IMAGE PROCESSING EDGE DETECTION

Made By:

Lakshya Singh(201901248)

Guided By -

Prof. - Yash Agrawal

Course-

EL467 (DIGITAL PROGRAMMING)

Abstract

In computer vision and image processing, edge detection of images is crucial. Real-time applications require hardware implementation of picture edge detection, which is utilized to accelerate processing. The Field Programmable Gate Array (FPGA) is essential to the hardware implementation of image processing applications. The suggested work implements image edge detection on an FPGA. Utilizing the most effective tool, Xilinx System Generator, the edge detection technique is implemented on hardware (XSG). The Xilinx System Generator (XSG) tool is used for FPGA programming and system modeling. The Xilinx System Generator tool is a new image processing programme that provides a model based design for processing. This paper aims at developing algorithmic models in MATLAB using Xilinx blockset for a specific role then creating workspace in MATLAB to process image pixels and performing hardware implementation on FPGA.

INTRODUCTION

- **Edge detection is an image processing technique for finding the boundaries of the object within an image.**
- **Edge detection is the name for a set of mathematical methods which aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities.**
- **Edge detection is a necessary tool used in image processing applications to obtain information from the frame.**
- **This process detects boundaries between objects and the background in the image.**
- **Edge detection is one of the most common preprocessing steps in image processing algorithms such as image enhancement , image segmentation, tracking and image/video coding ,it provokes great interest in the research fraternity.**

EDGE DETECTION ALGORITHMS

1. **Sobel Operator**
2. **Canny's Operator**

Sobel Operator

- **Sobel edge detection gives an averaging and smoothing effect over the image therefore it also takes care of the noise present in the image.**
- **This technique extracts all of the edges in an image, regardless of its direction.**
- **This includes a pair of 3•3 convolution masks.**
- **One mask is simple and the other rotated by 90°.**
- **The masks can be applied to the input image, to produce separate measurements of the gradient component in each direction.**
- **These masks are created to respond to edges running horizontally and vertically.**
- **The Sobel edge detector uses the masks shown in the above figure.**

-1	-2	-1
0	0	0
1	2	1

Gx

-1	0	1
-2	0	2
-1	0	1

Gv

- **The masks can be applied differently to the image, to produce separate measurements of tin gradient.**

Canny edge detection

- **A Canny edge detector is a multi-step algorithm to detect the edges for any input image.**

- **The algorithm runs in 5 separate steps:**

  1. Smoothing:

  - **Blurring of the image to remove noise.**
  - **It is inevitable that all images taken from a camera will contain some amount of noise. To prevent that noise from being mistaken for edges, noise must be reduced.**
  - **Therefore the image is first smoothed by applying a Gaussian filter.**
  - **The kernel of a Gaussian filter with a standard deviation of $\sigma = 1.4$ is shown in Equation (1).**

  2. Finding gradients:

  - **The edges should be marked where the gradients of the image have large magnitudes.**
  - **The Canny algorithm basically finds edges where the grayscale intensity of the image changes the most.**
  - **These areas are found by determining gradients of the image.**
  - **Basically this is done by preserving all local maxima in the gradient image, and deleting everything else.**

  3. Non-maximum suppression:

  - **Only local maxima should be marked as edges.**
  - **The purpose of this step is to convert the "blurred" edges in the image of the gradient magnitudes to "sharp" edges.**

4. Double thresholding:

- **Potential edges are determined by thresholding.**
- **The edge-pixels remaining after the non-maximum suppression step are (still) marked with their strength pixel-by-pixel.**
- **Many of these will probably be true edges in the image, but some may be caused by noise or color variations for instance due to rough surfaces.**

5. Edge tracking by hysteresis:

- **Final edges are determined by suppressing all edges that are not connected to a very certain (strong) edge.**
- **Weak edges are included if and only if they are connected to strong edges.**

The entire operation of edge detection using Simulink and Xilinx blocks goes through three phases-

