IC Course – Third evaluation

**SYED MAARIJ**

**Roll No: CS-21022**

**Abstract:**

This project focuses on the design and verification of a low-power pixel operator module for image processing, implemented in Verilog. The system is optimized for dynamic power reduction using clock gating and functional modularity. The design processes pixel data to perform operations such as brightness adjustment, thresholding, and inversion, based on a selection signal. Verification and analysis were conducted using a comprehensive testbench that evaluates functionality, power efficiency, and overall design robustness. Results confirm the module’s correctness and low-power characteristics.

**Introduction:**

Low-power image processing plays a significant role in modern embedded systems, particularly for battery-powered devices. The primary goal of this project is to develop an efficient pixel operator capable of performing multiple image processing tasks with optimized power consumption. The implementation uses clock gating to minimize unnecessary switching activity and employs modular design principles to facilitate debugging and scalability. Verification includes simulation-based functional and power analysis.

**High-level System Architecture:**

**Image Processing Workflow:**

Due to limitations in Vivado, where direct image processing support is unavailable, a Python script was developed to bridge the gap. The workflow includes:

1. **Hexadecimal Conversion:**
   * A Python script reads an input image and converts it into a hexadecimal .txt file, which serves as the input to the Verilog pipeline.
2. **Pipeline Processing:**
   * The Verilog design processes the hexadecimal data and generates an output .txt file in hexadecimal format.
3. **Image Reconstruction**:
   * Another Python script takes the output hexadecimal .txt file and converts it back into an image format, enabling visual verification of the results.

This approach ensures seamless integration of image data with the Verilog pipeline while providing flexibility for functional validation.

**System Architecture:**

**Overview:**

The pixel operator is a modular design implemented in Verilog, comprising the following components:

**Input Module:** Accepts pixel data (inbyte) and control signals (threshold, value, select).

**Processing Module:** Executes one of four operations (brightness increase, brightness decrease, thresholding, or inversion) based on the select signal.

**Clock Gating Module:** Enables clock signals only when processing is necessary, reducing dynamic power consumption.

**Pipeline Registers**: Use D flip-flops to ensure proper data propagation and synchronization.

**Output Module:** Outputs the processed pixel value (outbyte).

**Design Details:**

**Functional Modules:**

There are the following functional modules in the code:

1. **Processing Logic:**
   * select = 2'b00: Brightness increase (inbyte + value).
   * select = 2'b01: Brightness decrease (inbyte - value).
   * select = 2'b10: Thresholding (inbyte > threshold ? 8'hFF : 8'h00).
   * select = 2'b11: Pixel inversion (~inbyte).
2. **Clock Gating:**
   * A latch-based clock gating circuit ensures that the processing logic operates only when input pixel data changes, reducing switching activity.
3. **Pipeline Register:**
   * D flip-flops are used to store intermediate results and synchronize data flow.

#### **Enhanced Features:**

* **Reset Signal:**  Ensures clean initialization of all registers.
* **Dynamic Power Analysis:** Tracks switching events in the test bench for normalized dynamic power measurement.

**Verification and Testing:**

**Testbench Design:**

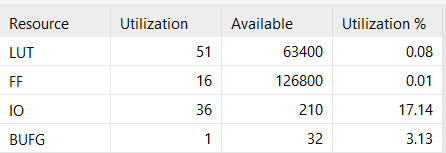
A comprehensive testbench was developed to verify the functionality and power efficiency of the pixel operator module. The testbench includes:

1. **Clock Generation:** 10 ns clock cycle.
2. **Functional Verification:** Validates output correctness for all select operations.
3. **Power Analysis:** Measures switching activity and calculates dynamic power consumption.
4. **Image Processing Simulation:** Processes a 98304-pixel image, with results compared against expected values.

