Canny Edge Detection using High Level Synthesis

By

B. V. Venu Gopal

Index

Introduction

2.1 Introduction

The Project is about implementing Canny edge detection which is a edge detection method used for real time lane detection in High Level Synthesis which is C Synthesis into Hardware Description Langauge.

2.2 High Level Synthesis

High-level synthesis (HLS), sometimes referred to as C synthesis, electronic system-level (ESL) synthesis, algorithmic synthesis, or behavioral synthesis, is an automated design process that interprets an algorithmic description of a desired behavior and creates digital hardware that implements that behavior. The code is analyzed, architecturally constrained, and scheduled to transcompile into a register-transfer level (RTL) design in a hardware description language (HDL), which is in turn commonly synthesized to the gate level by the use of a logic synthesis tool. The goal of HLS is to let hardware designers efficiently build and verify hardware, by giving them better control over optimization of their design architecture, and through the nature of allowing the designer to describe the design at a higher level of abstraction while the tool does the RTL implementation. Verification of the RTL is an important part of the process.

2.3 OpenCV

OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision. OpenCV is widely used everywhere for Image processing and for machine learning. OpenCV is compatible with many languages and frameworks. OpenCV makes image processing very comfortable.

2.4 xfOpenCV

The xfOpenCV library is a set of 50+ kernels, optimized for Xilinx FPGAs and SoCs, based on the OpenCV computer vision library. The kernels in the xfOpenCV library are optimized and supported in the Xilinx SDx Tool Suite. xfOpenCV is a library which is compatible with HLS and FPGA programming. xfOpenCV has almost all the functionality of OpenCV.xfOpenCV is written in C++ and is used for FPGA programming using HLS.

2.5 Canny Edge Detection

2.5.0 Introduction

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a computational theory of edge detection explaining why the technique works. The stages in Canny Edge Detection are:

1.Noise Reduction

2.Finding Intensity Gradient of the image

3.Non-Maximum Suppression

4.Hysteresis Thresholding

2.5.1 Gaussian Blur

In image processing, a Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function . It is a widely used effect in graphics software, typically to reduce image noise and reduce detail. The visual effect of this blurring technique is a smooth blur resembling that of viewing the image through a translucent screen, distinctly different from the bokeh effect produced by an out-of-focus lens or the shadow of an object under usual illumination. Gaussian smoothing is also used as a pre-processing stage in computer vision algorithms in order to enhance image structures at different scales—see scale space representation and scale space implementation.

2.5.2 Sobel Edge Detection

The Sobel operator, sometimes called the Sobel–Feldman operator or Sobel filter, is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasising edges. At each point in the image, the result of the Sobel–Feldman operator is either the corresponding gradient vector or the norm of this vector. 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.

2.5.3 Non-Maximum Suppression

Non-maximum suppression is an edge thinning technique. Non-maximum suppression is applied to find "the largest" edge. After applying gradient calculation, the edge extracted from the gradient value is still quite blurred. With respect to criterion 3, there should only

be one accurate response to the edge. Thus non-maximum suppression can help to suppress all the gradient values (by setting them to 0) except the local maxima, which indicate locations with the sharpest change of intensity value.

2.5.4 Double Thresholding

After application of non-maximum suppression, remaining edge pixels provide a more accurate representation of real edges in an image. However, some edge pixels remain that are caused by noise and color variation. In order to account for these spurious responses, it is essential to filter out edge pixels with a weak gradient value and preserve edge pixels with a high gradient value. This is accomplished by selecting high and low threshold values. If an edge pixel's gradient value is higher than the high threshold value, it is marked as a strong edge pixel. If an edge pixel's gradient value is smaller than the high threshold value and larger than the low threshold value, it is marked as a weak edge pixel. If an edge pixel's value is smaller than the low threshold value, it will be suppressed. The two threshold values are empirically determined and their definition will depend on the content of a given input image.

2.5.5 Hysteresis Edge Tracking

So far, the strong edge pixels should certainly be involved in the final edge image, as they are extracted from the true edges in the image. However, there will be some debate on the weak edge pixels, as these pixels can either be extracted from the true edge, or the noise/color variations. To achieve an accurate result, the weak edges caused by the latter reasons should be removed. Usually a weak edge pixel caused from true edges will be connected to a strong edge pixel while noise responses are unconnected.

2.6 Conclusion

In this chapter we discussed about canny edge detection and High Level Synthesis at introduction level.

Image Processing

3.1 Introduction

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them.

3.2 Image as a Matrix

Grayscale images can be represented by matrices. Each element of the matrix determines the intensity of the corresponding pixel. For convenience, most of the current digital files use integer numbers between 0 (to indicate black, the color of minimal intensity) and 255 (to indicate white, maximum intensity), giving a total of 256 = 28 different levels of gray2. Color images, in turn, can be represented by three matrices. Each matrix specifies the amount of Red, Green and Blue that makes up the image. This color system is known as RGB3. The elements of these matrices are integer numbers between 0 and 255, and they determine the intensity of the pixel with respect to the color of the matrix. Thus, in the RGB system, it is possible to represent 2563 = 224 = 16777216 different colors. 1A pixel is the smallest graphical element of a mat

3.3 Image Convolution

Convolution is a simple mathematical operation which is fundamental to many common image processing operators. Convolution provides a way of `multiplying together' two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values.

