گزارش فاز اول پروژه نهایی FPGA

زبان برنامه نویسی Verilog و نرم افزار Xilinx

نسرین کریمی ۹۷۲۳۶۰۸۱ - nasrinkarimi7979@gmail.com

دانشکده مهندسی برق شهید عباسپور – دانشگاه شهید بهشتی

مقدمه

در این تمرین با استفاده از وریلاگ یک ماژول پردازش تصویر برای استخراج لبه های یک تصویر طراحی می نماییم. بدین منظور از فیلتر سوبل (Sobel) استفاده می نماییم. فیلتر سوبل در واقع یک عملیات ماتریسی است که بر روی تصویر (که خود یک ماتریس از پیکسلها است) اجرا می شود. در این فیلتر، پیکسل واقع در سطر m ام و ستون n ام از خروجی، به صورت زیر تعریف میشود:

$$y(m.n) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} x(m-i.n-j)h(i.j)$$

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

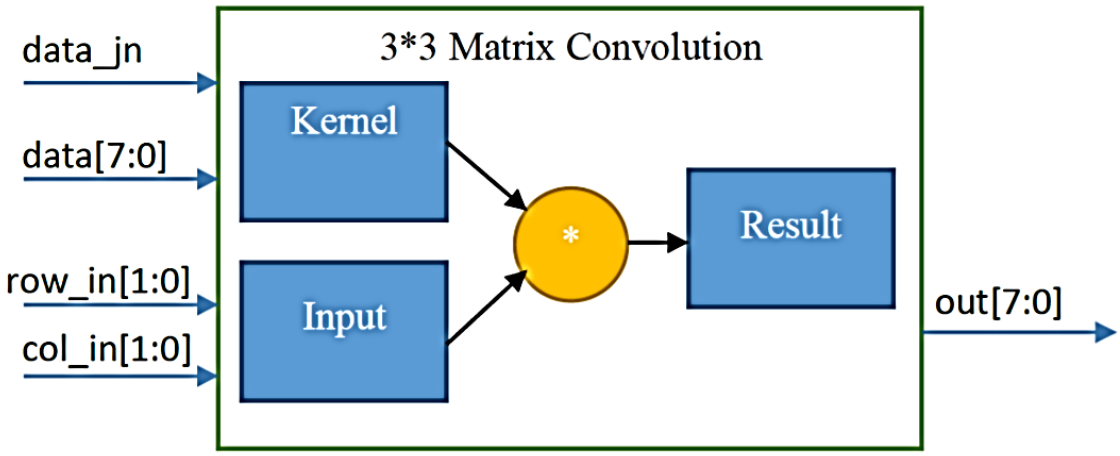
در رابطه بالا، x ماتریس ورودی (تصویر ورودی) با ابعاد M*N و h اصطلاحاً kernel نامیده می شود (البته کرنل مربوط به فیلتر سوبل شامل یک مولفه دیگر نیز هست که برای سادگی در این پروژه از آن صرف نظر میکنیم). همچنین در اینجا منظور از سطر و ستون (0,0) سطر و ستون وسط ماتریس h هستند. همچنین، از آنجا که ماتریس x سطر و ستون منفی ندارد، فرض می کنیم تمام درایه های سطر و ستون -1 از این ماتریس صفر هستند. همچنین، فرض می کنیم تمام درایه های سطر M+1 و ستون N+1 از x نیز صفر هستند تا مشکلی در کرانه های حاصلجمع های بالا به وجود نیاید (مثلا برای محاسبه y(0,0) نیاز به x(-1,-1) داریم که باید آنرا صفر بگیریم. یا برای محاسبه y(M,N) نیاز به مقدار x(M+1, N+1) داریم که باید آنها را صفر در نظر بگیریم.)

در این آزمایش قصد داریم ابتدا ماژولی طراحی کنیم که حاصل جمع و ضرب های بالا (کانولوشن) را محاسبه نماید. سپس با استفاده از کنترلر و حافظه، این فیلتر را روی یک تصویر اعمال نماید.

ماژول محاسبه کانولوشن

این ماژول شامل یک حافظه دوبعدی ۳*۳ برای نگهداری Kernel است که می توانید آن را به صورت ROM پیاده سازی کنید (چون مقادیر داخل آن ثابت است و نیازی نداریم تغییر کند. بنابراین فقط کافیت reg را تعریف کرده و به آن مقدار اولیه بدهید). همچنین، ماتریس ۳*۳ ورودی را به کمک سه ورودی data_in، row_in[1:0] و col_in[1:0] دریافت می کنیم. به این ترتیب که هر موقع کاربر می خواهد ورودی را وارد کند، سیگنال data_in را یک می کند و با row_in و col_in مشخص می کند که روی چه سطر و ستونی از حافظه می خواهد مقدار data[7:0] را بنویسد. (مانند سخت افزار پروژه قبلی).

توجه کنید که این ماژول فقط حاصل کانولوشن دو ماتریس ۳*۳ را با فرض اینکه مرکز دو ماتریس کاملا بر هم منطبق است محاسبه می کند (عبارت $\sum_{i=-1}^{1}\sum_{j=-1}^{1} x(-i.-j)h(i.j)$).



شکل ۱ – شماتیک ماژول محاسبه کانولوشن.

چکیده :

در این فاز قصد داریم که کانولوشن دو ماتریس ۳ در ۳ را محاسبه کنیم. یکی از ماتریس ها بصورت ثابت درایه هاش مشخص شده است که همان ماتریس کرنل ما است و تنها درایه های یک ماتریس را بصورت رندوم قرار خواهیم داد.

مقدمه:

قبل از نوشتن کد این برنامه باید با دستور کانولوشن آشنا باشیم.

