گزارش فاز اول پروژه درس FPGA

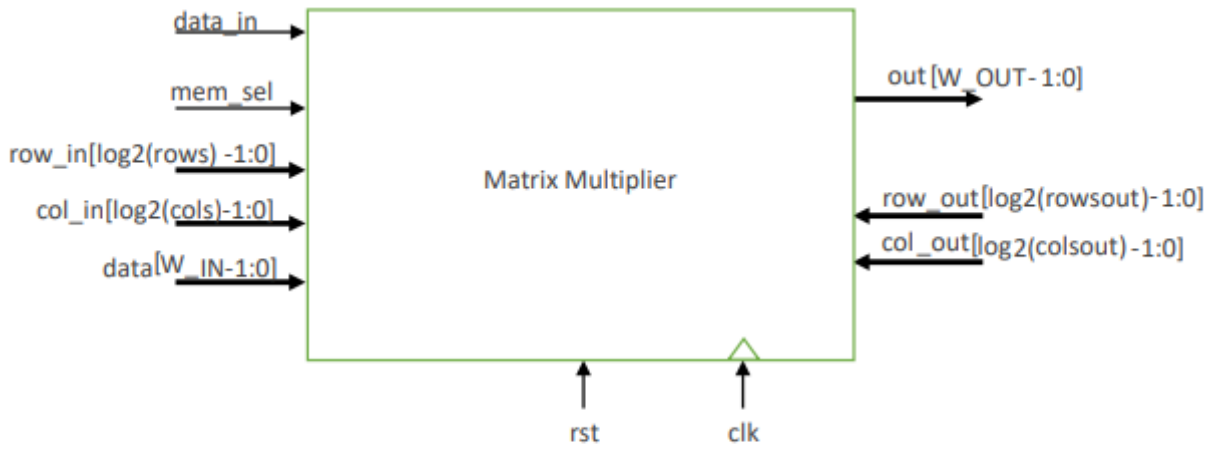
زبان برنامه نویسی Verilog و نرم افزار Xilinx

نسرین کریمی (۹۷۲۳۶۰۸۱)

Nasrinkarimi7979@gmail.com

دانشکده مهندسی برق دانشگاه شهید بهشتی – پردیس فنی و مهندسی شهید عباسپور

در این پروژه با استفاده از وریلاگ یک ضرب کننده ماتریسی را توصیف و تست می نماییم. شمای کلی این ضرب کننده در شکل زیر نشان داده شده است. در این ماژول ماتریس اول A و ماتریس دوم B است.



درایه های دو ماتریس ورودی A و B در دو حافظه (آرایه رجیستر) درون ماژول ذخیره می شوند. به منظور دریافت درایه های ماتریس ها از کاربر، از ورودی های mem_sel ، row_in، col_in، data و data_in استفاده می کنیم. به این ترتیب که با یک کردن ورودی data_in به ماژول اعلام می‌کنیم که می‌خواهیم داده جدیدی درون ماتریس بنویسیم. اگر mem_sel=0 باشد، عددی که از طریق data به ماژول وارد می کنیم، روی درایه (row_in , col_in) از ماتریس A نوشته می شود و اگر mem_sel=1 باشد، data روی درایه (row_in , col_in) از ماتریس B نوشته می شود. (در واقع، ورودی‌هایrow_in و col_in به عنوان آدرس حافظه استفاده می‌شوند.)

همچنین، درایه های ماتریس حاصلضرب result نیز در یک حافظه درون ماژول ذخیره می شوند و کاربر در هر پالس ساعت می تواند یکی از درایه های ماتریس حاصلضرب را در خروجی out بخواند. برای این کار، کاربر باید سطر و ستون درایه مورد نظر را از طریق ورودی‌های row_out و col_out به ماژول اعلام کند.

چکیده:

در این پروژه قصد داریم که دو ماتریس از کاربر دریافت کرده و حاصل ضرب آن دو ماتریس را بدست آوریم.

مقدمه:

قبل از شروع به کار برنامه نویسی ماژول مورد نظر ابتدا باید از شیوه ی صحیح ضرب ماتریسی آگاه باشیم. همانطور که می دانیم در ضرب دو ماتریس با یکدیگر باید تعداد ستون ماتریس اول با سطر ماتریس دوم برابر باشد. در این پروژه ماتریس ورودی A ما یک ماتریس ۱*۲ است یعنی دارای ۲ سطر و یک ستون می باشد. ماتریس ورودی دیگر ما به نام ماتریس B یک ماتریس ۳*۱ است یعنی دارای یک سطر و ۳ ستون می باشد. درنتیجه این ضرب ماتریسی قابل انجام است.

دستور اصلی ضرب ماتریسی به شکل زیر است:

اگر A یک ماتریس از مرتبه m*n و B یک ماتریس از مرتبه n*k با درایه های حقیقی باشد، در حاصل ضرب ماتریسی نتیجه آن را که برای مثال ماتریس C می نامیم، درایه های ماتریس C از رابطه زیر بدست می آید:

$$C_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{nj} = \sum_{r=1}^{n} a_{ir}b_{rj}$$

در این موردی که ما در این پروژه با آن روبرو هستیم چون ماتریس اول دارای یک ستون و ماتریس دوم دارای یک سطر است یعنی دیگر به جمع کردن حاصل ضرب ها در کد نویسی نیازی نداریم در نتیجه دارایه ها کافیست با یکدیگر ضرب ساده شوند.

