Systolic Array Based on FPGA

Hang Ye
Columbia University
hy2891@columbia.edu

Yu Jia
Columbia University
yj2839@columbia.edu

Sicheng Hua
Columbia University
sh4605@columbia.edu

Siyuan Li
Columbia University
sl5590@columbia.edu

Xuanmin Zheng
Columbia University
xz3372@columbia.edu

Contents

1 Project Overview

1.1 Background and Motivation

Convolutional engines such as systolic arrays form the computational backbone of many high-performance signal-processing and neural-network workloads by efficiently executing multiply–accumulate (MAC) operations. A 9×1 linear systolic array provides a compact architecture for 1D convolutions and dot-product kernels, demonstrating high throughput with minimal control complexity. In this project, we concentrate on the RTL design and hardware verification of a 9×1 PE array on an FPGA platform, establishing a core accelerator building block without pursuing a full application-level integration.

1.2 Objectives and Scope

The primary objective is to design, implement, and verify a parameterizable 9×1 systolic array of processing elements (PEs) on an FPGA, and to demonstrate cycle-accurate correctness via golden-model comparison. Specifically, we will:

- **Design the PE array:** chain 9 identical 8-bit MAC units in a linear systolic topology with local data forwarding.
- **Implement Avalon-HPS interface:** stream input vectors from the HPS into the array and retrieve results over the Avalon bus.
- **Verify functionality:** compare each output with a C reference golden model to assert bit-exact agreement.
- **Analyze performance:** report clock frequency, throughput (MACs/s), and resource utilization (LUTs, DSPs, BRAM) on the target FPGA.

1.3 Key Contributions

- **9×1 Systolic Array RTL:** a pipelined PE array supporting 8-bit inputs and 16-bit accumulation, optimized for local inter-PE timing.
- **Avalon-HPS Data Path:** lightweight bus-master logic to handle read/write transactions over the HPS–FPGA bridge.
- **Golden-Model Verification:** an automated testbench that injects randomized test vectors, captures outputs, and checks cycle-level conformance against a C reference.
- **Resource and Timing Report:** synthesis and place-and-route summaries showing >100 MHz operation with under 10% DSP and BRAM utilization.

2 Detailed Project Design

2.1 System Architecture

The system consists of a hardware accelerator interfaced with a Hard Processor System (HPS) via an Avalon bus. The HPS is responsible only for data and weight transfer, as well as reading back results. The control logic resides entirely within FPGA Control Unit. The architecture includes the following components:

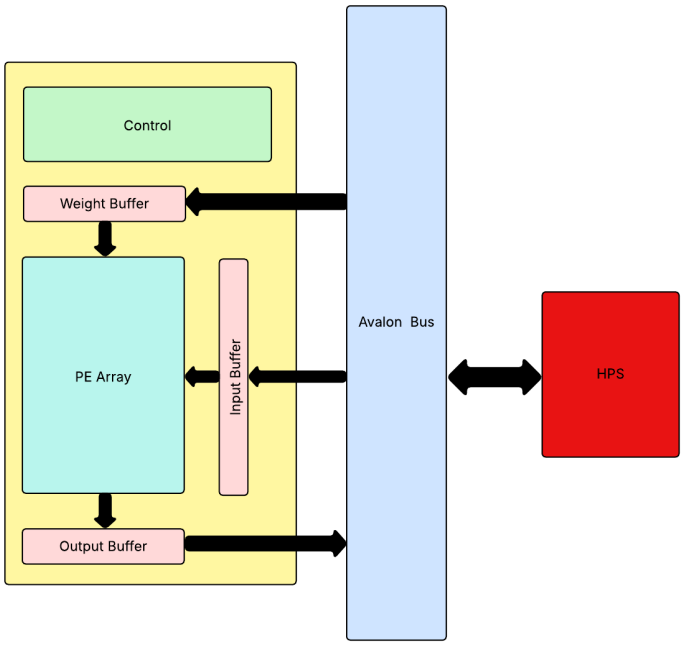

Figure 1: Block Diagram of System

2.1.1 HPS (Hard Processor System)

The HPS handles all data movement between the processor and the accelerator. Its responsibilities include:

- Transferring input feature data to the accelerator.
- Transferring neural network weights to the accelerator.
- Reading the final output results from the accelerator.

The HPS does not manage control flow or provide execution parameters.

2.1.2 Avalon Bus

The Avalon bus provides a standardized communication interface between the HPS and the accelerator. It facilitates the transfer of input data, weights, and output results.

2.1.3 FPGA

- **Control Unit:** The control module is the internal command unit of the accelerator. It issues all control signals for the computation process, including:
  - Managing the timing of weight and input loading.
  - Triggering the start of computation in the PE array.
  - Coordinating result write-back to the output buffer.
- **Weight Buffer:** This module stores weight data received from the HPS. The control module manages when and how these weights are sent to the PE array for computation.
- **Input Buffer:** The input buffer temporarily holds input features from the HPS. It feeds data into the PE array under the direction of the control module.

- **PE Array (Processing Element Array):** The PE array is the computational core of the accelerator. It performs parallel multiply-accumulate (MAC) operations and handles the main convolution workload.

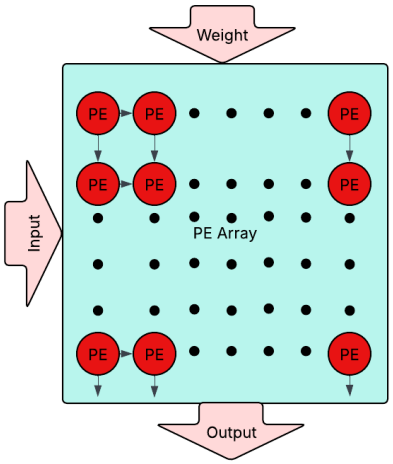

Figure 2: PE Array

- **Output Buffer:** This module holds the results produced by the PE array. The control module manages the output process, and results are later fetched by the HPS via the Avalon bus.

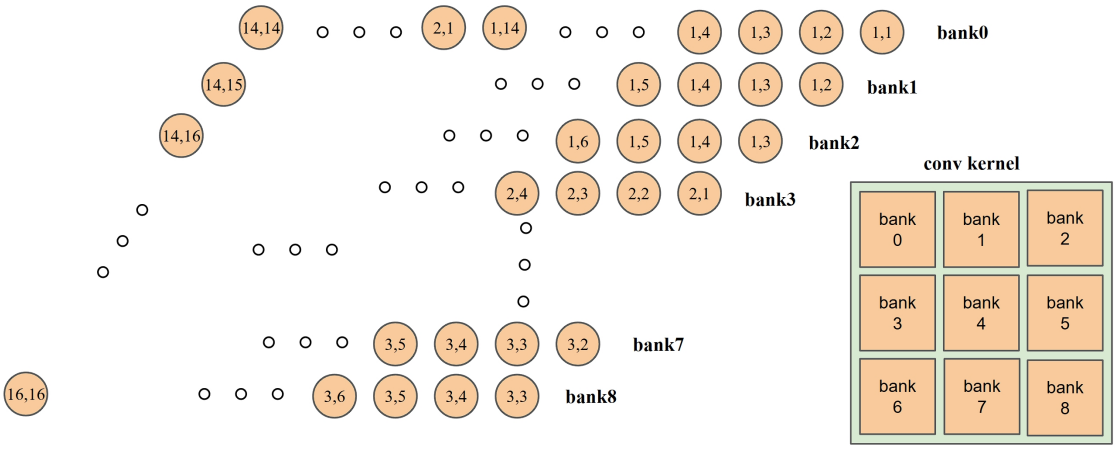

Figure 3: Input Banks Re-order

2.2 Data Flow and Control

Data flow, as shown in 2, just required on three aspects: Input, weight and output. The store on weight and output is quite easy, just enabling the store signal, and make the address plus one. However, the PE array require the input data to re-order, since the convolution has a data reuse on the input. The store logic is shown in the Fig.3, the number in the circle represent

the coordinate of the data in a 16*16 Grey-scale image, with the data size of 8 bits. Since a multiply is happen with 9 different operands, we made the banks to store the operands in the same frame on the corresponding position in sequences. When the computation starts, we can pick out data from different banks using a global address.

When doing a convolution multiply, the operand will capture a data from the global address in each banks, and make the global address plus one. So the input buffer provides a 72 bits out from 9 different banks, with a shared global address index. The advantages for this is that we can easly index the data we want to use in one cycle. However, it require with a size of 72*(14*14+9)*8 bits to store our input, nearly increase the input size 9 times.

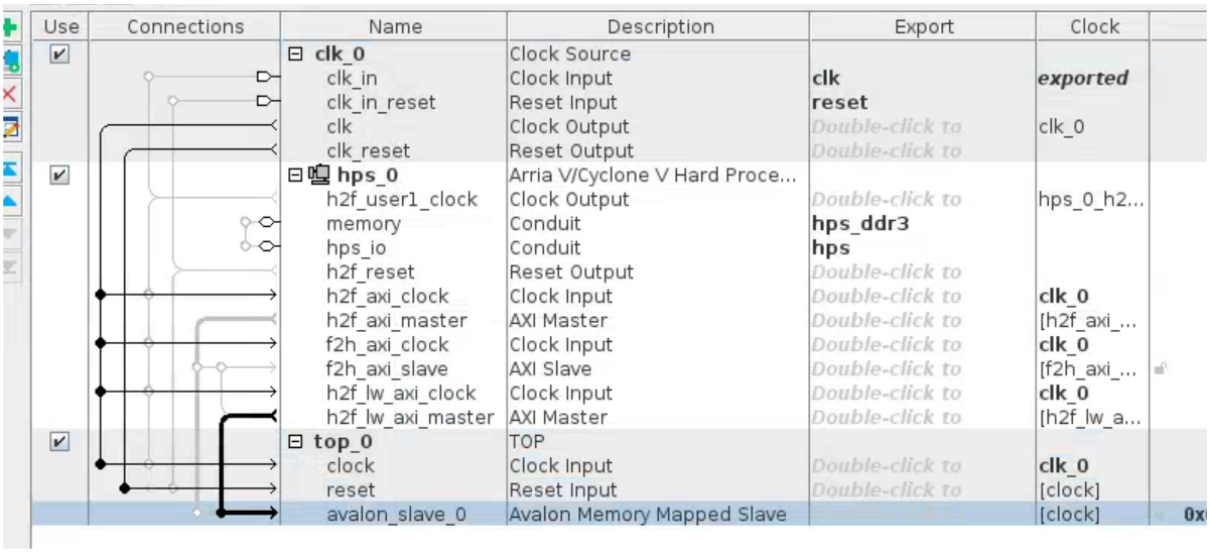

Figure 4: Hardware Implementation

2.3 Hardware Implementation

Our implementation on DE1SoC is as shown in Fig.4 .Thanks to the Avalon Bus, we can reach our hardware using the simple system call - *ioread* and *iowrite*. The section of on-board testing will go into the details about the communications. After the synthesize, we can see the results from tools.

Table 1: FPGA Resource Utilization

Resource	Used	Percentage
LUTs(in ALMs)	10,708	33%
DSP48	9	10%
BRAM(bits)	1,640	< 1 %
PINS	362	79%

Analyze: We checked the report generated by the tools, observed as Table.1:

- Fmax is 135Mhz, which makes the design possible to synthesize under clock_50. As default defined in the tcl files provided in lab3, the clock_50 cycles is 20ns, with the frequency of 50 MHz. Our design exceed the standard frequency, make it possible to implement.
- Ram usage is lower than 1%. After the checking on the rpt files, the RAM rpt shows that only the output buffer and weight buffer are recognized as the RAM, while the input

buffer did not. The reason for this may because I have the banks of buffer in the same module. If I can create a new submodule for the banks, and instance them in the buffer, the tool may be able to implement them as RAMs.

- LUT usage is close to 33%. The report shows the main part(60%) of LUTs are be used as the registers. That may be the implementation on these input buffers.
- DSP usage is 10%. It is very easy to understand. 9*1 PE has 9 multiply units.

2.4 Verification and Simulation

2.4.1 Simulation of Top

Figure 5 shows the ModelSim timing diagram of state transition from IDLE to LOAD_WEIGHT for the `ctrl` FSM. At time 0 ns, The FSM enters the `IDLE` state, with all control outputs (`weight_start`, `input_start`, `output_start`) held low. At time 8195 ns the design comes out of reset (rst_n deasserts).

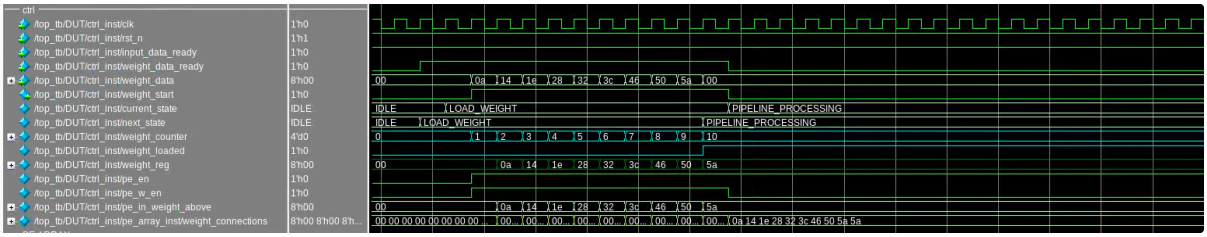

Figure 5: IDLE_TO_LOADWEIGHT Transition

IDLE → LOAD_WEIGHT The host first writes all nine weight values into the `weight_buffer` module. Inside `weight_buffer`, a local counter increments on each write, and when it reaches nine (i.e. all weights have been written), `weight_buffer` asserts `weight_data_ready` to the `ctrl` FSM. Upon detecting this pulse in the `IDLE` state, the FSM transitions to `LOAD_WEIGHT` on the next rising clock edge, at which point `weight_start`, `pe_en`, and `pe_w_en` are asserted to begin loading the weights into the PE array.

LOAD_WEIGHT Each incoming `weight_data_ready` handshake latches one new byte into `weight_reg` and increments `weight_counter`. When `weight_counter` reaches 9 (the parameter `MAX_WEIGHTS`), the internal flag `weight_loaded` is set. Note that after loading the ninth weight, `weight_start` deasserts but `pe_en` remains high to keep the PE array primed.

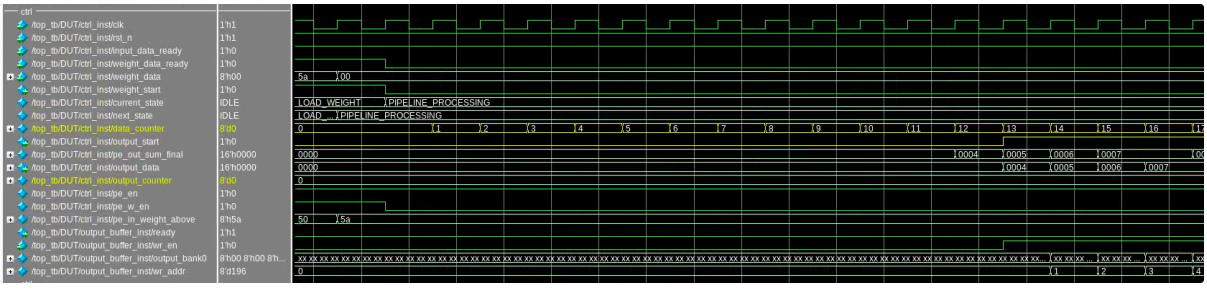

Figure 6: LOAD_WEIGHT TO PIPELINE_PROCESSING Transition

LOAD_WEIGHT → PIPELINE_PROCESSING On the very next cycle where both `weight_loaded` and `input_data_ready` are high, the FSM advances to `PIPELINE_PROCESSING`. In this state, `input_start` is asserted to feed new 72-bit words into the PE array, and the local counter `data_counter` begins incrementing each time `input_data_ready` pulses.

PIPELINE_PROCESSING The array performs one MAC per clock. After the pipeline latency (11 cycles) the first valid `pe_out_sum_final` appears on the bus, at which point `output_start` pulses to capture it. When `data_counter` reaches 205 (`MAX_DATA_COUNT`), the logic moves to `FINISH_PROCESSING`. In the `FINISH_PROCESSING` state, the FSM waits an additional two cycles to flush all in-flight results; as shown in Figure 8, once these two cycles complete, `output_start` is deasserted and the `output_buffer` internal counter reaches 196 (the total number of outputs), at which point `output_buffer_ready` is asserted to signal the software that the result buffer is full and ready for readback.

