SOBEL EDGE DETECTOR

DCD Mini Project | 28-11-2022
Submitted for mini project of DCD Laboratory as partial fulfillment of B.Tech Course

Submitted by 2nd year - ECE A-section – Batch 2
Gyana Chandu – 21ECB0A73
Sri Abhiraami K – 21ECB0F01
Shiva Santhosh- 21ECB0F02

Submitted to:
Dr Prithvi Pothupogu
Dr Vadthiya Narendar

November, 2022

Index

Abstract

Problem Statement:

Simulating a digital circuit, specially designed for Sobel Edge Detection for image edge detection in Xilinx vivado suite.

Introduction

Image edge detection is a process of locating the edge of an image which is important in finding the approximate absolute gradient magnitude at each point I of an input grayscale image.

The problem of getting an appropriate absolute gradient magnitude for edges lies in the method used. The Sobel operator performs a 2-D spatial gradient measurement on images. Transferring a 2-D pixel array into statistically uncorrelated data set enhances the removal of redundant data, as a result, reduction of the amount of data is required to represent a digital image.

Methodology

The project will be implemented using Verilog as programming language. The Sobel edge detector uses a pair of 3 x 3 convolution masks, one estimating gradient in the x-direction and the other estimating gradient in y–direction. The Sobel detector is incredibly sensitive to noise in pictures, it effectively highlights them as edges. Hence, Sobel operator is recommended in massive data communication found in data transfer.

Outcome

We will see the result of applying Sobel's edge detection in noised images.

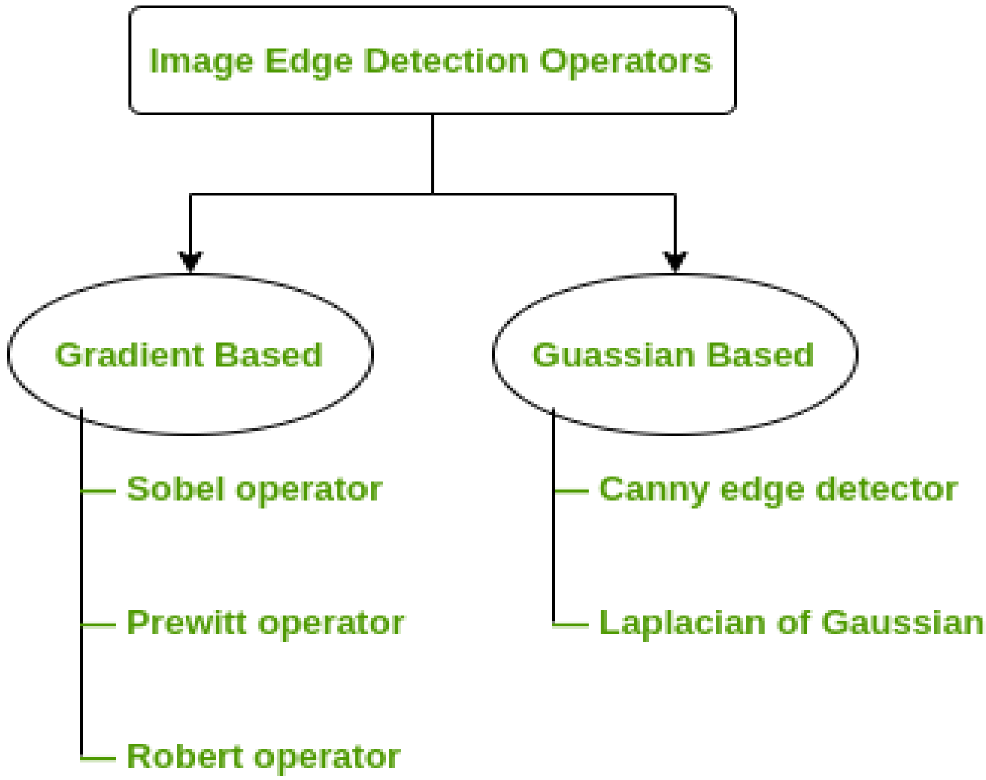
Theory:

- ## What is Edge Detection?

Edge Detection is when we use matrix math to calculate areas of different intensities of an image. Areas where there are extreme differences in the intensities of the pixel usually indicate an edge of an object.

After finding all of the large differences in intensities in a picture, we have discovered all of the edges in the picture. Sobel Edge detection is a widely used algorithm of edge detection in image processing.

Along with Canny and Prewitt, Sobel is one of the most popular edge detection algorithms used in today's technology.



- ## Steps in Edge Detection

Algorithms for edge detection contain three steps:

➢ Filtering:

Since gradient computation based on intensity values of only two points are susceptible to noise and other vagaries in discrete computations, filtering is commonly used to improve the performance of an edge detector with respect to noise. More filtering to reduce noise results in a loss of edge strength.

- **Enhancement:**

  In order to facilitate the detection of edges, it is essential to determine changes in intensity in the neighborhood of a point. Enhancement emphasizes pixels where there is a significant change in local intensity values and is usually performed by computing the gradient magnitude.

- **Detection:**

  We only want points with strong edge content. However, many points in an image have a nonzero value for the gradient, and not all of these points are edges for a particular application. Therefore, some method should be used to determine which points are edge points. Frequently, thresholding provides the criterion used for detection.

