**ABSTRACT**

Edge detection is a fundamental process in image processing, crucial for applications such as object recognition, computer vision, and medical imaging. In this paper, we propose a hardware implementation of the Sobel edge detection algorithm on an FPGA platform. The design utilizes parallelism and pipelining to accelerate gradient computation in both horizontal and vertical directions using Sobel operators, followed by gradient magnitude estimation. The architecture is developed in Verilog and verified using ModelSim simulation. MATLAB is used to preprocess input images and visualize the output. By leveraging hardware-oriented optimizations, such as 2's complement-based magnitude calculation and simplified absolute value computations, the design achieves a balance between edge accuracy and resource utilization.

**TABLE OF CONTENTS**

|  |
| --- |
| Introduction |
| Block Diagrams and implementation |
| Code and Explanation |
| Simulation in Quartus Prime and Modelsim |
| Results |
| Conclusion |
| References |
| Biodata |

**INTRODUCTION**

Edge detection serves as a critical preprocessing step in many image analysis and computer vision applications, including facial recognition, lane detection, and medical diagnostics. It highlights significant transitions in pixel intensity, corresponding to object boundaries and structural features. Among various operators, the Sobel operator is widely favored for its simplicity and effectiveness in detecting edges in both vertical and horizontal directions.

Traditional edge detection algorithms, though effective in software, often face limitations in speed and power consumption when deployed in real-time embedded systems. To overcome these constraints, FPGA-based implementations offer an attractive alternative due to their inherent parallelism, configurability, and low-latency processing capabilities.

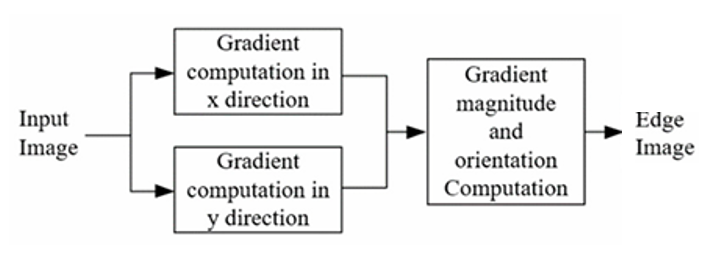
In this paper, we present a complete design and implementation of the Sobel-based edge detection system using Verilog HDL on FPGA. The system takes grayscale images as input, performs 3×3 convolution using Sobel kernels to compute gradients fx​(x,y) and fy​(x,y), and derives edge strength through a hardware-friendly magnitude approximation ∣fx​∣+∣fy​∣. Dedicated hardware architectures are designed for gradient computation and magnitude estimation, optimizing speed and logic utilization. Input and output image handling is facilitated by MATLAB, which converts image pixels to hexadecimal format and reconstructs the processed output image.

The rest of the paper details the architecture of the gradient computation blocks, magnitude computation, Verilog module integration, and simulation results. The system is demonstrated to be robust, scalable, and suitable for deployment in real-time edge detection applications on low-cost FPGA platforms.

**BLOCK DIAGRAMS AND EQUATIONS**

This section presents the architectural and mathematical foundations of the Sobel-based edge detection system implemented on FPGA. The system follows a modular pipeline that begins with gradient computation in the X and Y directions, followed by magnitude estimation to generate the final edge-detected image.

The below given high-level block diagram illustrates the overall flow of the edge detection system. The input image is passed through two parallel Sobel convolution blocks that compute the gradients along the X and Y directions, respectively. These gradients are then fed into the magnitude computation block, which estimates the edge intensity. The final edge image is derived from the combined magnitude values.



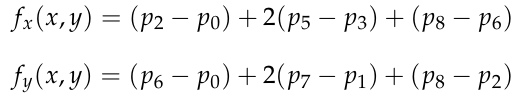
High level Block Diagram for Edge detection in Images

A screenshot of a grid

AI-generated content may be incorrect.Edge detection is performed using a 3×3 window that slides across the image. For each central pixel (p4)​, its eight neighbors are denoted as p0​ through p8​, forming the pixel window used for convolution. The Sobel operators for the X and Y directions are shown below.

Diagram of input 3x3 pixels and Sobel filters in x and y directions

Using the Sobel operators, the gradients at a pixel location (x,y) are computed by:

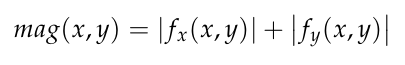


A diagram of a circuit

AI-generated content may be incorrect.These computations are implemented using shift operations and additions in hardware to reduce complexity. The Verilog-based architecture for computing fx​(x,y) is depicted below.

Architecture to compute the gradient in x direction. The same Architecture is used for the y direction as well

To avoid the high hardware cost of computing the square root in the Euclidean norm, a simplified approximation is used for magnitude:



A diagram of a computer

AI-generated content may be incorrect.The absolute values are calculated using 2's complement logic and fed into an adder to compute the total edge strength. The hardware architecture for this block is shown below.

Architecture for Magnitude calculation from gradients

**CODE AND EXPLANATION**

This section elaborates on the implementation of the FPGA-based Sobel edge detection pipeline, highlighting the functionality of each code module and the supporting MATLAB scripts used for preprocessing and result visualization.

The input to the FPGA system is a grayscale image in hexadecimal format. A MATLAB script is used to convert a standard RGB image into a grayscale matrix and then export its pixel values as two-character hexadecimal entries. This format ensures compatibility with the ModelSim input simulation.

The sobelmag.v module performs the core computation of the Sobel operator by calculating the horizontal and vertical gradients and estimating the magnitude of the edge.

Here:

* sum1 and sum2 compute the weighted sums for the X-direction using the Sobel kernel.
* sum3 and sum4 do the same for the Y-direction.
* Absolute values are computed using 2’s complement logic for both xval and yval.
* Finally, a simplified magnitude is calculated and the least significant 8 bits are used as output (magval), which corresponds to the edge intensity at the current pixel.

The imageprocessor.v module acts as a top-level wrapper around sobelmag. It handles:

* Image buffering using a 2D register array min[i][j]
* Sliding 3×3 window extraction for convolution
* Output storage into the result matrix mou

Each pixel and its surrounding neighbors are passed to sobelmag, which computes the edge magnitude. The result is stored in mou[i-1][j-1] and also output through dou.The control signal dlay is used to distinguish between the image loading phase and the processing phase.

After simulation in ModelSim, the edge-detected output (in hex format) is read back into MATLAB and visualized. The script ensures resilience to simulation artifacts such as undefined values (xx or xxxxxxxx), replacing them with black pixels for visualization.