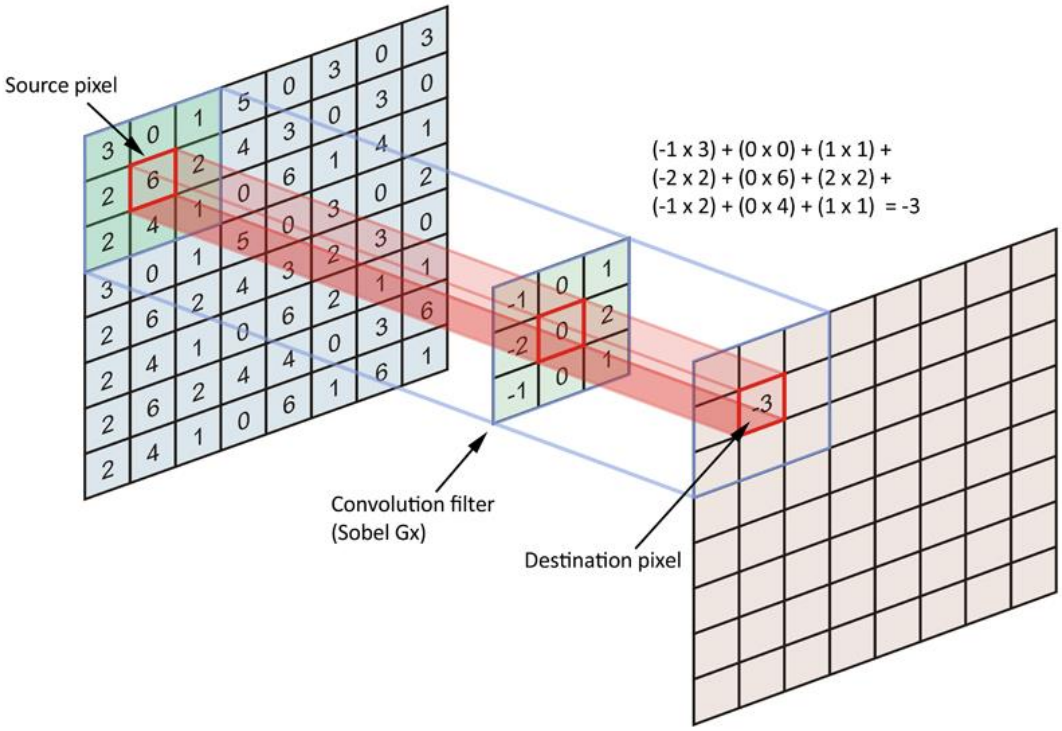

Figure: 3.3 Image Convolution

3.4 Conclusion

This chapter discusses about image processing techniques and basics of the image processing.

High Level Synthesis

4.1 Introduction

High Level Synthesis is automated process of conversion of algorithmic description into hardware design by conversion into RTL.High Level Synthesis makes it easy for complex hardware deciption easy by convertin algoirthmic logic into hardware design.

4.2 Vivado HLS

Vivado HLS is an IDE software developed by Xilinx which is used widely for HLS design for Xilinx FPGA boards.Vivado HLS provides wide range of tools for Synthesis,Debugging and testing purposes.Vivado HLS support many FPGA boards designed by Xilinx and also supports thrid party FPGA Boards.

4.3 Vivado HLS Tools

Vivado HLS provides a wide range of tools.In this section we will cover few of them.Vivado tools provide assistance in debugging , synthesis and testing of the logic for a wide range of hardware boards.Vivado HLS tools also help in simulation of the logic.

4.3.1 HLS C Simulation

In HLS C Simulation the program is compiled using the gcc compiler and gives the C output of the program.HLS C Simulation is done to verify the whether the program has any bugs or the program gives the desired output.

4.3.2 HLS C Synthesis

HLS C Synthesis gives the maximum and minimum Latency(Latency is no. of clock cycles taken from when input given to when the output is received) and maximum and minimum Interval(No. of clock cycles taken for another input to be given).C Synthesis also provied the No. of BRAM ,DSP ,Flip Flops and LUTs used by the program.HLS C Synthesis considers all the pragmas and directives and optimizes the program according to them to decrease the latency and interval also the memory accordingly.

4.3.3 HLS C/RTL CoSimulation

HLS C/RTL CoSimulation converts the given C Source code to RTL and then compares it with the output of the C Simulation.If there are any errors in the C source code or in the pragmas given then it could lead to failing of the cosimulation or hanging of the cosimulation.If the output of the RTL and C are same then the Cosimulation is passed otherwise it is failed.

4.4 Conclusion

In this chapter we discussed about High Level Synthesis and Vivado HLS IDE.We also Discussed about variosu Vivado HLS tools such as HLS C Synthesis,HLS C Simulation,Vivado HLS RTL/C Cosimulation.

Gaussian Blur

5.1 Introduction

Gaussian Blur is Blurring an image using the gaussian function.Gaussian Blur is done by Convolution of a gaussian distribution kernel and an image.Gaussian kernel is a squared matrix which has a gaussian or normal distribution.By convolution of image and gaussian kernel the image is smoothened and the noise is reduced.