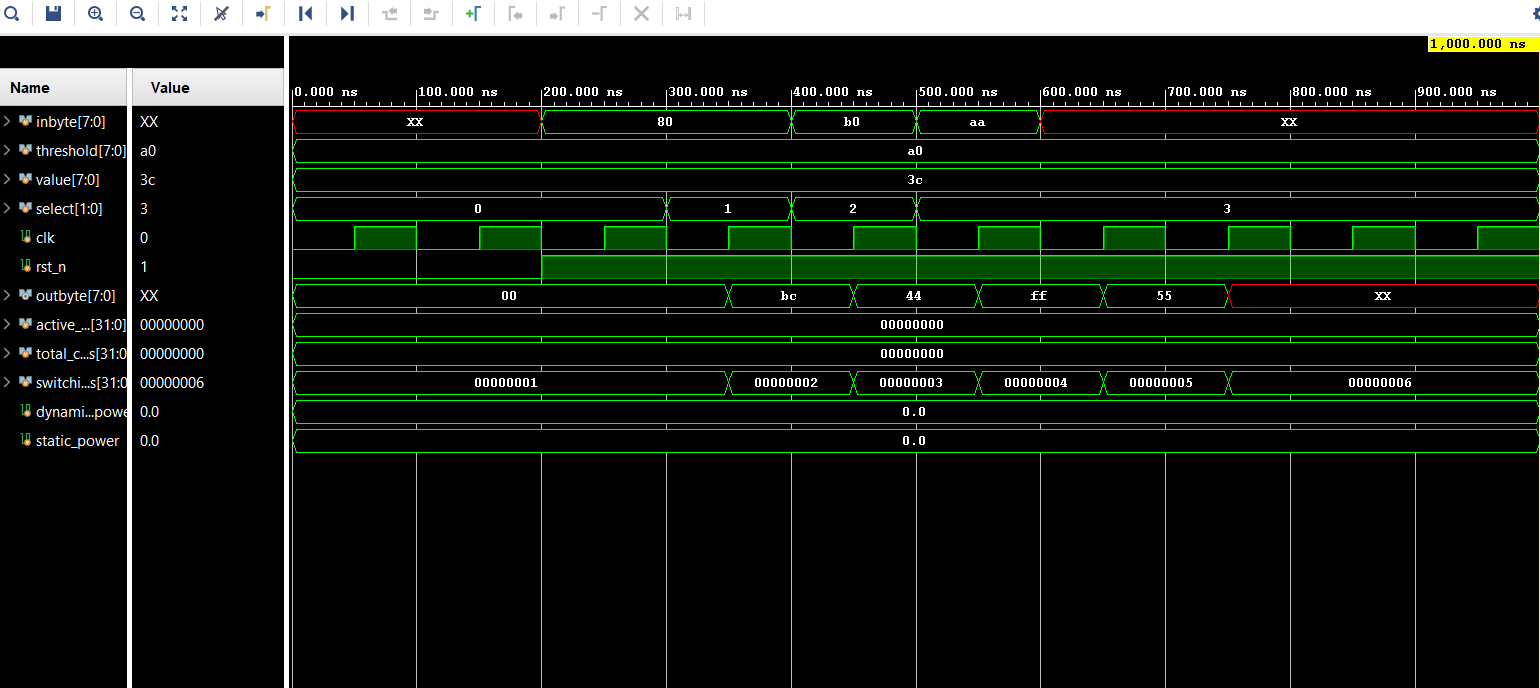
**Results**

1. **Functional Verification:**
   * All operations produced correct outputs for various test cases.
   * **Examples:**
     + **Brightness Increase:** Input 0x80, Value 0x3C, Output 0xBC.
     + **Thresholding:** Input 0xB0, Threshold 0xA0, Output 0xFF.
2. **Power Analysis:**
   * **Normalized Dynamic Power:** 4.677
   * S**tatic Power**: 0.155
   * **Total Power:** 4.792 W
3. **Image Processing:**
   * Input image data processed successfully.
   * Output data matched expected results.

**Resource Utilization**



**Simulation Result:**

****

**Output Results:**

* 1. **Actual Output :**



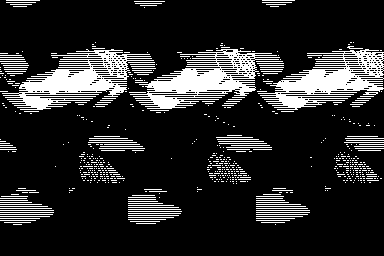
* 1. **Brightness Increase**

1. 