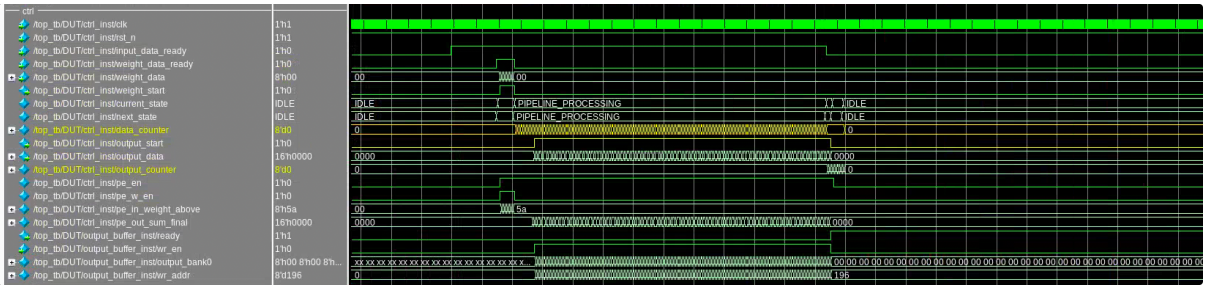

Figure 7: PIPELINE_PROCESSING Transition Overview

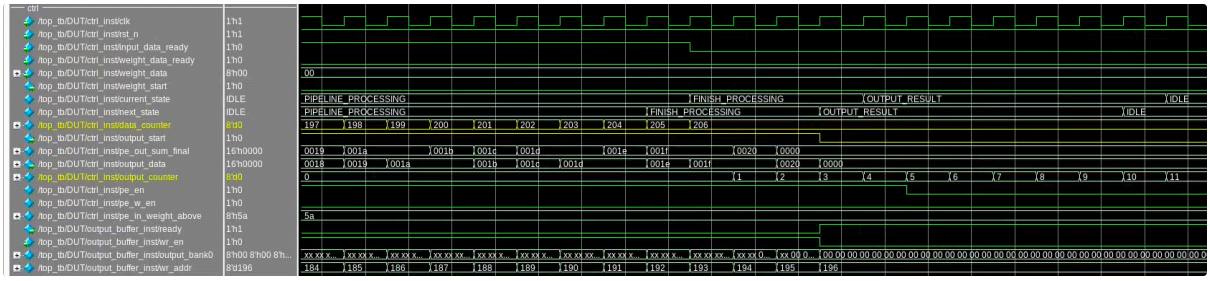

Figure 8: FINISH_PROCESSING Transition

FINISH_PROCESSING → OUTPUT_RESULT Once all in-flight data words have cleared the array (indicated by `processing_done`), the FSM enters `OUTPUT_RESULT`. Here `pe_en` deasserts, and `output_counter` provides a two-cycle delay to flush the final sums back to the host.

OUTPUT_RESULT → IDLE After waiting `OUTPUT_CYCLES` (10) cycles in `OUTPUT_RESULT`, the FSM returns to `IDLE`, ready for the next batch. All control signals are once again driven low, matching the waveform in Figure 8.

2.4.2 Simulation Verification

To verify the correctness of the design at the simulation level, several 16×16 black-and-white segmented images were used as test inputs. These images were carefully constructed to include distinct and predictable edge patterns. Such patterns make it easier to visually assess the system's edge detection capabilities.

The testbench (TB) was responsible for feeding input data, weights (for SobelX and SobelY filters), and collecting the output results from the systolic array. To ensure accurate verification and facilitate comparison, full-precision intermediate results were preserved during simulation. The final outputs were written to `.txt` files for post-processing.

For visualization and comparison, a Python script was used to:

- Load the simulation output from `.txt` files;
- Reconstruct the resulting edge-detected images;
- Generate reference results using NumPy-based Sobel implementations;
- Visually and numerically compare hardware outputs with the expected values.

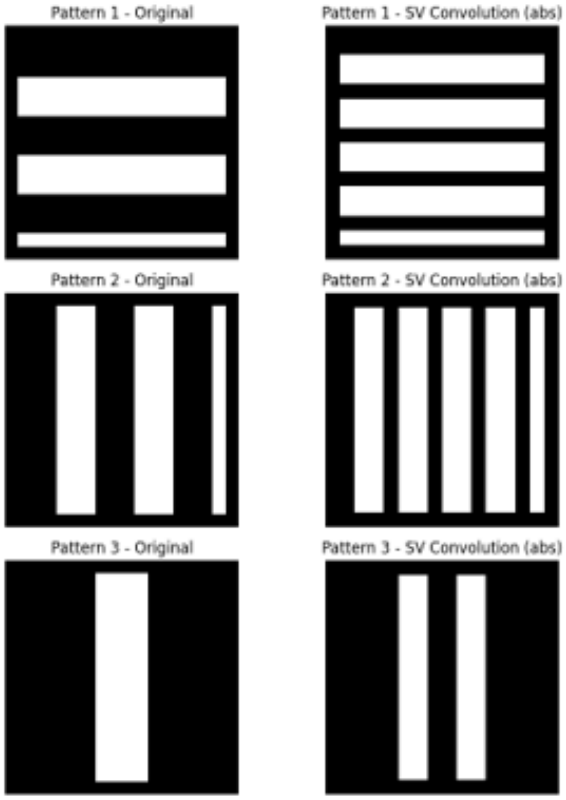

Figure 9: Pattern Comparison

From the generated output images, the effectiveness of the design was visually evident. All edges in the input images were successfully detected. In the resulting visualizations, white pixels clearly indicate detected edges, while black pixels represent non-edge areas. The sharp contrast of the output further confirmed the proper functionality of the Sobel filters implemented in hardware.

This strong visual clarity not only validates the correctness of the edge detection logic but also demonstrates the suitability of the testbench and verification methodology. The match between hardware results and reference software outputs reinforces confidence in the robustness of the design prior to hardware deployment.

2.5 Board-level Testing

Since it is a hardware & software co-design, the board-level test required a thoroughly understanding on hardware and software side. Each side structures are shown in Fig.10.

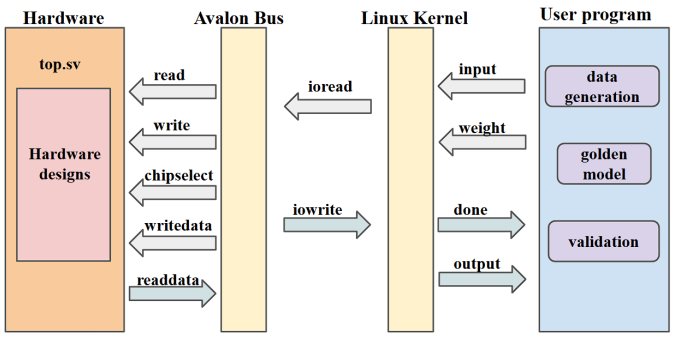

Figure 10: Communications between Software and Hardware

```
for (i = 0; i < 14*14; i++) {
   output_data[i] = read_output_data();
   //  output_data[i] = (int)(golden_data[i]/4);
    printf("  output_data[%d] = %d\n", i, output_data[i]);
    // Verify output_data
    if (golden_data[i] !=0){
    if ((abs(output_data[i] * 4 - golden_data[i]) / golden_data[i]) > 0.5) {
        fprintf(stderr, "Error: output_data[%d] = %d (expected %d)\n",
                i, output_data[i], golden_data[i]);
    }}
}
for (i = 0; i < 14; i++) {
    for (j = 0; j < 14; j++) {
        printf("  %d, ", output_data[i*14+j]);
        }
       printf("\n");
       }
```

Figure 11: Compare with golden model

- Software code will generate the input data, which located on the ddr memory on HPS side.
- Software code call the iowrite to write the code to the hardware register by address, as shown in Tab.2. The top module then pass the data into the correspoding buffers. When the data in buffers is enough for hardware, i.e. the buffer is full, the hardware will start its process automatically.
- Hardware will prepare the done signal in the register. Once receive the ioread, the hardware will push the done signal to the readdata ports. It is worth to mention that, a read latch is required for ioread, since the lasting time for read enable signal would be more than 1 cycle. We will only update the readdata when read is high and read in last cycle is low.
- Software waits until read out a valid done signal, and begin to read the output data. After a proper scaling, the software will compare the output data with golden model in Fig. 11.
- We passed the comparison on random input data, and edge picture detection.

Table 2: Address for software to index

ADDRESS	REGISTER
4'h0	image_size
4'h1	input_data
4'h2	weight_data
4'h3	done
4'h4	output_data

3 Team Contributions

Name	Contributions
Xuanmin Zheng	SystemVerilog Writing of PE Unit, PE Row, and PE Array modules. DFF design and debugging for timing correctness. Output Buffer development for efficient data flow management. Overall architecture design.
Hang Ye	Understand the systolic array in advance and share with group members. Design the re-order input buffer. Synthesize and implement on FPGA. Focus on interface, ports and communications, including hardware and software. Lesson: Never leave the synthesize until last seconds.
Yu Jia	The `ctrl` module was designed and implemented, encompassing all state-transition and control signals for weight loading, pipeline processing, and result output. A comprehensive ModelSim testbench was developed and executed to verify both the FSM and the top-level integration. End-to-end hardware testing covered `input_buffer`, `weight_buffer`, `PE_array`, and `output_buffer`; timing and functional issues in the control, input_buffer, weight_buffer, and top modules were identified and debugged, ensuring cycle-accurate and bit-accurate operation across the system.
Sicheng Hua	Responsibilities in this project included debugging and modification of the Processing Element (PE) module, verification of control unit timing and logic, and top-level system debugging. Additionally, simulation testbench development and test dataset creation for functional verification were carried out.
Siyuan Li	Responsible for developing testbenches and datasets for the control unit and top-level module. Also handled simulation output processing, result comparison against reference data, and implementation of Python-based verification scripts.

4 Source File Listings

List of all handwritten source files:

Table 3: Handwritten Source Files with Development Roles

File Name	**Description**	**Written By**	**Debugged By**
rtl/top.sv	Top-level integration wrapper	Hang Ye	Hang Ye, Yu jia
rtl/ctrl.sv	Control FSM module	Yu Jia	Yu Jia, Sicheng Hua
rtl/dffr.v	D-flip-flop register	Xuanmin Zheng	Xuanmin Zheng
rtl/input_buffer.sv	Input buffering module	Hang Ye	Hang Ye, Yu jia
rtl/output_buffer.sv	Output buffering module	Xuanmin Zheng	Hang Ye, Yu jia
rtl/PE_array.v	9×1 PE array	Xuanmin Zheng	Sicheng Hua, Yu Jia
rtl/PE_row.v	PE row wrapper	Xuanmin Zheng	Sicheng Hua, Yu Jia
rtl/PE.v	Single PE unit	Xuanmin Zheng	Sicheng Hua, Yu Jia
rtl/weight_buffer.sv	Weight buffering module	Sicheng Hua	Hang Ye, Yu jia
test/comparison.py	Golden-model comparison script	Siyuan Li	Siyuan Li
test/inputbuffer_PE_tb.sv	Input buffer testbench	Yu Jia	Yu Jia
test/rumsim_buffer_PE.do	ModelSim run script (buffer PE)	Yu Jia	Yu Jia
test/runsim_top.do	ModelSim run script (top level)	Yu Jia	Yu Jia
test/top_tb.sv	End-to-end testbench	Yu Jia	Yu Jia
test/top_tb1.sv	Data test testbench	Siyuan Li	Siyuan Li
test/wave_inputbuffer_PE.do	Waveform script (input buffer)	Yu Jia	Yu Jia
test/wave_top.do	Waveform script (top level)	Yu Jia	Yu Jia
main.c	User program	Hang Ye	Hang Ye
top.c	Kernel Program	Hang Ye	Hang Ye
top.h	Kernel Program header	Hang Ye	Hang Ye

5 Archive Contents for Build

Create a file named `myfiles` with the following entries and run:

```
# tar zcf project.tar.gz $(cat myfiles)

myfiles/                # Root directory of the project
|-- rtl/                # RTL source files
|   |-- ctrl.sv         # Control FSM module
|   |-- dffrv.v         # D-flip-flop register
|   |-- input_buffer.sv # Input buffering module
|   |-- output_buffer.sv# Output buffering module
|   |-- PE_array.v      # 9×1 PE array top level
|   |-- PE_row.v        # PE row wrapper
|   |-- PE.v            # Single PE unit
|   |-- top.sv          # Top-level integration wrapper
|   `-- weight_buffer.sv# Weight buffering module
|
|-- test/               # Simulation and testbench scripts
|   |-- comparison.py           # Golden-model comparison script
|   |-- inputbuffer_PE_tb.sv    # Testbench for input buffer
|   |-- rumsim_buffer_PE.do     # ModelSim run script for buffer PE
|   |-- runsim_top.do           # ModelSim run script for top level
|   |-- top_tb.sv               # End-to-end SystemVerilog testbench
|   |-- top_tb1.sv              # Alternative testbench
|   |-- wave_inputbuffer_PE.do  # Waveform script for input buffer
|   `-- wave_top.do             # Waveform script for top level
|
|-- main.c              # Host-FPGA interface application
|-- top.c               # Test harness main program
`-- top.h               # Test harness header file
```

5.1 Control FSM (rtl/ctrl.sv)

Listing 1: Control FSM module (‘rtl/ctrl.sv‘)

```
module ctrl(
    input  logic         clk,
    input  logic         rst_n,
    input  logic         input_data_ready,
    input  logic [71:0]  input_data,
    input  logic         weight_data_ready,
    input  logic [7:0]   weight_data,
    output logic         output_start,
    output logic         input_start,
    output logic         weight_start,
    output logic [15:0]  output_data
);


    typedef enum logic [2:0] {
        IDLE,
        LOAD_WEIGHT,
        PIPELINE_PROCESSING,
        FINISH_PROCESSING,
```

```
        OUTPUT_RESULT
    } state_t;

    state_t current_state, next_state;


    logic [7:0] weight_reg;
    logic [71:0] input_reg;
    logic [15:0] output_reg;

    logic unsigned [3:0] weight_counter;
    logic weight_loaded;
    logic processing_done;

    logic [7:0] data_counter;
    logic [7:0] output_counter;


    localparam MAX_WEIGHTS = 9;
    localparam MAX_DATA_COUNT = 205;
    localparam OUTPUT_CYCLES = 10;


    logic pe_en;
    logic pe_w_en;
    logic [71:0] pe_active_left;
    logic [7:0] pe_in_weight_above;
    logic [15:0] pe_out_sum_final;


    assign pe_active_left = input_reg;
    assign pe_in_weight_above = weight_reg;


    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            current_state <= IDLE;
        end else begin
            current_state <= next_state;
        end
    end


    always_comb begin
        next_state = current_state;

        case (current_state)
            IDLE: begin
                if (weight_data_ready) begin
                    next_state = LOAD_WEIGHT;
                end
            end

            LOAD_WEIGHT: begin
                if (weight_loaded && input_data_ready) begin
                    next_state = PIPELINE_PROCESSING;
                end
            end
```

```
78
79              PIPELINE_PROCESSING: begin
80                  if (data_counter >= MAX_DATA_COUNT) begin
81                      next_state = FINISH_PROCESSING;
82                  end
83              end
84
85              FINISH_PROCESSING: begin
86                  if (processing_done) begin
87                      next_state = OUTPUT_RESULT;
88                  end
89              end
90
91              OUTPUT_RESULT: begin
92                  if (output_counter >= OUTPUT_CYCLES) begin
93                      next_state = IDLE;
94                  end
95              end
96
97              default: next_state = IDLE;
98          endcase
99      end
100
101
102     always_ff @(posedge clk or negedge rst_n) begin
103         if (!rst_n) begin
104             weight_reg <= 8'b0;
105             input_reg <= 72'b0;
106             output_reg <= 16'b0;
107
108             weight_counter <= 4'b0;
109             weight_loaded <= 1'b0;
110             processing_done <= 1'b0;
111
112             data_counter <= 8'b0;
113             output_counter <= 8'b0;
114
115             output_start <= 1'b0;
116             input_start <= 1'b0;
117             weight_start <= 1'b0;
118             output_data <= 16'b0;
119
120
121             pe_en <= 1'b0;
122             pe_w_en <= 1'b0;
123         end else begin
124             case (current_state)
125                 IDLE: begin
126                     weight_counter <= 4'b0;
127                     weight_loaded <= 1'b0;
128                     processing_done <= 1'b0;
129
130                     data_counter <= 8'b0;
131                     output_counter <= 8'b0;
132
133                     output_start <= 1'b0;
134                     input_start <= 1'b0;
135                     weight_start <= 1'b0;
```

```
136
137                     pe_en <= 1'b0;
138                     pe_w_en <= 1'b0;
139                 end
140
141                 LOAD_WEIGHT: begin
142                     if (!weight_loaded) begin
143                         weight_start <= 1'b1;
144                         pe_en <= 1'b1;
145                         pe_w_en <= 1'b1;
146
147
148                         if (weight_data_ready) begin
149                             weight_reg <= weight_data;
150                             weight_counter <= weight_counter + 1'b1;
151
152                             if (weight_counter == (MAX_WEIGHTS)) begin
153                                 weight_loaded <= 1'b1;
154                             end
155                         end
156                     end else begin
157                         weight_start <= 1'b0;
158                         pe_en <= 1'b1;
159                         pe_w_en <= 1'b0;
160                     end
161                 end
162
163
164                 PIPELINE_PROCESSING: begin
165                     weight_start <= 1'b0;
166                     pe_w_en <= 1'b0;
167                     input_start <= 1'b1;
168                     pe_en <= 1'b1;
169
170
171                     if (input_data_ready) begin
172                         input_reg <= input_data;
173                         data_counter <= data_counter + 1'b1;
174                         output_reg <= pe_out_sum_final;
175
176                         if (data_counter > 11) begin
177                             output_start <= 1'b1;
178                             output_data <= pe_out_sum_final;
179                         end
180                     end
181                 end
182
183                 FINISH_PROCESSING: begin
184                     input_start <= 1'b0;
185                     output_data <= pe_out_sum_final;
186
187
188                     output_counter <= output_counter + 1'b1;
189
190                     if (output_counter >= 2) begin
191                         processing_done <= 1'b1;
192                         output_start <= 1'b0;
193                     end
```

```
                end

                OUTPUT_RESULT: begin

                    pe_en <= 1'b0;

                    output_counter <= output_counter + 1'b1;
                end

                default: begin

                end
            endcase
        end
    end


    PE_array #(
        .num1(9),
        .num2(1)
    ) pe_array_inst (
        .CLK(clk),
        .RESET(rst_n),
        .EN(pe_en),
        .W_EN(pe_w_en),
        .active_left(pe_active_left),
        .in_weight_above(pe_in_weight_above),
        .out_sum_final(pe_out_sum_final),
        .out_weight_final()
    );

endmodule
```

