درس طراحی کامپیوتری سیستم‌های دیجیتال

گزارش شماره‌ی: 18

اعضای گروه: نرگس قاسمی ، تارا برقیان

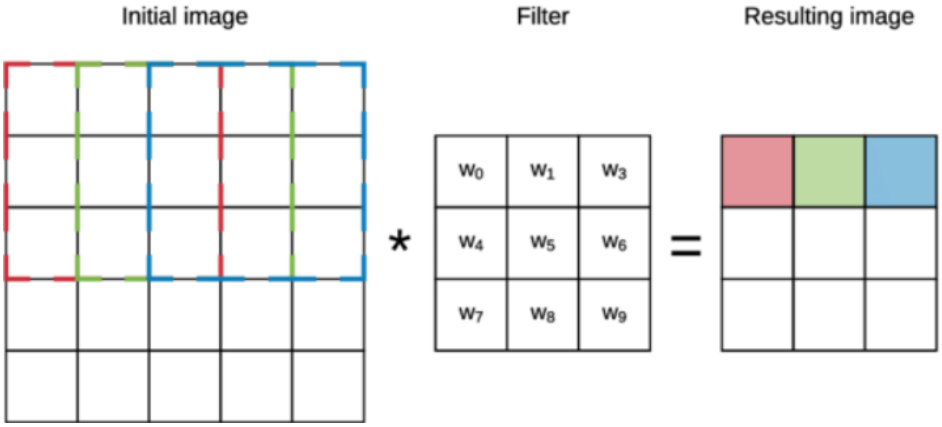
چکیده:

در این آزمایش، یک فیلتر شارپ را با کمک کانولوشن، به صورت concurrent بر روی یک تصویر سیاه سفید اعمال کنیم. برای این کار به کمک زبان پایتون یک ماتریس دو بعدی از تصویر استخراج کرده و با سینتکس های مربوط به اجرای همروند، یک کرنل sharp را روی آن اعمال کردیم. سپس خروجی را با کد مشابه ولی به زبان متلب مقایسه کردیم.

فهرست مطالب

1 : مختصری بر کانولوشن

در این تمرین با کمک کانولوشن فیلتر روی تصویر اعمال میشود به این صورت که ماتریس کرنل از بالا و چپ ترین حالت ممکن شروع به چرخش میکند و در هر بار ضرب شدن، یکی از خانه های ماتریس خروجی ساخته میشود. مانند تصویر :



نکته قابل توجه این است که تصویر خروجی حاصل، از تصویر ورودی کوچک تر است. برای هم اندازه کردن انها، مستطیل های حاشیه با مقادیر اصلی مقدار دهی میشوند که در ادامه به وضوح توضیح داده میشود.

2 : توضیحات فایل Convolution.vhd

1_2 : entity فایل convolution.vhd

```
entity convolution is
    generic (i_width  :integer ;
             i_height :integer  ;
             k_width  :integer  ;
             k_height :integer  );
    port (
        clock:in std_logic;
        img  : in vec2 (0 to i_height-1 , 0 to i_width-1);
        krnl : in vec2 (0 to 2 , 0 to 2 );
        new_img:out vec2 (0 to i_height-1 , 0 to i_width-1)

    );
end convolution;
```

طول و عرض ماتریس ورودی و کرنل را به صورت جنریک و از تایپ integer در نظر گرفتیم.

در قسمت ورودی کلاک، تصویر، کرنل و در خروجی ماتریس مربوط به تصویر بعدی را خواهیم داشت.

Vec2 نیز، یک user type است که یک ارایه دوبعدی از integer میباشد. (رجوع شود به قسمت پکیج کانولوشن.)

