EE587 – MILESTONE 1

HARIHARASUDAN R.

E/20/130

SEMESTER 7

13/02/2026

Test Image

This is the image I used to run the tests

Figure 1 - Test Image 128*128

Reference output

This image displays the reference output produced by implementing the Sobel filter algorithm in Python, serving as the high-level language model.

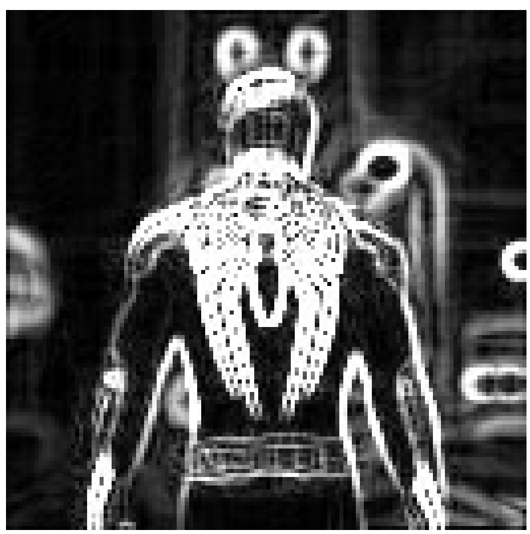

Figure 2 - Reference output

Output

This image displays the reconstructed output generated by the Vivado behavioral testbench simulation.

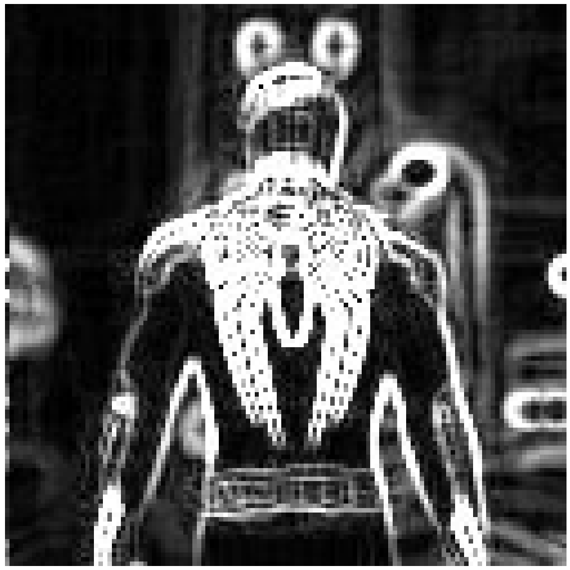

Figure 3 – Output

Source code

```
`timescale 1ns / 1ps
module Sobel (
    input  wire clk,
    input  wire rst,
       // The 3x3 pixel window (8-bit grayscale)
    input  wire [7:0] p11, p12, p13, // Top row
    input  wire [7:0] p21, p22, p23, // Middle row
    input  wire [7:0] p31, p32, p33, // Bottom row
    // Output edge magnitude
    output reg  [7:0] pixel_out
);
    // Gradients require signed arithmetic.
    // The maximum possible value is +/- 1020, so 11 bits are needed to prevent overflow.
    reg signed [10:0] Gx, Gy;
    reg [10:0] abs_Gx, abs_Gy;
    reg [11:0] sum;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            Gx <= 0;
            Gy <= 0;
```

```
            pixel_out <= 0;
        end else begin
            // ------------------------------------------------------
            // Stage 1: Calculate gradients
            // Note: $signed({3'b0, pXX}) zero-extends the 8-bit unsigned
            // pixel to an 11-bit signed value before doing the math.
            // ------------------------------------------------------
            Gx <= ($signed({3'b0, p13}) - $signed({3'b0, p11})) +
                  (($signed({3'b0, p23}) - $signed({3'b0, p21})) <<< 1) +
                  ($signed({3'b0, p33}) - $signed({3'b0, p31}));

            Gy <= ($signed({3'b0, p31}) - $signed({3'b0, p11})) +
                  (($signed({3'b0, p32}) - $signed({3'b0, p12})) <<< 1) +
                  ($signed({3'b0, p33}) - $signed({3'b0, p13}));

            // ------------------------------------------------------
            // Stage 2: Get Absolute Values
            // If the 11th bit (sign bit) is 1, it's negative, so invert it.
            // ------------------------------------------------------
            abs_Gx = (Gx[10]) ? -Gx : Gx;
            abs_Gy = (Gy[10]) ? -Gy : Gy;

            // ------------------------------------------------------
            // Stage 3: Sum and Clip
            // If the sum exceeds the 8-bit maximum (255), clip it to 255.
            // ------------------------------------------------------
            sum = abs_Gx + abs_Gy;
            if (sum > 255)
                pixel_out <= 8'd255;
            else
                pixel_out <= sum[7:0];
        end
    end
endmodule
```

Testbench code

```
`timescale 1ns / 1ps
module tb_Sobel;
    // Parameters for a small test image
    parameter WIDTH = 128;
    parameter HEIGHT = 128;
    parameter IMAGE_SIZE = WIDTH * HEIGHT;
    // Inputs to the DUT (Device Under Test)
    reg clk;
    reg rst;
    reg [7:0] p11, p12, p13;
    reg [7:0] p21, p22, p23;
    reg [7:0] p31, p32, p33;
    // Output from the DUT
    wire [7:0] pixel_out;
    // Memories for file I/O
    reg [7:0] image_in [0:IMAGE_SIZE-1];
    integer file_out;
    integer r, c;
    // Instantiate the Sobel Core
    Sobel uut (
        .clk(clk),
        .rst(rst),
        .p11(p11), .p12(p12), .p13(p13),
        .p21(p21), .p22(p22), .p23(p23),
        .p31(p31), .p32(p32), .p33(p33),
        .pixel_out(pixel_out)
    );
    // Clock generation
    initial begin
        clk = 0;
        forever #5 clk = ~clk; // 10ns period (100MHz)
    end
```

```
// Test sequence
initial begin
    // 1. Initialize and open files
    rst = 1;
    p11=0; p12=0; p13=0;
    p21=0; p22=0; p23=0;
    p31=0; p32=0; p33=0;

    // Load the hex text file generated by Python
    $readmemh("input_image.hex", image_in);
    file_out = $fopen("output_image.hex", "w");

    #20;
    rst = 0;
    #20;
    // 2. Iterate through the image (leaving a 1-pixel border for the 3x3 window)
    for (r = 1; r < HEIGHT - 1; r = r + 1) begin
        for (c = 1; c < WIDTH - 1; c = c + 1) begin

            // Feed the 3x3 window at the rising edge
            @(posedge clk);
            p11 <= image_in[(r-1)*WIDTH + (c-1)];
            p12 <= image_in[(r-1)*WIDTH + c];
            p13 <= image_in[(r-1)*WIDTH + (c+1)];

            p21 <= image_in[r*WIDTH + (c-1)];
            p22 <= image_in[r*WIDTH + c];
            p23 <= image_in[r*WIDTH + (c+1)];

            p31 <= image_in[(r+1)*WIDTH + (c-1)];
            p32 <= image_in[(r+1)*WIDTH + c];
            p33 <= image_in[(r+1)*WIDTH + (c+1)];

            // Wait one cycle for the pipelined math to compute
```

```
            @(posedge clk);

            // Write the resulting pixel to the output file
            $fwrite(file_out, "%02x\n", pixel_out);
          end
        end

        // 3. Close file and finish simulation
        $fclose(file_out);
        $display("Simulation Complete.");
        $finish;
    end

endmodule
```