Image Processing Implementation using Basys3 FPGA

Bibhore Goswami
Department of Electronic System Engineering
Indian Institute of Science
Bengaluru, India
ddebasmita@iisc.ac.in

Debasmita Deoghuria
Department of Electronic System Engineering
Indian Institute of Science
Bengaluru, India
bibhoreg@iisc.ac.in

***Abstract*—This report presents a detailed analysis of various image processing implementations on the Basys3 FPGA for black and white images, including Gaussian blurring, Sobel edge detection using different approximations, and image sharpening. The second part focuses on displaying these processed images via VGA.**

***Index Terms*—image processing, sobel, gaussian**

I. Introduction

The image processor module takes a clock signal, reset, and a 256×256 black and white image provided as a `.mem` file. Each pixel is passed through a two-stage pipelined Sobel edge detection module, producing an output called **effective_G**.

The VGA display module also takes a clock and reset as inputs and generates outputs including `video_on`, `hsync`, `vsync`, `p_tick`, `x`, and `y`. It divides the input clock t generate a 25 MHz `p_tick` signal. The image processo operates on this `p_tick` clock.

A detailed explanation of each module is provided in th following sections.

II. Sobel image Processor Module

A. Address Calculation

In the input `.mem` file, pixels are stored sequentially fror the 0th to the 65535th memory location. Each location hold a 4-bit pixel value, allowing for 16 grayscale intensity level ranging from 0 to 15.

Although an image is conceptually stored as a 256×25 matrix with each element representing a 4-bit pixel, Verilo does not support image formats like `.jpeg` or `.png`. There fore, image data must be accessed using memory addresses. For processing, pixel positions are typically handled using their `(row, column)` indices, which must be converted to a linear memory address to read or modify pixel values and their neighbors. This conversion is essential for implementing operations such as convolution or edge detection in Verilog. Suppose we have a pixel(E) with $row = row$ and $column = col$.So the following formulae gives the address of a pixels and its neighbors in .mem file stored as $reg[3:0]$ $bram[0:65535]$ in verilog:

- $A = bram[(row-1)*256+(col-1)]$
- $B = bram[(row-1)*256+(col)]$
- $C = bram[(row-1)*256+(col+1)]$
- $D = bram[(row)*256+(col-1)]$
- $E = bram[(row)*256+(col)]$
- $F = bram[(row)*256+(col+1)]$
- $G = bram[(row+1)*256+(col-1)]$
- $H = bram[(row+1)*256+(col)]$
- $I = bram[(row+1)*256+(col+1)]$

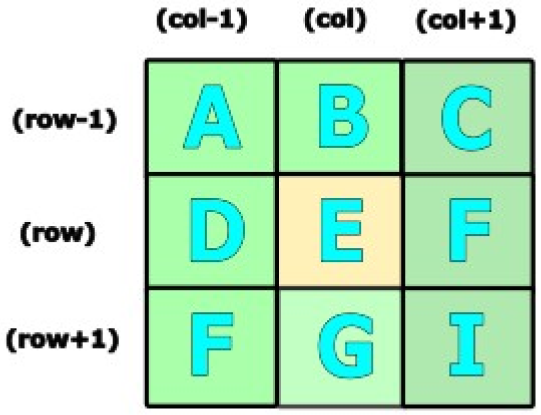


Fig. 1. Diagram of a pixel

B. Sobel Edge Detector

- The VGA operator uses two 3×3 kernels to compute the gradient of image intensity in horizontal and vertical

Identify applicable funding agency here. If none, delete this.

directions.
The horizontal Sobel kernel G_x and vertical Sobel kernel G_y are defined as:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Let the 3×3 image region around a pixel be represented as:

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix}$$

The convolution operations to compute the gradients are:

$$G_x = -A + C - 2D + 2F - G + I \quad (1)$$

$$G_y = A + 2B + C - G - 2H - I \quad (2)$$

- Now the effective G can be calculated by

$$G_{effective} = \sqrt{G_x^2 + G_y^2} \quad (3)$$

But calculating square root requires lot of complex hardware making the model inefficient. Instead of using this formula we can approximate it by either

$$G = |G_x| + |G_y| \quad (4)$$

or

$$G = 0.9 * |max(G_x, G_y)| + 0.4 * |min(G_x, G_y)| \quad (5)$$

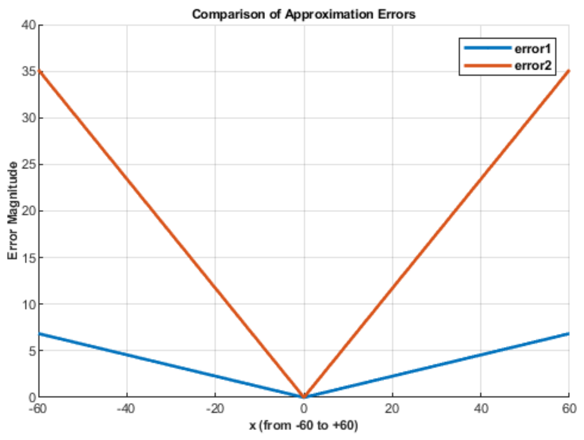


Fig. 2. Comparision between sobel approximators

- In the figure 2, error1 represents (3) - (5) whereas the error2 represents (3) - (4). So equation (5) is a better approximation of equation (3).
- This entire operation is executed using two seperate modules. First one calculates the G_x, G_y and the second one calculates the equation (5). The block diagram of the sobel edge detection algorithm is given in figure 3

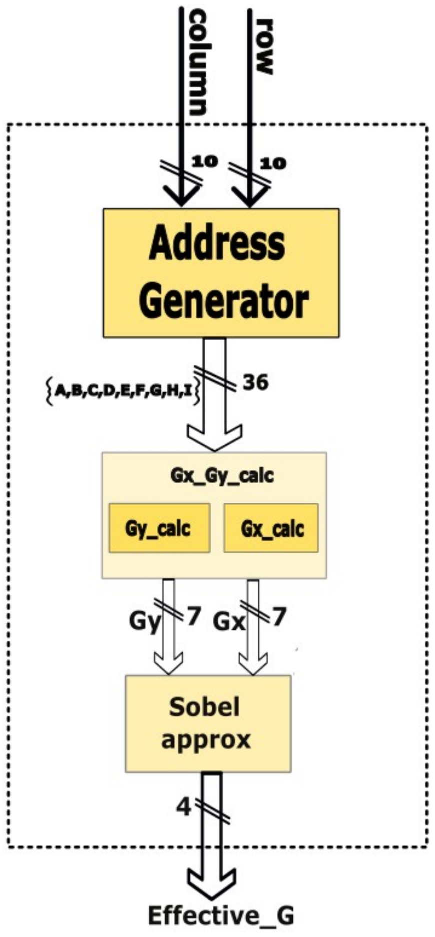


Fig. 3. Block Diagram of Sobel Image Processor

C. Gaussian smoothening

For gaussian smoothing a seperate kernel $G_{gaussian}$ is used.

$$G_{gaussian} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix},$$

$$G_{output} = \frac{(A + 2B + C + 2D + 4E + 2F + G + 2H + I)}{16}$$

The elements of this matrix replecates the guassian distribution which gives a relatively smooth image.

D. Sharpening

Sharpening operator compares each of the pixel with its neighboring pixels. Depening on the intensity differences it intenisify that particular pixel. Hence we get a sharp image. Although our verilog module calculates the output pixels properly we don't get a perfect as the image pixel depth is only 4 bits. The kernel used for sharpening is $G_{sharpening}$ where,

