module image\_line\_buffer (

input clk, // Clock signal

input reset, // Reset signal

input [7:0] pixel\_in, // 8-bit input pixel (could be grayscale)

input valid\_in, // Input valid signal

output reg [7:0] pixel\_out, // Output pixel after reconstruction

output reg valid\_out, // Output valid signal

input [4:0] x, // Horizontal position of pixel

input [4:0] y // Vertical position of pixel

);

// Declare a 3x3 line buffer

reg [7:0] line\_buffer [2:0][2:0]; // 3x3 buffer for 8-bit pixels

// Shift the line buffer and store new pixel values

always @(posedge clk or posedge reset) begin

if (reset) begin

// Clear the buffer on reset

line\_buffer[0][0] <= 8'b0;

line\_buffer[0][1] <= 8'b0;

line\_buffer[0][2] <= 8'b0;

line\_buffer[1][0] <= 8'b0;

line\_buffer[1][1] <= 8'b0;

line\_buffer[1][2] <= 8'b0;

line\_buffer[2][0] <= 8'b0;

line\_buffer[2][1] <= 8'b0;

line\_buffer[2][2] <= 8'b0;

valid\_out <= 0;

end

else if (valid\_in) begin

// Shift the lines and store the incoming pixel in the appropriate spot

line\_buffer[0][0] <= line\_buffer[1][0];

line\_buffer[0][1] <= line\_buffer[1][1];

line\_buffer[0][2] <= line\_buffer[1][2];

line\_buffer[1][0] <= line\_buffer[2][0];

line\_buffer[1][1] <= line\_buffer[2][1];

line\_buffer[1][2] <= line\_buffer[2][2];

line\_buffer[2][0] <= pixel\_in; // New pixel at the bottom of the buffer

line\_buffer[2][1] <= pixel\_in;

line\_buffer[2][2] <= pixel\_in;

end

end

// Output the 3x3 reconstructed pixel block based on the x, y position

always @(\*) begin

if (valid\_in) begin

case ({x[0], y[0]})

2'b00: pixel\_out = line\_buffer[0][0];

2'b01: pixel\_out = line\_buffer[0][1];

2'b10: pixel\_out = line\_buffer[0][2];

2'b11: pixel\_out = line\_buffer[1][0];

default: pixel\_out = 8'b0;

endcase

end

end

endmodule

module tb\_image\_line\_buffer;

// Testbench signals

reg clk; // Clock signal

reg reset; // Reset signal

reg [7:0] pixel\_in; // 8-bit pixel input

reg valid\_in; // Input valid signal

reg [4:0] x; // Horizontal position of pixel

reg [4:0] y; // Vertical position of pixel

wire [7:0] pixel\_out; // Output pixel

wire valid\_out; // Output valid signal

// Instantiate the image\_line\_buffer module

image\_line\_buffer uut (

.clk(clk),

.reset(reset),

.pixel\_in(pixel\_in),

.valid\_in(valid\_in),

.pixel\_out(pixel\_out),

.valid\_out(valid\_out),

.x(x),

.y(y)

);

// Clock generation

always begin

#5 clk = ~clk; // Toggle clock every 5 time units

end

// Test sequence

initial begin

// Initialize signals

clk = 0;

reset = 0;

pixel\_in = 8'b0;

valid\_in = 0;

x = 0;

y = 0;

// Apply reset

reset = 1;

#10 reset = 0; // Reset for 10 time units

// Test case 1: Input some pixels and test the buffer

// First row (x = 0, 1, 2)

pixel\_in = 8'd10; valid\_in = 1; x = 0; y = 0; #10;

pixel\_in = 8'd20; valid\_in = 1; x = 1; y = 0; #10;

pixel\_in = 8'd30; valid\_in = 1; x = 2; y = 0; #10;

valid\_in = 0; // End of row 1

// Second row (x = 0, 1, 2)

pixel\_in = 8'd40; valid\_in = 1; x = 0; y = 1; #10;

pixel\_in = 8'd50; valid\_in = 1; x = 1; y = 1; #10;

pixel\_in = 8'd60; valid\_in = 1; x = 2; y = 1; #10;

valid\_in = 0; // End of row 2

// Third row (x = 0, 1, 2)

pixel\_in = 8'd70; valid\_in = 1; x = 0; y = 2; #10;

pixel\_in = 8'd80; valid\_in = 1; x = 1; y = 2; #10;

pixel\_in = 8'd90; valid\_in = 1; x = 2; y = 2; #10;

valid\_in = 0; // End of row 3

// Test accessing pixels (reconstruction)

// After a complete 3x3 block is filled, we test reconstruction

x = 0; y = 0; #10; // Should be 10

x = 1; y = 0; #10; // Should be 20

x = 2; y = 0; #10; // Should be 30

x = 0; y = 1; #10; // Should be 40

x = 1; y = 1; #10; // Should be 50

x = 2; y = 1; #10; // Should be 60

x = 0; y = 2; #10; // Should be 70

x = 1; y = 2; #10; // Should be 80

x = 2; y = 2; #10; // Should be 90

// Test reset behavior

reset = 1; #10;

reset = 0; #10;

// Reapply new pixel values after reset

pixel\_in = 8'd100; valid\_in = 1; x = 0; y = 0; #10;

pixel\_in = 8'd110; valid\_in = 1; x = 1; y = 0; #10;

pixel\_in = 8'd120; valid\_in = 1; x = 2; y = 0; #10;

valid\_in = 0;

// Check output after reset and new input

x = 0; y = 0; #10; // Should be 100

x = 1; y = 0; #10; // Should be 110

x = 2; y = 0; #10; // Should be 120

// End of simulation

$stop;

end

endmodule