Project Report

Sobel Edge Detection Implementation on VHDL

Course: **Automatic Design of Digital Circuits (FPGA)**

Professor: **Dr. Soheila Nazari**

Creator: **Milad Rabiei**

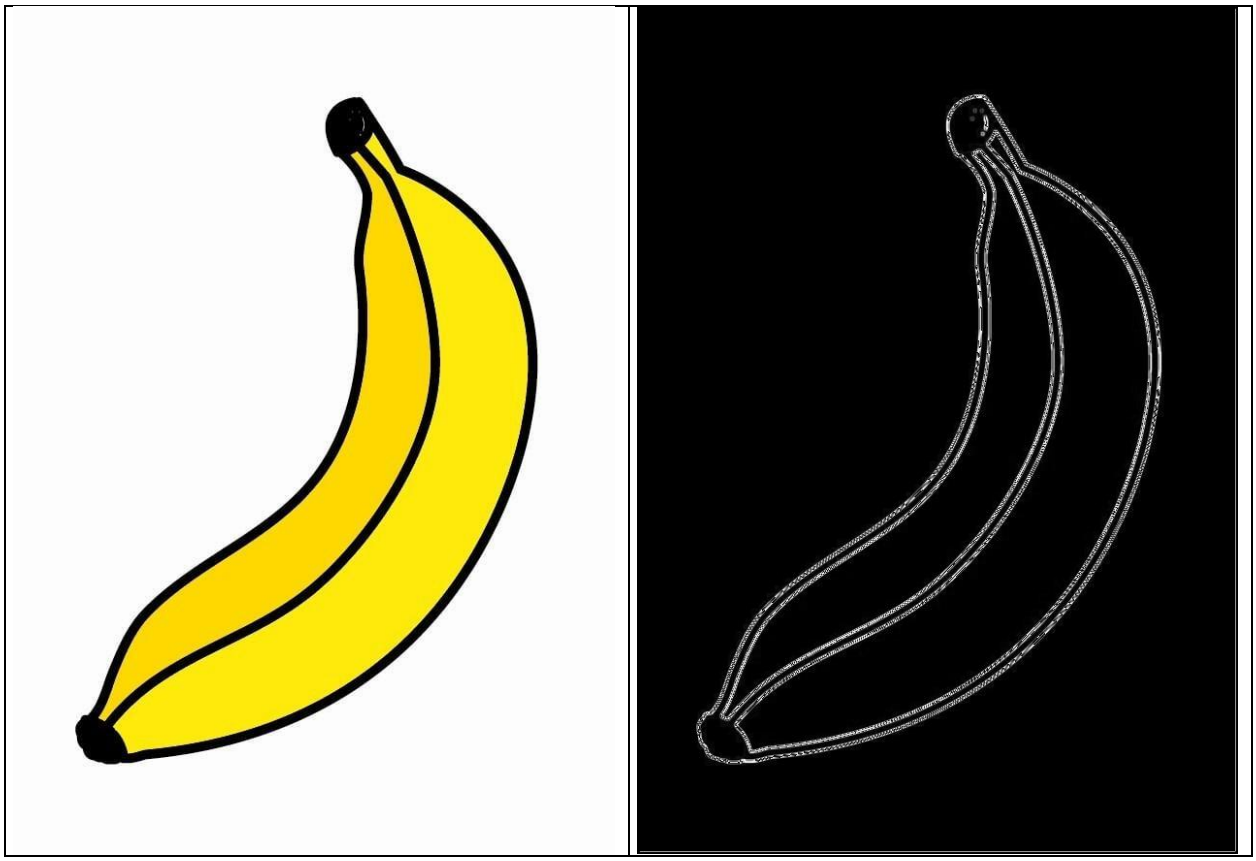
Student ID: **98242066**

Student of Electrical Engineering at Shahid Beheshti University, Tehran

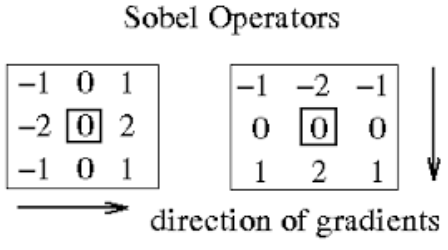
Date: **Spring 2023 (1402)**

عنوان پروژه : Sobel edge detector algorithm implementation in VHDL for FPGA

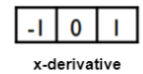
این پروژه به کمک نرم افزار ISE و Python (برای پردازش های فایل) انجام شده است. کد ها و فایل های شبیه سازی در پوشه همین گزارش موجود است.