$$G_{inversion} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix},$$

$$G_{output} = 9E - (A + B + C + D + F + H + G + I)$$

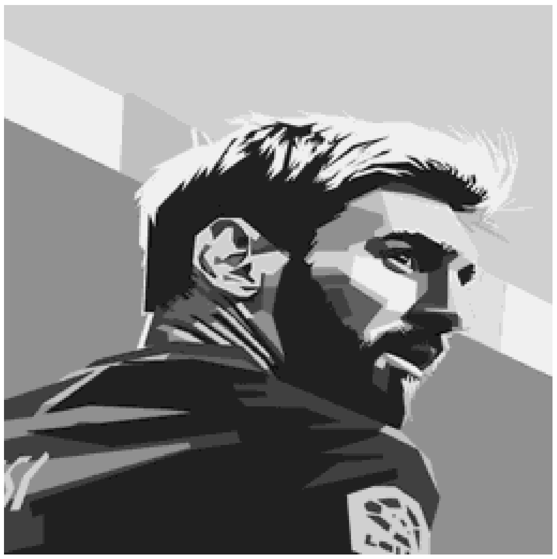


Fig. 4. 8 bit 256x256 grayscale Input image

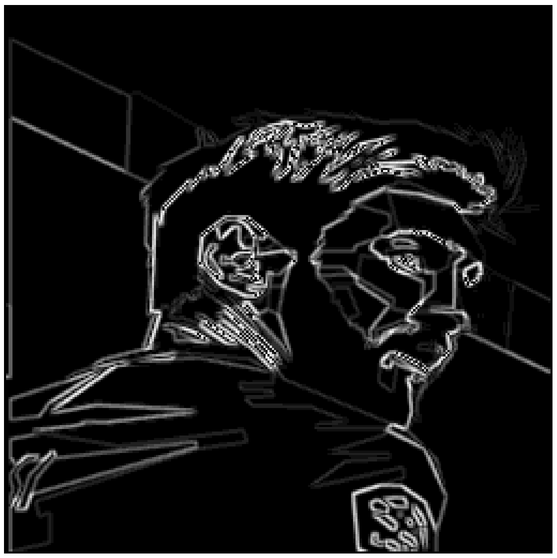


Fig. 5. Edge detected output using $G_{eff} = |G_x| + |G_y|$ on python

III. VGA Controller and VGA display

VGA stands for Video Graphics Array, and it's a display standard developed by IBM in 1987. In the context of FPGA or digital systems, VGA refers to the interface protocol used to drive monitors using RGB signals and synchronization pulses. VGA, or Video Graphics Array,

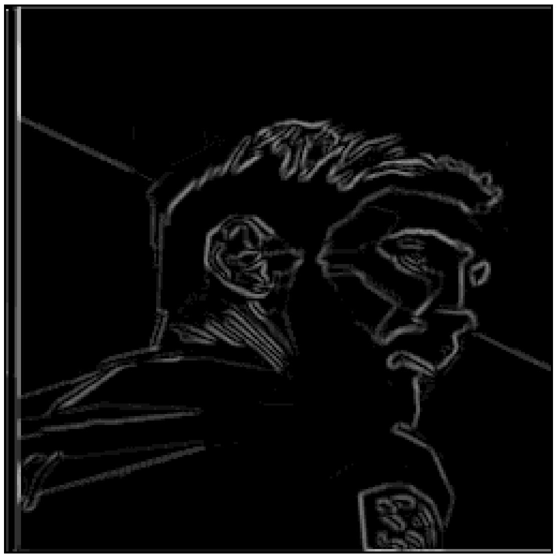


Fig. 6. Edge detection using $G_{eff} = 0.9 * abs((max(G_x, G_y)) + 0.4 * abs(min(G_y, G_x))$