یعنی به صورت زیر محاسبه خواهند شد:

$$\mathbf{AB} = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix} [\, B_1 \quad B_2 \quad \ldots \quad B_p \,] =$$

$$\begin{bmatrix} (A_1 \cdot B_1) & (A_1 \cdot B_2) & \ldots & (A_1 \cdot B_p) \\ (A_2 \cdot B_1) & (A_2 \cdot B_2) & \ldots & (A_2 \cdot B_p) \\ \vdots & \vdots & \ddots & \vdots \\ (A_m \cdot B_1) & (A_m \cdot B_2) & \ldots & (A_m \cdot B_p) \end{bmatrix}$$

شرح روند انجام پروژه (کد ماژول) :

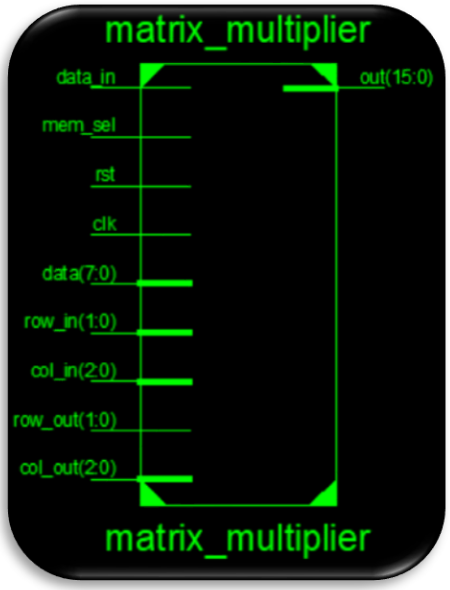
برای نوشتن دستور و کد هر برنامه ای، ابتدا باید از نحوه اجرای آن در واقعیت آشنا بود و بعد آن را در قالب برنامه پیاده سازی و شبیه سازی کرد.

ما در ابتدا به طریقه انجام ضرب ماتریسی اشاره کردیم در نتیجه با استفاده از آن به شرح نحوه ی انجام فاز اول پروژه خواهیم پرداخت.

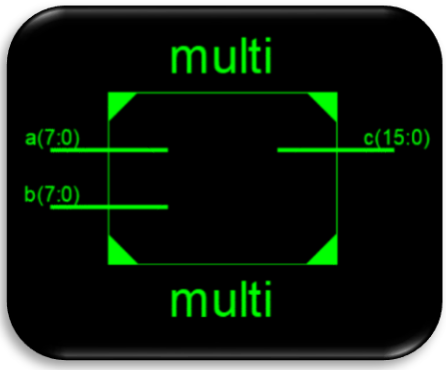
همانگونه که در دستورکار آمده است، ما دو ورودی ماتریس به نام های A و B داریم که در دو حافظه (آرایه رجیستر) در ماژول ذخیره خواهند شد. ورودی دیگری تعیین می کنیم به اسم data_in . با یک شدن data_in به ماژول اعلام می کنیم که می خواهیم داده جدید را درون ماتریس قرار دهیم. تمام اعداد وارد شده از طریق data در بخش row_in و col_in در ماتریس مورد نظر نوشته می شوند. یک ورودی دیگر داریم به اسم mem_sel . دو شرایط صفر و یک شدن را برایش درنظر می گیریم.

به این ترتیب که اگر mem_sel=0 شود اعداد وارد شده در ماتریس A ذخیره شوند. (همان سطر و ستون دیتا که توضیح دادیم) و اگر mem_sel=1 باشد، در ماتریس B.

بعد از اینکه عمل ضرب نیز انجام شد، نیاز به حافظه ای داریم که نتیجه را در آن ذخیره کنیم. به این منظور یک حافظه دیگری به نام result در ماژول طراحی می کنیم که نتیجه در آن یادداشت شوند. و برای آن نیز دو ورودی col_out و row_out تعریف می کنیم و کارب در هر پالس ساعت می تواند یکی از داریه های ماریس حاصلضرب را در خروجی out بخواند. برای این کار، کاربر باید سطر و ستون درایه مورد نظر را از طریق ورودی های col_out و row_out به ماژول اعلام کند.



Technology Schematic



RLT Schematic

در شروع کد نویسی در بالای صفحه گزینه time scale وجود دارد و من آن را به ۱۰ پیکو ثانیه تغییر دادم.

```
`timescale 1ns / 10ps
```

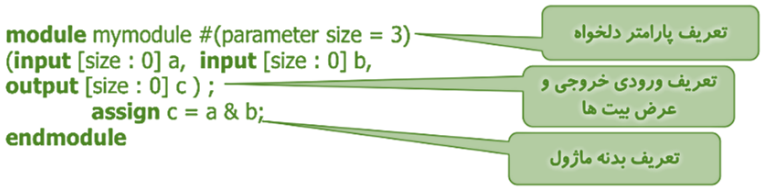
سپس در قسمت ماژول ابتدا عرض بیت های ورودی و خروجی را مشخص می کنیم طبق جدولی که در دستورکار آمده است.

جدول 1- پارامترهای سخت افزار

پارامتر	شرح	پیش فرض
W_IN	عرض بیت درایه های ماتریسهای ورودی	8
W_OUT	عرض بیت درایه های ماتریس خروجی	16
ROWS_A	تعداد سطر های ماتریس A	2
COLS_A	تعداد ستون های ماتریس A	2
ROWS_B	تعداد سطرهای ماتریس B	2
COLS_B	تعداد ستون های ماتریس B	2

باید توجه داشته باشیم که اعداد جدول داده شده برای فاز اول پروژه مناسب نیستند زیرا تعداد سطر و ستون ها متفاوت است و باید آن را متناسب با ماتریس های خود تصحیح کرد.

برای تعریف ماژول از روش زیر استفاده می کنیم:



در نتیجه کد ما به این شکل شد.:

```
module multiplier #(parameter W_IN=8, W_OUT=16, ROWS_A=2, COLS_A=1, ROWS_B=1, COLS_B=3)
```

در مرحله بعدی تعیین ورودی ها و خروجی هارا انجام می دهیم که به این صورت خواهد شد:

```
(input clk, input reset, input data_in, input mem_sel, input [1:0]row_in,
.nput [2:0]col_in, input [1:0]row_out, input [2:0]col_out, input signed[W_IN-1:0]data
)utput reg signed[W_OUT-1:0]out );
:eg signed [7:0]A[0:1];
:eg signed [7:0]B[0:2];
:eg signed [15:0]result[0:1][0:2];
.nteger i,j;
```

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 از اینکه 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 اگر mem_sel=0 شود اعداد وارد شده در ماتریس A ذخیره شوند و اگر mem_sel=1 باشد، در ماتریس B.