5.2 D-Flip-Flop Register (rtl/dffrv.v)

Listing 2: D-flip-flop register (‘rtl/dffrv.v‘)

```
module dffr #(
    parameter WIDTH = 1
)(
    input wire clk,
    input wire rst_n,
    input wire [WIDTH-1:0] d,
    output reg [WIDTH-1:0] q
);


    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            q <= {WIDTH{1'b0}};
        end else begin
            q <= d;
        end
    end

endmodule
```

5.3 Input Buffer (rtl/input_buffer.sv)

Listing 3: Input buffering module (‘rtl/input_buffer.sv‘)

```
‘define N 16
module input_buffer(
    input logic clk,
    input logic rst_n,
    input logic [7:0] data_in,
    input logic rd_en,
    input logic wr_en,
    output logic [71:0] data_out ,
    output logic ready,
    output logic done
);
    logic [7:0] data_in_bank0 [‘N*‘N+9];
    logic [7:0] data_in_bank1 [‘N*‘N+9];
    logic [7:0] data_in_bank2 [‘N*‘N+9];
    logic [7:0] data_in_bank3 [‘N*‘N+9];
    logic [7:0] data_in_bank4 [‘N*‘N+9];
    logic [7:0] data_in_bank5 [‘N*‘N+9];
    logic [7:0] data_in_bank6 [‘N*‘N+9];
    logic [7:0] data_in_bank7 [‘N*‘N+9];
    logic [7:0] data_in_bank8 [‘N*‘N+9];
    logic [$clog2(‘N*‘N)-1:0] index_wr;
    logic [$clog2(‘N*‘N +9)-1:0] index_rd;
    logic [$clog2(‘N*‘N+9)-1:0] index_bank0;
    logic [$clog2(‘N)-1 :0] counter;
    logic [$clog2(‘N*‘N+9)-1:0] index_bank1;
    logic [$clog2(‘N*‘N+9)-1:0] index_bank2;
    logic [$clog2(‘N*‘N+9)-1:0] index_bank3;
    logic [$clog2(‘N*‘N+9)-1:0] index_bank4;
    logic [$clog2(‘N*‘N+9)-1:0] index_bank5;
    logic [$clog2(‘N*‘N+9)-1:0] index_bank6;
    logic [$clog2(‘N*‘N+9)-1:0] index_bank7;
    logic [$clog2(‘N*‘N+9)-1:0] index_bank8;

    always_ff @(posedge clk) begin
        if (!rst_n) begin
            index_wr <= 0;
            counter <=0;
            ready <= 0;
        end else begin
            if (wr_en) begin
                if (index_wr == ‘N -1 ) begin
                    index_wr <= 0;
                    counter <= counter + 1;

                end else begin
                    index_wr <= index_wr + 1;
                    if (counter == ‘N -1 && index_wr == ‘N-2) begin
                        ready <= 1;
                    end
                    else begin
                        ready <= 0;
                    end
                    assert (counter < ‘N )
                        else $fatal("Assertion failed: counter (%0d) is
                            less than N (%0d)", counter+1, ‘N);
```

```
                end
            end
            else if (done) begin
                index_wr <= 0;
                counter <=0;
                ready <= 0;
            end
        end
    end
    always_ff @(posedge clk) begin
        if (!rst_n) begin
            index_rd <= 0;
            done <= 0;
        end else begin
            if (rd_en) begin
                index_rd <= index_rd + 1;
                if (index_rd == (`N-2)*(`N-2)+7) begin
                    done <= 1;
                end
            end
            else if (!ready) begin
                index_rd <= 0;
                done <= 0;

            end
        end
    end
    //bank 0
    always_ff @(posedge clk) begin
        if (!rst_n) begin
            index_bank0 <= 0;
            data_in_bank0[(`N-2)*(`N-2) + 1] <= 0;
            data_in_bank0[(`N-2)*(`N-2) + 2] <= 0;
            data_in_bank0[(`N-2)*(`N-2) + 3] <= 0;
            data_in_bank0[(`N-2)*(`N-2) + 4] <= 0;
            data_in_bank0[(`N-2)*(`N-2) + 5] <= 0;
            data_in_bank0[(`N-2)*(`N-2) + 6] <= 0;
            data_in_bank0[(`N-2)*(`N-2) + 7] <= 0;
            data_in_bank0[(`N-2)*(`N-2)] <= 0;
        end
        else begin
            if (wr_en) begin
                if (index_wr >= 0 && index_wr < `N-2 && counter >= 0 &&
                    counter < `N-2) begin
                    data_in_bank0[index_bank0] <= data_in;
                    index_bank0 <= index_bank0 + 1;
                end
            end
        end
    end
    //bank 1
    always_ff @(posedge clk) begin
        if (!rst_n) begin
            index_bank1 <= 1;
            data_in_bank1[0] <= 0;
            data_in_bank1[(`N-2)*(`N-2) + 2] <= 0;
            data_in_bank1[(`N-2)*(`N-2) + 3] <= 0;
            data_in_bank1[(`N-2)*(`N-2) + 4] <= 0;
```

```
112             data_in_bank1[(`N-2)*(`N-2) + 5] <= 0;
113             data_in_bank1[(`N-2)*(`N-2) + 6] <= 0;
114             data_in_bank1[(`N-2)*(`N-2) + 7] <= 0;
115             data_in_bank1[(`N-2)*(`N-2) + 1] <= 0;
116         end
117         else begin
118             if (wr_en) begin
119                 if (index_wr >= 1 && index_wr < `N-1 && counter >= 0 &&
                        counter < `N-2) begin
120                     data_in_bank1[index_bank1] <= data_in;
121                     index_bank1 <= index_bank1 + 1;
122                 end
123             end
124         end
125     end
126     //bank 2
127     always_ff @(posedge clk) begin
128         if (!rst_n) begin
129             index_bank2 <= 2;
130             data_in_bank2[0] <= 0;
131             data_in_bank2[1] <= 0;
132             data_in_bank2[(`N-2)*(`N-2) + 3] <= 0;
133             data_in_bank2[(`N-2)*(`N-2) + 4] <= 0;
134             data_in_bank2[(`N-2)*(`N-2) + 5] <= 0;
135             data_in_bank2[(`N-2)*(`N-2) + 6] <= 0;
136             data_in_bank2[(`N-2)*(`N-2) + 7] <= 0;
137             data_in_bank2[(`N-2)*(`N-2) + 2] <= 0;
138         end
139         else begin
140             if (wr_en) begin
141                 if (index_wr >= 2 && index_wr < `N && counter >= 0 &&
                        counter < `N-2) begin
142                     data_in_bank2[index_bank2] <= data_in;
143                     index_bank2 <= index_bank2 + 1;
144                 end
145             end
146         end
147     end
148     //bank 3
149     always_ff @(posedge clk) begin
150         if (!rst_n) begin
151             index_bank3 <= 3;
152             data_in_bank3[0] <= 0;
153             data_in_bank3[1] <= 0;
154             data_in_bank3[2] <= 0;
155             data_in_bank3[(`N-2)*(`N-2) + 4] <= 0;
156             data_in_bank3[(`N-2)*(`N-2) + 5] <= 0;
157             data_in_bank3[(`N-2)*(`N-2) + 6] <= 0;
158             data_in_bank3[(`N-2)*(`N-2) + 7] <= 0;
159             data_in_bank3[(`N-2)*(`N-2) + 3] <= 0;
160         end
161         else begin
162             if (wr_en) begin
163                 if (index_wr >= 0 && index_wr < `N-2 && counter >= 1 &&
                        counter < `N-1) begin
164                     data_in_bank3[index_bank3] <= data_in;
165                     index_bank3 <= index_bank3 + 1;
166                 end
```

```
            end
        end
    end
    //bank 4
    always_ff @(posedge clk) begin
        if (!rst_n) begin
            index_bank4 <= 4;
            data_in_bank4[0] <= 0;
            data_in_bank4[1] <= 0;
            data_in_bank4[2] <= 0;
            data_in_bank4[3] <= 0;
            data_in_bank4[(`N-2)*(`N-2) + 5] <= 0;
            data_in_bank4[(`N-2)*(`N-2) + 6] <= 0;
            data_in_bank4[(`N-2)*(`N-2) + 7] <= 0;
            data_in_bank4[(`N-2)*(`N-2) + 4] <= 0;
        end
        else begin
            if (wr_en) begin
                if (index_wr >= 1 && index_wr < `N-1 && counter >= 1 &&
                    counter < `N-1) begin
                    data_in_bank4[index_bank4] <= data_in;
                    index_bank4 <= index_bank4 + 1;
                end
            end
        end
    end
    //bank 5
    always_ff @(posedge clk) begin
        if (!rst_n) begin
            index_bank5 <= 5;
            data_in_bank5[0] <= 0;
            data_in_bank5[1] <= 0;
            data_in_bank5[2] <= 0;
            data_in_bank5[3] <= 0;
            data_in_bank5[4] <= 0;
            data_in_bank5[(`N-2)*(`N-2) + 6] <= 0;
            data_in_bank5[(`N-2)*(`N-2) + 7] <= 0;
            data_in_bank5[(`N-2)*(`N-2) + 5] <= 0;
        end
        else begin
            if (wr_en) begin
                if (index_wr >= 2 && index_wr < `N && counter >= 1 &&
                   counter < `N-1) begin
                    data_in_bank5[index_bank5] <= data_in;
                    index_bank5 <= index_bank5 + 1;
                end
            end
        end
    end
    //bank 6
    always_ff @(posedge clk) begin
        if (!rst_n) begin
            index_bank6 <= 6;
            data_in_bank6[0] <= 0;
            data_in_bank6[1] <= 0;
            data_in_bank6[2] <= 0;
            data_in_bank6[3] <= 0;
            data_in_bank6[4] <= 0;
```

```
223             data_in_bank6[5] <= 0;
224             data_in_bank6[(`N-2)*(`N-2) + 7] <= 0;
225             data_in_bank6[(`N-2)*(`N-2) + 6] <= 0;
226         end
227         else begin
228             if (wr_en) begin
229                 if (index_wr >= 0 && index_wr < `N-2 && counter >= 2 &&
                        counter < `N) begin
230                     data_in_bank6[index_bank6] <= data_in;
231                     index_bank6 <= index_bank6 + 1;
232                 end
233             end
234         end
235     end
236     //bank 7
237     always_ff @(posedge clk) begin
238         if (!rst_n) begin
239             index_bank7 <= 7;
240             data_in_bank7[0] <= 0;
241             data_in_bank7[1] <= 0;
242             data_in_bank7[2] <= 0;
243             data_in_bank7[3] <= 0;
244             data_in_bank7[4] <= 0;
245             data_in_bank7[5] <= 0;
246             data_in_bank7[6] <= 0;
247             data_in_bank7[(`N-2)*(`N-2) + 7] <= 0;
248         end
249         else begin
250             if (wr_en) begin
251                 if (index_wr >= 1 && index_wr < `N-1 && counter >= 2 &&
                        counter < `N) begin
252                     data_in_bank7[index_bank7] <= data_in;
253                     index_bank7 <= index_bank7 + 1;
254                 end
255             end
256         end
257     end
258     //bank 8
259     always_ff @(posedge clk) begin
260         if (!rst_n) begin
261             index_bank8 <= 8;
262             data_in_bank8[0] <= 0;
263             data_in_bank8[1] <= 0;
264             data_in_bank8[2] <= 0;
265             data_in_bank8[3] <= 0;
266             data_in_bank8[4] <= 0;
267             data_in_bank8[5] <= 0;
268             data_in_bank8[6] <= 0;
269             data_in_bank8[7] <= 0;
270         end
271         else begin
272             if (wr_en) begin
273                 if (index_wr >= 2 && index_wr < `N && counter >= 2 &&
                        counter < `N) begin
274                     data_in_bank8[index_bank8] <= data_in;
275                     index_bank8 <= index_bank8 + 1;
276                 end
277             end
```

```
        end
    end
    always_comb begin
        data_out = done ? '0 : {data_in_bank8[index_rd],
                                data_in_bank7[index_rd],
                                data_in_bank6[index_rd],
                                data_in_bank5[index_rd],
                                data_in_bank4[index_rd],
                                data_in_bank3[index_rd],
                                data_in_bank2[index_rd],
                                data_in_bank1[index_rd],
                                data_in_bank0[index_rd]};
    end
endmodule
```

5.4 Output Buffer (rtl/output_buffer.sv)

Listing 4: Output buffering module (‘rtl/output_buffer.sv‘)

```
`define OUTPUT_SIZE 14
module output_buffer(
    input logic clk,
    input logic rst_n,
    input logic [7:0] data_in,
    input logic wr_en,
    input logic rd_en,
    output logic [7:0] data_out,
    output logic ready,
    output logic done
);

    // Memory banks
    logic [7:0] output_bank0 [`OUTPUT_SIZE * `OUTPUT_SIZE-1:0];

    // Write and read address counters
    logic [$clog2(`OUTPUT_SIZE * `OUTPUT_SIZE)-1:0] wr_addr;
    logic [$clog2(`OUTPUT_SIZE * `OUTPUT_SIZE)-1:0] rd_addr;



    // Write logic
    always_ff @(posedge clk) begin
        if (!rst_n) begin
            wr_addr <= 0;
            ready <= 0;
        end else if (wr_en) begin
            output_bank0[wr_addr] <= data_in;
            wr_addr <= wr_addr + 1;
            if (wr_addr == `OUTPUT_SIZE * `OUTPUT_SIZE-1) begin
                ready <= 1;
            end
            else begin
                ready <= 0; // Reset ready when not writing
            end
        end
        else if (done) begin
            wr_addr <= 0;
```

```
            ready <= 0; // Reset ready when done
        end
    end

    // Read logic
    always_ff @(posedge clk ) begin
        if (!rst_n) begin
            rd_addr <= 0;
            done <= 0;
        end else if (rd_en) begin
            rd_addr <= rd_addr + 1;
            if (rd_addr == `OUTPUT_SIZE * `OUTPUT_SIZE-1) begin
                done <= 1; // Indicate that reading is done
            end else begin
                done <= 0; // Reset done when not reading
            end
        end if (!ready) begin
            rd_addr <= 0;
            done <= 0; // Reset done when not ready
        end
    end

    // Output logic
    always_comb begin
        if (rd_en && (rd_addr < `OUTPUT_SIZE*`OUTPUT_SIZE)) begin
            data_out = output_bank0[rd_addr];
        end else begin
            data_out = 8'b0; // Default value when not reading
        end
    end
endmodule
```