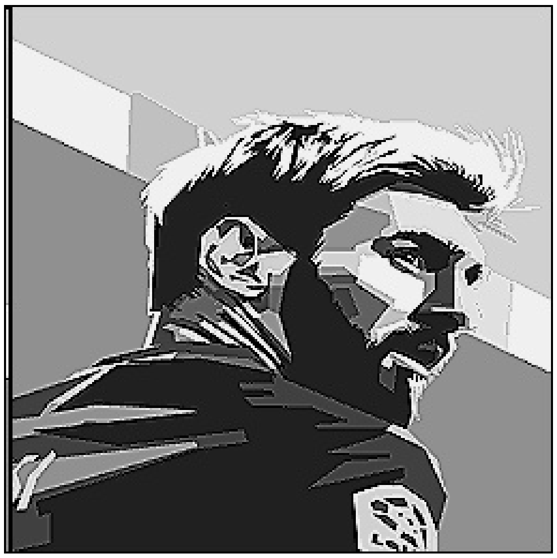


Fig. 7. Sharpened image output in python

is a popular video standard for showing graphics on a computer monitor or television screen. The Basys 3 board uses 14 FPGA signals to create a VGA port with 4-bits per color and the two standard sync signals (HS – Horizontal Sync, and VS – Vertical Sync). The color signals use resistor-divider circuits that work in conjunction with the 75 ohm termination resistance of the VGA display to create 16 signal levels each on the red, green, and blue VGA signals. Basically, a ADC. Using this circuit, 4096 different colors can be displayed, one for each unique 12-bit pattern. A video controller circuit must be created in the FPGA to drive the sync and color signals with the correct timing in order to produce a working display system.

Fig. 8. Smoothened image outputon python

1. Pixel Clock and Refresh Rate

For a display resolution of 640×480 at a 60 Hz refresh rate, the required pixel clock is:

$$f_{\text{pixel}} = 25 \text{ MHz}$$

This frequency drives the VGA controller and determines the timing of pixel generation and synchronization pulses.

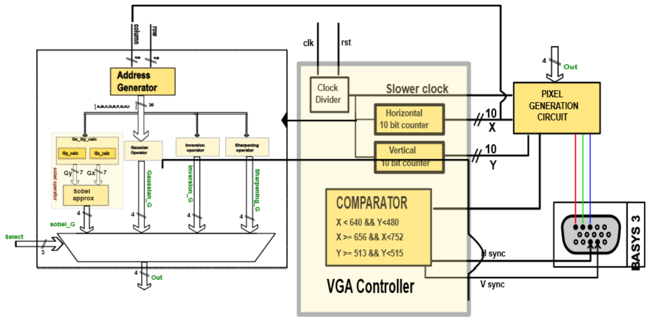


Fig. 9. Overall Architecture

2. Horizontal Scanning and Synchronization

Each line on the screen includes both visible pixels and non-visible timing intervals known as the **blanking period**, which is divided into:
leftmargin=1.5cm

- **Front Porch**
- **Horizontal Sync Pulse**
- **Back Porch**

These intervals were originally introduced for CRT monitors to allow time for the electron beam to return to the start of the next line. Modern displays retain them for backward compatibility.

2.1 Horizontal Timing Parameters

- Visible area: 640 pixels
- Front porch: 16 pixels
- Sync pulse width: 96 pixels
- Back porch: 48 pixels

Total horizontal period:

$$H_{\text{total}} = 640 + 16 + 96 + 48 = 800 \text{ frames}$$

Horizontal sync pulse duration:

$$t_{\text{hsync}} = \frac{96}{25 \times 10^6} = 3.84\ \mu s$$

Back porch duration:

$$t_{\text{back}} = \frac{48}{25 \times 10^6} = 1.92\ \mu s$$

3. Vertical Scanning and Synchronization

The vertical dimension includes visible rows and vertical blanking periods. These are similarly divided into front porch, sync pulse, and back porch.

3.1 Vertical Timing Parameters

- Visible area: 480 lines
- Front porch: 10 lines
- Sync pulse width: 2 lines
- Back porch: 33 lines

Total vertical period:

$$V_{\text{total}} = 480 + 10 + 2 + 33 = 525 \text{ lines}$$

4. Counters and Signal Generation

To generate VGA timing, two counters are used:

4.1 Horizontal Counter (mod-800)

Counts from 0 to 799, resetting each line.

$$\text{pixel\_x} = H_{\text{count}} \mod 800$$

4.2 Vertical Counter (mod-525)

Increments after every horizontal scan, counts from 0 to 524.

