به نام خدا

توضیحات تمرین دوم درس طراحی سیستم های دیجیتال – دکتر فصحتی

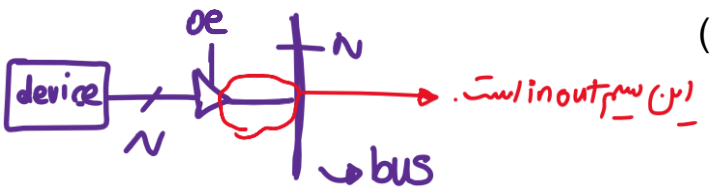
امیرمحمد شربتی ناوان 402106112

1404/2/30

1. در این سوال باید یک گذرگاه مشترک ایجاد کنیم. این کار را با دو روش متفاوت انجام میدهیم. یک بار به کمک بافر سه‌حالته و یک بار هم به کمک تسهیم کننده. ابتدا به بافر سه‌حالته می‌پردازیم:

ابتدا یک tri-state buffer طراحی می‌کنیم. در ابتدا یک parameter تعریف میکنیم که مقدار پیش فرض آن 8 است. این مقدار نشان دهنده سایز بافر ما هست. یعنی قادر به عبور N می‌شود. طبیعتا ورودی N بیتی میخواهیم و یک سیگنال کنترلی به نام oe (output enable). اگر این سیگنال یک باشد، خروجی همان سیگنال ورودی است. در غیر این صورت مقدار z در آن قرار میگیرد.

این سیگنال خروجی از جنس output نیست. بلکه از جنس inout است. در وریلاگ inout port ها به اصطلاح دو طرفه یا همان bidirectional هستند. در اینجا ما میخواهیم هم بتوانیم دیتای ورودی را به گذرگاه منتقل کنیم (یعنی bi_date باید در نقش خروجی عمل کند) و هم بعضی مواقع میخواهیم روی device چیزی بنویسیم. (یعنی محتویات گذرگاه را به دستگاه منتقل کنیم). پس نیازمند یک wire دوطرفه هستیم. (سیمی که به bus وصل می‌شود دوطرفه است.)



بعد از اینکه از استاد در مورد استفاده از تری استیت آماده وریلاگ پرسیدم، آز آن استفاده کردم و از این tri-state ای که طراحی کردم، استفاده نکردم.

در فایل دوم یک کنترلر (bus_controller) برای این گذرگاه ساختم. چون شکل سوال شامل دو دستگاه است، پس کنترلر هم فقط برای کنترل دو دستگاه پیاده سازی شد. در ورودی این ماژول، دو سیم برای request دو دستگاه داریم. یعنی دستگاه ها برای بارگذاری اطلاعات خود بر گذرگاه درخواست میدهند. دو wire هم داریم که کلاک و ریست هستند. دو خروجی داریم که اجازه دسترسی به دستگاه را صادر و کنترل میکنند. اگر reset نباشد، اولویت با درخواست دستگاه اول است. grant1 = 1 , grant2 = 0 می‌شود. برای درخواست دستگاه دوم هم همین است. و اگر هیچ کدام درخواستی نداشتند، grant برای هر دو صفر میشود.

به فایل اصلی می‌رسیم. به کمک این دو فایل گذرگاه مشترک با بافر سه‌حالته را می‌سازیم. ورودی این ماژول شامل متغیر های زیر است:

بعد از متغیر های کلاک و ریست، data_in_x میگوید دستگاه x میخواهد چه داده ای روی bus قرار دهد. reqX یعنی دستگاه X می‌خواهد روی باس داده را قرار دهد. Data_out_x هم داده ای است که دستگاه x از روی گذرگاه می‌خواند.

گذرگاه مشترک همان bi_data است. g1 و g2 برای کنترل استفاده میشوند.

در خطوط بعد از ماژول هایی که داشتیم استفاده میکنیم. ابتدا کنترلر را instantiate میکنیم. خروجی grant را در g1 , g2 می‌ریزیم. از همین سیگنال ها در دو tri-state buffer ای که بعدا instantiate میکنیم، استفاده میکنیم. حداکثر یکی از این بافر های سه حالته فعال میشود و داده خود را روی باس قرار میدهد. دقیقا چیزی که از باس با بافر سه حالته انتظار داریم. در نهایت هم دیتای باس به عنوان خروجی device ها نشان داده میشود.

