به نام خدا

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

امیرمحمد شربتی – 402106112

1404/1/31



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];

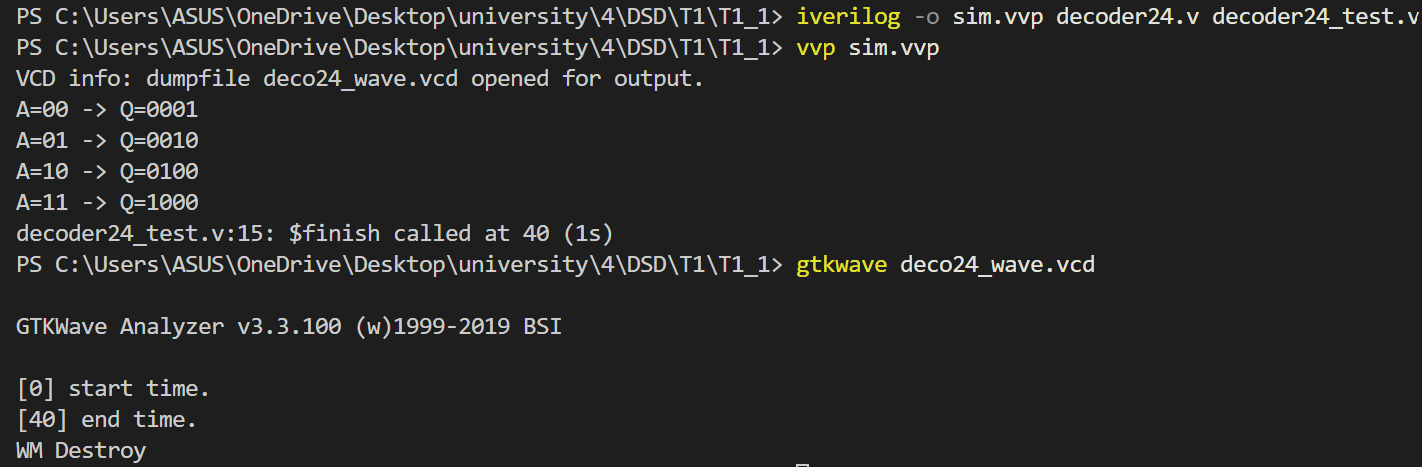
assign Q[3] = A[0] & A[1];

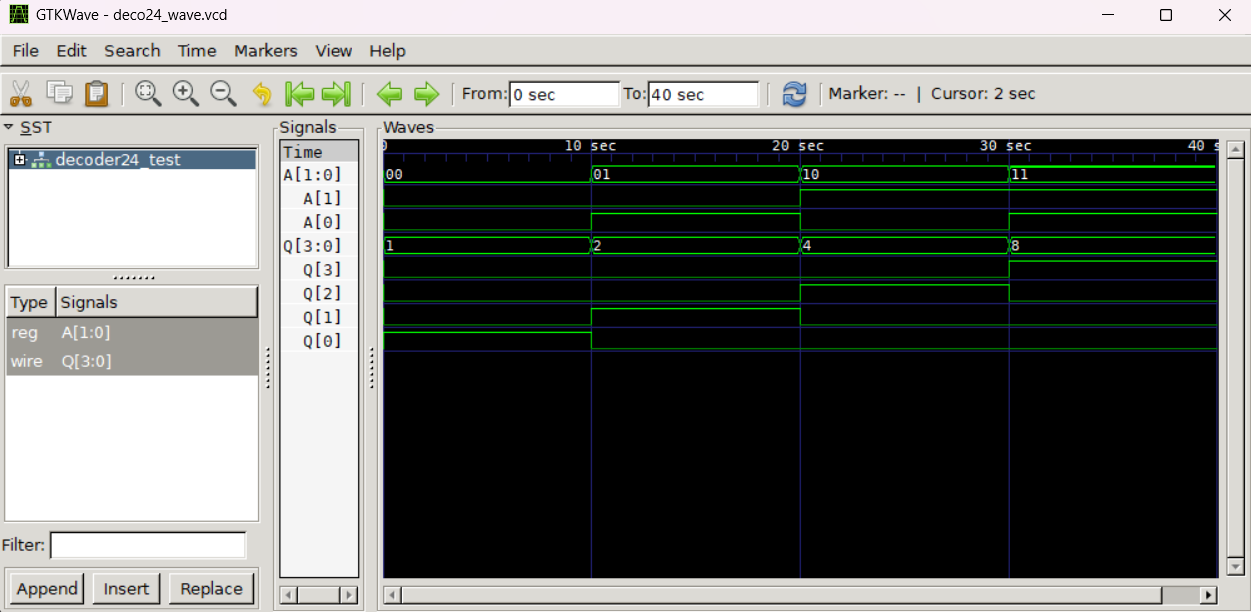
پس از آن ماژول تستی برای این دیکودر می‌نویسیم. فقط اینکه بنده هم از Quartus و هم از Icarus استفاده می‌کنم. دو خط در test bench اضافه می‌کنم که برای مشاهده waveform به کمک Icarus و gtk است:

$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.





در گام بعد از این decoder 2\*4 یک decoder 3\*8 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;

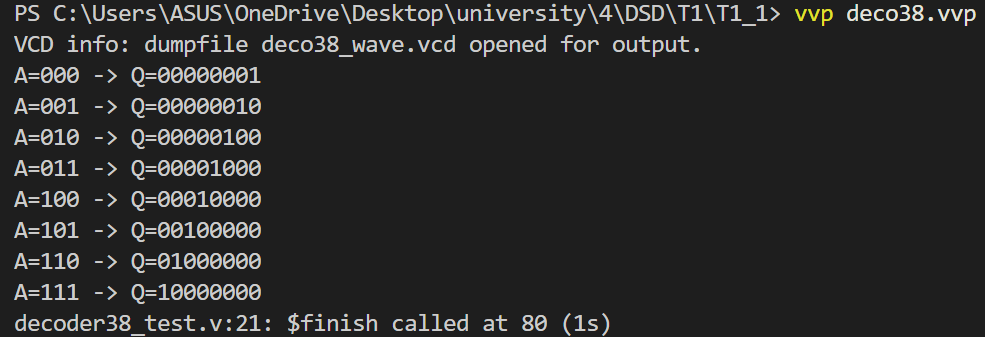
decoder24 deco(A[1:0], QQ);

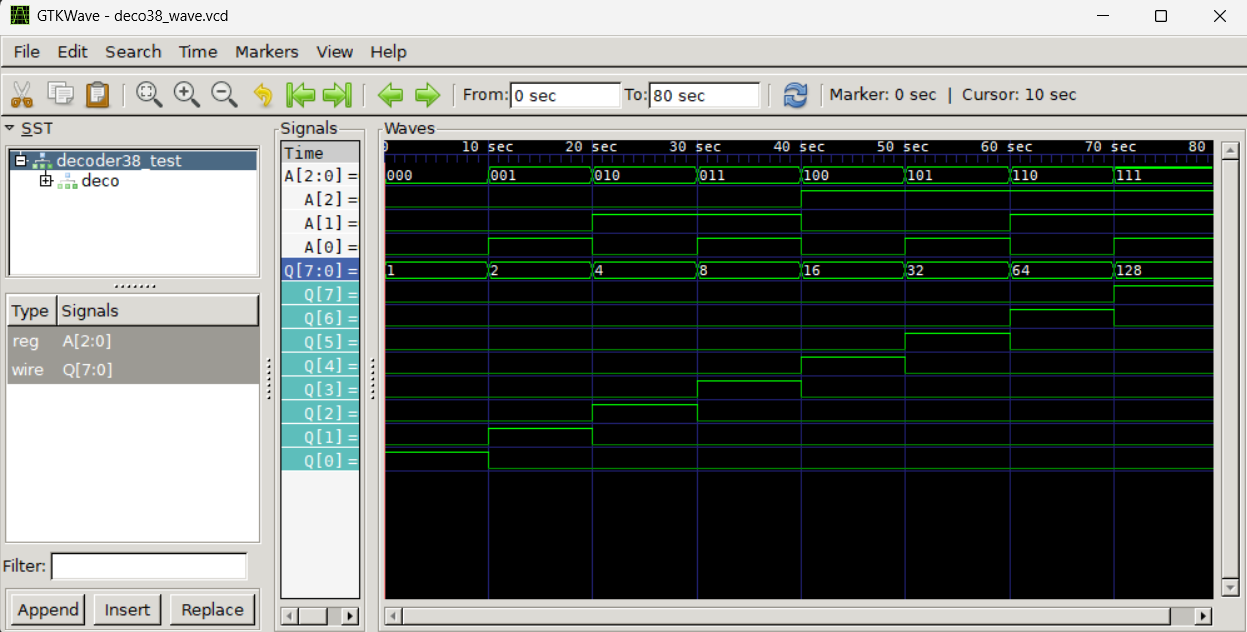
assign Q = A[2] ? {QQ, 4'b0000} : {4'b0000, QQ};

