan <= (curr_src_chan == num_src_chans - 'd1) ? 'd0
                 : (curr_src_chan + 'd1);
              curr_dst_chan <= (curr_src_chan == num_src_chans - 'd1) ? ((
                 curr_dst_chan == num_dst_chans - 'd1) ? 'd0 : (
                 curr_dst_chan + 'd1)) : curr_dst_chan;
            end
          end
        end
      end
      S_POOL_DONE: begin
        curr_src_chan <= (curr_src_chan == num_src_chans - 'd1) ? 'd0 : (
            curr_src_chan + 'd1);
        curr_dst_chan <= (curr_src_chan == num_src_chans - 'd1) ? ((
            curr_dst_chan == num_dst_chans - 'd1) ? 'd0 : (curr_dst_chan + '
            d1)) : curr_dst_chan;
      end
    endcase
  end
end

always @(*)begin
  curr_pixel_pos = 'd0;
  if (curr_state == S_KERNEL_LOOP) begin
    case (cycle_cnt)
      'd0:  curr_pixel_pos = curr_kernel_pos - num_cols - 'd1;
      'd1:  curr_pixel_pos = curr_kernel_pos - num_cols;
      'd2:  curr_pixel_pos = curr_kernel_pos - num_cols + 'd1;
      'd3:  curr_pixel_pos = curr_kernel_pos - 'd1;
      'd4:  curr_pixel_pos = curr_kernel_pos;
      'd5:  curr_pixel_pos = curr_kernel_pos + 'd1;
      'd6:  curr_pixel_pos = curr_kernel_pos + num_cols - 'd1;
      'd7:  curr_pixel_pos = curr_kernel_pos + num_cols;
      'd8:  curr_pixel_pos = curr_kernel_pos + num_cols + 'd1;
    endcase
  end
end

//logic [31:0] kernel_acc;
//logic [31:0] weight_mult_src;
//logic [15:0] prev_src_result;

//logic [15:0] mem_src_data;
//logic [15:0] mem_weight_data;
```

```
//always @(posedge clk) begin
//  if (reset) begin
//    kernel_acc <= 'd0;
//  end else begin
//    if (curr_state == S_COMPUTE_POS) kernel_acc <= 'd0;
//    else if (curr_state == S_KERNEL_LOOP) begin
//      if (cycle_cnt < 'd9) kernel_acc <= kernel_acc + weight_mult_src;
//      else if (cycle_cnt == 'd18) kernel_acc <= kernel_acc +
   prev_src_result;
//    end
//  end
//end

//assign weight_mult_src = mem_src_data * mem_weight_data;

logic [18:0] mem_rd_addr;
logic [18:0] mem_wr_addr;
logic [15:0] mem_wr_data;

assign mem_rd_addr_o = mem_rd ? mem_rd_addr : 'hBEEF;
assign mem_wr_addr_o = mem_wr ? mem_wr_addr : 'hBEEF;
assign mem_wr_data_o = mem_wr ? mem_wr_data : 'd0;

//TODO OR this rd signal
assign mem_rd = mem_weight_rd | mem_data_rd | mem_prev_acc_rd | mem_bias_rd
   | mem_pool_rd | mem_fc_rd;

assign mem_wr = mem_output_wr | mem_pool_wr | mem_fc_wr;

always @(*) begin
  mem_weight_rd = 0;
  case (curr_state)
    S_LOAD_WEIGHT: begin
      if (mem_weight_rd_addr < 'd9) mem_weight_rd = 1;
    end
  endcase
end

always @(posedge clk) begin
  mem_weight_valid      <= mem_weight_rd;
  mem_weight_rd_addr_delayed  <= mem_weight_rd_addr;
end

always @(posedge clk) begin
  if (reset) begin
    mem_weight_rd_addr <= 'hBEEF;
  end else begin
    case (curr_state)
      S_SET_WEIGHT_ADDR:  mem_weight_rd_addr <= 'd0;
      S_LOAD_WEIGHT:      mem_weight_rd_addr <= (mem_weight_rd_addr == 'd8) ?
          'hBEEF : (mem_weight_rd_addr + 'd1);
    endcase
  end
end
```

```
always @(posedge clk) begin
  if (mem_weight_valid) begin
    weight_data[mem_weight_rd_addr_delayed] <= mem_rd_data_i;
  end
end

always @(*) begin
  mem_data_rd = 0;
  case (curr_state)
    S_KERNEL_LOOP: begin
      if (cycle_cnt < 'd9) mem_data_rd = 1;
    end
  endcase
end

always @(posedge clk) begin
  mem_data_valid        <= mem_data_rd;
  mem_data_rd_addr_delayed  <= cycle_cnt;
end

assign mem_data_rd_addr = curr_pixel_pos;

logic signed [31:0] kernel_acc;
logic signed [15:0] kernel_acc_real;

assign kernel_acc_real = kernel_acc >>> 8;

always @(posedge clk) begin
  if (reset) begin
    kernel_acc <= 'd0;
  end else begin
    case (curr_state)
      S_KERNEL_LOOP: begin
        if (mem_data_valid) begin
          kernel_acc <= kernel_acc + $signed(mem_rd_data_i) * $signed(
              weight_data[mem_data_rd_addr_delayed]);
        end
      end
      S_FETECH_PREV_SRC_CH: begin
        if (curr_src_chan == 'd0) kernel_acc <= 'd0;
        else kernel_acc <= $signed(mem_rd_data_i) <<< 8;
      end
    endcase
  end
end

logic signed [31:0] result_to_write;

logic signed [31:0] relu_result;

assign relu_result = result_to_write + bias_data;

always @(posedge clk) begin
```

```
  if (reset) begin
    result_to_write <= 'd0;
  end else begin
    case (curr_state)
      S_CHECK_CH:     result_to_write <= kernel_acc >>> 8;
      S_RELU:       result_to_write <= (~relu_result[31]) ? (relu_result) :
          'd0;
    endcase
  end
end