ماژول bus_with_mux مربوط به طراحی باس با mux است.

برای طراحی به کمک mux میتوان از این دستور ternary condition استفاده کرد:

```
assign bus = (sel == 1'b0) ? data_in_1 : data_in_2;
```

اما چون می‌خواهیم تاخیر ها را به درستی حساب کنیم، شاید روش gate level modeling روش بهتری باشد. پس منطق این خط را به این خط که در واقع پیاده سازی داخلی mux است، تغییر دادم:

```
assign bus = (~sel & data_in_1) | (sel & data_in_2);
```

برای بدست آوردن و استفاده کردن از تاخیر، باید این منطق به طراحی gate level تبدیل شود. در هنگام بررسی تاخیر ها، کد این بخش هم در گزارش قرار دادم.

بدین ترتیب منطق bus راحت بدست آمد. همین bus را در خطوط بعد به data_out مربوط به دو دستگاه منتسب می‌کنیم.

در ماژول های تست هم بنده سعی کردم حالات مختلف را بررسی کنم.

2) در اینجا میخواهم در مورد تاخیر صحبت کنم. **(بنده فرض کردم اعداد چپ rise است)**

اول اینکه چون تمام بیت ها به صورت موازی اجرا می‌شوند، پس تاخیر برای N های مختلف ثابت است. با تغییر N در این کد میتوان به راحتی این را بررسی کرد.

برای پیاده سازی تاخیردار این مدار، همانطور که قبلا گفته شد، پیاده سازی به سمت gate level modeling تغییر پیدا کرد. ابتدا طراحی ب کمک mux را بررسی کنیم.

این اصل کد مربوط به این قسمت است:

```
not #(not_max_rise: not_typ_rise: not_min_rise,
      not_max_fall: not_typ_fall: not_min_fall) (not_select, select);
  genvar i;
  generate
   for (i = 0; i < N; i = i + 1) begin : mux_bits
    wire a1, a2;
    and #(and_max_rise: and_typ_rise: and_min_rise,
        and_max_fall: and_typ_fall: and_min_fall) (a1, not_select, data_in_1[i]);
    and #(and_max_rise: and_typ_rise: and_min_rise,
        and_max_fall: and_typ_fall: and_min_fall) (a2, select, data_in_2[i]);
    or  #(or_max_rise: or_typ_rise: or_min_rise,
        or_max_fall: or_typ_fall: or_min_fall) (bus[i], a1, a2);
```

```
    end

  endgenerate
```

در واقع برای دیدن تاخیر باید از سینتکس () ()# or مثلا برای or استفاده کرد. (البته میتوان نام آن را هم مشخص کرد.) در پرانتز دوم پین های ورودی و خروجی گیت ست می‌شوند. اما در پرانتز اول و بعد از # تاخیر ها به ترتیبی که باید داده میشوند. Rise, fall, turnoff و در حالت max:typ:min ست میشوند. برای طراحی بهتر بنده از پارامتر استفاده کردم. (در ابتدا فکر کردم ممکن است این تاخیر ها در چند ماژول استفاده شوند و اینها را در یک ماژول جدا قرار دادم، ولی بعدا دیدم برای این سوال فقط در یک ماژول استفاده میشوند.)

برای دیدن تاخیر در حالات مختلف، بنده که با icarus کد را ران میکنم، میتوان از فلگ های Tmin- و.. استفاده کرد.

max

```
t=0 select=0 bus=xx dout1=xx dout2=xx
t=11 select=0 bus=XX dout1=xx dout2=XX
t=17 select=0 bus=a2 dout1=xx dout2=a2
t=20 select=1 bus=a2 dout1=xx dout2=a2
t=33 select=1 bus=be dout1=be dout2=a2
t=34 select=1 bus=9c dout1=9c dout2=a2
t=40 select=0 bus=9c dout1=9c dout2=a2
t=51 select=0 bus=00 dout1=9c dout2=00
t=57 select=0 bus=a2 dout1=9c dout2=a2
```