همانطور که در بخش دستور کار آمده بود، فرمول کانولوشن برابرست با :

$$y(m.n) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} x(m-i.n-j)h(i.j)$$

و نیز ماتریس کرنل ما که در اینجا با **h** نام گذاری شده است برابر با مقدار زیر است:

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

درایه های متناظر دو ماتریس ۳ در ۳ طبق فرمول در یکدیگر ضرب شده و برنامه عدد نهایی را به خروجی اعلام خواهد کرد.

شرح روند انجام پروژه (کد ماژول) :

برای نوشتن دستور و کد هر برنامه ای، ابتدا باید از نحوه اجرای آن در واقعیت آشنا بود و بعد آن را در قالب برنامه پیاده سازی و شبیه سازی کرد.

ما در بخش مقدمه به نحوه ی انجام محاسبه کانولوشن اشاره کردیم و در نتیجه با استفاده از آن به شرح نحوه ی انجام فاز اول پروژه دوم خواهیم پرداخت.

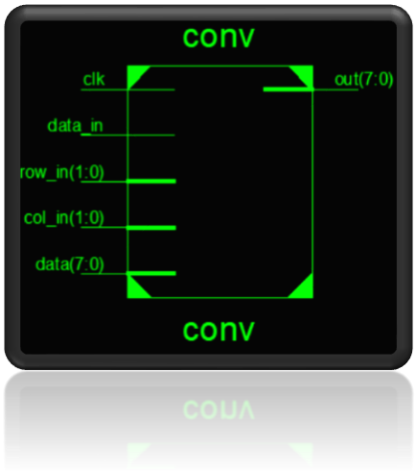
همانطور که در دستور کار آمده است ما یک ماتریس ورودی به نام **x** داریم که در حافظه (آرایه رجیستر) در ماژول ذخیره خواهد شد به این ترتیب که در هر پالس یک درایه جدید وارد خواهد شد.

ورودی دیگری تعریف می کنیم به نام **data_in** . با یک شدن **data_in** به ماژول اعلام می کنیم که می خواهیم داده درون ماتریس **x** قرار دهیم.

تمام اعداد وارد شده از طریق **data** در بخش **row_in** و **col_in** در ماتریس مورد نظر نوشته خواهد شد. یک ورودی دیگر به نام کلاک داریم که در بخش های بعدی بطور کامل آن را معرفی خواهیم کرد.

بعد از اینکه اعمال ضرب و جمع فرمول اجرا شد، نیاز به حافظه ای داریم که نتیجه را در آن ذخیره کنیم. به این منظور حافظه دیگری به نام **out** از نوع (آرایه رجیستر) تعریف می کنیم که نتایج در آن یادداشت شوند و نیز در لبه کلاک نتیجه مشاهده خواهد شد.

شماتیک پروژه بصورت زیر شد:



در شروع کد نویسی در بالای صفحه گزینه

scale time وجود دارد و می توان آن را به ۱۰ پیکو

ثانیه تغییر داد.

```
`timescale 1ns / 1ps
```

در مرحله بعدی تعیین ورودی ها و خروجی هارا

انجام می دهیم که به این صورت خواهد شد:

```
module conv( input clk, input data_in,
input [1:0]row_in, input [1:0]col_in,
input signed [7:0]data, output reg signed [7:0]out
);
```

در مرحله ماتریس ورودی و کرنل را از نوع آرایه رجیستر

معرفی خواهیم کرد که بصورت زیر خواهد بود:

```
reg [7:0]x[2:0][2:0];
reg signed [3:0]kernel[2:0][2:0];
```

در مرحله بعدی متغیر هایی از نوع انتیجر معرفی می کنیم

که در مقدار دهی به فرمول کانولوشن و محاسبه مقدار آن

نیاز پیدا خواهیم کرد:

```
integer m,n,sum;
```

در مرحله بعدی به مقداری دهی ماتریس کرنل خواهیم

پرداخت. چون یک ماتریس مشخص است فقط به آرایه

های متناظر مقدار مشخص شده را اختصاص می دهیم .در

نتیجه داریم:

```
initial begin
      kernel[0][0]= 1;
      kernel[0][1]= 2;
      kernel[0][2]= 1;
      kernel[1][0]= 0;
      kernel[1][1]= 0;
      kernel[1][2]= 0;
      kernel[2][0]= -1;
      kernel[2][1]= -2;
      kernel[2][2]= -1;
end
```

بعد از تعیین ورودی ها و خروجی ها به تعریف بدنه ماژول می پردازیم، در این بخش برنامه ما باید حساس به لبه کلاک باشد .

پس برای این مورد از **always @(posedge clk)** استفاده کردیم. سپس با یک شدن **data_in** و صفر بودن ریست به ماژول اعلام می کنیم که می خواهیم داده جدید را درون ماتریس قرار دهیم و یا خروجی را به نمایش بگذاریم. در نتیجه داریم:

```
always @(posedge clk) begin
if (data_in == 1) begin
x[row_in][col_in] = data;
end
```

در مرحله بعدی وقتی **data_in** برابر صفر شد یعنی دیگر داده جدیدی در درایه های ماتریس قرار نخواهد گرفت، به محاسبه خروجی خواهیم پرداخت. برای این منظور دو حلقه تعریف کردیم که مرحله به مرحله مقدار **sum** جدید را محاسبه کرده و با مقدار قبلی جمع کند.

در نهایت مقدار نهایی را در رجیستر خروجی **out** برای نمایش قرار دهد.

