گزارش کار تمرین پیاده سازی 1 درس DSD دکتر اجلالی

امیرمحمد کوشکی، 400109673

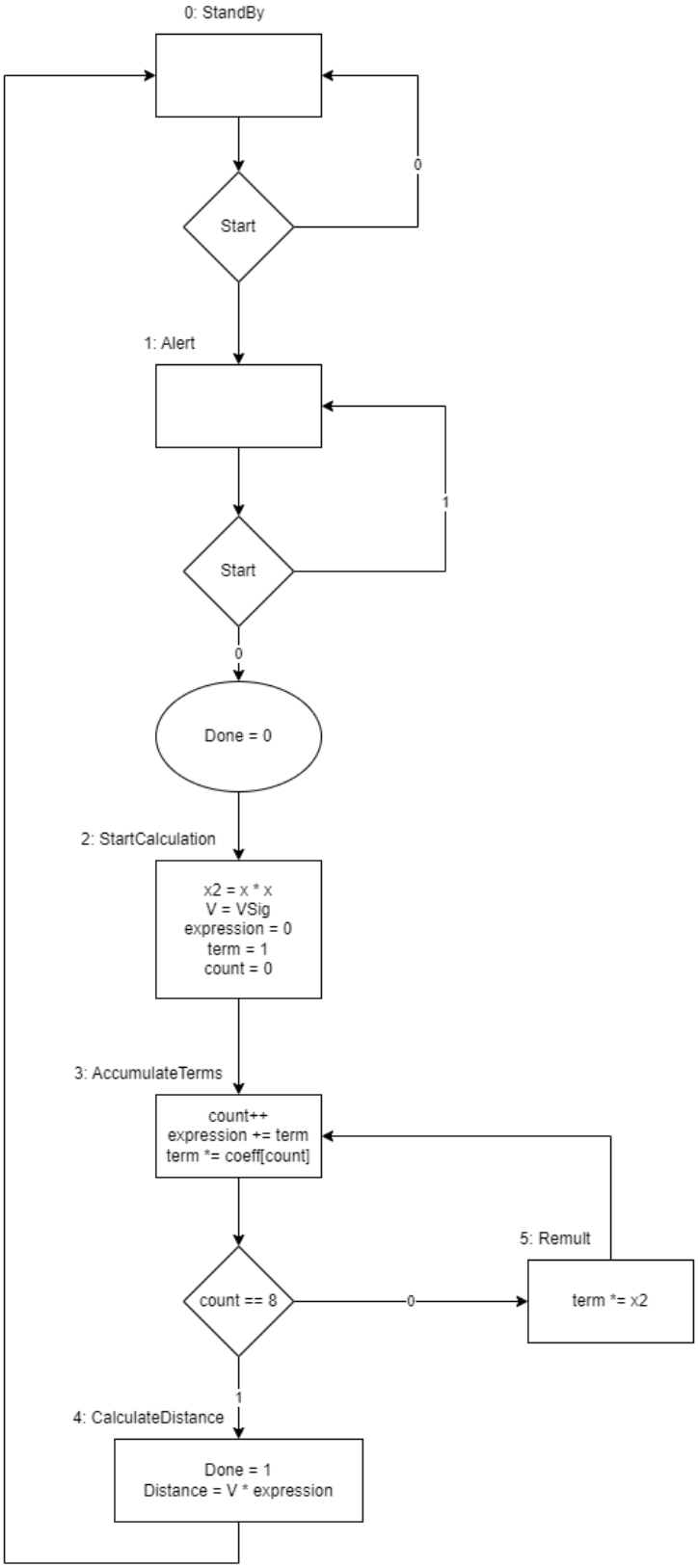
بخش اول: ساخت مسیرداده و واحد کنترلی:

برای اینکه مسیر داده و واحد کنترلی از نوع مدار ترتیبی مور را بسازیم، یک توصیف ASM از مدار مدنظرمان ایجاد کرده و آنرا سنتز میکنیم. سپس مسیر داده را بصورت ساختاری در وریلاگ توصیف کرده و قسمت کنترلی را هم بصورت behavioral میسازیم.

در توصیف ASM این محدودیت را داریم که حق نداریم در یک بلوک بیشتر از یک عملیات ضرب یا جمع داشته باشیم زیرا در مسیر داده محدود به تنها یک واحد ضرب‌کننده و یک واحد جمع‌کننده هستیم.