در نتیجه خواهیم داشت:

```
always @(posedge clk)
if (reset && data_in)begin
if(mem_sel)begin
B[row_in]<=data;
end
if(mem_sel==0)begin
A[col_in]<=data;
end
```

در مرحله بعدی بعد از اینکه داده ها را دریافت کردیم و به فرمول اصلی برنامه می رسیم که چگونگی ضرب را نشان خواهد داد. همانطور که در ابتدا در بخش ضرب ماتریسی توضیح دادیم چون ماتریس اول دارای یک ستون و ماتریس دوم دارای یک سطر است یعنی دیگر به جمع کردن حاصل ضرب ها در کد نویسی نیازی نداریم در نتیجه دارایه ها کافیست با یکدیگر ضرب ساده شوند. پس کافیست دو متغیر i , j را بعنوان شماره سطر و ستون ها معرفی کنیم و دستور ضرب درایه اول از سطر اول ماتریس A در درایه اول از ستون اول ماتریس B بنویسیم که به شرح زیر است:

```
for(i=0;i<2;i=i+1)begin
for(j=0;j<3;j=j+1)begin
result[i][j]=A[i]*B[j];
end
end
```

حال شرایطی را تعیین می کنیم که در صورت فرخوانی آدرس سطر و ستون توسط کاربر حاصل ضرب انجام شده آن درایه نمایش داده شود و همچنین شرایطی که در آن ریست یک شود به معنی اینکه خروجی را صفر کند، در نتیجه از دستور زیر استفاده می کنیم و برنامه را با endmodule به پایان رسانده و برنامه را ذخیره می کنیم:

```
always @ (posedge clk)begin
if(reset==0 && data_in==1)begin
out=result[row_out][col_out];
end else begin
out=0;
end
end
endmodule
```

شرح نوشتن کد تست بنچ:

پس از ایجاد فایل تست بنچ در ابتدا time scale را به ۱۰ پیکو ثانیه تغییر دادم که میتوان تغییر نداد.

همانطور که انتظار داریم ورودی ها خروجی ها را مشاهده کردیم:

```
module testbench_mutiplier;

   // Inputs
   reg clk;
   reg reset;
   reg data_in;
   reg mem_sel;
   reg [1:0] row_in;
   reg [2:0] col_in;
   reg [1:0] row_out;
   reg [2:0] col_out;
   reg [7:0] data;

   // Outputs
   wire [15:0] out;
```

در حالت اولیه تمامی مقدار ها صفر در نظر گرفته می شوند:

```
initial begin
   // Initialize Inputs
   clk = 0;
   reset = 0;
   data_in = 0;
   mem_sel = 0;
   row_in = 0;
   col_in = 0;
   row_out = 0;
   col_out = 0;
   data = 0;
```

در اینجا خود با دادن مقدار های تصادفی شکل موج های خروجی خواسته شده را بدست خواهیم آورد.

در ابتدا برای تولید پالس ساعت از حلقه های نامتنهای به نام forever استفاده می کنیم. توجه داریم که از یک تاخیر در حلقه استفاده کنیم تا دچار هنگ نشود:

```
forever begin
clk=!clk;
#10;
end
```

در بخش بعدی به صورت رندوم اعداد را مقدار دهی می کنیم:

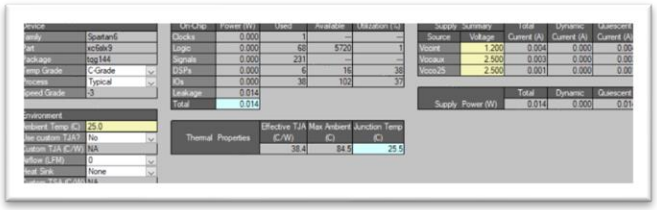
```
#20;
data_in = 1;
mem_sel = 0;
reset = 0;
data = $random;
row_in = 0;
col_in = 0;
#20;
```

در حالت های بعدی نیز که در کد کامل مشاهده خواهید کرد به همین منوال پیش خواهیم رفت.

در انتها نیز کد را با یک endmodule به پایان می رسانیم.

توان مصرفی ماژول:

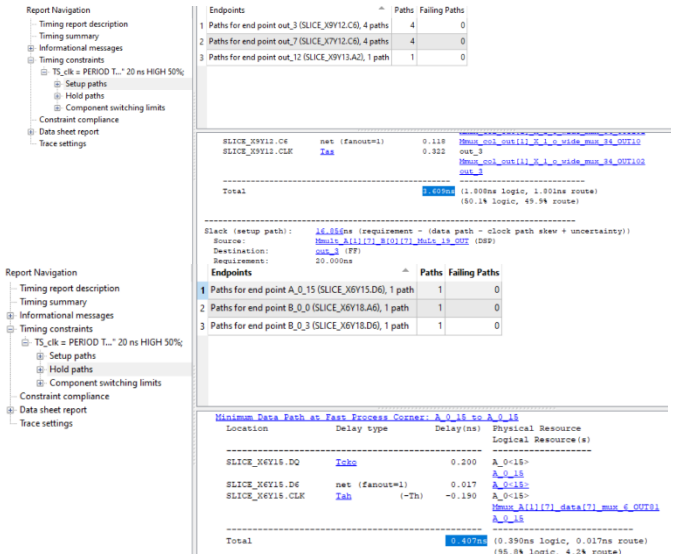
برای بدست آوردن توان مصرفی ماژول از بخش tools گزینه xpower analiyzer را انتخاب کرده و توان مصرفی را مشاهده خواهیم کرد:



توان مصرفی برحسب وات برابر۰.۰۱۴ بدست آمد.