در نتیجه داریم:

```
else if (data_in == 0) begin
sum = 0;
for (m = 0 ; m <= 2 ; m = m + 1) begin
for (n = 0 ; n <= 2 ; n = n + 1) begin
sum = sum + x[m][n]*kernel[2-m][2-n];
end
end
out = sum;
end
```

حال باید شرایط خاص برنامه را تعیین کنیم. به این شکل که اگر عدد نهایی از ۲۵۵ بیشتر شود یا عدد مورد نظر منفی شود. چون ما دیتا را ۸ بیت تعریف کردیم حداکثر عدد ما ۲۵۵ ($۲^{۸}-۱ = ۲۵۵$) است در نتیجه این شرط را قرار می دهیم که اگر خروجی بیشتر از ۲۵۵ شد عدد ۲۵۵ و اگر منفی شد عدد صفر را در خروجی قرار دهد و در نهایت برنامه ماژول را با یک **endmodule** به پایان می رسانیم. در نتیجه خواهیم داشت:

```
if (sum > 255 ) begin
out = 255 ;
end
if (sum < 0 ) begin
out = 0 ;
end
end
endmodule
```

شرح روند انجام پروژه (کد تست بنچ) :

در شروع کد نویسی در بالای صفحه گزینه

scale time وجود دارد و می توان آن را به ۱۰ پیکو

ثانیه تغییر داد.

```
`timescale 1ns / 1ps
```

همانطور که انتظار داریم ورودی ها خروجی ها را

مشاهده کردیم:

```
module testconv;

   // Inputs
   reg clk;
   reg data_in;
   reg [1:0] row_in;
   reg [1:0] col_in;
   reg [7:0] data;

   // Outputs
   wire [7:0] out;

   // Instantiate the Unit Under Test (UUT)
   conv uut (
      .clk(clk),
      .data_in(data_in),
      .row_in(row_in),
      .col_in(col_in),
      .data(data),
      .out(out)
   );
```

در حالت اولیه تمامی مقدار ها را صفر در نظر می گیریم:

```
initial begin
      clk = 0;
      data_in=0;
      row_in=0;
      col_in=0;
      data=0;
```

در مرحله بعدی یک تاخیر ایجاد کرده و حالت های مختلف

را برای ورودی ها در نظر میگیرم و سپس دوباره کلاک را

صفر کرده و تاخیر قرار داده و دوباره با شرایط کلاک برابر

یک حالت دیگری را برای ورودی ها قرار می دهیم و این

روند را تا زمانی که تمام حالات را بررسی کنیم ادامه پیدا

می کند. در نتیجه داریم:

```
#100;
      clk = 1;
      data_in = 1;
      row_in = 0;
      col_in = 0;
      data = 1;
#10;
      clk = 0;
#10;
      clk = 1;
      data_in = 1;
      row_in = 0;
      col_in = 1;
      data = 2;
#10;
      clk = 0;
#10;
```

برای **data** می توان مقدار **$random** را قرار داد ولی چون با قرار دادن در آن جا خروجی منفی بدست آمد و صفر را گزارش داد برای نمایش و درک بهتر خروجی از اعداد بهینه تری استفاده کردم که خروجی به صورت واضح قابل رویت و درک باشد.

در انتها نیز دستور را با یک **endmodule** به پایان می رسانیم:

```
#10;
clk=0;
   end

endmodule
```

توضیح روند کلاک در برنامه :

در مدار های ترتیبی هر دستوری با کلاک انجام می شود.

همانطور که میدانیم مدار های ترتیبی به این گونه اند که با کلاک کار می کردند و با استناد به مدار منطقی **t_setuptime** و **t_holdtime** بین لبه بالا رونده اول و لبه بالا رونده بعدی دستورات را اجرا می کنند و دیتا پایدار می شود.

در **fpga** برای اینکه تعریف کنیم با یک کلاک تنظیم شود و دستورات را انجام دهد، خود یک ورودی ای را تعریف می کنیم به نام **clk** که مفهوم کلاک را برای کاربر دارد.

همانطور که می دانیم همه مدار های ترتیبی در بلوک **always** توصیف می شوند. به همین دلیل در این مورد از آن بلوک استفاده کرده و داخل شرط آن قرار می دهیم که حساس به لبه ی کالک باشد، **posedge** لبه بالا رونده، و سپس دستورات را انجام دهد. شرط های **always** بصورت موازی انجام می شوند یعنی اینکه در هر لبه کلاک در هر تعداد بلوک همه دستوات را چک می کند و دستورات را اجرا می کند.

کد ماژول :

```
module conv( input clk, input data_in,

input [1:0]row_in, input [1:0]col_in,

input signed [7:0]data, output reg signed

[7:0]out

;(

reg [7:0]x[2:0;[۲:۰][

reg signed [3:0]kernel[2:0 ;[۲:۰][

integer m,n,sum;

initial begin

    kernel[0][0]= 1;

        kernel[0][1]= 2;

        kernel[0][2]= 1;

        kernel[1][0]= 0;

        kernel[1][1]= 0;

        kernel[1][2]= 0;

        kernel[2][0]= -1;

        kernel[2][1]= -2;

        kernel[2][2]= -1;

end

always @(posedge clk) begin

if (data_in == 1) begin

x[row_in][col_in] = data;

end

else if (data_in == 0) begin

sum = 0;

for (m = 0 ; m <= 2 ; m = m + 1) begin

for (n = 0 ; n <= 2 ; n = n + 1) begin

sum = sum + x[m][n]*kernel[2-m][2-n];

end

end

out = sum;

end

if (sum > 255 ) begin

out = 255;

end
```

```
if (sum < 0 ) begin

out = 0;

end

end

endmodule
```

کد تست بنچ:

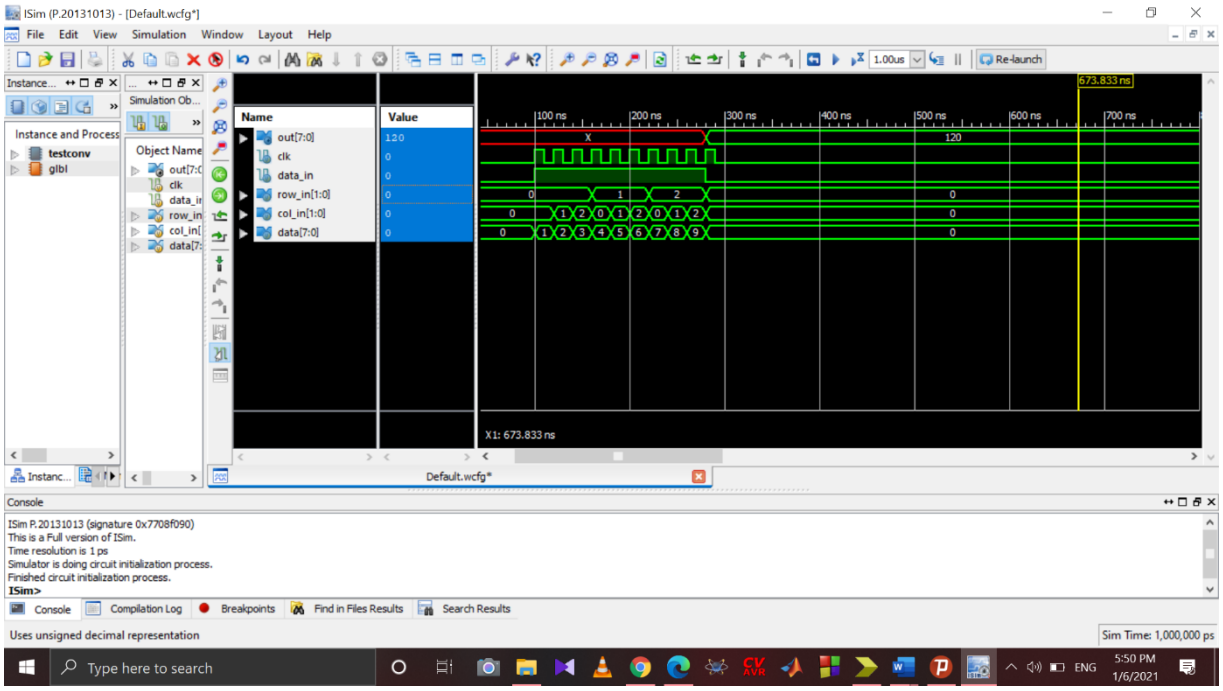
```
module testconv;
    //Inputs
    reg clk;
    reg data_in;
    reg [1:0] row_in;
    reg [1:0] col_in;
    reg [7:0] data;
    //Outputs
    wire [7:0] out;
    //Instantiate the Unit Under Test (UUT)
    conv uut(
        .clk(clk) ,
        .data_in(data_in) ,
        .row_in(row_in) ,
        .col_in(col_in) ,
        .data(data) ,
        .out(out)
    );
    //Initialize Inputs
initial begin
        clk = 0;
        data_in=0;
        row_in=0;
        col_in=0;
        data=0;
;#١٠٠
        clk = 1;
        data_in = 1;
        row_in = 0;
        col_in = 0;
        data = 1;
```

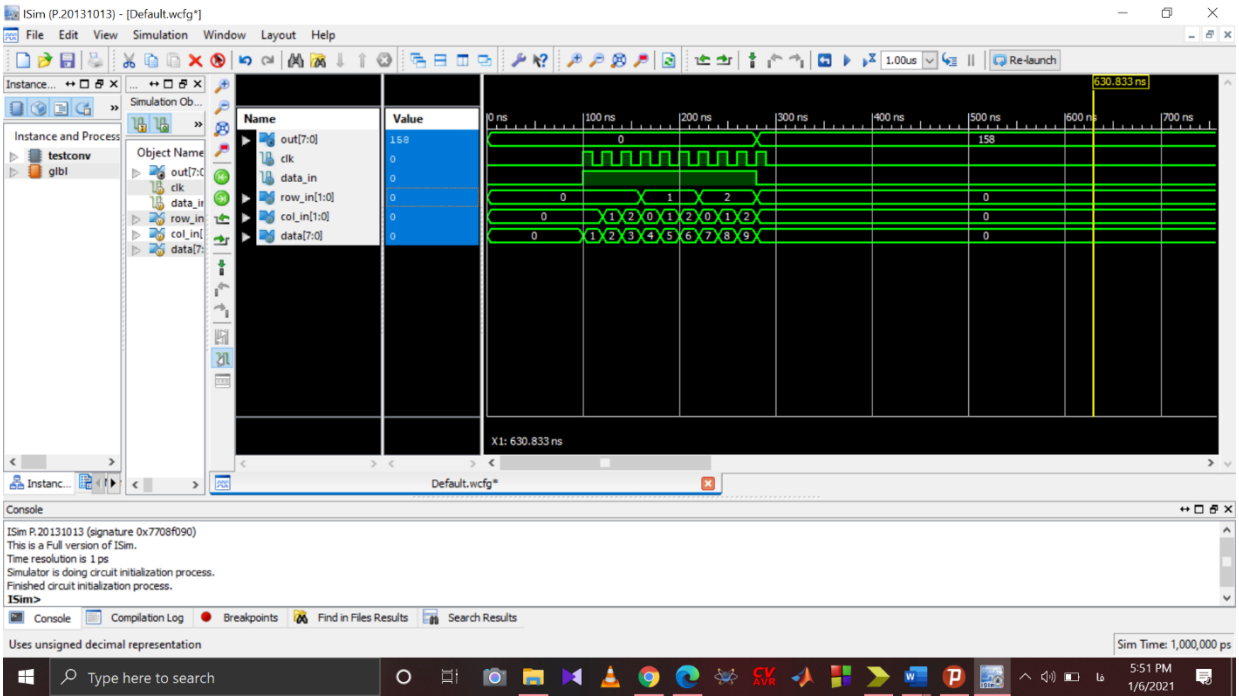
```
;#١٠
        clk = 1;
        data_in = 1;
        row_in = 1;
        col_in = 0;
        data = 4;
;#١٠
        clk = 0;

;#١٠
        clk = 1;
        data_in = 1;
        row_in = 1;
        col_in = 1;
        data = 5;
;#١٠
        clk = 0;

;#١٠
        clk = 0;
;#١٠
        clk = 1;
        data_in = 1;
        row_in = 0;
        col_in = 1;
        data = 2;
;#١٠
        clk = 0;
;#١٠
        clk = 1;
        data_in = 1;
        row_in = 0;
        col_in = 2;
        data = 3;
;#١٠
        clk = 0;
```

