به نام خدا

توضیحات مربوط به تمرین اول درس طراحی سیستم های دیجیتال
دکتر فصحتی

امیرمحمد شربتی – 402106112

1404/1/31

.1

منظور از دستورات continues assignments همان دستور assign است. هر وقت سمت راست انتساب تغییر کند، سمت چپ assign نیز تغییر می‌کند. در این صورت نیازی به استفاده از always در بسیاری موارد نداریم. assign برای مدارات ترکیبی استفاده میشود و همچنین سمت چپ انتساب هم wire است.

ابتدا یک ماژول برای decoder 2*4 تعریف میکنیم. ورودی این ماژول آرایه ای دو عضوی و خروجی آن آرایه ای چهار عضوی است. در داخل ماژول هم چهار حالت and دو سیگنال و not آنها را می‌نویسیم:

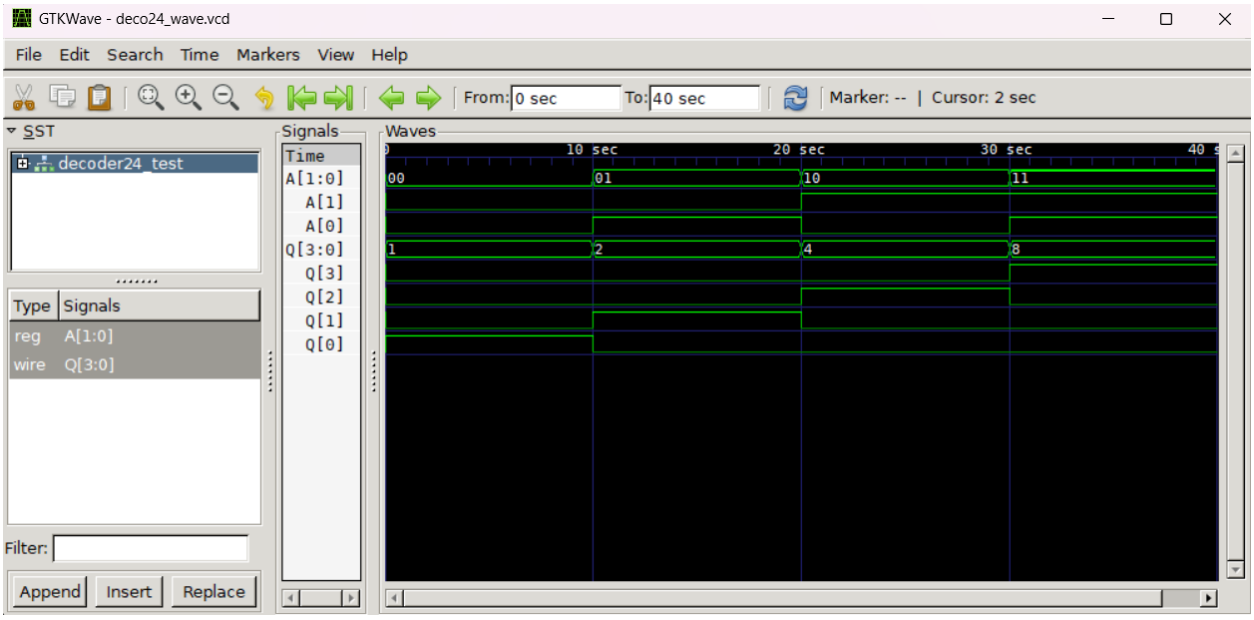
```
assign Q[0] = ~A[0] & ~A[1];

assign Q[1] = A[0] & ~A[1];

assign Q[2] = ~A[0] & A[1];

assign Q[3] = A[0] & A[1];
```

پس از آن ماژول تستی برای این دیکودر می‌نویسیم. فقط اینکه بنده هم از Quartus و هم از Icarus استفاده می‌کنم. دو خط در test bench اضافه می‌کنم که برای مشاهده waveform به کمک Icarus و gtk است:

```
$dumpfile("deco24_wave.vcd");

$dumpvars(0, decoder24_test);
```

پس از آن هم هر چهار حالت ممکن برای دو سیگنال ورودی را با تاخیر 10 واحدی به instance ماژول دیکودر دو به چهار میدهیم. خروجی هم چاپ میشود و هم از طریق waveform قابل مشاهده است.

```
PS C:\Users\ASUS\OneDrive\Desktop\university\4\DSD\T1\T1_1> iverilog -o sim.vvp decoder24.v decoder24_test.v
PS C:\Users\ASUS\OneDrive\Desktop\university\4\DSD\T1\T1_1> vvp sim.vvp
VCD info: dumpfile deco24_wave.vcd opened for output.
A=00 -> Q=0001
A=01 -> Q=0010
A=10 -> Q=0100
A=11 -> Q=1000
decoder24_test.v:15: $finish called at 40 (1s)
PS C:\Users\ASUS\OneDrive\Desktop\university\4\DSD\T1\T1_1> gtkwave deco24_wave.vcd

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

[0] start time.
[40] end time.
WM Destroy
```



در گام بعد از این 2*4 decoder یک 3*8 decoder می‌سازیم. (یعنی می‌خواهیم طراحی سلسله مراتبی داشته باشیم.)