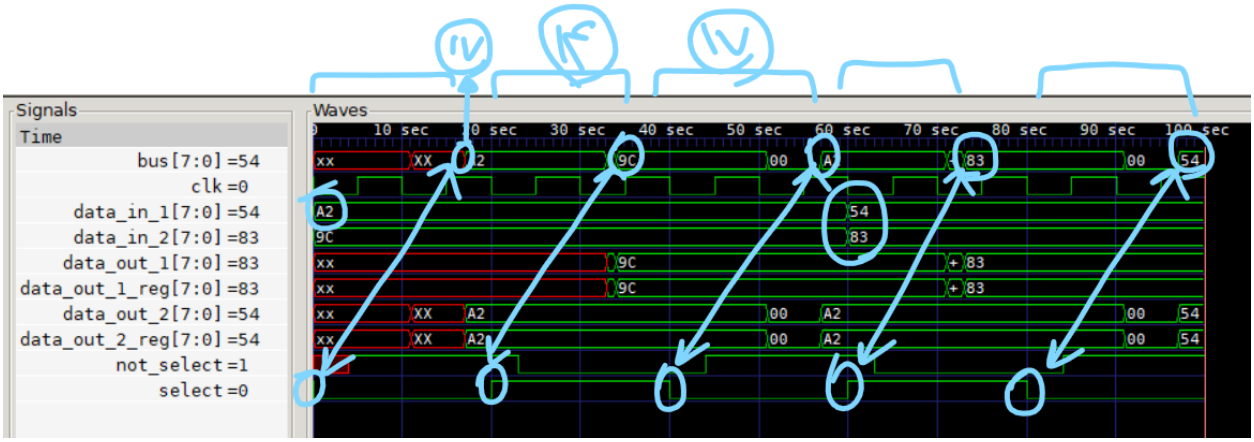
۱۷ = ۴ + ۶ + ۷

۱۴ = ۳ + ۵ + ۶

۱۷

t=60 select=1 bus=a2 dout1=9c dout2=a2

... (چند خط هم خروجی داریم)



