FPGA

Edge detection using the Sobel filter

گزارش کار پروژه نهایی درس fpga

استاد درس: دکتر سهیلا نظری

نام ونام خانوادگی: مرضیه غیور نجف آبادی

شماره دانشجویی: 98242112

- فیلتر Sobel:

برای تشخیص گوشه های تصویر و نواحی با تغییر رنگ شدید از سفید به مشکی استفاده میشود.
نحوه کار این فیلتر در زیر توضیح داده شده است:

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

X kernel

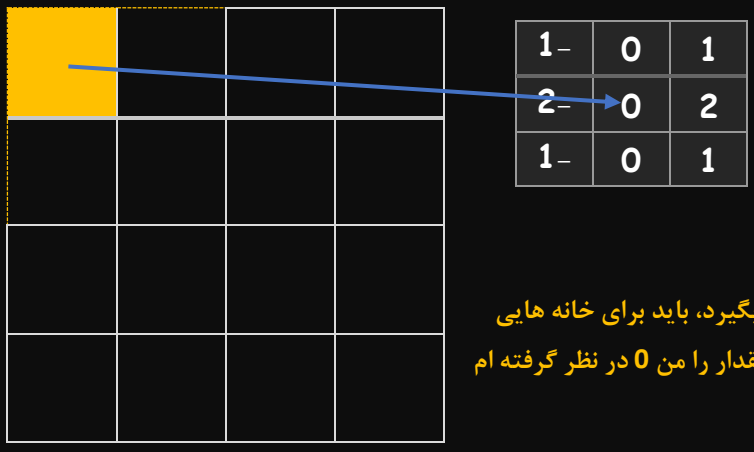
1-	-2	1-
0	0	0
1	2	1

Y kernel

دو مربع بالا kernel هایی هستند که برای پیاده سازی فیلتر سوبل استفاده میشوند. نحوه کار این فیلتر به این شکل است که برای هر داده pixel یک عکس gray scale که بین 0 تا 255 می باشد، این فیلتر روی آن قرار گرفته و مربع قرمز در هر کرنل نشان دهنده pixel مورد نظر است. برای اینکه مشخص شود شدت تغییر رنگ در نواحی این pixel به چه شکل است، سمت چپ این pixel با اعداد منفی و سمت راستش با اعداد مثبت ضرب شده، و جمع همه این اعداد داخل pixel مورد نظر ذخیره میشود.

و به این ترتیب بر روی کل pixel های یک عکس میتوان این فیلتر را اعمال کرد.

برای pixel هایی که در گوشه قرار دارند باید این درنظر گرفته شود که چه مقداری باید با این اعداد جمع شوند؟



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

برای مثال اگر این فیلتر بر روی خانه زرد رنگ قرار بگیرد، باید برای خانه هایی که وجود ندارند مقدار پیش فرضی تصور کرد، این مقدار را من 0 در نظر گرفته ام اما بعضا مقدار خود pixel هم درنظر میگیرند.

- **پیاده سازی فیلتر:**

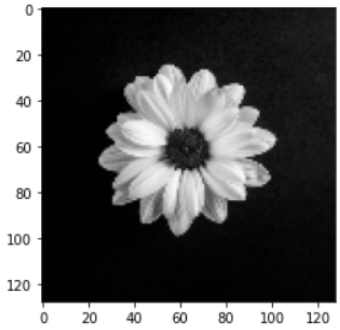
برای پیاده سازی این فیلتر ابتدا احتیاج داریم توسط python یا MATLAB یا روش های دیگر، تصویر را به gray scale تبدیل کنیم و همچنین مقادیر pixel ها را در یک فایل txt. ذخیر کنیم.

من از زبان برنامه نویسی python برای اینکار استفاده کردم که مراحل در زیر توضیح داده شده است.

