SRI HARI A S
sriharias2204@gmail.com

FPGA-Optimized Otsu Thresholding Algorithm

Abstract :

This report presents a high-performance, synthesizable System-Verilog implementation of Otsu's Thresholding Algorithm. By utilizing a Finite State Machine (FSM) and optimized arithmetic units, the design achieves bit-perfect accuracy compared to software-based models while maintaining a low resource footprint.

Introduction to Otsu's Algorithm :

Otsu's method is used to perform automatic image thresholding. It involves:

1. Calculating the histogram of the image.

2. Iterating through all possible threshold values (0-255).

3. Maximizing the between-class variance (σ_b^2):

$$N = \text{total pixels}$$

$$S = \sum i \cdot h(i) \text{(total image intensity)}$$

$$w_B = \sum_{i \le t} h(i)$$

$$s_B = \sum_{i \le t} i \cdot h(i)$$

Then:

$$\mu_B = \frac{s_B}{w_B}, \mu_F = \frac{S - s_B}{N - w_B}$$

Plug into Otsu:

$$\sigma^2(t) = w_B(N - w_B)\left(\frac{s_B}{w_B} - \frac{S - s_B}{N - w_B}\right)^2$$

Hardware Architecture :

The design is implemented using a 6-state FSM:

-**IDLE:** Wait for start signal.

- **CLEAR:** Initialize Histogram RAM to zero.

- **BUILD:** Stream pixels and increment histogram bins.

- **SUM:** Calculate total intensity sum of the image.

- **OTSU:** Iterate 256 cycles to find the maximum variance.

- **DONE:** Assert completion and output the threshold.

Key Implementation Challenges :

Arithmetic Overflow: Standard 32-bit registers fail during the squaring of variance components. The design utilizes 48-bit intermediate products and 64-bit metric registers.

Normalization: To avoid 90-bit multipliers, the variance numerator is scaled via bit-shifting ($>> 12$) before squaring, preserving the "peak" of the function while fitting within hardware limits.

Timing Alignment: A delayed address register (`addr_delayed`) was implemented to synchronize the FSM's threshold index with the non-blocking updates of the weight and sum registers.

Verification Methodology :

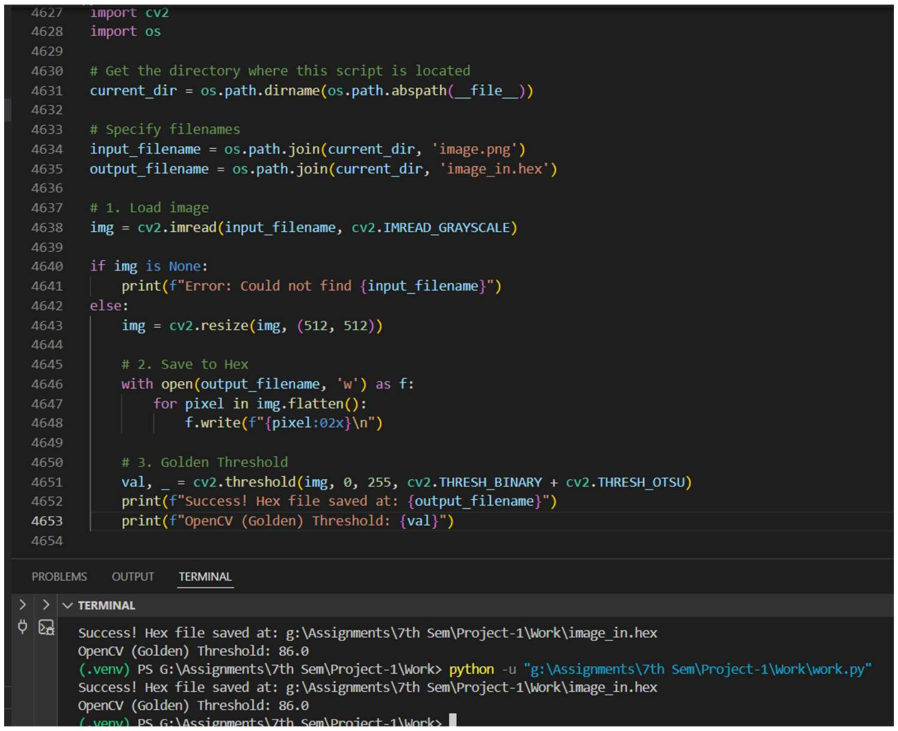
Verification was conducted using a co-simulation approach:

- **Software Model:** Python + OpenCV served as the "Golden Reference."

- **Data Conversion:** Images were flattened into hexadecimal format using a custom script.

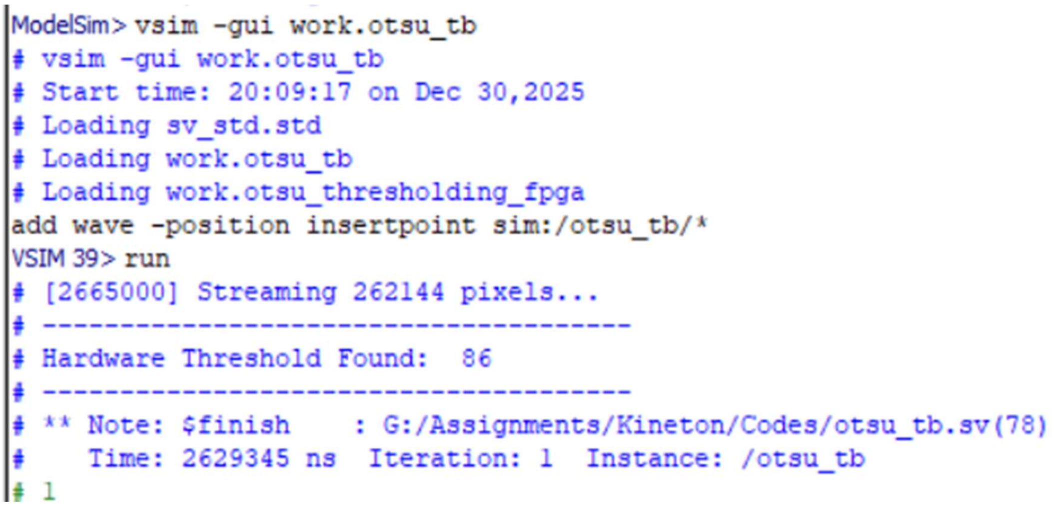
- **RTL Simulation:** The System-Verilog testbench processed a 512 - times - 512 - image, asserting the `done` signal upon calculation completion.

