Ministry of Higher Education and Scientific Research
La Manouba University
National School of Computer Science

Integration Project Report

Subject

HARDWARE IMPLEMENTATION OF SOBEL EDGE DETECTION ALGORITHM

Directed By
Mortadha Touzi
Mohamed Ilyess Benzarti

Supervised By : **Mohamed Masmoudi & Lobna Kriaa**

E-mail : **mohamed.masmoudi@ensi-uma.tn**

lobna.kriaa@ensi-uma.tn

Academic Year : 2017/2018

Abstract

This report covers the design and development of an image processing algorithm wich will be executed on an FPGA. In particular we focus on the edge detection algorithms, precisely Sobel filter, and we will be coding our project using VHDL as a programming language,as it is compatible with FPGA's.

All these features will be simulated using Model Sim, and integrated in a project provided by the vivado softare, that will take as an input a HDMi streaming and outputs a VGA streaming of the dedicated treatment .

Keywords: FPGA,image processing, Sobel filter,VHDL,Vivado, Model Sim.

Acknowledgement

In conducting this report, we have received meaningful assistance from our professor, which we would like to put on record here with deep gratitude and great pleasure.

First and foremost, we express our sincere gratitude to our supervisor,
Mr Mohamed Masmoudi, who extended his complete support and helped to make me deliver our best, for his guidance and valuable pieces of advice during different phases of the elaboration of the project, he even provided us with an FPGA card to make the tests. He also welcome us with a warm heart in his company, provided us a place to work and materials. Simply put, we could not have done this work without the amount of help and trust we have received from him. we appreciate their patience in checking and reviewing our work.

we would also like to thank all of our professors at the National School of Computer Science for their continuous help and training during our study years.

Finally, special thanks to the jury members who honored us by examining and evaluating this modest contribution.

Contents

List of Figures

List of Tables

General Introduction

In recent decades, image processing became one of the most wanted and strong fields in reasearch, because the world is in need of applications that needs computer vision as a necessary part to reach a high level of perception. for exmaple robotics and autonomus vehicles are becoming the main focus in this era.

In this context,our solution is an implementation of the computer vision algorithm, Sobel filter, with an aim of a real time performance.

Our work is presented throughout this report accordingly to the following plan. In the first chapter,we present the project overview. Throughout the second chapter,we focus on the description of our solution and the steps that we took for the implementation of the algorithm.In the third chapter we shed the light on the architecture that we have designed for our solution. In the fourth chapter, we show some of the Simulation results,in addition to some screen shots of the algorithm results. In the fifth chapter, we mention some of the problems and challenges that we have faced during all the steps of this project.

Finally, we conclude our work by giving future prospects and potentials upgrades for our project.

Chapter 1

Project Overview

1.1 Motivations for Edge Detection

Edges are the points at which image brightness changes sharply are typically organized into a set of curved line segments. The purpose of detecting sharp changes in image brightness is to capture important events and changes in properties of the world. Discontinuities in image brightness corresponds to :

- Discontinuities in depth.
- Discontinuities in surface orientation.
- Changes in material properties.
- Variations in scene illumination.

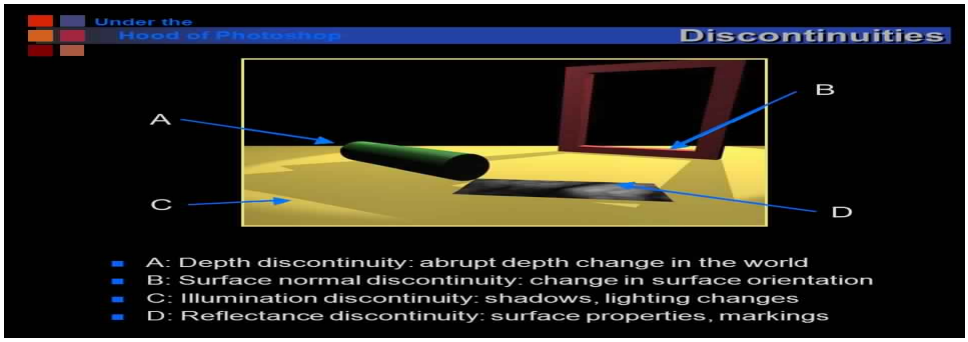

Figure 1.1: Discontinuities in images

Applying an edge detection algorithm to an image can significantly reduce the amount of data to be processed and may therefore filter out information that may be regarded as less relevant, while preserving the important structural properties of an image.

To conclude, Edge detection is a fundamental tool in image processing and computer vision, particularly in the areas of feature detection and feature extraction.

1.2 Existing Edge Detectors

there are multiple Edge detectors that are famous with a very strong performance, you can find in this link a detailed Study and Comparison of Various Image Edge Detection Techniques.