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...

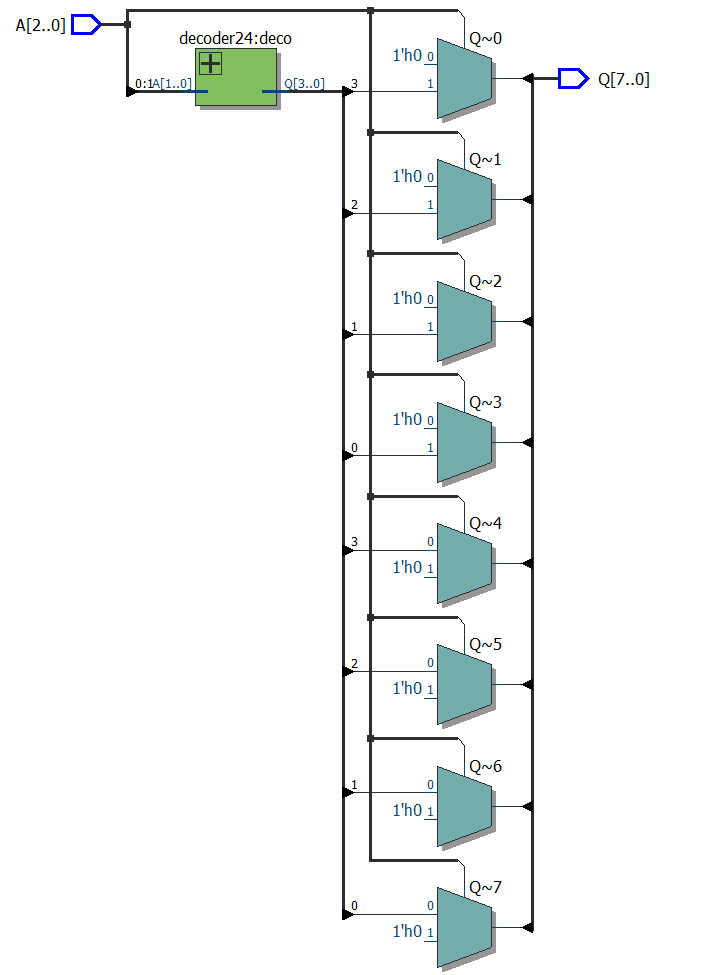
خلاصه که برای پیاده سازی این روش های متفاوتی وجود دارد ولی بنده از روش ternary یا همان mux استفاده کردم.