**MATLAB Code 1 (Image to Grayscale Hexadecimal):**

clc

clear all

img=imread("C:\Users\Jayakrishnan Menon\Desktop\matlabforfpga\nando-jpeg-quality-001.jpg"); % Replace with the path of the target image stored on your system

img=rgb2gray(img);

[height, width, ~] = size(img);

% Initialize an array to store hex values

imghexgray = strings(height, width);

% Loop through each pixel and convert RGB to hex

for i = 1:height

for j = 1:width

% Extract values for the current pixel

gray = img(i, j);

% Convert channel to 2-character hex

hexgray = dec2hex(gray, 2);

% Combine into a 6-character hex value

imghexgray(i+1,j+1) = hexgray;

end

end

writematrix(imghexgray,'grayimg.txt','Delimiter',' '); % Grayscale Bytes in txt file

type grayimg.txt

**MATLAB Code 2 (Output Edge Information to Image):**

clc

clear all

filename = "C:\Users\Jayakrishnan Menon\Desktop\msimimgprocess\edgeout.txt"; % Replace with the path of the processed text file stored on your system

% Open file

FID = fopen(filename);

datafromfile = textscan(FID, '%s'); % read as strings

fclose(FID);

% Extract the actual cell array

data = datafromfile{1};

% Replace any invalid entries like 'xx', 'xxxxxxxx', etc. with '00'

for i = 1:length(data)

if isempty(regexp(data{i}, '^[0-9A-Fa-f]{1,2}$', 'once'))

data{i} = '00'; % Replace with black pixel

end

end

% Convert hex strings to decimal uint8

decData = uint8(hex2dec(data));

% Optional: reshape if you know the image dimensions

rows = 600;

cols = 800;

if numel(decData) ~= rows \* cols

warning('Pixel count mismatch: expected %d, got %d', rows\*cols, numel(decData));

end

img = reshape(decData, [cols, rows])'; % transpose for row-major

figure

imshow(img, []);

title('Edge-detected Image');

% Save image if needed

imwrite(img, 'edge\_output\_image.png');

**Verilog Implementation of Sobel Filter (sobelmag.v):**

module sobelmag(input clk, input [7:0]p0,p1,p2,p3,p5,p6,p7,p8, output reg [7:0]magval);

reg [10:0]xval,yval;

reg [9:0]trval;

reg [10:0]sum1,sum2,sum3,sum4,p33,p55,p11,p77;

initial begin xval=11'b00000000000; yval=11'b00000000000; end

always@(posedge clk) begin

magval=8'bxxxxxxxx;

p33=p3<<1;

p55<=p5<<1;

sum1=p0+p33+p6;

sum2<=p2+p55+p8;

p11=p1<<1;

p77<=p7<<1;

sum3=p0+p11+p2;

sum4<=p6+p77+p8;

xval=sum2-sum1;

yval<=sum4-sum3;

//x mag

if(xval[10]==1'b1) begin

xval=~xval;

xval=xval+1'b1;

end

//y mag

if(yval[10]==1'b1) begin

yval=~yval;

yval=yval+1'b1;

end

trval=xval[9:0]+yval[9:0];

magval <= trval[9:2]; //try 9:2 and 7:0 also

//(7:0 is very detailed but noisy ad lines are coarse)

//(8:1 is good enough, in between)

//(9:2 is less detailed, less noisy, but lines are smoother)

end

endmodule

**Verilog Implementation of the processor (imageprocessor.v):**

//Intended to take the image input as 1 piece and output as 1 piece

//It calls the sobel filter in between to do the processing

module imageprocessor(input clk, dlay, input [7:0]din, output reg [7:0]dou);

parameter rows = 600, cols = 800; //change based on image

reg [7:0]p0,p1,p2,p3,p5,p6,p7,p8;

wire[7:0]magval;

reg [7:0] min [0:rows+1][0:cols+1];

reg [7:0] mou [0:rows-1][0:cols-1];

integer i,j;

reg clk1;

initial begin

dou = 8'b00000000;

p0 = 8'b00000000;

p1 = 8'b00000000;

p2 = 8'b00000000;

p3 = 8'b00000000;

p5 = 8'b00000000;

p6 = 8'b00000000;

p7 = 8'b00000000;

p8 = 8'b00000000;

clk1=0;

for(i=0;i<rows+2;i=i+1)begin

for(j=0;j<cols+2;j=j+1)begin

min[i][j]=8'b00000000;

end

end

end

always@(\*)begin //Input here

if(dlay==1'b1)begin

for(i=1;i<rows+1;i=i+1)begin

for(j=1;j<cols+1;j=j+1)begin

@(posedge clk)begin

min[i][j]=din;

end

end

end

end

end

always@(\*)begin

if(dlay==1'b0)begin

for(i=1;i<rows+1;i=i+1)begin

for(j=1;j<cols+1;j=j+1)begin

@(posedge clk)begin

p0 = min[i-1][j-1]; p1 = min[i-1][j+0]; p2 = min[i-1][j+1];

p3 = min[i+0][j-1]; p5 = min[i+0][j+1];

p6 = min[i+1][j-1]; p7 = min[i+1][j+0]; p8 = min[i+1][j+1];

dou = magval;

mou[i-1][j-1] = dou; //[i][j];

end

end

end

end

end

sobelmag sobmg(clk,p0,p1,p2,p3,p5,p6,p7,p8, magval);

endmodule

**Verilog Implementation of the Testbench (process.v):**

module process;

reg clk, clk1, clk2;

integer fptr1, fptr2;

integer i, j;

reg [7:0] din;

wire [7:0] dou;

wire [1:0] stat;

reg dlay;

initial begin

// Open files

fptr1 = $fopen("grayimg.txt", "r");

fptr2 = $fopen("edgeout.txt", "w");

clk = 1;

#100

forever #1 clk = ~clk;

dlay=1'bz;

end

// Input/Output file handling

initial begin

// Read input image and write output

dlay=1;

for (i = 0; i < 600; i = i + 1) begin //rows or height

for (j = 0; j < 800; j = j + 1) begin //cols or width

@(posedge clk)begin

$fscanf(fptr1, "%h", din);

//$fwrite(fptr2, "%h ", dou);

end

end

//$fwrite(fptr2, "\n");

end

dlay=0;

for (i = 0; i < 600; i = i + 1) begin

for (j = 0; j < 800; j = j + 1) begin

@(posedge clk)begin

$fwrite(fptr2, "%h ", dou);

end

end

$fwrite(fptr2, "\n");

end

dlay=1'bz;

// Close files

$fclose(fptr1);

$fclose(fptr2);

$finish;

end

// Instantiate your image processing module

imageprocessor imggg(

.dlay(dlay),

.clk(clk),

.din(din),

.dou(dou)

