University of Manouba
National School of Computer Sciences

Integration Project Report

Topic

Hardware implementation of an edge detection algorithm

Realized by

Bahri Abdessattar
Ben kahla Bechir
Hnezli Mohamed

Supervised by

Drs-Ings.

Lobna Kriaa **Mouhamed Masmoudi**

Academic Year : 2016/2017

Contents

List of Figures

Overview

Nowadays artificial intelligence is defined as the area of computer sciences that aims at giving machines the ability to imitate the behavior reputed intelligent of humans so that we can produce machines that are able to perceive its environment and take actions that maximize its chance of success at some goals.
Among the intelligent behaviors a human had we find the visual perception for which we associate the machine vision field. But in order to achieve a machine vision we need some images processing tools. The most important one is edges detection !
In this context, we present our project named of RT_Sobel
Sobel is the most famous edge detection operator it consists of a set of mathematical methods that works at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. These points are typically organized into a set of curved line segments termed edges.

So our task will be to find an optimized hardware solution to implement this algorithm so that it can perform a real-time edges detection!

This report is structured around four chapters. In the first chapter, we will present the theoritical stduy, in the second one, we will move to the requirement specification of our problem and the solution description. The third chapter is devoted to the solution design. And, As for the fourth chapter, it will be dedicated to the implementation and integration into an FPGA circuit. Finally, we will present a general conclusion and some perspectives on possible future improvements to the project.

1 Preliminary study

Introduction

Each project must begin with a theoretical study which aims to clarify the key points of the tackled problem so that we better understand the work we are supposed to do

1.1 Edge overview

An edge is defined as a significant local change of the image intensity. It occurs on the boundary between two different regions of the figure for example the border between a block of red color and a block of yellow one represents an edge.

1.1.1 Edge origins

Edges are caused by a variety of factors :

- Surface normal discontimuity.
- Depth discontimuity
- Surface color discontimuity
- Illumination discontimuity

The figure below illustrates what we have said.

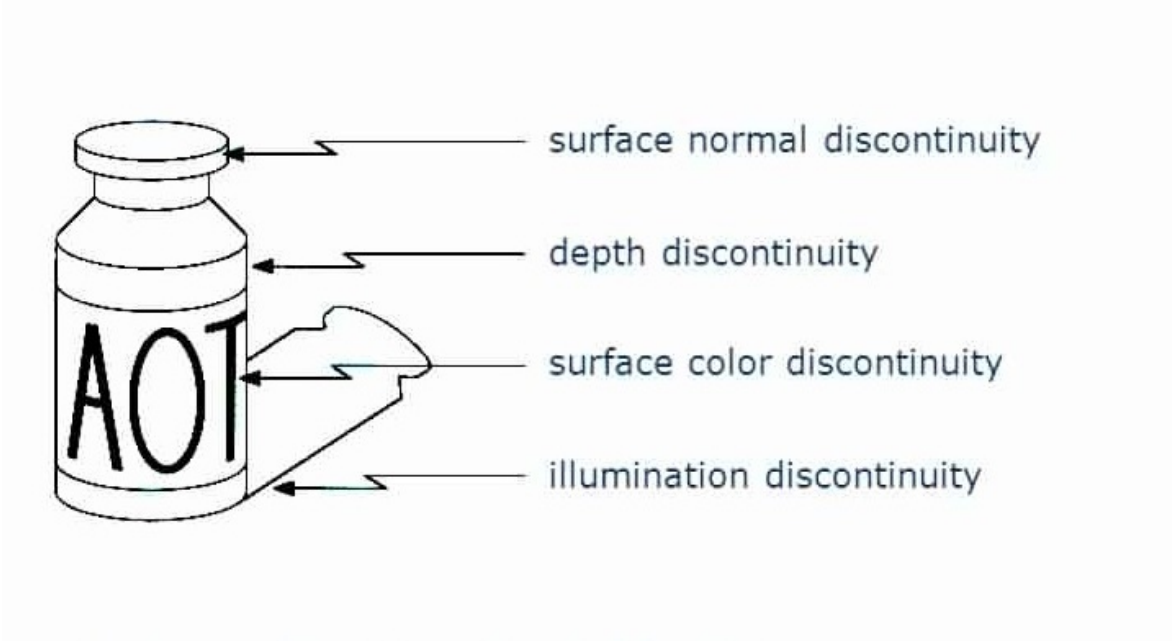

Figure 1.1: Origin of edges

1.2 Edge detection

The Edge detector represents a fundamental image processing tool that allow us to convert a 2d image into a set of curves more compact and easy to use than pixels so that we can extract the salient features of the scene.
The extraction of edges from a given image reduces significantly the amount of data to process by removing useless information while preserving the image's structural characteristics.
we find several edge detection techniques !
but the majority of them could be classified into tow categories :

- The first one works by the detection of the first derivative extremums
- The second one detects the zeros of the second derivative

In 1D change is measured by the derivate but in 2D we use the gradient operator.
As images are discrete functions (matrix of discrete values) we must discrete the gradient operator.
Therefore we find different operator approximating the gradient like (Sobel, Roberts, Prewitt, Canny ...).

We don't forget to sit that edge detection techniques works on gray scale images !

1.2.1 Sobel operator

Among the possible edge detection operator Sobel is the most easiest one to implement. it works by replacing the value of each pixel (expect those on the border) by a new one depending on the neighbor pixels valuation.

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

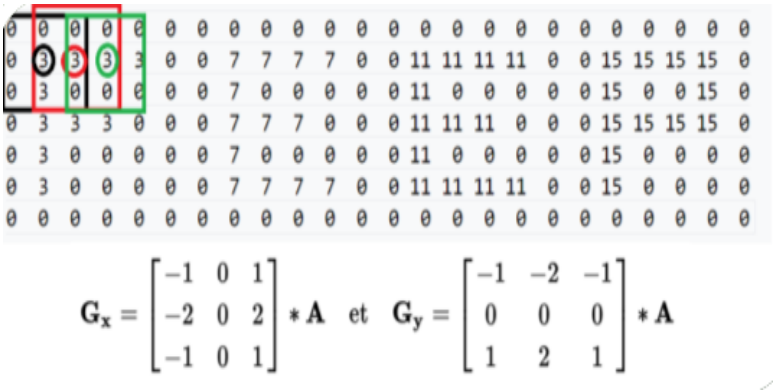

Figure 1.2: Sobel algorithm processing

Conclusion

In this chapter, we presented the results of our theoretical study we talked about edges, their origins, the edge detection process and finally we presented an edge detection operator Sobel.