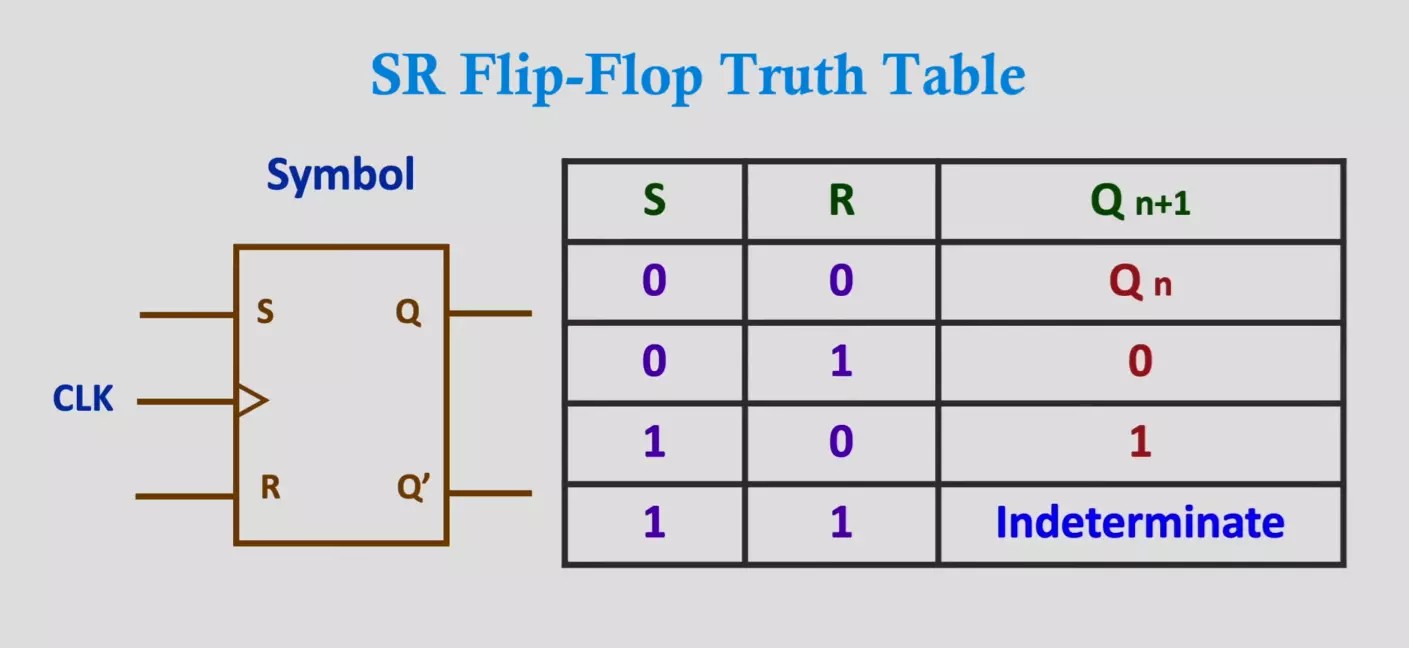
این هم نتیجه تست مدار :

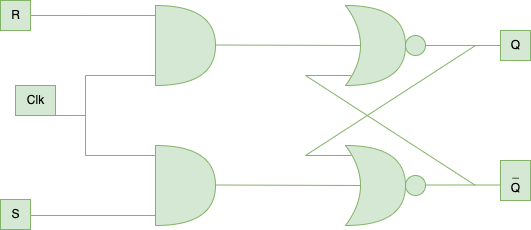




این هم شماتیکی ست که کوارتوس ترسیم کرده:



1. 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 Qn+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};