Operator	Advantages	Disadvantages
Classical (Sobel, prewitt, Kirsch,...)	Simplicity, Detection of edges and their orientations	Sensitivity to noise, Inaccurate
Zero Crossing(Laplacian, Second directional derivative)	Detection of edges and their orientations. Having fixed characteristics in all directions	Responding to some of the existing edges, Sensitivity to noise
Laplacian of Gaussian(LoG) (Marr-Hildreth)	Finding the correct places of edges, Testing wider area around the pixel	Malfunctioning at the corners, curves and where the gray level intensity function varies. Not finding the orientation of edge because of using the Laplacian filter
Gaussian(Canny, Shen-Castan)	Using probability for finding error rate, Localization and response. Improving signal to noise ratio, Better detection specially in noise conditions	Complex Computations, False zero crossing, Time consuming

Figure 1.2: Advantages and Disadvantages of Existing Edge Detectors

For the sake of simplicity we will be implementing the Sobel Edge detector with the programming language VHDL that is compatible with our FPGA.

1.3 Sobel Edge Detector

The Sobel Edge detector computes an approximation of the gradient of the image intensity function. At each pixel in the image the result of the Sobel operator is the corresponding norm of its gradient vector. The Sobel operator only considers the two orientations which are 0 and 90 degrees convolution 3*3 kernels which are convolved with the original image to calculate approximations of the derivatives.

- Gx is the 3*3 kernel for horizontal changes.
- Gy is the 3*3 kernel for vertical changes.

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

Figure 1.3: Sobel 3*3 Kernels

The output of the image convoluted with these kernels for each pixel can be combined together to find the absolute magnitude of the gradient at each point. The gradient magnitude is given by:

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

To get a faster computation we will be using this approximation:

$$|G| = |Gx| + |Gy|$$

The figure below demonstrates precisely the data flow for the Sobel edge detector computation.

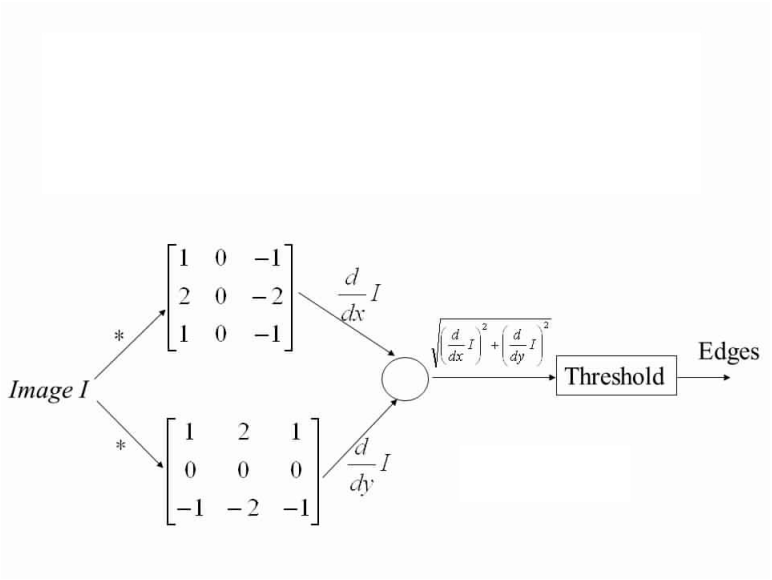

Figure 1.4: Sobel Edge Detector Flow

In this chapter, we talked about edges, why we should use edge detection in computer vision, the edge detection process and finally we presented the Sobel edge detector.

Chapter 2

Requirements specification and Solution Description

2.1 Requirements specification

Sobel edge detector is very used in most of the programming languages, it is available in OpenCv (Python and C++), and Matlab etc. But these implementation lacks an important criteria, which is the Real Time processing. Matlab and OpenCv facilitates the usage and the manipulation of th sobel Edge detector.However, we need a hardware implementation, with it we can approach to Real time processing. Implementing the Sobel edge detector within the FPGA fits the needs.

2.2 Solution Description

The main aim for our solution is a Real time Sobel edge detector as we discussed in the previous section.

We have to choose and deign a perfect combination of hardware components to have the reach a high performance.

2.2.1 Work Process

The images that we will process should be in **pgm** format. We will be using this VHDL library to read image pixels.

The Steps of our Process are the following:

1. Read an image with a pgm format.
2. Design and implement a hardware buffering system to read one pixels and its eight neighboring pixels at the same time.
3. Save the nine extracted pixels in a matrix that will be called later as a mask matrix.
4. apply the Sobel Operator for each mask matrix.
5. Save the computed picture of the sobel operator in an image with a pgm format.