**3) Brightness Decrease:**



**4) Threshold Increase:**



**5) Invert:**



**Codes:**

**Hex\_to\_image\_converter**

import numpy as np

from PIL import Image

with open('select=3.txt', 'r') as file:

    data = file.read()

pixel\_values = [int(value, 16) for value in data.split()]

width = 256

height = 256

image\_array = np.array(pixel\_values, dtype=np.uint8).reshape((height, width))

image = Image.fromarray(image\_array)

image.save('select=3output.png')

print("Image saved as select=3output.png")

**Image\_to\_Hex\_Converter**

from PIL import Image

import numpy as np

image\_path = 'output\_image.png'

output\_file = 'output\_hex.txt'

image = Image.open(image\_path).convert('L')

image = image.resize((128, 128))

image\_array = np.array(image)

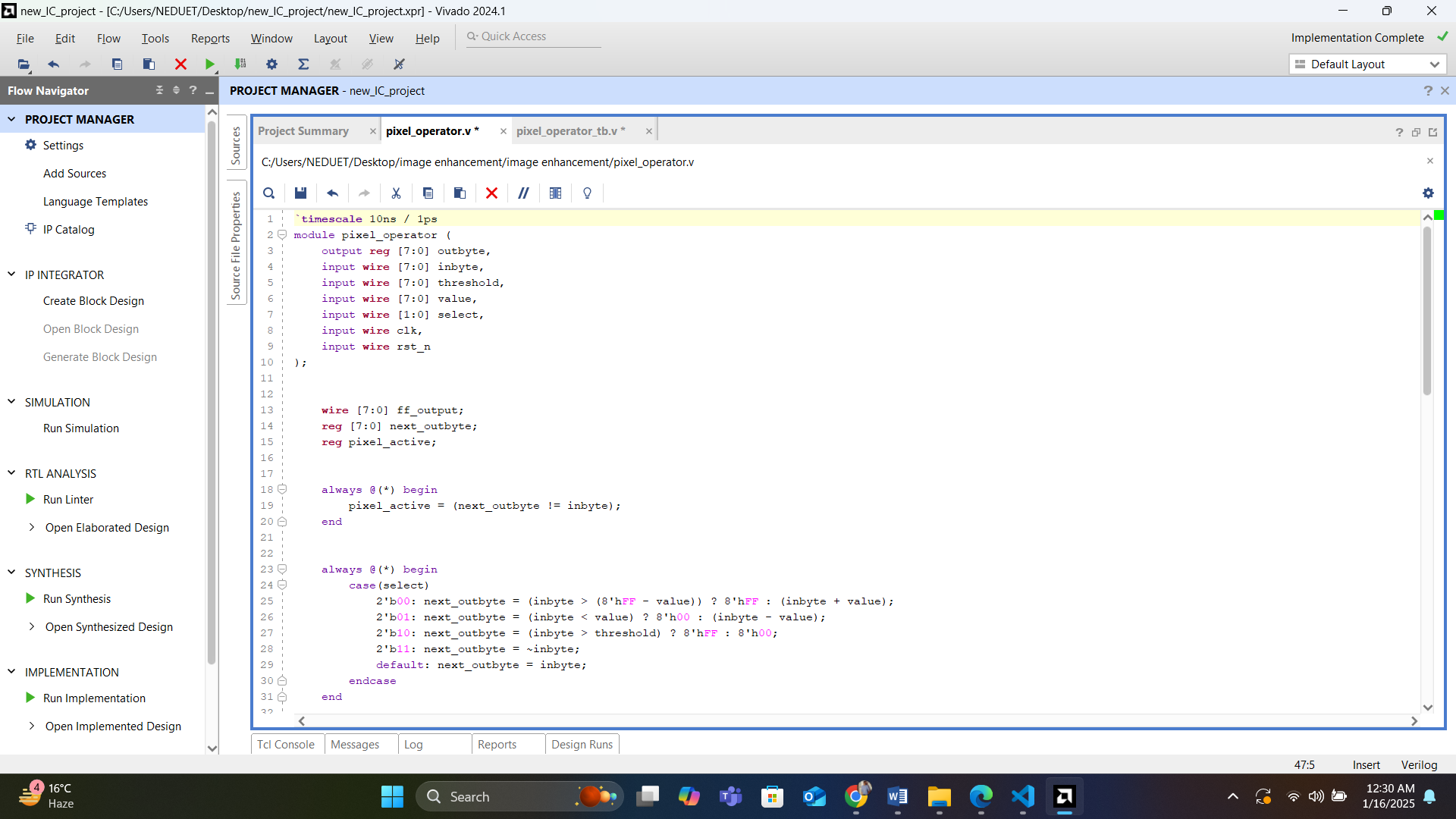
pixel\_values = image\_array.flatten()

