# HLS-VL602

# FINAL PROJECT

IMPLEMENTING EDGE DETECTION
ON FPGA USING SOBEL FILTER

**Pratyush Nandi – IMT2017518**
**Navneet Kour – MT2020516**

# Contents

# 1 Edge Detection

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges. The same problem of finding discontinuities in one-dimensional signals is known as step detection and the problem of finding signal discontinuities over time is known as change detection. Edge detection is a fundamental tool in image processing, machine vision and computer vision, particularly in the areas of feature detection and feature extraction. The architecture used in our project is shown in figure 1.

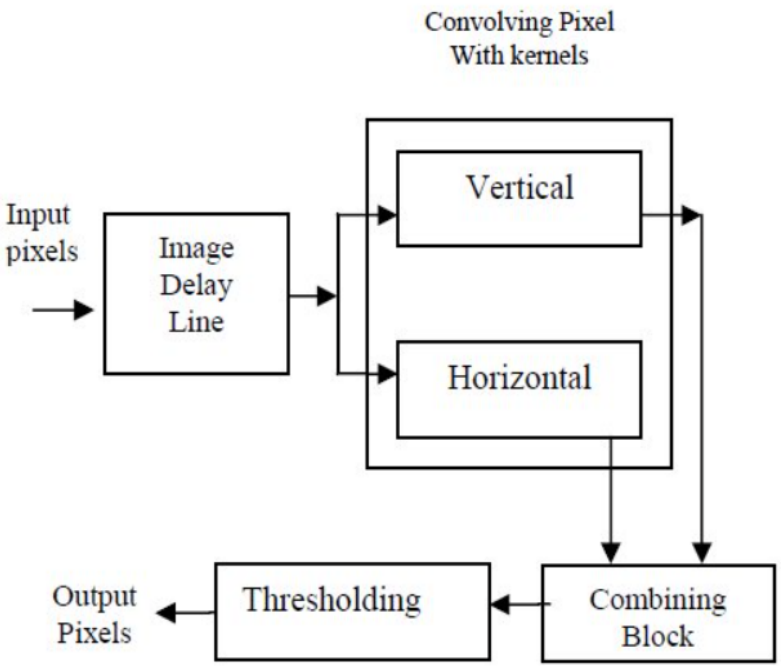


Figure 1: Edge detection architecture

## 1.1 Sobel Filter

The Sobel filter is used for edge detection. It works by calculating the gradient of image intensity at each pixel within the image. It finds the direction of the largest increase from light to dark and the rate of change in that direction. The result shows how abruptly or smoothly the image changes at each pixel, and therefore how likely it is that that pixel represents an edge. It also shows how that edge is likely to be oriented. The result of applying the filter to a pixel in a region of constant intensity is a zero vector. The result of applying it to a pixel on an edge is a vector that points across the edge from darker to brighter values.

| | | |
|---|---|---|
| -1 | 0 | +1 |
| -2 | 0 | +2 |
| -1 | 0 | +1 |

Gx

| | | |
|---|---|---|
| +1 | +2 | +1 |
| 0 | 0 | 0 |
| -1 | -2 | -1 |

Gy

Figure 2: Gx and Gy matrices

The sobel filter uses two 3 x 3 kernels. One for changes in the horizontal direction, and one for changes in the vertical direction. The two kernels are convolved with the original image to calculate the approximations of the derivatives. If we define G_x and G_y as two images that contain the horizontal and vertical derivative approximations respectively. The G_x and G_y matrices can be seen from the figure 2.

The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these G_x and G_y). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by: $\sqrt{G_x^2 + G_y^2}$.

