Hardware Software Co Design

Course Project

Name:	**Md Shahid**
Roll Number:	**M25EEV008**
Program:	**MTECH VLSI**

1 Course Project

1.1 Objective

Implement these 3 image processing techniques-blurring, sharpening, edge detection through hardware software co-design approach

1.2 Components used

1. **Software Tools:**
   - Xilinx Vivado Design Suite (for HDL Simulation)
   - Python 3 (Google Colab/Local Environment)
2. **Python Libraries:**
   - `opencv-python` (Image I/O and Resizing)
   - `numpy` (Array Manipulation)
   - `matplotlib` (Visualization)
3. **Hardware Files:**
   - `image_filter.v` (Accelerator Core)
   - `sobel_tb.v` (Testbench)

1.3 Solution: Step-by-Step Implementation

1.3.1 Step 1: Host Pre-Processing (Python)

The first step involved preparing the input data for the hardware. Since the hardware accelerator is designed for a fixed resolution (100 × 100), the input image had to be resized and converted into a format the Verilog simulator could read (Hexadecimal Memory File).

Actions Taken:

- Loaded the `landscape.jpeg` image using OpenCV.
- Resized it to 100 × 100 pixels.
- Flattened the 2D matrix into a 1D array and wrote each pixel as a 2-digit hex value to `image_in.mem`.

```python
import cv2
import numpy as np

HEX_FILE = "image_in.mem"
IMG_W, IMG_H = 100, 100

def convert_custom_image():
    img = cv2.imread("landscape.jpeg", cv2.IMREAD_GRAYSCALE)
    img_resized = cv2.resize(img, (IMG_W, IMG_H))

    with open(HEX_FILE, 'w') as f:
        for row in range(IMG_H):
            for col in range(IMG_W):
                pixel = img_resized[row, col]
                f.write(f"{pixel:02x}\n")
    print("Created image_in.mem")
```
Listing 1: Python Pre-Processing Script (host_pre.py)

1.3.2 Step 2: Hardware Accelerator Design (Verilog)

The core processing logic was implemented in Verilog. A module named `image_filter` was created to perform 3x3 convolution operations.

Architecture Elaboration: The module operates on a 3×3 pixel window ($p1$ to $p9$).

- **Blur Mode (0):** Calculates the average of all 9 pixels: $Sum/9$.
- **Sharpen Mode (1):** Applies a standard sharpening kernel: Center pixel $\times 5$ minus neighbors.
- **Edge Mode (2):** Implements the Sobel operator by calculating gradients in X and Y directions (G_x, G_y) and computing the approximate magnitude $|G_x| + |G_y|$.

```verilog
module image_filter (
    input [1:0] mode,       // 0=Blur, 1=Sharpen, 2=Edge
    input [7:0] p1, p2, p3, // 3x3 Window Inputs
    input [7:0] p4, p5, p6,
    input [7:0] p7, p8, p9,
    output reg [7:0] out_pixel
);
    parameter THRESHOLD = 30;
    integer sum, sharp_val, gx, gy, mag;

    always @(*) begin
        if (mode == 0) begin // Blur
            sum = p1 + p2 + p3 + p4 + p5 + p6 + p7 + p8 + p9;
            out_pixel = sum / 9;
        end
        else if (mode == 1) begin // Sharpen
            sharp_val = (p5 * 5) - (p2 + p4 + p6 + p8);
            if (sharp_val < 0) out_pixel = 0;
            else if (sharp_val > 255) out_pixel = 255;
            else out_pixel = sharp_val[7:0];
        end
        else begin // Edge
            gx = (p3 + 2*p6 + p9) - (p1 + 2*p4 + p7);
            gy = (p7 + 2*p8 + p9) - (p1 + 2*p2 + p3);
            mag = (gx < 0 ? -gx : gx) + (gy < 0 ? -gy : gy);

            if (mag < THRESHOLD) out_pixel = 0;
            else if (mag > 255) out_pixel = 255;
            else out_pixel = mag[7:0];
        end
    end
endmodule
```
Listing 2: Hardware Accelerator (image_filter.v)

1.3.3 Step 3: Vivado System Design (Proposed)

For physical implementation on the FPGA (e.g., PYNQ-Z2), the design follows the standard Zynq PS-PL Co-Design architecture. **Vivado Block Design Architecture:**

1. **Processing System (PS):** The ARM Cortex-A9 processor executes the Python software. It handles image loading and visualization.