- **Python Code**: فایل (Picture to txt and visa versa.ipynb):

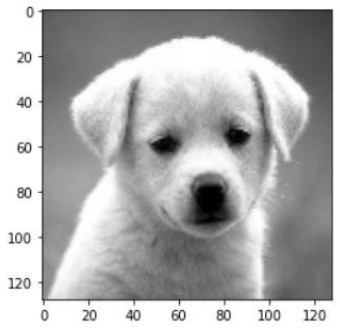
```
my_image = cv.imread('Picture1.jpg')
resized_image = cv.resize(my_image, (128, 128))
gray_image = cv.cvtColor(resized_image, cv.COLOR_BGR2GRAY)
print(gray_image.shape)
plt.imshow(gray_image, cmap="gray")
```

```
(128, 128)
<matplotlib.image.AxesImage at 0x1af24f383d0>
```



```
my_image2 = cv.imread('Picture2.jpg')
resized_image2 = cv.resize(my_image2, (128, 128))
gray_image2 = cv.cvtColor(resized_image2, cv.COLOR_BGR2GRAY)
print(gray_image2.shape)
plt.imshow(gray_image2, cmap="gray")
```

```
(128, 128)
<matplotlib.image.AxesImage at 0x1af24d65400>
```



با استفاده از cv2 تصاویر را gray کرده، به ابعاد مطلوب resize کرده و تصاویر خام را نمایش داده ام.

```
print(gray_image)
print(gray_image.shape)

with open('values1.txt', 'w') as f:
    for j in range(128):
        for i in range(128):
            f.write(gray_image[j,i].astype(str)+'\n')
```

```
[[ 1  1  1 ... 28 25 31]
 [ 1  1  1 ... 27 29 28]
 [ 1  1  1 ... 27 28 28]
 ...
 [ 1  1  1 ...  4  4  4]
 [ 1  1  1 ...  3  4  3]
 [ 1  1  1 ...  7  4  2]]
(128, 128)
```

```
print(gray_image2)
print(gray_image2.shape)

with open('values2.txt', 'w') as f:
    for j in range(128):
        for i in range(128):
            f.write(gray_image2[j,i].astype(str)+'\n')
```

```
[[ 89  88  90 ...  62  61  61]
 [ 90  90  92 ...  61  61  61]
 [ 92  92  93 ...  63  63  62]
 ...
 [145 142 151 ... 107 104 102]
 [144 145 164 ... 107 104 103]
 [144 155 185 ... 107 104 103]]
(128, 128)
```

سپس مقادیر integer را در فایل txt. به صورت سریالی ذخیره میکنیم.

فایل های "Values1" و "Values2".

VHDL Code :-

1. کتابخانه ها:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use std.textio.all;
```

از کتابخانه ieee و std.textio استفاده کرده ام و package های ieee برای کد اصلی VHDL و استفاده از یکسری از تایپ ها اد شده اند.
Std.textio برای کار با فایل text و read و write کردن فایل استفاده میشود. که جلوتر دستورات آن را جلوتر خواهیم دید.

2. Entity:

```
entity SobelFilter is
      generic(PicSize : integer := 128);
      port(
            clk     : in std_logic;
            rst     : in std_logic
            );
end SobelFilter;
```

یک generic استفاده شده است برای مشخص کردن ابعاد تصویر ورودی. همچنین این کد فقط دو پورت ورودی clock و reset دارد.

3. Architecture:

```
architecture arch of SobelFilter is
type Matrix is array(0 to PicSize+1, Picsize+1 downto 0) of integer;
type state is (read_file, calculate_kernel, sum_kernels, filter_pic, write_file);
signal pr_state, nx_state : state := read_file;
signal Picture, Xkernel, Ykernel, filtered_pic : Matrix;
--signal x_c, y_c : integer;
file inputpic : text;
file outputpic : text;
```

در این بخش قبل از دستور begin سیگنال های مورد نیاز تعریف شده اند. با توجه به اینکه این کد با fsm پیاده سازی شده است به type state احتیاج داشتم که آن را تعریف کرده و همه state های موجود در state machine در آن قرار داده شده اند. همچنین یک آرایه 2D برای ذخیره اطلاعات تصویر درست شده است و 4 تا از این آرایه در کد وجود دارد که Picture برای ذخیره اعداد خام تصویر است، Xkernel , Ykernel برای ذخیره اعداد مربوط به فیلتر برای هر pixel هستند و filtered_pic برای ذخیره اعداد نهایی فیلتر سوبل است.

4. Lower Section:

```
begin
lower:      process(clk)--rst
            begin
               if(rst = '1') then
                  pr_state <= read_file;
               elsif(rising_edge(clk)) then
                  pr_state <= nx_state;
               end if;
            end process lower;
```

برای پیاده سازی fsm احتیاج است که در ابتدا یک process حساس به clk و در صورت تمایل به وجود آپشن ریست، حساس به clk, rst قرار داد. در این process اگر ریست 1 شده باشد، برنامه به state اولیه اش برمیگردد و در غیر این صورت اگر clk لبه بالارونده خورده باشد، present state مقدار state بعدی که next state باشد را به خود میگیرد.

5. Upper Section:

در این بخش کل محاسبات مربوط به next state انجام میشود. در ابتدا باید variable های مورد نیاز برای process مربوطه را قبل از begin تعریف کنیم. که 4 variable، دوتا از جنس line برای خواندن فایل به صورت خط به خط و 2 تا از جنس integer برای ذخیره داده های هر خانه از فایل تعریف شده اند.

```
upper:      process(pr_state)

            --file specifications
            variable pixeldata,pixeldata2 : integer;
            variable txtline,txtline2 : line;
```

```
begin
   case pr_state is
   --read image from txt file
      when read_file =>
         --file_open(inputpic,"values1.txt",read_mode); --flower picture(pic1)
         file_open(inputpic,"values2.txt",read_mode); --dog picture(pic2)

         for y_c in 0 to PicSize+1 loop
            for x_c in 0 to PicSize+1 loop
               if(y_c = 0 or y_c = PicSize+1 or x_c = 0 or x_c = PicSize+1) then
                  Picture(y_c,x_c) <= 0;
               else
                  readline(inputpic, txtline);
                  read(txtline, pixeldata);
                  Picture(y_c,x_c) <= pixeldata;
               end if;
            end loop;
         end loop;
         file_close(inputpic);
         nx_state <= calculate_kernel;
```

بعد از دستور begin یک case-when قرار داده شده است که همه حالات present state را در برمیگیرد و با توجه به آن محاسبات مربوط به next state را انجام میدهد.

5.1. read_file:

در این بخش فایل txt. تولید شده (با استفاده از پایتون) در حالت read mode باز میشود و در دو لوپ همه داده های آن در آرایه‌ی Picture ذخیره میشوند. آرایه های از جنس matrix که در قسمت قبل از begin مربوط به Architecture تعریف شده اند، از ابعاد تصویر 2 سطر و 2 ستون بیشتر بزرگتر اند به این دلیل که بتوان فیلتر سوبل را بر روی خانه های گوشه‌ی تصویر هم قرار داد و این مقادیر بعدا حذف میشوند.

فایل های values 1و2 هرکدام برای یکی از تصاویر موجود در فایل ها نوشته شده اند که برای دیدن نتیجه‌ی هر کدام که میخواهیم باید خط مربوط به همان تصویر را uncomment کنیم.