2 Requirements specification and Solution description

Introduction

The theoretical study was an important step that allowed us to familiarize ourselves with the notions tackled, but above all to be able to identify the problematic, the aim and understand the needs of the project. In this chapter, we will analyze and specify the requirements and give a solution description

2.1 Requirements specification

We need a real time solution for the edge detection problem !

2.2 Constraints

We need to read the image pixel by pixel and at each clock cycle insert the new pixel in the processing chain.
With new high-resolution standards, edge detection operators must receive input data at a rate up to 3 Giga samples per second. For example, the HD video standard (1080x1920 p) requires 60 fps, with approximately 2.1 megapixel per frame, and 126 megapixels per second resulting a data rate of 3 Gigabit per second using uncompressed RGB encoding. So edge detection operators require high computation power!

2.3 Solution Description

Considering the requirement and the real time constraints of the problem we choose to design a specific hardware architecture that performs the edge detection in a real time way.
For Test and simplification reasons the solution will be run in simulation mode

and we will use tow Test Bench in order to read the input image and to save the resulting one.

2.4 Work steps

In order to achieve our goal we are supposed to follow these steps :

- Read a PGM image
- Insert the image pixels one by one in the processing chain.
- Generate a mask containing the pixel to be processed and its 8 neighbors .
- compute the convolution product between the image mask and Sobel kernels
- Store the processed image.

N.B : We will use a VHDL library which will help us in reading the image pixels. This library was written by Mr Martin Thoompson and was found on Github at the following a address :
`https://github.com/nkkav/image_processing_examples`

Conclusion

During this chapter we specified the requirement that our edge detection solution must fit, then we gave a general description of the work we are going to do something which will be more clear in the design chapter.

3 Architecture Design

Introduction

After specifying the different needs to be met,this chapter will focus on the general and detailed architecture of our solution.

3.1 Global Architecture design

In this part of the report we will describe the global design of our solution,represented by the figure below, we will show the different entities that we will use and the interconnections between them.

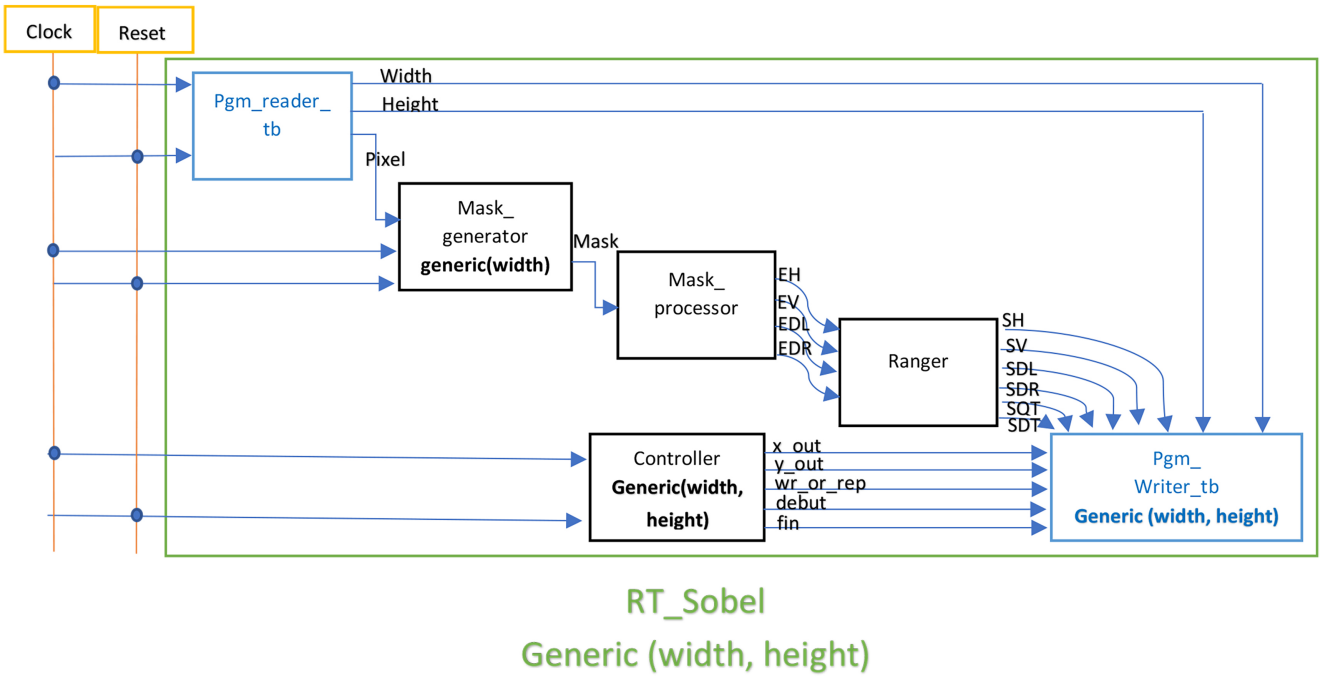

Figure 3.1: Global Architecture design

The solution consists of 4 synthtesizable entities which represent the pro-

cessing chain of our RT_Sobel solution(the black entities). In order to be able to test our architecture we performed tow test bench (blue entities) which will allow us to read a pgm image file in simulation mode, so that we could supply our processing chain with the necessary pixels stream, and then write them back into a resulting pgm image thanks to a second test bench.

3.2 Detailed Architecture Design and Entities description

After presenting the global solution's architecture we will try in this section to explain more in details the role of each particular entity we have used as well as the entities internal architecture.