Results :

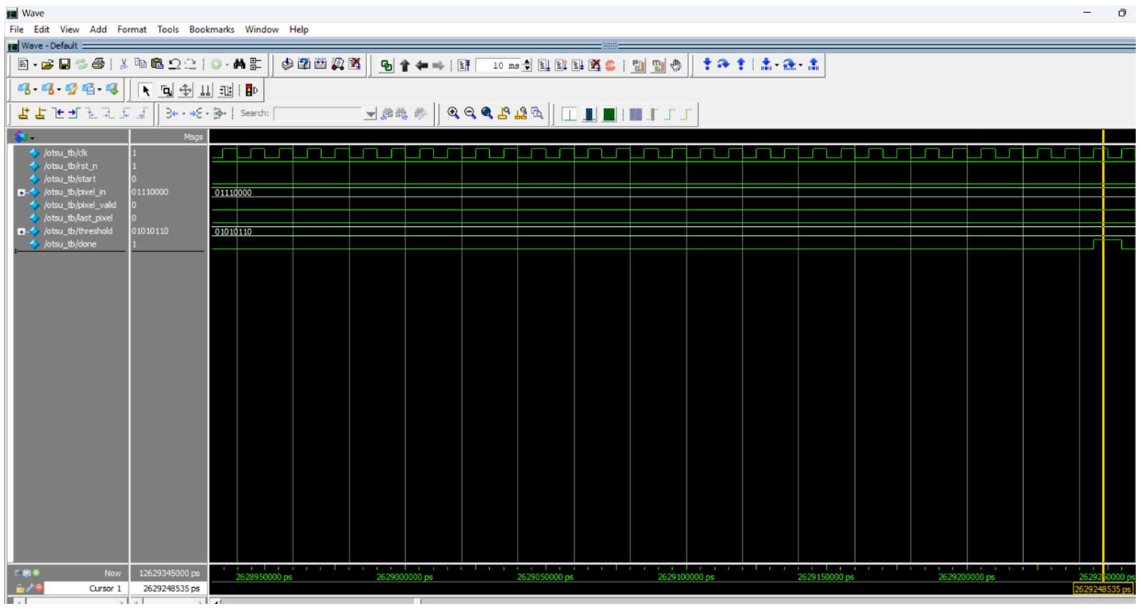


```python
import cv2
import os

# Get the directory where this script is located
current_dir = os.path.dirname(os.path.abspath(__file__))

# Specify filenames
input_filename = os.path.join(current_dir, 'image.png')
output_filename = os.path.join(current_dir, 'image_in.hex')

# 1. Load image
img = cv2.imread(input_filename, cv2.IMREAD_GRAYSCALE)

if img is None:
    print(f"Error: Could not find {input_filename}")
else:
    img = cv2.resize(img, (512, 512))

    # 2. Save to Hex
    with open(output_filename, 'w') as f:
        for pixel in img.flatten():
            f.write(f"{pixel:02x}\n")

    # 3. Golden Threshold
    val, _ = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    print(f"Success! Hex file saved at: {output_filename}")
    print(f"OpenCV (Golden) Threshold: {val}")
```

```
Success! Hex file saved at: g:\Assignments\7th Sem\Project-1\Work\image_in.hex
OpenCV (Golden) Threshold: 86.0
(.venv) PS G:\Assignments\7th Sem\Project-1\Work> python -u "g:\Assignments\7th Sem\Project-1\Work\work.py"
Success! Hex file saved at: g:\Assignments\7th Sem\Project-1\Work\image_in.hex
OpenCV (Golden) Threshold: 86.0
(.venv) PS G:\Assignments\7th Sem\Project-1\Work>
```



```
ModelSim> vsim -gui work.otsu_tb
# vsim -gui work.otsu_tb
# Start time: 20:09:17 on Dec 30,2025
# Loading sv_std.std
# Loading work.otsu_tb
# Loading work.otsu_thresholding_fpga
add wave -position insertpoint sim:/otsu_tb/*
VSIM 39> run
# [2665000] Streaming 262144 pixels...
# ------------------------------------
# Hardware Threshold Found:  86
# ------------------------------------
# ** Note: $finish    : G:/Assignments/Kineton/Codes/otsu_tb.sv(78)
#    Time: 2629345 ns  Iteration: 1  Instance: /otsu_tb
# 1
```



```
/otsu_tb/clk          1
/otsu_tb/rst_n        1
/otsu_tb/start        0
/otsu_tb/pixel_in     01110000
/otsu_tb/pixel_valid  0
/otsu_tb/last_pixel   0
/otsu_tb/threshold    01010110
/otsu_tb/done         1

Now       12629345000 ps
Cursor 1  2629248535 ps
```

Conclusion :

The implementation successfully finds the optimal threshold for various bimodal images. The design is ready for integration into larger Image Processing pipelines for ASICs or FPGAs.