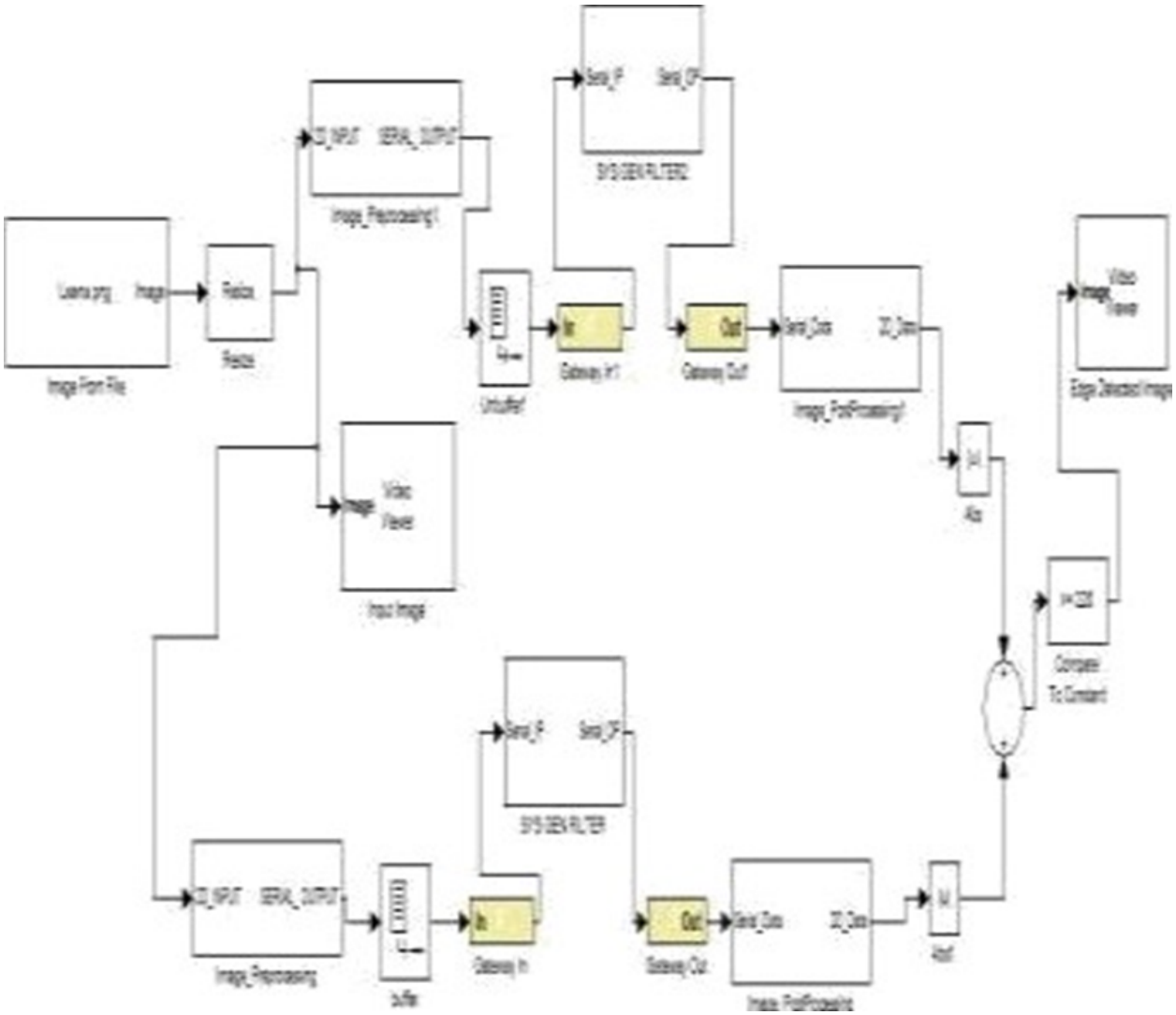
1. **Image Pre-Processing Blocks**
2. **Edge Detection using XSCi**
3. **Image Post-Processing Blocks**

1. Image Pre-Processing Blocks

- **As an image is two dimensional (2D) array size with R*C where R,C represent the tin row and column of an image respectively.**
- **Image pre- processing blocks are used to convert 2D image data to an ID array which is shown in Fig.**
- **Image Pre-processing blocks include the Transpose, Convert 2-D to I -D, Frame conversion and Unbuffed block.**
- **The transpose block transposes tin R*C input image matrix into C*R sized matrix, Convert 2-D to I -D block reshapes a C*R matrix input to a 1-D rector.**
- **Frame conversion block set the output signal to frame based data and provides an unbuffed block which converts this frame to sector samples output samples at a higher sampling rate.**