برای این کار ایده کلی این است که دو ورودی از سه ورودی decoder 3*8 را به decoder 2*4 بدهیم و سپس به کمک چهار خروجی و یک ورودی دیگر، 8 حالت ممکن را بسازیم. برای انجام این کار روش های متفاوتی وجود دارد.

یک ایده استفاده از mux یا ternary condition است. ابتدا چهار خروجی را به کمک دو بیت کم ارزش تولید می‌کنیم. سپس به کمک بیت سوم ورودی، تعیین میکنیم که این چهار بیت در چهار بیت چپ خروجی اصلی باشند یا چهار بیت راست:

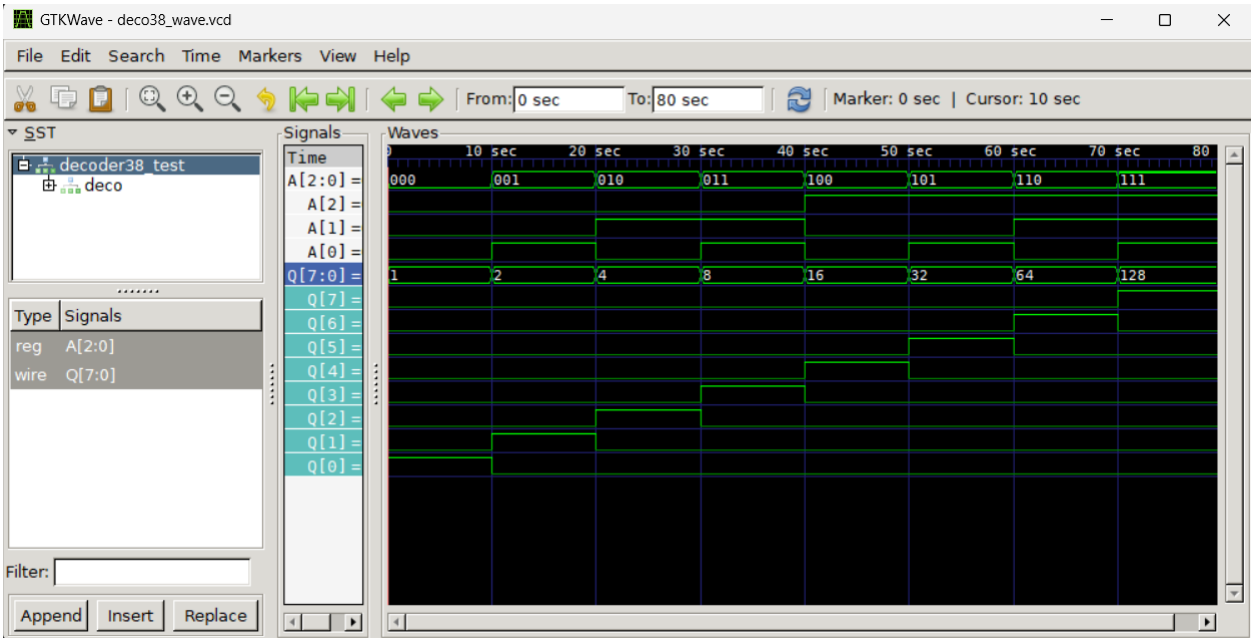
```
wire [3:0] QQ;

decoder24 deco(A[1:0], QQ);

assign Q = A[2] ? {QQ, 4'b0000} : {4'b0000, QQ};
```

ایده دیگر مثلا قرار دادن enable برای decoder 2*4 است. می‌توان بیت سوم را به آن وصل کرد. ایده دیگر مثلا and کردن بیت سوم با خروجی هاست...

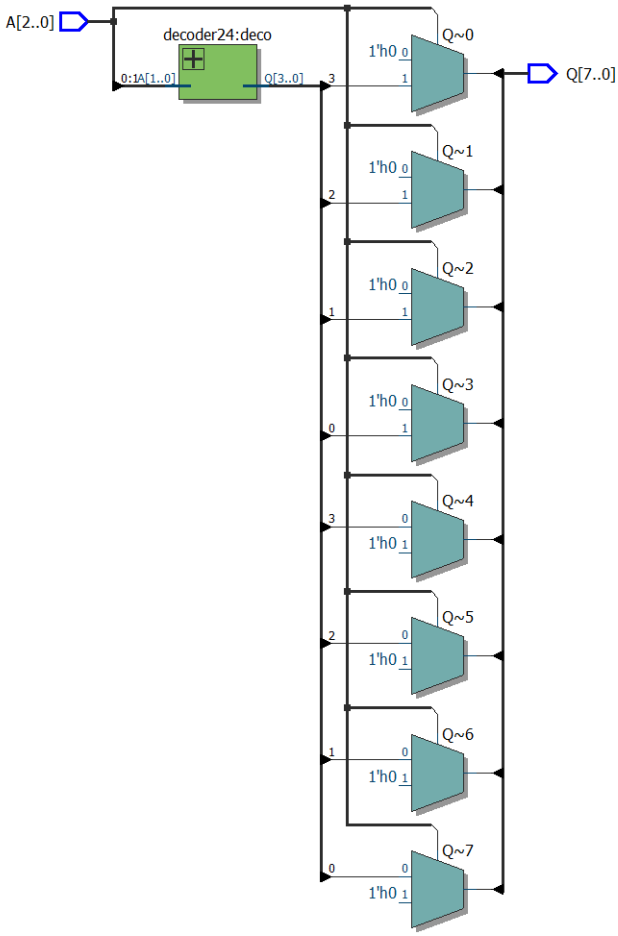
خلاصه که برای پیاده سازی این روش های متفاوتی وجود دارد ولی بنده از روش ternary یا همان mux استفاده کردم.