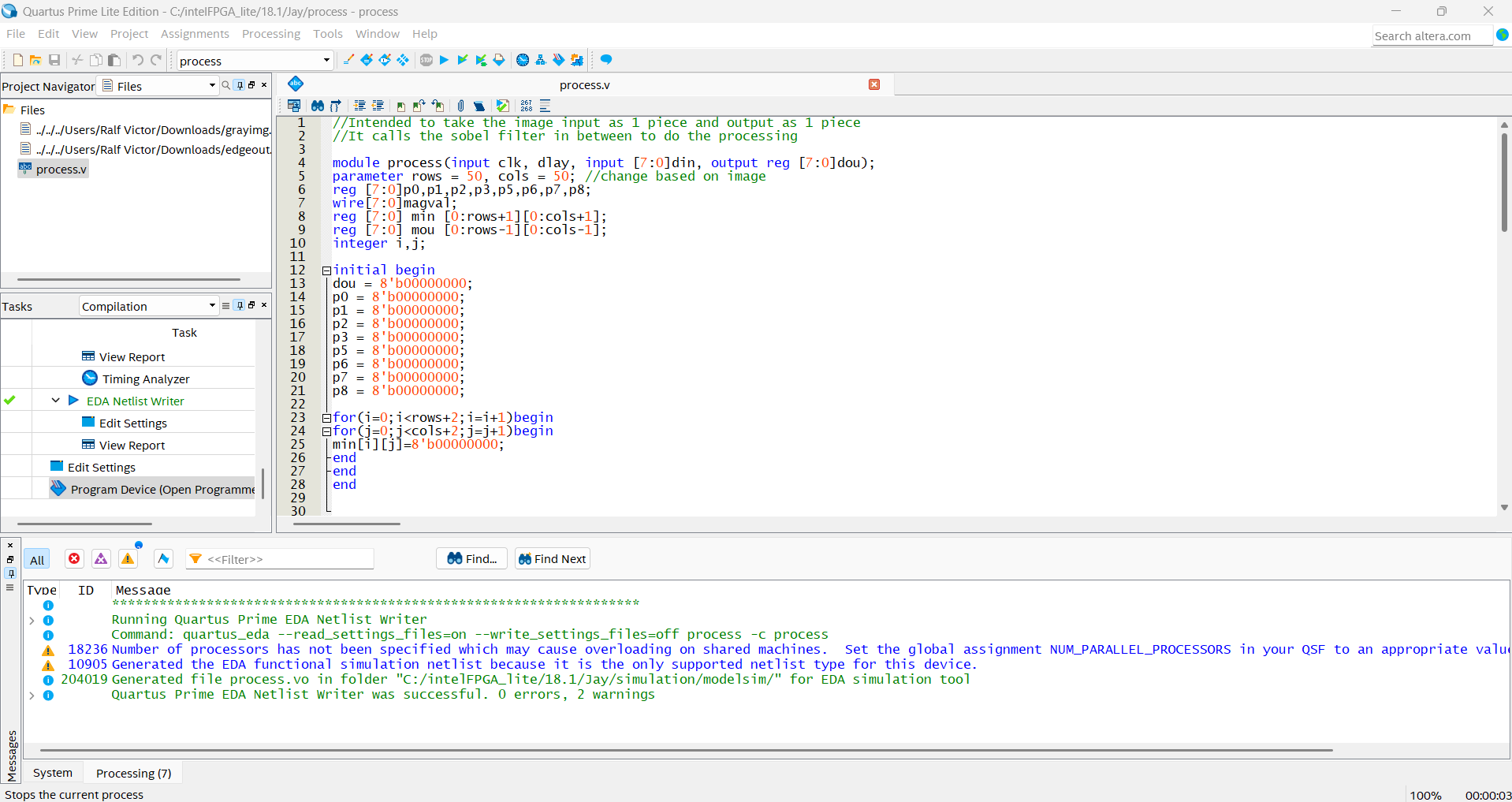
);

endmodule

**SIMULATION IN QUARTUS PRIME AND MODELSIM**

To verify the functionality of the FPGA-based edge detection system, simulation and synthesis were performed using Intel Quartus Prime Lite Edition and ModelSim-Intel FPGA Edition.

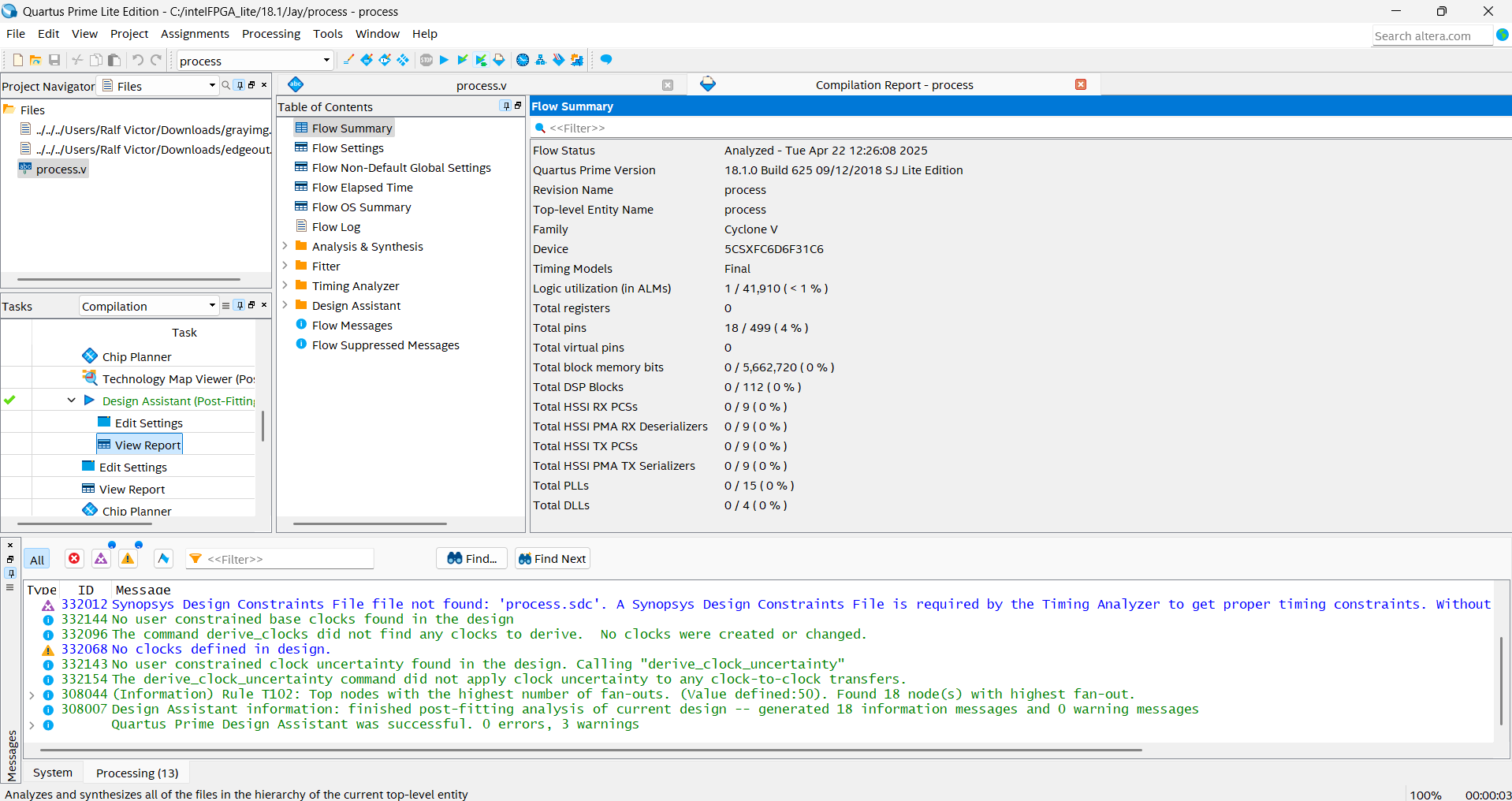
The Verilog module process.v, which integrates the Sobel edge detection logic, was added to a new Quartus project.