Chapter 3

Architecture Design

3.1 Global Architecture design

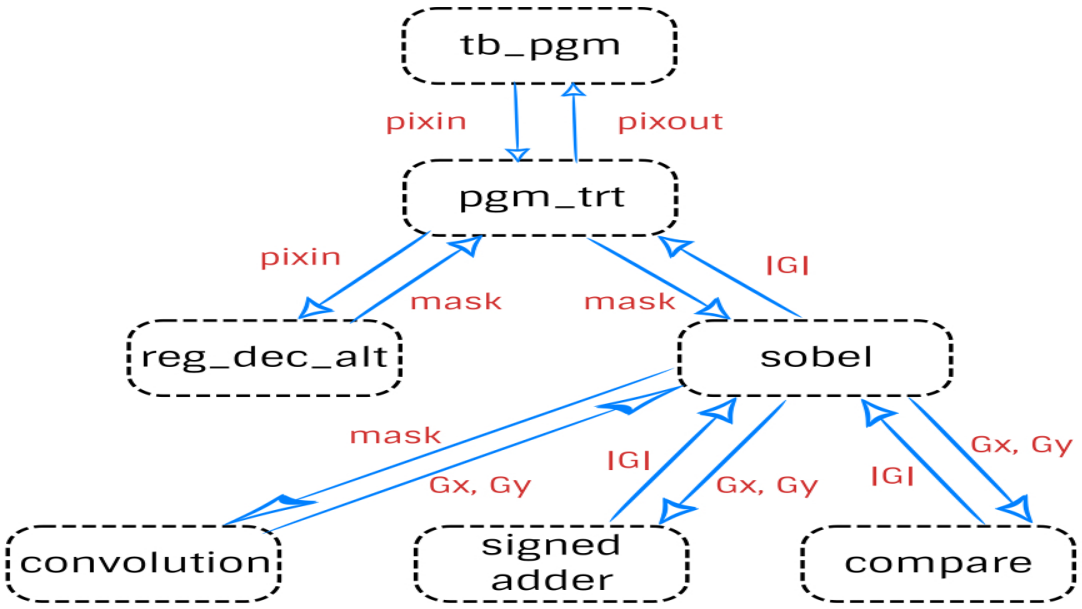

Figure 3.1: Global Architecture

The figure above describes the global architecture and process of our implemented Sobel Edge Detector. First of all the test bench (tb pgm) will read the image pixel by pixel , and each

raising clock signal the pgm trt entity will receive a pixel and send to reg dec alt entity. The reg dec alt entity will return a matrix by the name mask, which will be received and sent by the same entity pgm trt to the sobel entity. The sobel entity will use the entities convolution, signed adder and compare to compute the new values of the image pixels. Finally the sobel filter will return the new values to pgm trt, and then for the test bench to be saved in the output image.

3.2 Detailed Architecture Design & Entities description

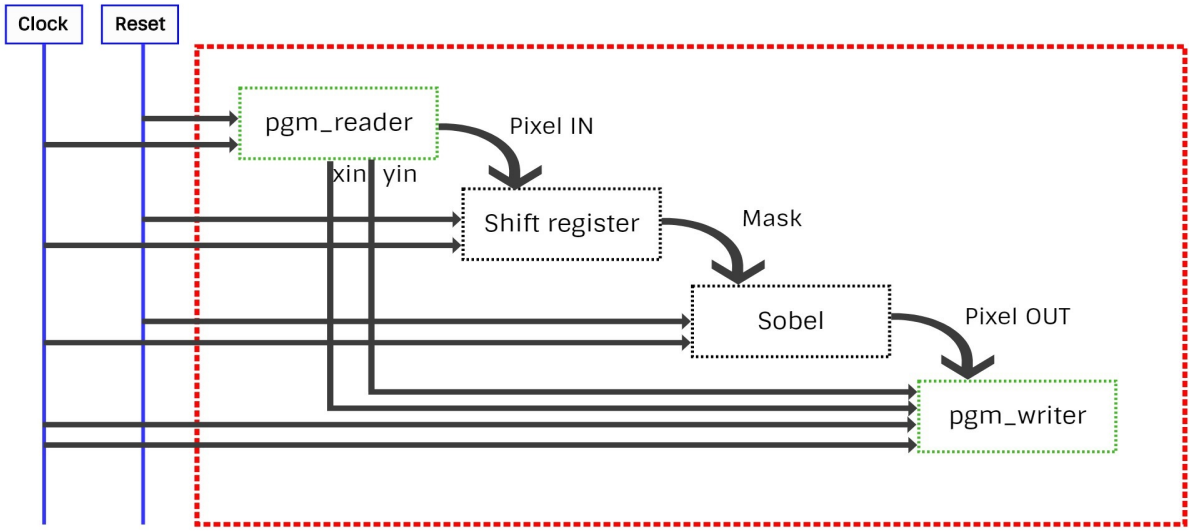

Figure 3.2: Detailed Architecture Design

The figure above shows the most important entities in the project .pgm_reader and pgm_writer are designed for the purpose of the simulation, they will be removed during the synthesis phase.