# 2 MATLAB Implementation

We used *MATLAB* as golden reference and verification purposes. The code was taken from a reference(mentioned in reference section). The snippet of the code is attached in figure 3.

```
Mx = [-1 0 1; -2 0 2; -1 0 1];
My = [-1 -2 -1; 0 0 0; 1 2 1];

% Edge Detection Process
% When i = 1 and j = 1, then filtered_image pixel
% position will be filtered_image(2, 2)
% The mask is of 3x3, so we need to traverse
% to filtered_image(size(input_image, 1) - 2
%, size(input_image, 2) - 2)
% Thus we are not considering the borders.
for i = 1:size(input_image, 1) - 2
    for j = 1:size(input_image, 2) - 2

        % Gradient approximations
        Gx = sum(sum(Mx.*input_image(i:i+2, j:j+2)));
        Gy = sum(sum(My.*input_image(i:i+2, j:j+2)));

        % Calculate magnitude of vector
        filtered_image(i+1, j+1) = sqrt(Gx.^2 + Gy.^2);

    end
end
```

Figure 3: MATLAB code snippet

Here we can see that there 2 nested loops. The use of these loops are to cover the whole 2-D matrix of the image and leave the edges. Inside this loop we are doing element-wise matrix multiplication of a 3X3 matrix with a sobel mask. The output of the code can be seen in figure 4. The edge detection was applied on the butterfly image and we can see that edge detection gives us good result.

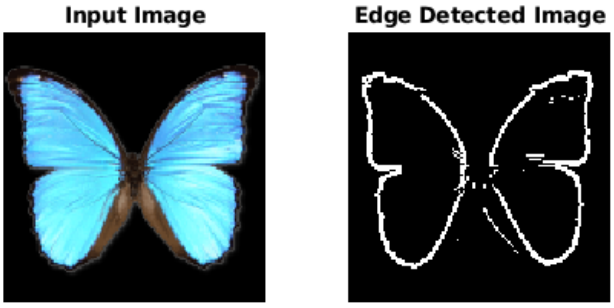


Figure 4: Output from MATLAB code

# 3 Verilog Implementation

The MATLAB code was taken as foundation code. The hardware implementation was done on Xillinx platform. The synthesizable code was burnt on to a **basys3** fpga board.

## 3.1 Functional Description & Hierarchy

The code is divided into several modules for making it more understandable and feasible. The functional description of the modules are as following:

- *topcontroller*: This module is the central module of our code. It calls other modules such as Uart and sobel. This code initializes registers, matrices and blockrams. The crux of the code lies in the FSM that we have designed. The 4 state FSM first receives data from pc through the UART and fills it in the image blockram. Then next state uses a sophisticated algorithm to extract a sub-matrix from a single dimension blockram. The next state uses a trigger flag(received from sobel module) to store the data received into filtered image blockram which is initialized for filtered image after sobel module sends out the computed data. The last state sends the data to pc through UART.
- *sobel*: This module performs a 3X3 element-wise matrix multiplication. We used a optimizing method of loop unrolling to reduce the delay in matrix multiplication. We also used a floating point ip to calculate square root.
- *Uart*: This module is mainly for transferring and receiving data from pc to fpga and viceversa.

The above hierarchy can be better understood by the flow chart shown in figure 5.

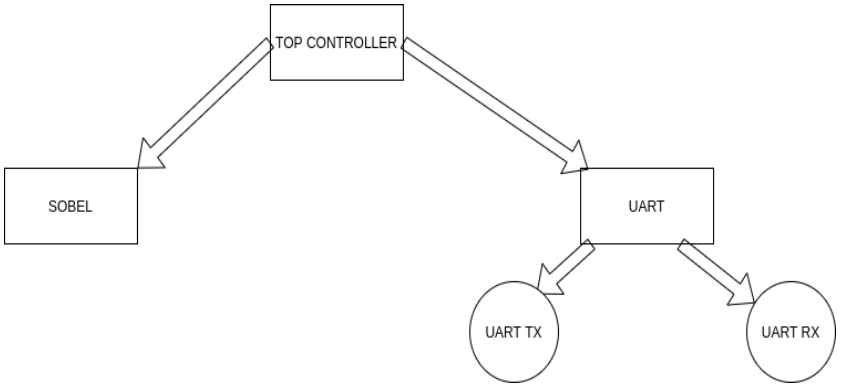


Figure 5: Flow Chart of our Design

## 3.2 Steps to make a synthesizable code

We first made our RTL level code in verilog. The next step was to make it synthesizable. We had to remove dependencies such as hierarchical referencing, loops and commands such as $realtobits etc.