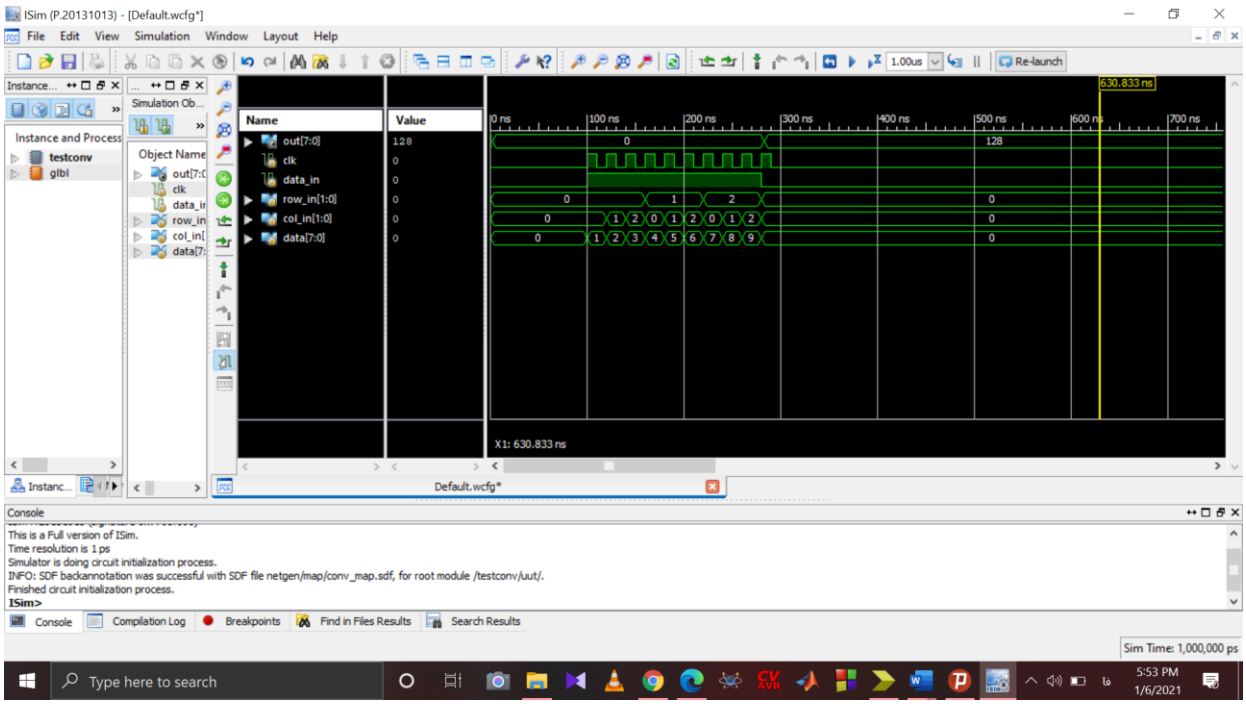
```
                data_in = 1;
                row_in = 2;
                col_in = 1;
                data = 8;
;#١٠
                clk = 0;
;#١٠
                clk = 1;
                data_in = 1;
                row_in = 2;
                col_in = 2;
                data = 9;
;#١٠
    clk = 0;
;#١٠
    clk = 1;
                data_in = 0;
                row_in = 0;
;#١٠
                clk = 1;
                data_in = 1;
                row_in = 1;
                col_in = 2;
                data = 6;
;#١٠
                clk = 0;
;#١٠
                clk = 1;
                data_in = 1;
                row_in = 2;
                col_in = 0;
                data = 7;
;#١٠
                clk = 0;
;#١٠
                clk = 1;
```