حداکثر فرکانس کلاک:

پس از اینکه در منوی toolsکلاک را بعنوان ورودی معرفی کردیم در بخش گزارش برنامه این دیتا را بدست آوردیم:



	Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
Yes	TS_clk = PERIOD TIMEGRP "clk" 20 ns HIGH 50%	SETUP	16.341ns	3.659ns	0	0
		HOLD	0.407ns		0	0

Source Clock	Src:Rise|Dest:Rise	Src:Fall|Dest:Rise	Src:Rise|Dest:Fall	Src:Fall|Dest:Fall
clk	3.659			

در نتیجه خواهیم داشت:

$$3.659 + 0.407 = 4.066ns$$

و از آنجایی که داریم:

$$F = \frac{1}{T} = \frac{1}{4.066ns} = 246MHz$$

$$F = \frac{1}{T} = \frac{1}{3.659ns} = 273MHz$$

توضیح روند کار 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 های منطقی t_holdtime , t_setuptime بین لبه بالارونده اول و لبه بالارونده 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 مدا 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 کلاک باشد، posedge لبه بالارونده، 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.

نتایج بدست آمده در برنامه:

$$A = \begin{bmatrix} 36 \\ -127 \end{bmatrix}$$

$$B = [9 \quad 99 \quad 13]$$

$$AB = \begin{bmatrix} 324 & 3564 & 468 \\ -1143 & -12573 & -1651 \end{bmatrix}$$

همانطور که انتظار می رفت اعداد صحیحی بدست آمدند.

کد ماژول:

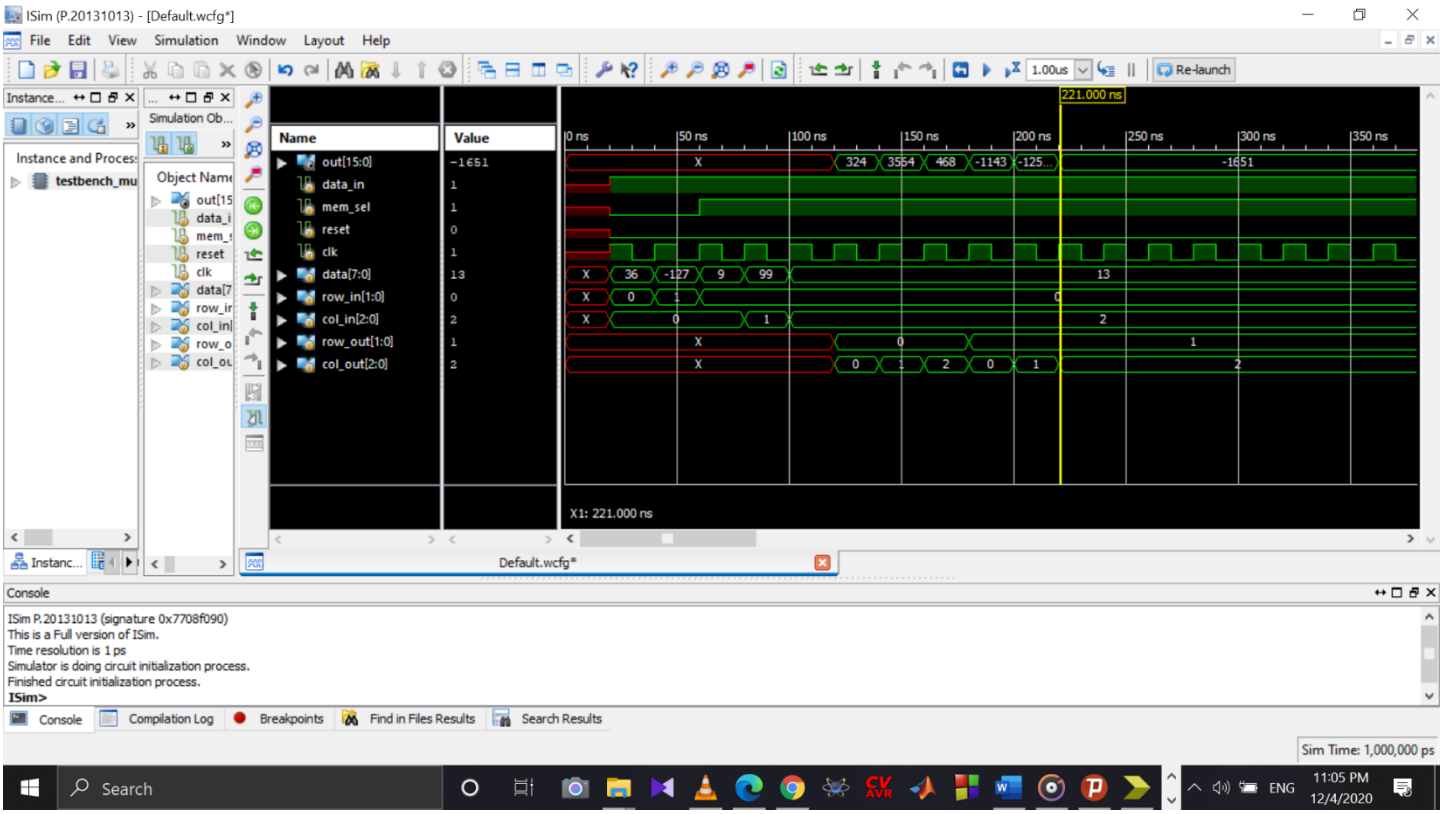
```
module mutrix_multiplier
#(parameter W_IN=8 , W_OUT=16 ,
ROWS_A=2 , COLS_A=1 ,
ROWS_B=1 , COLS_B=3 )
(input data_in, input mem_sel, input
reset, input clk,
input signed[W_IN-1 : 0]data ,input
[1 : 0]row_in, input [2 : 0]col_in,
input [1 :
0]row_out, input [2 : 0]col_out,
output reg signed [W_OUT-1 : 0]out
 );
reg signed[7:0]A[1:0];
reg signed[7:0]B[2:0];
reg signed[15:0]result[1:0][2:0];
integer i,j;
always @ (posedge clk)
 if(reset==0 && data_in==1)begin
 if(mem_sel==0)begin
 A[row_in] <=data;
end
if(mem_sel==1)begin
 B[col_in]<=data;
end
 for(i=0;i<2;i=i+1)begin
 for(j=0;j<3;j=j+1)begin
 result[i][j] =A[i]*B[j];
end
end
end
always @ (posedge clk)
if(reset==0 && data_in==1)begin
out=result[row_out][col_out];
end else begin
out=0;
end
endmodule
```