## 3.3 Summary of Resource Utilization

In figure 6 we can see the summary of resource utilization post synthesis.

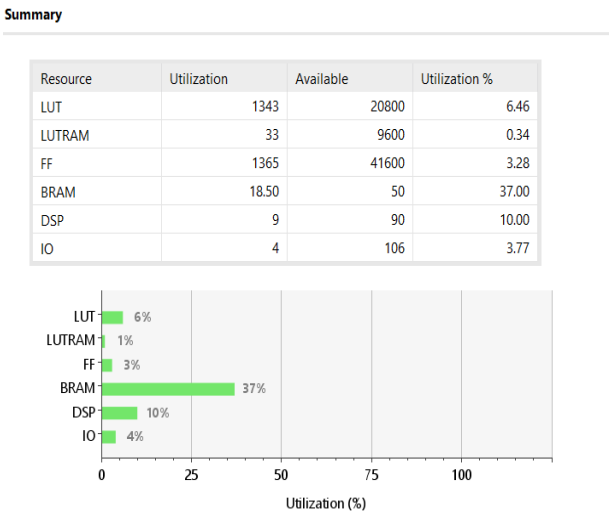


Figure 6: Resource Utilization Post Synthesis

In figure 7 we can see the device schematic report post synthesis.

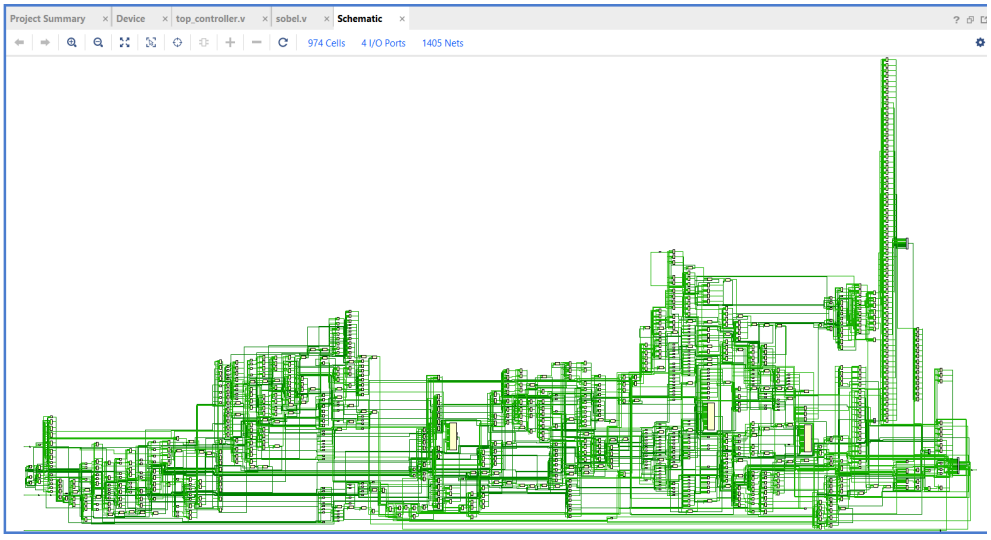


Figure 7: RTL Schematic

In figure 8 we can see the power report post synthesis.

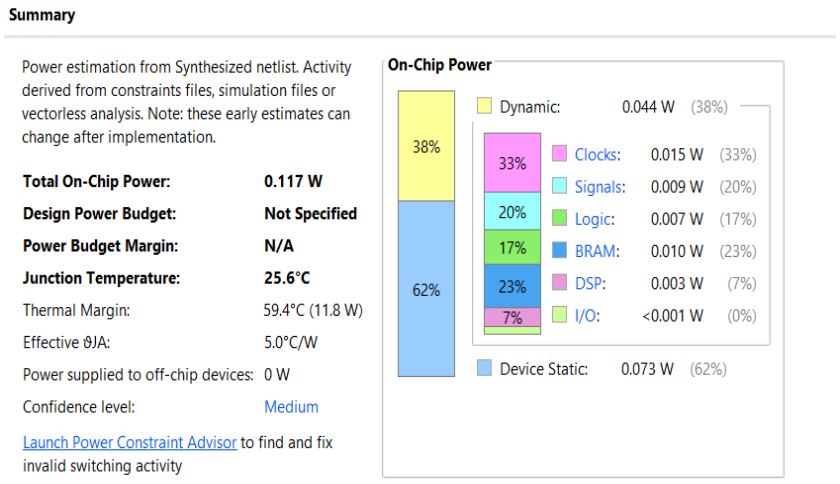


Figure 8: Power Report Post Synthesis

In figure 9 we can see the summary of resource utilization post implementation.