این هم نتیجه تست مدار :

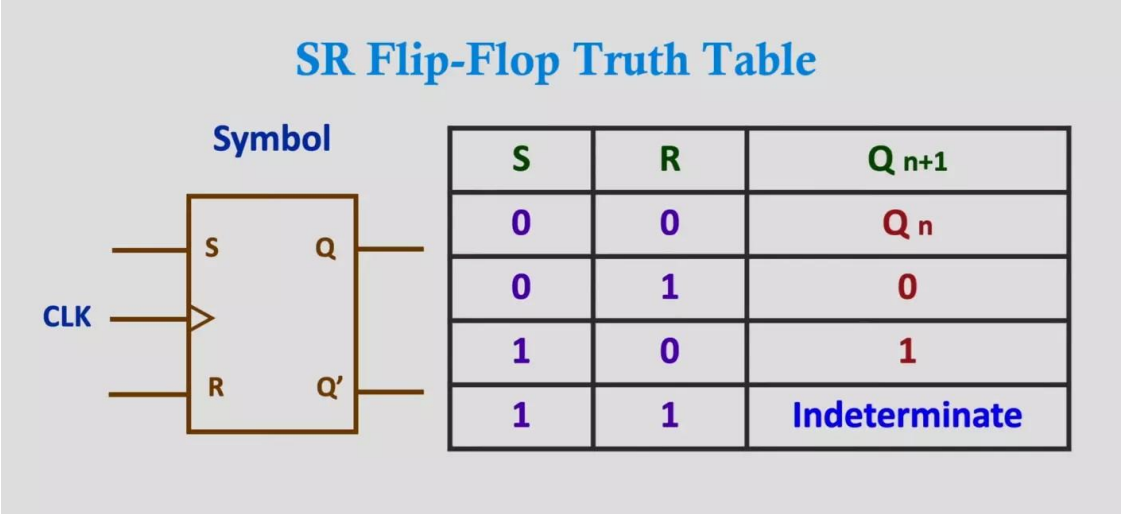
```
PS C:\Users\ASUS\OneDrive\Desktop\university\4\DSD\T1\T1_1> vvp deco38.vvp
VCD info: dumpfile deco38_wave.vcd opened for output.
A=000 -> Q=00000001
A=001 -> Q=00000010
A=010 -> Q=00000100
A=011 -> Q=00001000
A=100 -> Q=00010000
A=101 -> Q=00100000
A=110 -> Q=01000000
A=111 -> Q=10000000
decoder38_test.v:21: $finish called at 80 (1s)
```



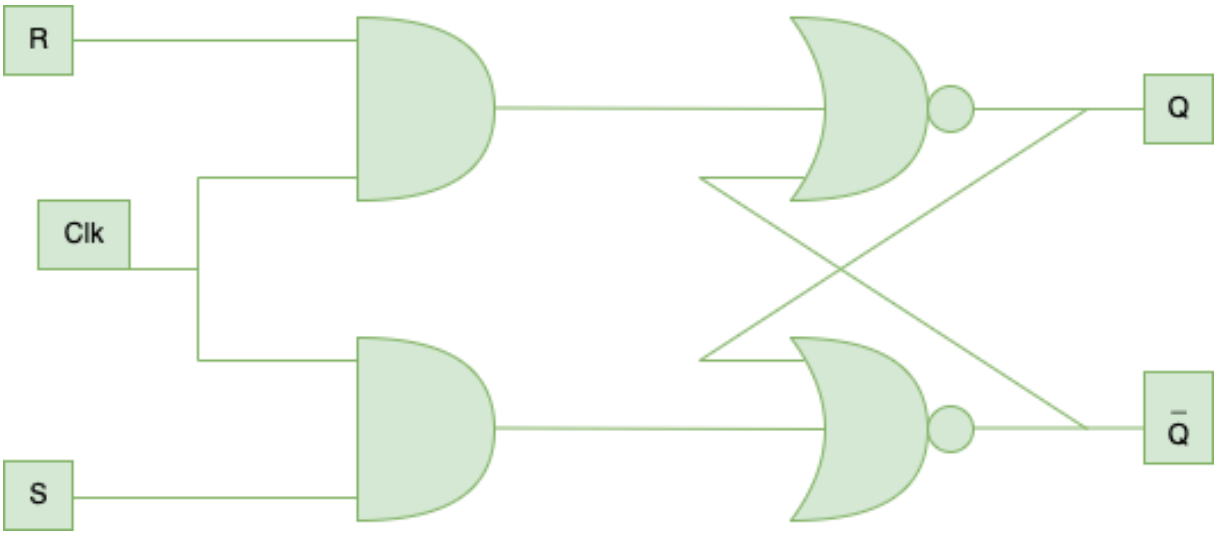
این هم شماتیکی ست که کوارتوس ترسیم کرده:



2. این جدول درستی SR FF است:



با دانستن ساختار داخلی بهتر متوجه این جدول درستی می‌شویم:



در این سخت افزار از SR FF استفاده شده است که همواره S = ~R . پس در واقع از D FF استفاده شده است. در این حالت $Q_{n+1} = S$. پس اگر سیگنال های کنترلی اجازه ورود داده را بدهند، این سخت افزار شیفت چپ را انجام داده و بیت ورودی به عنوان بیت کم ارش قرار میگیرد. حال اگر Sr_En غیرفعال باشد نقیض بیت سوم به عنوان بیت کم ارزش قرار

می‌گیرد، در واقع مشابه circular shift با این تفاوت که یک بیت not می‌شود. منظورم از اجازه دادن بیت های کنترلی هم این بود که اگر CLR = 1، در این صورت تمام بیت ها صفر میشوند. در اولویت بعدی اگر SET = 1 شود، تمام خروجی ها 1 می‌شوند.

برای پیاده سازی عملکرد این مدار این کد را میتوان پیاده سازی کرد:

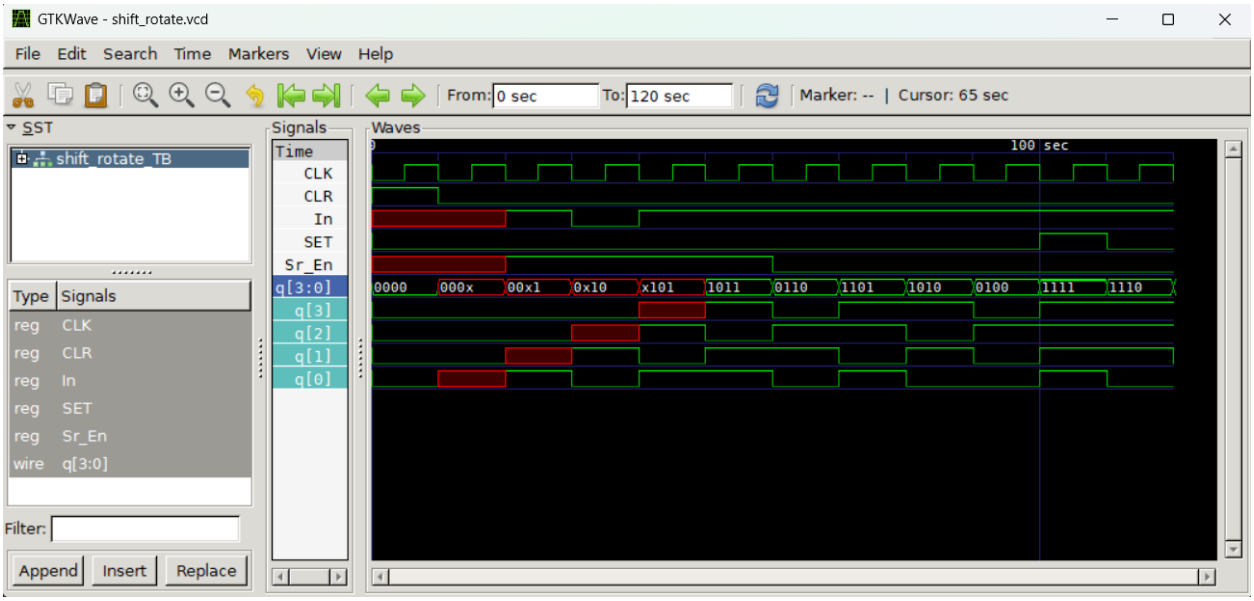
```
wire shift_in = Sr_En ? In : ~q[3

always @(negedge CLK, posedge SET, posedge CLR)

begin

    if (CLR) q <= 4'b0000;

    else if(SET) q <= 4'b1111;

    else q <= {q[2:0] , shift_in};

end
```

آسنکرون بودن سیگنال های کنترلی در نظر گرفته شده. خط اول هم به این شیوه قابل پیاده سازی می باشد که بیشتر به سخت افزار شبیه است:

```
shift_in = (Sr_En & In) | (~Sr_En & ~q[3])
```

ب) هر خروجی ممکنی این مدار میتواند داشته باشد. چون ورودی یکی یکی به چپ شیفت می‌خورد، پس بر اساس ورودی هر خروجی ای برای این مدار ممکن است.