always @(posedge clk) begin
  if (reset) begin
    curr_output_x <= 'd0;
    curr_output_y <= 'd0;
  end else begin
    case (curr_state)
      S_CONV: begin
        curr_output_x <= 'd0;
        curr_output_y <= 'd0;
      end
      S_UPDATE_POS_AND_CH: begin
        curr_output_x <= (curr_output_x == num_cols - 'd3) ? 'd0 : (
            curr_output_x + 'd1);
        curr_output_y <= (curr_output_x == num_cols - 'd3) ? ((curr_output_y
            == num_rows - 'd3) ? 'd0 : (curr_output_y + 'd1)) :
            curr_output_y;
      end
    endcase
  end
end


always @(*) begin
  mem_prev_acc_rd = 0;
  mem_prev_acc_rd_addr = 'hBEEF;
  case (curr_state)
    S_SET_PREV_ADDR: begin
      mem_prev_acc_rd     = 1;
      mem_prev_acc_rd_addr  = curr_output_y * (num_cols - 'd2) +
          curr_output_x + curr_dst_chan * (num_cols - 'd2) * (num_rows - 'd2
          );
    end
  endcase
end

always @(*) begin
  mem_rd_addr = 'hBEEF;
  case (curr_state)
    S_SET_BIAS_ADDR:     mem_rd_addr = mem_bias_rd_addr + bias_starting_addr;
    S_SET_PREV_ADDR:     mem_rd_addr = mem_prev_acc_rd_addr +
        output_starting_addr;
    S_LOAD_WEIGHT:       mem_rd_addr = mem_weight_rd_addr +
        weight_starting_addr + 9 * curr_src_chan + 9 * num_src_chans *
```

```
        curr_dst_chan;
    S_KERNEL_LOOP:       mem_rd_addr = curr_pixel_pos + data_starting_addr +
        num_cols * num_rows * curr_src_chan;
    S_POOL_FETCH_LOOP:     mem_rd_addr = mem_pool_rd_addr +
        output_starting_addr;
    S_FC_LOOP:          mem_rd_addr = mem_fc_rd_addr + data_starting_addr;
    S_FC_LOOP_WEIGHT:    mem_rd_addr = mem_fc_rd_addr + weight_starting_addr;
    S_FC_SET_BIAS_ADDR:    mem_rd_addr = mem_fc_rd_addr + bias_starting_addr;
  endcase
end

always @(*) begin
  mem_bias_rd = 0;
  mem_bias_rd_addr = 'hBEEF;
  case (curr_state)
    S_SET_BIAS_ADDR: begin
      mem_bias_rd     = 1;
      mem_bias_rd_addr  = curr_dst_chan;
    end
  endcase
end

always @(posedge clk) begin
  if (reset) begin
    bias_data <= 'd0;
  end else begin
    case (curr_state)
      S_LOAD_BIAS: bias_data <= mem_rd_data_i;
    endcase
  end
end

always @(*) begin
  mem_output_wr = 0;
  mem_output_wr_addr = 'hBEEF;
  mem_output_wr_data = 'd0;
  case (curr_state)
    S_RESULT_WRITE_BACK: begin
      mem_output_wr = 1;
      mem_output_wr_addr = curr_output_y * (num_cols - 'd2) + curr_output_x
          + curr_dst_chan * (num_cols - 'd2) * (num_rows - 'd2);
      mem_output_wr_data = result_to_write[0 +: 16];
    end
  endcase
end

always @(*) begin
  mem_wr_addr = 'd0;
  mem_wr_data = 'd0;
  case (curr_state)
    S_RESULT_WRITE_BACK: begin
      mem_wr_addr = mem_output_wr_addr + output_starting_addr;
      mem_wr_data = mem_output_wr_data;
    end
```

```
    S_POOL_WRITE_BACK: begin
      mem_wr_addr = mem_pool_wr_addr + output_starting_addr;
      mem_wr_data = mem_pool_wr_data;
    end
    S_FC_WRITE_BACK: begin
      mem_wr_addr = mem_fc_wr_addr + output_starting_addr;
      mem_wr_data = mem_fc_wr_data;
    end
  endcase
end

/********* POOL ********/

always @(posedge clk) begin
  mem_pool_valid        <= mem_pool_rd;
  curr_pool_stride_x_delayed  <= curr_pool_stride_x;
  curr_pool_stride_y_delayed  <= curr_pool_stride_y;
end

always @(posedge clk) begin
  if (reset) begin
    curr_pool_x <= 'd0;
    curr_pool_y <= 'd0;
  end else begin
    case (curr_state)
      S_POOL: begin
        curr_pool_x <= 'd0;
        curr_pool_y <= 'd0;
      end
      S_POOL_UPDATE_POS: begin
        //if ((curr_pool_stride_x == pool_size - 'd1) && (curr_pool_stride_y
            == pool_size - 'd1)) begin
          curr_pool_x <= ((curr_pool_x * pool_size) == (num_cols - pool_size
              - 'd2)) ? 'd0 : (curr_pool_x + 'd1);
          curr_pool_y <= ((curr_pool_x * pool_size) == (num_cols - pool_size
              - 'd2)) ? (((curr_pool_y * pool_size) == (num_rows -
             pool_size - 'd2)) ? 'd0 : (curr_pool_y + 'd1)) : curr_pool_y;
        //end
      end
    endcase

  end
end

always @(posedge clk) begin
  if (reset) begin
    curr_pool_stride_x <= 'd0;
    curr_pool_stride_y <= 'd0;
  end else begin
    case (curr_state)
      S_POOL_FETCH_LOOP: begin
        curr_pool_stride_x <= (curr_pool_stride_x == pool_size - 'd1) ? 'd0
            : (curr_pool_stride_x + 'd1);
        curr_pool_stride_y <= (curr_pool_stride_x == pool_size - 'd1) ? ((
```