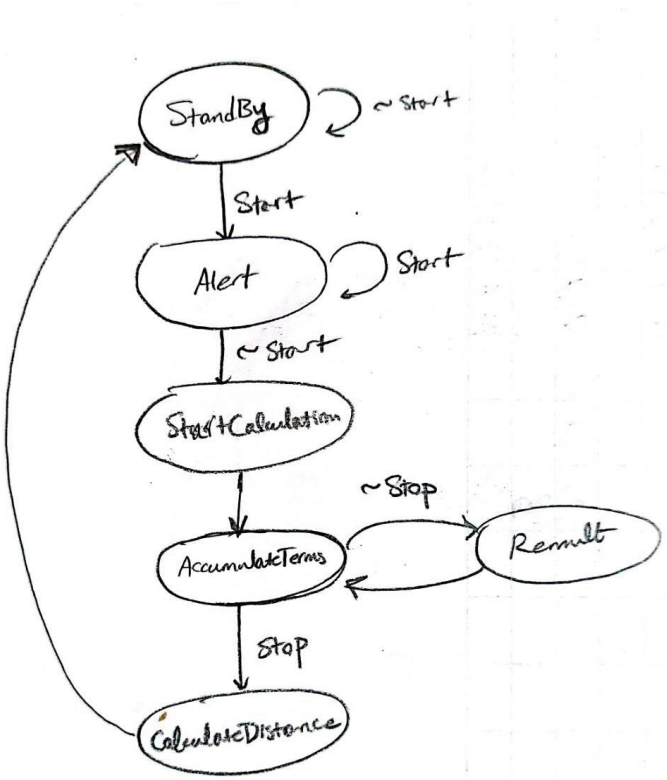
توصیف ASM را به این صورت انجام میدهیم: (در صفحه بعد آمده است)

در این توصیف میخواهیم پس از دریافت سیگنال شروع، با الگوریتمی که در صورت سوال آمده بود، ابتدا کسینوس را حساب کرده و در ثبات expression بریزیم. در این مسیر با استفاده از یک ROM که آدرس ورودی اش شمارنده حلقه خواهد بود، ضرایب ثابت مثل $\frac{-1}{4 \times 3}$ را بدون محاسبه استفاده خواهیم کرد (زیرا در همین رام ثبت شده اند و فقط کافی‌ست آنها را بخوانیم.) همچنین باید هربار term را در توان دوی x ضرب کنیم. چون دو عملیات ضرب داریم، ناچاریم دو استیت برای آپدیت کردن term استفاده کنیم. در نهایت هم به سادگی expression را در مقدار V که سیو کرده بودیم ضرب میکنیم و فاصله مدنظر بدست می‌آید.


0: StandBy
0
Start
1: Alert
1
Start
0
Done = 0
2: StartCalculation
x2 = x * x
V = VSig
expression = 0
term = 1
count = 0
3: AccumulateTerms
count++
expression += term
term *= coeff[count]
5: Remult
count == 8
0
term *= x2
1
4: CalculateDistance
Done = 1
Distance = V * expression

نام هر بلاک و شماره حالت متناظر به آن در کنار هر State Box آمده است. حالا ابتدا control unit را میسازیم. ابتدا state diagram را میکشیم:

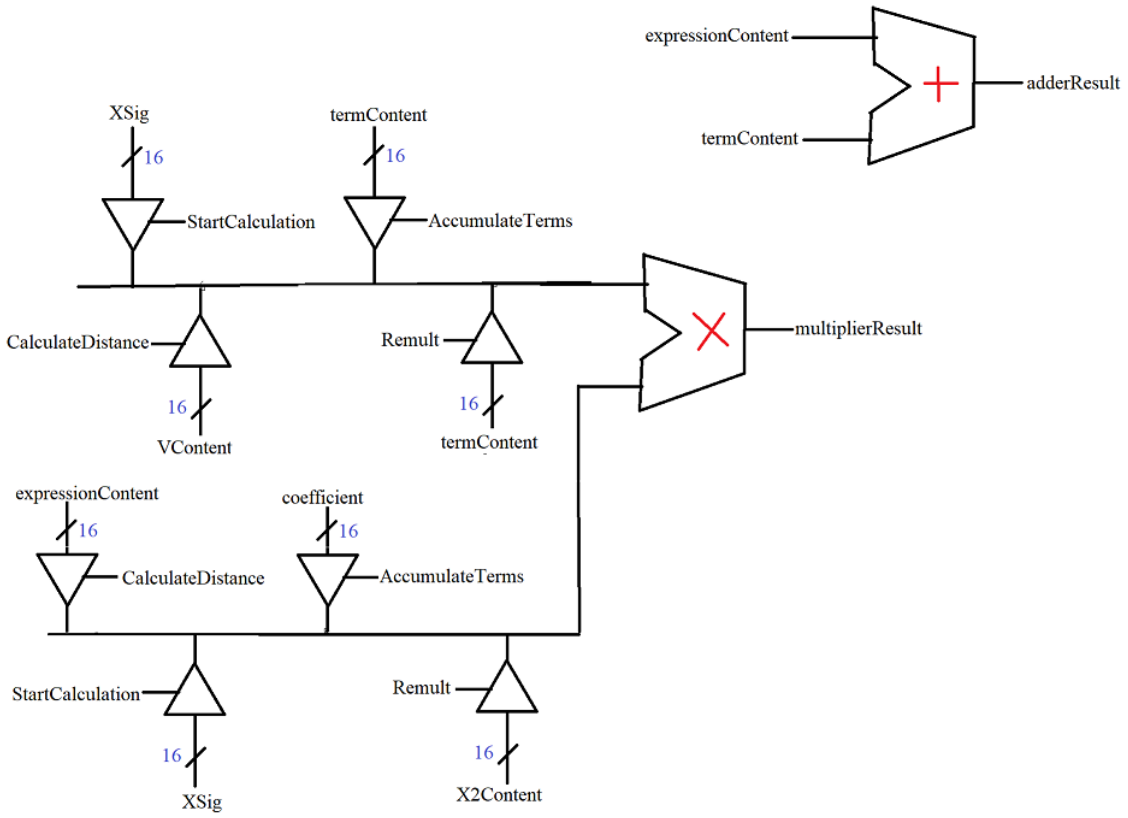
دقت شود سیگنال stop را بعنوان (count == 8) تعریف و پیاده‌سازی میکنیم.



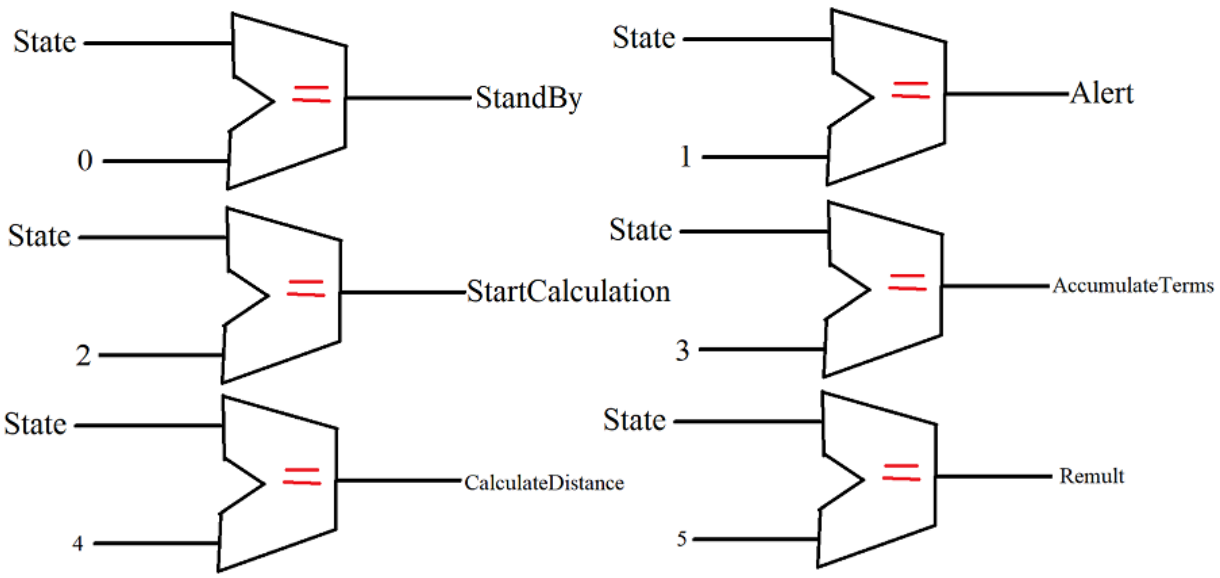
برای اینکه واحد کنترلمان از نوع مور باشد، باید خروجی‌های آن فقط تابع حالت فعلی آن باشند، پس خروجی های آنرا خود حالتش مینهیم. (حالت آن توسط یک بیت وکتور 4 بیتی نشان داده میشود. اعداد متناظر با هر حالت در ASM Chart آمده است.)