3.2.1 PGM reader

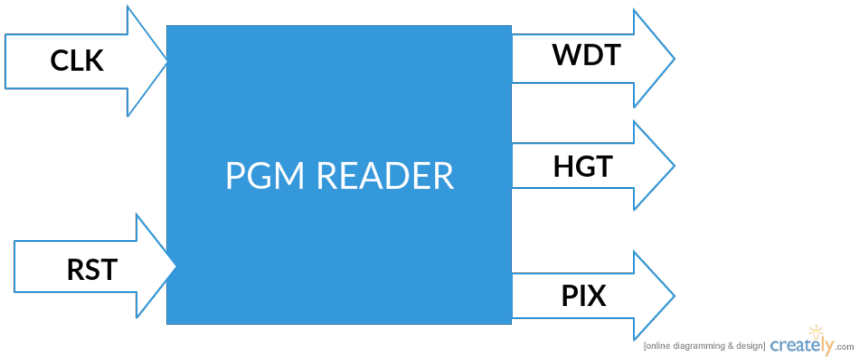

Figure 3.2: PGM reader

The Pgm_reader test bench is used only for delivering the image pixel by pixel at each clock cycle it uses the libraries of Mr Martin Thompson previously cited.

```
library ieee;
use ieee.std_logic_1164.all;
use work.pgm.all;
use work.common.all;

-- we suppose that the max image resolution is 4096x4096

entity pgm_reader_tb is
   port (  c,r : IN std_logic;
         wdt,hgt : out dimension;
         pix : out pixel
```

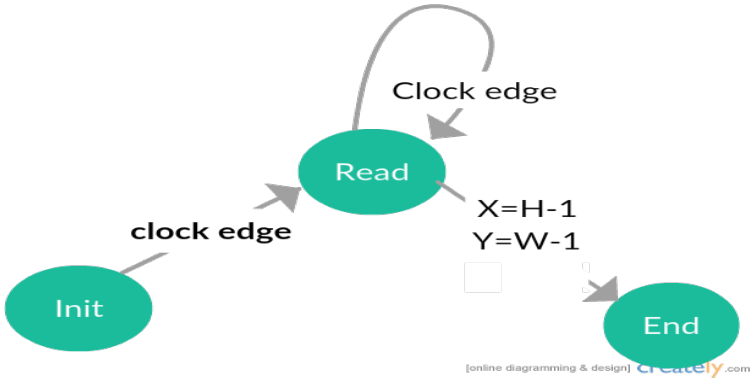

Figure 3.3: PGM reader state machine

```
        );
end entity;

architecture pgm_reader_arch_tb of pgm_reader_tb is
   Type ST is (INIT,RD,FIN);
   Signal state: ST:=INIT;


begin
   P : Process(c)
      variable i : pixel_matrix_ptr;
      variable H : dimension :=0;
      variable W : dimension :=0;
      variable x : dimension :=0;
      variable y : dimension :=0;
   Begin
   if(c='1') then
      IF (state = INIT) then
         i := pgm_read("feep.pgm");
         W:=i.all'length(1);
         wdt<=W;
         H:=i.all'length(2);
         hgt<=H;
         state<=RD;
         -- Pour ne pas perdre un cycle d'horloage
         pix<=i.all(y,x);
         if(x=H-1) then
            if(y=W-1) then
               state<=FIN;
            elsif(y<W-1) then
               x:=0;
               y:=y+1;
            end if;
         elsif(x<H-1)then
            if(y=W-1)then
               y:=0;
               x:=x+1;
            elsif(y<W-1) then
```

```
            y:=y+1;
          end if;
        end if;
      elsif (state = RD) then
        pix<=i.all(y,x);
        if(x=H-1) then
          if(y=W-1) then
            state<=FIN;
          elsif(y<W-1) then
            x:=0;
            y:=y+1;
          end if;
        elsif(x<H-1)then
          if(y=W-1)then
            y:=0;
            x:=x+1;
          elsif(y<W-1) then
            y:=y+1;
          end if;
        end if;

      end if;
    end if ;
    End Process P;

end pgm_reader_arch_tb ;
```

3.2.2 Mask generator

Figure 3.4: Mask generator Entity

The mask generator entities allow the extraction of an image mask which is a 3x3 pixels matrix the central pixels is the pixel to be treated by the Sobel operator.

at each clock cycle a new pixel matrix is generated representing the next pixel and its neighbors.
The figure below explains more in detail the internal architecture of the entity.
In fact it is composed of 3 cascade shift registers.

Figure 3.5: Mask generator Architecture

the 2 first shift registers are of size the width of the image to process, that is why we used the generic parameter width, And a third one of size 3.
the pixels enters from the small one, passes through the other big one and finally are ejected from the mask processor in order the move the window and get the next mask.
N.B : shift registers of size N are composed of N Flip Flop each one capable of storing a data oh 1 byte which is the pixel.

Mask_generator.vhd

we find below the VHDL description of the mask_generator.

```
library ieee;
use ieee.std_logic_1164.all;
use work.common.all;

entity mask_generator is
  generic (width : integer range 0 to 4096 :=24);--
  port ( datain : IN pixel ;
       clock : IN std_logic ;
       reset : IN std_logic ;
       mask : OUT pixel_matrix(0 to 2,0 to 2)
       );
end entity;

architecture arch of mask_generator is
```

```
  signal S : pixel_array(2 downto 0);
  signal CR2,CR3 : pixel_array( (width-1) downto 0 );
  signal CR1 : pixel_array(2 downto 0);
  signal pix : pixel;
  component reg_dec_n is
    generic(N : integer :=8);
    port( d : IN pixel;
      clk,rst: in std_logic;
      o: out pixel;
      output: out pixel_array((N-1) downto 0)
    );
  end component;

begin

  reg_dec1 : reg_dec_n
    generic map (N=>3)
    port map (clk=>clock,d=>datain,rst=>reset,o=>S(0),output=>CR1);

  reg_dec2 : reg_dec_n
    generic map (N=>width)
    port map (clk=>clock,d=>S(0),rst=>reset,o=>S(1),output=>CR2);

  reg_dec3 : reg_dec_n
    generic map (N=>width)
    port map (clk=>clock,d=>S(1),rst=>reset,o=>S(2),output=>CR3);

  mask_gliss:process(clock)
  variable cpt : integer range 0 to 5001:=0;
  begin
    if(clock='1')then
      if (cpt<2*width+3) then
        cpt:=cpt+1;
      else
        mask(0,0)<=CR3((width-1));
        mask(0,1)<=CR3((width-2));
        mask(0,2)<=CR3((width-3));
        mask(1,0)<=CR2((width-1));
        mask(1,1)<=CR2((width-2));
        mask(1,2)<=CR2((width-3));
        mask(2,0)<=CR1(2);
        mask(2,1)<=CR1(1);
        mask(2,2)<=CR1(0);
      end if;
    end if;

  end process;


end arch;
```