5.2 Gaussian Distribution

Gaussian or Normal Distribution is a distribution which is a bell curve which is symmetrical both sides the Axis.

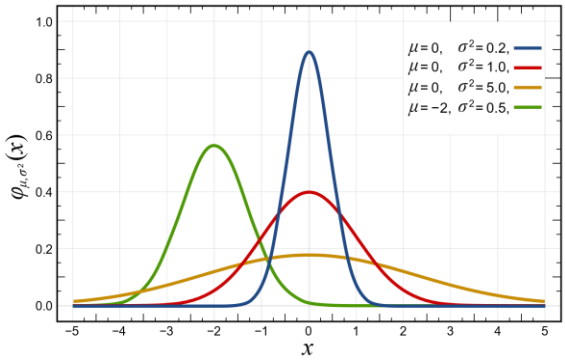

Figure: 5.2 Gaussian Distribution

$$y = \frac{1}{\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma}$$

μ = Mean

σ = Standard Deviation

$\pi \approx 3.14159$

$e \approx 2.71828$

Figure: 5.2 Gaussian Distriution Equation

5.3 Smoothing and Blurring

By using the gaussian Function we can smooth an image by convolution with the gaussian kernel.By increasing the Gaussian kernel size it can the smoothening increases but it costs a lot more computation.

0.075	0.124	0.075
0.124	0.204	0.124
0.075	0.124	0.075

5.4 Implementation

```
void gaussian_blur(AXI_STREAM& INPUT_STREAM,AXI_STREAM& OUTPUT_STREAM , unsigned int r, unsigned int c) {

#pragma HLS INTERFACE axis port=INPUT_STREAM
#pragma HLS INTERFACE axis port=OUTPUT_STREAM
#pragma HLS INTERFACE ap_stable port=r
#pragma HLS INTERFACE ap_stable port=c

     static IMAGE _src(r,c);
     static IMAGE _dst(r,c);

     xf::AXIvideo2xfMat(INPUT_STREAM,_src);

     ap_fixed<16,1> kernel[KERNEL_WIDTH*KERNEL_HEIGHT]={0.075,0.124,0.075,0.124,0.204,0.124,0.075,0.124,0.075};

    int x,y,j,i;
    int sum=0;
    unsigned int b=_src.cols,a=_src.rows;

#pragma HLS pipeline

    l1:for(x=1;x<a-1;x++){
#pragma HLS loop_tripcount min=1 max=1080
        l2:for(y=1;y<b-1;y++){
#pragma HLS loop_tripcount min=1 max=1920
            sum=0;
            l3:for(i=-1;i<=1;i++){
                l4:for(j=-1;j<=1;j++){
                    sum=sum+kernel[(i+1)*(KERNEL_WIDTH)+j+1]*_src.data[b*(x-i)+(y-j)];
                    }
                }
            _dst.data[b*x+y]=sum;
            }
        }


        xf:: xfMat2AXIvideo(_dst,OUTPUT_STREAM);

    }
```

The image is given as input by an AXI stream.The input Stream is then converted into a xf::mat container by using AXIvideo2xfMat function in which the convolution is done.

Here kernel is a fixed point array containing gaussian distributed values as it is easy to compute than floatinng point.Here the convolution is done with kernel and image in mat container _src and save the output to the _dst mat container.

Pipeline pragma is used for pipelining the convolution process.Pipelining makes the latency less by allowing maximum utilization.tripcount pragma is used for giving the minimum and maximum iterations of the loop as the C synthesis tool cannot determine the loop counut if they are variable in nature.

The loops l1 and l2 are for selecting a pixel from a row and a column.The loops l3 and l4 are used for convolution of the pixel with the kernel.

The _dst is then converted into AXI output stream by xfMat2AXIvideo function.

5.5 Results

Figure: Input Image

Figure: Output Image

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.316	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
17	22817168	17	22817168	none

Figure: C Synthesis Report

5.6 Problem Summary

The usage of the floating point was a problem in the program as the cosimulation was taking a lot of time.By changing the floating point variables to fixed point the cosimulation hanging did not occur and the Latency was also decreased.

5.7 Conclusion

In this chapter we discussed about the Gaussian blur methodology and the implementation of the gaussian blur in HLS.

Sobel Edge Detection

6.1 Introduction

Sobel edge detection is a method in which the change in the intensity is found out by finding the change in the intensity of the pixel by applying a sobel gradient kernel to it in the x and y directions.The gradient of the image is found out by using the sobel operator.

6.2 Gradient

The gradient is found by using the gradient kernels.The gradient kernels are covoluted around the image to get the gradient of the image.The gradient of the image tells us how the image intensity is changing in the given direction.

X – Direction Kernel

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

Y – Direction Kernel

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

Figure: Sobel Gradient Kernels

6.3 Edge Detection

We can find the edges of the images using the sobel operator.The sobel operator gives the change in intensity across the x and y directions.The magnitude of the x and y direction gradient images gives us the image of the edges of the image.An edge is nothing but suddden change in intensity in an image.By finding out the gradients we are finding the edges in the given direction.But we need the Edges in all directions rather than in a given direction.For that we find the Magnitude of the both gradient images.