- **Localization:**

  The location of the edge can be estimated with subpixel resolution if required for the application. The edge orientation can also be estimated.

- It is important to note that detection merely indicates that an edge is present near a pixel in an image, but does not necessarily provide an accurate estimate of edge location or orientation. The errors in edge detection are errors of misclassification: false edges and missing edges.

Sobel Edge Detection:

- Sobel Edge Detection uses a filter that gives more emphasis to the centre of the filter. A special gradient magnitude operation has been done in both horizontal and vertical directions.

- Compared to other edge operator, Sobel has two main advantages:
  - Since the introduction of the average factor, it has some smoothing effect to the random noise of the image.
  - Because it is the differential of two rows or two columns, so the elements of the edge on both sides has been enhanced, so that the edge seems thick and bright.

Sobel Operator:

- The Sobel–Feldman operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations.

- The operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivations – one for horizontal changes, and one for vertical.

- If we define A as the source image, and Gx and Gy are two images which at each point contain the horizontal and vertical derivative approximations respectively, the computations are as follows:

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

- Since the Sobel kernels can be decomposed as the products of an averaging and a differentiation kernel, they compute the gradient with smoothing. For example, Gx can be written as

$$\mathbf{G}_x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \left(\begin{bmatrix} +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \right) \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix} * \left(\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \mathbf{A} \right)$$

- These kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations.

- The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these Gx and Gy).

- 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:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

- The angle of orientation of the edge (relative to the pixel grid) giving rise to the spatial gradient is given by:

$$\theta = \arctan(Gy/Gx)$$

- In this case, orientation 0 is taken to mean that the direction of maximum contrast from black to white runs from left to right on the image, and other angles are measured anti-clockwise.

- Often, this absolute magnitude is the only output the user sees the two components of the gradient are conveniently computed and added in a single pass over the input image using the pseudo-convolution operator.

P_1	P_2	P_3
P_4	P_5	P_6
P_7	P_8	P_9

$$|G| = |(P_1 + 2 \times P_2 + P_3) - (P_7 + 2 \times P_8 + P_9)| + |(P_3 + 2 \times P_6 + P_9) - (P_1 + 2 \times P_4 + P_7)|$$

Method of the Filter Design:

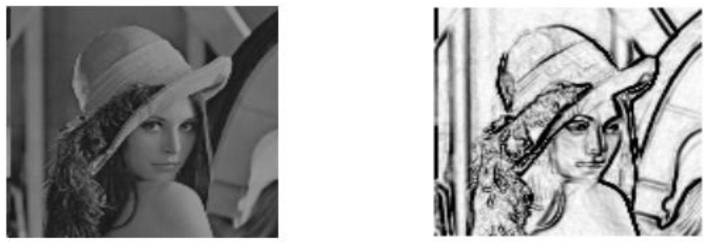
There are many methods of detecting edges; the majority of different methods may be grouped into these two categories:

- **Gradient:**
  The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image. For example, Roberts, Prewitt, Sobel where detected features have very sharp edges.

- **Laplacian:**
  The Laplacian method searches for zero crossings in the second derivative of the image to find edges e.g. Marr-Hildreth, Laplacian of Gaussian etc. An edge has one-dimensional shape of a ramp and calculating the derivative of the image can highlight its location.

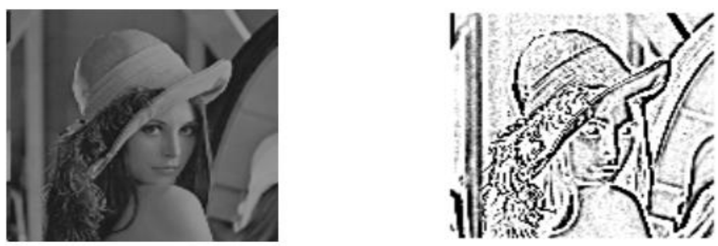
Edges may be viewpoint dependent: these are edges that may change as the viewpoint changes and typically reflect the geometry of the scene which in turn reflects the properties of the viewed objects such as surface markings and surface shape.

A typical edge might be the border between a block of red color and a block of yellow, in contrast. However, what happens when one looks at the pixels of that image is that all visible portion of one edge are compacted.