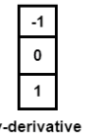
در این پروژه با اعمال فیلتر Sobel روی یک تصـویر، الگوریتم لبه یابی را شـبیه سـازی میکنیم. فیلتر های مختلفی برای عملیات لبه یابی ارائه شـده اند، مانند Sobel و Scharr. با انجام عملیات کانولوشـن این فیلتر ها روی تصـاویر، میتوان عملیات مشتق که برای سیگنال های پیوسته کاربرد دارد را تقریب زد و بر اساس خود این فیلتر ها، این تقریب میتواند دقیق تر شود. (Theory)



$$f(x,y)=\frac{f(x+h,y)-f(x-h,y)}{2h}$$



$$f(x,y)=\frac{f(x,y+h)-f(x,y-h)}{2h}$$



بر اساس سایز تصاویر و کاربرد، فیلتر های Sobel در سایز های مختلفی ارائه شده اند. در این پروژه فیلتر های با سایز های 3x3 و 5x5 و با اســتاندارد Intel IPP Library اعمال شــده اند. فیلتر بزرگتر همســایگی بزرگتری را در مرحله محاسبات در نظر میگیرد، که باعث افزایش قدرت الگریتم (robustness) و افزایش محاسبات (computational cost) خواهد شد. همچنین فیلتر کوچکتر میتواند باعث در نظر نگرفتن لبه های جزئی تر شود.

در این پروژه با تغییر ورودی kernel (bit) از 0 (3x3) به 1 (5x5)، اندازه این فیلتر را تغییر میدهیم.

$$G_x = \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \\ 4 & 8 & 0 & -8 & -4 \\ 6 & 12 & 0 & -12 & -6 \\ 4 & 8 & 0 & -8 & -4 \\ 1 & 2 & 0 & -2 & -1 \end{bmatrix} * A \quad \text{and} \quad G_y = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 2 & 8 & 12 & 8 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & -8 & -12 & -8 & -2 \\ -1 & -4 & -6 & -4 & -1 \end{bmatrix} * A$$

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

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

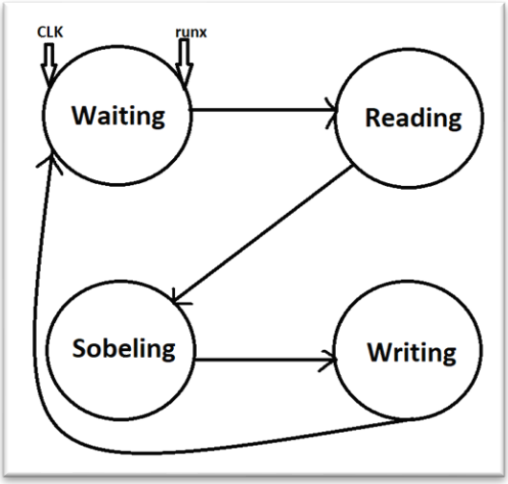
پیاده سازی:

```
entity fileread is
        generic (width    : integer := 620; height   : integer := 875);
        port (clk         :        in               std_logic;
                runx      :        in               std_ulogic       := '0';
                kernel    :        in               bit              := '0');
end fileread;
```

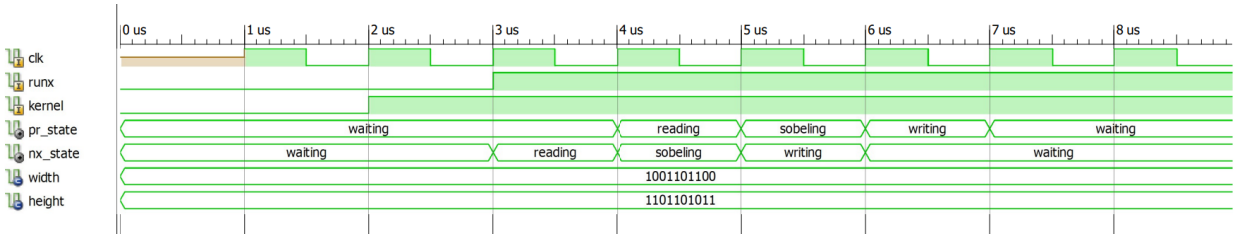
ابتدا به کمک یک کد Py و OpenCV، فایل تصویر مورد نظر را خوانده، ابعاد آن را کم میکنیم و مقادیر پیکسل ها را به صورت سریالی در یک فایل text ذخیره میکنیم.

```
import cv2
import numpy as np
img = cv2.imread("image.jpg")
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
pxs = np.reshape(img, (-1))
with open("pixels.txt", 'wt') as f:
    for px in pxs:
        f.writelines(str(px)+'\n')
f.close()
```

برای این پروسه، یک Finite State Machine طراحی شده است. در حالت عادی مدل در وضعیت waiting قرار دارد. با تغییر ورودی runx از 0 به 1، عملیات اصلی آغاز میشود. ابتدا فایل حاوی اطلاعات تصویر خوانده میشود. سپس این اطلاعات پردازش شده و لبه یابی اعمال میشود. نهایتا با ذخیره اطلاعات خروجی، مدل مجددا به وضعیت ابتدایی خود بازمیگردد.