6.4 Implementation

```
void sobel_edge(AXI_STREAM& INPUT_STREAM , AXI_STREAM& OUTPUT_STREAM_X , AXI_STREAM& OUTPUT_STREAM_Y ,AXI_STREAM& OUTPUT_STREAM, unsigned int r ,unsigned  int c) {


#pragma HLS INTERFACE axis port=INPUT_STREAM
#pragma HLS INTERFACE axis port=OUTPUT_STREAM_X
#pragma HLS INTERFACE axis port=OUTPUT_STREAM_Y
#pragma HLS INTERFACE axis port=OUTPUT_STREAM

#pragma HLS INTERFACE ap_stable port=r
#pragma HLS INTERFACE ap_stable port=c


     static IMAGE _src(r,c);
     static IMAGE _dstx(r,c);
     static IMAGE _dsty(r,c);
     static IMAGE _dst(r,c);


    xf::AXIvideo2xfMat(INPUT_STREAM,_src);

    int i,j,x,y;
    int sum_x,sum_y;
    int gdx[FILTER_WIDTH*FILTER_HEIGHT]={-1,0,1,-2,0,2,-1,0,1};
    int gdy[FILTER_WIDTH*FILTER_HEIGHT]={-1,-2,-1,0,0,0,1,2,1};

    l1:for(x=1;x<r-1;x++){
#pragma HLS pipeline
        #pragma HLS loop_tripcount min=1 max=1080
        l2:for(y=1;y<c-1;y++){
            #pragma HLS loop_tripcount min=1 max=1920
            sum_x=0;
            sum_y=0;
            conv1:for(i=-1;i<=1;i++){
                    conv2:for(j=-1;j<=1;j++){
                            sum_x=sum_x+gdx[(i+1)*(FILTER_WIDTH)+j+1]*_src.data[c*(x-i)+(y-j)];
                            sum_y=sum_y+gdy[(i+1)*(FILTER_WIDTH)+j+1]*_src.data[c*(x-i)+(y-j)];
                            }
                        }
            _dsty.data[c*x + y]=sum_y;
            _dstx.data[c*x + y]=sum_x;
            _dst.data[c*x + y]=sqrt(float(sum_x*sum_x+sum_y*sum_y));
         }

    }
#pragma HLS dataflow
    xf:: xfMat2AXIvideo(_dstx,OUTPUT_STREAM_X);
    xf:: xfMat2AXIvideo(_dsty,OUTPUT_STREAM_Y);
    xf:: xfMat2AXIvideo(_dst,OUTPUT_STREAM);

}
```

Here the AXI stream is first converted into a xf::mat container _src using AXIvideo2xMat.Then the gradient kernels are declared which are gdx,gdy.

Then by using the l1 and l2 loops we go through each pixel in the video.We convolute each pixel with the gdx and gdy kernels and adding the sum to sum_ and sum_y.These are then given to the _dsty and _dstx.the magnitude of the sum_x and sum_y is taken and value is given to the _dst.

Pipeline pragma is used to pipeline the convolution.Tripcounts pragmas are used to make the C synthesis know the maximum and minmum iterations in the loop as that is variable.

Dataflow pragma is used for the parallel operation of the conversion of the mat to AXI stream using xfMat2AXIvideo.

6.4 Result

Figure: Input Image

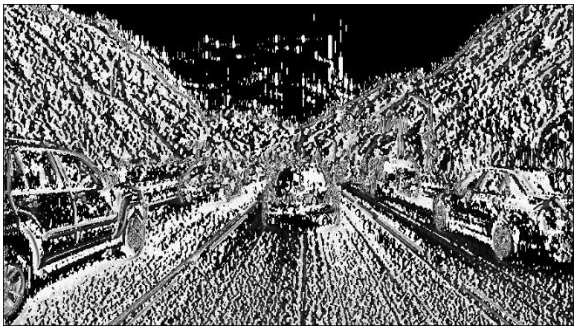

Figure: Sobel gradient X

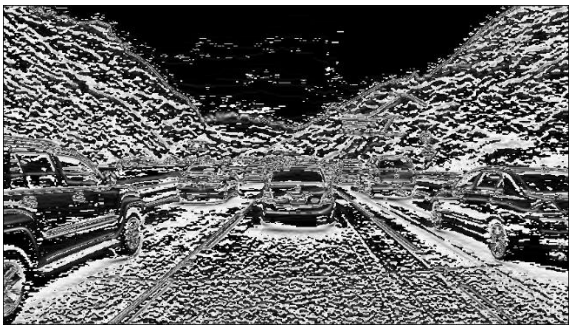

Figure:Sobel gradient Y

Figure:Sobel Edge

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	7.894	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
55	14524963	41	10368036	dataflow

Figure: C Synthesis report

6.5 Problem Summary

The latency was very high due to conversion of three matrices to AXI streams and also the convolution of the two gradient kernels over the image.The latency here was cut down by the usage of the pipeline pragma for convolution and dataflow pragma for mat to AXI stream conversion.

6.6 Conclusion

In this chapter we discussed about the Sobel Edge Detection.We also discussed the implementation of the sobel operator using the High Level Synthesis.