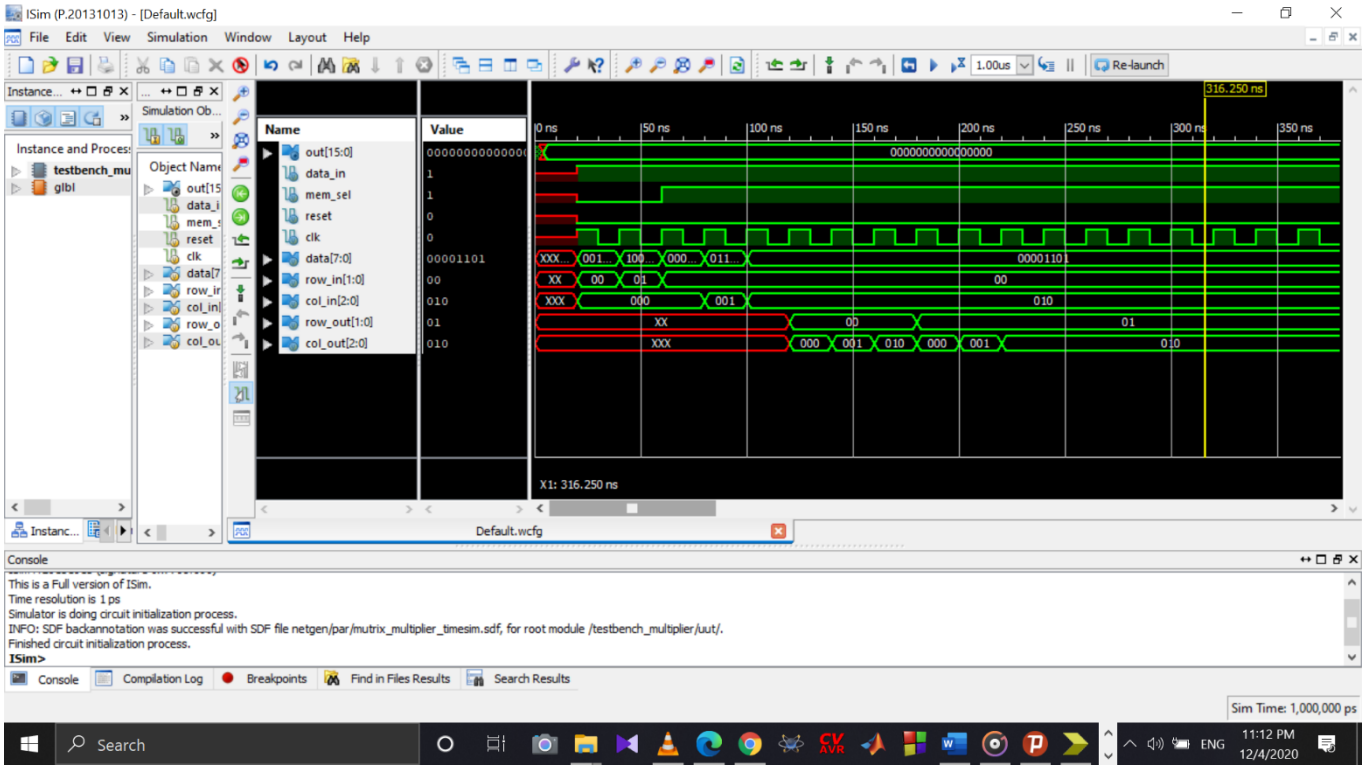
کد تست بنچ:

```
module testbench_multiplier;
// Inputs
reg data_in;
reg mem_sel;
reg reset;
reg clk;
reg [7:0] data;
reg [1:0] row_in;
reg [2:0] col_in;
reg [1:0] row_out;
reg [2:0] col_out;
// Outputs
wire [15:0] out;
// Instantiate the Unit Under Test (UUT)
mutrix_multiplier uut (
.data_in(data_in),
.mem_sel(mem_sel),
.reset(reset),
.clk(clk),
.data(data),
.row_in(row_in),
.col_in(col_in),
.row_out(row_out),
.col_out(col_out),
.out(out)
);
initial begin
#20;
clk=1;
forever begin
#10 clk =!clk;
end
end
initial begin
#20;
data_in = 1;
mem_sel = 0;
reset = 0;
data = $random;
row_in = 0;
col_in = 0;
#20;
data_in = 1;
mem_sel = 0;
reset = 0;
data = $random;
row_in = 1;
col_in = 0;
```