3.2.1 Test Bench

Our Test Bench will use the pgm Reader Function which is available in the pgm VHDL library mentioned earlier. This function reads the height and width of the image, then creates a matrix of size (height,width) where it outputs a pixel and place it in the right position.

Our Test bench will reads a pixel form that matrix, for each rising clock signal, performs the needed computations, then save the result of the computed pixel. Finally when the image i fully processed, it will be saved in an image with a pgm format.

The VHDL source code below explains the process of our test bench.

```
test: process
        variable data : pixel_array_ptr;
        variable ret: pixel_array_ptr;
  variable x,y : integer range 0 to 2047;

    begin  -- process
          -- test on a proper image
        data := pgm_read(input_path);
        ret    := new pixel_array(0 to data.all'length(1) - 1, 0 to data.all'length(2) -
                -- loop on pixels
                frminval<='0';
                pixinval<='1';
                for y in 0 to data.all'length(2) - 1 loop
                  for x in 0 to data.all'length(1) - 1 loop
                                xin<=x;
                                yin<=y;
                                pixin<=data(x,y)*16;
                                if(pixoutval='1')then
                                  ret(xout,yout):=pixout;
                                  end if;
                                pixinclk<='1';
                                wait for 1 ns;
                                pixinclk <='0';
                                wait for 1 ns;
                        end loop;
                end loop;
                frminval<='1';
                pixinval<='0';
                while frmoutval='0' loop
                        if(xin=1279) then
```

```
                        xin<=0;
                        if(yin=719) then
                                yin<=0;
                        else
                                yin<=yin+1;
                        end if;
                else
                        xin<=xin+1;
                end if;
                pixin<=data(xin,yin);
                if(pixoutval='1') then
                          ret(xout,yout):=pixout;
                end if;
                pixinclk<='1';
                wait for 1 ns;
                pixinclk <='0';
                wait for 1 ns;
        end loop;
        -- Write output
    pgm_write(output_path, ret.all);
    report "End of tests" severity note;
    wait;
end process test;
```

3.2.2 Hardware Buffering Design

3.2.2.1 Flip Flop D for one Bit

Figure 3.3: Flip Flop D

A pixel, in our project is of length 12 bits, we are able to have 4096 values for each pixel. The main purpose of designing a Flip Flop is to be able to hold on memory one bit. this will be used later to create a N bit Flip Flop mainly to save the 12 bits of the Pixel.

```
Library ieee;
use ieee.std_logic_1164.all ;
entity BASCULE_D is
  port(D, CLK, rst: in std_logic;
       Q: OUT std_logic );
end BASCULE_D;
architecture ACH_BASCULE_D of BASCULE_D is
  begin
   process (CLK)
     begin
       if (CLK='1') then
                if(rst='1') then
                        Q<='0';
                else
                        Q <= D ;
                end if;
       end if ;
     end process;
end ACH_BASCULE_D;
```