در این پیاده سازی، next state با هر لبه بالارونده clock، به present state منتقل میشود. با تغییر present state یا ورودی runx (در sensitivity list)، سیگنال present state بررسی شده (case-when) و عملیات مد نظر انجام میشود.



با شروع عملیات و تبدیل وضعیت به Reading، خروجی مقادیر پیکسل ها را به کمک پکیج textio خوانده و در یک آرایه دو بعدی نگهداری میکنیم. اندازه تصویر به صورت مقادیر وارد شده به ماژول مشخص شده و به صورت generic تعریف شده اند.

از آنجا که بر اساس سایز تصویر میتوان الگویی (برای کم کردن پردازش ها) یافت، تبدیل مقادیر پیکسل ها از وضعیت سریالی به دوبعدی، عملیات الزامی نخواهد بود; در اینجا به علت اینکه صرفا شبیه سازی است، پیاده سازی به این صورت بوده است.

```
when     reading  =>
file_open(inp_buf, "pixels.txt", read_mode);
for i in 0 to height-1 loop
         for j in 0 to width-1 loop
                  readline(inp_buf, read_col_from_inp_buf);
                  read(read_col_from_inp_buf, val_col1);
                  arrayed (i, j) := val_col1;
         end loop;
end loop;
file_close(inp_buf);
nx_state <= sobeling;
```

با گذر زمان و در rising_edge(clock) بعدی، وضـــعیت از Reading به Sobel تغییر میکند. در این مرحله عملیات کانولوشـن فیلتر دوبعدی بر روی آرایه تصـویر دوبعدی انجام میشـود; به این صـورت، فیلتر از سـمت بالا-چپ تصـویر (موقعیت x=0, y=0) به سـمت راسـت و پایین به صـورت یکپارچه حرکت کرده و ضـرب-جمع را برای کانولوشـن انجام میدهد. در این قسمت همچنین، فیلتر 3x3 و 5x5 میتوانند پیاده شوند (بر اساس ورودی (kernel (bit).

```
when     sobeling =>
if kernel = '0' then
         kernel_select := 3;
else
         kernel_select := 5;
end if;
for i in kernel_select-2 to height-kernel_select+2 loop  حلقه برای حرکت عمودی
         for j in kernel_select-2 to width-kernel_select+2 loop  حلقه برای حرکت افقی
                  xval := 0;
                  yval := 0;
                  for m in 0 to kernel_select-1 loop
                           for n in 0 to kernel_select-1 loop  حلقه های ضرب فیلتر در مقادیر پیکسل ها
                                    if kernel = '0' then
                                             xval := xval + horizn3(m,n) * arrayed(i+m-1, j+n-1);
                                    else
                                             xval := xval + horizn5(m,n) * arrayed(i+m-1, j+n-1);
                                    end if;
                           end loop;
                  end loop;
                  for m in 0 to kernel_select-1 loop
                           for n in 0 to kernel_select-1 loop
                                    if kernel = '0' then
                                             yval := yval + vertic3(m,n) * arrayed(i+m-1, j+n-1);
                                    else
                                             yval := yval + vertic5(m,n) * arrayed(i+m-1, j+n-1);
                                    end if;
                           end loop;
                  end loop;
                  maked (i, j) := (abs(xval) + abs(yval)) / (kernel_select * kernel_select);
         end loop;
end loop;
nx_state <= writing;
```

پس از انجام عملیات اصلی الگریتم، وضعیت به Writing تبدیل شده و خروجی (آرایه دوبعدی عکس لبه یابی شده) به صورت سریالی مجدد در یک فایل txt ذخیره میشود. پس از آن این خروجی به کمک یک قطعه کد Python و OpenCV، خوانده شده، نمایش داده و ذخیره میشود.

```
file_open(out_buf, "resultpixels.txt", write_mode);
for i_y in 0 to height-1 loop
        for i_x in 0 to width-1 loop
                write(write_col_to_out_buf, maked (i_y, i_x)); نوشتن مقدار پیکسل روی بافر لاین
                writeline(out_buf, write_col_to_out_buf); نوشتن مقدار بافر لاین در فایل
        end loop;
end loop;
file_close(out_buf);
nx_state <= waiting;
```

نهایتا، با یک قطعه کد Py، این خروجی فایل text را به تصویر دوبعدی تبدیل میکنیم، نمایش داده و ذخیره میکنیم.

```
import cv2
import numpy as np
l = []
with open("resultpixels.txt", 'rt') as f:
    for line in f.readlines():
        l.append(int(abs(int(line))))
f.close()
img_array = np.array(l, dtype=np.uint8)
img_array = np.reshape(img_array, (875, 620))
cv2.imshow('ny', img_array)
cv2.waitKey()
cv2.destroyAllWindows()
cv2.imwrite("results.jpg", img_array)
```