$$\text{pixel\_y} = V_{\text{count}} \mod 525$$

5. Control Signals

- **hsync**: Low during horizontal sync pulse (e.g., $656 \leq H_{\text{count}} < 752$)
- **vsync**: Low during vertical sync pulse (e.g., $490 \leq V_{\text{count}} < 492$)
- **video_on**: High only when within displayable region:

$$0 \leq \text{pixel\_x} < 640, \quad 0 \leq \text{pixel\_y} < 480$$

- **pixel_tick**: Pulses at 25 MHz rate to indicate pixel timing

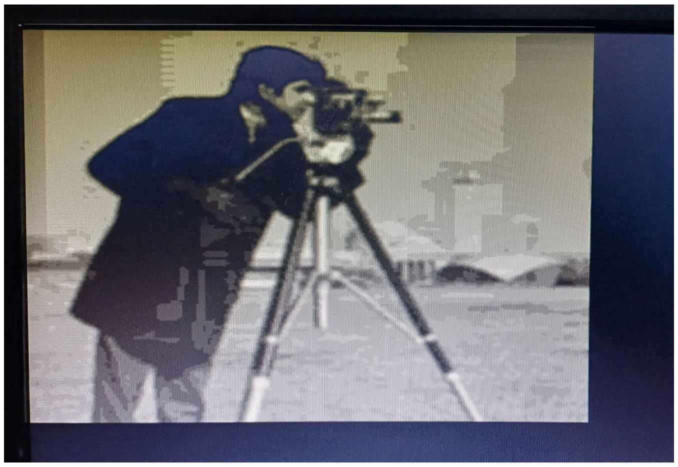


Fig. 10. Gaussian Blurred Image of a 256x256 grayscale image

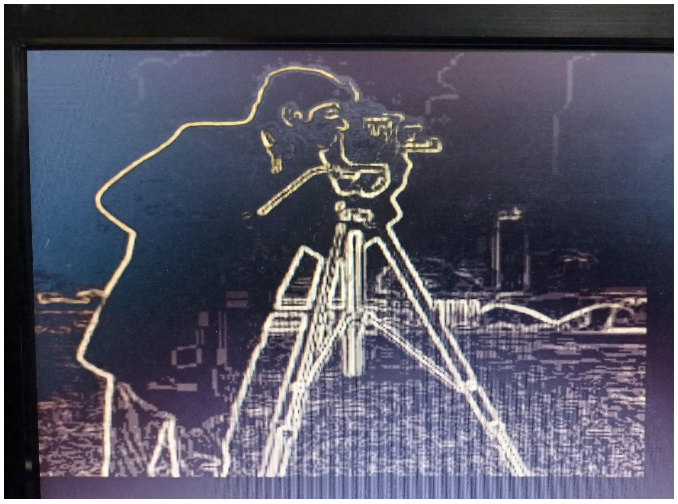


Fig. 11. Edge Detected Image of a 256x256 grayscale image

6. Display Behavior During Blanking

During blanking intervals (front porch, sync, and back porch), the RGB signals should be set to 0 (black) to avoid unintended visuals. This ensures compatibility and clean display output.

IV. CHALLENGES FACED

- All multiplications are approximated using left and right shifts to reduce hardware complexity and improve efficiency.
- Initially, we faced a resource overutilization issue. To address this, we implemented pipelining: a 3-stage pipeline for smoothing and inversion, and a 4-stage
- In the final implementation phase, the synthesis tool mistakenly treated the reset signal (`rst`) as a clock, resulting in a "Place and Route" error.