3.2.2.2 Flip Flop D of N Bit (Pixel)

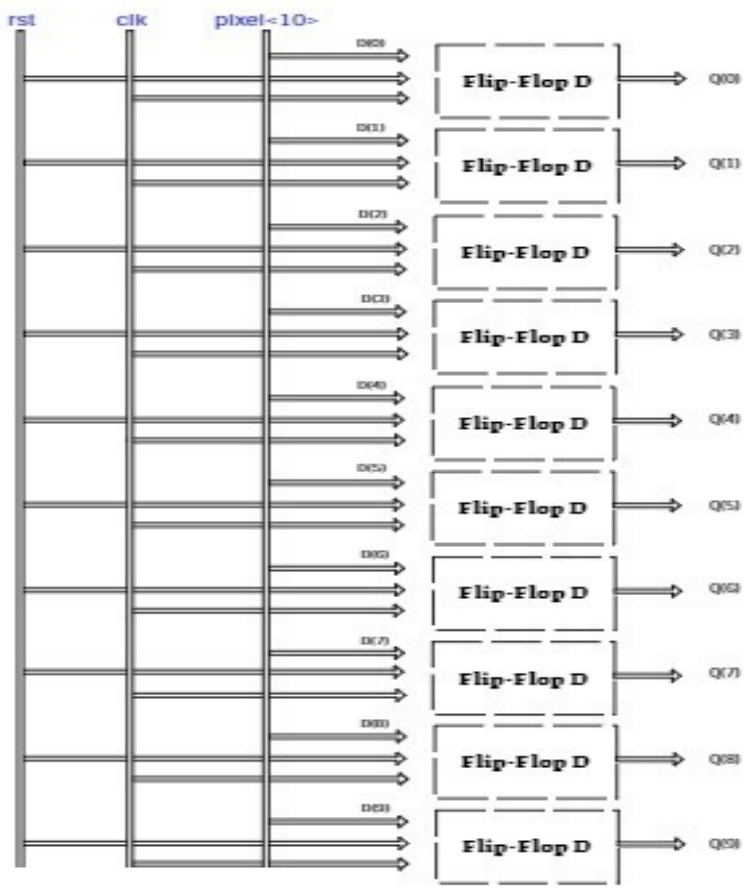

Figure 3.4: N Flip Flop D

As discussed in the previous section a Flip Flop D of N bit is used to save a pixel. The N flip flop D are synchronized by the same clock signal.

```
Library ieee;
use ieee.std_logic_1164.all ;
entity BAS_NBIT is
        generic(N: integer := 12);
        port(
                        D : IN std_logic_vector(N-1 downto 0);
                        CLK, rst: in std_logic;
                        Q: OUT std_logic_vector(N-1 downto 0)
```

```
                );
end BAS_NBIT;
architecture ACH_BAS_NBIT of BAS_NBIT is
        component BASCULE_D is
                port(D, CLK, rst: in std_logic;
                Q: OUT std_logic );
        end Component;
        begin
        BAS_N: FOR i in 0 to N-1 generate
                BAS_i: bascule_d port map(d(i),CLK,RST,q(i));
        end generate;
end ACH_BAS_NBIT;
```

3.2.2.3 Shift Register

Figure 3.5: Shift Register

The Shift register is composed of W pixels, W is defined as the width of the input image. for each clock signal a new pixel enters, we shift all the pixels in the register by one and the last pixel in register goes out.

```
Library ieee;
use ieee.std_logic_1164.all ;
entity REG_DEC_PIXEL is
        generic(N: integer := 10;W: integer:=320);
        port(
                        grayscale : IN std_logic_vector(N-1 downto 0);
                        CLK, rst: in std_logic;
                        grayscale_out: OUT std_logic_vector(N-1 downto 0)
                );
end REG_DEC_PIXEL;
```

```
architecture ACH_REG_DEC_PIXEL of REG_DEC_PIXEL is
        Component BAS_PIXEL is
                generic (N: integer :=10);
                port(
                        grayscale : IN std_logic_vector(N-1 downto 0);
                        CLK, rst: in std_logic;
                        grayscale_out: OUT std_logic_vector(N-1 downto 0)
                );
        end Component;
        signal tmp : std_logic_vector (((w+1)*N) -1 downto 0);
begin
        tmp(N - 1 downto 0) <=grayscale;

        BAS_P: for i in 0 to w-1 generate
                BAS_Pi: BAS_PIXEL generic map (N) port map (tmp((i+1)*N-1 downto i*N),CL
        end generate;

        grayscale_out<=tmp(((w+1)*N) - 1 downto (w*N));

end ACH_REG_DEC_PIXEL;
```