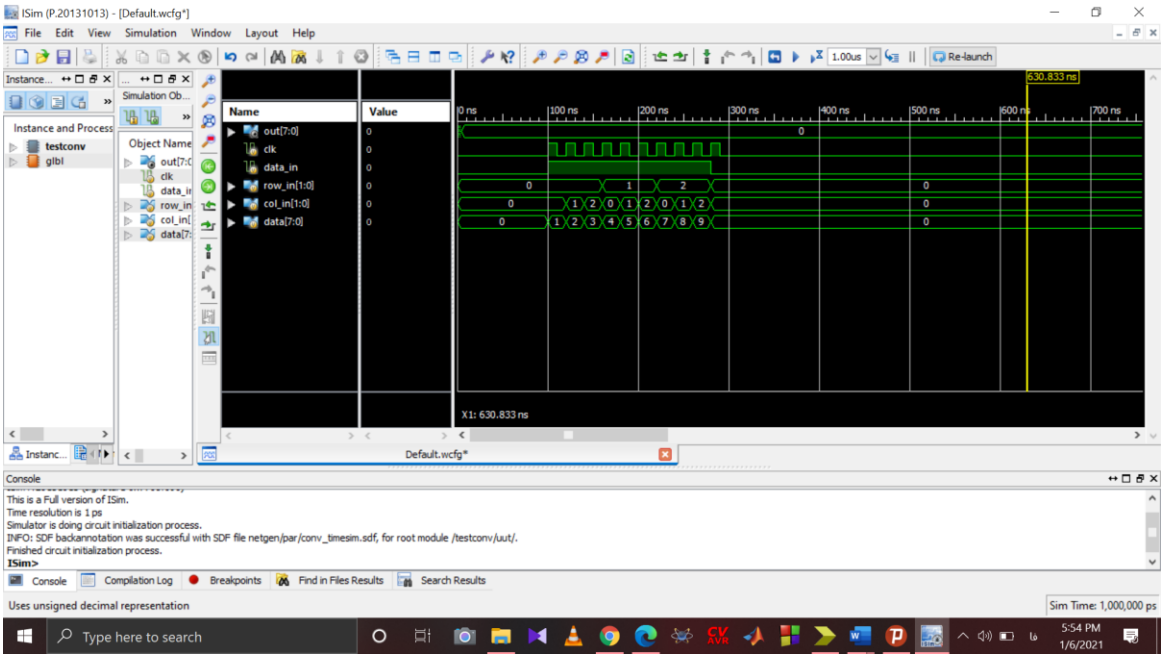
```
col_in = 0;

data=0;

;#۱۰clk=0;#10;

clk=1;#10;clk=0;#10;clk=1;#10;
clk=0;#10;clk=1;#10;clk=0;#10;
clk=1;#10;clk=0;#10;clk=1;#10;
clk=0;#10;clk=1;#10;clk=0;#10;
clk=1;#10;clk=0;#10;clk=1;#10;
clk=0;#10;clk=1;#10;clk=0;#10;
clk=1;#10;clk=0;#10;clk=1;#10;
clk=0;#10;clk=1;#10;clk=0;#10;
clk=1;#10;clk=0;#10;clk=1;#10;
clk=0;#10;clk=1;#10;clk=0;#10;
clk=1;#10;clk=0;#10;clk=1;#10;
clk=0;#10;clk=1;#10;clk=0;#10;
clk=1;#10;clk=0;#10;clk=1;#10;
clk=0;#10;clk=1;#10;clk=0;#10;
clk=1;#10;clk=0;#10;clk=1;#10;
clk=0;#10;clk=1;#10;clk=0;#10;
clk=1;#10;clk=0;#10;clk=1;#10;
clk=0;#10;clk=1;#10;clk=0;#10;
clk=1;#10;clk=0;#10;clk=1;#10;
clk=0;#10;clk=1;#10;clk=0;#10;
clk=1;#10;clk=0;#10;clk=1;#10;
clk=0;#10;clk=1;#10;clk=0;#10;
clk=1;#10;clk=0;#10;clk=1;#10;
clk=0;#10;clk=1;#10;clk=0;

end

endmodule
```

(کلاک های آخری قرار داده شدند تا فضای ۱ میکرو ثانیه پر شود در صورتی که تنها با اولین کلاک بعد از صفر شدن Data_in به کلاک های دیگر نیازی نبود.)

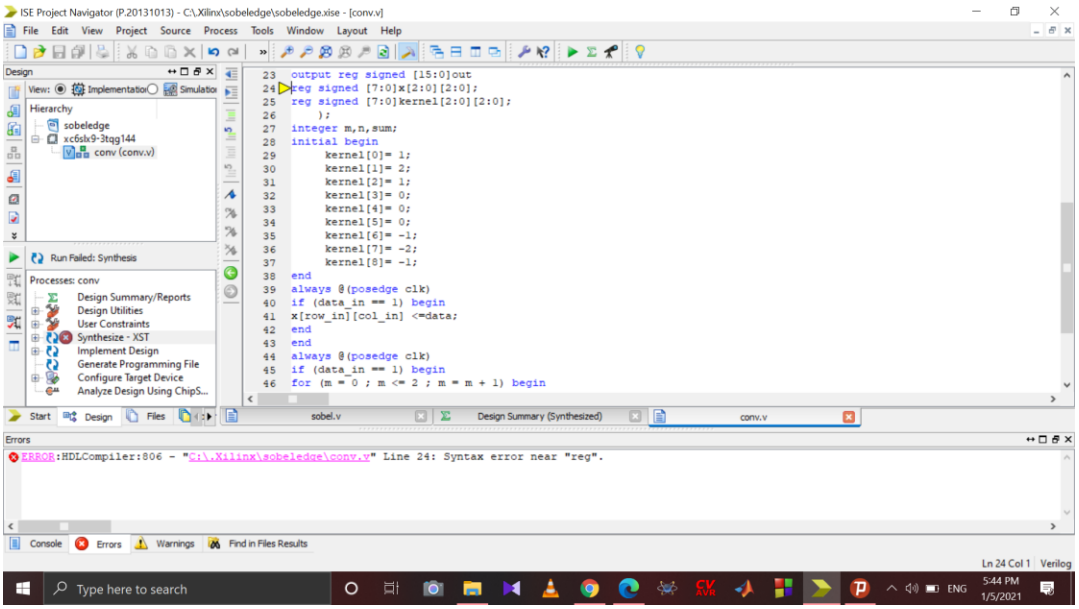

ISim (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
673.833 ns
Name
Value
out[7:0]
clk
data_in
row_in[1:0]
col_in[1:0]
data[7:0]
X1: 673.833 ns
Default.wcfg*
Console
ISim P.20131013 (signature 0x7708f090)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
Console
Compilation Log
Breakpoints
Find in Files Results
Search Results
Uses unsigned decimal representation
Sim Time: 1,000,000 ps
Type here to search
ENG
5:50 PM
1/6/2021