2. **AXI Interconnect:** A standard AXI4-Lite bus connects the PS to the Programmable Logic (PL).

3. **Custom IP (PL):** The `image_filter.v` module is packaged as a custom IP with an AXI4-Lite wrapper.

   - The software writes the 9 input pixels to AXI Registers (Reg0, Reg1, Reg2).
   - The hardware processes the data combinationally.
   - The result is available on AXI Register 3 (Reg3) for the software to read.

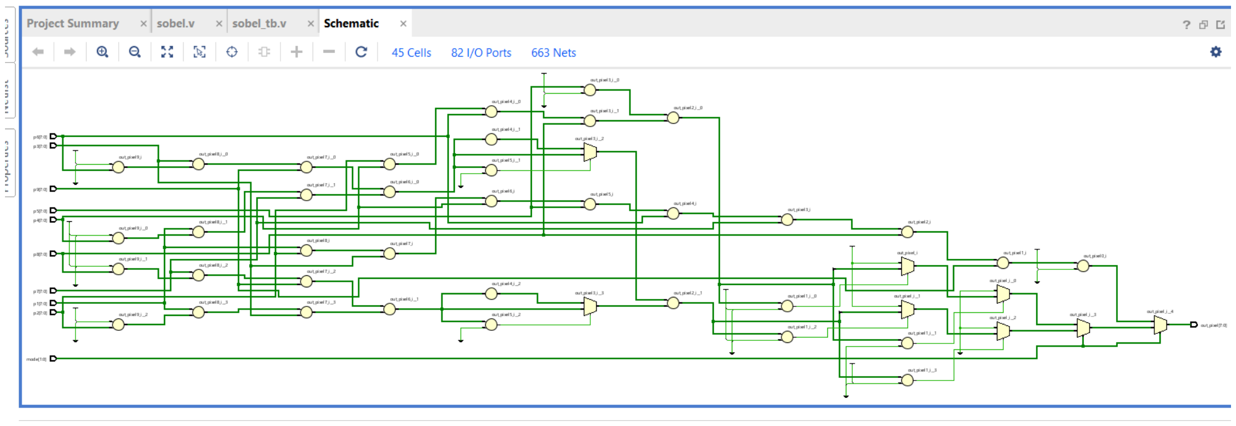


Figure 1: Proposed Vivado Block Design for FPGA Implementation

1.3.4 Step 4: Simulation and Verification

A testbench (`sobel_tb.v`) was created to simulate the entire system. This testbench acts as the "Host Processor" driving the accelerator.

Simulation Steps Performed:

1. Loaded `image_in.mem` into the testbench memory array.

2. Opened a single output file `combined_output.mem`.

3. **Pass 1 (Blur):** Set `mode=0` and processed the entire image.

4. **Pass 2 (Sharpen):** Set `mode=1` and processed the image again.

5. **Pass 3 (Edge):** Set `mode=2` and processed the image a final time.

6. Appended all results into the single output file.

1.3.5 Step 5: Post-Processing and Visualization (Python)

The final step involved verifying the hardware output. A Python script was used to read the `combined_output.mem` file and split the data into individual images for visualization.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os

OUTPUT_FILE = "combined_output.mem"
INPUT_FILE = "landscape.jpeg"
IMG_W = 100
IMG_H = 100

def view_combined_results():
    if not os.path.exists(OUTPUT_FILE):
        return

    with open(OUTPUT_FILE, 'r') as f:
        lines = f.readlines()

    pixels = [int(line.strip(), 16) for line in lines if line.strip()]
    target_h = IMG_H - 2
    target_w = IMG_W - 2
    one_img_size = target_h * target_w

    titles = ["Blur", "Sharpen", "Edge Detection"]

    plt.figure(figsize=(20, 5))

    plt.subplot(1, 4, 1)
    if os.path.exists(INPUT_FILE):
        img_in = cv2.imread(INPUT_FILE, cv2.IMREAD_GRAYSCALE)
        if img_in is not None:
            img_in = cv2.resize(img_in, (IMG_W, IMG_H))
            plt.imshow(img_in, cmap='gray')
            plt.title("Original Input")
    plt.axis('off')

    for i in range(3):
        start_idx = i * one_img_size
        end_idx = start_idx + one_img_size
        slice_pixels = pixels[start_idx:end_idx]