3.2.2.4 Concatenation Of 3 Shift Registers

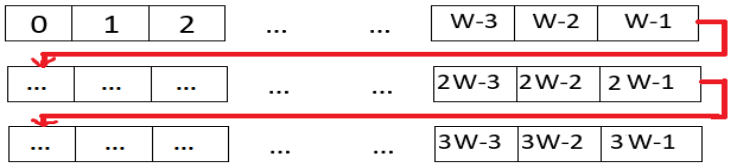

Figure 3.6: 3 Shift Registers

Since we are reading the image pixel by pixel, we can't find out the neighbouring pixels of the main treated pixel.The solution is to create a shift register of 3 rows of the image and has the same width of the image. Now we read a pixel we extract the mask matrix, which is the last 3 columns of the shift register.

```
Library ieee;
```

```
use ieee.std_logic_1164.all ;
use work.extra_types.all;
use ieee.numeric_std.all;


entity REG_DEC_PIXEL_ALT_3x3 is
        generic(N: integer := 10;W: integer:=320);
        port(
                        grayscale : IN std_logic_vector(N-1 downto 0);
                        CLK, rst: in std_logic;
                        mask : out pixel_matrix(0 to 2,0 to 2)
                );
end REG_DEC_PIXEL_ALT_3x3;
architecture ACH_REG_DEC_PIXEL_ALT_3x3 of REG_DEC_PIXEL_ALT_3x3 is
        Component REG_DEC_PIXEL is
        generic(N: integer := 10;W: integer:=320);
                port(
                                grayscale : IN std_logic_vector(N-1 downto 0);
                                CLK, rst: in std_logic;
                                grayscale_out: OUT std_logic_vector(N-1 downto 0)
                        );
        end Component;
        Component BAS_PIXEL is
                generic (N: integer :=10);
                port(
                        grayscale : IN std_logic_vector(N-1 downto 0);
                        CLK, rst: in std_logic;
                        grayscale_out: OUT std_logic_vector(N-1 downto 0)
                );
        end Component;
        signal M11,M12,M13,M21,M22,M23,M31,M32,M33 : std_logic_vector (N-1 downto 0);
begin
        REG_DEC_1: REG_DEC_PIXEL generic map (N,W) port map (grayscale,clk,rst,M33);
        REG_DEC_2: REG_DEC_PIXEL generic map (N,W) port map (M33,clk,rst,M23);
        REG_DEC_3: REG_DEC_PIXEL generic map (N,W) port map (M23,clk,rst,M13);
```

```
        BAS_PIX_33_32: BAS_PIXEL generic map (N) port map (M33,clk,rst,M32);
        BAS_PIX_23_22: BAS_PIXEL generic map (N) port map (M23,clk,rst,M22);
        BAS_PIX_13_12: BAS_PIXEL generic map (N) port map (M13,clk,rst,M12);

        BAS_PIX_32_31: BAS_PIXEL generic map (N) port map (M32,clk,rst,M31);
        BAS_PIX_22_21: BAS_PIXEL generic map (N) port map (M22,clk,rst,M21);
        BAS_PIX_12_11: BAS_PIXEL generic map (N) port map (M12,clk,rst,M11);

        mask(0,0)<=to_integer(unsigned(M11));
        mask(0,1)<=to_integer(unsigned(M12));
        mask(0,2)<=to_integer(unsigned(M13));
        mask(1,0)<=to_integer(unsigned(M21));
        mask(1,1)<=to_integer(unsigned(M22));
        mask(1,2)<=to_integer(unsigned(M23));
        mask(2,0)<=to_integer(unsigned(M31));
        mask(2,1)<=to_integer(unsigned(M32));
        mask(2,2)<=to_integer(unsigned(M33));

end ACH_REG_DEC_PIXEL_ALT_3x3;
```

3.2.3 Sobel Edge Detector

Figure 3.7: Overview of Sobel entity

3.2.3.1 Convolution Operator

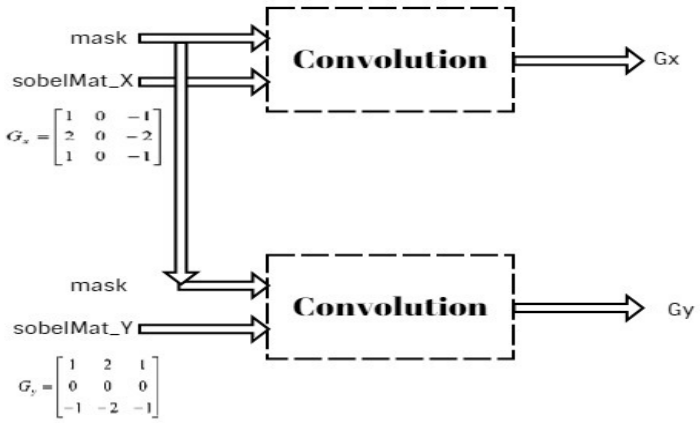

Figure 3.8: Convolution Operator

After receiving the mask matrix of each pixel, we need to compute its convolution for both vertical and horizontal orientations

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.pgm.all;
use work.extra_types.all;
```

```
entity convolution is
        port(
                pix_mat : IN pixel_matrix;
                sobel_mat : IN matrice_int;
    G : OUT integer
        );
end entity;

architecture convolution_arch of convolution is

        begin
                -->return the absolute convolution value of pix_mat & sobel_mat
        G<=abs((pix_mat(0,0)*sobel_mat(0,0)+pix_mat(0,1)*sobel_mat(1,0)
            +pix_mat(0,2)*sobel_mat(2,0))+(pix_mat(1,0)*sobel_mat(0,1)
            +pix_mat(1,1)*sobel_mat(1,1)+pix_mat(1,2)*sobel_mat(2,1))
            +(pix_mat(2,0)*sobel_mat(0,2)+pix_mat(2,1)*sobel_mat(1,2)
            +pix_mat(2,2)*sobel_mat(2,2)));