2_2 : architecture فایل convolution.vhd

```
SIGNAL n_i_width:integer  := i_width -(k_width-1);
SIGNAL  n_i_height:integer := i_height-(k_height-1);
SIGNAL sum :vec3 (0 to n_i_height , 0 to n_i_width ,0 to 9)      ;
```

دو سیگنال اول برای تعیین تعداد چرخش دستور FOR و سیگنال sum که یک ماتریس 3 بعدی است، به عنوان یک متغییر کمکی استفاده شده است. برای جلوگیری از وقوع خطای multiple driven این سیگنال را سه بعدی در نظر گرفتیم.

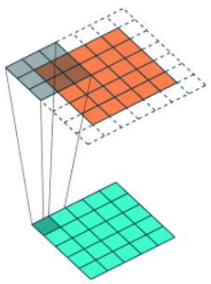
(در ادامه مفصل توضیح داده میشود.)

```
B1: BLOCK (clock'EVENT AND clock = '1' )
```

با کمک یک بلاکِ GUARDED، علاوه بر افزایش نظم و خوانایی کد، دستورات را به صورت موازی (کانکارنت) دراوردیم.

```
G1:for y in 1 to (i_height-2) generate

    new_img(y,0) <=GUARDED img(y,0)-1  WHEN (img(y,0)>0 ) ELSE 0 ;
    new_img(y,i_width-1) <=GUARDED img(y,i_width-1)-1  WHEN (img(y,i_width-1)>0 ) ELSE 0 ;

      end generate G1;

 G2:for x in 0 to (i_width-1) generate

     new_img(0,x) <=GUARDED img(0,x)-1  WHEN (img(0,x)>0 ) ELSE 0 ;
     new_img(i_height-1,x) <=GUARDED img(i_height-1,x)-1  WHEN (img(i_height-1,x)>0 ) ELSE 0 ;

       end generate G2;
```

همانطور که پیش تر اشاره شد (و در تصویر زیر میبینید) ماتریس خروجی حاصل از کانولوشن دو بعدی، از ماتریس ورودی کوچک تر است. برای رفع این مشکل، در ابتدا حاشیه های ماتریس خروجی را با کمک FOR های بالا مقدار دهی کردیم.



ماتریس نارنجی : ورودی

ماتریس سبز : خروجی

ماتریس طوسی : کرنل

```
G3:for y in 0 to (n_i_height-1) generate
   G4:for x in 0 to (n_i_width-1) generate
     sum(y,x,0) <=0;
      G5:for k_r in 0 to (k_height-1) generate
     G6:for k_c in 0 to (k_width-1) generate
        sum(y,x, k_r * 3 + k_c+1) <= GUARDED sum (y,x,k_r * 3 + k_c) + img((y+k_r),(x+k_c)) * krnl(k_r,k_c);

     end generate G6;
      end generate G5;

        new_img(y+1,x+1) <= GUARDED (sum(y,x,9)-1) WHEN sum(y,x,9)>0 ELSE 0 ;

      end generate G4;
       end generate G3;
```

در این بخش، به کمک FOR های تو در تو، حاصل ضرب کرنل بر روی تصویر را محاسبه کردیم و بعد از اتمام G5,G6 بر روی ماتریس خروجی حاصل را اضافه کردیم.

در این بخش با چالش های زیادی رو به رو شدیم. که مهم ترین انها خطای multiple driven بود. ما در ابتدا، کد را به صورت ترتیبی زده بودیم تا کلیت پروژه را متوجه شویم، سپس سعی کردیم همان کد را کانکارنت کنیم. در ابتدا سیگنال sum یک ارایه یک بعدی بود و در چرخش FOR ، خانه هایی یکسان از مقدار های متفاوت میگرفت و خطایی نداشت.

اما وقتی به صورت همزمان قرار است این دستورات اجرا شوند، یک سیگنال نمیتواند بیشتر از یکبار مقدار دهی شود.