Input Image Output Edges

Figure 1: The Gradient Method



Input Image Output Edges

Figure 2: the Laplacian Method

- Pseudo-codes for Sobel edge detection method

Input: A Sample Image

Output: Detected Edges

Step 1: Accept the input image

Step 2: Apply mask Gx, Gy to the input image

Step 3: Apply Sobel edge detection algorithm and the gradient

Step 4: Masks manipulation of Gx,Gy separately on the input image

Step 5: Results combined to find the absolute magnitude of the gradient

$$|G| = \sqrt{Gx^2 + Gy^2}$$

Step 6: the absolute magnitude is the output edges

CODE

```
module imageProcessTop(
input   axi_clk,
input   axi_reset_n,
//slave interface
input   i_data_valid,
input [7:0] i_data,
output  o_data_ready,
//master interface
output  o_data_valid,
output [7:0] o_data,
input   i_data_ready,

//interrupt
output  o_intr
);
wire [71:0] pixel_data;
wire pixel_data_valid;
wire axis_prog_full;
wire [7:0] convolved_data;
wire convolved_data_valid;
assign o_data_ready = !axis_prog_full;
    imageControl IC(
    .i_clk(axi_clk),
    .i_rst(!axi_reset_n),
```

```
    .i_pixel_data(i_data),
    .i_pixel_data_valid(i_data_valid),
    .o_pixel_data(pixel_data),
    .o_pixel_data_valid(pixel_data_valid),
    .o_intr(o_intr)
  );
 conv conv(
     .i_clk(axi_clk),
     .i_pixel_data(pixel_data),
     .i_pixel_data_valid(pixel_data_valid),
     .o_convolved_data(convolved_data),
     .o_convolved_data_valid(convolved_data_valid)
 );
outputBuffer OB (
   .wr_rst_busy(),        // output wire wr_rst_busy
   .rd_rst_busy(),        // output wire rd_rst_busy
   .s_aclk(axi_clk),                  // input wire s_aclk
   .s_aresetn(axi_reset_n),           // input wire s_aresetn
   .s_axis_tvalid(convolved_data_valid),      //  input  wire
s_axis_tvalid
   .s_axis_tready(),    // output wire s_axis_tready
   .s_axis_tdata(convolved_data),      // input wire [7 : 0]
s_axis_tdata
   .m_axis_tvalid(o_data_valid),           //   output   wire
m_axis_tvalid
```

```
   .m_axis_tready(i_data_ready),            //   input   wire
m_axis_tready
   .m_axis_tdata(o_data),          //  output  wire  [7  :  0]
m_axis_tdata
   .axis_prog_full(axis_prog_full)       //    output    wire
axis_prog_full
 );
Endmodule
module imageControl(
input                    i_clk,
input                    i_rst,
input [7:0]              i_pixel_data,
input                    i_pixel_data_valid,
output reg [71:0]        o_pixel_data,
output                   o_pixel_data_valid,
output reg               o_intr
);
reg [8:0] pixelCounter;
reg [1:0] currentWrLineBuffer;
reg [3:0] lineBuffDataValid;
reg [3:0] lineBuffRdData;
reg [1:0] currentRdLineBuffer;
wire [23:0] lb0data;
wire [23:0] lb1data;
wire [23:0] lb2data;
wire [23:0] lb3data;
```

```
reg [8:0] rdCounter;
reg rd_line_buffer;
reg [11:0] totalPixelCounter;
reg rdState;
localparam IDLE = 'b0,
           RD_BUFFER = 'b1;
assign o_pixel_data_valid = rd_line_buffer;
always @(posedge i_clk)
begin
    if(i_rst)
        totalPixelCounter <= 0;
    else
    begin
        if(i_pixel_data_valid & !rd_line_buffer)
            totalPixelCounter <= totalPixelCounter + 1;
        else if(!i_pixel_data_valid & rd_line_buffer)
            totalPixelCounter <= totalPixelCounter - 1;
    end
end

always @(posedge i_clk)
begin
    if(i_rst)
    begin
        rdState <= IDLE;
```

```
        rd_line_buffer <= 1'b0;
        o_intr <= 1'b0;
    end
    else
    begin
        case(rdState)
            IDLE:begin
                o_intr <= 1'b0;
                if(totalPixelCounter >= 1536)
                begin
                    rd_line_buffer <= 1'b1;
                    rdState <= RD_BUFFER;
                end
            end
            RD_BUFFER:begin
                if(rdCounter == 511)
                begin
                    rdState <= IDLE;
                    rd_line_buffer <= 1'b0;
                    o_intr <= 1'b1;
                end
            end
        endcase
    end
end
```

```
always @(posedge i_clk)
begin
    if(i_rst)
        pixelCounter <= 0;
    else
    begin
        if(i_pixel_data_valid)
            pixelCounter <= pixelCounter + 1;
    end
end


always @(posedge i_clk)
begin
    if(i_rst)
        currentWrLineBuffer <= 0;
    else
    begin
        if(pixelCounter == 511 & i_pixel_data_valid)
            currentWrLineBuffer <= currentWrLineBuffer+1;
    end
end
always @(*)
begin
```

```
    lineBuffDataValid = 4'h0;
    lineBuffDataValid[currentWrLineBuffer] =
i_pixel_data_valid;
end
always @(posedge i_clk)
begin
    if(i_rst)
        rdCounter <= 0;
    else
    begin
        if(rd_line_buffer)
