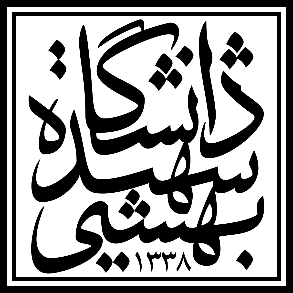
**Project Report**

**Sobel Edge Detection Implementation on VHDL**



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

Professor: **Dr. Soheila Nazari**

Creator: **Milad Rabiei**

Student ID: **98242066**

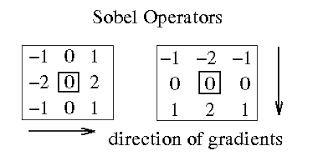
Student of Electrical Engineering at Shahid Beheshti University, Tehran

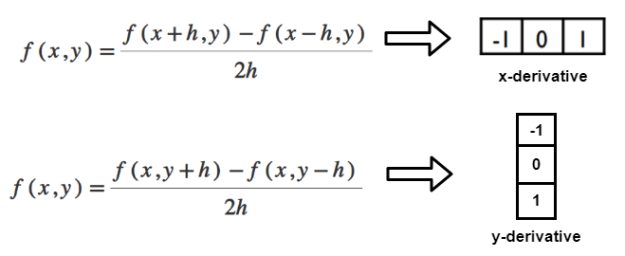
Date: **Spring 2023 (1402)**

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

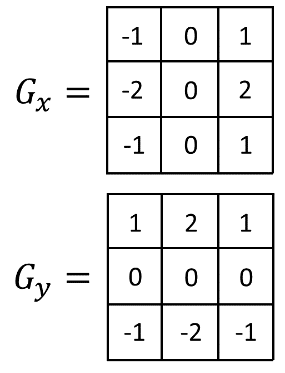
این پروژه به کمک نرم افزار ISE و Python (برای پردازش های فایل) انجام شده است. کد ها و فایل های شبیه سازی در پوشه همین گزارش موجود است.

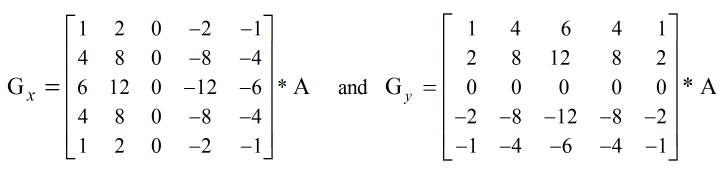
|  |  |
| --- | --- |
|  |  |

در این پروژه با اعمال فیلتر Sobel روی یک تصویر، الگوریتم لبه یابی را شبیه سازی میکنیم. فیلتر های مختلفی برای عملیات لبه یابی ارائه شده اند، مانند Sobel و Scharr. با انجام عملیات کانولوشن این فیلتر ها روی تصاویر، 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](https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm))



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 استاندارد Intel IPP Library اعمال شده اند. فیلتر بزرگتر همسایگی 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.



**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))

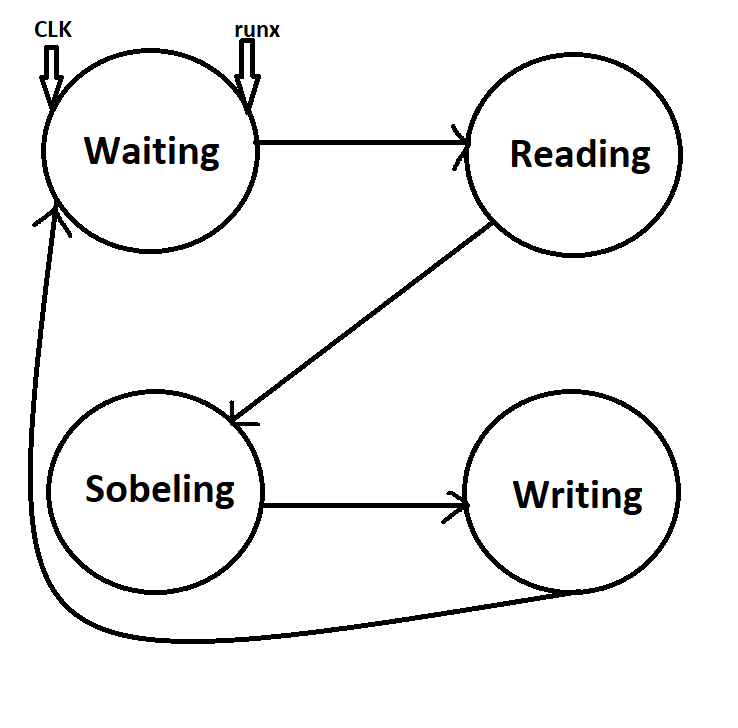
with open("pixels.txt", 'wt') as f:

for px in pxs:

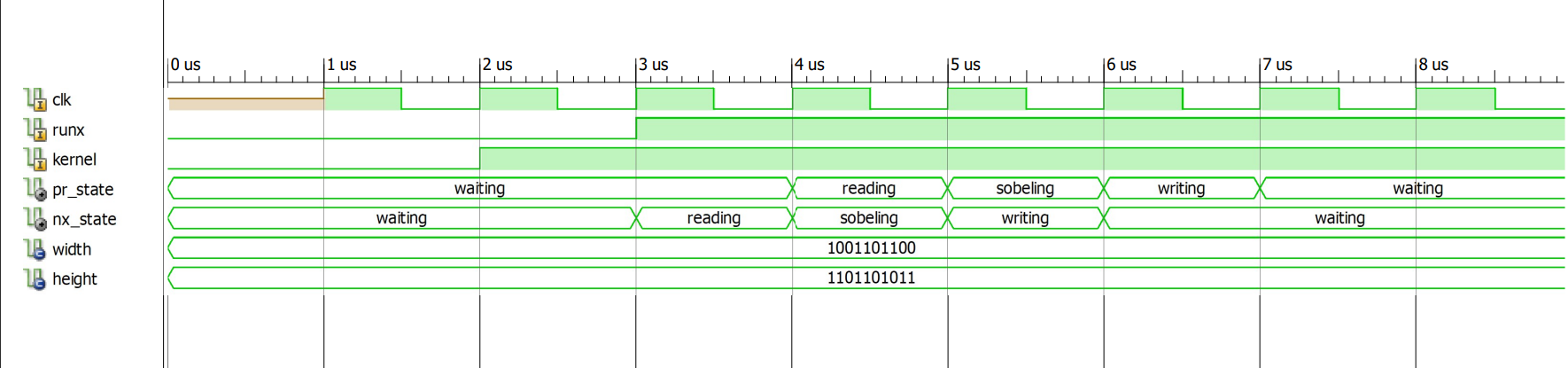
f.writelines(str(px)+'\n')

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، وضعیت از Reading به Sobel تغییر میکند. در این مرحله عملیات کانولوشن فیلتر دوبعدی بر روی آرایه تصویر دوبعدی انجام میشود; به این صورت، فیلتر از سمت بالا-چپ تصویر (موقعیت x=0, y=0) به سمت راست و پایین به صورت یکپارچه حرکت کرده و ضرب-جمع را برای کانولوشن 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)