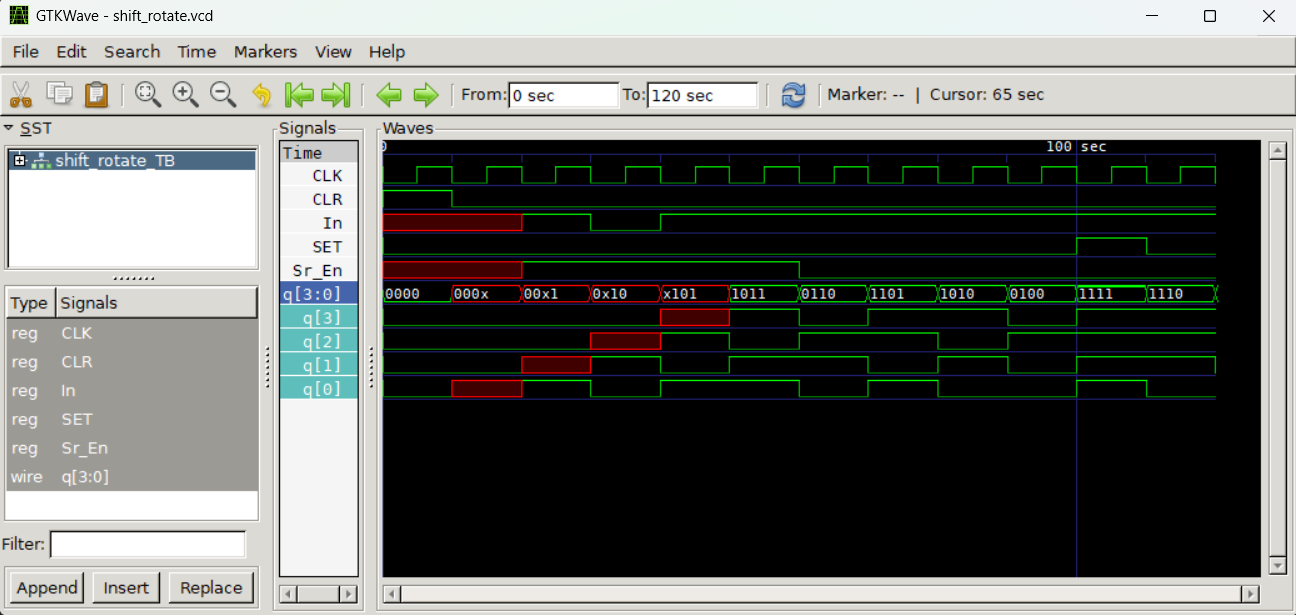
end

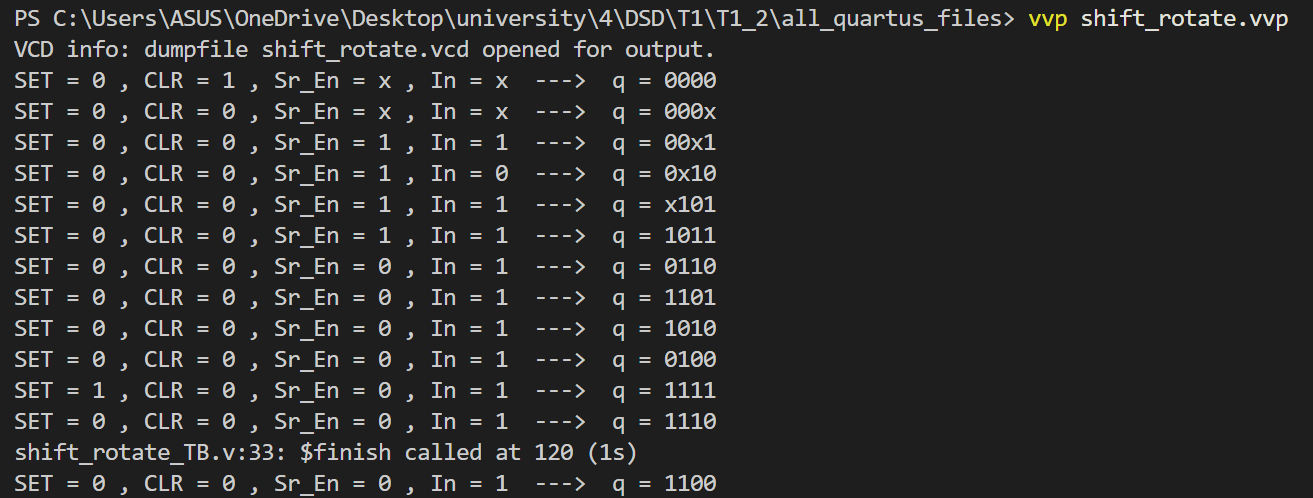
آسنکرون بودن سیگنال های کنترلی در نظر گرفته شده. خط اول هم به این شیوه قابل پیاده سازی می باشد که بیشتر به سخت افزار شبیه است:

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.