The design was successfully compiled, and the EDA Netlist Writer was invoked to generate a simulation-compatible netlist for ModelSim. Key observations from the compilation report include:

* Zero critical errors
* Minimal logic utilization (1 ALM)
* Proper recognition of input and output ports (clk, dlay, din[7:0], dou[7:0])

Warnings such as missing timing constraints (.sdc file) and lack of user-defined clocks were expected and do not affect functional simulation.



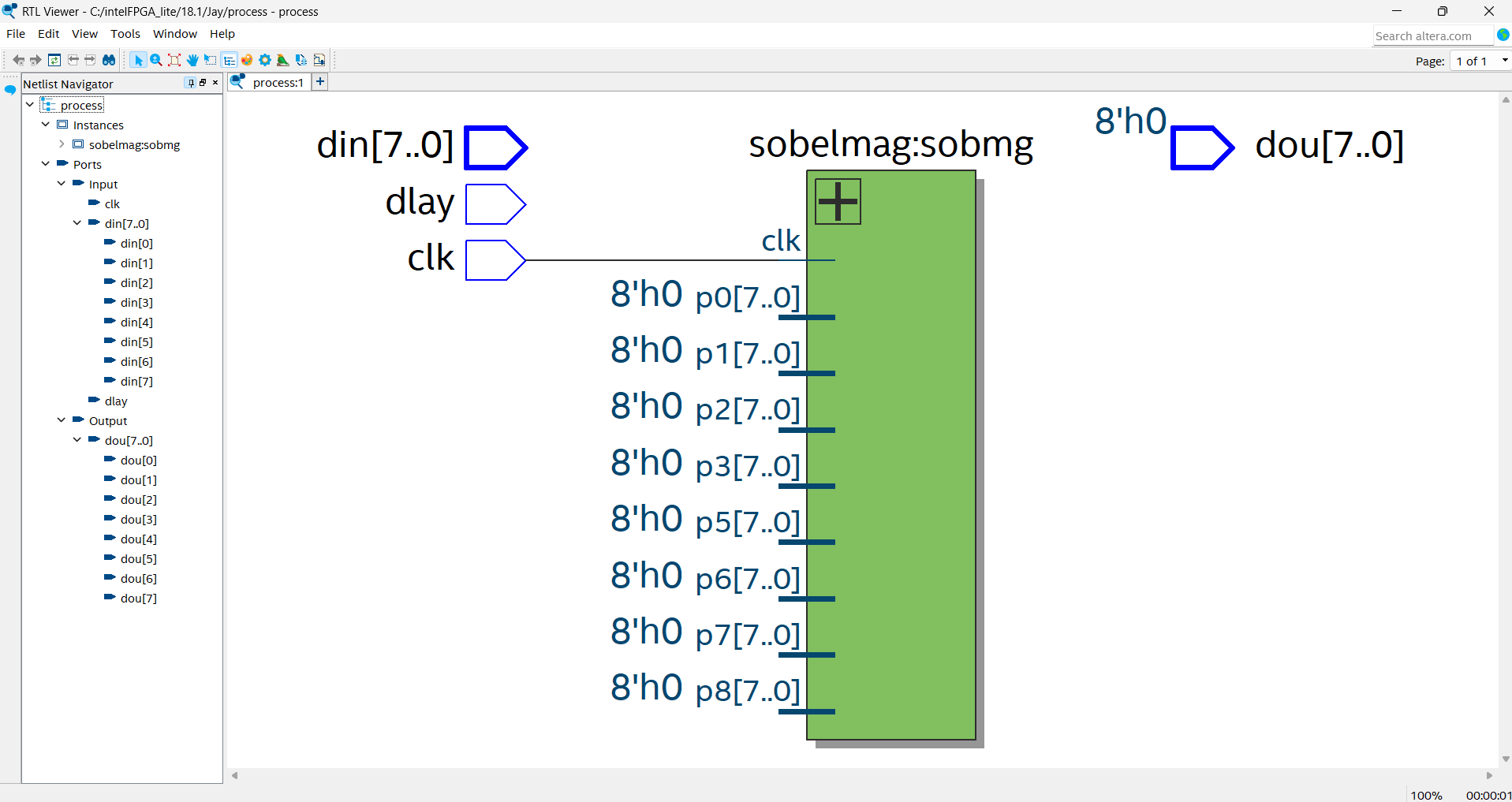
Using RTL Viewer, the synthesized structure of the Sobel processing logic was examined. The hierarchical schematic clearly shows:

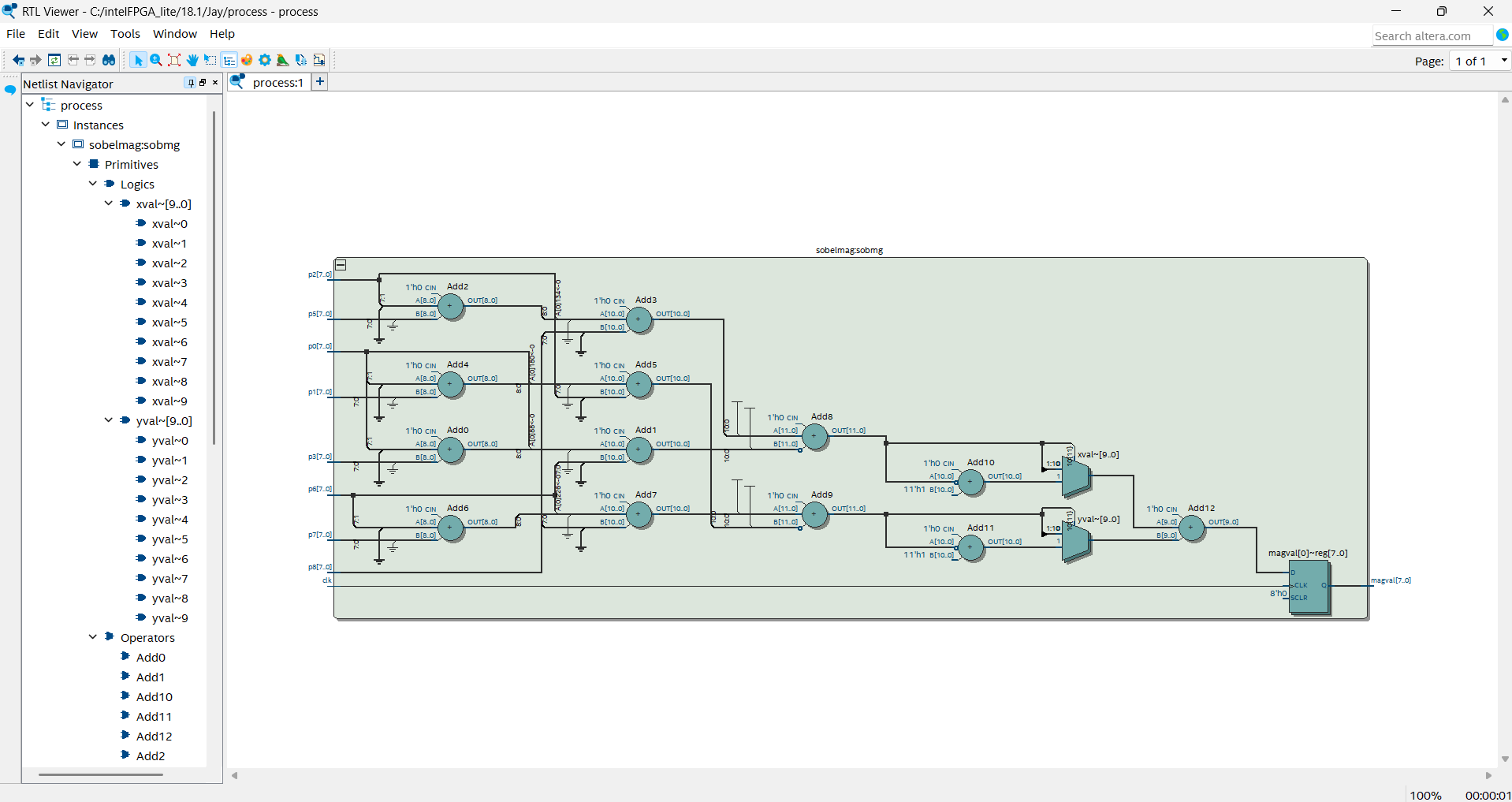
Instantiation of the sobelmag module within process

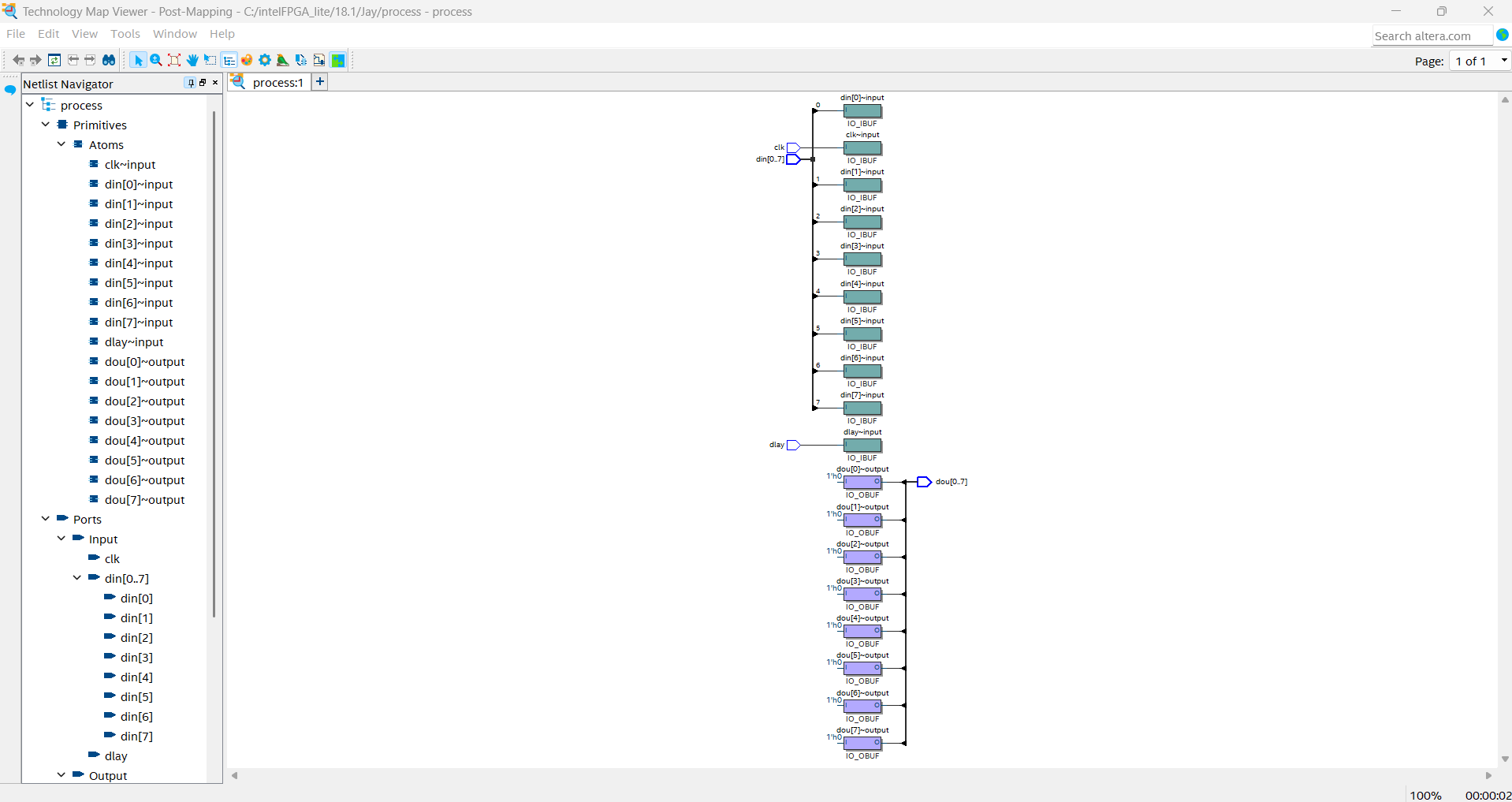
Internal computation of gradient values using adders, shift operations, and absolute difference logic

Clear routing of 3×3 pixel data into the Sobel kernel ports (p0 to p8)

Additionally, the Technology Map Viewer (Post-Mapping) displayed the physical representation of the design in terms of input buffers (IO\_IBUF), output buffers (IO\_OBUF), and logic blocks, confirming correct I/O connectivity and synthesis.