typical

t=0 select=0 bus=xx dout1=xx dout2=xx

t=9 select=0 bus=XX dout1=xx dout2=XX

t=14 select=0 bus=a2 dout1=xx dout2=a2

t=20 select=1 bus=a2 dout1=xx dout2=a2

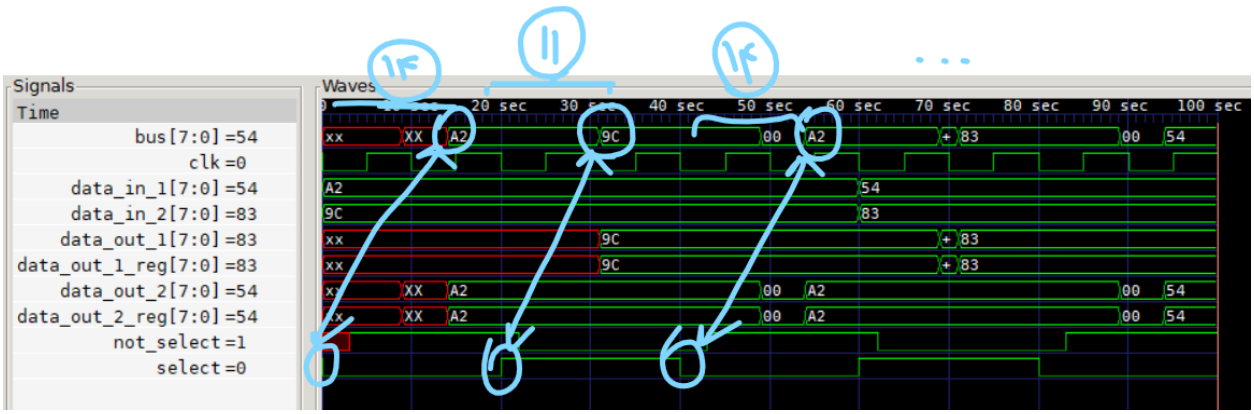
t=31 select=1 bus=9c dout1=9c dout2=a2

t=40 select=0 bus=9c dout1=9c dout2=a2

t=49 select=0 bus=00 dout1=9c dout2=00

t=54 select=0 bus=a2 dout1=9c dout2=a2

t=60 select=1 bus=a2 dout1=9c dout2=a2



Min

t=0 select=0 bus=xx dout1=xx dout2=xx

t=7 select=0 bus=XX dout1=xx dout2=XX

t=11 select=0 bus=a2 dout1=xx dout2=a2

t=20 select=1 bus=a2 dout1=xx dout2=a2

t=28 select=1 bus=80 dout1=80 dout2=a2

t=29 select=1 bus=9c dout1=9c dout2=a2

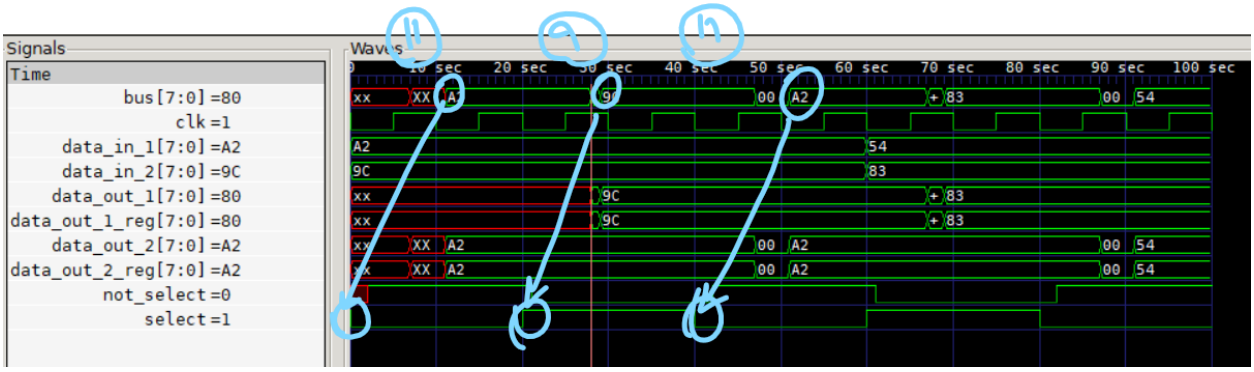
t=40 select=0 bus=9c dout1=9c dout2=a2

t=47 select=0 bus=00 dout1=9c dout2=00

t=51 select=0 bus=a2 dout1=9c dout2=a2

t=60 select=1 bus=a2 dout1=9c dout2=a2

به نظرم باید تاخیر میانی طبق داده های جدول 8 میشد. اما خب خروجی این را نشان میدهد. و لو اینکه در 8 واحد بعد یک دیتا غلط وارد باس میشود.



همانطور که قبلا هم توضیح داده شد تاخیر برای تمام N ها برابر است، زیرا تمام بیت ها به صورت موازی عملیات ها را انجام میدهند. برای دیدن شهودی هم می‌توان N را تغییر داد و نتیجه را به راحتی دید.

برای tri-state هم مشابها داریم:

```
genvar i;
generate
   for (i = 0; i < N; i = i + 1) begin : ts_buf_gen_1
      bufif1 #(tri_min_rise: tri_typ_rise: tri_max_rise,
            tri_min_fall: tri_typ_fall: tri_max_fall,
            tri_min_turnoff: tri_typ_turnoff : tri_max_turnoff)
            ts_buffer1 (bi_data[i], data_in_1[i], g1);
   end
endgenerate
```

در اینجا هم هر دو generator موازی اجرا میشوند. این اجرا برای تمام بیت ها یکسان است. در واقه به ازای تعداد بیت مختلف، تاخیر فرقی نمیکند.

نتیجه تست ها:

max

t=0 g1=0 g2=0 bus=xx din1=aa din2=55 dout1=xx dout2=xx

t=6 g1=0 g2=0 bus=zz din1=aa din2=55 dout1=xx dout2=xx

t=25 g1=1 g2=0 bus=zz din1=aa din2=55 dout1=xx dout2=xx

t=31 g1=1 g2=0 bus=ZZ din1=aa din2=55 dout1=xx dout2=ZZ

t=32 g1=1 g2=0 bus=aa din1=aa din2=55 dout1=xx dout2=aa

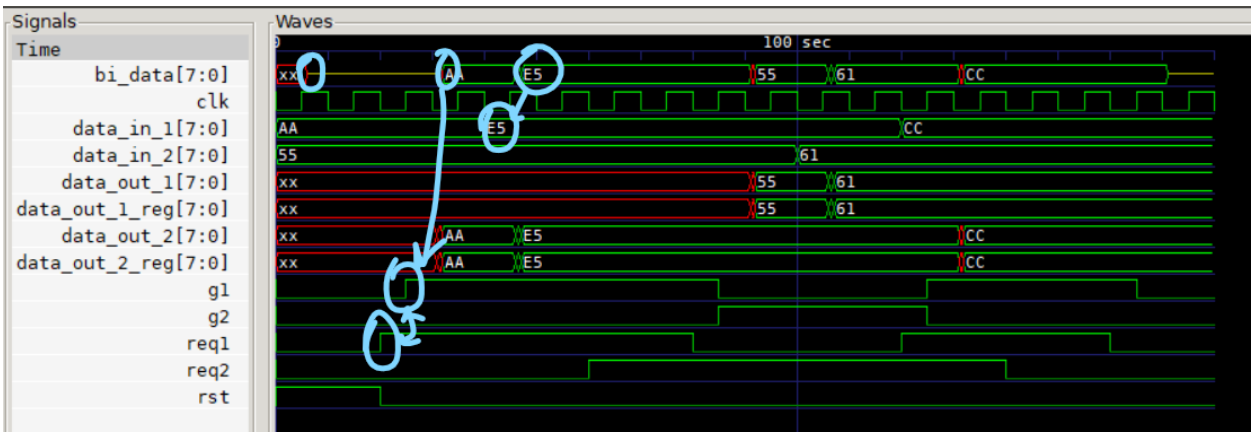
t=40 g1=1 g2=0 bus=aa din1=e5 din2=55 dout1=xx dout2=aa

t=46 g1=1 g2=0 bus=a0 din1=e5 din2=55 dout1=xx dout2=a0

t=47 g1=1 g2=0 bus=e5 din1=e5 din2=55 dout1=xx dout2=e5

t=85 g1=0 g2=1 bus=e5 din1=e5 din2=55 dout1=xx dout2=e5

(تعداد خطوط خروجی بیشتر است) .....



typical

t=0 g1=0 g2=0 bus=xx din1=aa din2=55 dout1=xx dout2=xx

t=5 g1=0 g2=0 bus=zz din1=aa din2=55 dout1=xx dout2=xx

t=25 g1=1 g2=0 bus=zz din1=aa din2=55 dout1=xx dout2=xx

t=30 g1=1 g2=0 bus=ZZ din1=aa din2=55 dout1=xx dout2=ZZ

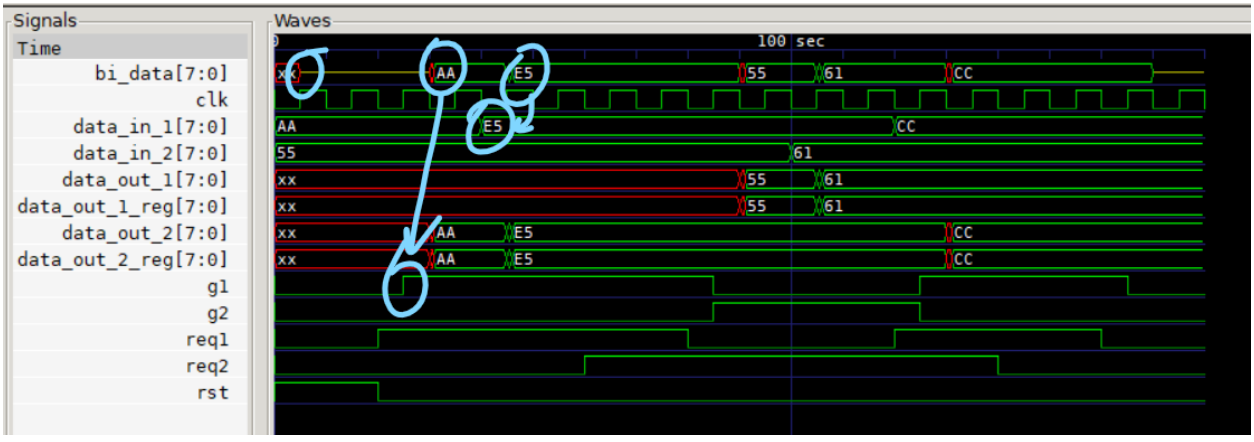
t=31 g1=1 g2=0 bus=aa din1=aa din2=55 dout1=xx dout2=aa