```
            curr_pool_stride_y == pool_size - 'd1) ? 'd0 : (
            curr_pool_stride_y + 'd1)) : curr_pool_stride_y;
      end
      S_POOL_UPDATE_POS: begin
        curr_pool_stride_x <= 'd0;
        curr_pool_stride_y <= 'd0;
      end
    endcase
  end
end

logic [15:0] pool_cnt_max;
assign pool_cnt_max = pool_size * pool_size;

logic [15:0] pool_cnt;

always @(posedge clk) begin
  if (reset) begin
    pool_cnt <= 'd0;
  end else begin
    case (curr_state)
      S_POOL_FETCH_LOOP: pool_cnt <= pool_cnt + 'd1;
      S_POOL_UPDATE_POS: pool_cnt <= 'd0;
    endcase
  end
end


always @(*) begin
  mem_pool_rd_addr  = 'd0;
  mem_pool_rd      = 0;
  case (curr_state)
    S_POOL_FETCH_LOOP: begin
      if (pool_cnt < pool_cnt_max) begin
        mem_pool_rd_addr = (curr_pool_y * 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);
        mem_pool_rd    = 1;
      end
    end
  endcase
end

always @(posedge clk) begin
  if (reset) begin
    curr_pool_max <= 'h8000;
  end else begin
    case (curr_state)
      S_POOL:        curr_pool_max <= 'h8000;
      S_POOL_FETCH_LOOP:  begin
        if (mem_pool_valid && $signed(mem_rd_data_i) > curr_pool_max) begin
          curr_pool_max <= $signed(mem_rd_data_i);
        end
      end
```

```
      S_POOL_UPDATE_POS:  curr_pool_max <= 'h8000;
    endcase
  end
end

always @(*) begin
  mem_pool_wr_addr  = 'd0;
  mem_pool_wr      = 0;
  mem_pool_wr_data  = 'd0;
  case (curr_state)
    S_POOL_WRITE_BACK: begin
      if (pool_size == 'd2) begin
        mem_pool_wr_addr  = (curr_pool_y) * ((num_cols - 'd2) >>> 1) + (
            curr_pool_x);
      end else begin
        mem_pool_wr_addr  = (curr_pool_y) * ((num_cols - 'd2) >>> 3) + (
            curr_pool_x);
      end
      mem_pool_wr      = 1;
      mem_pool_wr_data  = curr_pool_max;
    end
  endcase
end

/******** FC *********/
always @(posedge clk) begin
  if (reset) begin
    curr_fc_w_row <= 'd0;
    curr_fc_w_col <= 'd0;
  end else begin
    case (curr_state)
      S_FC_LOOP_ACC: begin
        curr_fc_w_row <= (curr_fc_w_row == num_src_chans - 'd1) ? 'd0 : (
            curr_fc_w_row + 'd1);
      end
      S_FC_UPDATE_POS: begin
        curr_fc_w_col <= (curr_fc_w_col == num_dst_chans - 'd1) ? 'd0 : (
            curr_fc_w_col + 'd1);
      end
    endcase
  end
end

always @(*) begin
  mem_fc_rd_addr    = 'd0;
  mem_fc_rd      = 0;
  case (curr_state)
    S_FC_LOOP:         begin
      mem_fc_rd_addr  = curr_fc_w_row;
      mem_fc_rd    = 1;
    end
    S_FC_LOOP_WEIGHT: begin
      mem_fc_rd_addr  = curr_fc_w_row * num_dst_chans + curr_fc_w_col;
      mem_fc_rd    = 1;
```

```
      end
      S_FC_SET_BIAS_ADDR: begin
        mem_fc_rd_addr  = curr_fc_w_col;
        mem_fc_rd   = 1;
      end
      default:      begin
        mem_fc_rd_addr  = 'd0;
        mem_fc_rd   = 0;
      end
    endcase
  end

  logic signed [31:0] fc_acc;
  logic signed [15:0] fc_data;

  always @(posedge clk) begin
    if (reset) begin
      fc_acc  <= 'd0;
      fc_data <= 'd0;
    end else begin
      case (curr_state)
        S_FC:          fc_acc <= 'd0;
        S_FC_LOOP_WEIGHT:   fc_data <= mem_rd_data_i;
        S_FC_LOOP_ACC:     fc_acc <= fc_acc + $signed(fc_data) * $signed(
            mem_rd_data_i);
        S_FC_BIAS:       fc_acc <= fc_acc + ($signed(mem_rd_data_i) <<< 8);
        S_FC_UPDATE_POS:  fc_acc <= 'd0;
      endcase
    end
  end

  always @(*) begin
    mem_fc_wr_addr    = 'd0;
    mem_fc_wr     = 0;
    mem_fc_wr_data    = 'd0;
    case (curr_state)
      S_FC_WRITE_BACK:  begin
        mem_fc_wr_addr  = curr_fc_w_col;
        mem_fc_wr   = 1;
        mem_fc_wr_data  = fc_acc >>> 8;
      end
    endcase
  end
endmodule
```

9.1.2 Testbench

```
    module tb_top ();
  logic clk;
  logic reset;

  initial begin
    clk = 1;
    forever #5 clk = ~clk;
```

```
end

reg chipselect;
reg [19:0] addr;
reg wr;
reg [15:0] data;

initial begin
  reset <= 1;
  #20;
  reset <= 0;
  chipselect <= 1;
  wr <= 1;

  addr <= 0 ;
  data <= 'h2f ;
  #10;
  addr <= 1 ;
  data <= 'h4e ;
  #10;
  addr <= 2 ;
  data <= 'h86 ;
  #10;
  addr <= 3 ;
  data <= 'h6f ;
  #10;
  addr <= 4 ;
  data <= 'h4b ;
  #10;
  addr <= 5 ;
  data <= 'h9d ;
  #10;

      //Remaining Address Write
      ...

  //Start Config
  addr <= {1'b1, 19'd0};
  data <= 64;
  #10;
  addr <= {1'b1, 19'd1};
  data <= 256;
  #10;
  addr <= {1'b1, 19'd2};
  data <= 10;
  #10;
  addr <= {1'b1, 19'd3};
  data <= 10;
  #10;
  addr <= {1'b1, 19'd4};
  data <= 0;
  #10;
  addr <= {1'b1, 19'd5};
  data <= 1;
```

```
    #10;
    addr <= {1'b1, 19'd6};
    data <= 8;
    #10;
    addr <= {1'b1, 19'd7};
    data <= 8;
    #10;
    addr <= {1'b1, 19'd8};
    data <= 'h0;
    #10;
    addr <= {1'b1, 19'd9};
    data <= 'h0;
    #10;
    addr <= {1'b1, 19'd10};
    data <= 'd6400;
    #10;
    addr <= {1'b1, 19'd11};
    data <= 'h0;
    #10;
    addr <= {1'b1, 19'd12};
    data <= 'd153856;
    #10;
    addr <= {1'b1, 19'd13};
    data <= 'h0;
    #10;
    addr <= {1'b1, 19'd14};
    data <= 'd154112;
    #10;
    addr <= {1'b1, 19'd15};
    data <= 'h0;
    #10;
    addr <= {1'b1, 19'd16};
    data <= 'd1;
    #10;
    wr <= 0;
    #500000000;
    $stop();
  end

  cnn_interface cnn_interface_inst (
    .clk(clk),
    .reset(reset),
    .writedata(data),
    .write(wr),
    .chipselect(chipselect),
    .address(addr)
  );
endmodule
```

9.2 Software

9.2.1 main function