To validate the correctness of the Sobel edge detection core and the complete image processing pipeline, waveform simulation was conducted in ModelSim, and numerical results were manually compared with MATLAB computations.

The two images given below show waveform simulation of the sobelmag module. The signals p0 to p8 represent the 3×3 input pixel window, while internal signals like sum1, sum2, xval, yval, and magval demonstrate step-by-step gradient and magnitude calculation.

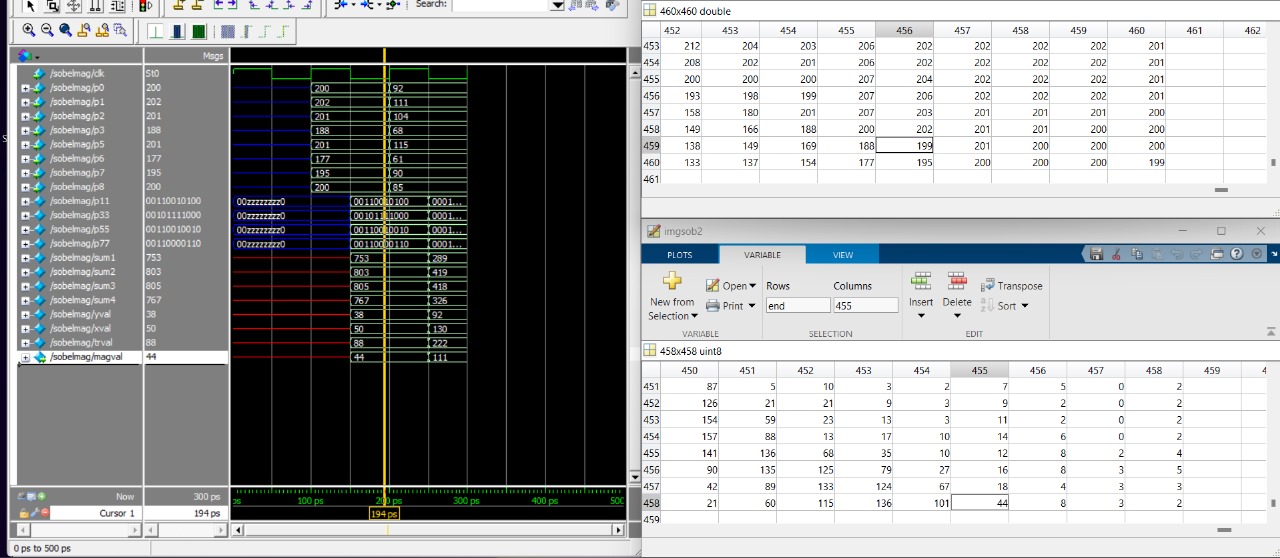
On the right side, MATLAB tables show two selected rows from an input image (460×460) and the corresponding computed edge magnitudes (stored in imgsobb2, a matrix of type uint8).

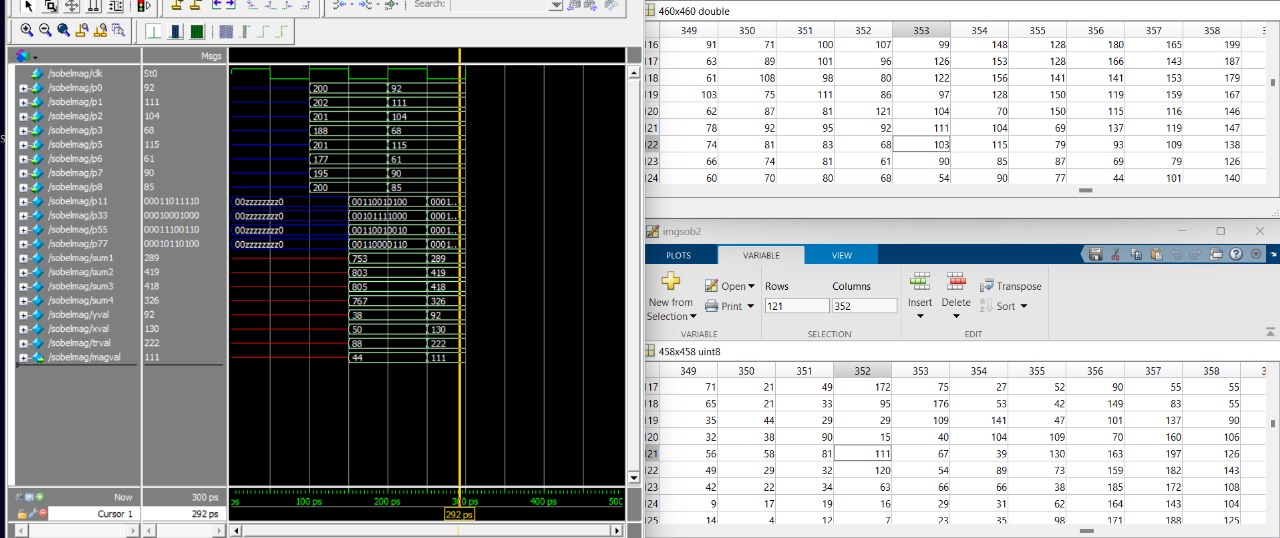
In both test cases, the magval output from the simulation exactly matches the value at the corresponding pixel in the MATLAB-processed Sobel image, confirming that the hardware correctly implements the software-defined Sobel operator.

For example:

* At 194 ps, a different set of pixel values compute to a magval of 44, matching the value at position (end, 455).
* At 292 ps, the pixel window values compute to a magval of 111, matching the MATLAB output at position (121, 352).

These results confirm bit-accurate matching between the hardware implementation and MATLAB's software model.