ما ابتدا sum را به یک ارایه دو بعدی تبدیل کردیم اما همچنان همین خطا را داشتیم. با کمی بررسی و trace کد، متوجه شدیم وقتی دو حلقه ی اول G3 , G4 تغییر میکنند، sum از اول مقدار دهی میشود. پس برای جلوگیری از خطا، باید به ازای هر yوx متمایز، یک ارایه sum مقدار دهی شود. لذا یک ماتریس سه بعدی خواهیم داشت. نکته مهم دیگر قرار دهی درست اندیس ها بود تا خطای out of bound در ارایه مواجه نشویم.

پس از پایان کامل این 4 حلقه، ماتریس خروجی اماده است.

ConvolutionTB : 3

entity : 1_3 فایل convolutionTB.vhd

در این بخش مقدارهای اختیاری برای تست صحت کد قرار داده شده است.

```
entity convolutionTB is generic (
                i_width  :integer  := 5;
                i_height :integer  := 8;
                k_width  :integer  := 3;
                k_height :integer  := 3);
END convolutionTB;
```

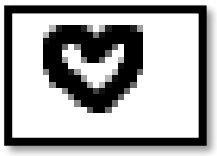
2_3: architecture فایل convolutionTB.vhd

```
begin
    cut : convolution GENERIC MAP  (
            i_width   => 5 ,
            i_height  => 8 ,
            k_width   => 3 ,
            k_height  => 3)
            PORT MAP (clock_tb,img_tb,krnl_tb,new_img_tb);

    clock_tb <= NOT clock_tb AFTER 10 ns;
krnl_tb <= (
           (0,1,0),
           (1,-4,1),
           (0,1,0));

 img_tb <=   (
          (2, 2, 1, 1, 1),
          (2, 2, 1, 1, 1),
          (1, 1, 1, 1, 1),
          (4, 2, 1, 1, 6),
          (2, 2, 1, 5, 1),
          (2, 2, 1, 1, 1),
          (1, 1, 1, 1, 1),
          (3, 3, 1, 1, 1));
END tb;
```

در قسمت krnl_tb ، فیلتر مورد نظر و در img_tb ماتریس ورودی دلخواه را قرار میدهیم. (نمونه ورودی های بیشتر برای تست در پیوست قرار دارد.)

4 : نحوه ی مقایسه خروجی ها با متلب و نتیجه گیری

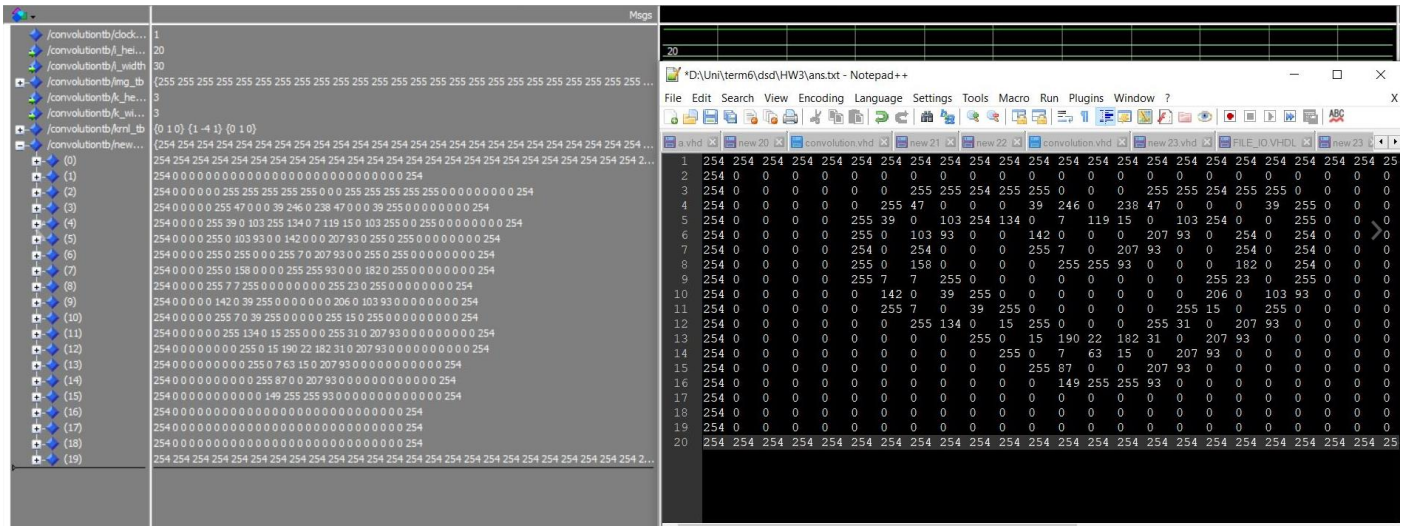
فایل ورودی را با متلب(یا پایتون) به صورت text در می‌اوریم و در ورودی تست بنچ قرار دادیم.خروجی کد vhdl و خروجی کد اصلی متلب به دست آمده را بایکدیگر مقایسه کردیم و یکسان بود.