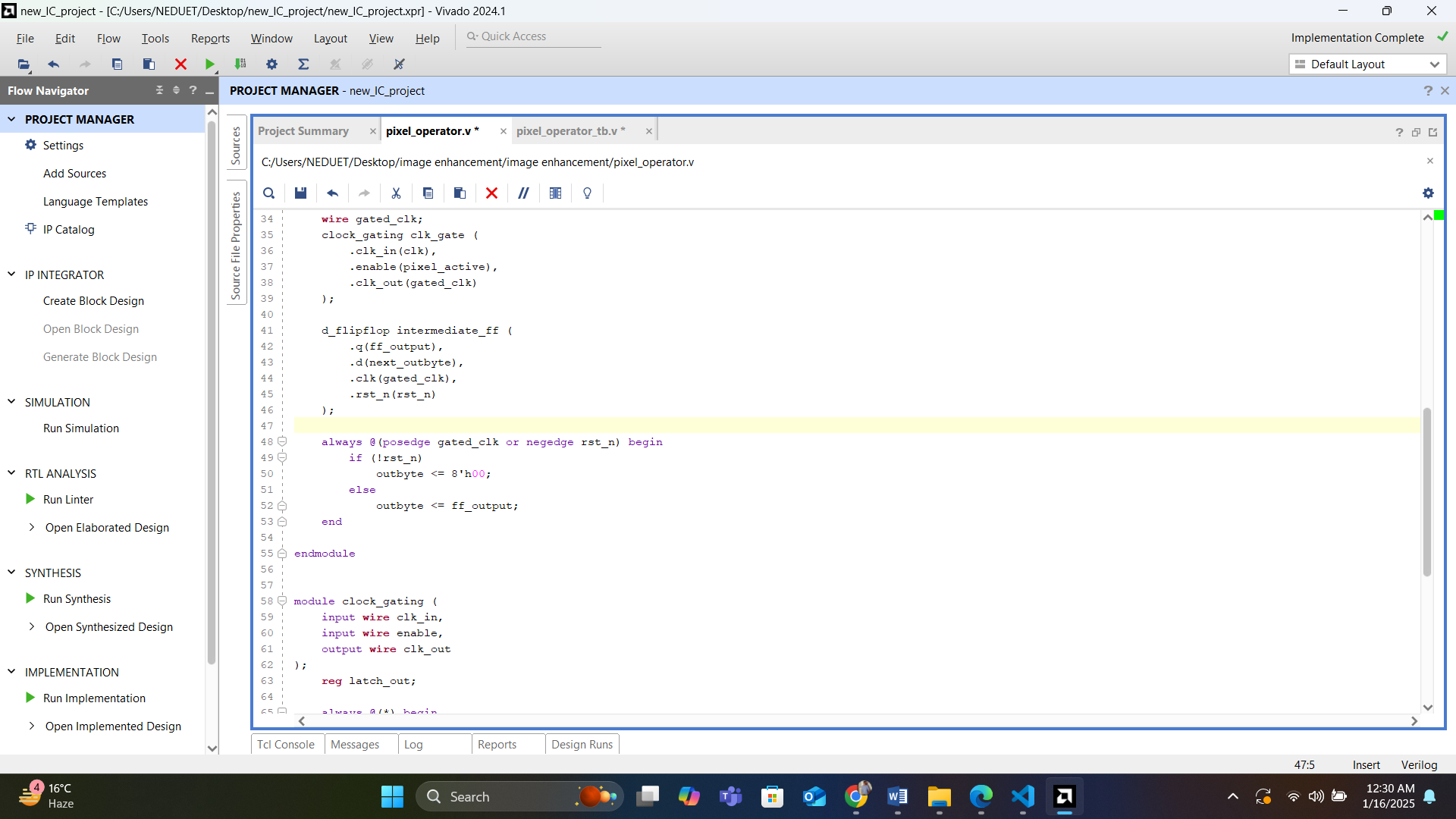
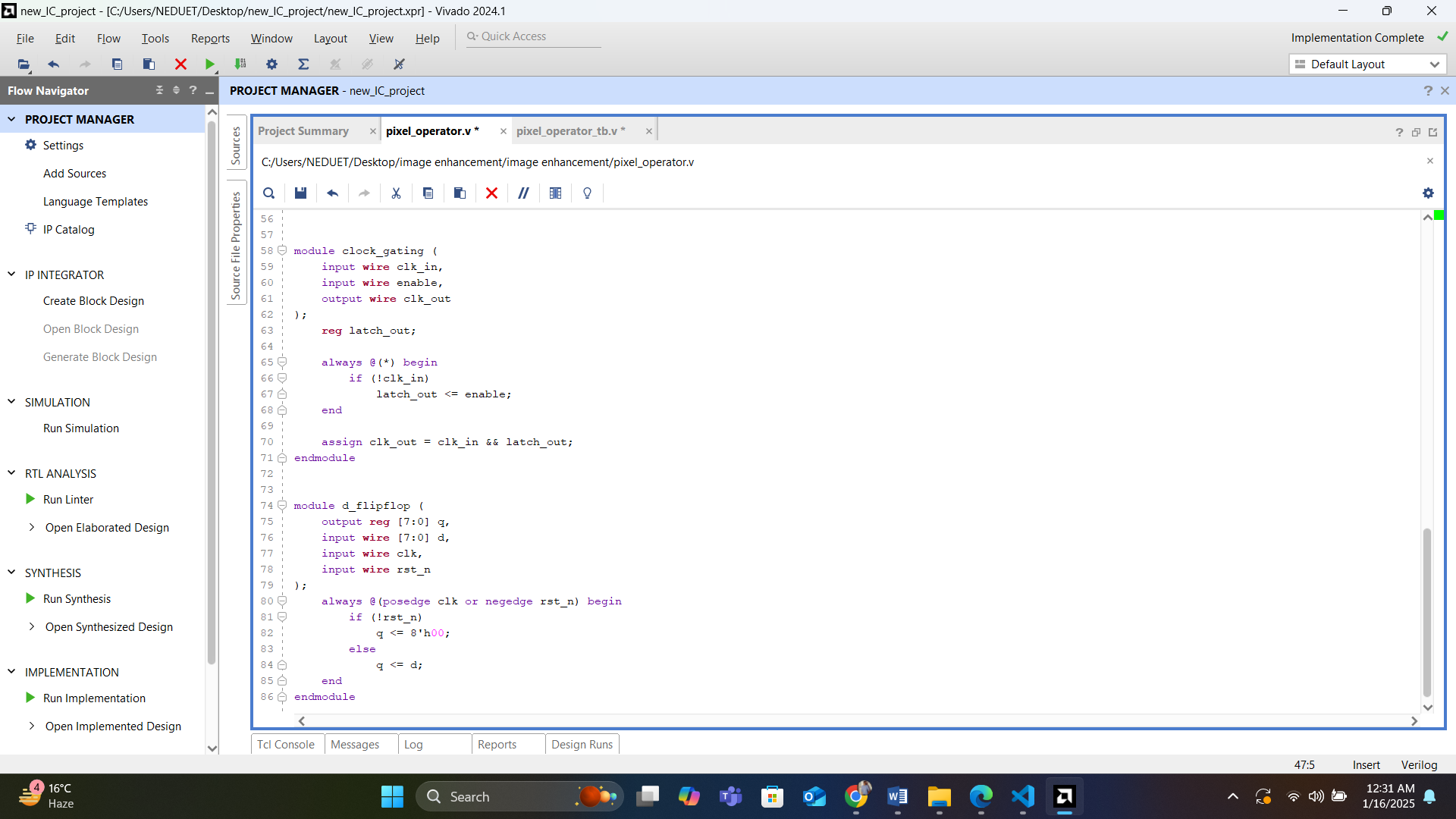
hex\_values = [format(value, '02x') for value in pixel\_values]