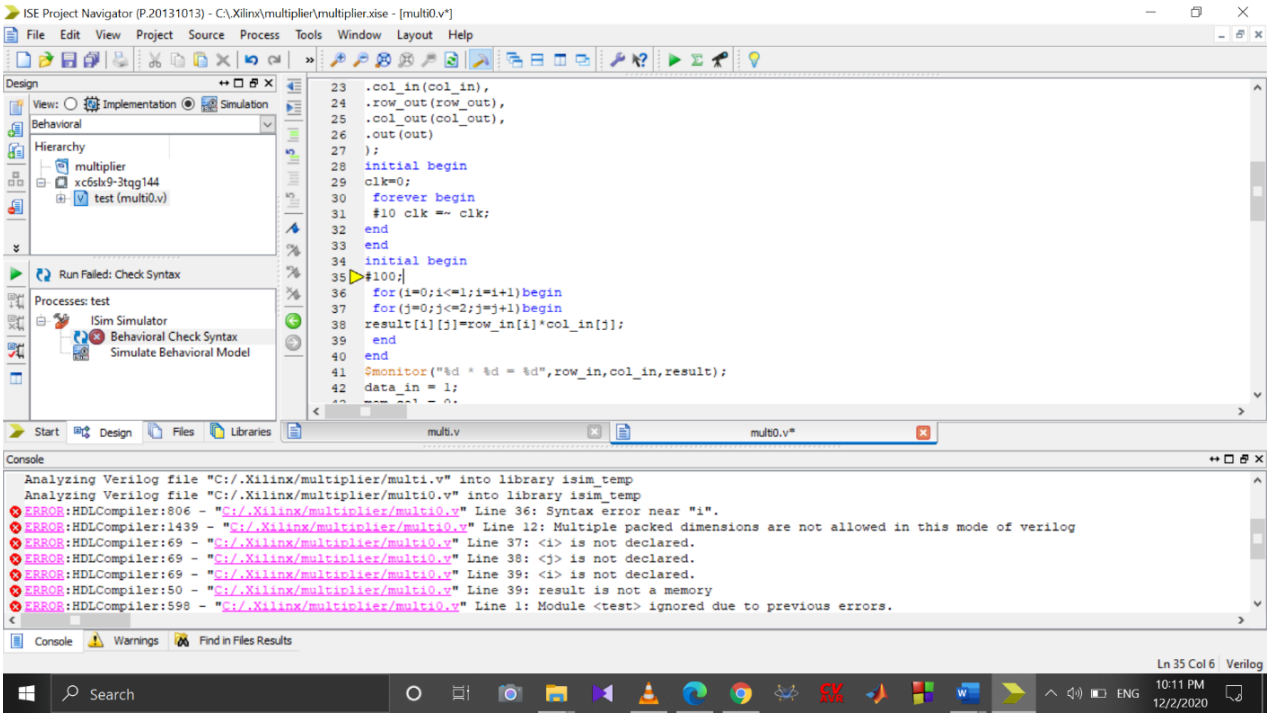
```
row_out=0;
col_out=1;
#20;
row_out=0;
col_out=2;
#20;
row_out=1;
col_out=0;
#20;
row_out=1;
col_out=1;
#20;
row_out=1;
col_out=2;
end

endmodule
```

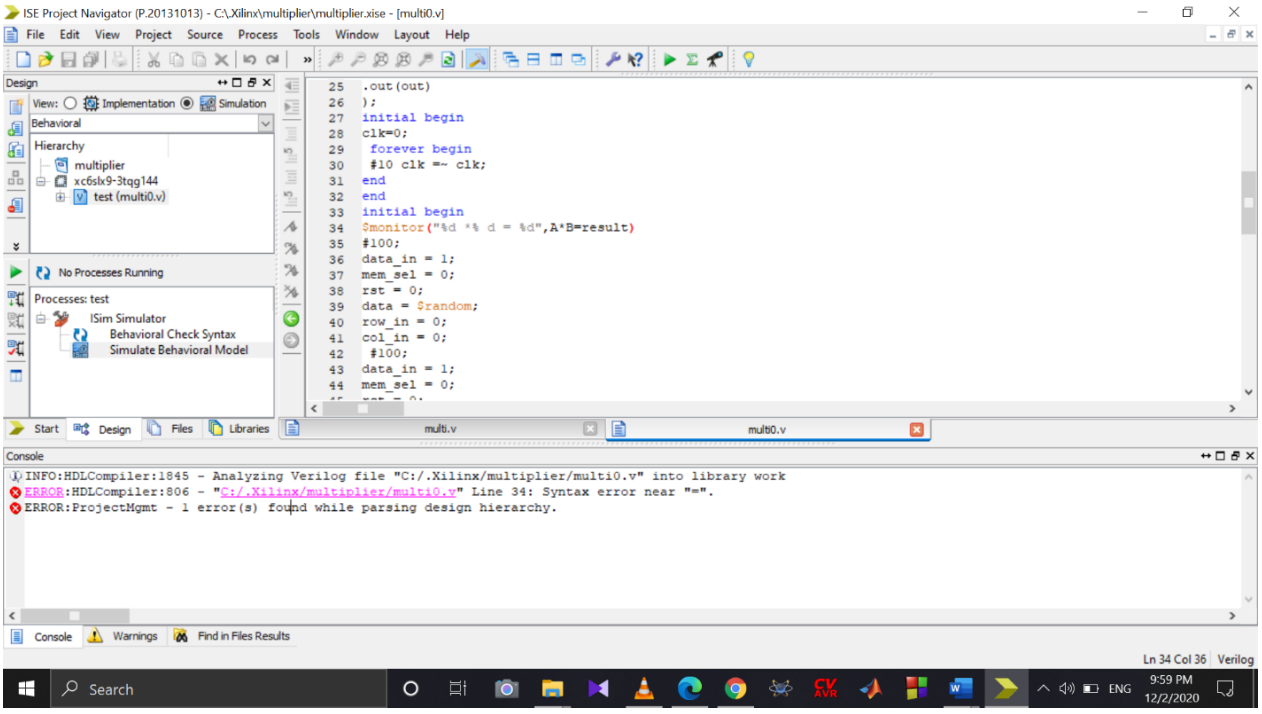
```
#20;
data_in = 1;
mem_sel = 1;
reset = 0;
data = $random;
row_in = 0;
col_in = 0;
#20;
data_in = 1;
mem_sel = 1;
reset = 0;
data = $random;
row_in = 0;
col_in =1;
#20;
data_in = 1;
mem_sel = 1;
reset = 0;
data = $random;
row_in = 0;
col_in = 2;
#20;
row_out=0;
col_out=0;
#20;
```