t=40 g1=1 g2=0 bus=aa din1=e5 din2=55 dout1=xx dout2=aa

t=45 g1=1 g2=0 bus=a0 din1=e5 din2=55 dout1=xx dout2=a0

t=46 g1=1 g2=0 bus=e5 din1=e5 din2=55 dout1=xx dout2=e5

t=85 g1=0 g2=1 bus=e5 din1=e5 din2=55 dout1=xx dout2=e5



Min

t=0 g1=0 g2=0 bus=xx din1=aa din2=55 dout1=xx dout2=xx

t=4 g1=0 g2=0 bus=zz din1=aa din2=55 dout1=xx dout2=xx

t=25 g1=1 g2=0 bus=zz din1=aa din2=55 dout1=xx dout2=xx

t=29 g1=1 g2=0 bus=ZZ din1=aa din2=55 dout1=xx dout2=ZZ

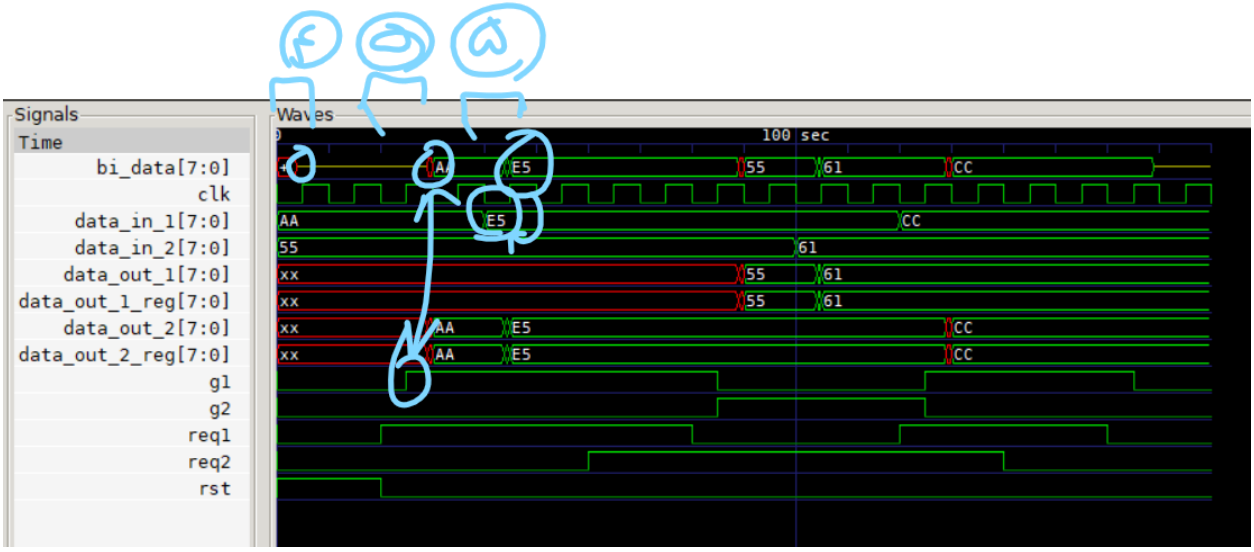
t=30 g1=1 g2=0 bus=aa din1=aa din2=55 dout1=xx dout2=aa