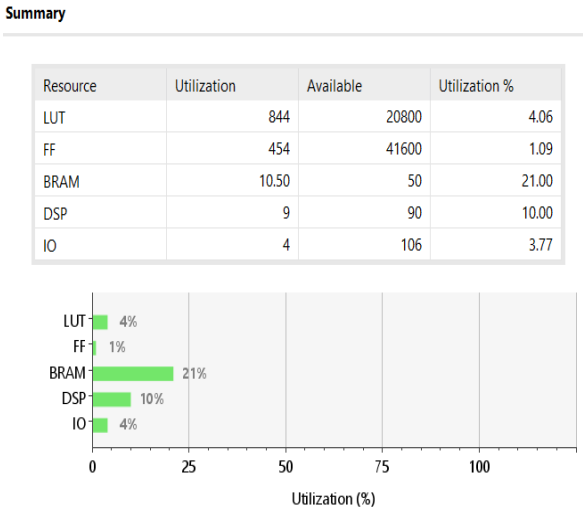


Figure 9: Resource Utilization Post Implementation

In figure 10 we can see the power report post implementation.

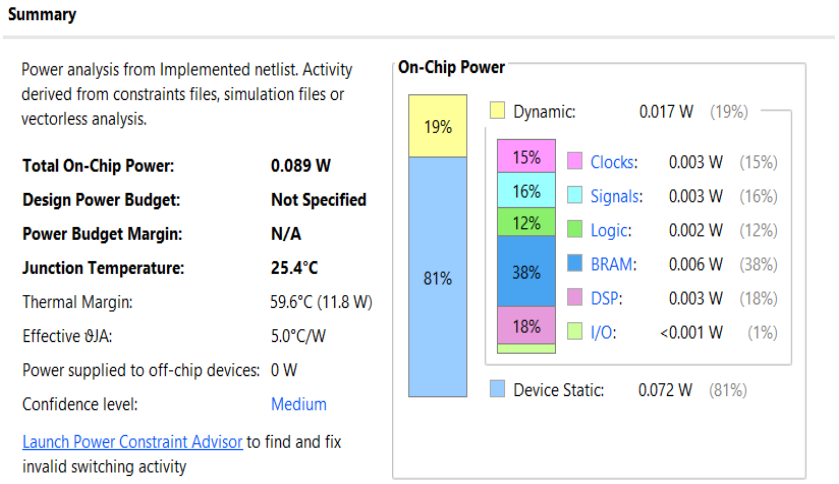


Figure 10: Power Report Post Synthesis

In figure 11 we can see device mapping done on Basys3 Board.