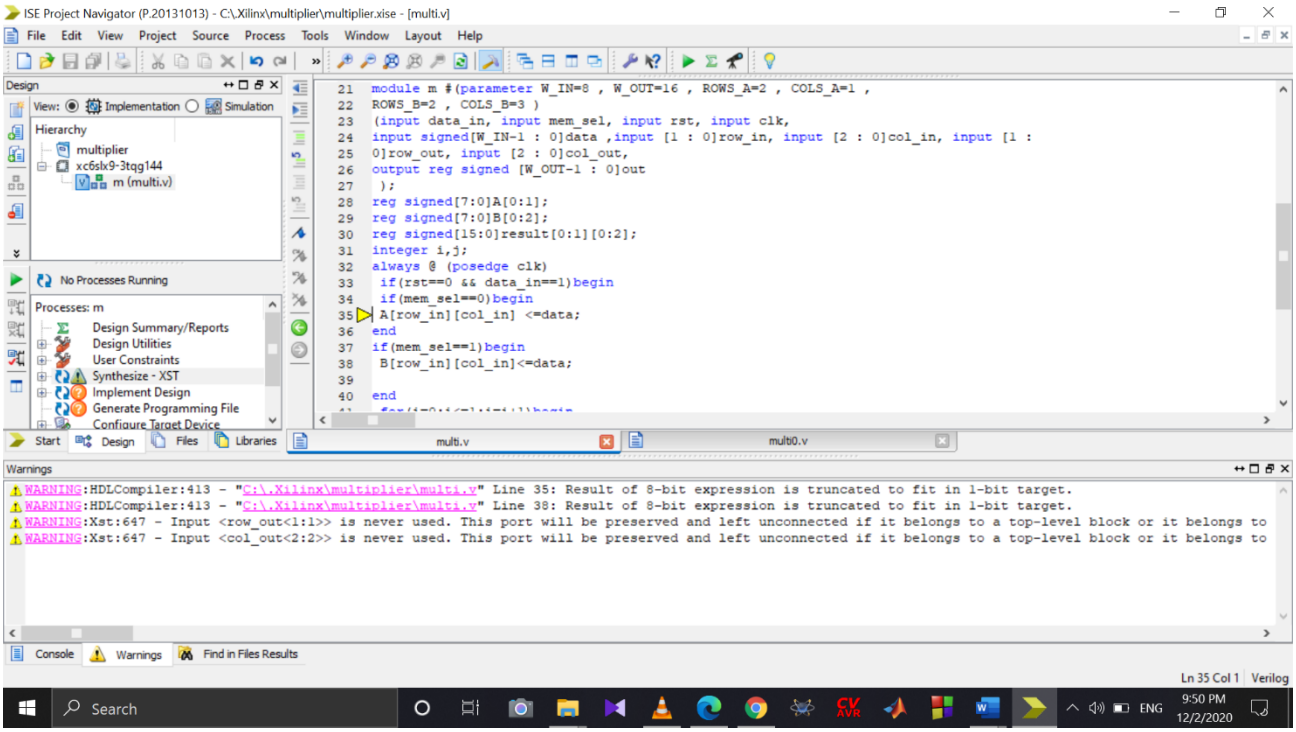

ISim (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
Name
Value
out[15:0]
data_in
mem_sel
reset
clk
data[7:0]
row_in[1:0]
col_in[2:0]
row_out[1:0]
col_out[2:0]
-1651
1
1
0
1
13
0
2
1
2
221.000 ns
X1: 221.000 ns
Default.wcfg*
Console
ISim P.20131013 (signature 0x7708f090)
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
Sim Time: 1,000,000 ps
Search
ENG
11:05 PM
12/4/2020