2. Edge Detection using X SQL

- **Figure shows the model based design using Xilinx system generator blocks for processing the input image for edge detection.**
- **To perform the edge detection, the input image is convolved with the horizontal and vertical mask which is made up of delay block, adder and subtractor.**



3.Image Post-Processing Block sets

- **Image post-processing is as shown in figure 5. Image post processing helps recreating images from ID arrays.**
- **Primary significance of image post-processing is to make processed data available and with suitable data rate.**
- **It also includes a buffer block which converts sector samples to frame output at lower sampling rate, followed by convert I -D to 2-D,**

Verilog Code for Sobel Edge Detection

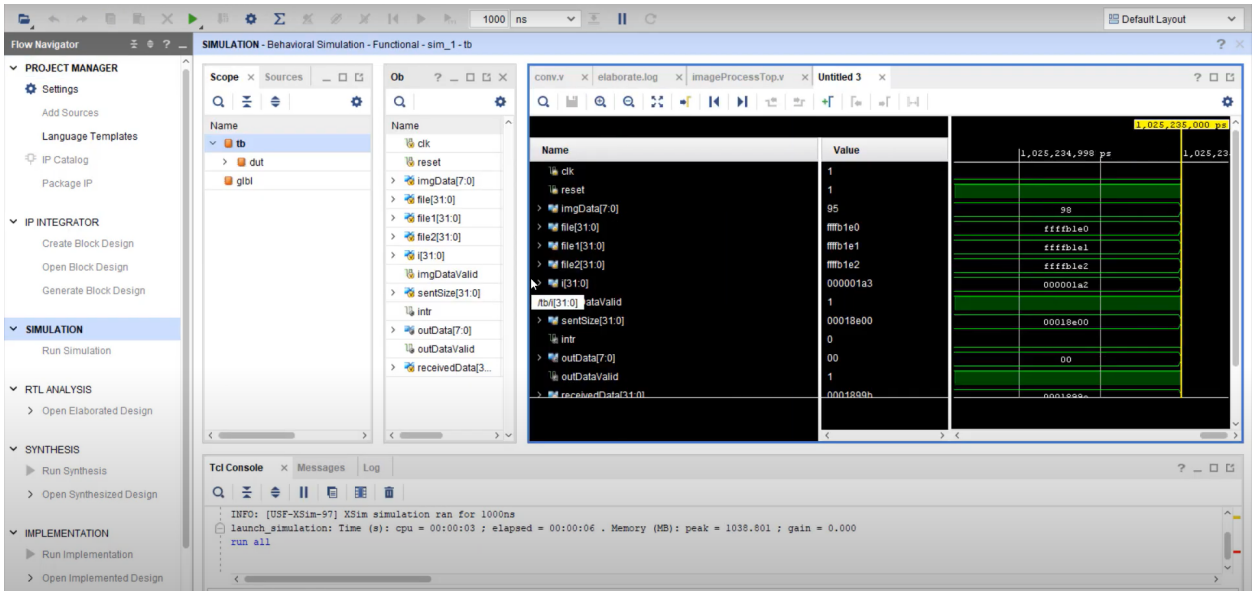
```
module conv(
input       i_clk,
input [71:0] i_pixel_data,
input       i_pixel_data_valid,
output reg [7:0] o_convolved_data,
output reg   o_convolved_data_valid
   );

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;
reg [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;

kernel1[0] =1;
kernel1[1] =2;
kernel1[2] =1;
kernel1[3] =0;
kernel1[4] =0;
kernel1[5] =0;
kernel1[6] =-1;
kernel1[7] =-2;
kernel1[8] =-1;
end

always @(posedge i_clk)
begin
   for(i=0;i<9;i=i+1)
   begin
      multData[i] <= $signed(kernel1[i])*$signed({1'b0,i_pixel_data[i*8+:8]});
 multData[i] <= $signed(kernel2[i])*$signed({1'b0,i_pixel_data[i*8+:8]});
```