ج) ماژول تست این مدار را بدین صورت نوشتم که ابتدا سیگنال کنترلی CLR بررسی میشود. سپس اجازه ورود داده با Sr_En = 1 داده میشود. سپس این سگنال not می‌شود و اجازه کارکردی مشابه circular shift داده می‌شود. در نهایت هم سیگنال SET بررسی می‌شود. نتیجه waveform و چاپ خروجی:

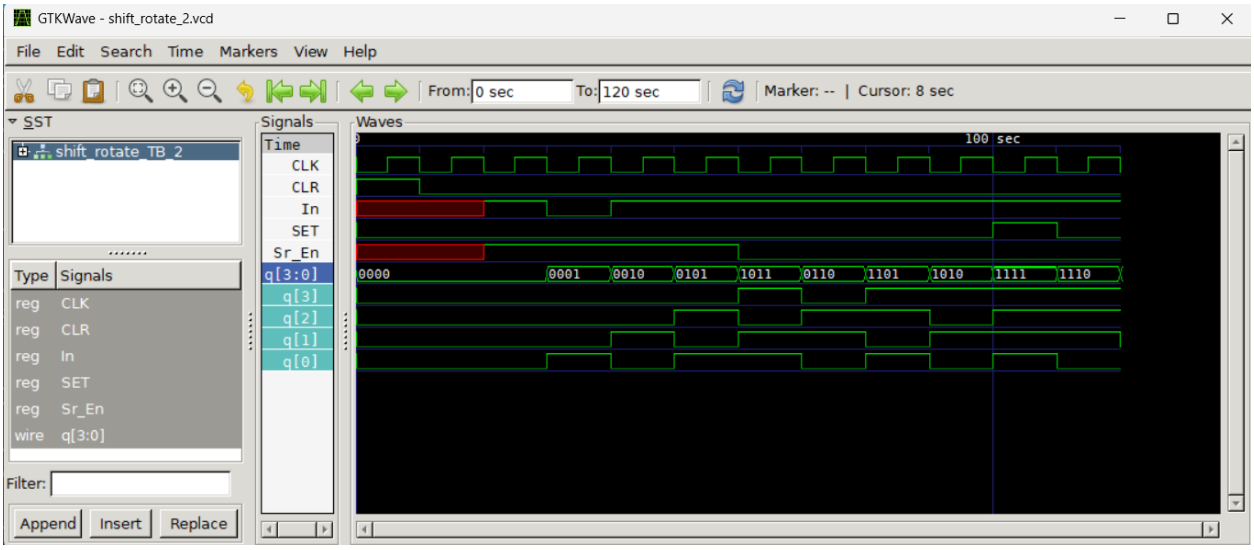



```
PS C:\Users\ASUS\OneDrive\Desktop\university\4\DSD\T1\T1_2\all_quartus_files> vvp shift_rotate.vvp
VCD info: dumpfile shift_rotate.vcd opened for output.
SET = 0 , CLR = 1 , Sr_En = x , In = x  --->  q = 0000
SET = 0 , CLR = 0 , Sr_En = x , In = x  --->  q = 000x
SET = 0 , CLR = 0 , Sr_En = 1 , In = 1  --->  q = 00x1
SET = 0 , CLR = 0 , Sr_En = 1 , In = 0  --->  q = 0x10
SET = 0 , CLR = 0 , Sr_En = 1 , In = 1  --->  q = x101
SET = 0 , CLR = 0 , Sr_En = 1 , In = 1  --->  q = 1011
SET = 0 , CLR = 0 , Sr_En = 0 , In = 1  --->  q = 0110
SET = 0 , CLR = 0 , Sr_En = 0 , In = 1  --->  q = 1101
SET = 0 , CLR = 0 , Sr_En = 0 , In = 1  --->  q = 1010
SET = 0 , CLR = 0 , Sr_En = 0 , In = 1  --->  q = 0100
SET = 1 , CLR = 0 , Sr_En = 0 , In = 1  --->  q = 1111
SET = 0 , CLR = 0 , Sr_En = 0 , In = 1  --->  q = 1110
shift_rotate_TB.v:33: $finish called at 120 (1s)
SET = 0 , CLR = 0 , Sr_En = 0 , In = 1  --->  q = 1100
```