```
#include <stdio.h>
#include "memory.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdbool.h>
#include "network.h"
#include "get_jgp.h"

#define INIT_X 300
#define INIT_Y 200

int memory_fd;

void read_value() {
  mem_arg_t vla;
  vla.pos = 10;
  if (ioctl(memory_fd, MEM_READ, &vla)) {
    perror("ioctl(MEM_READ) failed");
  }
  printf("%d\n", vla.value);
}

void write_value() {
  mem_arg_t vla;
  vla.pos = 10;
  vla.value = 20;
  if (ioctl(memory_fd, MEM_WRITE, &vla)) {
    perror("ioctl(MEM_READ) failed");
  }
}

Conv_IO get_input() {
  FILE* f_in = fopen("input", "r");
  int w, h, d;

  fscanf(f_in, "%d%d%d", &d, &w, &h);
  Conv_IO input = CreateConv(w, h, d);
  for (int i = 0; i < w * h * d; i ++ ) {
    fscanf(f_in, "%f", &input.parmas[i]);
  }
  return input;
}

Conv_IO run_conv(Conv_IO input, Conv conv, int pool_size) {
  int num_input = input.w*input.h*input.d, num_bias = conv.out,
      num_weights = conv.out*conv.in*conv.kw*conv.kh;
  short *input_p, *network, *bias, *output;
  input_p = (short*)malloc(num_input*sizeof(short));
  network = (short*)malloc(num_weights*sizeof(short));
```

```
  bias = (short*)malloc(num_bias*sizeof(short));

  for(int i = 0; i < num_input; i ++ ) {
    input_p[i] = float_to_fixed(input.parmas[i]);
    if (pool_size == 8) {
      input_p[i] <<= 2;
    }
  }
  for(int i = 0; i < num_weights; i ++ ) {
    network[i] = float_to_fixed(conv.parmas[i]);
    if (pool_size == 8)
      network[i] >>= 2;
  }
  for(int i = 0; i < num_bias; i ++ ) {
    bias[i] = float_to_fixed(conv.bias[i]);
  }
  int output_size = conv.out*(input.w)*(input.h);
  if (pool_size) {
    output_size /= pool_size * pool_size;
  }
  output = (short*)malloc(output_size*sizeof(short));
  conv_arg_t args = {num_input, num_bias, num_weights,
                     conv.in, conv.out, input.w, input.h, pool_size,
                     input_p, network, bias, output};
  float* n_in = (float*)malloc(output_size*sizeof(float));

  if (ioctl(memory_fd, CONV_WRITE, &args)) {
    perror("ioctl(CONV_WRITE) failed");
  }
  printf("Conv run ok!\n");
  printf("%d\n", output_size);
  for(int i = 0; i < output_size; i ++ ) {
    if(pool_size == 8)
      printf("%x\n", output[i]);
    n_in[i] = fixed_to_float(output[i]);
  }
  Conv_IO next_IO = {input.w, input.h, conv.out, n_in};
  if (pool_size != 0) {
    next_IO.w /= pool_size, next_IO.h /= pool_size;
  }
  return next_IO;
}

Fc_IO fc_input() {
  Fc_IO input;
  FILE *f = fopen("input", "r");
  int len = 0;
  printf("reading fc input");
  fscanf(f, "%d\n", &len);
  float *parma = (float*)malloc(sizeof(float)*len);
  input.parmas = parma;
  input.width = len;
  for(int i = 0; i < len; i ++ ) {
    fscanf(f, "%f", &parma[i]);
```

```
  }
  return input;
}

Fc_IO run_fc(Fc_IO input, Fc fc) {
  short *input_p, *network, *bias, *output;
  int num_input = input.width, num_bias = fc.out,
      num_weights = fc.in * fc.out;
  input_p = (short*)malloc(num_input*sizeof(short));
  network = (short*)malloc(num_weights*sizeof(short));
  bias = (short*)malloc(num_bias*sizeof(short));
  output = (short*)malloc(fc.out*sizeof(short));
  float* f =  (float*)malloc(fc.out*sizeof(float));

  for(int i = 0; i < num_input; i ++ ) {
    input_p[i] = float_to_fixed(input.parmas[i]);
  }
  for(int i = 0; i < num_weights; i ++ ) {
    network[i] = float_to_fixed(fc.parmas[i]);
  }
  for(int i = 0; i < num_bias; i ++ ) {
    bias[i] = float_to_fixed(fc.bias[i]);
  }
  conv_arg_t args = {num_input, num_bias, num_weights,
                     fc.in, fc.out, 0, 0, 0,
                     input_p, network, bias, output};
  if (ioctl(memory_fd, FC_WRITE, &args)) {
    perror("ioctl(CONV_WRITE) failed");
  }
  printf("FCINFO:%d %d\n", fc.in, fc.out);
  Fc_IO out = {fc.out, f};
  for(int i = 0; i < fc.out; i ++ ) {
  //  printf("%x\n", output[i]);
    f[i] = fixed_to_float(output[i]);
    if (f[i] < 0) f[i] = 0;
  }
  printf("fully connected layer ok\n");
  return out;
}

void compute_network(Network net, Conv_IO input) {
  Conv_IO input_2 = run_conv(input, net.con1, 0);
  Conv_IO input_3 = run_conv(input_2, net.con2, 2);
  Conv_IO input_4 = run_conv(input_3, net.con3, 2);
  for(int i = 0; i < 60; i ++ ) {
    printf("%f\n", input_4.parmas[i]);
  }
  Conv_IO input_5 = run_conv(input_4, net.con4, 2);

  Fc_IO fc = {input_5.d, input_5.parmas};

  Fc_IO fc1 = run_fc(fc, net.fc1);
  Fc_IO fc2 = run_fc(fc1, net.fc2);
  for(int i = 0; i < fc2.width; i ++ ) {
```