5.5 PE Array Top (rtl/PE_array.v)

Listing 5: 9×1 PE array top level (`rtl/PE_array.v`)

```

module PE_array #(

    parameter num1 = 9,

    parameter num2 = 1

)(

    // interface to system
    input wire CLK,                             // CLK = 200MHz
    input wire RESET,                           // RESET, Negedge is active
    input wire EN,                              // enable signal for the
       accelerator, high for active
    input wire W_EN,                            // enable weight to flow



    input wire [num1*8-1:0] active_left,
    input wire [num2*8-1:0] in_weight_above,
```

```
    output wire [num2*8-1:0] out_weight_final,
    output wire [num2*16-1:0] out_sum_final


`ifdef DEBUG

    , output wire [num1*8-1:0] active_right

`endif

);


    //

    wire [num2*8-1:0] weight_connections[num1:0];

    wire [num2*16-1:0] sum_connections[num1:0];


    //

    assign sum_connections[0] = {(num2*16){1'b0}};

    assign weight_connections[0] = in_weight_above;


    //

    assign out_sum_final = sum_connections[num1];

    assign out_weight_final = weight_connections[num1];


    //        PE_row

    genvar gi;

    generate

        for (gi = 0; gi < num1; gi = gi + 1) begin: label

            PE_row #(

                .num(num2)

            ) PE_row_unit (

                .CLK(CLK),

                .RESET(RESET),

                .EN(EN),
```

```
                .W_EN(W_EN),

                .active_left(active_left[(gi+1)*8-1:gi*8]),

                .in_sum(sum_connections[gi]),

                .out_sum(sum_connections[gi+1]),

                .in_weight_above(weight_connections[gi]),

                .out_weight_below(weight_connections[gi+1])

            `ifdef DEBUG

                , .out_active_right(active_right[(gi+1)*8-1:gi*8])

            `endif

            );

        end

    endgenerate


endmodule
```

5.6 PE Row Wrapper (rtl/PE_row.v)

Listing 6: PE row wrapper (‘rtl/PE_row.v‘)

```
module PE_row #(
    parameter num = 9
)(
    //
    input wire CLK,
    input wire RESET,
    input wire EN,
    input wire W_EN,

    //    P   E
    input wire signed [7:0] active_left,
    input wire [num*8-1:0] in_weight_above,
    output wire [num*8-1:0] out_weight_below,
    input wire [num*16-1:0] in_sum,
    output wire [num*16-1:0] out_sum

`ifdef DEBUG
    , output wire [num*8-1:0] out_active_right
`endif
);

    //
    wire [num*8-1:0] active_right;

```

```
`ifdef DEBUG
    assign out_active_right = active_right;
`endif

    genvar gi;
    generate
        for (gi = 0; gi < num; gi = gi + 1) begin: label
            if (gi == 0) begin
                PE PE_unit (
                    .CLK(CLK),
                    .RESET(RESET),
                    .EN(EN),
                    .W_EN(W_EN),
                    .active_left(active_left),
                    .active_right(active_right[8*(gi+1)-1:8*gi]),
                    .in_sum(in_sum[16*(gi+1)-1:16*gi]),
                    .out_sum(out_sum[16*(gi+1)-1:16*gi]),
                    .in_weight_above(in_weight_above[8*(gi+1)-1:8*gi]),
                    .out_weight_below(out_weight_below[8*(gi+1)-1:8*gi
                        ])
                );
            end else begin
                PE PE_unit (
                    .CLK(CLK),
                    .RESET(RESET),
                    .EN(EN),
                    .W_EN(W_EN),
                    .active_left(active_right[8*gi-1:8*(gi-1)]),
                    .active_right(active_right[8*(gi+1)-1:8*gi]),
                    .in_sum(in_sum[16*(gi+1)-1:16*gi]),
                    .out_sum(out_sum[16*(gi+1)-1:16*gi]),
                    .in_weight_above(in_weight_above[8*(gi+1)-1:8*gi]),
                    .out_weight_below(out_weight_below[8*(gi+1)-1:8*gi
                        ])
                );
            end
        end
    endgenerate

endmodule
```

5.7 Single PE Unit (rtl/PE.v)

Listing 7: Single PE unit (‘rtl/PE.v‘)

```
module PE(
    // interface to system
    input wire CLK,                             // CLK = 200MHz
    input wire RESET,                           // RESET, Negedge is active
    input wire EN,                              // enable signal for the
        accelerator, high for active
    input wire W_EN,                            // enable weight to flow
    // interface to PE row .....
    input wire signed [7:0] active_left,
    output reg signed [7:0] active_right,
    input wire signed [15:0] in_sum,
    output wire signed [15:0] out_sum,
```

```
    input wire signed [7:0] in_weight_above,
    output wire signed [7:0] out_weight_below
);

    //
    wire signed [7:0] weight_next, weight_q;
    wire signed [7:0] input_next, input_q;

    wire signed [15:0] sum_next, sum_q;

    wire signed [15:0] out_next, out_q;


    //
    assign input_next = EN ? active_left : input_q;

    //
    assign sum_next = EN ? in_sum : sum_q;

    // weight
    assign weight_next = (EN && W_EN) ? in_weight_above : weight_q;

    //
    wire signed [19:0] mul_result;
    assign mul_result = weight_q * input_q;

    //
    wire signed [19:0] add_result;
    assign add_result = mul_result + in_sum;

    //
    assign out_next = EN ? add_result[19:4] : out_q;

    // active_right (always)
    always @(posedge CLK) begin
        active_right <= input_q;
    end

    assign out_sum = out_q;
    assign out_weight_below = weight_q;

    //
    dffr #(.WIDTH(8)) input_reg (
        .clk(CLK),
        .rst_n(RESET),
        .d(input_next),
        .q(input_q)
    );

    dffr #(.WIDTH(8)) weight_reg (
        .clk(CLK),
        .rst_n(RESET),
        .d(weight_next),
        .q(weight_q)
    );

    dffr #(.WIDTH(16)) sum_reg (
        .clk(CLK),
```

```
        .rst_n(RESET),
        .d(sum_next),
        .q(sum_q)
    );

    dffr #(.WIDTH(16)) out_reg (
        .clk(CLK),
        .rst_n(RESET),
        .d(out_next),
        .q(out_q)
    );


endmodule
```

5.8 Top-level Wrapper (rtl/top.sv)

Listing 8: Top-level integration wrapper (‘rtl/top.sv‘)

```
‘define N 16        // input img size
‘define Col 8       // weight column
‘define Row 8       // weight row

// Input need to re-order in the buffer, so there would be no need to
// has the col and row.
// 16’h0000             :     img_size
// 16’h0001-16’h0046    :     weight_data
// 16’h0050-16’h2050    :     input_data


// Why we don’t use the addr map here?
// we don’t really need to store the data in the top, but guide the
   data
// to the correct buffer.
// 4’h0: img_size
// 4’h1: weight_data
// 4’h2: input_data
// 4’h3: output_ready
// 4’h4: output_data

module top(
    input logic clk,
    input logic reset,
    input logic [7:0] data_in,
    input logic write,
    input logic read,
    input logic [3:0] addr,
    input logic chipselect,
    output logic [7:0] data_out
);
    logic [7:0] img_size;
    logic [7:0] input_data;
    logic [7:0] weight_data;
    logic [7:0] output_data;
    logic input_ld_en;
    logic weight_ld_en;
    logic output_sw_en;
```

```
    logic [71:0] input_data_buffer;
    logic [7:0]  weight_data_buffer;
    logic [15:0] output_data_pe;
    logic input_ready;
    logic weight_ready;
    logic output_ready;
    logic input_done;
    logic weight_done;
    logic output_done;
    logic input_start;
    logic weight_start;
    logic output_start;
    // iowrite
    always_ff @(posedge clk) begin
        if (reset) begin
            img_size <= 16; // set default to 16
            input_data <= 0;
            weight_data <= 0;
            input_ld_en <= 0;
            weight_ld_en <= 0;
        end
        else begin
            if (chipselect && write) begin
                case (addr)
                    4'h0: begin
                        img_size <= data_in;
                        weight_ld_en <= 0;
                        input_ld_en <= 0;
                    end
                    4'h1: begin
                        weight_data <= data_in;
                        weight_ld_en <= 1;
                        input_ld_en <= 0;
                    end
                    4'h2: begin
                        input_data <= data_in;
                        input_ld_en <= 1;
                        weight_ld_en <= 0;
                    end
                    default: begin
                        weight_ld_en <= 0;
                        input_ld_en <= 0;
                    end
                endcase
            end
            else begin
                weight_ld_en <= 0;
                input_ld_en <= 0;
            end
        end
    end
    // ioread
    logic read_latched;
    logic read_prev;    // read signal in the previous clock cycle

    always_ff @(posedge clk) begin
        if (reset) begin
            read_latched <= 0;
```

```
            read_prev <= 0;
        end
        else begin
            // keep read latched until read is toggle
            if (chipselect && read && !read_prev) begin
                read_latched <= 1;
            end
            else begin
                read_latched <= 0;
            end
            read_prev <= read;
        end
    end
    always_ff @(posedge clk) begin
        if (reset) begin
            data_out <= 0;
            output_sw_en <= 0;
        end
        else begin
            if (chipselect && read && !read_prev) begin
                case (addr)
                    4'h3: begin
                        output_sw_en <= 0;
                        data_out <= output_ready;
                    end
                    4'h4: begin
                        output_sw_en <= 1;
                        data_out <= output_data;
                    end
                    default: begin
                        data_out <= 0;
                        output_sw_en <= 0;
                    end
                endcase
            end
            else begin
                output_sw_en <= 0;
            end
        end
    end

    // Instantiate the buffer
    input_buffer input_buffer_inst(
        .clk(clk),
        .rst_n(!reset),
        .data_in(input_data),
        .rd_en(input_start),
        .wr_en(input_ld_en),
        .data_out(input_data_buffer),
        .ready(input_ready),
        .done(input_done)
    );

    weight_buffer weight_buffer_inst(
        .clk(clk),
        .rst_n(!reset),
        .data_in(weight_data),
        .rd_en(weight_start),
```

```
        .wr_en(weight_ld_en),
        .data_out(weight_data_buffer),
        .ready(weight_ready),
        .done(weight_done)
    );
    output_buffer output_buffer_inst(
        .clk(clk),
        .rst_n(!reset),
        .data_in(output_data_pe[15:8]),
        .rd_en(output_sw_en),
        .wr_en(output_start),
        .data_out(output_data),
        .ready(output_ready),
        .done(output_done)
    );
    ctrl ctrl_inst(
        .clk(clk),
        .rst_n(!reset),
        .input_data_ready(input_ready),
        .input_data(input_data_buffer),
        .weight_data_ready(weight_ready),
        .weight_data(weight_data_buffer),
        .output_start(output_start),
        .input_start(input_start),
        .weight_start(weight_start),
        .output_data(output_data_pe)
    );

endmodule
```

5.9 Weight Buffer (rtl/weight_buffer.sv)

Listing 9: Weight buffering module (‘rtl/weight_buffer.sv‘)

```
‘define WEIGHT_DEPTH 9
module weight_buffer(
    input logic clk,
    input logic rst_n,
    input logic [7:0] data_in,
    input logic wr_en,
    input logic rd_en,
    output logic [7:0] data_out,
    output logic ready,
    output logic done
);

    // Memory banks
    logic [7:0] weight_bank0 [‘WEIGHT_DEPTH-1:0];

    // Write and read address counters
    logic [$clog2(‘WEIGHT_DEPTH)-1:0] wr_addr;
    logic [$clog2(‘WEIGHT_DEPTH)-1:0] rd_addr;



    // Write logic
    always_ff @(posedge clk) begin
```

```
        if (!rst_n) begin
            wr_addr <= 0;
            ready <= 0;
        end else if (wr_en) begin
            weight_bank0[wr_addr] <= data_in;
            wr_addr <= wr_addr + 1;
            if (wr_addr == `WEIGHT_DEPTH - 1) begin
                ready <= 1;
            end
            else begin
                ready <= 0; // Reset ready when not writing
            end
        end
        else if (done) begin
            wr_addr <= 0;
            ready <= 0; // Reset ready when done
        end
    end

    // Read logic
    always_ff @(posedge clk ) begin
        if (!rst_n) begin
            rd_addr <= 0;
            done <= 0;
        end else if (rd_en) begin
            rd_addr <= rd_addr + 1;
            if (rd_addr == `WEIGHT_DEPTH - 1) begin
                done <= 1; // Indicate that reading is done
            end else begin
                done <= 0; // Reset done when not reading
            end
        end
        else if (!ready) begin
            rd_addr <= 0;
            done <= 0; // Reset done when not ready
        end
    end

    // Output logic
    always_comb begin
        if (rd_en && (rd_addr < `WEIGHT_DEPTH)) begin
            data_out = weight_bank0[rd_addr];
        end else begin
            data_out = 8'b0; // Default value when not reading
        end
    end

endmodule
```

5.10 Golden-model Comparison Script (test/comparison.py)

Listing 10: Golden-model comparison script (‘test/comparison.py‘)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import convolve2d

```

```
IMG_SIZE = 16
SV_OUTPUT_SIZE = 14
INPUT_PATH = r"D:\output.txt.txt"

def generate_pattern(pattern_type):
    img = np.zeros((IMG_SIZE, IMG_SIZE), dtype=np.uint8)

    if pattern_type == 2:  #
        for i in range(IMG_SIZE):
            val = 127 if (i // 3) % 2 == 1 else 0
            img[i, :] = val
    elif pattern_type == 3:  #
        for j in range(IMG_SIZE):
            val = 127 if (j // 3) % 2 == 1 else 0
            img[:, j] = val
    elif pattern_type == 4:  #
        pattern_flat = [0,0,0,0,0,0,127,127,127,127,0,0,0,0,0,0] * 16
        img = np.array(pattern_flat, dtype=np.uint8).reshape((IMG_SIZE,
            IMG_SIZE))
    else:
        raise ValueError("Only pattern_type 2, 3, 4 are supported")
    return img

def load_sv_multiple_outputs(filepath, output_size):
    with open(filepath, 'r') as f:
        lines = f.readlines()
    blocks = []
    current_block = []
    for line in lines:
        line = line.strip()
        if not line or set(line) == {'/'}:
            if current_block and len(current_block) == output_size *
                output_size:
                array = np.array(current_block, dtype=np.uint16).astype
                    (np.int16)
                blocks.append(array.reshape((output_size, output_size))
                    )
            current_block = []
        else:
            values = [int(x) for x in line.split()]
            current_block.extend(values)
    if current_block and len(current_block) == output_size *
        output_size:
        array = np.array(current_block, dtype=np.uint16).astype(np.
            int16)
        blocks.append(array.reshape((output_size, output_size)))
    return blocks

def apply_sobel_fixedpoint(img, direction='x'):
    if direction == 'x':
        kernel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=
            np.int16)
    elif direction == 'y':
        kernel = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype=
            np.int16)
    else:
        raise ValueError("Direction must be 'x' or 'y'")
    img_int = img.astype(np.int16)
```