reg_dec_n.vhd

we find below the VHDL description of the shift register.

```
library ieee;
use ieee.std_logic_1164.all;
use work.common.all;

entity reg_dec_n is
   generic(N : integer :=8);
   port( d : IN pixel;
      clk,rst: in std_logic;
      o: out pixel;
      output: out pixel_array((N-1) downto 0)
   );
end entity;

architecture arch of reg_dec_n is

component bascule_d is
   port( D:IN pixel;
      H,R : IN std_logic ;
      Q : OUT pixel
   );
end component ;

signal c:pixel_array(N downto 0);
begin
   c(0)<=D;
   o<=c(N);
   output<=c(N downto 1);
   for1:for i in 0 to N-1 generate
      basi: bascule_d port map(H=>clk,D=>c(i),R=>rst,Q=>c(i+1));
   end generate;
end arch;
```

bascule.vhd

we find below the VHDL description of the flip flop.

```
library ieee;
use ieee.std_logic_1164.all;
use work.common.all;

entity bascule_d is
   port( D:IN pixel;
      H,R : IN std_logic ;
      Q : OUT pixel
   );
```

```
end bascule_d ;

architecture bascule_comp of bascule_d is
begin
   process(H)
   begin
      --if (R'event) then
         --Q<=0;
      --end if;

      if(H='1') then
         Q<=D;
      end if;
   end process;
end bascule_comp;
```

3.2.3 Mask Processor

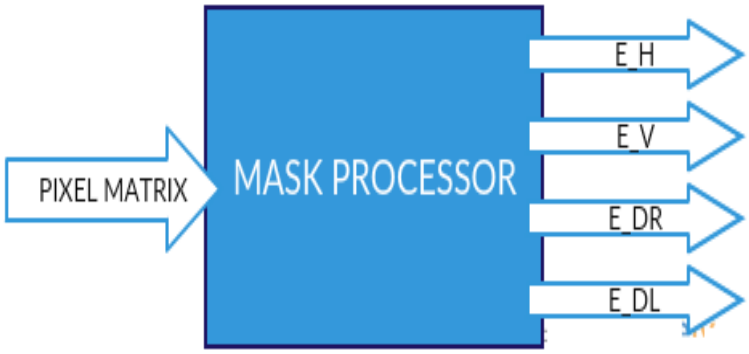

Figure 3.6: Mask processor entity

The mask processor entity computes a new value for each central pixel of the mask that it gets at each clock cycle.
the new pixels value is weighted by its 8 neighbors valuation.
the new value is get through the convolution between the mask and Sobels kernels.
We used 4 sobels kernels in order to detect :

- ✓ Vertical edge
- ✓ Horizontal edge.
- ✓ Left diagonal edge.

✓ Right diagonal edge.

the entity gives as output 5 pixels :

✓ EV : the current pixel's new value for Vertical edges detection.

✓ EH : the current pixel's new value for Horizontal edges detection.

✓ EDR : the current pixel's new value for right diagonal edges detection.

✓ EDL : the current pixel's new value for left diagonal edges detection.

The entity is composed of 4 Convoltioneer.

Mask_Processor.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use work.common.all;


entity mask_processor is
   port ( msk : IN pixel_matrix(0 to 2,0 to 2);
        EH,EV,EDL,EDR : out long6pixel
        );
end entity;

architecture mask_processor_arch of mask_processor is
signal MH,MV,MDL,MDR : pixellong_matrix(0 to 2,0 to 2);

component convolutionneur is
   port ( mask : IN pixel_matrix(0 to 2,0 to 2);
        ker : IN pixellong_matrix(0 to 2,0 to 2);
        respix: out long6pixel
        );
end component;

begin



MH(0,0)<=1;MH(0,1)<=2;MH(0,2)<=1;
MH(1,0)<=0;MH(1,1)<=0;MH(1,2)<=0;
MH(2,0)<=-1;MH(2,1)<=-2;MH(2,2)<=-1;
```

```
MV(0,0)<=-1;MV(0,1)<=0;MV(0,2)<=1;
MV(1,0)<=-2;MV(1,1)<=0;MV(1,2)<=2;
MV(2,0)<=-1;MV(2,1)<=0;MV(2,2)<=1;

MDR(0,0)<=2;MDR(0,1)<=1;MDR(0,2)<=0;
MDR(1,0)<=1;MDR(1,1)<=0;MDR(1,2)<=-1;
MDR(2,0)<=0;MDR(2,1)<=-1;MDR(2,2)<=-2;

MDL(0,0)<=0;MDL(0,1)<=1;MDL(0,2)<=2;
MDL(1,0)<=-1;MDL(1,1)<=0;MDL(1,2)<=1;
MDL(2,0)<=-2;MDL(2,1)<=-1;MDL(2,2)<=0;

hconv : convolutionneur port map(mask=>msk,ker=>MH,respix=>EH);
vconv : convolutionneur port map(mask=>msk,ker=>MV,respix=>EV);
dlconv : convolutionneur port map(mask=>msk,ker=>MDL,respix=>EDL);
drconv : convolutionneur port map(mask=>msk,ker=>MDR,respix=>EDR);


end mask_processor_arch;
```

Convolutionner.vhd

The Convolutionner is a specific architecture for 3x3 matrix convolution. The figure below shows the internal architecture of the convolutionner.

And We found below the VHDL description of the entity.
N.B : We have used some multiplier and adders.

```
library ieee;
use ieee.std_logic_1164.all;
use work.common.all;


entity convolutionneur is
   port ( mask : IN pixel_matrix(0 to 2,0 to 2);
        ker : IN pixellong_matrix(0 to 2,0 to 2);
        respix: out long6pixel
        );
end entity;

architecture arch_convolutionneur of convolutionneur is

signal s1 : longlongpixel_array(0 to 8);
signal s2 : longlonglongpixel_array(0 to 3);
signal s3 : long4pixel_array(0 to 1);
signal s4 : long5pixel_array(0 to 1);
```