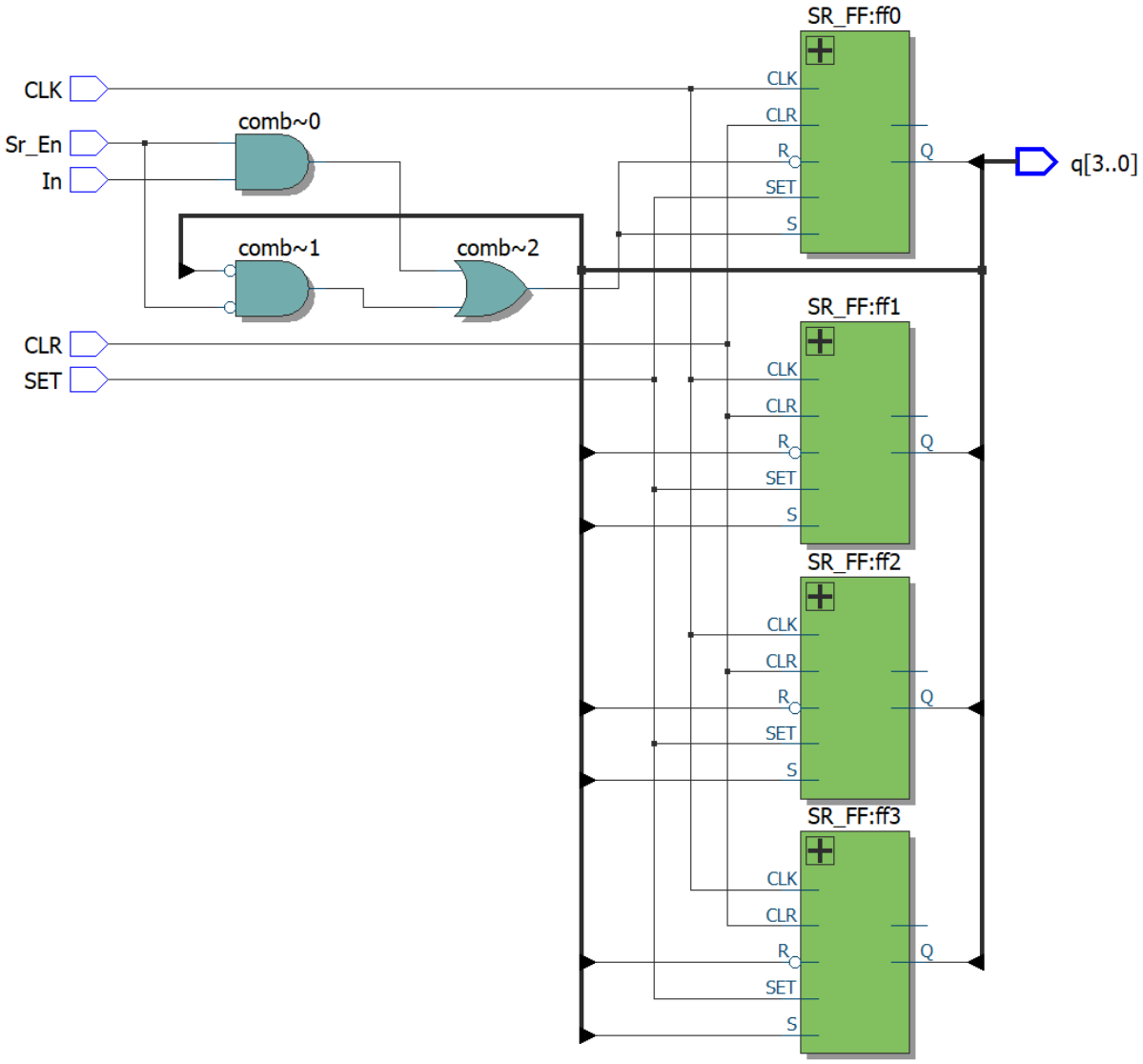
توجه: بعد از اینکه تی ای ها فرمودن که کدی بزنیم که از نظر ساختاری (و نه فقط رفتاری) شبیه سخت افزار سوال باشد، بنده کد و تست دیگری زدم که شامل پیاده سازی SR_FF هم هست.

این نتیجه تست این کد است:

```
PS C:\Users\ASUS\OneDrive\Desktop\university\4\DSD\T1\T1_2\all_quartus_files> iverilog -o shift_rotate_2.vvp SR_FF.v shift_rotate
_2.v shift_rotate_TB_2.v
PS C:\Users\ASUS\OneDrive\Desktop\university\4\DSD\T1\T1_2\all_quartus_files> vvp shift_rotate_2.vvp
VCD info: dumpfile shift_rotate_2.vcd opened for output.
SET = 0 , CLR = 1 , Sr_En = x , In = x  --->  q = 0000
SET = 0 , CLR = 0 , Sr_En = x , In = x  --->  q = 0000
SET = 0 , CLR = 0 , Sr_En = 1 , In = 1  --->  q = 0000
SET = 0 , CLR = 0 , Sr_En = 1 , In = 0  --->  q = 0001
SET = 0 , CLR = 0 , Sr_En = 1 , In = 1  --->  q = 0010
SET = 0 , CLR = 0 , Sr_En = 1 , In = 1  --->  q = 0101
SET = 0 , CLR = 0 , Sr_En = 0 , In = 1  --->  q = 1011
SET = 0 , CLR = 0 , Sr_En = 0 , In = 1  --->  q = 0110
SET = 0 , CLR = 0 , Sr_En = 0 , In = 1  --->  q = 1101
SET = 0 , CLR = 0 , Sr_En = 0 , In = 1  --->  q = 1010
SET = 1 , CLR = 0 , Sr_En = 0 , In = 1  --->  q = 1111
SET = 0 , CLR = 0 , Sr_En = 0 , In = 1  --->  q = 1110
shift_rotate_TB_2.v:33: $finish called at 120 (1s)
SET = 0 , CLR = 0 , Sr_En = 0 , In = 1  --->  q = 1100
```



همانطور که معلوم است، این نتیجه کمی فرق دارد. دلیل آن هم کلاک و پیاده سازی های مربوط به reg و wire در وریلاگ است. خلاصه که هر دو کار مشابهی انجام میدهند ولی از نظر تاخیر، کد دوم بیشتر شبیه به سخت افزار سوال است.