            rdCounter <= rdCounter + 1;
    end
end
always @(posedge i_clk)
begin
    if(i_rst)
    begin
        currentRdLineBuffer <= 0;
    end
    else
    begin
        if(rdCounter == 511 & rd_line_buffer)
            currentRdLineBuffer <= currentRdLineBuffer + 1;
    end
```

```
end



always @(*)
begin
    case(currentRdLineBuffer)
        0:begin
            o_pixel_data = {lb2data,lb1data,lb0data};
        end
        1:begin
            o_pixel_data = {lb3data,lb2data,lb1data};
        end
        2:begin
            o_pixel_data = {lb0data,lb3data,lb2data};
        end
        3:begin
            o_pixel_data = {lb1data,lb0data,lb3data};
        end
    endcase
end
always @(*)
begin
    case(currentRdLineBuffer)
        0:begin
            lineBuffRdData[0] = rd_line_buffer;
```

```
            lineBuffRdData[1] = rd_line_buffer;
            lineBuffRdData[2] = rd_line_buffer;
            lineBuffRdData[3] = 1'b0;
        end
       1:begin
            lineBuffRdData[0] = 1'b0;
            lineBuffRdData[1] = rd_line_buffer;
            lineBuffRdData[2] = rd_line_buffer;
            lineBuffRdData[3] = rd_line_buffer;
        end
       2:begin
             lineBuffRdData[0] = rd_line_buffer;
             lineBuffRdData[1] = 1'b0;
             lineBuffRdData[2] = rd_line_buffer;
             lineBuffRdData[3] = rd_line_buffer;
       end
      3:begin
             lineBuffRdData[0] = rd_line_buffer;
             lineBuffRdData[1] = rd_line_buffer;
             lineBuffRdData[2] = 1'b0;
             lineBuffRdData[3] = rd_line_buffer;
       end
    endcase
end
lineBuffer lB0(
```

```
    .i_clk(i_clk),
    .i_rst(i_rst),
    .i_data(i_pixel_data),
    .i_data_valid(lineBuffDataValid[0]),
    .o_data(lb0data),
    .i_rd_data(lineBuffRdData[0])
);
 lineBuffer lB1(
     .i_clk(i_clk),
     .i_rst(i_rst),
     .i_data(i_pixel_data),
     .i_data_valid(lineBuffDataValid[1]),
     .o_data(lb1data),
     .i_rd_data(lineBuffRdData[1])
 );
 lineBuffer lB2(
      .i_clk(i_clk),
      .i_rst(i_rst),
      .i_data(i_pixel_data),
      .i_data_valid(lineBuffDataValid[2]),
      .o_data(lb2data),
      .i_rd_data(lineBuffRdData[2])
  );

  lineBuffer lB3(
```

```
        .i_clk(i_clk),
        .i_rst(i_rst),
        .i_data(i_pixel_data),
        .i_data_valid(lineBuffDataValid[3]),
        .o_data(lb3data),
        .i_rd_data(lineBuffRdData[3])
    );
  Endmodule

module conv(
input        i_clk,
input [71:0] i_pixel_data,
input        i_pixel_data_valid,
output reg [7:0] o_convolved_data,
output reg   o_convolved_data_valid
    );

integer i;
reg [7:0] kernel1 [8:0];
reg [7:0] kernel2 [8:0];
reg [10:0] multData1[8:0];
reg [10:0] multData2[8:0];
reg [10:0] sumDataInt1;
reg [10:0] sumDataInt2;
reg [10:0] sumData1;
```

```
reg [10:0] sumData2;
reg multDataValid;
reg sumDataValid;
reg convolved_data_valid;
reg [20:0] convolved_data_int1;
reg [20:0] convolved_data_int2;
wire [21:0] convolved_data_int;
reg convolved_data_int_valid;
initial
begin
    kernel1[0] =  1;
    kernel1[1] =  0;
    kernel1[2] = -1;
    kernel1[3] =  2;
    kernel1[4] =  0;
    kernel1[5] = -2;
    kernel1[6] =  1;
    kernel1[7] =  0;
    kernel1[8] = -1;
    kernel2[0] =  1;
    kernel2[1] =  2;
    kernel2[2] =  1;
    kernel2[3] =  0;
    kernel2[4] =  0;
    kernel2[5] =  0;
```

```
    kernel2[6] = -1;
    kernel2[7] = -2;
    kernel2[8] = -1;
end
always @(posedge i_clk)
begin
    for(i=0;i<9;i=i+1)
    begin
        multData1[i] <= $signed(kernel1[i])*$signed({1'b0,i_pixel_data[i*8+:8]});
        multData2[i] <= $signed(kernel2[i])*$signed({1'b0,i_pixel_data[i*8+:8]});
    end
    multDataValid <= i_pixel_data_valid;
end
always @(*)
begin
    sumDataInt1 = 0;
    sumDataInt2 = 0;
    for(i=0;i<9;i=i+1)
    begin
        sumDataInt1 = $signed(sumDataInt1) + $signed(multData1[i]);
        sumDataInt2 = $signed(sumDataInt2) + $signed(multData2[i]);
    end
```

```
end
always @(posedge i_clk)
begin
    sumData1 <= sumDataInt1;
    sumData2 <= sumDataInt2;
    sumDataValid <= multDataValid;
end
always @(posedge i_clk)
begin
    convolved_data_int1                                   <=
$signed(sumData1)*$signed(sumData1);
    convolved_data_int2                                   <=
$signed(sumData2)*$signed(sumData2);
    convolved_data_int_valid <= sumDataValid;
end
assign                  convolved_data_int                 =
convolved_data_int1+convolved_data_int2;
always @(posedge i_clk)
begin
    if(convolved_data_int > 4000)
        o_convolved_data <= 8'hff;
    else
        o_convolved_data <= 8'h00;
    o_convolved_data_valid <= convolved_data_int_valid;
end
endmodule
```