```
    result = convolve2d(img_int, kernel, mode='valid', boundary='symm')
    result = np.clip(result, -32768, 32767).astype(np.int16)
    return result

def normalize_to_uint8(img, use_abs=True):
    if use_abs:
        img = np.abs(img)
    img_norm = (img - img.min()) / np.ptp(img) * 255
    return np.clip(img_norm, 0, 255).astype(np.uint8)

def compare_results(sv, py):
    diff = sv.astype(np.int32) - py.astype(np.int32)
    abs_diff = np.abs(diff)
    return abs_diff

def visualize_all_comparisons(original_images, sv_conv_images,
   error_maps, pattern_ids):
    num_patterns = len(original_images)
    fig, axs = plt.subplots(num_patterns, 3, figsize=(12, 3 *
       num_patterns))
    for i in range(num_patterns):
        img = np.pad(original_images[i], pad_width=1, mode='constant',
           constant_values=0)
        conv = np.pad(sv_conv_images[i], pad_width=1, mode='constant',
           constant_values=0)
        err = np.pad(error_maps[i], pad_width=1, mode='constant',
           constant_values=0)

        axs[i][0].imshow(img, cmap='gray', vmin=0, vmax=127)
        axs[i][0].set_title(f'Pattern {pattern_ids[i]} - Original')
        axs[i][0].axis('off')

        axs[i][1].imshow(conv, cmap='gray', vmin=0, vmax=255)
        axs[i][1].set_title(f'Pattern {pattern_ids[i]} - SV Convolution
           ')
        axs[i][1].axis('off')

        axs[i][2].imshow(err, cmap='hot')
        axs[i][2].set_title(f'Pattern {pattern_ids[i]} - Error Map')
        axs[i][2].axis('off')

    plt.tight_layout()
    plt.savefig('compare_all_patterns.png')
    plt.show()

if __name__ == '__main__':
    sv_outputs = load_sv_multiple_outputs(INPUT_PATH, output_size=
       SV_OUTPUT_SIZE)

    pattern_ids = [1, 2, 3]
    pattern_types = [2, 3, 4]
    original_images = []
    sv_conv_images = []
    error_maps = []

    for idx in range(len(pattern_types)):
        img = generate_pattern(pattern_types[idx])
        direction = 'y' if pattern_types[idx] == 2 else 'x'
```

```
        py_conv = apply_sobel_fixedpoint(img, direction=direction)
        sv_raw = sv_outputs[idx]


        sv_img = normalize_to_uint8(sv_raw, use_abs=True)

        original_images.append(img)
        sv_conv_images.append(sv_img)


    visualize_all_comparisons(original_images, sv_conv_images,
       error_maps, pattern_ids)
```

5.11 Input Buffer Testbench (test/inputbuffer_PE_tb.sv)

Listing 11: Input buffer testbench (‘test/inputbuffer_PE_tb.sv‘)

```
`timescale 1ns/100ps
`define N 16

module testbench;
    // Test parameters
    localparam CLK_PERIOD = 10; // 10ns for 100MHz clock

    // System signals
    logic clk;
    logic rst_n;
    logic en;
    logic w_en;

    // Input buffer signals
    logic [7:0] data_in;
    logic rd_en;
    logic wr_en;
    logic [7:0] data_out [0:8];
    logic buffer_done;

    // PE array signals
    wire [9*8-1:0] active_left;
    wire [1*8-1:0] in_weight_above;
    wire [1*8-1:0] out_weight_final;
    wire [1*16-1:0] out_sum_final;

    // Test control signals
    int errors = 0;
    int total_tests = 0;

    // Internal storage to track input/output values
    logic [7:0] input_values [`N*`N];
    logic [1*16-1:0] output_values [(`N-2)*(`N-2)];
    int output_index = 0;

    // Connect input buffer outputs to PE array inputs
    genvar i;
    generate
        for (i = 0; i < 9; i++) begin
            assign active_left[(i+1)*8-1:i*8] = data_out[i];
```

```
        end
    endgenerate

    // Clock generation
    initial begin
        clk = 0;
        forever #(CLK_PERIOD/2) clk = ~clk;
    end

    // Use a constant weight for predictable results
    assign in_weight_above = 8'b01000000; //

    // Instantiate input buffer
    input_buffer input_buffer_inst (
        .clk(clk),
        .rst_n(rst_n),
        .data_in(data_in),
        .rd_en(rd_en),
        .wr_en(wr_en),
        .data_out(data_out),
        .done(buffer_done)
    );

    // Instantiate PE array
    PE_array #(
        .num1(9),
        .num2(1)
    ) pe_array_inst (
        .CLK(clk),
        .RESET(rst_n),
        .EN(en),
        .W_EN(w_en),
        .active_left(active_left),
        .in_weight_above(in_weight_above),
        .out_weight_final(out_weight_final),
        .out_sum_final(out_sum_final)
    );

    // Task to initialize the testbench
    task initialize();
        // Initialize signals
        rst_n = 1;
        en = 0;
        w_en = 0;
        rd_en = 0;
        wr_en = 0;
        data_in = 0;

        // Apply reset
        #(CLK_PERIOD*2);
        rst_n = 0;
        #(CLK_PERIOD*20);
        rst_n = 1;

        // Enable modules
        #(CLK_PERIOD);
        en = 1;
    endtask
```

```
99
100     // Task to generate test data
101     task generate_test_data();
102         // Initialize with some pattern for predictable testing
103         for (int i = 0; i < `N*`N; i++) begin
104             input_values[i] = (i % 256) + 1; // Values 1-16 repeating
105         end
106     endtask
107
108     // Task to write data to input buffer
109     task write_to_buffer();
110         @(posedge clk);
111         wr_en = 1;
112
113         $display("Starting to write data to input buffer");
114         for (int i = 0; i < `N*`N; i++) begin
115             data_in = input_values[i];
116             $display("Writing data[%0d] = %0d", i, data_in);
117             @(posedge clk);
118         end
119
120         wr_en = 0;
121         $display("Finished writing data to input buffer");
122     endtask
123
124     task load_weights();
125     $display("Loading weights into PE array");
126     @(posedge clk);
127     w_en = 1;
128
129     repeat (20) @(posedge clk);
130
131     w_en = 0;
132     $display("Weights loaded and fixed");
133     endtask
134
135     // Task to read from input buffer and process through PE array
136     task process_data();
137         // Wait a few cycles for data propagation
138         #(CLK_PERIOD*5);
139
140         $display("Starting to read and process data");
141         @(posedge clk);
142         rd_en = 1;
143         w_en = 0;
144         // Capture output values while reading
145         output_index = 0;
146
147         // Wait for processing to complete
148         while (!buffer_done) begin
149             @(posedge clk);
150             if (rd_en) begin
151                 // Store output for verification
152                 output_values[output_index] = out_sum_final;
153                 $display("Cycle %0d: Reading out_sum_final = %0d",
                        output_index, out_sum_final);
154                 output_index = output_index + 1;
155             end
```

```
156         end
157
158         // Continue a few more cycles to get final outputs
159         for (int i = 0; i < 10; i++) begin
160             @(posedge clk);
161             if (output_index < (`N-2)*(`N-2)) begin
162                 output_values[output_index] = out_sum_final;
163                 $display("Final cycle %0d: Reading out_sum_final = %0d"
                        , output_index, out_sum_final);
164                 output_index = output_index + 1;
165             end
166         end
167
168         rd_en = 0;
169         $display("Finished reading and processing data");
170     endtask
171
172     // Task to verify results
173     task verify_results();
174         logic [15:0] expected;
175         logic [15:0] actual;
176         int valid_outputs;
177
178         $display("\n===== VERIFICATION RESULTS =====");
179         valid_outputs = (`N-2)*(`N-2);
180         // Check only valid outputs (depends on your specific
            architecture)
181         for (int i = 0; i < valid_outputs; i++) begin
182             // Calculate expected output
183             // For this simple example, we're multiplying each input by
                the weight (5)
184             // In your real design, you'll need to implement the proper
                convolution calculation
185             expected = input_values[i] * 5; // Simplified for testing
186             actual = output_values[i];
187
188             total_tests++;
189
190             if (actual != expected) begin
191                 $display("ERROR: Output[%0d] = %0d, Expected = %0d", i,
                        actual, expected);
192                 errors++;
193             end else begin
194                 $display("PASS: Output[%0d] = %0d, Expected = %0d", i,
                        actual, expected);
195             end
196         end
197
198         // Print summary
199         $display("\n===== TEST SUMMARY =====");
200         $display("Total tests: %0d", total_tests);
201         $display("Passed: %0d", total_tests - errors);
202         $display("Failed: %0d", errors);
203
204         if (errors == 0) begin
205             $display("TEST PASSED: All outputs match expected values");
206         end else begin
```

```
207             $display("TEST FAILED: Some outputs did not match expected
                    values");
208         end
209     endtask
210
211     // Main test sequence
212     initial begin
213         // Setup waveform dumping for ModelSim
214         $dumpfile("waveform.vcd");
215         $dumpvars(0, testbench);
216
217         // Initialize testbench
218         initialize();
219
220         // Generate test data
221         generate_test_data();
222
223         load_weights();
224
225         // Write data to input buffer
226         write_to_buffer();
227
228         // Process data through PE array
229         process_data();
230
231         // Verify results
232         verify_results();
233
234         // End simulation
235         #(CLK_PERIOD*10);
236         $finish;
237     end
238
239     // Monitor signals for debugging
240     initial begin
241         $monitor("Time: %t, Buffer Done: %b, OutSum: %h, OutWeight: %h"
                ,
242                  $time, buffer_done, out_sum_final, out_weight_final);
243     end
244
245 endmodule
```

5.12 ModelSim Run Script for Buffer PE (test/rumsim_buffer_PE.do)

Listing 12: ModelSim run script for buffer PE (‘test/rumsim_buffer_PE.do‘)

```
1  ##################################################
2  #  Modelsim do file to run simuilation for Control
3  ##################################################
4  #Setup
5  vlib work
6  vmap work work
7
8  #Include Netlist and Testbench
9  vlog +acc -incr ../../rtl/dffr.v
10 vlog +acc -incr ../../rtl/PE_array.v
11 vlog +acc -incr ../../rtl/PE_row.v
```

```
vlog +acc -incr ../../rtl/PE.v
vlog +acc -incr ../../rtl/input_buffer.sv
vlog +acc -incr inputbuffer_PE_tb.sv

#Run Simulator
vsim -voptargs=+acc -t ps -lib work testbench


do wave_inputbuffer_PE.do


#
run -all
```

5.13 ModelSim Run Script for Top Level (test/runsim_top.do)

Listing 13: ModelSim run script for top level (‘test/runsim_top.do‘)

```
#####################################################
#  Modelsim do file to run simuilation for Control
#####################################################
#Setup
vlib work
vmap work work

#Include Netlist and Testbench
vlog +acc -incr ../../rtl/dffr.v
vlog +acc -incr ../../rtl/PE_array.v
vlog +acc -incr ../../rtl/PE_row.v
vlog +acc -incr ../../rtl/PE.v
vlog +acc -incr ../../rtl/input_buffer.sv
vlog +acc -incr ../../rtl/output_buffer.sv
vlog +acc -incr ../../rtl/weight_buffer.sv
vlog +acc -incr ../../rtl/top.sv
vlog +acc -incr ../../rtl/ctrl.sv
vlog +acc -incr top_tb.sv

#Run Simulator
vsim -voptargs=+acc -t ps -lib work top_tb


do wave_top.do


#
run -all
```

5.14 End-to-end Testbench (test/top_tb.sv)

Listing 14: End-to-end SystemVerilog testbench (‘test/top_tb.sv‘)

```
‘timescale 1ns/1ps
‘define N 16

module top_tb;

```

```
    // Top module signals
    logic clk;
    logic reset;
    logic [7:0] data_in;
    logic write;
    logic read;
    logic [3:0] addr;
    logic chipselect;
    logic [7:0] data_out;

    // Clock period parameter
    parameter CLK_PERIOD = 10; // 10ns = 100MHz

    // Test data parameters
    parameter IMG_SIZE = `N;       // 16x16 image
    parameter INPUT_SIZE = IMG_SIZE * IMG_SIZE;  // input size
    parameter WEIGHT_SIZE = 3 * 3; // 3x3 kernel
    parameter OUTPUT_SIZE = (IMG_SIZE - 3 + 1) * (IMG_SIZE - 3 + 1); //
        output size = 14x14 = 196

    // Test sequence parameter
    parameter TEST_SEQUENCE = 1;  // 0: weights first then data, 1:
       data first then weights
    parameter MAX_WAIT_CYCLES = 5000; // maximum wait cycles

    // Test data
    logic [7:0] test_input [0:INPUT_SIZE-1];
    logic [7:0] test_weight [0:WEIGHT_SIZE-1];
    logic [7:0] test_output [0:OUTPUT_SIZE-1];

    // Convolution results for software simulation (optional)
    logic [19:0] expected_output [0:OUTPUT_SIZE-1];
    logic [19:0] sum;

    // Test statistics
    int output_counter = 0;
    int cycle_counter = 0;
    bit test_passed = 1;

    // Instantiate top-level module
    top DUT (
        .clk(clk),
        .reset(reset),
        .data_in(data_in),
        .write(write),
        .read(read),
        .addr(addr),
        .chipselect(chipselect),
        .data_out(data_out)
    );

    // Clock generation
    initial begin
        clk = 0;
        forever #(CLK_PERIOD/2) clk = ~clk;
    end

    // Counter
```

```
62      always @(posedge clk) begin
63          if (reset)
64              cycle_counter <= 0;
65          else
66              cycle_counter <= cycle_counter + 1;
67      end
68
69      // Initialize test data
70      initial begin
71          // Initialize input image - using (i*2 + j + 1) % 256 pattern
72          for (int i = 0; i < IMG_SIZE; i++) begin
73              for (int j = 0; j < IMG_SIZE; j++) begin
74                  test_input[i*IMG_SIZE + j] = (i*2 + j + 1) % 256;
75              end
76          end
77
78          // Initialize weights - using 10*(i+1) pattern (weights:
               10,20,30,...,90)
79          for (int i = 0; i < WEIGHT_SIZE; i++) begin
80              test_weight[i] = 10*(i+1);
81          end
82
83          // Initialize output array
84          for (int i = 0; i < OUTPUT_SIZE; i++) begin
85              test_output[i] = 0;
86              expected_output[i] = 0;
87          end
88
89          // Pre-calculate expected output (software simulated
               convolution)
90          for (int y = 0; y < IMG_SIZE-2; y++) begin
91              for (int x = 0; x < IMG_SIZE-2; x++) begin
92                  sum = 0;
93                  for (int ky = 0; ky < 3; ky++) begin
94                      for (int kx = 0; kx < 3; kx++) begin
95                          sum += test_input[(y+ky)*IMG_SIZE + (x+kx)] *
                               test_weight[ky*3 + kx];
96                      end
97                  end
98                  expected_output[y*(IMG_SIZE-2) + x] = sum;
99              end
100         end
101     end
102
103
104     //*********************************task
           ****************************//
105     // Task: write data
106     task write_data(input [3:0] address, input [7:0] data);
107         @(posedge clk);
108         addr = address;
109         data_in = data;
110         write = 1;
111         chipselect = 1;
112         @(posedge clk);
113         write = 0;
114         chipselect = 0;
115         @(posedge clk); // extra clock cycle
```

```
116     endtask
117
118     // Task: read data
119     task read_data(input [3:0] address, output [7:0] data);
120         @(posedge clk);
121         addr = address;
122         read = 1;
123         chipselect = 1;
124         @(posedge clk);
125         @(posedge clk); // wait for data to stabilize
126         data = data_out;
127         read = 0;
128         chipselect = 0;
129         @(posedge clk); // extra clock cycle
130     endtask
131
132     // Task: wait for completion
133     task wait_for_completion(input int max_cycles);
134         logic [7:0] ready_status;
135         int wait_count;
136
137         do begin
138             wait_count = 0;
139             read_data(4'h3, ready_status);
140             wait_count++;
141
142             if (wait_count % 50 == 0) begin
143                 $display("Waiting for output ready... (%0d checks)",
                        wait_count);
144             end
145
146             if (wait_count >= max_cycles) begin
147                 $display("Error: wait timed out! Checked %0d times",
                        wait_count);
148                 test_passed = 0;
149                 break;
150             end
151         end while (ready_status != 8'd1);
152
153         $display("Output ready! After %0d checks, %0d clock cycles",
                wait_count, cycle_counter);
154     endtask
155
156     // Task: verify output
157     task verify_output();
158         int mismatch_count;
159
160         begin
161             mismatch_count = 0;
162             for (int i = 0; i < OUTPUT_SIZE; i++) begin
163                 // Simple check: ensure output is not zero repeatedly
164                 if (i > 0 && test_output[i] == 0 && test_output[i-1] ==
                        0) begin
165                     mismatch_count++;
166                     if (mismatch_count <= 5) begin
167                         $display("Warning: output[%0d] and output[%0d]
                                are both zero", i-1, i);
168                     end
```