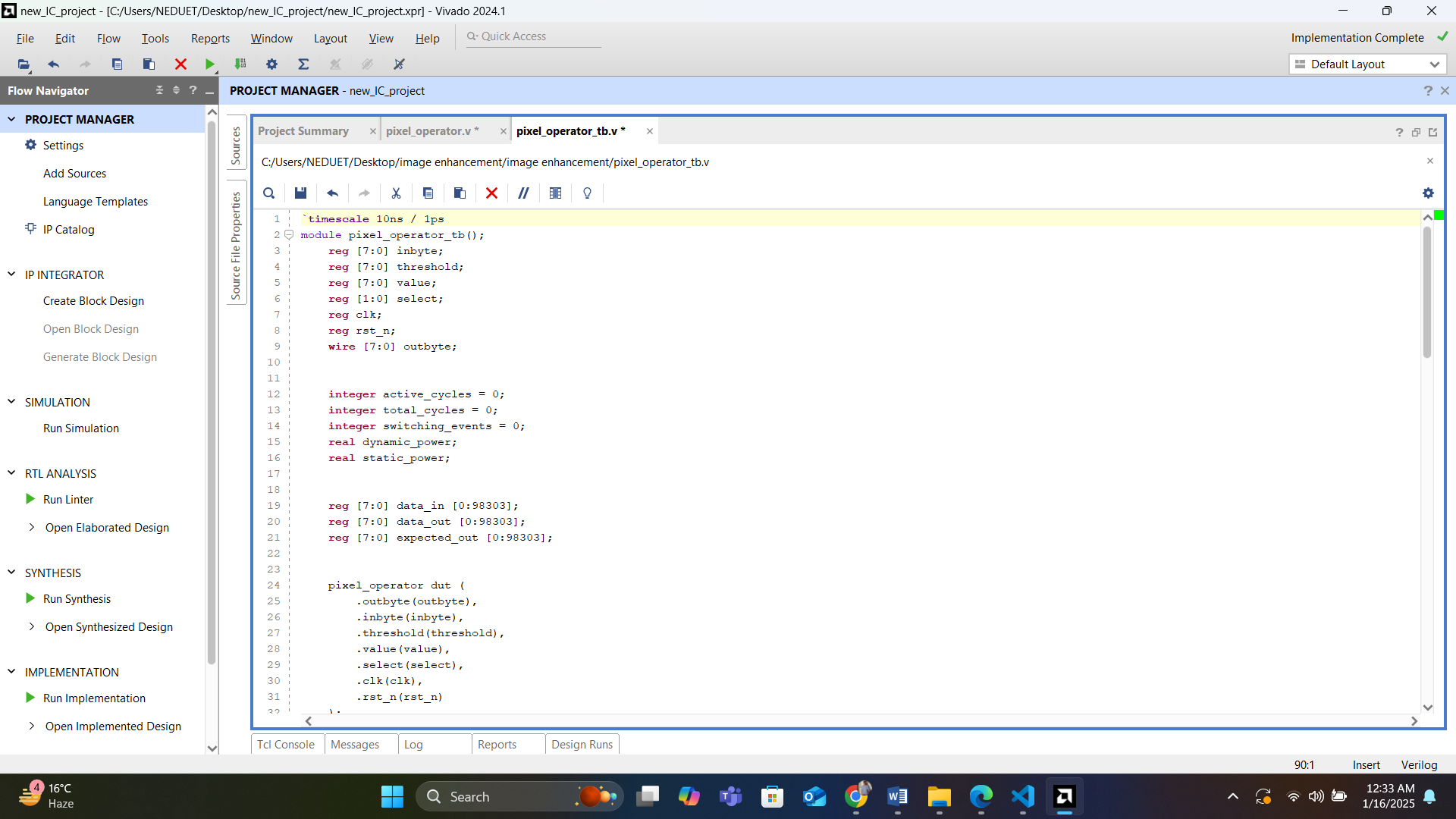
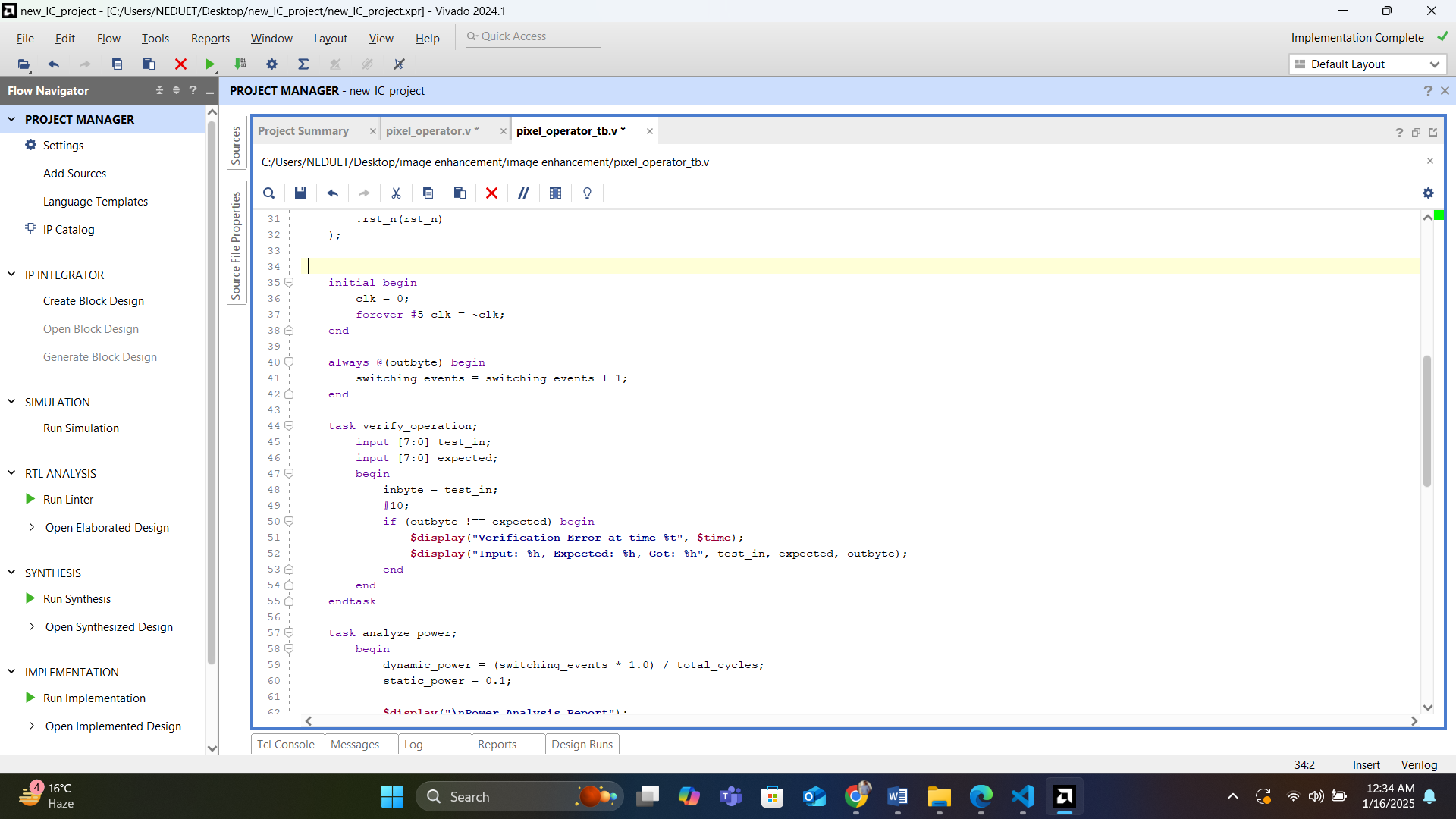
with open(output\_file, 'w') as file:

    file.write(' '.join(hex\_values))

print(f"Hexadecimal values saved to {output\_file}")

**Module Code**



**Testbench Code**