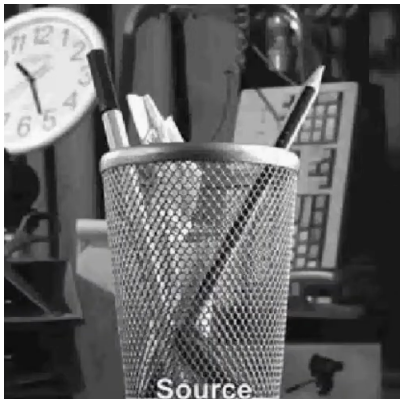
```
   end
   multDataValid <= i_pixel_data_valid;
end


always @(*)
begin
   sumDataInt1= 0;
sumDataInt2= 0;
   for(i=0;i<9;i=i+1)
   begin
      sumDataInt1 = $signed(sumDataInt1) + $signed(multData[i]);
sumDataInt2 = $signed(sumDataInt2) + $signed(multData[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_valid <= sumDataValid;
end

assign convolved_data_int = convolved_data_int1+ convolved_int2;

always @(posedge i_clk)
begin
   if(convolved_data_int>1000)
        o_convolved_data <= 8*hff;
else
       o_convolved_data <= 8*h00;
     o_convolved_data_valid <= convolved_data_int_valid;
end

Endmodule
```