The below given image illustrates the waveform output for the top-level imageprocessor module, which manages reading the image, applying the Sobel kernel, and writing the edge-detected output.

Key observations:

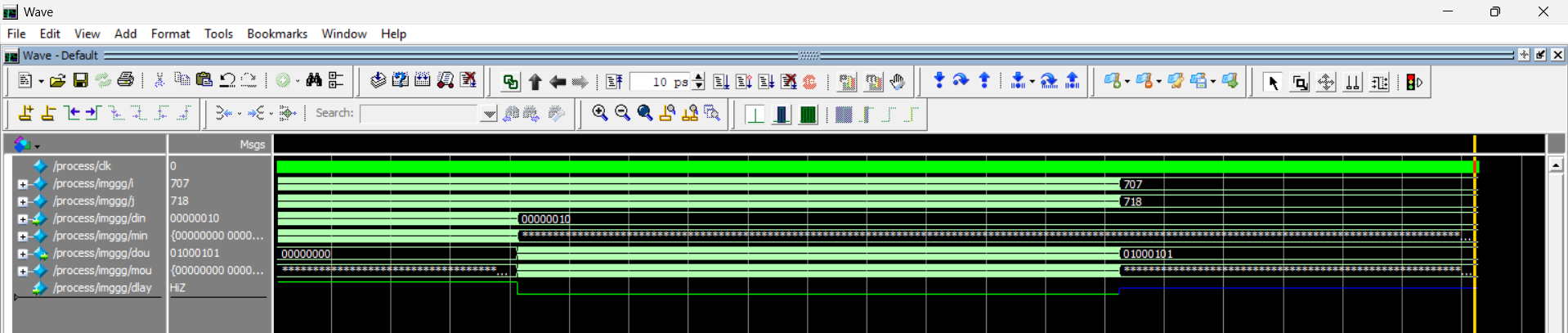
The signal dlay acts as a control flag:

dlay = 1: Reading phase – pixel values are loaded into internal memory.

dlay = 0: Processing phase – edge magnitudes (dou) are computed and written to output.

dlay = Z (high impedance): End of processing – signaling completion of image traversal.

As shown in the simulation timeline, once all pixels are processed, the dlay signal transitions to high impedance (HZ), and no further changes occur on din or dou.



These simulations confirms the correct temporal behavior of the FSM logic driving the image processing control flow. The transition from image loading to processing, and finally to completion, reflects the expected functionality of the hardware pipeline.

**RESULTS**

The proposed FPGA-based Sobel edge detection system was evaluated using a variety of grayscale images, including natural scenes, anatomical X-rays, MRI scans, and road environments. The output was generated through ModelSim simulation and post-processed in MATLAB to reconstruct and visualize the resulting edge maps.

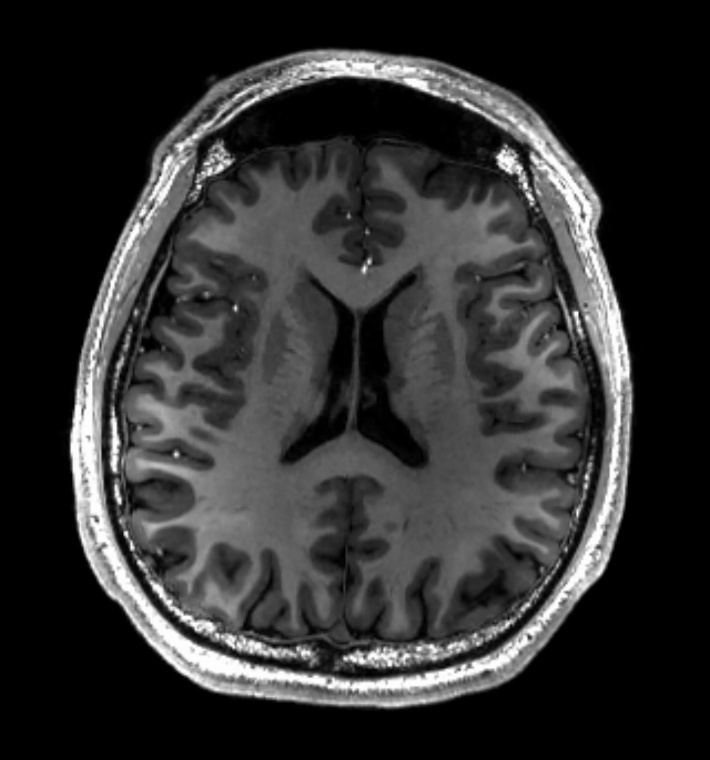
The system effectively highlighted significant image boundaries while preserving structural details. The edge maps clearly capture the contours of natural elements such as mountains and objects in a scene, as well as intricate features in medical imaging, such as bone outlines and tissue transitions. These results demonstrate the versatility of the implementation across diverse image domains.

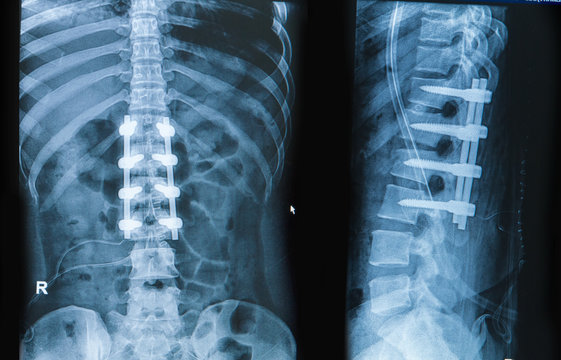
Among the below given images, a scenic image and a pinwheel were processed, revealing prominent outlines in landscapes and the pinwheel structures.

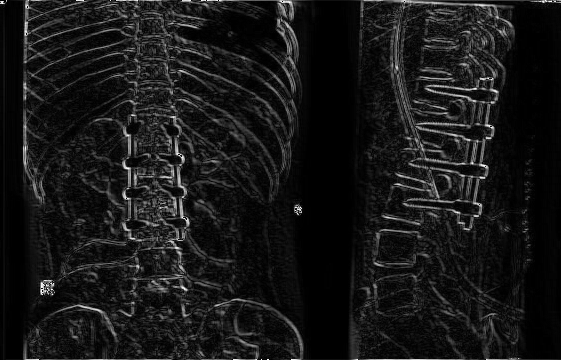


The below given Images show the system’s capability in medical diagnostics, where skeletal features in X-rays and soft tissue contrasts in MRI brain scans are sharply identified. Notably, spinal implants and vertebral edges are well-resolved in spine X-rays, proving the suitability of the design for biomedical edge analysis and clinical imaging. Additionally, the system was also tested on a panoramic dental X-ray image.