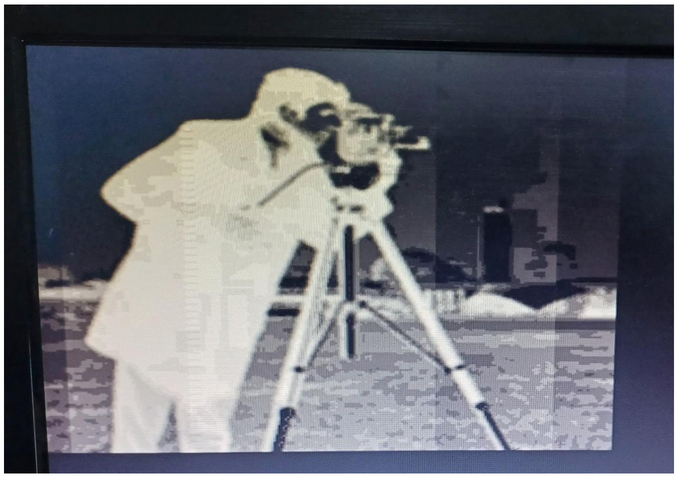


Fig. 12. Inverted Image of a 256x256 grayscale image

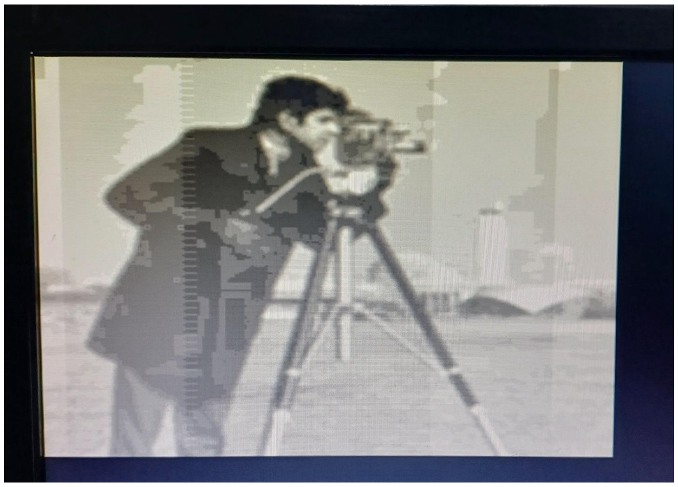


Fig. 13. Sharpened Image of a 256x256 grayscale image

V. CONCLUSION

This project successfully demonstrated the capability of FPGA-based real-time image processing using the Basys3 development board. We implemented critical image processing algorithms, including Gaussian blurring, Sobel edge detection (using optimized approximations), and image sharpening. Each algorithm was carefully adapted for efficient FPGA implementation, considering hardware limitations such as pixel depth and computational complexity.

The use of FPGA facilitated parallel processing, enabling high-speed, real-time performance crucial for embedded vision applications. Moreover, the integration of VGA for displaying processed images highlighted the practical applicability of the system in real-world scenarios.

Future extensions of this work could involve enhancing pixel depth for improved image quality, implementing more complex filtering and edge detection algorithms, and exploring higher resolution images. Additionally, leveraging advanced FPGA platforms with greater resources could enable the implementation of sophisticated neural network-based image processing techniques, expanding the scope of FPGA applications in visual data processing.

VI. Future Work and Improvements

Here are some potential areas for further exploration:

- Implementing additional transformations like shearing and reflection.
- Experimenting with different image formats and color depths.
- Creating animations by applying transformations over time.
- Exploring the integration of user input for interactive graphics.
- Investigating the optimization of the pipeline for faster rendering.

VII. Results Post Implementation

1. Ressource Utilization

The input image used in this project is a grayscale image with a resolution of 256×256 pixels. Each pixel is represented using 4 bits, allowing 16 levels of intensity. Initially, this image was stored in a custom memory structure implemented as a two-dimensional array of dimensions 4×65536, where each entry holds a 4-bit pixel value. This approach ensured efficient access and processing during simulation and early stages of synthesis.

However, after the synthesis and implementation phases, the FPGA toolchain automatically inferred and mapped this custom memory to the on-chip Block RAMs (BRAMs) available on the Basys-3 FPGA board. This mapping helped optimize memory usage and improved access speed during real-time operation.