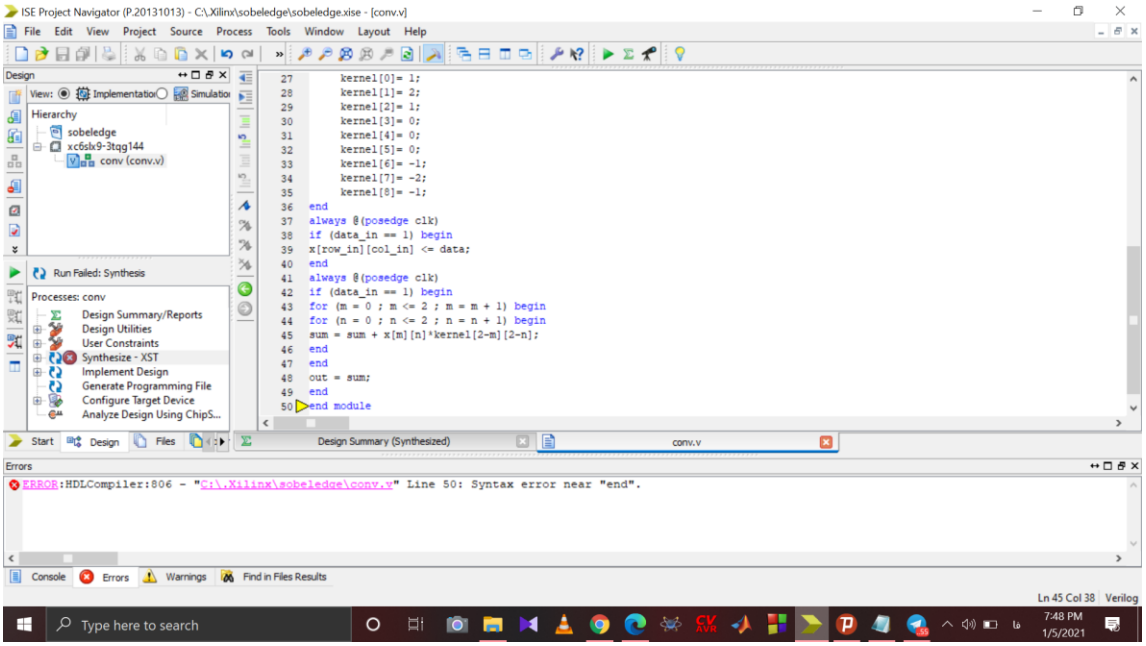

ISim (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
630.833 ns
Name
Value
out[7:0]
clk
data_in
row_in[1:0]
col_in[1:0]
data[7:0]
X1: 630.833 ns
Default.wcfg*
Console
ISim P.20131013 (signature 0x7708f090)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
Console
Compilation Log
Breakpoints
Find in Files Results
Search Results
Uses unsigned decimal representation
Sim Time: 1,000,000 ps
Type here to search
5:51 PM
1/6/2021

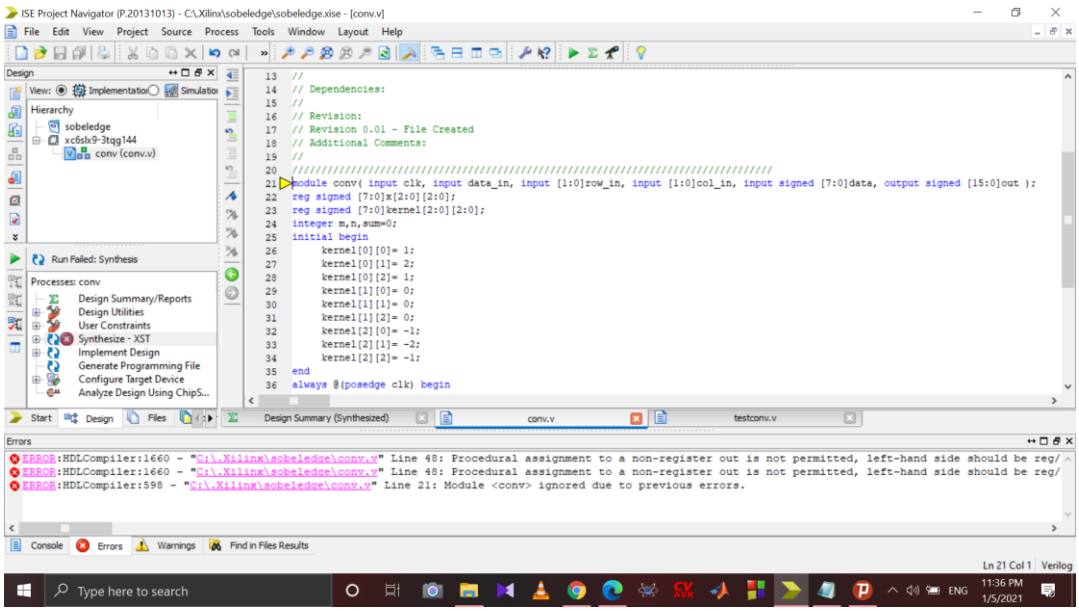

ISim (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
630.833 ns
Name
Value
out[7:0]
clk
data_in
row_in[1:0]
col_in[1:0]
data[7:0]
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
X1: 630.833 ns
Default.wcfg*
Console
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
INFO: SDF backannotation was successful with SDF file netgen/map/conv_map.sdf, for root module /testconv/uut/.
Finished circuit initialization process.
ISim>
Console
Compilation Log
Breakpoints
Find in Files Results
Search Results
Sim Time: 1,000,000 ps
Type here to search
5:53 PM
1/6/2021