```
--find Xkernel and Ykernel of each pixel
when calculate_kernel =>
   for y_c in 1 to PicSize loop
      for x_c in 1 to PicSize loop
         --Xkernel
         Xkernel(y_c,x_c) <= (-1*Picture(y_c-1,x_c-1)
          -2*Picture(y_c,x_c-1)
          -1*Picture(y_c+1,x_c-1)
          +1*Picture(y_c-1,x_c+1)
          +2*Picture(y_c,x_c+1)
          +1*Picture(y_c+1,x_c+1));
      end loop;
   end loop;
   for y_c in 1 to PicSize loop
      for x_c in 1 to PicSize loop
         --Ykernel
         Ykernel(y_c,x_c) <= (-1*Picture(y_c-1,x_c-1)
          -2*Picture(y_c-1,x_c)
          -1*Picture(y_c-1,x_c+1)
          +1*Picture(y_c+1,x_c-1)
          +2*Picture(y_c+1,x_c)
          +1*Picture(y_c+1,x_c+1));
      end loop;
   end loop;
   nx_state <= sum_kernels;
```

5.2. Calculate_kernel:

در این بخش 2 آرایه‌ی مربوط به کرنل برای هر pixel با توجه به 8 pixel اطرافش، محاسبه و ذخیره میشوند.

```
--sum them up
when sum_kernels =>
   for y_c in 1 to PicSize loop
      for x_c in 1 to PicSize loop
         filtered_pic(y_c,x_c) <= abs(Xkernel(y_c,x_c)) + abs(Ykernel(y_c,x_c));
      end loop;
   end loop;
   nx_state <= filter_pic;
```

5.3. Sum_kernels:

در این بخش، 2 تا آرایه‌ی کرنل ساخته شده با هم جمع شده و اعداد فیلتر شده توسط سوبل تولید میشوند و داخل آرایه‌ی filtered_pic ذخیره میشوند.

```
when filter_pic =>
   for y_c in 1 to PicSize loop
      for x_c in 1 to PicSize loop
         if(filtered_pic(y_c,x_c) >= 254) then
            filtered_pic(y_c,x_c) <= 255;
         elsif(filtered_pic(y_c,x_c) <= 50) then
            filtered_pic(y_c,x_c) <= 0;
         end if;
      end loop;
   end loop;
   nx_state <= write_file;
```

5.4. Filter_pic:

برای اینکه به فیلتر مطلوب تری برسیم میتوانیم یک فیلتر ساده بر روی pixel ها پیاده سازی کنیم به شکلی که اگر عدد pixel ای از مقداری کوچکتر بود آن را 0 و اگر از مقداری بزرگتر بود آن را 255 در نظر بگیرد.

```
      --write file
      when write_file =>
         --file_open(outputpic, "write1.txt", write_mode);  --flower picture(pic1)
         file_open(outputpic,"write2.txt",write_mode); --dog picture(pic2)

         for y_c in 1 to PicSize loop
            for x_c in 1 to PicSize loop
               pixeldata2 := filtered_pic(y_c,x_c);
               write(txtline2,pixeldata2);
               writeline(outputpic,txtline2);
            end loop;
         end loop;
         file_close(outputpic);
         nx_state <= read_file;

      end case;
end process upper;
```

5.5. Write_file:

در این حالت که state نهایی می باشد،

باید فایل txt. خالی ای که می خواهیم در آن اعداد تولید شده

را ذخیره کنیم را در حالت write mode باز کرده و تک به تک اعداد را در خط های مختلف آن به صورت سریالی ذخیره میکنیم.

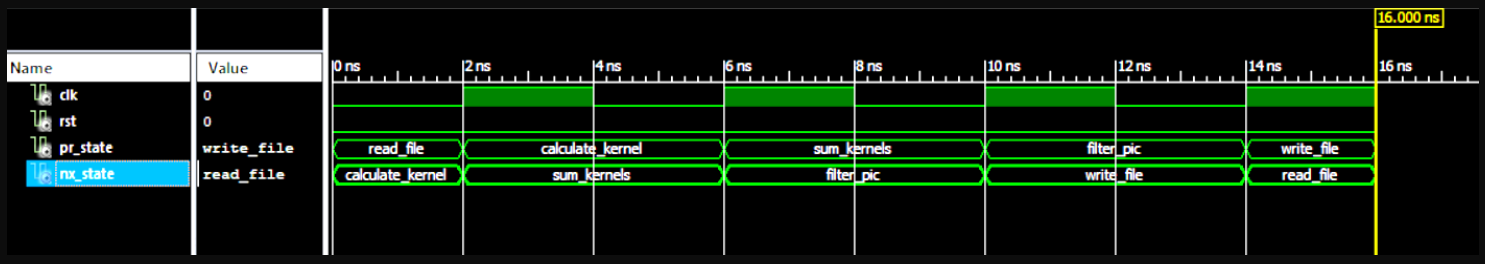
فایل های write 1و2 هرکدام برای یکی از تصاویر موجود در فایل ها نوشته شده اند که برای دیدن نتیجه‌ی هر کدام که میخواهیم باید خط مربوط به همان تصویر را uncomment کنیم.

6. Test bench:

یک تست بنچ برای مقدار دهی به generic و clk و مشاهده‌ی نتیجه شبیه سازی مینویسیم.