تصویر ورودی 30x20

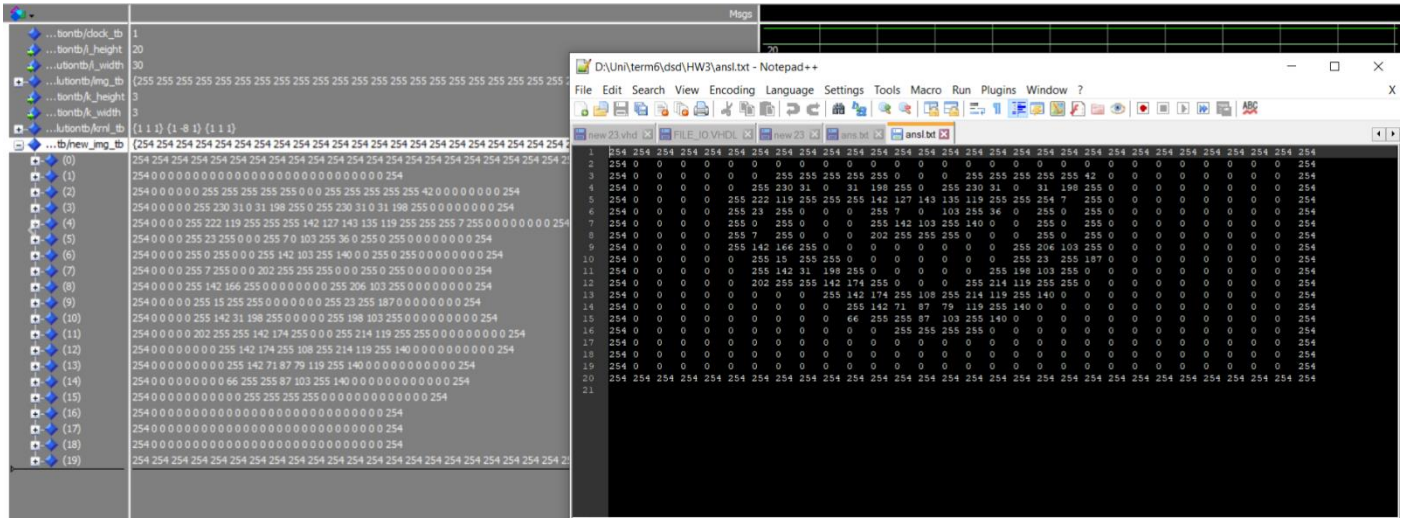
حاصل خروجی فیلتر اول :