```
    printf("%f\n", fc2.parmas[i]);
  }
}

int main()
{
  mem_arg_t vla; int i;
  static const char filename[] = "/dev/memory";

  printf("Memory Userspace program started\n");

  if ( (memory_fd = open(filename, O_RDWR)) == -1) {
    fprintf(stderr, "could not open %s\n", filename);
    return -1;
  }
  graph_t g;
  Network net = ReadNetwork("./weights.txt");
  float *d = read_jpg("./b.jpg", &g);
  Conv_IO input = {g.width, g.height, 3, d};
//  for(int i = 0; i < 20; i ++ ) {
//    printf("%f\n", d[i]);
//  }

  //Conv_IO input = get_input();
  //compute_network(net, get_input());
  compute_network(net, input);


//  FILE *f;
//  f = fopen("weights", "r");

  //Conv conv;
  //conv = ReadConv(f);
  //run_conv(input, conv);

//  Fc_IO input = fc_input();
//  Fc fc = ReadFc(f);
//  run_fc(input, fc);
//  //printf("%d\n", conv.kw);

//  for(int i = 0; i < g.width * g.height*3; i ++ ) {
//    printf("%d %f\n",i, d[i]);
//  }

  printf("VGA BALL Userspace program terminating\n");
  return 0;
}
```

9.2.2 Network related functions

```
#pragma once
#ifndef NETWORK
#define NETWORK
#include <stdio.h>
```

```
#include <stdint.h>

typedef struct {
    int w;
    int h;
    int d;
    float * parmas;
} Conv_IO;

typedef  struct{
  int out;
  int in;
  int kh;
  int kw;
  float* parmas;
  float* bias;
} Conv;

typedef  struct{
  int out;
  int in;
  float* parmas;
  float* bias;
} Fc;

typedef struct {
  int width;
  float* parmas;
} Fc_IO;

typedef struct {
  Conv con1;
  Conv con2;
  Conv con3;
  Conv con4;
  Fc fc1;
  Fc fc2;
} Network;

Conv_IO CreateConv(int w, int h, int d);
Network ReadNetwork(char* filename);
Fc ReadFc(FILE* fp);
Conv ReadConv(FILE*f );
int16_t float_to_fixed(float input);
float fixed_to_float(int16_t input);
void LoadNetwork(Network nt);

#endif
#include "network.h"
#include <stdlib.h>

int16_t float_to_fixed(float input) {
    return (int16_t)(input * 128.0f);
}
```

```
float fixed_to_float(int16_t input) {
    return (float)input/128.0;
}

Conv_IO CreateConv(int w, int h, int d)
{
    Conv_IO conv;
    conv.w = w;
    conv.h = h;
    conv.d = d;

    float * parmas;
    parmas = (float*)malloc(d*w*h*sizeof(float));

    conv.parmas = parmas;
    return conv;
}

Conv ReadConv(FILE* fp)
{
  Conv conv;
  int out, in, kw, kh, bias;
  fscanf(fp, "%d%d%d%d", &out, &in, &kh, &kw);

  int len = kh*kw*out*in;
  conv.parmas = (float*)malloc(len*sizeof(float));
  for(int i = 0; i < len; i ++ )
    fscanf(fp, "%f", &conv.parmas[i]);

  fscanf(fp, "%d", &bias);
  printf("bias size:%d\n", bias);
  conv.bias = (float*)malloc(bias*sizeof(float));
  for(int i = 0; i < bias; i ++ )
    fscanf(fp, "%f", &conv.bias[i]);

  conv.out = out;
  conv.in = in;
  conv.kh = kh;
  conv.kw = kw;

  return conv;
}

void LoadConv(Conv conv, FILE *f) {
  for (int i = 0; i < conv.in*conv.out*9; i ++ ) {
    fprintf(f, "%x\n", float_to_fixed(conv.parmas[i]));
  }
  for (int i = 0; i < conv.out; i ++) {
    fprintf(f, "%x\n", float_to_fixed(conv.bias[i]));
  }
  fprintf(f, "\n\n");
}

void LoadFully(Fc fc, FILE *f) {
```

```
  for (int i = 0; i < fc.in * fc.out; i ++ ) {
    fprintf(f, "%x\n", float_to_fixed(fc.parmas[i]));
  }
  for(int i = 0; i < fc.out; i ++ ) {
    fprintf(f, "%x\n", float_to_fixed(fc.bias[i]));
  }
  fprintf(f, "\n\n");
}

void LoadNetwork(Network nt) {
  FILE *f = fopen("weight.txt", "w");
  LoadConv(nt.con1, f);
  LoadConv(nt.con2, f);
  LoadConv(nt.con3, f);
  LoadConv(nt.con4, f);
  LoadFully(nt.fc1, f);
  LoadFully(nt.fc2, f);
  fclose(f);
}

Fc ReadFc(FILE* fp)
{
  Fc fc;
  int out, in, bias;
  fscanf(fp, "%d%d", &out, &in);
  printf("Reading fully: %d %d\n", out, in);

  int len = out*in;
  fc.parmas = (float*)malloc(len*sizeof(float));
  for(int i = 0; i < len; i ++ )
    fscanf(fp, "%f", &fc.parmas[i]);

  fscanf(fp, "%d", &bias);
  fc.bias = (float*)malloc(bias*sizeof(float));
  for(int i = 0; i < bias; i ++ )
    fscanf(fp, "%f", &fc.bias[i]);
  fc.out = out;
  fc.in = in;

  return fc;
}

Network ReadNetwork(char* filename)
{
  Network nt;
  FILE *fp = fopen(filename, "r");
  nt.con1 = ReadConv(fp);
  nt.con2 = ReadConv(fp);
  nt.con3 = ReadConv(fp);
  nt.con4 = ReadConv(fp);
  nt.fc1 = ReadFc(fp);
  nt.fc2 = ReadFc(fp);

  return nt;
```

```
}
```

9.2.3 IOctl functions

```
#ifndef _MEMORY_H
#define _MEMORY_H

#include <linux/ioctl.h>

#define MEMORY_MAGIC 'M'

typedef struct {
  int pos;
  short value;
} mem_arg_t;

typedef struct {
  int num_input;
  int num_bias;
  int num_weights;
  int src_chan;
  int dst_chan;
  int cols;
  int rows;
  int pool_size;
  short *input_p;
  short *network;
  short *bias;
  short *output;
} conv_arg_t;