Non-Maximum Suppression

7.1 Introduction

Non-Maximum Suppression is an edge thinning method.The edges came from the edge detection could be very thick which cannot be used.The non maximum suppression finds the egde in a direction and thins it.

7.2 Methodology

The non maximum suppression finds the direction in which the gradient is moving and then the local maxima is found out in that direction.The local maxima is set to the highest intensity and the other pixels are set to lowest.This makes the image clean and the edges are thin which is easy to use for any applications.

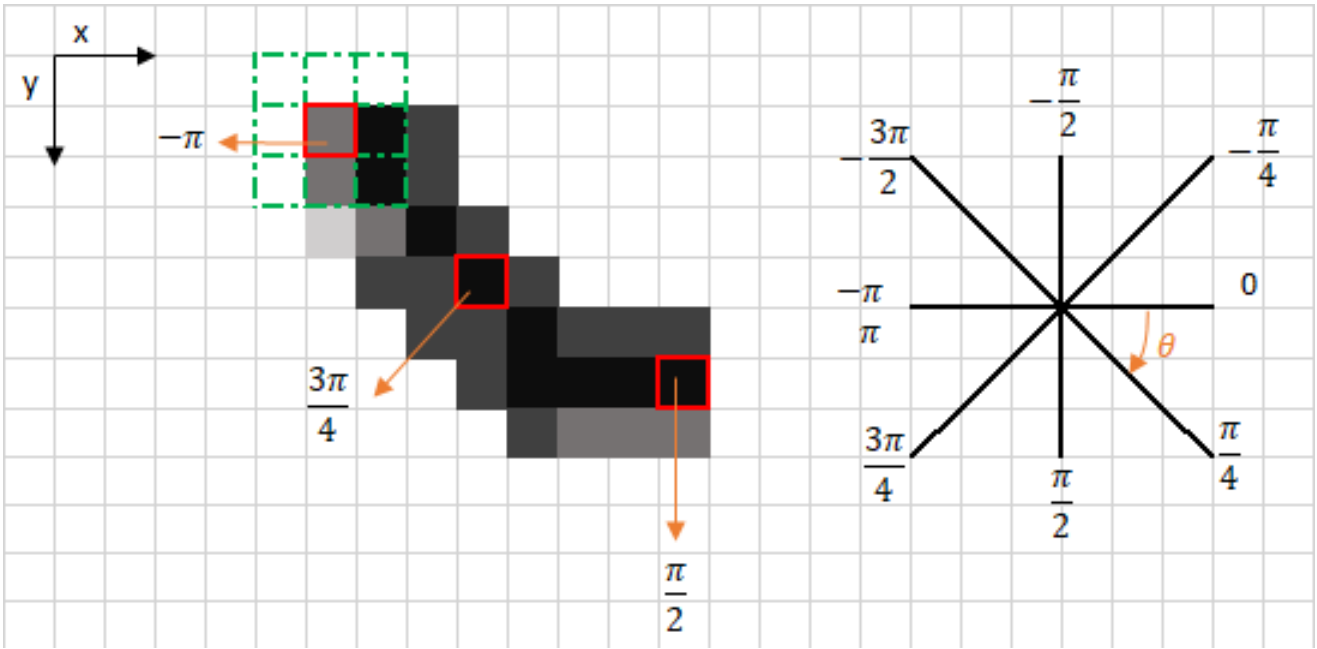

Figure: Non-Maximum Suppression

7.3 Implementation

```
void non_maximum_suppression(AXI_STREAM& INPUT_STREAM_X,AXI_STREAM& INPUT_STREAM_Y,AXI_STREAM& INPUT_SOBEL_STREAM,AXI_STREAM& OUTPUT_STREAM,unsigned int r,unsigned int c) {
#pragma HLS INTERFACE axis port=INPUT_STREAM_X
#pragma HLS INTERFACE axis port=INPUT_STREAM_Y
#pragma HLS INTERFACE axis port=INPUT_SOBEL_STREAM
#pragma HLS INTERFACE axis port=OUTPUT_STREAM

#pragma HLS INTERFACE ap_stable port=r
#pragma HLS INTERFACE ap_stable port=c

static IMAGE _gx(r,c);
static IMAGE _gy(r,c);
static IMAGE _sobel(r,c);
static IMAGE _nms(r,c);

#pragma HLS dataflow
xf::AXIvideo2xfMat(INPUT_STREAM_X,_gx);
xf::AXIvideo2xfMat(INPUT_STREAM_Y,_gy);
xf::AXIvideo2xfMat(INPUT_SOBEL_STREAM,_sobel);


    int x,y;
    int a=_sobel.rows,b=_sobel.cols,q,p;
    ap_fixed<16,8> angle=0,PI=3.14159265359;


    l1:for(x=1;x<a-1;x++){
        #pragma HLS loop_tripcount min=1 max=1080
#pragma HLS pipeline
        l2:for(y=1;y<b-1;y++){
            #pragma HLS loop_tripcount min=1 max=1920
            angle = (_gx.data[b*x+y] == 0) ? ap_fixed<16,8>(PI/2) : ap_fixed<16,8>(atanf(_gy.data[b*x+y]/_gx.data[b*x+y]));

            //angle= (angle<0) ? (PI+angle) : angle;

            q=255;

            p=255;

            if((angle>=0 && angle<=PI/8) || (7*PI/8<=angle && angle<=PI)) {

                q=_sobel.data[b*x + (y+1)];

                p=_sobel.data[b*x + (y-1)];

            }


            else if((angle>=PI/8) && (angle<=3*PI/8)) {

                q=_sobel.data[b*(x+1) + (y-1)];

                p=_sobel.data[b*(x-1) + (y+1)];

            }

            else if((angle>=3*PI/8) && (angle<=5*PI/8)) {

                q=_sobel.data[b*(x+1) + y];

                p=_sobel.data[b*(x-1) + y];

            }

            else if((angle>=5*PI/8) && (angle<=7*PI/8)) {

                q=_sobel.data[b*(x-1) + (y-1)];

                p=_sobel.data[b*(x+1) + (y+1)];

            }


            if((_sobel.data[b*x+y]>=q) && (_sobel.data[b*x+y]>=p)) {

                _nms.data[b*x+y]=_sobel.data[b*x+y];

            }

            else {

                _nms.data[b*x+y]=0;


            }
```