Test bench

```
`define headerSize 1080
`define imageSize 512*512

module tb(  );

 reg clk;
 reg reset;
 reg [7:0] imgData;
 integer file,file1,i;
 reg imgDataValid;
 integer sentSize;
 wire intr;
 wire [7:0] outData;
 wire outDataValid;
 integer receivedData=0;
 initial
 begin
    clk = 1'b0;
    forever
    begin
        #5 clk = ~clk;
    end
 end
 initial
```

```
begin
   reset = 0;
   sentSize = 0;
   imgDataValid = 0;
   #100;
   reset = 1;
   #100;
   file = $fopen("lena_gray.bmp","rb");
   file1 = $fopen("blurred_lena.bmp","wb");
   for(i=0;i<`headerSize;i=i+1)

   begin
       $fscanf(file,"%c",imgData);
       $fwrite(file1,"%c",imgData);
   end
   for(i=0;i<4*512;i=i+1)
   begin
       @(posedge clk);
       $fscanf(file,"%c",imgData);
       imgDataValid <= 1'b1;
   end
   sentSize = 4*512;
   @(posedge clk);
   imgDataValid <= 1'b0;
   while(sentSize < `imageSize)
```

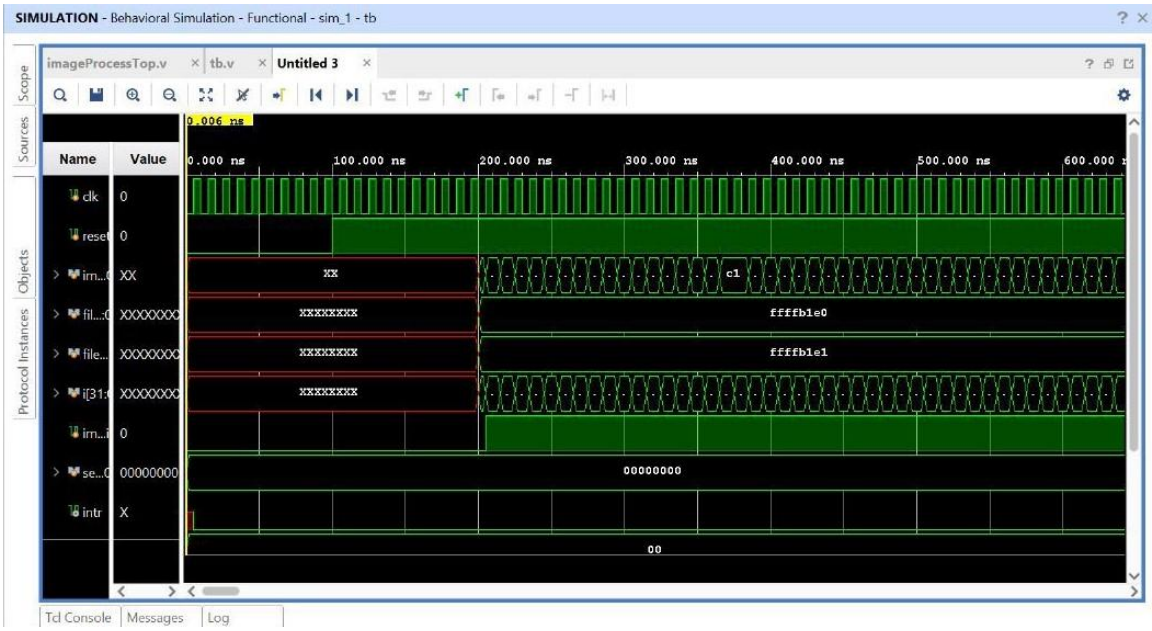
```
begin
    @(posedge intr);
    for(i=0;i<512;i=i+1)
    begin
        @(posedge clk);
        $fscanf(file,"%c",imgData);
        imgDataValid <= 1'b1;
    end

    @(posedge clk);
    imgDataValid <= 1'b0;
    sentSize = sentSize+512;
end

@(posedge clk);
imgDataValid <= 1'b0;
@(posedge intr);
for(i=0;i<512;i=i+1)
begin
    @(posedge clk);
    imgData <= 0;
    imgDataValid <= 1'b1;
end
@(posedge clk);
imgDataValid <= 1'b0;
```

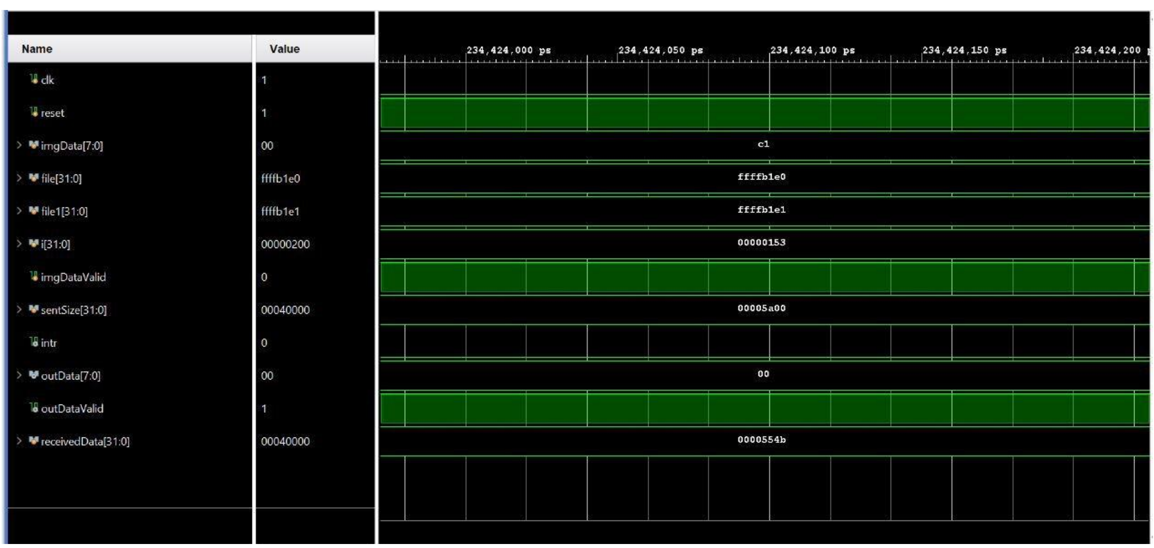
```
    @(posedge intr);
    for(i=0;i<512;i=i+1)
    begin

        @(posedge clk);
        imgData <= 0;
        imgDataValid <= 1'b1;
    end

    @(posedge clk);
    imgDataValid <= 1'b0;
    $fclose(file);
end

always @(posedge clk)
begin
    if(outDataValid)
    begin
        $fwrite(file1,"%c",outData);
        receivedData = receivedData+1;
    end



    if(receivedData == `imageSize)
    begin
```

```
        $fclose(file1);
        $stop;
     end
 end

 imageProcessTop dut(
    .axi_clk(clk),
    .axi_reset_n(reset),
    //slave interface
    .i_data_valid(imgDataValid),
    .i_data(imgData),
    .o_data_ready(),
    //master interface
    .o_data_valid(outDataValid),
    .o_data(outData),
    .i_data_ready(1'b1),
    //interrupt
    .o_intr(intr)
);
endmodule
```