#define MEM_WRITE _IOW(MEMORY_MAGIC, 1, mem_arg_t)
#define CONV_WRITE _IOW(MEMORY_MAGIC, 3, conv_arg_t)
#define FC_WRITE _IOW(MEMORY_MAGIC, 4, conv_arg_t)
#define MEM_READ _IOR(MEMORY_MAGIC, 2, mem_arg_t)
#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "memory.h"

#define DRIVER_NAME "memory"

#define MEM_SIZE 163840
#define MEM(x, y) ((x)+(y)*2)
```

```
#define SRCCHAN(x) ((x)+(1<<19)*2)
#define DSTCHAN(x) ((x)+(1<<19)*2+2)
#define NUMCOL(x) ((x)+(1<<19)*2+4)
#define NUMROW(x) ((x)+(1<<19)*2+6)
#define FC(x)     ((x)+(1<<19)*2+8)
#define DOPOOL(x) ((x)+(1<<19)*2+10)
#define POOLSZ(x) ((x)+(1<<19)*2+12)
#define POOLSD(x) ((x)+(1<<19)*2+14)
#define DATASL(x) ((x)+(1<<19)*2+16)
#define DATASH(x) ((x)+(1<<19)*2+18)
#define WEITSL(x) ((x)+(1<<19)*2+20)
#define WEITSH(x) ((x)+(1<<19)*2+22)
#define BIASSL(x) ((x)+(1<<19)*2+24)
#define BIASSH(x) ((x)+(1<<19)*2+26)
#define OUTPSL(x) ((x)+(1<<19)*2+28)
#define OUTPSH(x) ((x)+(1<<19)*2+30)
#define START(x)  ((x)+(1<<19)*2+32)

struct memory_dev {
  struct resource res; /* Resource: our registers */
  void __iomem *virtbase; /* Where registers can be accessed in memory */
} dev;
short read_value(int pos) {
  return ioread16(MEM(dev.virtbase, pos));
}

void write_value(int pos, short value) {
  iowrite16(value, MEM(dev.virtbase, pos));
}

void write_fc_arg(conv_arg_t conv_arg) {
  int pos = 0, i;
  for (i = 0; i < conv_arg.num_input; i ++ ) {
    iowrite16(conv_arg.input_p[i], MEM(dev.virtbase, pos++));
  }
  for (i = 0; i < conv_arg.num_weights; i ++ ) {
    iowrite16(conv_arg.network[i], MEM(dev.virtbase, pos++));
  }
  for (i = 0; i < conv_arg.num_bias; i ++ ) {
    iowrite16(conv_arg.bias[i], MEM(dev.virtbase, pos++));
  }
  printk("here\n");
  iowrite16(conv_arg.src_chan, SRCCHAN(dev.virtbase));
  iowrite16(conv_arg.dst_chan, DSTCHAN(dev.virtbase));
  iowrite16(conv_arg.cols, NUMCOL(dev.virtbase));
  iowrite16(conv_arg.rows, NUMROW(dev.virtbase));
  int data_start = 0, weight_start = conv_arg.num_input,
      bias_start = weight_start + conv_arg.num_weights,
      output_start = bias_start + conv_arg.num_bias;
  iowrite16(1, FC(dev.virtbase));
  iowrite16(0, DOPOOL(dev.virtbase));
  iowrite16(0, POOLSZ(dev.virtbase));
  iowrite16(0, POOLSD(dev.virtbase));
  iowrite16((short)(data_start & 0xFFFF), DATASL(dev.virtbase));
```

```
  iowrite16((short)(data_start >> 16),    DATASH(dev.virtbase));
  iowrite16((short)(weight_start & 0xFFFF), WEITSL(dev.virtbase));
  iowrite16((short)(weight_start >> 16),    WEITSH(dev.virtbase));
  iowrite16((short)(bias_start & 0xFFFF),   BIASSL(dev.virtbase));
  iowrite16((short)(bias_start >> 16),      BIASSH(dev.virtbase));
  iowrite16((short)(output_start & 0xFFFF), OUTPSL(dev.virtbase));
  iowrite16((short)(output_start >> 16),    OUTPSH(dev.virtbase));
  iowrite16(1,              START(dev.virtbase));
}

void read_fc(conv_arg_t *conv_arg) {
  while(!ioread16(FC(dev.virtbase))) {
  }
  int output_size = conv_arg->dst_chan;
  printk("%d\n", output_size);
  conv_arg->output = (short*)kmalloc(sizeof(short)*output_size, GFP_KERNEL);
  int s = conv_arg->num_input + conv_arg->num_bias + conv_arg->num_weights;
  int i = 0;
  for(i = 0; i < output_size; i ++ ) {
    conv_arg->output[i] = ioread16(MEM(dev.virtbase, i+s));
  }
  for(i = 0; i < output_size; i ++ ) {
    //printk("%x\n", conv_arg->output[i]);
  }
  printk("finish\n");
}

void write_conv_arg(conv_arg_t conv_arg) {
  int pos = 0, i, j, k, cnt = 0;
  // d, w, h
  int d = conv_arg.src_chan, w = conv_arg.cols + 2, h = conv_arg.rows + 2;
  for(i = 0; i < d; i ++ ) {
    for(j = 0; j < w; j ++ ) {
      for (k = 0; k < h; k ++ ) {
        if (k == 0 || k == h - 1 || j == 0 || j == w - 1 ) {
          iowrite16(0, MEM(dev.virtbase, pos++));
        } else {
          iowrite16(conv_arg.input_p[cnt++], MEM(dev.virtbase, pos++));
        }
      }
    }
  }
  for (i = 0; i < conv_arg.num_weights; i ++ ) {
    iowrite16(conv_arg.network[i], MEM(dev.virtbase, pos++));
  }
  for (i = 0; i < conv_arg.num_bias; i ++ ) {
    iowrite16(conv_arg.bias[i], MEM(dev.virtbase, pos++));
  }
  printk("here\n");
  iowrite16(conv_arg.src_chan, SRCCHAN(dev.virtbase));
  iowrite16(conv_arg.dst_chan, DSTCHAN(dev.virtbase));
  iowrite16(conv_arg.cols+2, NUMCOL(dev.virtbase));
  iowrite16(conv_arg.rows+2, NUMROW(dev.virtbase));
  int data_start = 0, weight_start = d*w*h,
```