```
                end
            end
        end

        if (mismatch_count > OUTPUT_SIZE / 2) begin
            $display("Error: more than half of outputs are zero or
                abnormal pattern!");
            test_passed = 0;
        end else if (mismatch_count > 0) begin
            $display("Warning: found %0d suspicious output patterns",
                mismatch_count);
        end else begin
            $display("Output format verification passed!");
        end

        // Print some sample data for manual check
        $display("Sample output data:");
        for (int i = 0; i < 5; i++) begin
            for (int j = 0; j < 5; j++) begin
                $write("%4d ", test_output[i*(IMG_SIZE-2) + j]);
            end
            $write("\n");
        end
    endtask


    //*********************************main test
        ***************************//
    // Main testbench procedure
    initial begin
        // Initialize signals
        reset = 1;
        write = 0;
        read = 0;
        addr = 0;
        data_in = 0;
        chipselect = 0;

        // Reset
        #(CLK_PERIOD*50);
        reset = 0;
        #(CLK_PERIOD*2);

        // Select load sequence based on TEST_SEQUENCE
        if (TEST_SEQUENCE == 0) begin
            // Sequence 1: weights first, then input data
            $display("\nTest Sequence 1: Load weights first, then input
                data");

            // Step 1: set image size
            $display("Step 1: Set image size to %0d x %0d", IMG_SIZE,
                IMG_SIZE);
            write_data(4'h0, IMG_SIZE);

            // Step 2: load weight data
            $display("Step 2: Load weight data (3x3 = 9 weights)");
            for (int i = 0; i < WEIGHT_SIZE; i++) begin
                write_data(4'h1, test_weight[i]);
```

```
222                 $display("  Loaded weight[%0d] = %0d", i, test_weight[i
                        ]);
223             end
224
225             // Step 3: load input image data
226             $display("Step 3: Load input image data (%0d pixels)",
                    INPUT_SIZE);
227             for (int i = 0; i < IMG_SIZE; i++) begin
228                 for (int j = 0; j < IMG_SIZE; j++) begin
229                     write_data(4'h2, test_input[i*IMG_SIZE + j]);
230                     if ((i*IMG_SIZE + j) % IMG_SIZE == 0)
231                         $display("  Loading image row %0d/%0d", i+1,
                                IMG_SIZE);
232                 end
233             end
234         end else begin
235             // Sequence 2: input data first, then weights
236             $display("\nTest Sequence 2: Load input data first, then
                    weights");
237
238             // Step 1: set image size
239             $display("Step 1: Set image size to %0d x %0d", IMG_SIZE,
                    IMG_SIZE);
240             write_data(4'h0, IMG_SIZE);
241
242             // Step 2: load input image data
243             $display("Step 2: Load input image data (%0d pixels)",
                    INPUT_SIZE);
244             for (int i = 0; i < IMG_SIZE; i++) begin
245                 for (int j = 0; j < IMG_SIZE; j++) begin
246                     write_data(4'h2, test_input[i*IMG_SIZE + j]);
247                     if ((i*IMG_SIZE + j) % IMG_SIZE == 0)
248                         $display("  Loading image row %0d/%0d", i+1,
                                IMG_SIZE);
249                 end
250             end
251
252             // Step 3: load weight data
253             $display("Step 3: Load weight data (3x3 = 9 weights)");
254             for (int i = 0; i < WEIGHT_SIZE; i++) begin
255                 write_data(4'h1, test_weight[i]);
256                 $display("  Loaded weight[%0d] = %0d", i, test_weight[i
                        ]);
257             end
258         end
259
260         // Step 4: waiting for convolution processing to complete
261         $display("Step 4: Waiting for convolution processing to
                complete");
262         wait_for_completion(MAX_WAIT_CYCLES);
263
264         // Step 5: read output data
265         $display("Step 5: Read output data (%0d values)", OUTPUT_SIZE);
266         for (int i = 0; i < OUTPUT_SIZE; i++) begin
267             read_data(4'h4, test_output[i]);
268             output_counter++;
269
270             if (i % (IMG_SIZE-2) == 0 && i > 0)
```

```
                $display("  Reading output row %0d/%0d", i/(IMG_SIZE-2)
                    , (IMG_SIZE-2));
        end

        // Step 6: verify output results
        $display("Step 6: Verify output results");
        verify_output();

        // Test summary
        $display("\nTest Summary:");
        $display("  Number of input pixels processed: %0d", INPUT_SIZE)
            ;
        $display("  Number of weights used: %0d", WEIGHT_SIZE);
        $display("  Number of outputs generated: %0d", OUTPUT_SIZE);
        $display("  Number of outputs read: %0d", output_counter);
        $display("  Total number of cycles executed: %0d",
            cycle_counter);

        if (test_passed)
            $display("\nTest completed: PASSED!");
        else
            $display("\nTest completed: FAILED! Please check the above
                error messages.");

        #(CLK_PERIOD*10);
        $finish;
    end

    // Monitor important signals (optional for debugging)
    initial begin
        $timeformat(-9, 2, " ns", 20);
    end

endmodule
```

5.15 Alternative Testbench (test/top_tb1.sv)

Listing 15: Alternative testbench (‘test/top_tb1.sv‘)

```
‘timescale 1ns/1ps
‘define N 16
‘define TEST_KERNEL_V
‘define PATTERN_HORI_STRIPE

module top_tb;

    // Top module signals
    logic clk;
    logic reset;
    logic [7:0] data_in;
    logic write;
    logic read;
    logic [3:0] addr;
    logic chipselect;
    logic [7:0] data_out;

    // Clock period parameter
```

```
19      parameter CLK_PERIOD = 10; // 10ns = 100MHz
20
21      // Test data parameters
22      parameter IMG_SIZE = `N;        // 16x16 image
23      parameter INPUT_SIZE = IMG_SIZE * IMG_SIZE;  // input size
24      parameter WEIGHT_SIZE = 3 * 3; // 3x3 kernel
25      parameter OUTPUT_SIZE = (IMG_SIZE - 3 + 1) * (IMG_SIZE - 3 + 1); //
            output size = 14x14 = 196
26
27      // Test sequence parameter
28      parameter TEST_SEQUENCE = 0;  // 0: weights first then data, 1:
            data first then weights
29      parameter MAX_WAIT_CYCLES = 300; // maximum wait cycles
30
31      // Test data
32      logic [7:0] test_input [0:INPUT_SIZE-1];
33      logic signed [7:0] test_weight [0:WEIGHT_SIZE-1];
34      logic [7:0] test_output [0:OUTPUT_SIZE-1];
35
36      // Convolution results for software simulation (optional)
37      logic [19:0] expected_output [0:OUTPUT_SIZE-1];
38      logic [19:0] sum;
39
40      // Test statistics
41      int output_counter = 0;
42      int cycle_counter = 0;
43      bit test_passed = 1;
44
45      //
46
47
48      // Output capture to txt
49        integer fout;
50        int col_counter = 0;
51        bit output_active = 0;
52        logic prev_output_start;
53
54
55      // Instantiate top-level module
56      top DUT (
57          .clk(clk),
58          .reset(reset),
59          .data_in(data_in),
60          .write(write),
61          .read(read),
62          .addr(addr),
63          .chipselect(chipselect),
64          .data_out(data_out)
65      );
66
67      // Clock generation
68      initial begin
69          clk = 0;
70          forever #(CLK_PERIOD/2) clk = ~clk;
71      end
72
73      // Counter
74      always @(posedge clk) begin
```

```
        if (reset)
            cycle_counter <= 0;
        else
            cycle_counter <= cycle_counter + 1;
    end


    // Pattern selection
    initial begin
        `ifdef PATTERN_HORI_EDGE
            for (int i = 0; i < IMG_SIZE; i++) begin
                for (int j = 0; j < IMG_SIZE; j++) begin
                    if (i < IMG_SIZE/4 || i >= 3*IMG_SIZE/4)
                        test_input[i*IMG_SIZE + j] = 8'd0;
                    else
                        test_input[i*IMG_SIZE + j] = 8'd127;
                end
            end
        `elsif PATTERN_VERT_EDGE
            for (int i = 0; i < IMG_SIZE; i++) begin
                for (int j = 0; j < IMG_SIZE; j++) begin
                    if (j < IMG_SIZE/4 || j >= 3*IMG_SIZE/4)
                        test_input[i*IMG_SIZE + j] = 8'd0;
                    else
                        test_input[i*IMG_SIZE + j] = 8'd127;
                end
            end
        `elsif PATTERN_HORI_STRIPE
            for (int i = 0; i < IMG_SIZE; i++) begin
                for (int j = 0; j < IMG_SIZE; j++) begin
                    if ((i / 3) % 2 == 0)
                        test_input[i*IMG_SIZE + j] = 8'd0;
                    else
                        test_input[i*IMG_SIZE + j] = 8'd127;
                end
            end
        `elsif PATTERN_VERT_STRIPE
            for (int i = 0; i < IMG_SIZE; i++) begin
                for (int j = 0; j < IMG_SIZE; j++) begin
                    if ((j / 3) % 2 == 0)
                        test_input[i*IMG_SIZE + j] = 8'd0;
                    else
                        test_input[i*IMG_SIZE + j] = 8'd127;
                end
            end
        `else
            $fatal("Undefined PATTERN macro");
        `endif
    end

    // Kernel selection
    initial begin
        `ifdef TEST_KERNEL_H
            test_weight[0] = -8'sd1;
            test_weight[1] =  8'sd0;
            test_weight[2] =  8'sd1;
            test_weight[3] = -8'sd2;
            test_weight[4] =  8'sd0;
```

```
133             test_weight[5] =  8'sd2;
134             test_weight[6] = -8'sd1;
135             test_weight[7] =  8'sd0;
136             test_weight[8] =  8'sd1;
137         `elsif TEST_KERNEL_V
138             test_weight[0] = -8'sd1;
139             test_weight[1] = -8'sd2;
140             test_weight[2] = -8'sd1;
141             test_weight[3] =  8'sd0;
142             test_weight[4] =  8'sd0;
143             test_weight[5] =  8'sd0;
144             test_weight[6] =  8'sd1;
145             test_weight[7] =  8'sd2;
146             test_weight[8] =  8'sd1;
147         `else
148             $fatal("Undefined TEST_KERNEL macro");
149         `endif
150     end
151
152
153
154     // Initialize test data
155     initial begin
156         // Initialize input image - using (i*2 + j + 1) % 256 pattern
157        // for (int i = 0; i < IMG_SIZE; i++) begin
158        //     for (int j = 0; j < IMG_SIZE; j++) begin
159        //         test_input[i*IMG_SIZE + j] = (i*2 + j + 1) % 256;
160        //     end
161        // end
162
163         // Initialize weights - using 10*(i+1) pattern (weights:
               10,20,30,...,90)
164         // for (int i = 0; i < WEIGHT_SIZE; i++) begin
165         //     test_weight[i] = 10*(i+1);
166         // end
167         fout = $fopen("output_data.txt", "w");
168         // Initialize output array
169         for (int i = 0; i < OUTPUT_SIZE; i++) begin
170             test_output[i] = 0;
171             expected_output[i] = 0;
172         end
173
174         // Pre-calculate expected output (software simulated
               convolution)
175         for (int y = 0; y < IMG_SIZE-2; y++) begin
176             for (int x = 0; x < IMG_SIZE-2; x++) begin
177                 sum = 0;
178                 for (int ky = 0; ky < 3; ky++) begin
179                     for (int kx = 0; kx < 3; kx++) begin
180                         sum += test_input[(y+ky)*IMG_SIZE + (x+kx)] *
                                   test_weight[ky*3 + kx];
181                     end
182                 end
183                 expected_output[y*(IMG_SIZE-2) + x] = sum;
184             end
185         end
186     end
187
```

```
188
189
190     always @(posedge clk) begin
191     prev_output_start <= DUT.ctrl_inst.output_start;
192
193     //
194     if (DUT.ctrl_inst.output_start && !prev_output_start) begin
195         output_active = 1;
196         $display("Output phase started at time %t", $time);
197     end
198
199     //
200     if (output_active && DUT.ctrl_inst.output_start) begin
201         $fwrite(fout, "%0d", DUT.ctrl_inst.output_data);
202         col_counter++;
203         if (col_counter < 14)
204             $fwrite(fout, " ");
205         else begin
206             $fwrite(fout, "\n");
207             col_counter = 0;
208         end
209     end
210
211     //
212     if (output_active && !DUT.ctrl_inst.output_start &&
        prev_output_start) begin
213         $display("Output phase finished at time %t", $time);
214         $fclose(fout);
215         output_active = 0;
216     end
217 end
218
219     //*********************************task
        ****************************//
220     // Task: write data
221     task write_data(input [3:0] address, input [7:0] data);
222         @(posedge clk);
223         addr = address;
224         data_in = data;
225         write = 1;
226         chipselect = 1;
227         @(posedge clk);
228         write = 0;
229         chipselect = 0;
230         @(posedge clk); // extra clock cycle
231     endtask
232
233     // Task: read data
234     task read_data(input [3:0] address, output [7:0] data);
235         @(posedge clk);
236         addr = address;
237         read = 1;
238         chipselect = 1;
239         @(posedge clk);
240         @(posedge clk); // wait for data to stabilize
241         data = data_out;
242         read = 0;
243         chipselect = 0;
```

```
244         @(posedge clk); // extra clock cycle
245     endtask
246 
247     // Task: wait for completion
248     task wait_for_completion(input int max_cycles);
249         logic [7:0] ready_status;
250         int wait_count;
251 
252         do begin
253             wait_count = 0;
254             read_data(4'h3, ready_status);
255             wait_count++;
256 
257             if (wait_count % 50 == 0) begin
258                 $display("Waiting for output ready... (%0d checks)",
                        wait_count);
259             end
260 
261             if (wait_count >= max_cycles) begin
262                 $display("Error: wait timed out! Checked %0d times",
                        wait_count);
263                 test_passed = 0;
264                 break;
265             end
266         end while (ready_status != 8'd1);
267 
268         $display("Output ready! After %0d checks, %0d clock cycles",
                wait_count, cycle_counter);
269     endtask
270 
271     // Task: verify output
272     task verify_output();
273         int mismatch_count;
274 
275         begin
276             mismatch_count = 0;
277             for (int i = 0; i < OUTPUT_SIZE; i++) begin
278                 // Simple check: ensure output is not zero repeatedly
279                 if (i > 0 && test_output[i] == 0 && test_output[i-1] ==
                        0) begin
280                     mismatch_count++;
281                     if (mismatch_count <= 5) begin
282                         $display("Warning: output[%0d] and output[%0d]
                                are both zero", i-1, i);
283                     end
284                 end
285             end
286         end
287 
288         if (mismatch_count > OUTPUT_SIZE / 2) begin
289             $display("Error: more than half of outputs are zero or
                    abnormal pattern!");
290             test_passed = 0;
291         end else if (mismatch_count > 0) begin
292             $display("Warning: found %0d suspicious output patterns",
                    mismatch_count);
293         end else begin
294             $display("Output format verification passed!");
```

```
295         end
296
297         // Print some sample data for manual check
298         $display("Sample output data:");
299         for (int i = 0; i < 5; i++) begin
300             for (int j = 0; j < 5; j++) begin
301                 $write("%4d ", test_output[i*(IMG_SIZE-2) + j]);
302             end
303             $write("\n");
304         end
305     endtask
306
307
308     //*******************************main test
        ***************************//
309     // Main testbench procedure
310     initial begin
311         // Initialize signals
312         reset = 0;
313         write = 0;
314         read = 0;
315         addr = 0;
316         data_in = 0;
317         chipselect = 0;
318
319         // Reset
320         #(CLK_PERIOD*50);
321         reset = 1;
322         #(CLK_PERIOD*2);
323         reset = 0;
324         // Select load sequence based on TEST_SEQUENCE
325         if (TEST_SEQUENCE == 0) begin
326             // Sequence 1: weights first, then input data
327             $display("\nTest Sequence 1: Load weights first, then input
                data");
328
329             // Step 1: set image size
330             $display("Step 1: Set image size to %0d x %0d", IMG_SIZE,
                IMG_SIZE);
331             write_data(4'h0, IMG_SIZE);
332
333             // Step 2: load weight data
334             $display("Step 2: Load weight data (3x3 = 9 weights)");
335             for (int i = 0; i < WEIGHT_SIZE; i++) begin
336                 write_data(4'h1, test_weight[i]);
337                 $display("  Loaded weight[%0d] = %0d", i, test_weight[i
                    ]);
338             end
339
340             // Step 3: load input image data
341             $display("Step 3: Load input image data (%0d pixels)",
                INPUT_SIZE);
342             for (int i = 0; i < IMG_SIZE; i++) begin
343                 for (int j = 0; j < IMG_SIZE; j++) begin
344                     write_data(4'h2, test_input[i*IMG_SIZE + j]);
345                     if ((i*IMG_SIZE + j) % IMG_SIZE == 0)
346                         $display("  Loading image row %0d/%0d", i+1,
                            IMG_SIZE);
```