end architecture;
```

3.2.3.2 Sobel Operator

After computing both of the convolutions values, the final step is to do an approximation of the mathematical equation in order to reduce the complexity of the computations and reach the real time criteria. the approximation is the following :

$$|G| = |Gx| + |Gy|$$

after calculating the pixels values of the new image, we must apply to them a threshold, so they want exceed the value 4095 for each pixel.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.pgm.all;
use ieee.numeric_std.all;
```

```
use work.extra_types.all;


entity sobel is
        port(
                pix_mat : IN pixel_matrix;
                pixout : OUT pixel
        );
end entity;

architecture sobel_arch of sobel is
  component CONVOLUTION is
                port(
                        pix_mat : IN pixel_matrix;
                        sobel_mat : IN matrice_int;
            G : OUT integer
                );
  end component;

        component comparaison is
                port(
                        A: in integer;
                        B: out integer
                );
        end component;

        component signed_adder is
                port(
                        A,B: IN integer;
                        C : OUT integer
                );
        end component;



  signal Gx,Gy,G : integer;
```

```
  signal sum,tmp : integer;
        signal sobel_X : matrice_int :=((-1,0,1),(-2,0,2),(-1,0,1));
        signal sobel_Y : matrice_int :=((-1,-2,-1),(0,0,0),(1,2,1));

  begin

                --> return  |Gx| = pix_mat * sobel_X
                convolut_x: CONVOLUTION port map(pix_mat, sobel_X, Gx);
        --> return  |Gy| = pix_mat * sobel_Y
                convolut_y: CONVOLUTION port map(pix_mat, sobel_Y, Gy);
                --> return |sum| = |Gx| + |Gy|
                somme : signed_adder port map(Gx,Gy,sum);
                --> return 4080 if sum exceed 2^12(max size of pixel)
                comp : comparaison port map(sum,G);
                --> return pixout after passing by sobel filter
                pixout<= to_integer(to_unsigned(G, 12));

end architecture;
```

Chapter 4

Implementation and integration

4.1 Simulation & results

4.1.1 Simulation

Before doing a hardware synthesis of our implementation, we need to make sure that the implementation is valid and the project that we have designed outputs a results as desired.

We have already created a test bench to simulate the project. We will be using **Model Sim** software for the simulation, and for the observance of the various signals between our entities. After processing the test bench, Model Sim will outputs a pgm image as a result, this image will indicate if we are in the right or the wrong path. If we don't get the desired output, we should check our signals and our code to correct the errors and simulate again. the simulation run under these conditions :

- ✓ A PGM input gray scale image (512px x 512 px) represented below
- ✓ a pixel should take 2ns ,the image should take (512px x 512 px x 2ns) 524288 ns , 0.000524288 s

4.1.2 Results

The simulation with an i7 Processor, 8Gb of RAM computer took about 1 hour, and the output is the following figure below :