این هم شماتیکی که کوارتوس ساخت. (به نظر شبیه سخت افزار سوال هست :))

.3

در این کد از الگوریتم Booth برای ضرب دو عدد علامت دار استفاده شده است. ایده اصلی این الگوریتم این است که تعداد یک پشت سرهم را می‌توان به صورت تفاضل دو توان 2 نوشت. مثلا عدد 01110 را می‌توان به صورت $2^4-2^1 = 14$ نوشت. پس میتوان با تقسیم بندی یکی از عوامل ضرب در دسته های دو تایی یا سه تایی (booth radix 4) شروطی را تعریف کرد. این جدول مربوط به تقسیم بندی دوتایی بیت های مضروب فیه است.

Yi	Yi-1	Partial Product
0	0	0 * Multiplicand
0	1	1 * Multiplicand
1	0	-1*Multiplicand
1	1	0 * Multiplicand

اما در این سوال از الگوریتم بوث پیشرفته استفاده شده است که ارقام مضروف فیه را در بسته های سه تایی تعیین میکند. یعنی اگر مثلا مضروب فیه را به صورت عدد 8 بیتی $y_7y_6y_5y_4y_3y_2y_1y_0$ درنظر بگیریم، بسته ها چنین می‌شوند:

$cc[0] = y_1y_0 0$, $cc[1] = y_3y_2y_1$, $cc[2] = y_5y_4y_3$, $cc[3] = y_7y_6y_5$

برای این الگوریتم هم جدول زیر را داریم:

Block	Partial product
000	0
001	1 * multiplicand
010	1 * multiplicand
011	2 * multiplicand
100	−2 * multiplicand
101	−1 * multiplicand
110	−1 * multiplicand
111	0

این هم در این قسمت کد پیاده سازی شده است:

```
case(cc[kk])
        3'b001 , 3'b010 : pp[kk] = {x[width-1],x};
        3'b011 : pp[kk] = {x,1'b0};
        3'b100 : pp[kk] = {inv_x[width-1:0],1'b0};
        3'b101 , 3'b110 : pp[kk] = inv_x;
        default : pp[kk] = 0;
endcase
```

در گام بعدی spp که signed pp است، بر حسب جایگاه ارقام، 0، 2، 4 و یا 6 تا به چپ شیفت میخورد. این قسمت را می‌توان به گونه ای دیگر پیاده سازی کرد:

```
spp[kk] = $signed(pp[kk]) <<< (2*kk);
```

در نهایت هم این مقادیر جمع می‌شوند تا حاصل ضرب بدست آید.

اما آیا این کد ایراد دارد یا خیر؟

برای تست کردن این مدار برنامه ای نوشتم تا تمام مقادیر را بررسی کند. اگر خروجی مدار با حاصل ضرب مقادیر آزمون برابر نبود، خطای چاپ کند. نتیجه چنین شد که 140 نمونه، خروجی اشتباه داد (Test finished with 140 errors):

```
ERROR: x=-128, y= -87 (10101001) --> Expected= 11136, Got=  9088 (0010001110000000)
ERROR: x=-128, y= -86 (10101010) --> Expected= 11008, Got= 10496 (0010100100000000)
ERROR: x=-128, y= -82 (10101110) --> Expected= 10496, Got=  9984 (0010011100000000)
ERROR: x=-128, y= -78 (10110010) --> Expected=  9984, Got=  9472 (0010010100000000)
ERROR: x=-128, y= -74 (10110110) --> Expected=  9472, Got=  8960 (0010001100000000)
ERROR: x=-128, y= -72 (10111000) --> Expected=  9216, Got=  7168 (0001110000000000)
ERROR: x=-128, y= -71 (10111001) --> Expected=  9088, Got=  7040 (0001101110000000)
ERROR: x=-128, y= -70 (10111010) --> Expected=  8960, Got=  8448 (0010000100000000)
ERROR: x=-128, y= -66 (10111110) --> Expected=  8448, Got=  7936 (0001111100000000)
ERROR: x=-128, y= -62 (11000010) --> Expected=  7936, Got=  7424 (0001110100000000)
ERROR: x=-128, y= -58 (11000110) --> Expected=  7424, Got=  6912 (0001101100000000)
ERROR: x=-128, y= -56 (11001000) --> Expected=  7168, Got=  5120 (0001010000000000)
ERROR: x=-128, y= -55 (11001001) --> Expected=  7040, Got=  4992 (0001001110000000)
ERROR: x=-128, y= -54 (11001010) --> Expected=  6912, Got=  6400 (0001100100000000)
ERROR: x=-128, y= -50 (11001110) --> Expected=  6400, Got=  5888 (0001011100000000)
ERROR: x=-128, y= -46 (11010010) --> Expected=  5888, Got=  5376 (0001010100000000)
ERROR: x=-128, y= -42 (11010110) --> Expected=  5376, Got=  4864 (0001001100000000)
ERROR: x=-128, y= -40 (11011000) --> Expected=  5120, Got=  3072 (0000110000000000)
ERROR: x=-128, y= -39 (11011001) --> Expected=  4992, Got=  2944 (0000101110000000)
ERROR: x=-128, y= -38 (11011010) --> Expected=  4864, Got=  4352 (0001000100000000)
ERROR: x=-128, y= -34 (11011110) --> Expected=  4352, Got=  3840 (0000111100000000)
ERROR: x=-128, y= -32 (11100000) --> Expected=  4096, Got= -4096 (1111000000000000)
ERROR: x=-128, y= -31 (11100001) --> Expected=  3968, Got= -4224 (1110111110000000)
ERROR: x=-128, y= -30 (11100010) --> Expected=  3840, Got= -4864 (1110110100000000)
ERROR: x=-128, y= -29 (11100011) --> Expected=  3712, Got= -4480 (1110111010000000)
ERROR: x=-128, y= -28 (11100100) --> Expected=  3584, Got= -4608 (1110111000000000)
```

همانطور که مشخص است، در حالتی که x = -128 است، این کد خطا می‌دهد. به ازای چه مقادیری از y؟ به ازای مقادیری که حداقل یکی از دسته های سه تایی ارقامش 100 باشد.