$$H_h = \frac{1}{4}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

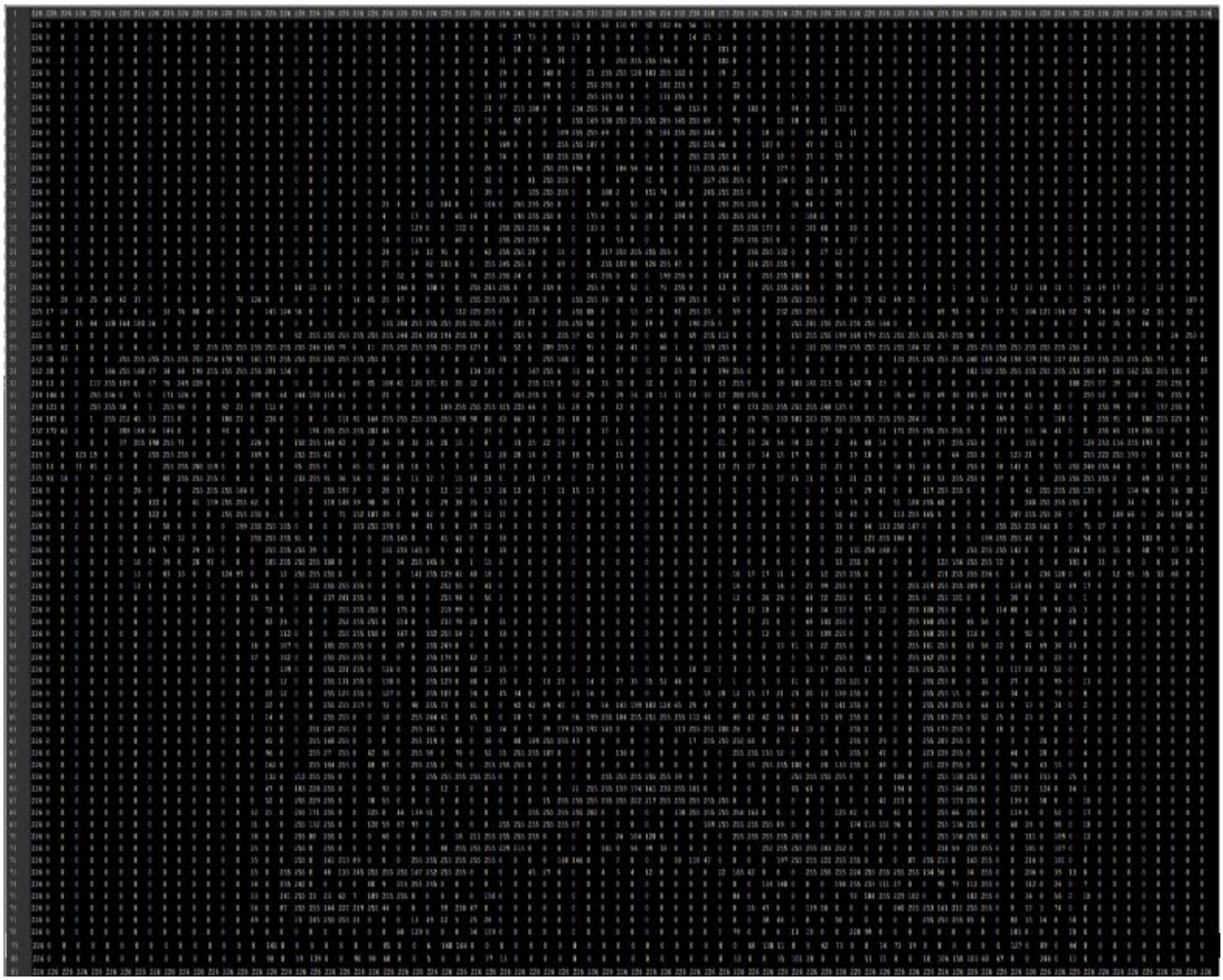


حاصل خروجی فیلتر دوم :

$$H_h = \frac{1}{8}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



یک نمونه دیگر :



5: منابع

- https://en.wikipedia.org/wiki/Kernel_(image_processing)
- https://www.codingame.com/playgrounds/2524/basic-image-manipulation/filtering