```
   component signed_mult is
   port(A : in longpixel;
      B : in longpixel;
      Y : out longlongpixel);
   end component;

component signed_1_adder is
port(A : in longlongpixel;
   B : in longlongpixel;
   Y : out longlonglongpixel);
end component;

component signed_2_adder is
port(A : in longlonglongpixel;
   B : in longlonglongpixel;
   Y : out long4pixel);
end component;

component signed_3_adder is
port(A : in long4pixel;
   B : in long4pixel;
   Y : out long5pixel);
end component;

component signed_4_adder is
port(A : in long5pixel;
   B : in long5pixel;
   Y : out long6pixel);
end component;

begin

m1:signed_mult port map(ker(0,0),mask(0,0),s1(0));
m2:signed_mult port map(ker(0,1),mask(1,0),s1(1));
m3:signed_mult port map(ker(0,2),mask(2,0),s1(2));
m4:signed_mult port map(ker(1,0),mask(0,1),s1(3));
m5:signed_mult port map(ker(1,1),mask(1,1),s1(4));
m6:signed_mult port map(ker(1,2),mask(2,1),s1(5));
m7:signed_mult port map(ker(2,0),mask(0,2),s1(6));
m8:signed_mult port map(ker(2,1),mask(1,2),s1(7));
m9:signed_mult port map(ker(2,2),mask(2,2),s1(8));


s4(1)<=s1(8);


a1:signed_1_adder port map(s1(0),s1(1),s2(0));
a2:signed_1_adder port map(s1(2),s1(3),s2(1));
a3:signed_1_adder port map(s1(4),s1(5),s2(2));
a4:signed_1_adder port map(s1(6),s1(7),s2(3));

a5:signed_2_adder port map(s2(0),s2(1),s3(0));
a6:signed_2_adder port map(s2(2),s2(3),s3(1));
```

```
a7:signed_3_adder port map(s3(0),s3(1),s4(0));

a8:signed_4_adder port map(s4(0),s4(1),respix);

end arch_convolutionneur;
```

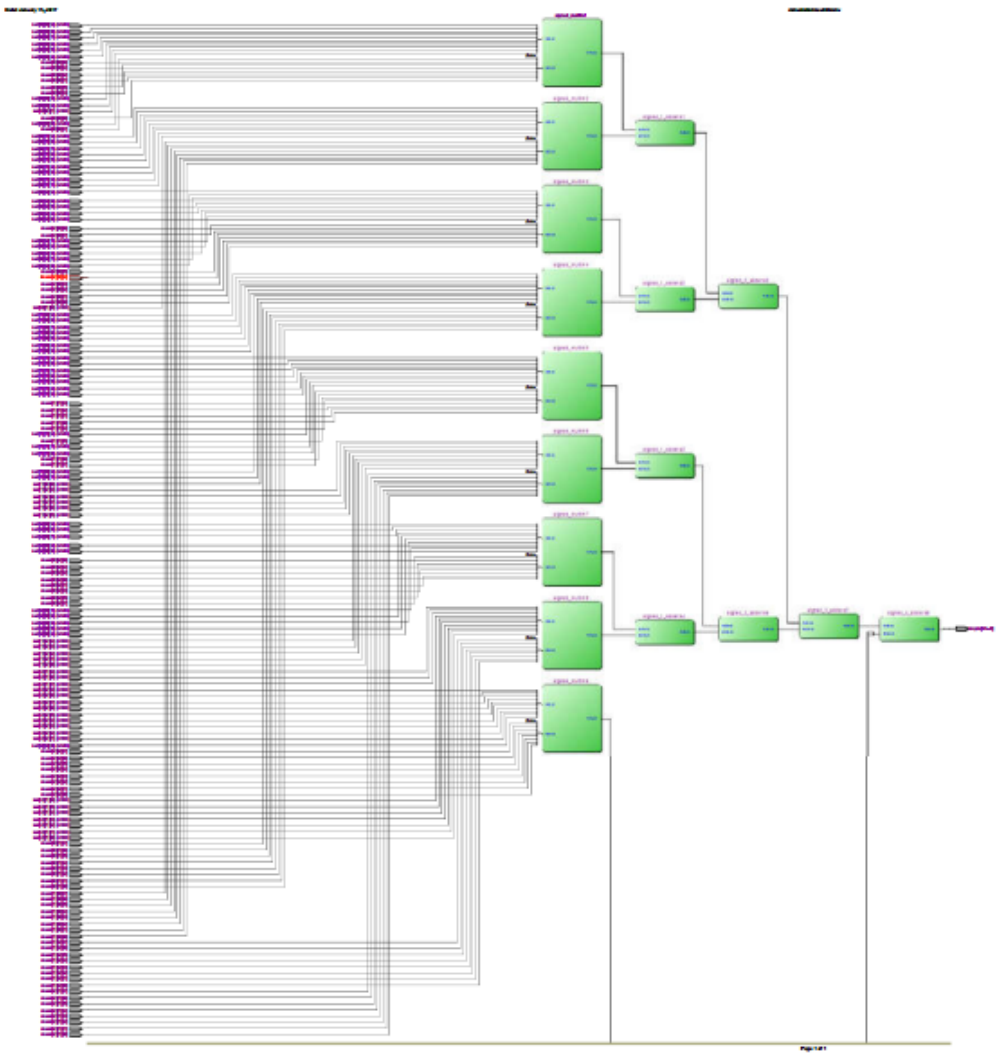

Figure 3.7: Convolutionner Architecture

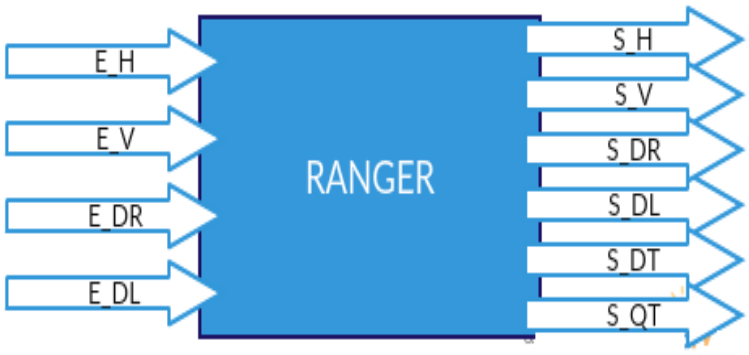

Figure 3.8: Ranger entity

3.2.4 Ranger

The ranger entity computes the absolute value for each entry coming from the mask processor as well as thresholding of this value to a maximum vale of 255. The ranger also computes the values of SQT and SDT which are combination between horizontal an vertical edges for SQT and diagonal left and right for SDT.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use WORK.COMMON.ALL;

-- main entity
entity ranger is
port (E_H, E_V, E_DL, E_DR       : in long6pixel;
   --max : IN pixel;
      S_H, S_V, S_DL, S_DR, S_QT, S_DT: out pixel);
end ranger;

architecture RANGER_ARCH of ranger is

signal SH: long7pixel:=0;
signal SV: long7pixel:=0;
signal SDL: long7pixel:=0;
signal SDR: long7pixel:=0;
signal SQT: long8pixel:=0;
signal SDT: long8pixel:=0;
begin
-- check if input is positif, else calculate its 2 complement

SH  <= (E_H) when ((to_signed(E_H,24) and "100000000000000000000000") =
    "000000000000000000000000") else
     to_integer(not (to_signed(E_H,25)) + 1) ;
```

```
SV  <= (E_V) when ((to_signed(E_V,24) and "100000000000000000000000") =
    "000000000000000000000000") else
     to_integer(not (to_signed(E_V,25)) + 1);
SDL <= (E_DL) when ( ( to_signed(E_DL,24) and "100000000000000000000000" ) =
    "000000000000000000000000" ) else
     to_integer(not ( to_signed(E_DL,25) ) +1);
SDR <= (E_DR) when ( ( to_signed(E_DR,24) and "100000000000000000000000" ) =
    "000000000000000000000000" ) else
     to_integer(not ( to_signed(E_DR,25) ) +1);

SQT<=SH+SV;

SDT<=SDL+SDR;

-- here we set values greater than 255 to 255

S_H  <= (SH) when (SH < 255) else
        (255);
S_V  <= (SV) when (SV < 255) else
        (255);
S_DL <= (SDL) when (SDL < 255) else
        (255);
S_DR <= (SDR) when (SDR < 255) else
        (255);
S_QT <= SQT when (SQT<255) else
     255;
S_DT <= SDT when (SDT<255) else
     255;

end RANGER_ARCH;
```