```
      bias_start = weight_start + conv_arg.num_weights,
      output_start = bias_start + conv_arg.num_bias;
  iowrite16(0, FC(dev.virtbase));
  if (conv_arg.pool_size != 0) {
    iowrite16(1, DOPOOL(dev.virtbase));
  } else {
    iowrite16(0, DOPOOL(dev.virtbase));
  }
  iowrite16(conv_arg.pool_size, POOLSZ(dev.virtbase));
  iowrite16(2, POOLSD(dev.virtbase));
  iowrite16((short)(data_start & 0xFFFF), DATASL(dev.virtbase));
  iowrite16((short)(data_start >> 16),    DATASH(dev.virtbase));
  iowrite16((short)(weight_start & 0xFFFF), WEITSL(dev.virtbase));
  iowrite16((short)(weight_start >> 16),    WEITSH(dev.virtbase));
  iowrite16((short)(bias_start & 0xFFFF),   BIASSL(dev.virtbase));
  iowrite16((short)(bias_start >> 16),      BIASSH(dev.virtbase));
  iowrite16((short)(output_start & 0xFFFF), OUTPSL(dev.virtbase));
  iowrite16((short)(output_start >> 16),    OUTPSH(dev.virtbase));
  iowrite16(1,              START(dev.virtbase));
}

void read_conv(conv_arg_t *conv_arg) {
  while(!ioread16(FC(dev.virtbase))) {
  }
  int output_size = conv_arg->dst_chan*(conv_arg->cols)*(conv_arg->rows);
  if (conv_arg->pool_size) {
    output_size /= conv_arg->pool_size * conv_arg->pool_size;
  }
    printk("poolsize:%d\n", conv_arg->pool_size);
  printk("%d\n", output_size);
  conv_arg->output = (short*)kmalloc(sizeof(short)*output_size, GFP_KERNEL);

  int d = conv_arg->src_chan, w = conv_arg->cols + 2, h = conv_arg->rows + 2;
  int s = d*w*h+ conv_arg->num_bias + conv_arg->num_weights;
  printk("%d\n", s);
  int i = 0;
  for(i = 0; i < output_size; i ++ ) {
    conv_arg->output[i] = ioread16(MEM(dev.virtbase, i+s));
  }

  printk("%x\n", output_size);
  if (conv_arg->pool_size==8)
  for(i = 0; i < output_size; i ++ ) {
    printk("%d %x\n", i, conv_arg->output[i]);
  }
  printk("finish\n");
}


/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
```

```
static long memory_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
  mem_arg_t vla;
  int i;
  conv_arg_t conv_arg;
  conv_arg_t arg2;

  switch (cmd) {
    case CONV_WRITE:
      if (copy_from_user(&conv_arg, (conv_arg_t *) arg, sizeof(conv_arg_t)))
        return -EACCES;
      arg2.input_p = (short*)kmalloc(sizeof(short)*conv_arg.num_input,
          GFP_KERNEL);
      arg2.network = (short*)kmalloc(sizeof(short)*conv_arg.num_weights,
          GFP_KERNEL);
      arg2.bias = (short*)kmalloc(sizeof(short)*conv_arg.num_bias, GFP_KERNEL)
          ;
      arg2.output = conv_arg.output;
      if (copy_from_user(arg2.input_p, conv_arg.input_p, sizeof(short)*
          conv_arg.num_input))
        return -EACCES;
      if (copy_from_user(arg2.network, conv_arg.network, sizeof(short)*
          conv_arg.num_weights))
        return -EACCES;
      if (copy_from_user(arg2.bias, conv_arg.bias, sizeof(short)*conv_arg.
          num_bias))
        return -EACCES;
      conv_arg.input_p = arg2.input_p;
      conv_arg.network = arg2.network;
      conv_arg.bias = arg2.bias;
      write_conv_arg(conv_arg);
      read_conv(&conv_arg);
      int output_size = conv_arg.dst_chan*(conv_arg.cols)*(conv_arg.rows);
      printk("%d %d %d", conv_arg.cols, conv_arg.rows, conv_arg.dst_chan);
      if (conv_arg.pool_size) {
        output_size /= conv_arg.pool_size * conv_arg.pool_size;
      }
      if (copy_to_user(arg2.output, conv_arg.output, sizeof(short)*output_size
          )) {
        return -EACCES;
      }
      break;

    case FC_WRITE:
      if (copy_from_user(&conv_arg, (conv_arg_t *) arg, sizeof(conv_arg_t)))
        return -EACCES;
        printk("%d\n", conv_arg.num_input);
      arg2.input_p = (short*)kmalloc(sizeof(short)*conv_arg.num_input,
          GFP_KERNEL);
      arg2.network = (short*)kmalloc(sizeof(short)*conv_arg.num_weights,
          GFP_KERNEL);
      arg2.bias = (short*)kmalloc(sizeof(short)*conv_arg.num_bias, GFP_KERNEL)
          ;
      arg2.output = conv_arg.output;
```

```
      if (copy_from_user(arg2.input_p, conv_arg.input_p, sizeof(short)*
          conv_arg.num_input))
        return -EACCES;
      if (copy_from_user(arg2.network, conv_arg.network, sizeof(short)*
          conv_arg.num_weights))
        return -EACCES;
      if (copy_from_user(arg2.bias, conv_arg.bias, sizeof(short)*conv_arg.
          num_bias))
        return -EACCES;
      conv_arg.input_p = arg2.input_p;
      conv_arg.network = arg2.network;
      conv_arg.bias = arg2.bias;
      write_fc_arg(conv_arg);
      read_fc(&conv_arg);
      printk("%d\n", conv_arg.dst_chan);
      if (copy_to_user(arg2.output, conv_arg.output, sizeof(short)*conv_arg.
          dst_chan)) {
        return -EACCES;
      }
      break;

    case MEM_READ:
      if (copy_from_user(&vla, (mem_arg_t *) arg, sizeof(mem_arg_t)))
        return -EACCES;
      vla.value = read_value(vla.pos);
      if (copy_to_user((mem_arg_t *) arg, &vla, sizeof(mem_arg_t)))
        return -EACCES;
      break;
    case MEM_WRITE:
      if (copy_from_user(&vla, (mem_arg_t *) arg, sizeof(mem_arg_t)))
        return -EACCES;
      write_value(vla.pos, vla.value);
      break;
  }
  return 0;
}
/* The operations our device knows how to do */
static const struct file_operations memory_fops = {
  .owner    = THIS_MODULE,
  .unlocked_ioctl = memory_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice memory_misc_device = {
  .minor    = MISC_DYNAMIC_MINOR,
  .name   = DRIVER_NAME,
  .fops   = &memory_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init memory_probe(struct platform_device *pdev)
```