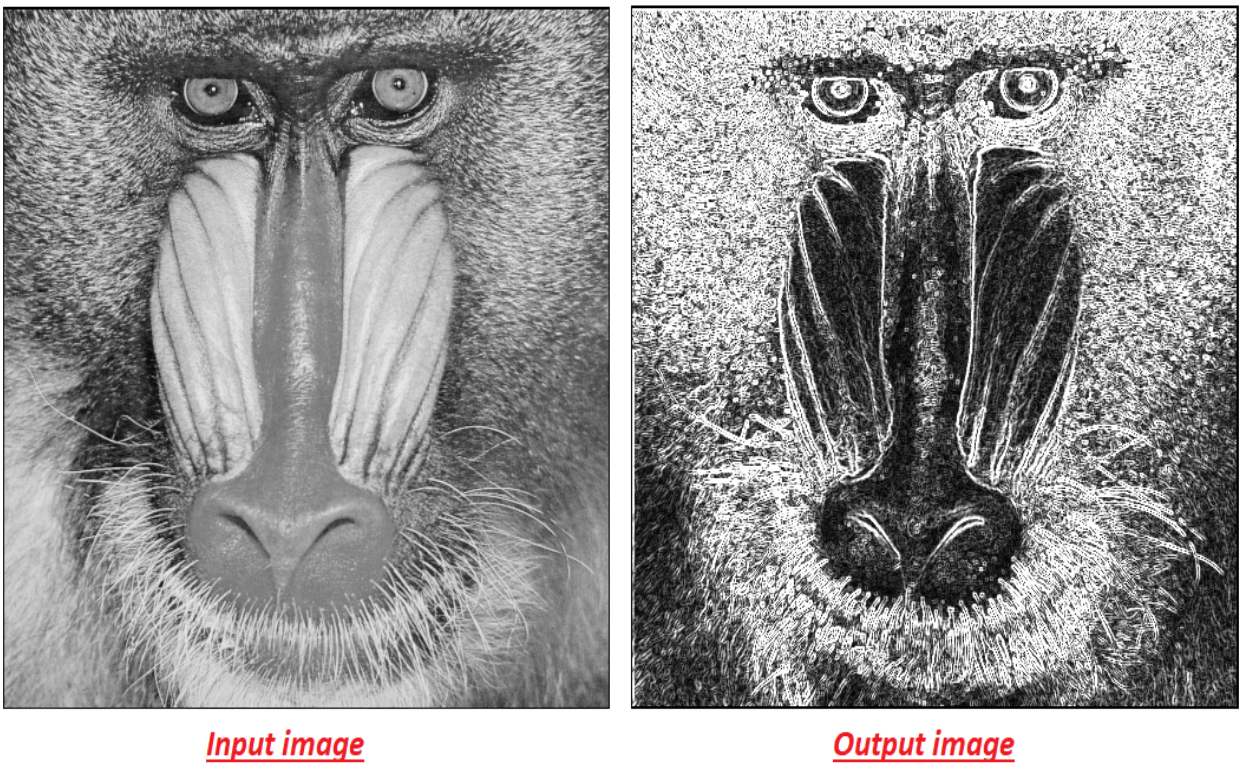

Figure 4.1: Results simulation of Sobel Edge Detector

Chapter 5

problems and challenges

5.1 VHDL type conversions

During the phase of implementation, Soble Edge detector computation, needs summation, multiplications of scalar values, however our input is a pixel which is modeled by the type vector of bits. From this problem comes the need to convert our pixel into a signed scalar value to proceed to the computations.We need to choose signed values because we could find negative values in the convolution phase of the sobel algorithm. Finally we return a value in range 0-4095. The figure below helped us doing the right conversions. It contains the functions that can make the conversations.

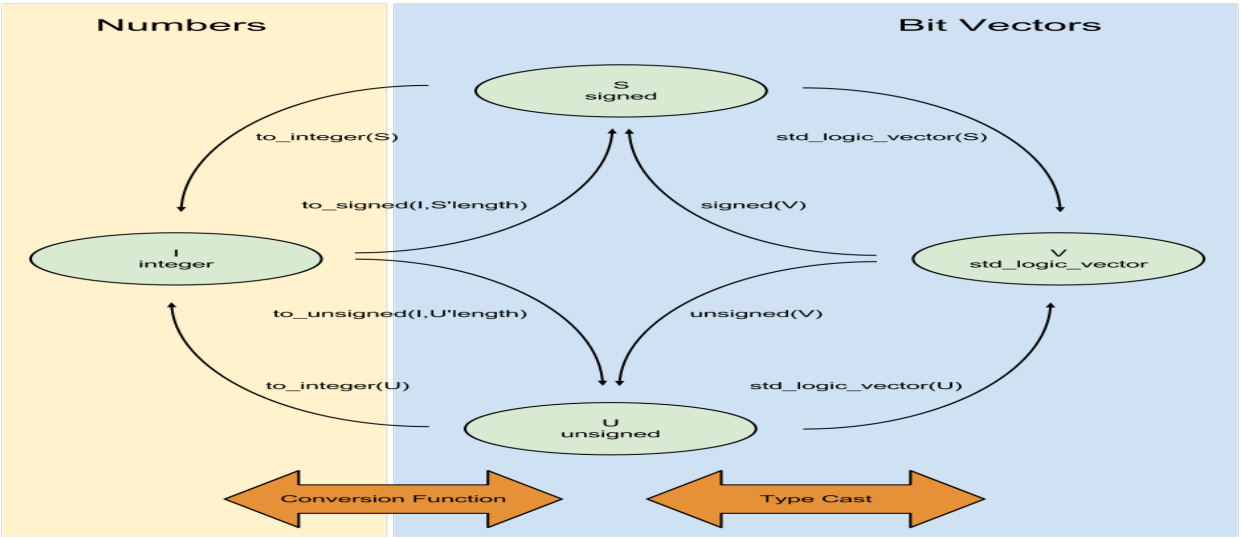

Figure 5.1: VHDL type conversions

General Conclusion

In this project,we learned some of the most important aspects of computer vision, and the constraints that we need to solve like the real time processing criteria. Real time Constraints oriented us to a full custom architecture. After multiple failed attempt to have the right result, we learned how to debug our code using signals. Most of the times we failed even to Design the right entity, but we learned from our mistakes and we kept improving until we had a good simulation result. we tried to synthesize the processing chain using the **Vivado** software. However we couldn't succeed until now to reduce the number of gates in our project to fit the FPGA. Once we reach that goal, we will be able to stream a video From a HDMI port and apply the sobel edge detector in real time.