        if len(slice_pixels) < one_img_size:
            slice_pixels += [0] * (one_img_size - len(slice_pixels))

        try:
            img_out = np.array(slice_pixels, dtype=np.uint8).reshape((
    target_h, target_w))
            fname = f"output_{titles[i].lower().replace(' ', '_')}.jpg"
            cv2.imwrite(fname, img_out)

            plt.subplot(1, 4, i+2)
```

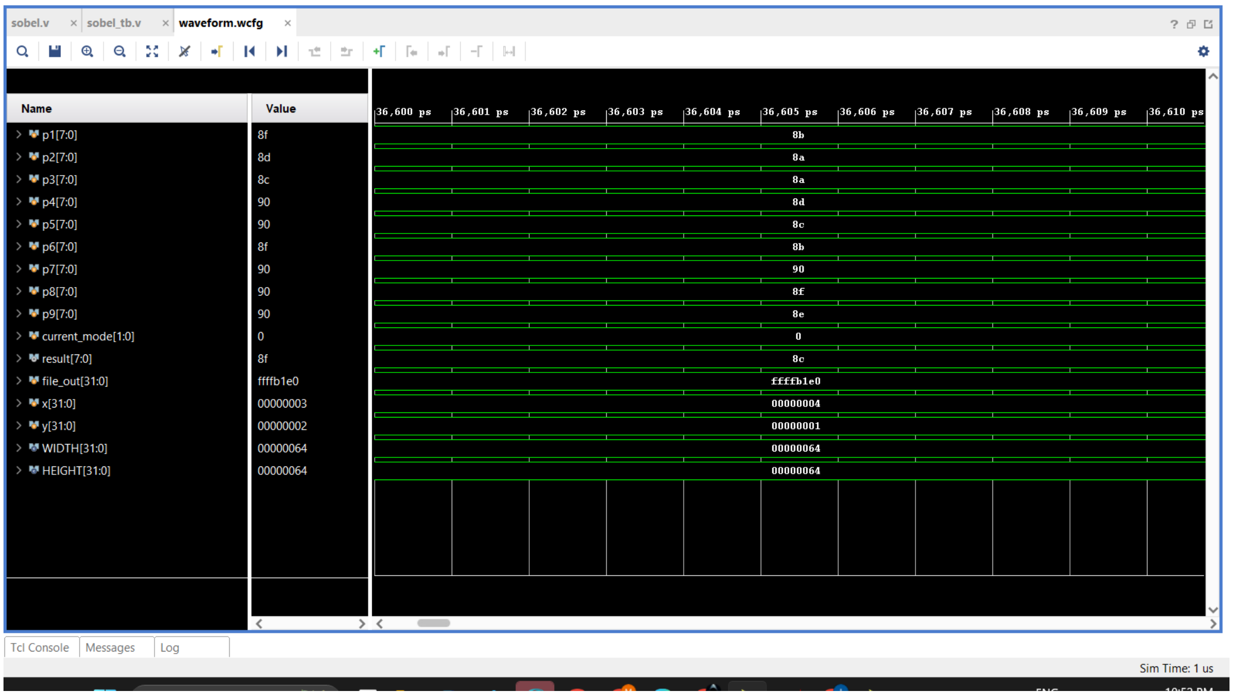
```python
                plt.imshow(img_out, cmap='gray')
                plt.title(f"Hardware: {titles[i]}")
                plt.axis('off')
        except Exception as e:
            print(f"Error processing image {i}: {e}")