```
entity tb_SobelEdgeDitection is
end tb_SobelEdgeDitection;

architecture Behavioral of tb_SobelEdgeDitection is
   component SobelFilter is
      generic(PicSize : integer := 128);
      port(
            clk   : in std_logic;
            rst   : in std_logic
            );
   end component;
   --inputs
   signal clk : std_logic := '0';
   signal rst : std_logic := '0';

   --outputs

begin
   -- Stimulus process
   stim_proc: process
      begin
            wait for 2ns;
            clk <= '1';
            wait for 2ns;
            clk <= '0';
      end process stim_proc;

   uut: SobelFilter generic map (128) port map(
      clk => clk,
      rst => rst
      );
end Behavioral;
```

* **نتیجه شبیه سازی در isim:** (در یک فایل pdf هم در ضمیمه report آورده شده است)



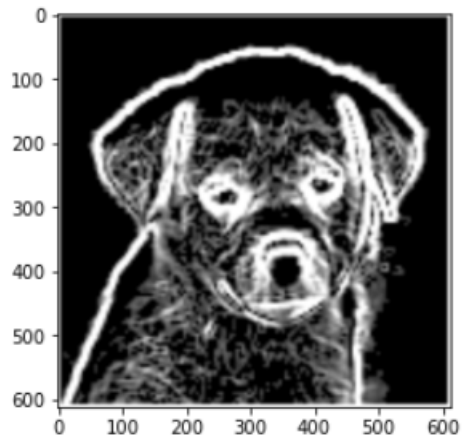
Python Code: فایل (Picture to txt and visa versa. ipynb):

مجددا با استفاده از دستورات پایتون و cv2 فایل های txt. را به تصویر jpg. تبدیل میکنیم و نتایج را در زیر مشاهده میکنیم

* فایل های نتایج نیز در پوشه **Report** قرار داده شده اند.

```
A = []
with open("write2.txt", 'rt') as f:
    for line in f.readlines():
        # print(line)
        A.append(int(line))
f.close()

print(len(A))
img_array2 = np.array(A, dtype=np.uint8)

img_array2 = np.reshape(img_array2, (128,128))
new_image2 = cv.resize(img_array2, (612, 612))
#cv.imshow('ny', new_image2)
plt.imshow(new_image2, cmap="gray")
cv.waitKey()
cv.destroyAllWindows()


cv.imwrite('Edge_picture2.jpg', new_image2)
```

```
16384

True
```




```
A = []
with open("write1.txt", 'rt') as f:
    for line in f.readlines():
        # print(line)
        A.append(int(line))
f.close()

print(len(A))
img_array = np.array(A, dtype=np.uint8)

img_array = np.reshape(img_array, (128,128))
new_image = cv.resize(img_array, (612, 612))
#cv.imshow('ny', new_image)
plt.imshow(new_image, cmap="gray")
cv.waitKey()
cv.destroyAllWindows()

cv.imwrite('Edge_picture1.jpg', new_image)
```

```
16384

True
```