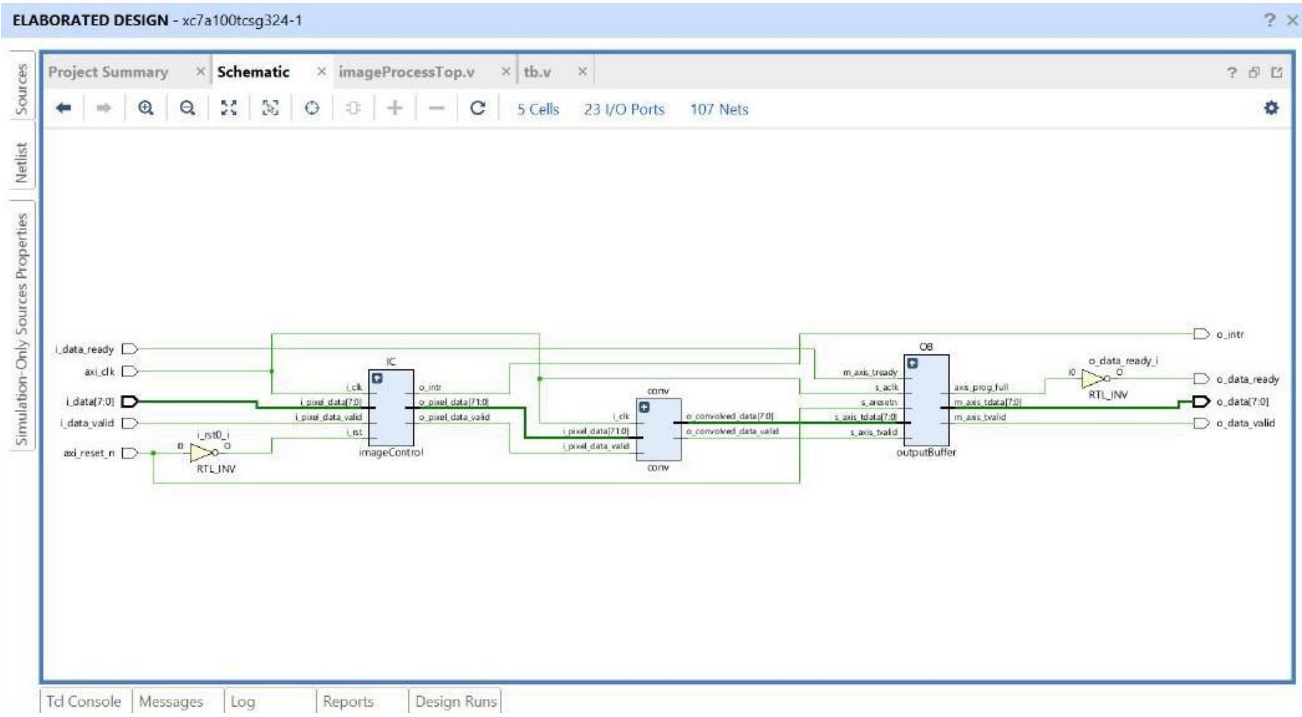
Result

Simulation:

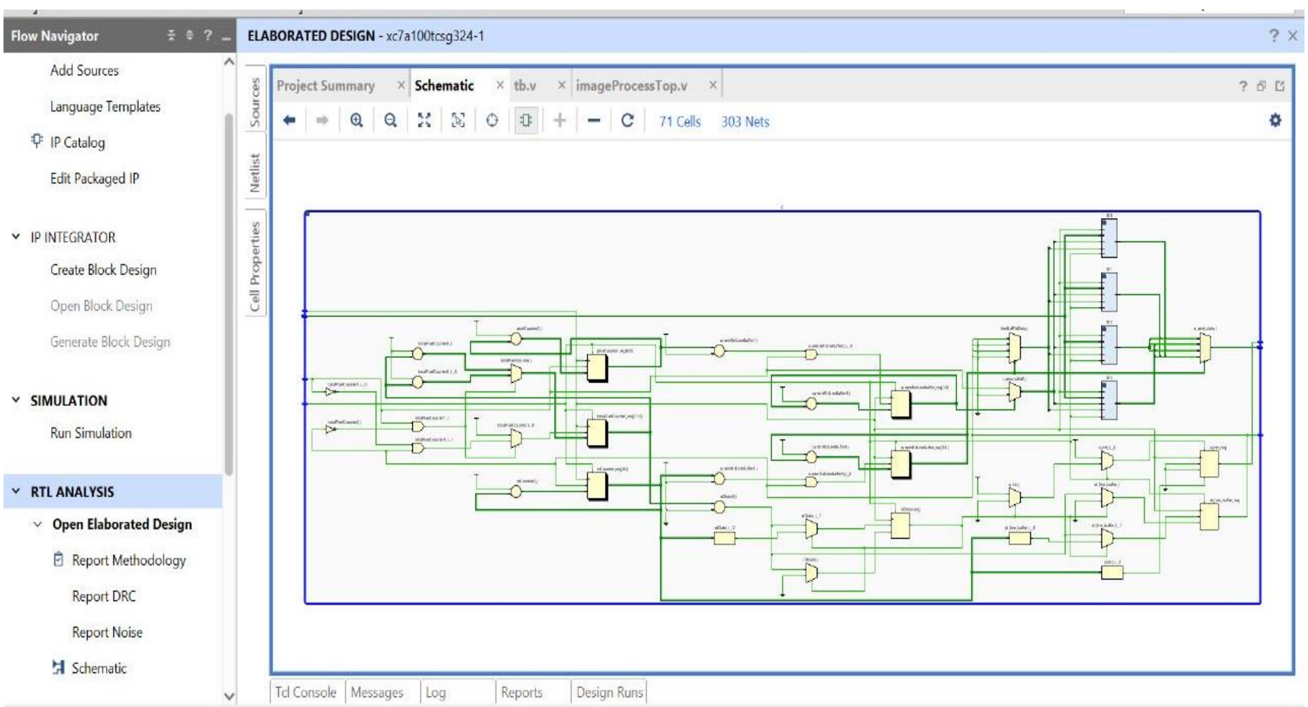


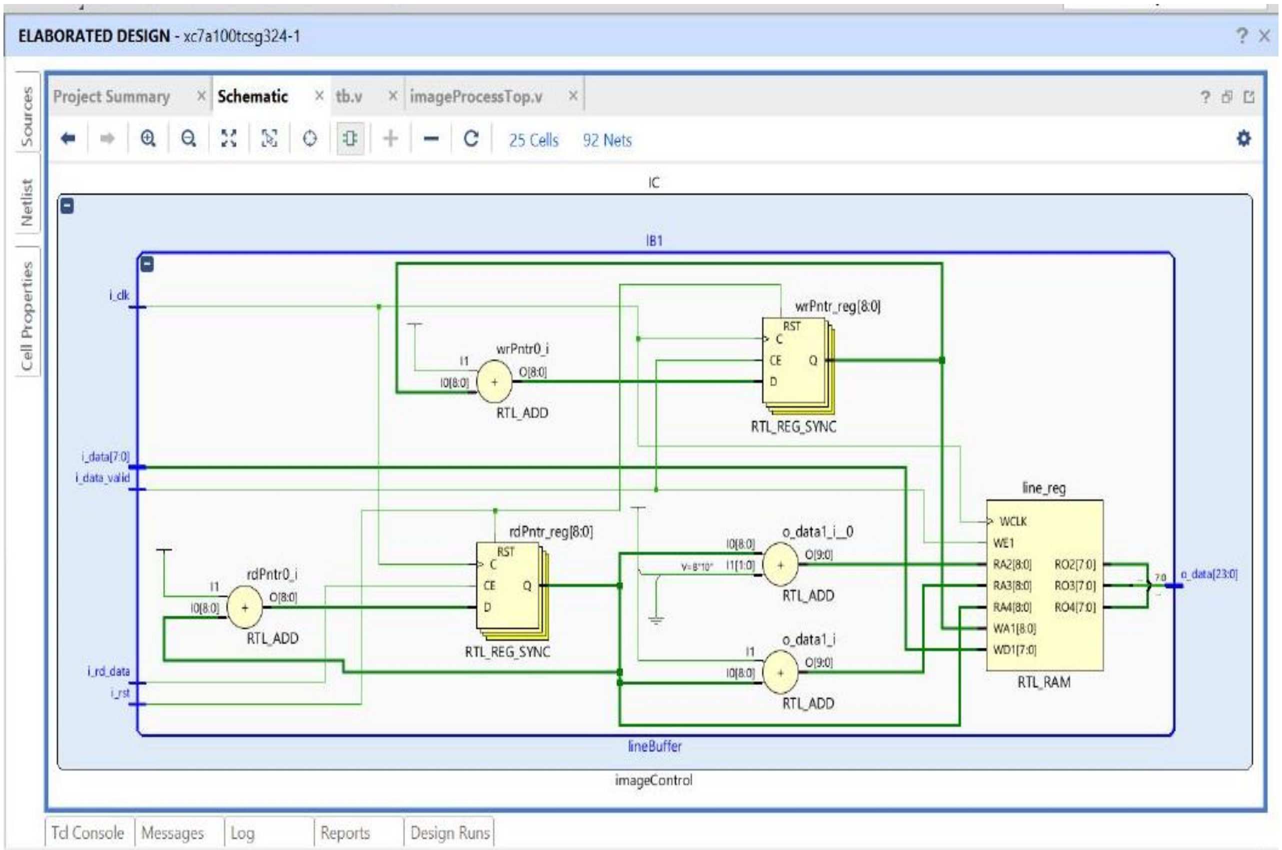


Block Diagram:



Schematic Diagram:





Practical Implications and Importance of Edge Detection

The following advantages of Sobel edge detector justify its superiority over other edge detection techniques:

- **Edge Orientation:**
  The geometry of the operator determines a characteristic direction in which it is most sensitive to edges. Operators can be optimized to look for horizontal, vertical, or diagonal edges.

- **Noise Environment:**
  Edge detection is difficult in noisy images, since both the noise and the edges contain high-frequency content. Attempts to reduce the noise result in blurred and distorted edges. Operators used on noisy images are typically larger in scope, so they can average enough data to discount localized noisy pixels. This results in less accurate localization of the detected edges.

- **Edge Structure:**
  Not all edges involve a step change in intensity. Effects such as refraction or poor focus can result in objects with boundaries defined by a gradual change in intensity. The operator is chosen to be responsive to such a gradual change in those cases. Newer wavelet-based techniques actually characterize the nature of the transition for each edge in order to distinguish, for example, edges associated with hair from edges associated with a face.

Future Scope

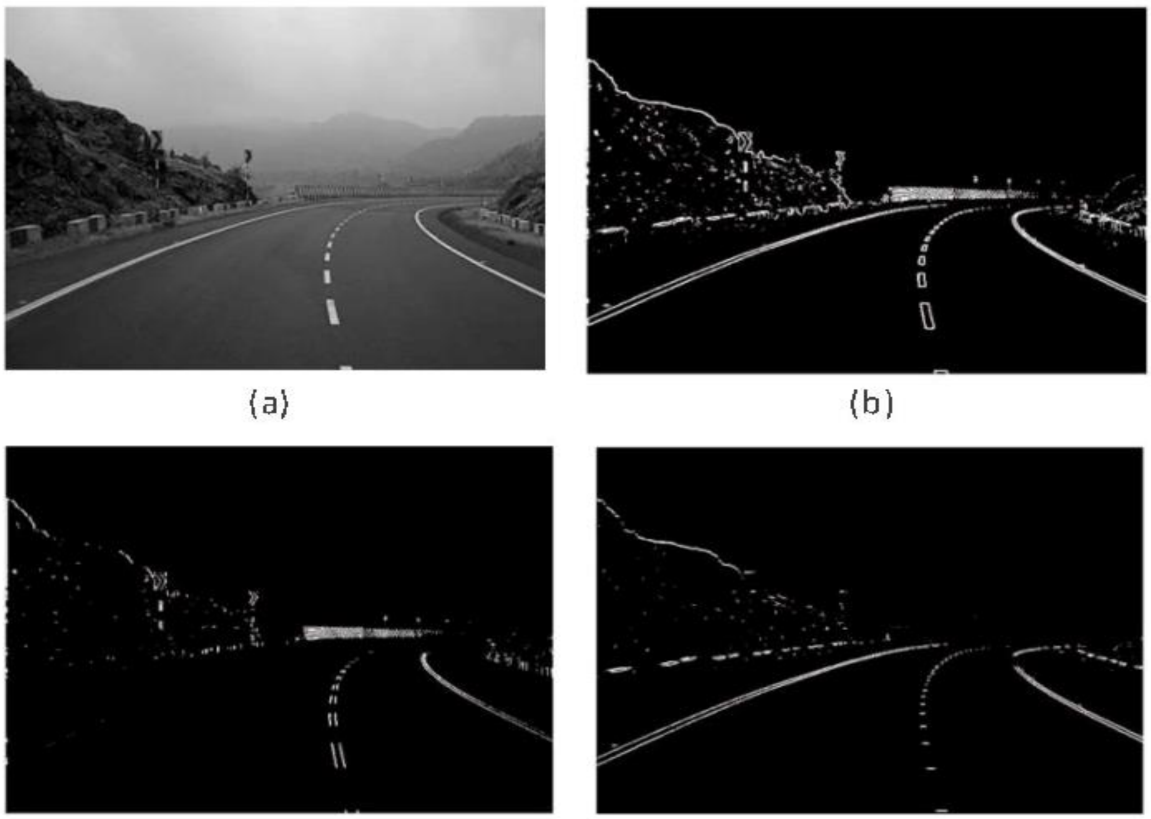
- The implementation of both Median filtering and Sobel edge detection is done on grayscale 100x100 pixels image.

- Furthermore implementation of RGB image processing can be done.

- The size of the image can also be increased using dividing the bigger image into smaller sister images and processing them individually.

Real Life Applications

- Computer Vision
- Anomalies in Medical Image
- video surveillance
- Lane Detection warning systems to detect edges of lanes.



(a) (b) (c) (d)

Conclusion:

The Sobel operator is based on convolving the image with a small, separable, and integer valued filter in horizontal and vertical direction and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation which it produces is relatively crude, in particular for high frequency variations in the image.