t=40 g1=1 g2=0 bus=aa din1=e5 din2=55 dout1=xx dout2=aa

t=44 g1=1 g2=0 bus=a0 din1=e5 din2=55 dout1=xx dout2=a0

t=45 g1=1 g2=0 bus=e5 din1=e5 din2=55 dout1=xx dout2=e5

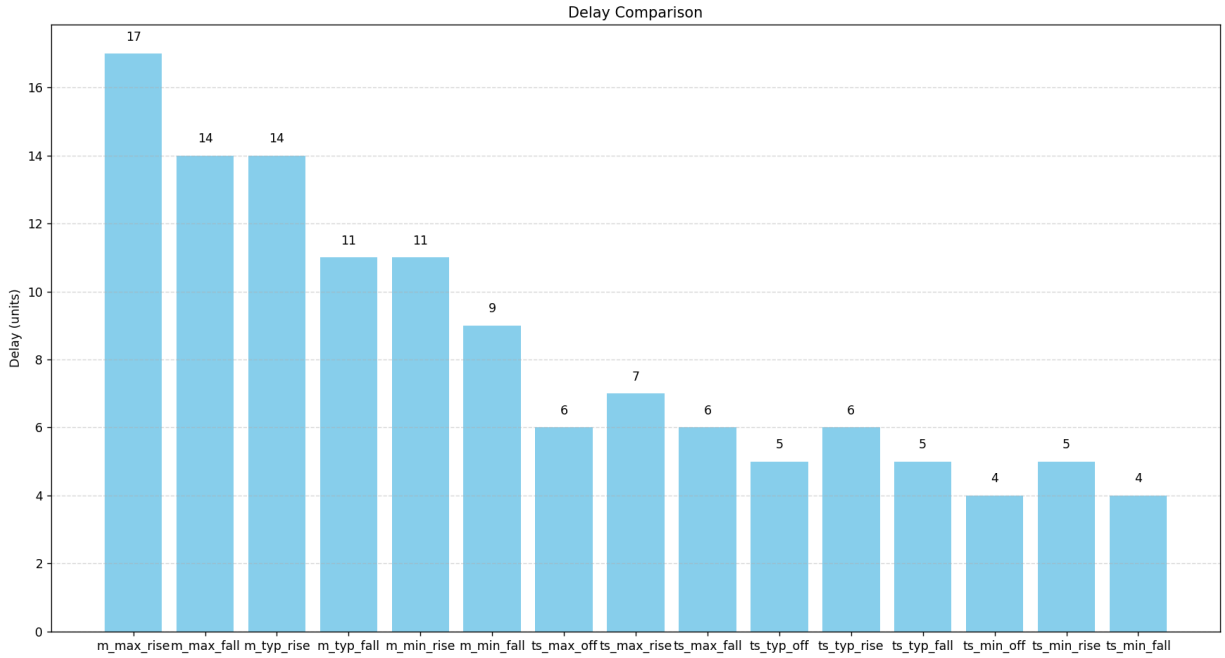
t=85 g1=0 g2=1 bus=e5 din1=e5 din2=55 dout1=xx dout2=e5



کاملا مشخص است که تاخیر mux بیشتر از tri-state است. چون در تری استیت بافر فقط یک گیت تعیین کننده است ولی در mux، دو الی سه گیت که تقریبا تاخیری شبیه به tri-state دارند، در نتیجه نهایی نقش دارند. (توجه شود بنده صرفا تغییر بافر را لحاظ کردم. در روش tristate چون کنترلر هم داریم، ممکن است با در نظر گرفتن کلاک طولانی تر، تاخیر درخواست دادن دستگاه و دیدن خروجی در دستگاه دیگر، بیشتر شود.)

3) و حالا نمودار تاخیر ها:

بنده به کمک کد پایتون (که البته به کمک gpt پیاده سازی شد) نمودار میله ای برخی مقادیر را مشخص کردم. این کد هم به پیوست ارسال میشود (اول اسامی کاملی برای label ها گذاشتم. ولی متاسفانه دیدم قاطی میشوند و در نمودار label ها روی هم می‌روند، به همین خاطر اسامی را کوتاه کردم):



همانطور که قبلا هم بار ها عرض کردم، همانطور که مشخص است، چون تعداد گیت های mux بیشتر است و طبق این تاخیر های داده شده، tristate تاخیر کمتری دارد. حدودا تاخیر یک گیت and با tristate برابر است، در حالی که در روش mux ، or , not هم داریم که در تاخیر دخیل هستند. البته اگر تاخیر کنترلر را به خاطر کلاک هم در نظر بگیریم، تاخیر در روش tri-state بیشتر هم میشود. ولی اینجا فقط تاخیر ناشی از گیت در نظر گرفته شد. همچینین نتیجه برای N های مختلف هم فرقی ندارد...