```
347                 end
348             end
349         end else begin
350             // Sequence 2: input data first, then weights
351             $display("\nTest Sequence 2: Load input data first, then
                    weights");
352 
353             // Step 1: set image size
354             $display("Step 1: Set image size to %0d x %0d", IMG_SIZE,
                    IMG_SIZE);
355             write_data(4'h0, IMG_SIZE);
356 
357             // Step 2: load input image data
358             $display("Step 2: Load input image data (%0d pixels)",
                    INPUT_SIZE);
359             for (int i = 0; i < IMG_SIZE; i++) begin
360                 for (int j = 0; j < IMG_SIZE; j++) begin
361                     write_data(4'h2, test_input[i*IMG_SIZE + j]);
362                     if ((i*IMG_SIZE + j) % IMG_SIZE == 0)
363                         $display("  Loading image row %0d/%0d", i+1,
                                IMG_SIZE);
364                 end
365             end
366 
367             // Step 3: load weight data
368             $display("Step 3: Load weight data (3x3 = 9 weights)");
369             for (int i = 0; i < WEIGHT_SIZE; i++) begin
370                 write_data(4'h1, test_weight[i]);
371                 $display("  Loaded weight[%0d] = %0d", i, test_weight[i
                        ]);
372             end
373         end
374 
375         // Step 4: waiting for convolution processing to complete
376         $display("Step 4: Waiting for convolution processing to
                complete");
377         wait_for_completion(MAX_WAIT_CYCLES);
378 
379         // Step 5: read output data
380         $display("Step 5: Read output data (%0d values)", OUTPUT_SIZE);
381         for (int i = 0; i < OUTPUT_SIZE; i++) begin
382             read_data(4'h4, test_output[i]);
383             output_counter++;
384 
385             if (i % (IMG_SIZE-2) == 0 && i > 0)
386                 $display("  Reading output row %0d/%0d", i/(IMG_SIZE-2)
                        , (IMG_SIZE-2));
387         end
388 
389         // Step 6: verify output results
390         $display("Step 6: Verify output results");
391         verify_output();
392 
393         // Test summary
394         $display("\nTest Summary:");
395         $display("  Number of input pixels processed: %0d", INPUT_SIZE)
                ;
396         $display("  Number of weights used: %0d", WEIGHT_SIZE);
```

```
397         $display("  Number of outputs generated: %0d", OUTPUT_SIZE);
398         $display("  Number of outputs read: %0d", output_counter);
399         $display("  Total number of cycles executed: %0d",
                cycle_counter);
400
401         if (test_passed)
402             $display("\nTest completed: PASSED!");
403         else
404             $display("\nTest completed: FAILED! Please check the above
                    error messages.");
405
406         #(CLK_PERIOD*10);
407         $finish;
408     end
409
410     // Monitor important signals (optional for debugging)
411     initial begin
412         $timeformat(-9, 2, " ns", 20);
413     end
414
415 endmodule
```

5.16 Waveform Script (Input Buffer) (test/wave_inputbuffer_PE.do)

Listing 16: Waveform script for input buffer (‘test/wave_inputbuffer_PE.do‘)

```
1  onerror {resume}
2  quietly WaveActivateNextPane {} 0
3  add wave -noupdate /testbench/clk
4  add wave -noupdate /testbench/rst_n
5  add wave -noupdate /testbench/en
6  add wave -noupdate /testbench/w_en
7  add wave -noupdate /testbench/data_in
8  add wave -noupdate /testbench/rd_en
9  add wave -noupdate /testbench/wr_en
10 add wave -noupdate /testbench/data_out
11 add wave -noupdate /testbench/buffer_done
12 add wave -noupdate /testbench/active_left
13 add wave -noupdate /testbench/in_weight_above
14 add wave -noupdate /testbench/out_weight_final
15 add wave -noupdate /testbench/out_sum_final
16 add wave -noupdate /testbench/errors
17 add wave -noupdate /testbench/total_tests
18 add wave -noupdate /testbench/input_values
19 add wave -noupdate /testbench/output_values
20 add wave -noupdate /testbench/output_index
21 add wave -noupdate -divider {New Divider}
22 add wave -noupdate /testbench/input_buffer_inst/data_in_bank0
23 add wave -noupdate /testbench/input_buffer_inst/data_in_bank1
24 add wave -noupdate /testbench/input_buffer_inst/data_in_bank2
25 add wave -noupdate /testbench/input_buffer_inst/data_in_bank3
26 add wave -noupdate /testbench/input_buffer_inst/data_in_bank4
27 add wave -noupdate /testbench/input_buffer_inst/data_in_bank5
28 add wave -noupdate /testbench/input_buffer_inst/data_in_bank6
29 add wave -noupdate /testbench/input_buffer_inst/data_in_bank7
30 add wave -noupdate /testbench/input_buffer_inst/data_in_bank8
31 add wave -noupdate /testbench/input_buffer_inst/index_wr
```

```
add wave -noupdate /testbench/input_buffer_inst/index_rd
add wave -noupdate /testbench/input_buffer_inst/index_bank0
add wave -noupdate /testbench/input_buffer_inst/counter
add wave -noupdate /testbench/input_buffer_inst/index_bank1
add wave -noupdate /testbench/input_buffer_inst/index_bank2
add wave -noupdate /testbench/input_buffer_inst/index_bank3
add wave -noupdate /testbench/input_buffer_inst/index_bank4
add wave -noupdate /testbench/input_buffer_inst/index_bank5
add wave -noupdate /testbench/input_buffer_inst/index_bank6
add wave -noupdate /testbench/input_buffer_inst/index_bank7
add wave -noupdate /testbench/input_buffer_inst/index_bank8
add wave -noupdate -divider {New Divider}
add wave -noupdate /testbench/pe_array_inst/CLK
add wave -noupdate /testbench/pe_array_inst/RESET
add wave -noupdate /testbench/pe_array_inst/EN
add wave -noupdate /testbench/pe_array_inst/W_EN
add wave -noupdate /testbench/pe_array_inst/active_left
add wave -noupdate /testbench/pe_array_inst/in_weight_above
add wave -noupdate /testbench/pe_array_inst/out_weight_final
add wave -noupdate /testbench/pe_array_inst/out_sum_final
add wave -noupdate /testbench/pe_array_inst/weight_connections
add wave -noupdate /testbench/pe_array_inst/sum_connections
add wave -noupdate -divider {New Divider}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/CLK}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/RESET}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/EN}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/W_EN}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/active_left}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/input_next}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/active_right}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/in_sum}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/out_sum}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/in_weight_above}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/out_weight_below}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/weight_next}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/weight_q}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/input_q}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/sum_next}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/sum_q}
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label
    [0]/genblk1/PE_unit/out_next}
```

```
add wave -noupdate {/testbench/pe_array_inst/label[0]/PE_row_unit/label[0]/genblk1/PE_unit/out_q}
add wave -noupdate -radix hexadecimal {/testbench/pe_array_inst/label[0]/PE_row_unit/label[0]/genblk1/PE_unit/mul_result}
add wave -noupdate -radix hexadecimal {/testbench/pe_array_inst/label[0]/PE_row_unit/label[0]/genblk1/PE_unit/add_result}
add wave -noupdate -divider {New Divider}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/CLK}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/RESET}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/EN}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/W_EN}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/active_left}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/active_right}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/in_sum}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/out_sum}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/in_weight_above}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/out_weight_below}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/weight_next}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/weight_q}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/input_next}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/input_q}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/sum_next}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/sum_q}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/out_next}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/out_q}
add wave -noupdate {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/mul_result}
add wave -noupdate -radix hexadecimal {/testbench/pe_array_inst/label[1]/PE_row_unit/label[0]/genblk1/PE_unit/add_result}
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {3073982 ps} 0}
quietly wave cursor active 1
configure wave -namecolwidth 538
configure wave -valuecolwidth 100
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
```

```
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ps
update
WaveRestoreZoom {0 ps} {5570250 ps}
```

5.17 Waveform Script (Top Level) (test/wave_top.do)

Listing 17: Waveform script for top level (‘test/wave_top.do‘)

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate /top_tb/clk
add wave -noupdate /top_tb/data_in
add wave -noupdate /top_tb/write
add wave -noupdate /top_tb/read
add wave -noupdate /top_tb/addr
add wave -noupdate /top_tb/chipselect
add wave -noupdate /top_tb/data_out
add wave -noupdate /top_tb/test_input
add wave -noupdate /top_tb/test_weight
add wave -noupdate /top_tb/test_output
add wave -noupdate /top_tb/expected_output
add wave -noupdate /top_tb/sum
add wave -noupdate /top_tb/output_counter
add wave -noupdate /top_tb/cycle_counter
add wave -noupdate -divider {input buffer}
add wave -noupdate /top_tb/DUT/input_buffer_inst/clk
add wave -noupdate /top_tb/DUT/input_buffer_inst/rst_n
add wave -noupdate /top_tb/DUT/input_buffer_inst/data_in
add wave -noupdate /top_tb/DUT/input_buffer_inst/rd_en
add wave -noupdate /top_tb/DUT/input_buffer_inst/wr_en
add wave -noupdate /top_tb/DUT/input_buffer_inst/data_out
add wave -noupdate /top_tb/DUT/input_buffer_inst/ready
add wave -noupdate /top_tb/DUT/input_buffer_inst/done
add wave -noupdate /top_tb/DUT/input_buffer_inst/data_in_bank0
add wave -noupdate /top_tb/DUT/input_buffer_inst/data_in_bank1
add wave -noupdate /top_tb/DUT/input_buffer_inst/data_in_bank2
add wave -noupdate /top_tb/DUT/input_buffer_inst/data_in_bank3
add wave -noupdate /top_tb/DUT/input_buffer_inst/data_in_bank4
add wave -noupdate /top_tb/DUT/input_buffer_inst/data_in_bank5
add wave -noupdate /top_tb/DUT/input_buffer_inst/data_in_bank6
add wave -noupdate /top_tb/DUT/input_buffer_inst/data_in_bank7
add wave -noupdate /top_tb/DUT/input_buffer_inst/data_in_bank8
add wave -noupdate /top_tb/DUT/input_buffer_inst/index_wr
add wave -noupdate /top_tb/DUT/input_buffer_inst/index_rd
add wave -noupdate /top_tb/DUT/input_buffer_inst/index_bank0
add wave -noupdate /top_tb/DUT/input_buffer_inst/counter
add wave -noupdate /top_tb/DUT/input_buffer_inst/index_bank1
add wave -noupdate /top_tb/DUT/input_buffer_inst/index_bank2
add wave -noupdate /top_tb/DUT/input_buffer_inst/index_bank3
add wave -noupdate /top_tb/DUT/input_buffer_inst/index_bank4
add wave -noupdate /top_tb/DUT/input_buffer_inst/index_bank5
add wave -noupdate /top_tb/DUT/input_buffer_inst/index_bank6
add wave -noupdate /top_tb/DUT/input_buffer_inst/index_bank7
```

```
add wave -noupdate /top_tb/DUT/input_buffer_inst/index_bank8
add wave -noupdate -color {Blue Violet} /top_tb/DUT/weight_buffer_inst/
    data_out
add wave -noupdate -color {Blue Violet} /top_tb/DUT/weight_buffer_inst/
    rd_addr
add wave -noupdate -color {Blue Violet} /top_tb/DUT/weight_buffer_inst/
    weight_bank0
add wave -noupdate -divider ctrl
add wave -noupdate /top_tb/DUT/ctrl_inst/clk
add wave -noupdate /top_tb/DUT/ctrl_inst/rst_n
add wave -noupdate /top_tb/DUT/ctrl_inst/input_data_ready
add wave -noupdate /top_tb/DUT/ctrl_inst/weight_data_ready
add wave -noupdate /top_tb/DUT/ctrl_inst/weight_data
add wave -noupdate /top_tb/DUT/ctrl_inst/input_start
add wave -noupdate /top_tb/DUT/ctrl_inst/weight_start
add wave -noupdate /top_tb/DUT/ctrl_inst/output_data
add wave -noupdate -color Yellow /top_tb/DUT/ctrl_inst/output_start
add wave -noupdate /top_tb/DUT/ctrl_inst/current_state
add wave -noupdate /top_tb/DUT/ctrl_inst/next_state
add wave -noupdate -color {Blue Violet} /top_tb/DUT/ctrl_inst/
    weight_reg
add wave -noupdate /top_tb/DUT/ctrl_inst/input_reg
add wave -noupdate /top_tb/DUT/ctrl_inst/output_reg
add wave -noupdate -color Cyan -radix unsigned /top_tb/DUT/ctrl_inst/
    weight_counter
add wave -noupdate -color Cyan /top_tb/DUT/ctrl_inst/weight_loaded
add wave -noupdate /top_tb/DUT/ctrl_inst/processing_done
add wave -noupdate -color Yellow /top_tb/DUT/ctrl_inst/input_data
add wave -noupdate -color Gold /top_tb/DUT/ctrl_inst/output_start
add wave -noupdate -color Coral -itemcolor Yellow -radix unsigned /
    top_tb/DUT/ctrl_inst/data_counter
add wave -noupdate -itemcolor Yellow -radix unsigned /top_tb/DUT/
    ctrl_inst/output_counter
add wave -noupdate /top_tb/DUT/output_buffer_inst/wr_en
add wave -noupdate /top_tb/DUT/output_buffer_inst/output_bank0
add wave -noupdate /top_tb/DUT/output_buffer_inst/wr_addr
add wave -noupdate /top_tb/DUT/output_buffer_inst/ready
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_en
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_w_en
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_active_left
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_in_weight_above
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_out_sum_final
add wave -noupdate -divider ctrl
add wave -noupdate /top_tb/DUT/ctrl_inst/clk
add wave -noupdate /top_tb/DUT/ctrl_inst/rst_n
add wave -noupdate /top_tb/DUT/ctrl_inst/input_data_ready
add wave -noupdate /top_tb/DUT/ctrl_inst/weight_data_ready
add wave -noupdate /top_tb/DUT/ctrl_inst/weight_data
add wave -noupdate /top_tb/DUT/ctrl_inst/weight_start
add wave -noupdate /top_tb/DUT/ctrl_inst/current_state
add wave -noupdate /top_tb/DUT/ctrl_inst/next_state
add wave -noupdate -color Cyan -radix unsigned /top_tb/DUT/ctrl_inst/
    weight_counter
add wave -noupdate -color Cyan /top_tb/DUT/ctrl_inst/weight_loaded
add wave -noupdate -color Green /top_tb/DUT/ctrl_inst/weight_reg
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_en
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_w_en
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_in_weight_above
```

```
add wave -noupdate -color {Green Yellow} /top_tb/DUT/ctrl_inst/
   pe_array_inst/weight_connections
add wave -noupdate -divider {PE ARRAY}
add wave -noupdate -color {Blue Violet} /top_tb/DUT/ctrl_inst/
   pe_array_inst/weight_connections
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_array_inst/CLK
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_array_inst/RESET
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_array_inst/EN
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_array_inst/W_EN
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_array_inst/active_left
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_array_inst/in_weight_above
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_array_inst/out_weight_final
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_array_inst/out_sum_final
add wave -noupdate /top_tb/DUT/ctrl_inst/pe_array_inst/sum_connections
add wave -noupdate -divider {weight buffer}
add wave -noupdate /top_tb/DUT/weight_buffer_inst/clk
add wave -noupdate /top_tb/DUT/weight_buffer_inst/rst_n
add wave -noupdate /top_tb/DUT/weight_buffer_inst/data_in
add wave -noupdate /top_tb/DUT/weight_buffer_inst/wr_en
add wave -noupdate /top_tb/DUT/weight_buffer_inst/rd_en
add wave -noupdate /top_tb/DUT/weight_buffer_inst/data_out
add wave -noupdate /top_tb/DUT/weight_buffer_inst/ready
add wave -noupdate /top_tb/DUT/weight_buffer_inst/weight_bank0
add wave -noupdate /top_tb/DUT/weight_buffer_inst/wr_addr
add wave -noupdate /top_tb/DUT/weight_buffer_inst/rd_addr
add wave -noupdate -divider {output buffer}
add wave -noupdate /top_tb/DUT/output_buffer_inst/clk
add wave -noupdate /top_tb/DUT/output_buffer_inst/rst_n
add wave -noupdate /top_tb/DUT/output_buffer_inst/data_in
add wave -noupdate /top_tb/DUT/output_buffer_inst/rd_en
add wave -noupdate /top_tb/DUT/output_buffer_inst/data_out
add wave -noupdate /top_tb/DUT/output_buffer_inst/ready
add wave -noupdate /top_tb/DUT/output_buffer_inst/wr_en
add wave -noupdate /top_tb/DUT/output_buffer_inst/output_bank0
add wave -noupdate /top_tb/DUT/output_buffer_inst/wr_addr
add wave -noupdate /top_tb/DUT/output_buffer_inst/rd_addr
add wave -noupdate /top_tb/DUT/weight_buffer_inst/rd_en
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {948170 ps} 0} {{Cursor 2} {5853498 ps}
   0}
quietly wave cursor active 2
configure wave -namecolwidth 365
configure wave -valuecolwidth 100
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ps
update
WaveRestoreZoom {8476137 ps} {8844703 ps}
```