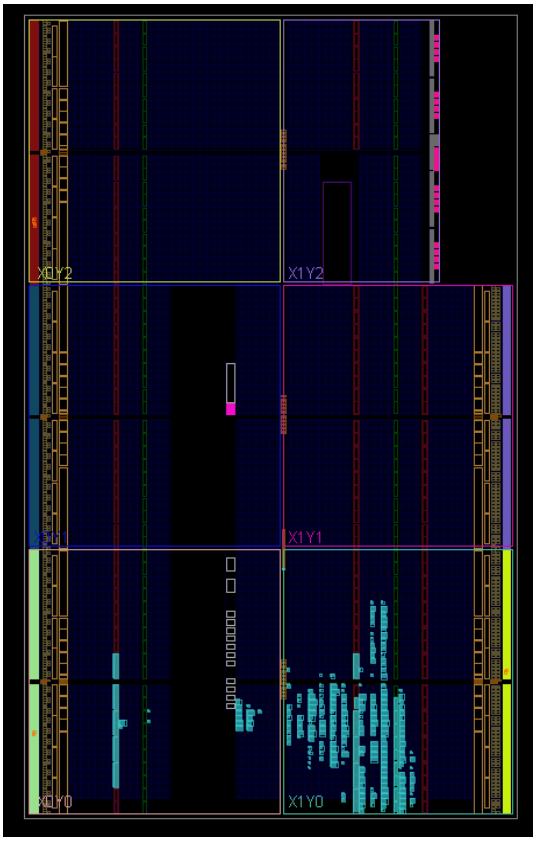


Figure 11: Device Mapping

## 3.4 Test-Bench

We used a test-bench to verify our working of the code. We sent data through a loop to the uart and received data from uart. Main purpose the implementation was to verify working of our code and as mentioned we used MATLAB as our golden reference.

The the result of our simulation is shown in figure 12.

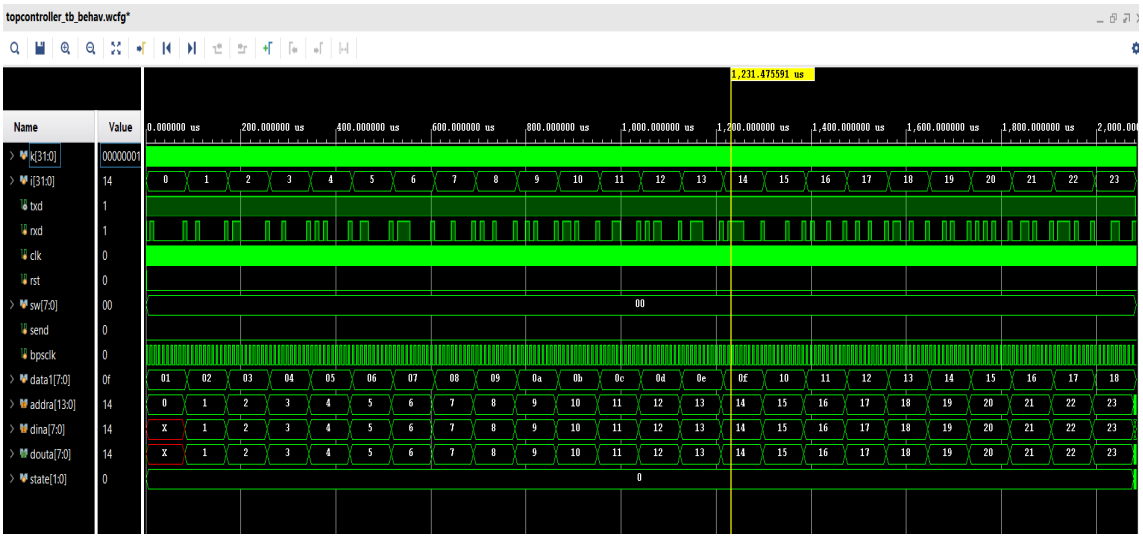


Figure 12: Simulation Waveform

# 4 Results

The final output of our project can be seen in figure 13. In this figure we can see that edge was detected and though there were several noise. We figured out that we didn't do image sharpening and noise removal hence we got some noise.

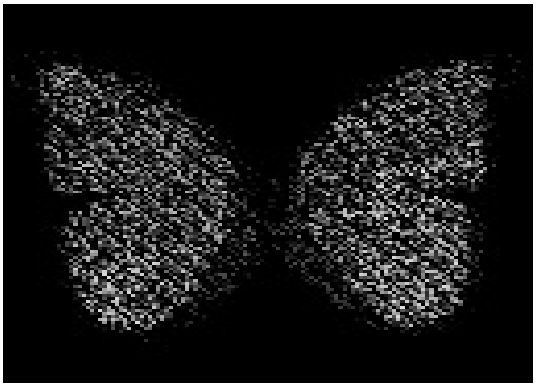

Figure 13: Edge Detected Image after Hardware Implementation

# 5 References

1. https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm

2. https://www.geeksforgeeks.org/matlab-edge-detection-of-an-image-without-using-in-built-function/

3. www.nandland.com

4. https://www.researchgate.net/publication/330280038_Review_on_Image_processingFPGA_implem