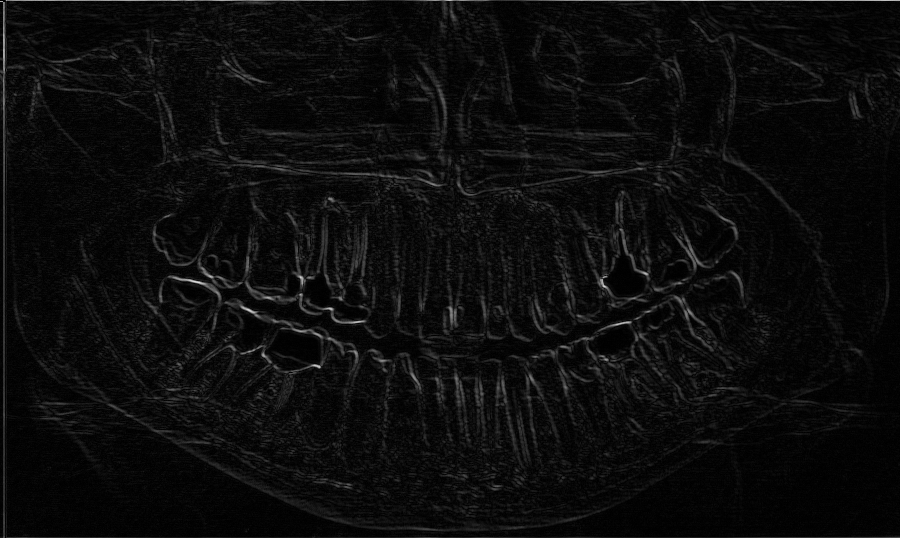












Finally, the system was tested on a highway footage, where it successfully detected fine details in tooth boundaries and road lane markings. This emphasizes the system’s applicability in autonomous navigation systems.





**CONCLUSION**

In this work, an efficient hardware implementation of the Sobel edge detection algorithm was developed and verified using Verilog HDL on an FPGA platform. The design successfully computes image gradients in the horizontal and vertical directions and estimates edge magnitude through a hardware-optimized approach using absolute differences. The sobelmag core and the complete imageprocessor module were verified through functional simulation in ModelSim, and the results were cross-validated with MATLAB outputs for numerical accuracy.

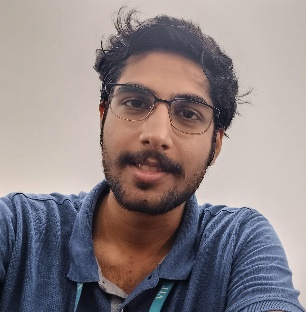
The system was synthesized using Intel Quartus Prime and visualized through RTL Viewer and Technology Map Viewer, confirming correct structural implementation. Simulation waveforms demonstrated accurate pixel-by-pixel Sobel convolution, and testbench-driven verification validated the functional correctness of both core and system-level modules. A control signal was successfully used to differentiate between the image loading, processing, and completion phases, enabling automated edge map generation.

Overall, the project showcases the viability of implementing real-time image processing algorithms like Sobel filtering on resource-constrained FPGA platforms. The design demonstrates low logic utilization and efficient performance, making it highly suitable for embedded vision systems and real-time edge detection tasks in applications such as surveillance, autonomous navigation, and medical imaging.

**REFERENCES**

1. Guo, L., & Wu, S. (2023). FPGA Implementation of a Real-Time Edge Detection System Based on an Improved Canny Algorithm. Applied Sciences, 13(2), 870. <https://doi.org/10.3390/app13020870>
2. Jain, R., Raina, K., & Mishra, A. (2020). FPGA implementation of real-time Sobel edge detection algorithm for high-speed image processing. Journal of Real-Time Image Processing, 17(6), 2091–2103.
3. Yousif, M., Ismail, M., & Ameen, M. (2016). FPGA implementation of a pipelined Sobel edge detector for real-time image processing. International Journal of Electronics and Computer Science Engineering, 5(1), 21–28.
4. Al-Mohair, H., & Hamad, A. (2019). Low-power FPGA-based edge detection using Sobel and Prewitt operators for real-time applications. Journal of Computer Science, 15(10), 1435–1443.
5. Bayoumi, M., & Kumar, S. (2011). FPGA-based edge detection using parallel architecture. In Proc. IEEE International SoC Design Conference (pp. 353–357).
6. Arafat, M. Y., & Salam, R. A. (2013). Real-time Sobel edge detection on an FPGA using Verilog HDL. International Conference on Electrical and Computer Engineering (ICECE), 2013, 471–474.
7. Choudhary, A., & Panigrahi, S. (2019). Design and implementation of Sobel edge detection using FPGA for real-time applications. Procedia Computer Science, 167, 2288–2296.
8. Bhuyan, M. K., & Chatterjee, S. (2010). Hardware-efficient Sobel edge detector using minimal logic and shift operations. International Journal of Computer Applications, 9(6), 0975–8887.
9. Zhang, D., & Han, J. (2015). A pipelined and memory-optimized FPGA implementation of real-time edge detection. IEEE Transactions on Industrial Informatics, 11(1), 235–243.
10. Ouyang, J., & Vasilakos, A. (2014). FPGA-based real-time image edge detection using efficient hardware structure. Microprocessors and Microsystems, 38(8), 1121–1128.
11. Bala, P., & Chaudhary, S. (2017). Design and implementation of high-speed edge detection filters on FPGA. International Journal of Engineering Trends and Technology, 43(2), 105–110.
12. Milinković, D., & Živanović, M. (2021). Real-time edge detection using FPGA with resource-efficient architecture. Electronics, 10(3), 255.
13. Suresh, B., & Thilagavathi, M. (2014). Hardware implementation of edge detection using Xilinx system generator for DSP. Journal of Signal and Image Processing, 5(2), 82–89.
14. Pereira, H., & Silva, V. (2015). FPGA-based implementation of a high-performance edge detection system for embedded vision. Journal of Systems Architecture, 61(5–6), 241–253.
15. Wei, L., & Zhang, W. (2020). An efficient Verilog HDL design of edge detection system using Sobel operator with optimized resource utilization on FPGA. Advances in Electrical and Computer Engineering, 20(3), 47–54.

**BIO DATA**

Name: Jayakrishnan Menon

Registration No: 22BEC1205

Email: [jayakrishnan.menon2022@vitstudent.ac.in](mailto:jayakrishnan.menon2022@vitstudent.ac.in)

Mobile number: 7200323590

Name: Kalyaneswar Pulipati

Registration No: 22BEC1219

Email: [pulipati.k2022@vitstudent.ac.in](mailto:pulipati.k2022@vitstudent.ac.in)

Mobile number: 6281484142