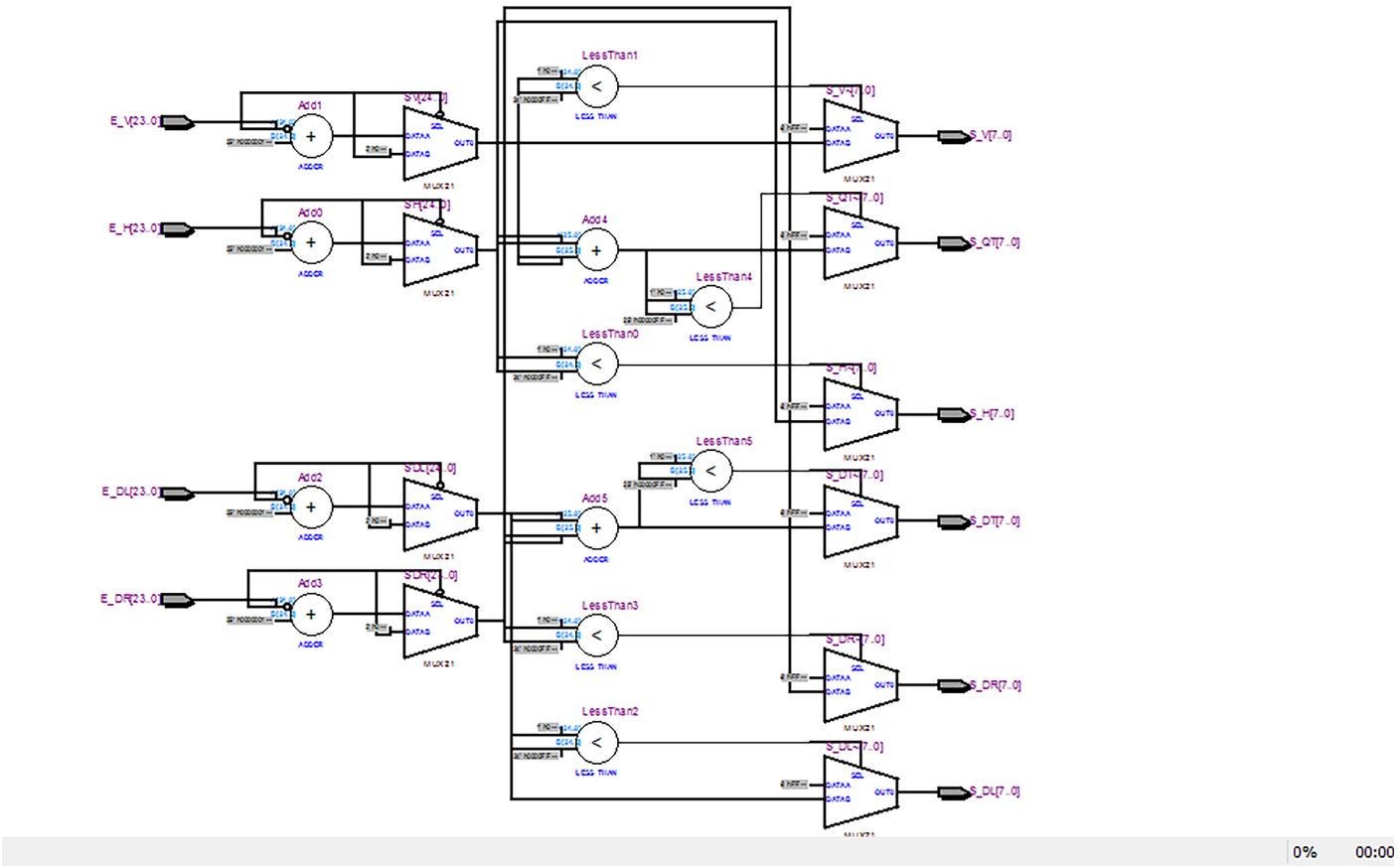

Figure 3.9: Ranger Architecture

3.2.5 Controllor

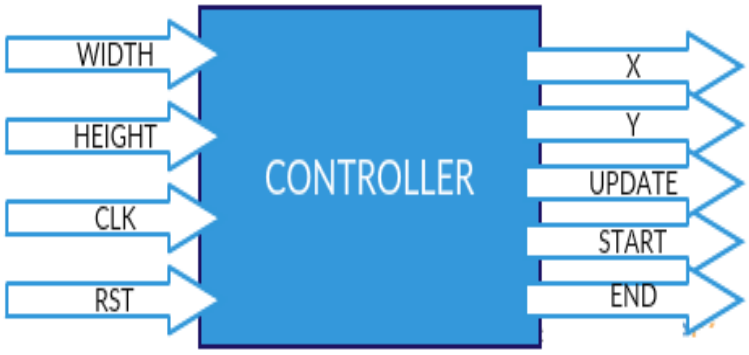

Figure 3.10: Controllor entity

The Mask processor entity computes the news values of the pixel on the fly but in some particular situations (border pixels) the matrix driven by the mask generator must not processed but only the value will be replaced by a 0

because we can not find all the 8 neighbors.
The role of the controller is to emit for the pgm_writer entity signals for synchronizing the pixel writing in the image file.

We find below the state machine of our entity and its VHDL description.