In terms of logic resource utilization, the implementation reported that approximately 39

2. Timing Analysis

The worst negative setup slack is 4.17ns while the worst negative hold slack is 0.151ns. The WNS could have been decreased but it would not have been fruitful. The maximum frequency is governed by the VGA display which can run ata maximum speed of 25MHHz. Hence we are at a bottleneck. The current operating frequency of the system is at **100MHz**.

3. Power Summary

The total on-chip power dissipated by the design is approximately 0.28W. A detailed analysis of the power components reveals that 73% of this power is dynamic, while the remaining 27% accounts for static (leakage) power. The high proportion of dynamic power can be attributed to the extensive use of sequential logic elements throughout the design, particularly flip-flops and clocked components within the pipelined architecture. These elements switch frequently during operation, contributing significantly to dynamic power consumption. The relatively lower static power indicates that leakage currents are well-managed, which is typical in designs optimized for real-time digital processing on FPGA platforms.

References

[1] A FPGA based implementation of Sobel edge detection Nazma Nausheena, Ayan Seal∗,a, Pritee Khannaa, Santanu Halder

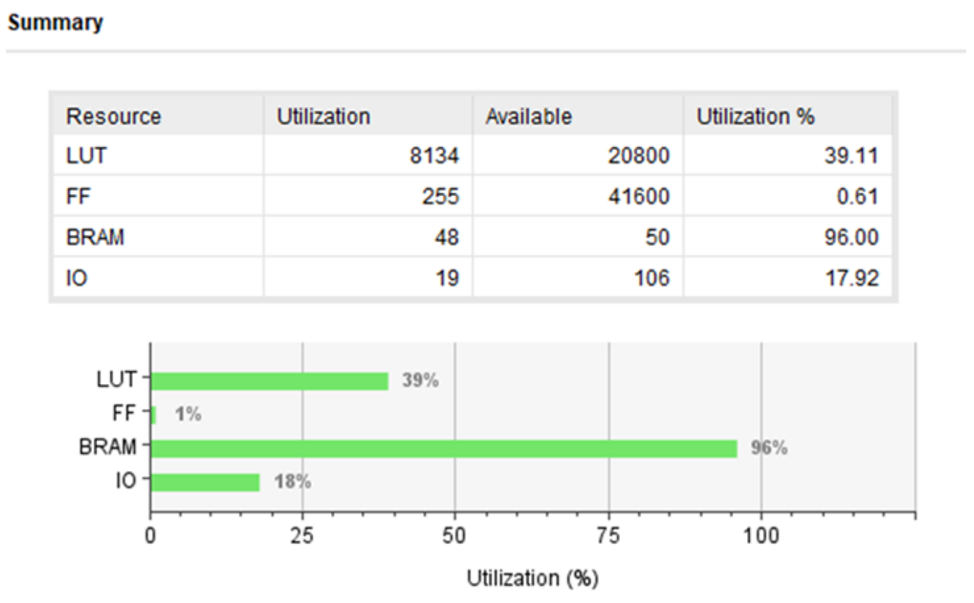
[2] Efficient Edge Detection on Low-Cost FPGAs Jamie Schiel Department of Mechatronic Engineering University of Canterbury Christchurch, New Zealand jms314@uclive.ac.nz

[3] Al Bovik's Lectures on Digital Image Processing Professor Alan C. Bovik, Director Laboratory for Image and Video Engineering

[4] Digital Image Processing by Rafael C. Gonzalez • Richard E. Woods

[5] Narayanan Sundararajan , "FPGA Implementation of Edge Detection for Sobel Operator in Eight Directions ," J.

