Assignment 4: Edge Detection

Edge Detection

In this assignment you will be implementing a Canny edge-detection algorithm in RTL, which is used in many computer vision applications. You will be implementing a streaming architecture using FIFOs and evaluating the throughput performance of the algorithm in terms of frames per second.

In this implementation, we will only focus on 2 of the 5 stages: grayscale conversion and sobel filter. Begin by compiling and running the edgedetect.c code provide to you. You may compile the code and select an input image to generate the output image at each of the 5 stages. However, you will need to disable the stages we will not be implementing, as follows:

1. In the **main** function, comment out the following function calls : **gaussian_blur**, **non_maximum_suppressor**, and **hysterisis_filter**.
2. In the **main** function, comment out the **write_grayscale_bmp** function calls immediately after the function calls listed above.
3. Change the input data to **sobel_filter** from **gb_data** to **gs_data**.

Goals:

- Create a streaming architecture that passes the image data from one block to another through FIFOs
- Create a SystemVerilog architecture for the grayscale and sobel functions
- Implement a shift-register buffer for the Sobel filter operations on a 3x3 window.
- Implement the UVM simulation architecture to verify your design is bit-true accurate.
- Report on performance results and improvements

Implementation:

- Compile and run the C-code to generate the input and output images for one image
    - **gcc edgedetect.c -o edgedetect**
    - **./edgedetect image.bmp**

- Run the grayscale UVM model to test and validate your design in simulation to make sure you understand how the UVM system works.
- Once you have verified your grayscale function works correctly, add the sobel filter.
- Develop a way of caching the rows of data required to do a sobel box operation on the 8 neighboring pixels. The sobel function should be its own entity.
- Use a loop so the operations in sobel loops generates the output in 1 cycle.
- Consider how you might pass image border pixels, which are not computed, to the output FIFO

Verification:

- Implement the UVM simulation architecture that reads BMP images files and write them to input FIFOs. Likewise, it should read the image from the output FIFO and compare results against the known data from your C program. You should do a byte-wise comparison in simulation.
- The testbench modules should be relatively similar to the UVM demo provided to you.
- Create a edge_detect_sim.do file that compiles the SystemVerilog and UVM files, sets up the wave form, and runs the simulation.
- Run the simulation in modelsim and verify the design is bit-true accurate.
- Compile the design in Synplify Premier to get high-level resource results. You don't need to go through Place and Route. You should not synthesize the UVM files.

Reporting:

- Submit a PDF file with simulation and synthesis results with the following:
  - Simulation results:
    - Clock cycle count
    - Errors reported (if any)
  - Synthesis results (some might not apply):
    - Maximum frequency
    - Registers / LUTs / Logic Elements
    - Memory utilization
    - Multipliers (DSPs)
    - Worst path (timing analysis)
    - Schematic architecture (RTL)
    - Performance / Speedup
    - Throughput (Frames-per-second)

- Include pictures and a brief description of your architecture from Synplify Premier.

Turn in your designs with the report. Your file should be zipped, and should include the SystemVerilog files, synthesis, simulation, and input/output files. **PLEASE MAKE SURE TO REMOVE ALL THE INTERMEDIATE WORK DIRECTORIES.**