Graphical Representation

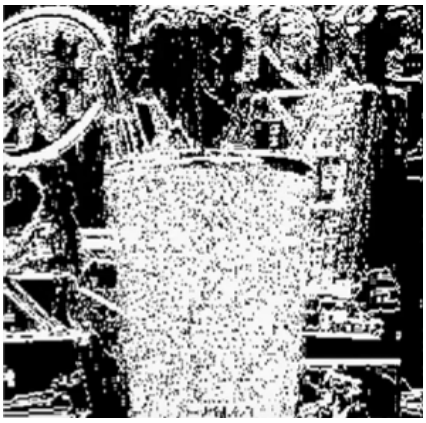
Simulation of Verilog



Result for edge detection from Sobel



INPUT IMAGE



OUTPUT IMAGE

Verilog Code for Canny's edge detection

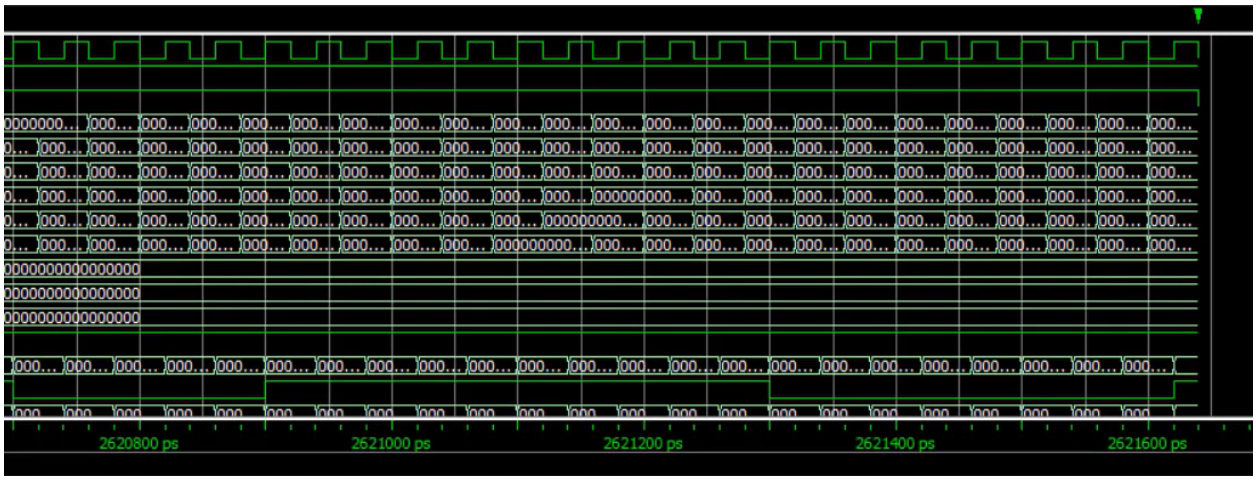
```
module canny(clk,
             reset,
             start,
             im11,
             im21,
             im31,
             im12,
             im22,
             im32,
             im13,
             im23,
             im33,
             dx_out,
             dx_out_sign,
             dy_out,
             dy_out_sign,
             dxy,
             data_occur);



input clk;
input reset;
input start;

input [15:0]im11;
input [15:0]im12;
input [15:0]im13;

input [15:0]im21;
input [15:0]im22;
input [15:0]im23;

input [15:0]im31;
input [15:0]im32;
input [15:0]im33;

output [15:0]dx_out;
output dx_out_sign;
```

```
output [15:0]dy_out;
output dy_out_sign;
output [15:0]dxy;

output data_occur;

wire [15:0]dxy;
wire  [15:0]dx_out;
wire dx_out_sign;

wire  [15:0]dy_out;
wire dy_out_sign;



reg [15:0]im11t;
reg [15:0]im12t;
reg [15:0]im13t;

reg [15:0]im21t;
reg [15:0]im22t;
reg [15:0]im23t;

reg [15:0]im31t;
reg [15:0]im32t;
reg [15:0]im33t;



wire temp3x_sign;
wire [15:0]temp1x,temp2x,temp3x;

wire temp3y_sign;
wire [15:0]temp1y,temp2y,temp3y;



reg data_occur;
wire [15:0]reg_add;



assign dy_out=temp3y;
```

```
assign dy_out_sign=temp3y_sign;
assign temp1y=(im31t+(im32t << 1)+ im33t);
assign temp2y=(im11t+(im12t << 1)+ im13t);
assign temp3y=(temp1y > temp2y)?{temp1y-temp2y}:
        (temp1y < temp2y)?{temp2y-temp1y}:{16'd0};

assign temp3y_sign=(temp1y > temp2y)?1'b1:1'b0;

assign reg_add=(data_occur)?(dx_out+dy_out):16'd0;
assign dxy=(data_occur && reg_add >= 16'd255)?16'd255:16'd0;

assign dx_out=temp3x;
assign dx_out_sign=temp3x_sign;

assign temp1x=(im11t+(im21t << 1)+ im31t);
assign temp2x=(im13t+(im23t << 1)+ im33t);

assign temp3x=(temp1x > temp2x)?{temp1x-temp2x}:
        (temp1x < temp2x)?{temp2x-temp1x}:{16'd0};

assign temp3x_sign=(temp1x > temp2x)?1'b1:1'b0;




always @(posedge clk)
begin

if(~reset)

begin
   im11t<=16'd0;
   im21t<=16'd0;
   im31t<=16'd0;
   im12t<=16'd0;
   im22t<=16'd0;
   im32t<=16'd0;
   im13t<=16'd0;
   im23t<=16'd0;
   im33t<=16'd0;
   data_occur<=1'b0;
```

```
end

else if(start )
begin

   im11t<=im11;
   im21t<=im21;
   im31t<=im31;
   im12t<=im12;
   im22t<=im22;
   im32t<=im32;
   im13t<=im13;
   im23t<=im23;
   im33t<=im33;
   data_occur<=1'b1;
end

else
begin
   im11t<=16'd0;
   im21t<=16'd0;
   im31t<=16'd0;
   im12t<=16'd0;
   im22t<=16'd0;
   im32t<=16'd0;
   im13t<=16'd0;
   im23t<=16'd0;
   im33t<=16'd0;
   data_occur<=1'b0;

end

end

endmodule
```

Graphical Representation



Result for edge detection from Canny's

Input Image

Horizontal Sobel

Vertical Sobel

Output image

CONCLUSION

In this paper, Sobel and Canny edge detection algorithms are designed and implemented on Sobel on Spartan 3E FPGA using Xilinx System Generator and Canny on MATLAB and MobilSIM.
It is simpler to generate a stream of bit files rather than writing thousands of code lines for implementing image processing techniques on FPGA.The PSNR value decreases as the number of pixels of input image increases.

REFERENCES

1.https://towardsai.net/p/computer-vision/what-is-a-canny-edge-detection-algorithm
2.https://www.cse.iitd.ac.in/~pkalra/col783-2017/canny.pdf
3.https://scholarworks.calstate.edu/downloads/j098zh307
4.https://www.academia.edu/36679507/SOBEL_EDGE_DETECTION_USING_HDL_CODES
5.https://www.fpga4student.com/2016/11/image-processing-on-fpga-verilog.html