    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    view_combined_results()
```

Listing 3: Python Visualization Script (host_post.py)

1.4 Results

1.4.1 Simulation Waveform

The waveform below captures the signal transitions during the simulation. It shows the `current_mode` switching and the `result` output changing as the pixel window (`p1-p9`) is updated.



Name	Value	
p1[7:0]	8f	8b
p2[7:0]	8d	8a
p3[7:0]	8c	8a
p4[7:0]	90	8d
p5[7:0]	90	8c
p6[7:0]	8f	8b
p7[7:0]	90	90
p8[7:0]	90	8f
p9[7:0]	90	8e
current_mode[1:0]	0	0
result[7:0]	8f	8c
file_out[31:0]	ffffb1e0	ffffb1e0
x[31:0]	00000003	00000004
y[31:0]	00000002	00000001
WIDTH[31:0]	00000064	00000064
HEIGHT[31:0]	00000064	00000064

Figure 2: Vivado Simulation Waveform showing valid signal processing

1.4.2 Visual Output Verification

The hardware successfully processed the landscape image. The Blur filter smoothed the details, the Sharpen filter enhanced texture, and the Edge Detection filter correctly identified the boundaries of the mountains and trees (white lines on black background).

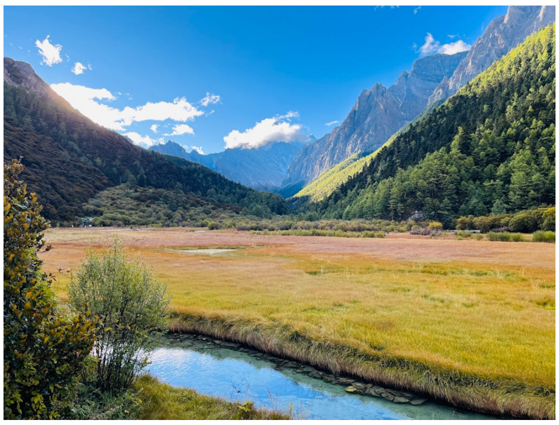


Figure 3: Original Input

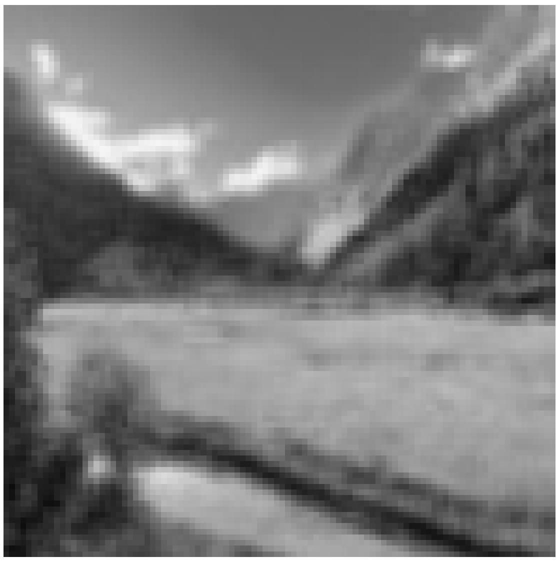


Figure 4: Hardware Blur

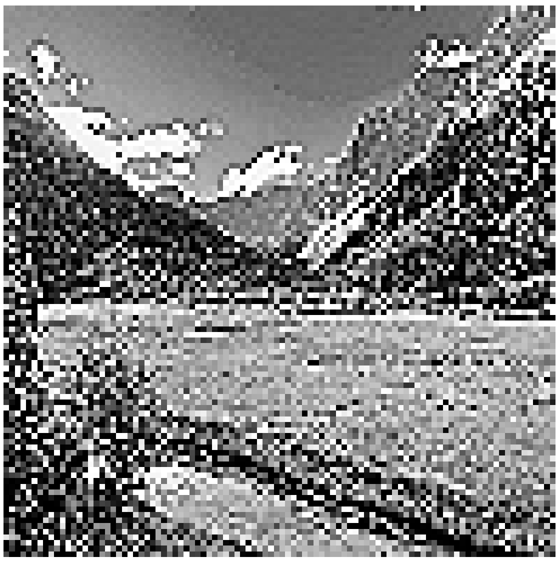


Figure 5: Hardware Sharpen

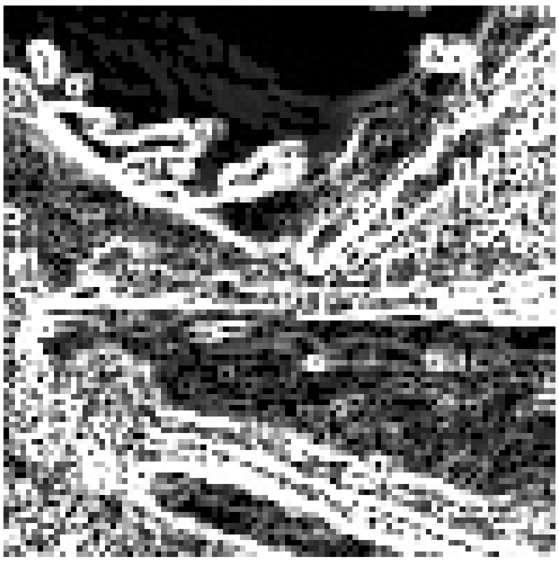


Figure 6: Hardware Edge Detection