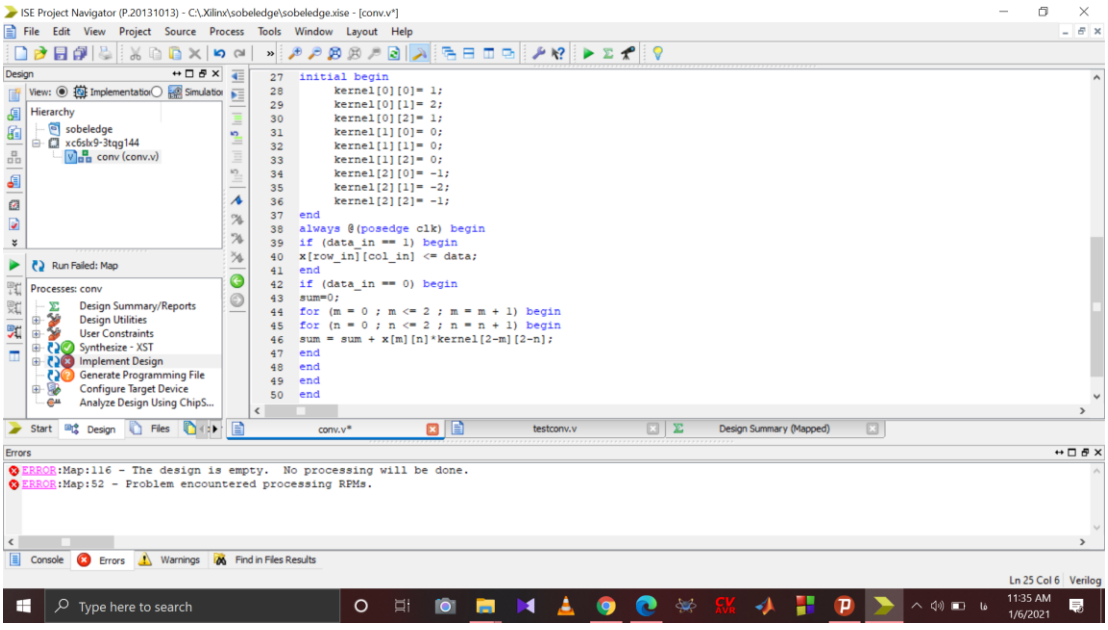

ISim (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
630.833 ns
Name
Value
out[7:0]
clk
data_in
row_in[1:0]
col_in[1:0]
data[7:0]
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
X1: 630.833 ns
Default.wcfg*
Console
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
INFO: SDF backannotation was successful with SDF file netgen/par/conv_timesim.sdf, for root module /testconv/uut/.
Finished circuit initialization process.
ISim>
Console
Compilation Log
Breakpoints
Find in Files Results
Search Results
Uses unsigned decimal representation
Sim Time: 1,000,000 ps
Type here to search
5:54 PM
1/6/2021

خطاهای مواجه شده:






ISE Project Navigator (P.20131013) - C:\Xilinx\sobeledge\sobeledge.xise - [conv.v]
File Edit View Project Source Process Tools Window Layout Help
Design
Hierarchy
sobeledge
xc6slx9-3tqg144
conv (conv.v)
Run Failed: Synthesis
Processes: conv
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipS...
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //
21 module conv(input clk, input data_in, input [1:0]row_in, input [1:0]col_in, input signed [7:0]data, output signed [15:0]out);
22 reg signed [7:0]x[2:0][2:0];
23 reg signed [7:0]kernel[2:0][2:0];
24 integer m,n,sum=0;
25 initial begin
26 kernel[0][0]= 1;
27 kernel[0][1]= 2;
28 kernel[0][2]= 1;
29 kernel[1][0]= 0;
30 kernel[1][1]= 0;
31 kernel[1][2]= 0;
32 kernel[2][0]= -1;
33 kernel[2][1]= -2;
34 kernel[2][2]= -1;
35 end
36 always @(posedge clk) begin
Design Summary (Synthesized)
conv.v
testconv.v
Errors
ERROR:HDLCompiler:1660 - "C:\.Xilinx\sobeledge\conv.v" Line 48: Procedural assignment to a non-register out is not permitted, left-hand side should be reg/
ERROR:HDLCompiler:1660 - "C:\.Xilinx\sobeledge\conv.v" Line 48: Procedural assignment to a non-register out is not permitted, left-hand side should be reg/
ERROR:HDLCompiler:598 - "C:\.Xilinx\sobeledge\conv.v" Line 21: Module <conv> ignored due to previous errors.
Console Errors Warnings Find in Files Results
Ln 21 Col 1 Verilog
Type here to search
ENG
11:36 PM
1/5/2021


ISE Project Navigator (P.20131013) - C:\Xilinx\sobeledge\sobeledge.xise - [conv.v*]
File Edit View Project Source Process Tools Window Layout Help
Design
Hierarchy
sobeledge
xc6slx9-3tqg144
conv (conv.v)
Run Failed: Map
Processes: conv
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipS...
27 initial begin
28 kernel[0][0]= 1;
29 kernel[0][1]= 2;
30 kernel[0][2]= 1;
31 kernel[1][0]= 0;
32 kernel[1][1]= 0;
33 kernel[1][2]= 0;
34 kernel[2][0]= -1;
35 kernel[2][1]= -2;
36 kernel[2][2]= -1;
37 end
38 always @(posedge clk) begin
39 if (data_in == 1) begin
40 x[row_in][col_in] <= data;
41 end
42 if (data_in == 0) begin
43 sum=0;
44 for (m = 0 ; m <= 2 ; m = m + 1) begin
45 for (n = 0 ; n <= 2 ; n = n + 1) begin
46 sum = sum + x[m][n]*kernel[2-m][2-n];
47 end
48 end
49 end
50 end
conv.v*
testconv.v
Design Summary (Mapped)
Errors
ERROR:Map:116 - The design is empty. No processing will be done.
ERROR:Map:52 - Problem encountered processing RPMs.
Console Errors Warnings Find in Files Results
Ln 25 Col 6 Verilog
Type here to search
11:35 AM
1/6/2021