The AXI stream is converted into xf::mat container _src using AXIvideo2xfMat.Each pixel is accessed by the loops l1 and l2.we find the angle by the finding the tan inverse of the gy/gx and the angle is then classified into 90,180,0,45,135 degrees.By the angle classified we find local maxima in the direction such as for 180 and 0 degrees we find the local maxima in (x-1,y),(x,y),(x,y+1).

Then the Mat is converted to AXI stream by using xfMat2AXIvideo.

7.4 Result

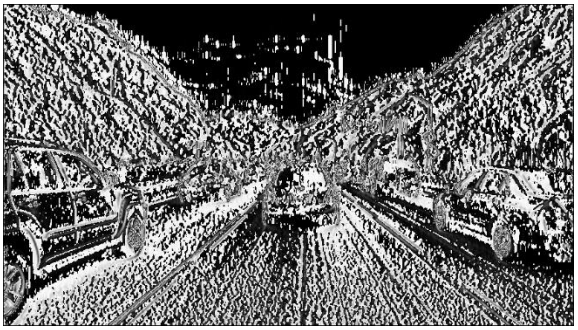

Figure: Sobel gradient X

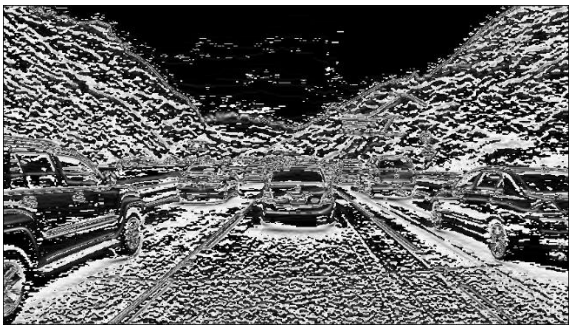

Figure:Sobel gradient Y

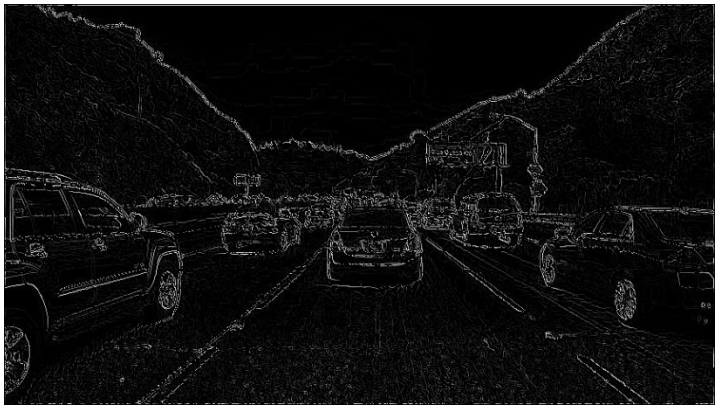

Figure:Output image

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.733	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
46	6225164	44	4147242	dataflow

Figure: C Synthesis Report

7.5 Problem Summary

The floating point usage here for the tan inverse caused RTL/C Cosimulation hanging.The problem was resolved by using the fixed point arbitrary data types.This cut down the latency too.The 0 value of the x gradient was not considered leading to arithmetic error which was rectified and given angle value pi/2 if xgradient value was 0.

7.6 Conclusion

We discussed about the non-maximum suppression and how we thinned the edges using it.We discussed the implemntation of non maximum suppression using HLS.

Double Threshold

8.1 Introduction

Double thresholding is classification of the pixels into three classes.Based on the classes the pixels can be ignored or used.Thresholding is used for segmentation of the image into segments.

8.2 Classification

We classify the pixels into strong , weak and rest.The strong pixels are very strong that they cannot be ignored.The weak pixels are very weak and are not considered.The rest pixels are not very weak and and are not very strong.These pixels are left for Hysteresis edge trtacking.The strong pixels are given the highest intensity , the weak pixels are given lowest intensity and the rest pixels are given a constant value which is to be used in hysteresis edge tracking.