5.18 Host–FPGA Interface (main.c)

Listing 18: Host–FPGA interface program (‘main.c‘)

```c
/*
 * Userspace program that communicates with the top device driver
 * through ioctls
 *
 * Hang Ye
 * Columbia University
 */

#include <stdio.h>
#include "top.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

int top_fd;

/* Write img_size */
void write_img_size(unsigned char img_size) {
    top_arg_t vla;
    vla.img_size.img_size = img_size;
    if (ioctl(top_fd, TOP_WRITE_IMG_SIZE, &vla)) {
        perror("ioctl(TOP_WRITE_IMG_SIZE) failed");
        return;
    }
}

/* Write input_data */
void write_input_data(unsigned char input_data) {
    top_arg_t vla;
    vla.input_data.input_data = input_data;
    if (ioctl(top_fd, TOP_WRITE_INPUT_DATA, &vla)) {
        perror("ioctl(TOP_WRITE_INPUT_DATA) failed");
        return;
    }
}

/* Write weight_data */
void write_weight_data(unsigned char weight_data) {
    top_arg_t vla;
    vla.weight_data.weight_data = weight_data;
    if (ioctl(top_fd, TOP_WRITE_WEIGHT_DATA, &vla)) {
        perror("ioctl(TOP_WRITE_WEIGHT_DATA) failed");
        return;
    }
}

/* Read done signal */
unsigned char read_done() {
    top_arg_t vla;
    if (ioctl(top_fd, TOP_READ_DONE, &vla)) {
        perror("ioctl(TOP_READ_DONE) failed");
```

```
56          return 0;
57      }
58      return vla.done.done;
59  }
60
61  /* Read output_data */
62  unsigned char read_output_data() {
63      top_arg_t vla;
64      if (ioctl(top_fd, TOP_READ_OUTPUT_DATA, &vla)) {
65          perror("ioctl(TOP_READ_OUTPUT_DATA) failed");
66          return 0;
67      }
68      return vla.output_data.output_data;
69  }
70
71  int main() {
72      static const char filename[] = "/dev/top";
73      unsigned char img_size = 16; // Example img_size
74      unsigned char input_data[16*16];
75      signed char weight_data[3*3] = {-1, 0, 1, -2, 0, 2, -1, 0, 1};
76  //     signed char weight_data[3*3];
77      unsigned char output_data[14*14];
78      int golden_data[14*14];
79      int i;
80      int j;
81
82      printf("Top Userspace program started\n");
83
84      if ((top_fd = open(filename, O_RDWR)) == -1) {
85          fprintf(stderr, "could not open %s\n", filename);
86          return -1;
87      }
88
89      // Initialize input_data and weight_data
90      for (i = 0; i < 16 ; i++) {
91         for (j = 0; j < 16; j++) {
92             if (j == 4 || j ==5 || j == 6 || j == 10 || j == 11 || j ==
                    12 || j == 15){
93                 input_data[i*16+j] = 127;
94             }
95             else {
96                 input_data[i*16+j] = 0;
97             }
98         }
99     }
100 //      // Initialize input_data and weight_data
101 //      for (i = 0; i < 16*16; i++) {
102 //         input_data[i] = rand() % 256; // Random value between 0 and
        255
103 //      }
104 //      for (i = 0; i < 3*3; i++) {
105 //          weight_data[i] = rand() % 32;      // Example weight data: 32
106 //      }
107      // Write img_size
108      printf("Writing img_size: %d\n", img_size);
109     //  write_img_size(img_size);
110
111      // Write input_data
```

```
112         printf("Writing input_data:\n");
113         for (i = 0; i < 16*16; i++) {
114             printf("  input_data[%d] = %d\n", i, input_data[i]);
115             write_input_data(input_data[i]);
116         }
117        for (i = 0; i < 14; i++)
118            for (j = 0; j < 14; j++) {
119                golden_data[i*14+j] =   input_data[i*16+j]          *
                       weight_data[0]+
120                                        input_data[i*16+j+1]        *
                           weight_data[1]+
121                                        input_data[i*16+j+2]        *
                           weight_data[2]+
122                                        input_data[(i+1)*16+j]      *
                           weight_data[3]+
123                                        input_data[(i+1)*16+j+1]    *
                           weight_data[4]+
124                                        input_data[(i+1)*16+j+2]    *
                           weight_data[5]+
125                                        input_data[(i+2)*16+j]      *
                           weight_data[6]+
126                                        input_data[(i+2)*16+j+1]    *
                           weight_data[7]+
127                                        input_data[(i+2)*16+j+2]    *
                           weight_data[8];
128            }
129
130
131         printf("Writing weight_data:\n");
132         for (i = 0; i < 9; i++) {
133             printf("  weight_data[%d] = %d\n", i, weight_data[i]);
134             write_weight_data(weight_data[i]);
135         }
136         // Wait for done signal
137         printf("Waiting for done signal...\n");
138         i = 0;
139         while (read_done() == 0){
140             printf("  done = 0\n");
141             i = i + 1;
142             usleep(100000); // Sleep for 100ms
143         }
144         printf("  done = 1\n");
145         printf("Done signal received!\n");
146
147         // Read output_data
148         printf("Reading output_data:\n");
149         for (i = 0; i < 14*14; i++) {
150            output_data[i] = read_output_data();
151            //  output_data[i] = (int)(golden_data[i]/4);
152             printf("  output_data[%d] = %d\n", i, output_data[i]);
153
154             // Verify output_data
155             if (golden_data[i] !=0){
156             if ((abs(output_data[i] * 4 - golden_data[i]) / golden_data[i
                   ]) > 0.5) {
157                 fprintf(stderr, "Error: output_data[%d] = %d (expected %d)
                       \n",
158                         i, output_data[i], golden_data[i]);
```

```
159             }}
160         }
161         for (i = 0; i < 14; i++) {
162             for (j = 0; j < 14; j++) {
163                 printf("  %d, ", output_data[i*14+j]);
164                 }
165                printf("\n");
166                }
167         printf("Top Userspace program terminating\n");
168         return 0;
169  }
170 //   /*
171 //   * Userspace program that communicates with the top device driver
172 //   * through ioctls
173 //   *
174 //   * Hang Ye
175 //   * Columbia University
176 //   */
```

5.19 Test Harness Main (top.c)

Listing 19: Test harness main program (‘top.c‘)

```
1  /* * Device driver for the VGA video generator
2   *
3   * A Platform device implemented using the misc subsystem
4   *
5   * Project final, csee4840
6   * Columbia University
7   * Hang Ye
8   *
9   * Stephen A. Edwards
10  * Columbia University
11  *
12  * References:
13  * Linux source: Documentation/driver-model/platform.txt
14  *               drivers/misc/arm-charlcd.c
15  * http://www.linuxforu.com/tag/linux-device-drivers/
16  * http://free-electrons.com/docs/
17  *
18  * "make" to build
19  * insmod vga_ball.ko
20  *
21  * Check code style with
22  * checkpatch.pl --file --no-tree vga_ball.c
23  */
24
25 #include <linux/module.h>
26 #include <linux/init.h>
27 #include <linux/errno.h>
28 #include <linux/version.h>
29 #include <linux/kernel.h>
30 #include <linux/platform_device.h>
31 #include <linux/miscdevice.h>
32 #include <linux/slab.h>
33 #include <linux/io.h>
34 #include <linux/of.h>
```

```
35 #include <linux/of_address.h>
36 #include <linux/fs.h>
37 #include <linux/uaccess.h>
38 #include "top.h"
39
40 #define DRIVER_NAME "top"
41
42 /* Device registers */
43 #define IMG_SIZE(x)   (x)
44 #define WEIGHT_DATA(x)  ((x)+1)
45 #define INPUT_DATA(x)   ((x)+2)
46 #define DONE(x)     ((x)+3)
47 #define OUTPUT_DATA(x)  ((x)+4)
48
49 /*
50  * Information about our device
51  */
52 struct top_dev {
53   struct resource res; /* Resource: our registers */
54   void __iomem *virtbase; /* Where registers can be accessed in memory
       */
55     top_input_data_t input_data;
56     top_weight_data_t weight_data;
57     top_img_size_t img_size;
58     top_output_data_t output_data;
59     top_done_t done;
60 } dev;
61
62 /*
63  * Write segments of a single digit
64  * Assumes digit is in range and the device information has been set up
65  */
66 static void write_img_size(top_img_size_t *img_size)
67 {
68   iowrite8(img_size->img_size, IMG_SIZE(dev.virtbase) );
69   dev.img_size = *img_size;
70 }
71 static void write_weight_data(top_weight_data_t *weight_data)
72 {
73   iowrite8(weight_data->weight_data, WEIGHT_DATA(dev.virtbase) );
74   dev.weight_data = *weight_data;
75 }
76 static void write_input_data(top_input_data_t *input_data)
77 {
78   iowrite8(input_data->input_data, INPUT_DATA(dev.virtbase) );
79   dev.input_data = *input_data;
80 }
81 static void read_output_data(top_output_data_t *output_data)
82 {
83     output_data->output_data = ioread8(OUTPUT_DATA(dev.virtbase));
84     dev.output_data = *output_data;
85 }
86
87 static void read_done(top_done_t *done)
88 {
89     done->done = ioread8(DONE(dev.virtbase));
90     dev.done = *done;
91 }
```

```
92 /*
93  * Handle ioctl() calls from userspace:
94  * Read or write the segments on single digits.
95  * Note extensive error checking of arguments
96  */
97  static long top_ioctl(struct file *f, unsigned int cmd, unsigned long
        arg)
98  {
99    top_arg_t vla;
100
101   switch (cmd) {
102   case TOP_WRITE_IMG_SIZE:
103     if (copy_from_user(&vla, (top_arg_t __user *)arg, sizeof(vla)))
104       return -EACCES;
105     write_img_size(&vla.img_size);
106     break;
107
108   case TOP_WRITE_WEIGHT_DATA:
109     if (copy_from_user(&vla, (top_arg_t __user *)arg, sizeof(vla)))
110       return -EACCES;
111     write_weight_data(&vla.weight_data);
112     break;
113
114   case TOP_WRITE_INPUT_DATA:
115     if (copy_from_user(&vla, (top_arg_t __user *)arg, sizeof(vla)))
116       return -EACCES;
117     write_input_data(&vla.input_data);
118     break;
119
120   case TOP_READ_OUTPUT_DATA:
121     read_output_data(&vla.output_data);
122     if (copy_to_user((top_arg_t __user *)arg, &vla, sizeof(vla)))
123       return -EACCES;
124     break;
125
126   case TOP_READ_DONE:
127     read_done(&vla.done);
128     if (copy_to_user((top_arg_t __user *)arg, &vla, sizeof(vla)))
129       return -EACCES;
130     break;
131
132   default:
133     return -EINVAL;
134   }
135
136   return 0;
137 }
138
139 /* The operations our device knows how to do */
140 static const struct file_operations top_fops = {
141   .owner    = THIS_MODULE,
142   .unlocked_ioctl = top_ioctl,
143 };
144
145 /* Information about our device for the "misc" framework -- like a char
        dev */
146 static struct miscdevice top_misc_device = {
147   .minor    = MISC_DYNAMIC_MINOR,
```

```
148   .name    = DRIVER_NAME,
149   .fops    = &top_fops,
150 };
151
152 /*
153  * Initialization code: get resources (registers) and display
154  * a welcome message
155  */
156 static int __init top_probe(struct platform_device *pdev)
157 {
158   int ret;
159
160     /* Register ourselves as a misc device: creates /dev/top */
161     ret = misc_register(&top_misc_device);
162
163     /* Get the address of our registers from the device tree */
164     ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
165     if (ret) {
166         ret = -ENOENT;
167         goto out_deregister;
168     }
169
170     /* Make sure we can use these registers */
171     if (request_mem_region(dev.res.start, resource_size(&dev.res),
172                    DRIVER_NAME) == NULL) {
173         ret = -EBUSY;
174         goto out_deregister;
175     }
176
177     /* Arrange access to our registers */
178     dev.virtbase = of_iomap(pdev->dev.of_node, 0);
179     if (dev.virtbase == NULL) {
180         ret = -ENOMEM;
181         goto out_release_mem_region;
182     }
183
184     return 0;
185
186 out_release_mem_region:
187     release_mem_region(dev.res.start, resource_size(&dev.res));
188 out_deregister:
189     misc_deregister(&top_misc_device);
190     return ret;
191 }
192
193 /* Clean-up code: release resources */
194 static int top_remove(struct platform_device *pdev)
195 {
196     iounmap(dev.virtbase);
197     release_mem_region(dev.res.start, resource_size(&dev.res));
198     misc_deregister(&top_misc_device);
199     return 0;
200 }
201
202 /* Which "compatible" string(s) to search for in the Device Tree */
203 #ifdef CONFIG_OF
204 static const struct of_device_id top_of_match[] = {
205     { .compatible = "csee4840,top-1.0" },
```

```
    {},
};
MODULE_DEVICE_TABLE(of, top_of_match);
#endif

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

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

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

module_init(top_init);
module_exit(top_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Hang Ye, Columbia University");
MODULE_DESCRIPTION("Top driver");
```

5.20 Test Harness Header (top.h)

Listing 20: Test harness header file (‘top.h‘)

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>

// typedef struct {
//   unsigned char red, green, blue;
// } vga_ball_color_t;


// typedef struct {
//   vga_ball_color_t background;
// } vga_ball_arg_t;
typedef struct {
  unsigned char weight_data;
}top_weight_data_t;
typedef struct {
```

```
  unsigned char input_data;
}top_input_data_t;
typedef struct {
  unsigned char img_size;
}top_img_size_t;
typedef struct {
  unsigned char done;
}top_done_t;
typedef struct {
  unsigned char output_data;
}top_output_data_t;
typedef struct {
  top_input_data_t input_data;
  top_weight_data_t weight_data;
  top_img_size_t img_size;
  top_output_data_t output_data;
  top_done_t done;
} top_arg_t;
#define TOP_MAGIC 'q'

/* ioctls and their arguments */
#define TOP_WRITE_IMG_SIZE      _IOW(TOP_MAGIC, 1, top_arg_t)
#define TOP_WRITE_WEIGHT_DATA   _IOW(TOP_MAGIC, 2, top_arg_t)
#define TOP_WRITE_INPUT_DATA    _IOW(TOP_MAGIC, 3, top_arg_t)
#define TOP_READ_OUTPUT_DATA    _IOR(TOP_MAGIC, 4, top_arg_t)
#define TOP_READ_DONE           _IOR(TOP_MAGIC, 5, top_arg_t)

#endif
```