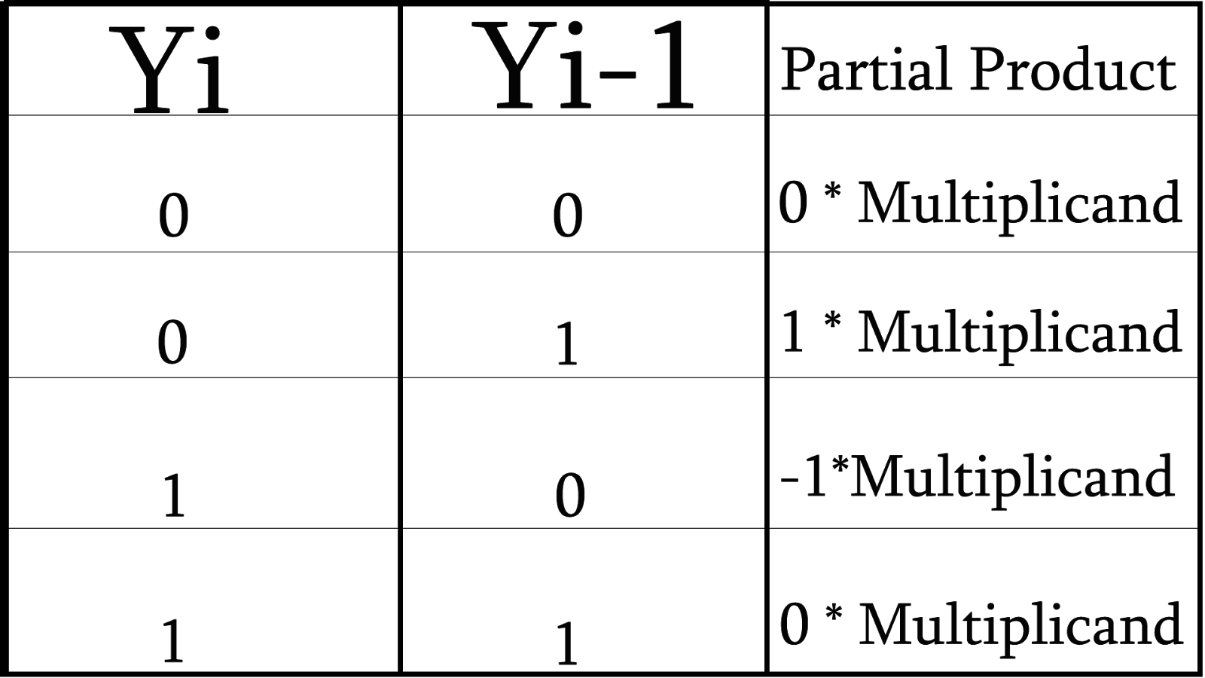
ج) ماژول تست این مدار را بدین صورت نوشتم که ابتدا سیگنال کنترلی CLR بررسی میشود. سپس اجازه ورود داده با Sr\_En = 1 داده میشود. سپس این سگنال not می‌شود و اجازه کارکردی مشابه circular shift داده می‌شود. در نهایت هم سیگنال SET بررسی می‌شود. نتیجه waveform و چاپ خروجی:







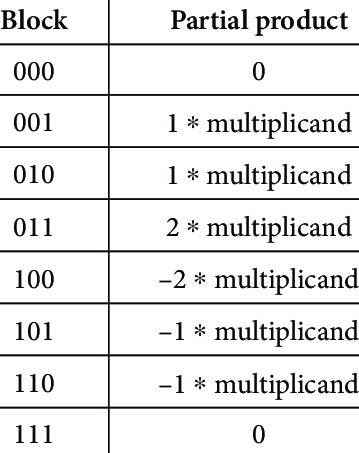
در این کد از الگوریتم Booth برای ضرب دو عدد علامت دار استفاده شده است. ایده اصلی این الگوریتم این است که تعداد یک پشت سرهم را می‌توان به صورت تفاضل دو توان 2 نوشت. مثلا عدد 01110 را می‌توان به صورت 24-21 = 14 نوشت. پس میتوان با تقسیم بندی یکی از عوامل ضرب در دسته های دو تایی یا سه تایی (booth radix 4) شروطی را تعریف کرد. این جدول مربوط به تقسیم بندی دوتایی بیت های مضروب فیه است.



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 y7y6y5y4y3y2y1y0 درنظر بگیریم، بسته ها چنین می‌شوند:

cc[0] = y1y00 , cc[1] = y3y2y1 , cc[2] = y5y4y3 , cc[3] = y7y6y5

برای این الگوریتم هم جدول زیر را داریم:



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

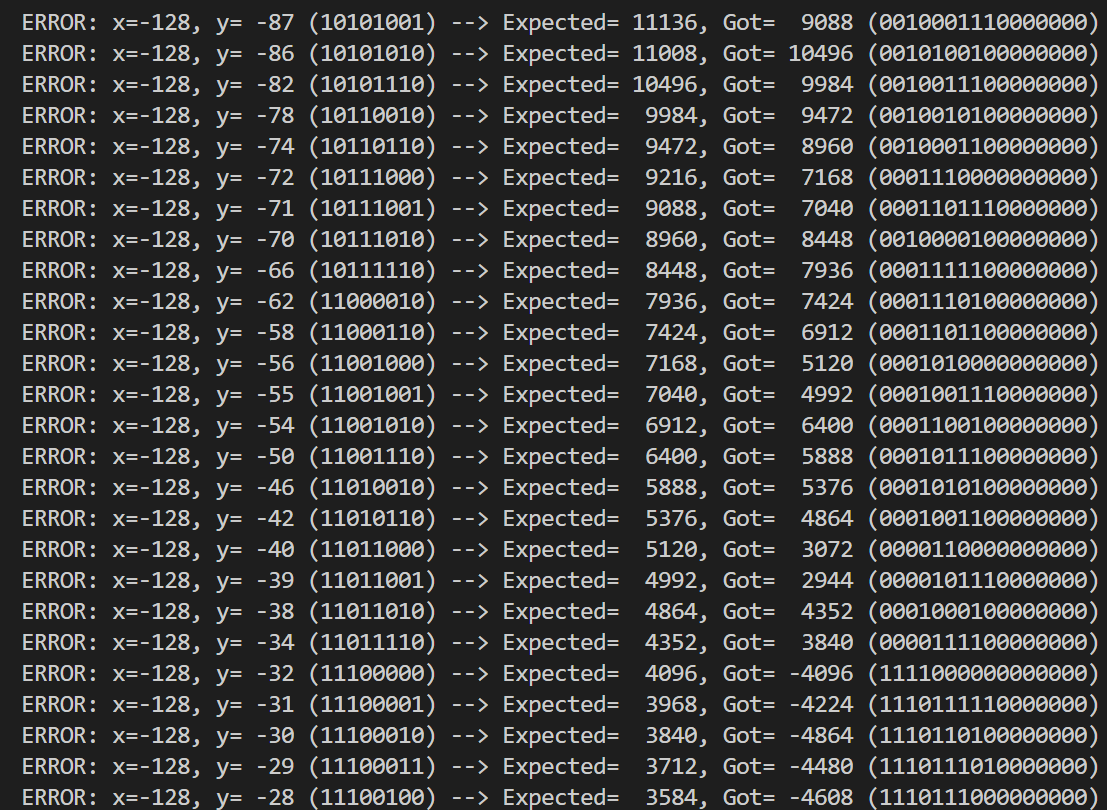
در گام بعدی spp که signed pp است، بر حسب جایگاه ارقام، 0، 2، 4 و یا 6 تا به چپ شیفت میخورد. این قسمت را می‌توان به گونه ای دیگر پیاده سازی کرد:

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

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

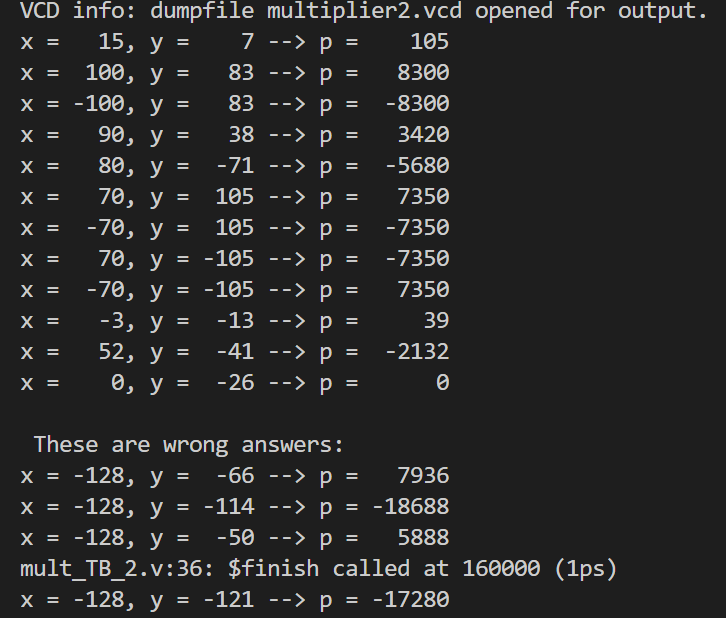
اما آیا این کد ایراد دارد یا خیر؟

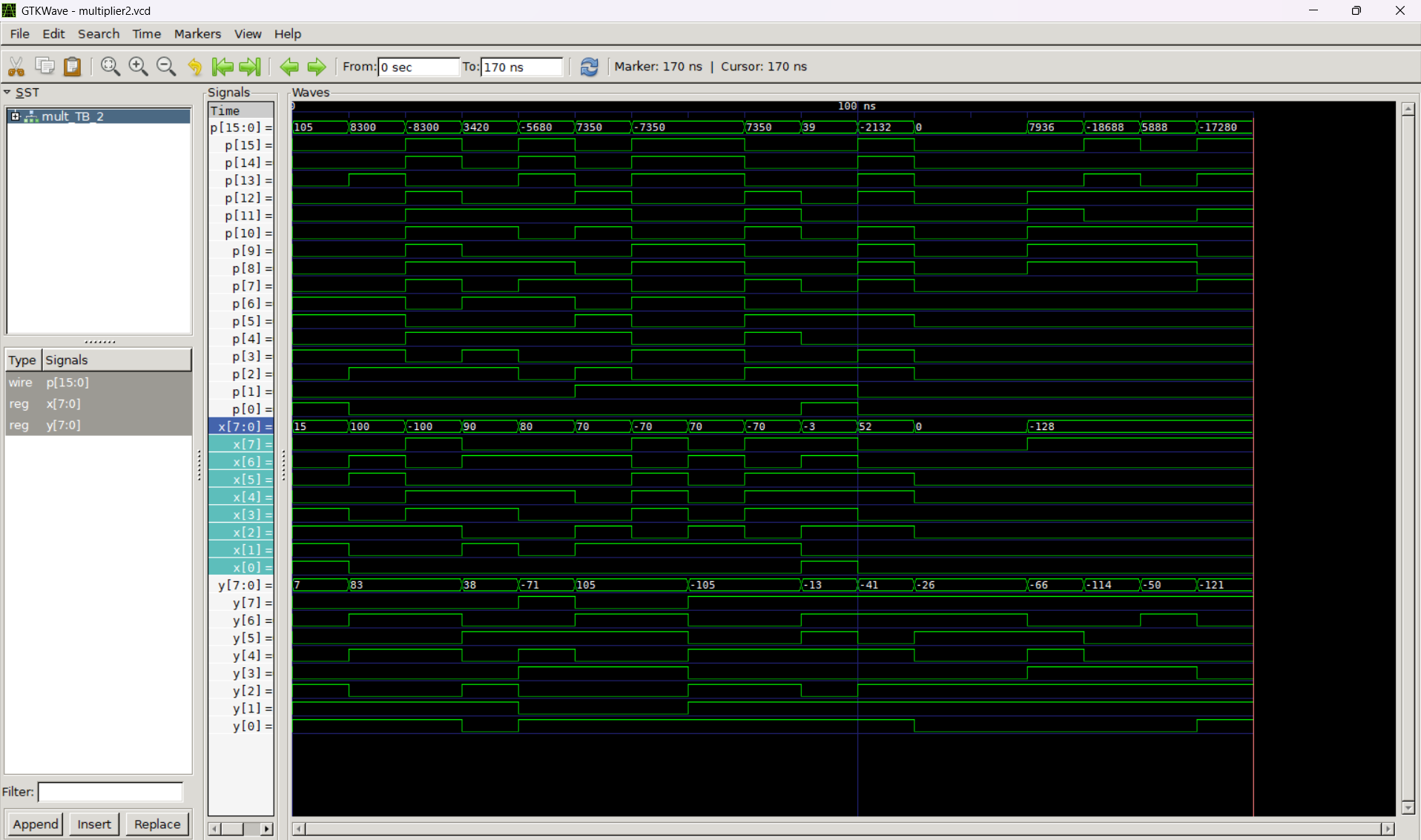
برای تست کردن این مدار برنامه ای نوشتم تا تمام مقادیر را بررسی کند. اگر خروجی مدار با حاصل ضرب مقادیر آزمون برابر نبود، خطای چاپ کند. نتیجه چنین شد که 140 نمونه، خروجی اشتباه داد (Test finished with 140 errors):



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 = 011101**10**)

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




فایل vcd هم به پیوست ارسال شد.