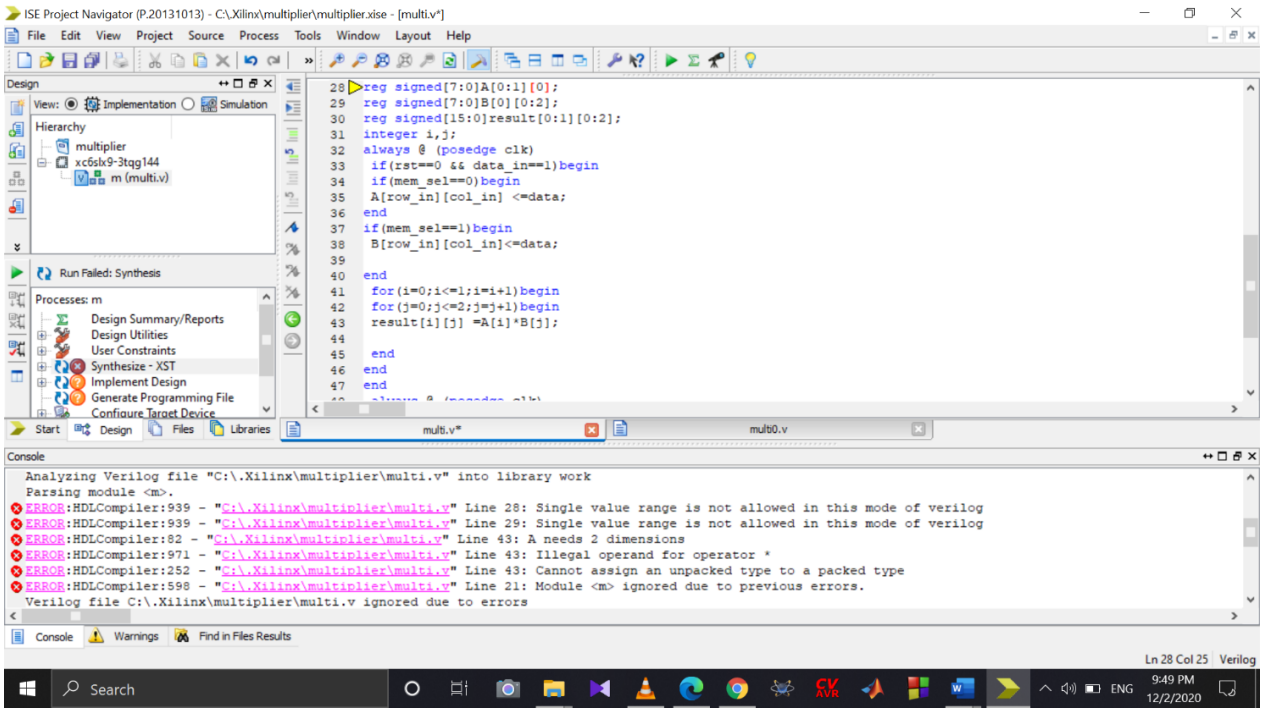

ISim (P.20131013) - [Default.wcfg]
File Edit View Simulation Window Layout Help
Re-launch
Name
Value
out[15:0]
data_in
mem_sel
reset
clk
data[7:0]
row_in[1:0]
col_in[2:0]
row_out[1:0]
col_out[2:0]
00001101
00
010
01
010
316.250 ns
X1: 316.250 ns
Default.wcfg
Console
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
INFO: SDF backannotation was successful with SDF file netgen/par/mutrix_multiplier_timesim.sdf, for root module /testbench_multiplier/uut/.
Finished circuit initialization process.
ISim>
Console
Compilation Log
Breakpoints
Find in Files Results
Search Results
Sim Time: 1,000,000 ps
Search
ENG
11:12 PM
12/4/2020

خطاهایی که در برنامه با آن ها روبرو شدم:






ISE Project Navigator (P.20131013) - C:\Xilinx\multiplier\multiplier.xise - [multi.v]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
multiplier
xc6slx9-3tqg144
m (multi.v)
No Processes Running
Processes: m
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Start Design Files Libraries
multi.v
multi0.v
21 module m #(parameter W_IN=8 , W_OUT=16 , ROWS_A=2 , COLS_A=1 ,
22 ROWS_B=2 , COLS_B=3)
23 (input data_in, input mem_sel, input rst, input clk,
24 input signed[W_IN-1 : 0]data ,input [1 : 0]row_in, input [2 : 0]col_in, input [1 :
25 0]row_out, input [2 : 0]col_out,
26 output reg signed [W_OUT-1 : 0]out
27);
28 reg signed[7:0]A[0:1];
29 reg signed[7:0]B[0:2];
30 reg signed[15:0]result[0:1][0:2];
31 integer i,j;
32 always @ (posedge clk)
33 if(rst==0 && data_in==1)begin
34 if(mem_sel==0)begin
35 A[row_in][col_in] <=data;
36 end
37 if(mem_sel==1)begin
38 B[row_in][col_in]<=data;
39
40 end
Warnings
WARNING:HDLCompiler:413 - "C:\.Xilinx\multiplier\multi.v" Line 35: Result of 8-bit expression is truncated to fit in 1-bit target.
WARNING:HDLCompiler:413 - "C:\.Xilinx\multiplier\multi.v" Line 38: Result of 8-bit expression is truncated to fit in 1-bit target.
WARNING:Xst:647 - Input <row_out<1:1>> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it belongs to
WARNING:Xst:647 - Input <col_out<2:2>> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it belongs to
Console Warnings Find in Files Results
Ln 35 Col 1 Verilog
Search
ENG
9:50 PM
12/2/2020


ISE Project Navigator (P.20131013) - C:\Xilinx\multiplier\multiplier.xise - [multi.v*]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
multiplier
xc6slx9-3tqg144
m (multi.v)
Run Failed: Synthesis
Processes: m
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Start Design Files Libraries
multi.v*
multi0.v
28 reg signed[7:0]A[0:1][0];
29 reg signed[7:0]B[0][0:2];
30 reg signed[15:0]result[0:1][0:2];
31 integer i,j;
32 always @ (posedge clk)
33 if(rst==0 && data_in==1)begin
34 if(mem_sel==0)begin
35 A[row_in][col_in] <=data;
36 end
37 if(mem_sel==1)begin
38 B[row_in][col_in]<=data;
39
40 end
41 for(i=0;i<=1;i=i+1)begin
42 for(j=0;j<=2;j=j+1)begin
43 result[i][j] =A[i]*B[j];
44
45 end
46 end
47 end
Console
Analyzing Verilog file "C:\.Xilinx\multiplier\multi.v" into library work
Parsing module <m>.
ERROR:HDLCompiler:939 - "C:\.Xilinx\multiplier\multi.v" Line 28: Single value range is not allowed in this mode of verilog
ERROR:HDLCompiler:939 - "C:\.Xilinx\multiplier\multi.v" Line 29: Single value range is not allowed in this mode of verilog
ERROR:HDLCompiler:82 - "C:\.Xilinx\multiplier\multi.v" Line 43: A needs 2 dimensions
ERROR:HDLCompiler:971 - "C:\.Xilinx\multiplier\multi.v" Line 43: Illegal operand for operator *
ERROR:HDLCompiler:252 - "C:\.Xilinx\multiplier\multi.v" Line 43: Cannot assign an unpacked type to a packed type
ERROR:HDLCompiler:598 - "C:\.Xilinx\multiplier\multi.v" Line 21: Module <m> ignored due to previous errors.
Verilog file C:\.Xilinx\multiplier\multi.v ignored due to errors
Console Warnings Find in Files Results
Ln 28 Col 25 Verilog
Search
ENG
9:49 PM
12/2/2020