```
{
  int ret;

  /* Register ourselves as a misc device: creates /dev/memory */
  ret = misc_register(&memory_misc_device);

  /* Get the address of our registers from the device tree */
  ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
  if (ret) {
    ret = -ENOENT;
    goto out_deregister;
  }

  /* Make sure we can use these registers */
  if (request_mem_region(dev.res.start, resource_size(&dev.res),
        DRIVER_NAME) == NULL) {
    ret = -EBUSY;
    goto out_deregister;
  }

  /* Arrange access to our registers */
  dev.virtbase = of_iomap(pdev->dev.of_node, 0);
  if (dev.virtbase == NULL) {
    ret = -ENOMEM;
    goto out_release_mem_region;
  }

  return 0;

out_release_mem_region:
  release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
  misc_deregister(&memory_misc_device);
  return ret;
}

/* Clean-up code: release resources */
static int memory_remove(struct platform_device *pdev)
{
  iounmap(dev.virtbase);
  release_mem_region(dev.res.start, resource_size(&dev.res));
  misc_deregister(&memory_misc_device);
  return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id memory_of_match[] = {
  { .compatible = "csee4840,cnn_interface-1.0" },
  {},
};
MODULE_DEVICE_TABLE(of, memory_of_match);
#endif
```

```
/* Information for registering ourselves as a "platform" driver */
static struct platform_driver memory_driver = {
  .driver = {
    .name = DRIVER_NAME,
    .owner  = THIS_MODULE,
    .of_match_table = of_match_ptr(memory_of_match),
  },
  .remove = __exit_p(memory_remove),
};

/* Called when the module is loaded: set things up */
static int __init memory_init(void)
{
  pr_info(DRIVER_NAME ": init\n");
  return platform_driver_probe(&memory_driver, memory_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit memory_exit(void)
{
  platform_driver_unregister(&memory_driver);
  pr_info(DRIVER_NAME ": exit\n");
}

module_init(memory_init);
module_exit(memory_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Xincheng Yu, Embedded System");
MODULE_DESCRIPTION("memory driver");

#endif
```

9.2.4 Functions handling pictures

```
#ifndef GRAPH
#define GRAPH
#include<stdio.h>
#include <jpeglib.h>

typedef struct {
  int width;
  int height;
  float *pixel;
} graph_t;

float *read_jpg(char* file_name, graph_t *graph);

#endif
#include "get_jgp.h"
#include <stdint.h>
#include <math.h>
#include <stdlib.h>
```

```
float* resample(int w, int h, float* f){
  int tw = 32, th = 32, ch = 3;
  int t = h * w;

  float dx = w / (float)tw, dy = h / (float)th;
  float *pic;
  pic = (float*)malloc(3*tw*th*sizeof(float));

  for(int i = 0; i < ch; i ++ ) {
    for(int j = 0; j < tw; j ++ ) {
      for(int k = 0; k < th; k ++ ) {
        float x = dx*j, y = dy*k, res = 0;

        int fx = floor((double)x), fy = floor((double)(y));

        float a = x-fx, b = y-fy;
        float lu = f[i*t+fy*h+fx], ld = f[i*t+(fy+1)*h+fx], ru = f[i*t+fy*h+fx
            +1], rd = f[i*t+(fy+1)*h+fx+1];
        pic[i*tw*th + k*th+j] = (1-a)*(1-b)*lu+a*(1-b)*ru+(1-a)*b*ld+a*b*rd;
      }
    }
  }
  return pic;
}

float* read_jpg(char* file_name, graph_t *graph)
{
  FILE *infile = fopen(file_name, "rb");
  struct jpeg_decompress_struct cinfo;
  struct jpeg_error_mgr jerr;

  cinfo.err = jpeg_std_error(&jerr);

  jpeg_create_decompress(&cinfo);
  jpeg_stdio_src(&cinfo, infile);
  int rc = jpeg_read_header(&cinfo, TRUE);  //read the jpeg header
  if (rc != 1) {
    printf("JPEG file read fail!!\n");
    exit(rc);
  }

  jpeg_start_decompress(&cinfo);
  int width = cinfo.output_width, cnt = 0;
  int height = cinfo.output_height;
  int channels = cinfo.output_components;

  float *f = (float*)malloc(sizeof(float)*width*height*channels);
  float *f_reordered = (float*)malloc(sizeof(float)*width*height*channels);

  graph->pixel = f_reordered;
  graph->height = height;
  graph->width = width;

  printf("JPEG image is %d x %d with %d color channels\n", width, height,
```

```
    channels);

  unsigned char *buf[1];
  buf[0] = (unsigned char*)malloc(sizeof(unsigned char)*width*cinfo.
      output_components);

  while (cinfo.output_scanline < cinfo.output_height) {
    jpeg_read_scanlines(&cinfo, &buf[0], 1);
    for(int i = 0; i < width*channels; i ++ ) {
      f[cnt++] = buf[0][i]/255.0;
    }
  }

  float mean[3] = {0.485, 0.456, 0.406};
  float std[3] = {0.229, 0.224, 0.2225};

  int pixel_id = 0;
  for(int i = 0; i < cnt; i ++ ) {
    int curr_chan = i % 3;
    f_reordered[pixel_id + curr_chan * height * width] = (f[i] - mean[
        curr_chan]) / std[curr_chan];
    if (curr_chan == 2) pixel_id++;
  }
  jpeg_finish_decompress(&cinfo);
  jpeg_destroy_decompress(&cinfo);
  fclose(infile);
  return f_reordered;
}
```

References

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