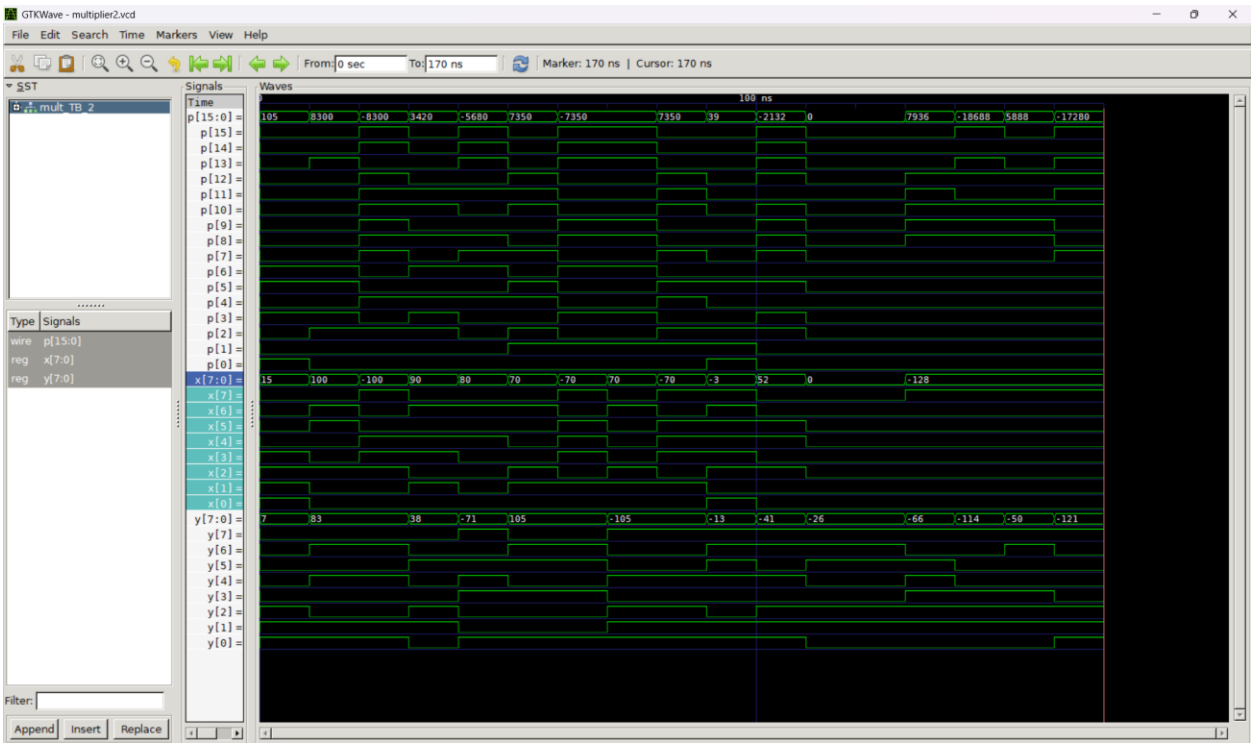
یعنی زمانی که در switch case به حالت زیر می‌رود:

3’b100 : pp[kk] = {inv_x[width-1:0],1'b0}

بنابراین وقتی x = -128 می‌شود و y در دسته های سه بیتی 100 دارد، خروجی نادرست چاپ می‌شود. (مثلا y = 01**100**101 or y = 01110**10**)

بنده یک فایل تست دیگر هم نوشتم. نتیجه تست به 3 صورت قابل مشاهده است:

```
VCD info: dumpfile multiplier2.vcd opened for output.
x =   15, y =    7 --> p =    105
x =  100, y =   83 --> p =   8300
x = -100, y =   83 --> p =  -8300
x =   90, y =   38 --> p =   3420
x =   80, y =  -71 --> p =  -5680
x =   70, y =  105 --> p =   7350
x =  -70, y =  105 --> p =  -7350
x =   70, y = -105 --> p =  -7350
x =  -70, y = -105 --> p =   7350
x =   -3, y =  -13 --> p =     39
x =   52, y =  -41 --> p =  -2132
x =    0, y =  -26 --> p =      0

 These are wrong answers:
x = -128, y =  -66 --> p =   7936
x = -128, y = -114 --> p = -18688
x = -128, y =  -50 --> p =   5888
mult_TB_2.v:36: $finish called at 160000 (1ps)
x = -128, y = -121 --> p = -17280
```



فایل vcd هم به پیوست ارسال شد.