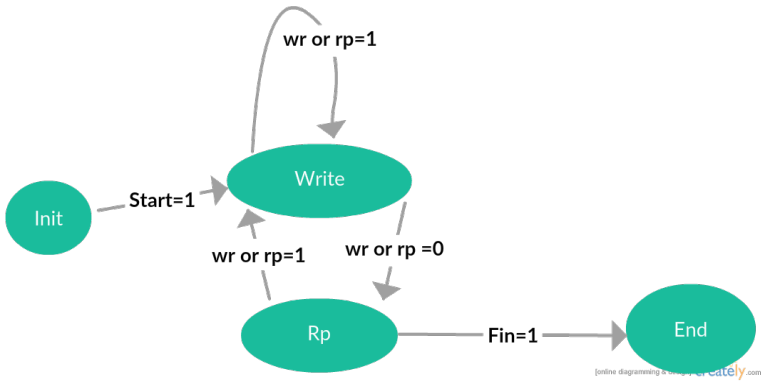

Figure 3.11: Controller state machine

```
library ieee;
use ieee.std_logic_1164.all;
use work.common.all;

entity controlleur is
   generic(wdt : IN dimension:=512;
      hgt: IN dimension:=512);
   port(clock,rst : IN std_logic;
      x_out,y_out : out dimension;
      wr_or_rep : OUT std_logic;
   deb,fin_out : out std_logic
   );
end entity;

architecture arch_controlleur of controlleur is
      Type STATE is (BFRSTART,START_WR,RP,FIN);
      Signal st: STATE:=BFRSTART;
      signal nst : STATE;
      signal cp : nbele;

      signal x : dimension:=0;
      signal y : dimension:=0;


begin

   State_update : Process(clock)
   Begin
```

```
   if (clock='1') then
      cp<=cp+1;
      st<= nst;
   end if ;
End Process State_update;

Actions :Process(st,clock, nst)
Begin
   -- Mise a jou de x et y
   if (clock'event) and (clock='1') then
      if(st=BFRSTART) and (nst=BFRSTART) THEN
         x<= (cp) / wdt;
         y<= (cp) rem wdt ;
            x_out<= (cp) / wdt;
         y_out<= (cp) rem wdt;

      ELSE
         x<= (cp-(wdt+2)) / wdt;
         y<= (cp-(wdt+2)) rem wdt ;
            x_out<= (cp-(wdt+2)) / wdt;
         y_out<= (cp-(wdt+2)) rem wdt;
      END IF ;
   end if ;
   -- Mise a jour de deb et fin
   CASE st IS
      WHEN BFRSTART =>
         IF(nst=st)THEN
            deb<='0';
            fin_out<='0';
         ELSE
            deb<='1';
            fin_out<='0';
         END IF ;

      WHEN START_WR => deb<='1';
               fin_out<='0';
            wr_or_rep<='1';

      WHEN RP =>  deb<='1';
            fin_out<='0';
            wr_or_rep<='0';

      WHEN FIN =>   deb<='0';
            fin_out<='1';

   END CASE;

End Process Actions;

next_state_update :Process(st,x,y,cp)
Begin
   CASE st IS
      WHEN BFRSTART => IF (cp<2*wdt+3) THEN
```

```
            nst<= st;
          ELSE
            nst<= START_WR;

          END IF;

      WHEN START_WR => IF (y>0)and(y<wdt-2) THEN
          nst<= st;
        ELSE
          nst<= RP;
        END IF;

      WHEN RP => IF (x=hgt-1) and (y= 0) THEN
          nst<= FIN;
              ELSIF (y=wdt-1) THEN
          nst<= st;
          ELSE
          nst<= START_WR;
          END IF;
      WHEN FIN =>
    END CASE;

  End Process next_state_update;


end arch_controlleur;
```

3.2.6 PGM writer

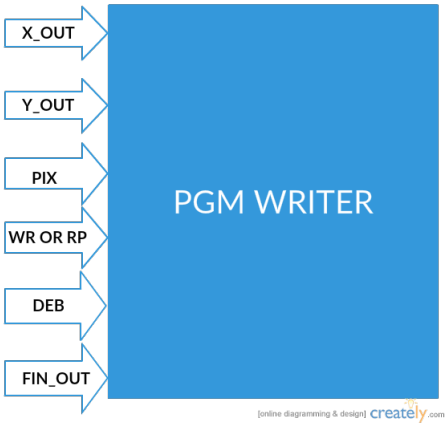

Figure 3.12: Pgm writer

The pgm_writer test bench is used for storing the resulting image pixels from the ranger according to signals comping from the controller.
it also uses the previously cited library.
We find below its VHDL description and state machine.

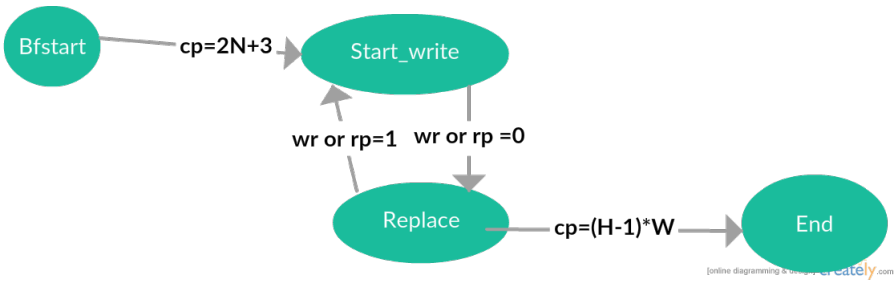

Figure 3.13: PGM writer state machine

```
library ieee;
use ieee.std_logic_1164.all;
use work.pgm.all;
use work.common.all;

-- we suppose that the max image resolution is 4096x4096

entity pgm_writer_tb is
   generic(wdt : IN dimension:=512;
      hgt: IN dimension:=512);
   port (  Pix: IN pixel;
         x_out,y_out : IN dimension;
         wr_or_rep : IN std_logic;
         deb,fin_out : IN std_logic
         );
end entity;

architecture pgm_writer_arch_tb of pgm_writer_tb is

   type ST is (INIT,WR_RP,FIN,SAVE);
   signal state: ST:=INIT;


begin
   P1 : Process(x_out,y_out)
   variable m  : pixel_matrix_ptr := null;

   Begin
      CASE state IS
         WHEN INIT =>    -- allocation de la matrice
                m:=new pixel_matrix(0 to wdt-1, 0 to hgt-1);
                -- Remplissa des N+1 1er pixels
                   for i in 0 to wdt-1 loop
                      m.all(i,0):=0;
                   end loop;
                   m.all(0,1):=0;
                -- attendre que deb = 1
                   --wait until deb='1';
                -- changer d'etat
```

```
                if(deb='1') THEN state<=WR_RP; END IF;
        WHEN WR_RP => IF(wr_or_rep='1') THEN
                m.all(y_out,x_out):=Pix;
               ELSE
                m.all(y_out,x_out):=0;
               END IF;
               iF(fin_out='1')THEN
                state<=FIN;
               END IF;
        WHEN FIN => -- Remplissa des N derniers pixels
                for i in 0 to wdt-1 loop
                   m.all(i,hgt-1):=0;
                end loop;
                state<= SAVE;
        WHEN SAVE => pgm_write ("sobel_out.pgm",m.all);
     END CASE;

   END Process P1;

end pgm_writer_arch_tb ;
```