8.3 Implementaion

```
void double_threshold(AXI_STREAM& INPUT_STREAM,AXI_STREAM& OUTPUT_STREAM,unsigned int r,unsigned int c)  {

#pragma HLS INTERFACE axis port=INPUT_STREAM
#pragma HLS INTERFACE axis port=OUTPUT_STREAM

#pragma HLS INTERFACE ap_stable port=r
#pragma HLS INTERFACE ap_stable port=c


    static IMAGE _nms(r,c);
    static IMAGE _dth(r,c);

    xf::AXIvideo2xfMat(INPUT_STREAM,_nms);



    int max_value=255;

    int high_thresh=max_value*0.3;

    int low_thresh=max_value*0.1;

    int a=_nms.rows,b=_nms.cols;

    int x,y;

    l1:for(x=1;x<a-1;x++) {
#pragma HLS loop_tripcount min=1 max=1080
#pragma HLS pipeline
        l2:for(y=1;y<b-1;y++) {
#pragma HLS loop_tripcount min=1 max=1920
            if(_nms.data[b*x + y]>=high_thresh) {

                _dth.data[b*x + y]=STRONG;

            }

            if((_nms.data[b*x + y] <= high_thresh) && (_nms.data[b*x + y] >= low_thresh)) {

                _dth.data[b*x + y]=WEAK;

            }
}
        }


    }
    xf:: xfMat2AXIvideo(_dth,OUTPUT_STREAM);
}
```

The input is given as AXI stream and it is converted into xf::mat using AXIvideo2xfMat()

We find the high threshold which we take as 30% value of the maximum value and the lower threshold to 10% of the maximum value.Based on these values we check each pixel and classify them into the classes.

The Mat container is converted into AXI stream using xfMat2AXIvideo().

8.4 Result

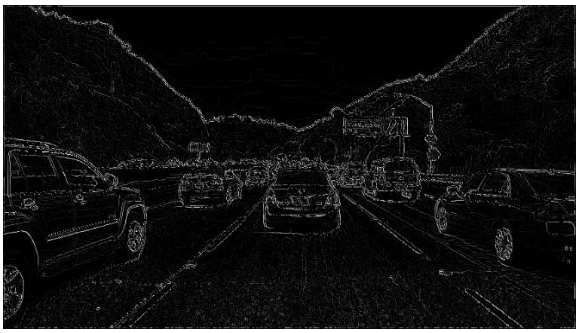

Figure: Input Image

Figure:Output Image

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	6.333	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
12	8303050	12	8303050	none

Figure: C Synthesis Report

8.5 Problem Summary

The value of the high threshold and low threshold were difficult to determine.After a brief reading of Documentation of the Double threshold in other popular libraries , the popular choice of high_thresh was taken as 30% of max value and low_thresh was taken as 10% of the max value.

8.5 Conclusion

We discussed classification of the pixels into classes.We discussed the double thresholding using HLS.

Hysteresis Edge Tracking

9.1 Introduction

The Hysteresis edge tracking checks whether there are any strong pixels near by any weak pixels.If there are any strong pixels then it makes that weak pixel into strong pixel otherwise we make that to zero.

9.2 Implementation

```
void hysteresis(AXI_STREAM& INPUT_STREAM,AXI_STREAM& OUTPUT_STREAM,unsigned int r,unsigned int c) {


#pragma HLS INTERFACE axis port=INPUT_STREAM
#pragma HLS INTERFACE axis port=OUTPUT_STREAM

#pragma HLS INTERFACE ap_stable port=r
#pragma HLS INTERFACE ap_stable port=c

    static IMAGE _dth(r,c);
    static IMAGE _hss(r,c);

    xf::AXIvideo2xfMat(INPUT_STREAM,_dth);

    int a=_dth.rows,b=_dth.cols,x,y;


    l1:for(x=1;x<a-1;x++) {
#pragma HLS loop_tripcount min=1 max=1080
#pragma HLS pipeline
        l2:for(y=1;y<b-1;y++) {
#pragma HLS loop_tripcount min=1 max=1920
            if(_dth.data[b*(x)+y]==WEAK) {
                if ((_dth.data[b*(x+1) + (y-1)] == STRONG) ||
                 (_dth.data[b*(x+1) + y] == STRONG) ||
                  (_dth.data[b*(x+1) + (y+1)] == STRONG) ||
                   (_dth.data[b*(x) + (y-1)] == STRONG) ||
                   (_dth.data[b*(x) + (y+1)] == STRONG) ||
                   (_dth.data[b*(x-1) + (y-1)] == STRONG) ||
                   (_dth.data[b*(x-1) + y] == STRONG) ||
                   (_dth.data[b*(x-1) + (y+1)] == STRONG)) {

                    _hss.data[b*x + y]=STRONG;
                }

                else {

                    _hss.data[b*x + y]=0;
                }
            }

            else {
                _hss.data[b*x+y]=_dth.data[b*x+y];
            }
        }

    }

    xf:: xfMat2AXIvideo(_hss,OUTPUT_STREAM);
}
```

The Input AXI stream is converted into xf::Mat using AXIvideo2Mat into _dts.

In hysteresis edge tracking we go thorugh all the rest pixels in the double thresholding process.We take those pixels and check whether there are any adjascent pixels near them.If there are change the pixel to strong or set it to zero.