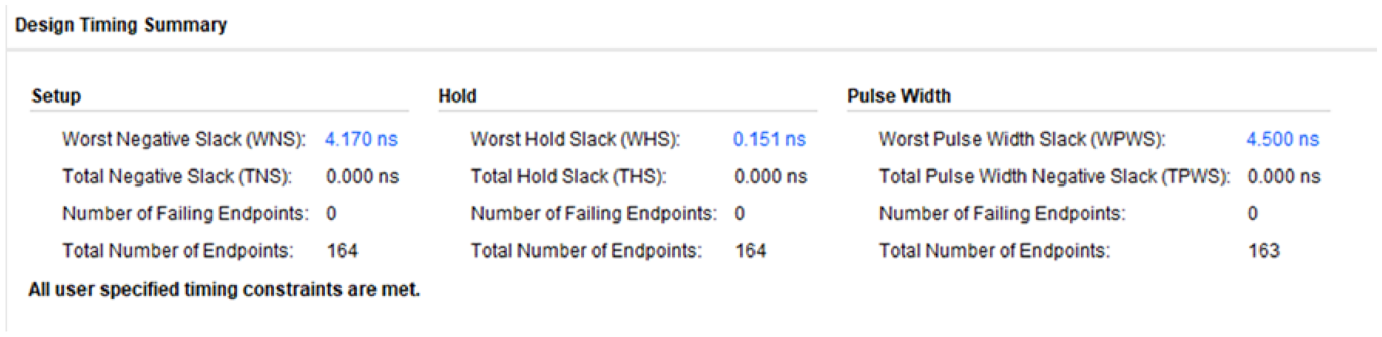
[6] Implementation of Edge Detection Using Fpga Model Based Approach Prof.(Dr.) P.K Dash Department of Electrical Engineering Institute of Technical Education Research, Khandagiri, BBSR-751030, Odisha, India.



Summary

Resource	Utilization	Available	Utilization %
LUT	8134	20800	39.11
FF	255	41600	0.61
BRAM	48	50	96.00
IO	19	106	17.92

Fig. 14. **RESOURCE UTILISATION REPORT**

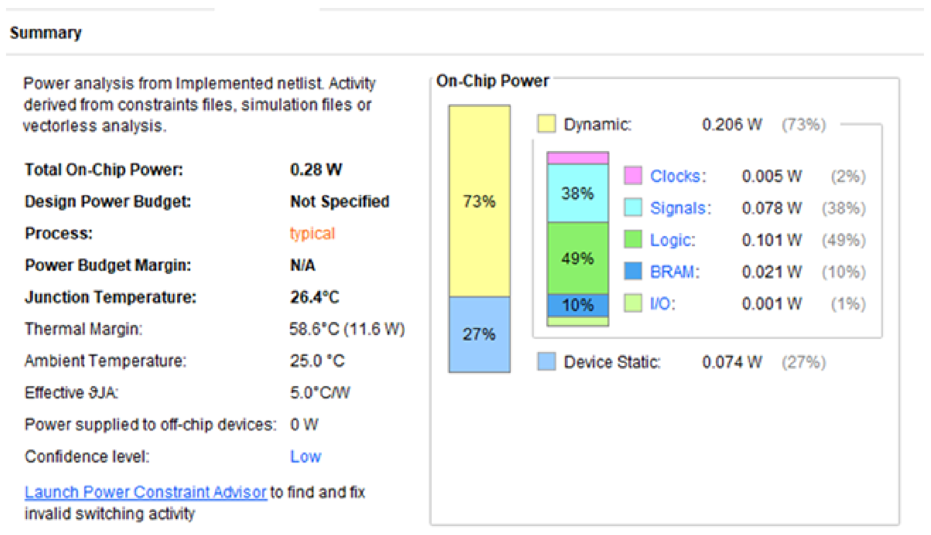


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.170 ns	Worst Hold Slack (WHS):	0.151 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	164	Total Number of Endpoints:	164	Total Number of Endpoints:	163

All user specified timing constraints are met.

Fig. 15. **TIMING SUMMARY**



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.28 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**26.4°C**
Thermal Margin:	58.6°C (11.6 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	0.206 W	(73%)	
Clocks:	0.005 W	(2%)	
Signals:	0.078 W	(38%)	
Logic:	0.101 W	(49%)	
BRAM:	0.021 W	(10%)	
I/O:	0.001 W	(1%)	
Device Static:	0.074 W	(27%)	

Fig. 16. **POWER SUMMARY**