Conclsion

We have presented in this chapter the conception we have established for this system. This design phase allowed us to describe in a comprehensive and detailed manner the desired functioning of the system in order to facilitate its realization and maintenance. In the next chapter, we will shows the realization stage results.

4 Implementation and integration

Introduction

This final chapter is devoted to the realization part. It shows our architecture simulation results and the synthesis report about the utilization of the FPGA board.

4.0.1 The Simulation results

this section focuses on the presentation of the simulation results.
Simulation was run under those conditions

- 100 ns clock (100 MHz clock)
- A PGM input gray scale image (300px x 246px) represented below

Figure 4.1: RT_Sobel test input

The figure below represents the simulation output after 73800(=300*246) clock cycles.
The simulation with an i5 processor, 8Gb RAM computer toked about 15 min. but it should take 7 380 000 ns = 7.38 s if it was executed on an FPGA board. Some optimization should be done for the real time constraints in video mode.

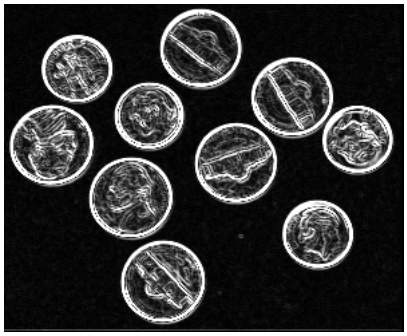

Figure 4.2: RT_Sobel test output

4.0.2 Synthesis Report

The aim of our project is to provide a real time edge detector .
And as the simulation results was satisfactory it is time to test the ability of Synthesizing the processing chain of our architecture,So we will remove the two test bench and represent on the next figure the synthesis report delivered by Quartus.
Synthesis was performed for an Altera Cyclone III EP3C25F324CSN FPGA board with its native design ,analyses and synthesis software Quarus II 13.0.
the report is presented in the figure below.

Flow Status	Successful - Thu Jan 12 10:00:55 2017
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	RT_Sobel
Top-level Entity Name	RT_Sobel
Family	Cyclone III
Device	EP3C25F324C6
Timing Models	Final
Total logic elements	2,984 / 24,624 (12 %)
Total combinational functions	2,972 / 24,624 (12 %)
Dedicated logic registers	81 / 24,624 (< 1 %)
Total registers	81
Total pins	79 / 216 (37 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 4.3: Quartus report

Conclusion

Through this Chapter we was able to validate our architecture design,so now we can move forward and think of replacing the test bench with other blocks in order to implement a fully synthesizable real time detection architecture and test it with our FPGA card

General conclusion

In this project, we started from a problem which is the real time edge detection, something that pushed us to a theoretical study which provides us with the necessary information about the problem we are dealing with.
Real time Constraints oriented us to a full custom architecture.
We designed our hardware edge detector and then we began implementing this design, entity by entity, and after entirely describing every entity unitary tests was performed via simulation with ModelSim.
Finally we reached a fully functional architecture which we tested with some input images and he results was satisfying, so we tried to synthesize the processing chain and we gave the FPGA board utilization report.
Further improvement could evidently be done, we think now use a video camera as image source and output the results into a VGA interface so that we can achieve the real gal of our project !

Types Declaration

We find below the new types we defined in the file common.vhd.

```
library ieee;
use ieee.std_logic_1164.all;


package common is
   subtype coordinate is natural;
   subtype dimension is integer range 0 to 4096;
   subtype nbele is integer range 0 to 16777216;
   subtype pixel is integer range 0 to 255;-- 8 bits
   subtype longpixel is integer range -512 to 511; --10 bits
   subtype longlongpixel is integer range -524288 to 524287; --20 bits
   subtype longlonglongpixel is integer range -1048576 to 1048575;--21 bits
   subtype long4pixel is integer range -2097152 to 2097151;--22 bits
   subtype long5pixel is integer range -4194304 to 4194303;--23 bits
   subtype long6pixel is integer range -8388608 to 8388607;--24 bits
   subtype long7pixel is integer range -16777216 to 16777215;--25 bits
   subtype long8pixel is integer range -33554432 to 33554431;--26 bits
   type pixel_array is array(natural range <>) of pixel;
   type pixel_array_ptr is access pixel_array;
   type longpixel_array is array(natural range <>) of longpixel;
   type longlongpixel_array is array(natural range <>) of longlongpixel;
   type longlonglongpixel_array is array(natural range <>) of longlonglongpixel;
   type long4pixel_array is array(natural range <>) of long4pixel;
   type long5pixel_array is array(natural range <>) of long5pixel;
   type long6pixel_array is array(natural range <>) of long6pixel;
    type pixel_matrix is array (coordinate range <>, coordinate range <>) of pixel;
   type pixellong_matrix is array (coordinate range <>, coordinate range <>) of longpixel;
    type pixel_matrix_ptr is access pixel_matrix;
end common ;
```

Netography & Bibliography

[1] Roth, H.C., 2004. Circuit Design with VHDL. Cambridge, MA: MIT Press., .

[2] Gonzalez, R.C. and Woods, R.E. 2003. Digital Image Processing., .

[3] Wolf, W. 2004. FPGA-Based System Design. Englewood Cliffs, NJ: Prentice-Hall., .

[4] Baese, U. M. 2007. Digital Signal Processing With Field Programmable Gate Arrays, Springer, 2007., .

[5] Haque, M. N. 2010. Implementation of a FPGA based Architecture of Prewitt Edge detection Algorithm using Verilog HDL. In Proceedings of Conference on Electronic and Telecommunication., .