The _hss mat is converted into Output AXI stream using AXIvideo2xfMat.

9.3 Result

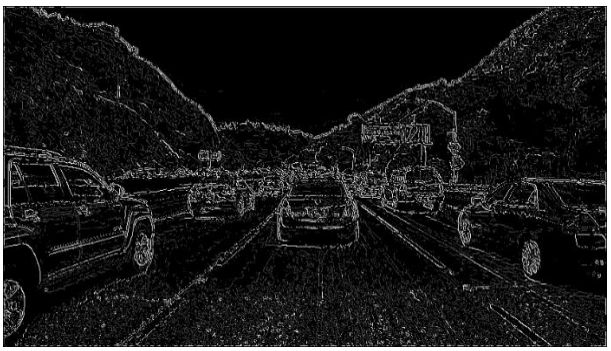

Figure: Input Image

Figure:Output Image

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	7.894	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
55	14524963	41	10368036	dataflow

Figure: C Synthesis Report

9.4 Problem Summary

Instead of considering only the weak pixels , I considered all the pixels which lead to thickening of edges as all the pixels were given strong value. The problem was rectified and only weak pixels were considered.

9.5 Conclusion

In this chapter we discussed about Hysteresis Edge Tracking.We discussed about implementing it using HLS.

Canny Edge Detection

10.1 Introduction

Canny Edge Detection is a multistage algorithm. Canny also produced a computational theory of edge detection explaining why the technique works. The stages in Canny Edge Detection are:

1.Noise Reduction

2.Finding Intensity Gradient of the image

3.Non-Maximum Suppression

4.Double Thresholding

5.Hysteresis Edge Tracking

10.2 Implementation

```
void canny_edge(AXI_STREAM& INPUT_STREAM,AXI_STREAM& OUTPUT_STREAM,unsigned int r,unsigned int c) {

    #pragma HLS INTERFACE axis port=INPUT_STREAM
    #pragma HLS INTERFACE axis port=OUTPUT_STREAM

    #pragma HLS INTERFACE ap_stable port=r
    #pragma HLS INTERFACE ap_stable port=c

    AXI_STREAM GAUSSIAN_STREAM,SOBELX_STREAM,SOBELY_STREAM,SOBEL_STREAM,NMS_STREAM,DT_STREAM,H_STREAM;

#pragma HLS STREAM variable=GAUSSIAN_STREAM depth=1080*1920
#pragma HLS STREAM variable=SOBELX_STREAM depth=1080*1920
#pragma HLS STREAM variable=SOBELY_STREAM depth=1080*1920
#pragma HLS STREAM variable=SOBEL_STREAM depth=1080*1920
#pragma HLS STREAM variable=NMS_STREAM depth=1080*1920
#pragma HLS STREAM variable=DT_STREAM depth=1080*1920
#pragma HLS STREAM variable=H_STREAM depth=1080*1920
#pragma HLS dataflow

    gaussian_blur(INPUT_STREAM,GAUSSIAN_STREAM,r,c);

    sobel_edge(GAUSSIAN_STREAM,SOBELX_STREAM,SOBELY_STREAM,SOBEL_STREAM,r,c);

    non_maximum_suppression(SOBELX_STREAM,SOBELY_STREAM,SOBEL_STREAM,NMS_STREAM,r,c);

    double_threshold(NMS_STREAM,DT_STREAM,r,c);

    hysteresis(DT_STREAM,OUTPUT_STREAM,r,c);


}
```

We first use gaussian_blur function on the input AXI stream to smoothen and reduce the noise.Then we call the sobel_edge function to find the edges.

Then we find the non_maximum_suppression function to thin the edges.Then we should use double_threshold to classify the pixels into 3 classes.By using the hysteresis function we find the pixels which are near to strong and are made strong or left alone.

10.3 Result

Figure: Input Image

Figure:Output Image

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.733	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
61	20743567	44	20743568	dataflow

Figure: C Synthesis Report

10.4 Problem Summary

The latency was a bit high as the functions called are very high computing functions.The problem was resolved by using dataflow pragma which speed up the process.

10.5 Conclusion

We discussed about the Canny Edge Detction.We discussed the implementation of the Canny Edge Detection using HLS.

Conclusion

In this project we gained knowledge about Image Processing techniques such as Gaussian Blur,Sobel Edge Detection,Non maximum Suppression ,Thresholding and Hysteresis Edge Tracking.These techniques are used for Canny Edge Tracking.

We also learned about High Level Synthesis.In High level synthesis we learned about usage of pragmas , directives and programming techniques for FPGA board logic.

We learned about the Vivado HLS tools and usage of them.Tools such as C Simulation , C Synthesis and C/RTL Cosimulation.

References

1. https://en.wikipedia.org/wiki/Canny_edge_detector
2. https://en.wikipedia.org/wiki/OpenCV
3. https://en.wikipedia.org/wiki/High-level_synthesis
4. https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf
5. https://www.xilinx.com/content/dam/xilinx/support/documentation/sw_manuals/xilinx2018_3/ug871-vivado-high-level-synthesis-tutorial.pdf
6. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug902-vivado-high-level-synthesis.pdf#nameddest=xApplyingOptimizationDirectives