حالا به سراغ data path میرویم. باید با استفاده از ورودی‌های کنترلی state که 4 بیت هستند، عملیات‌های مناسب در هر مرحله را انجام دهیم. سنتز این بخش به این صورت میشود: (در صفحه بعدی آمده است. دقت شود که بدلیل بزرگ بودن مدار، آنرا بصورت قطعه قطعه نمایش داده ایم. جهت مشاهده مدار کامل به Report/Datapath.png مراجعه شود.)

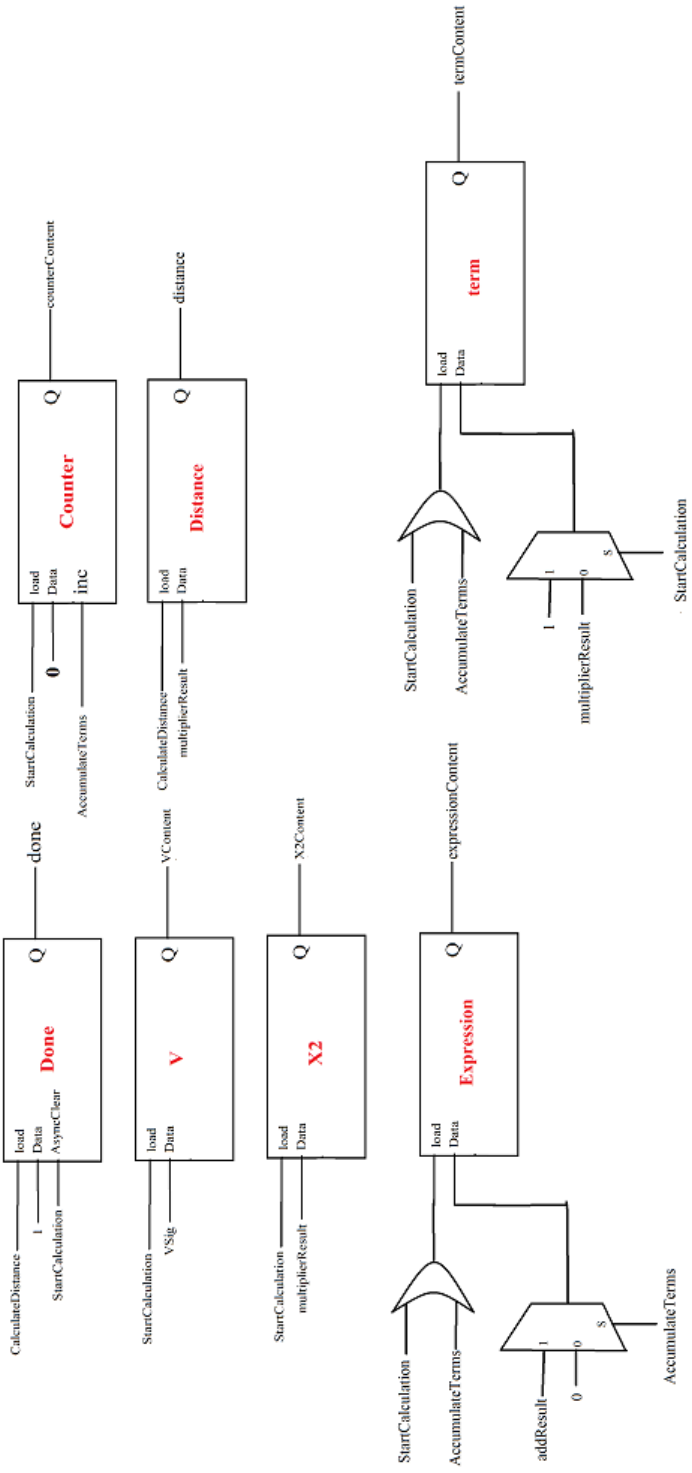
قسمت اول: جمع کننده و ضرب کننده:



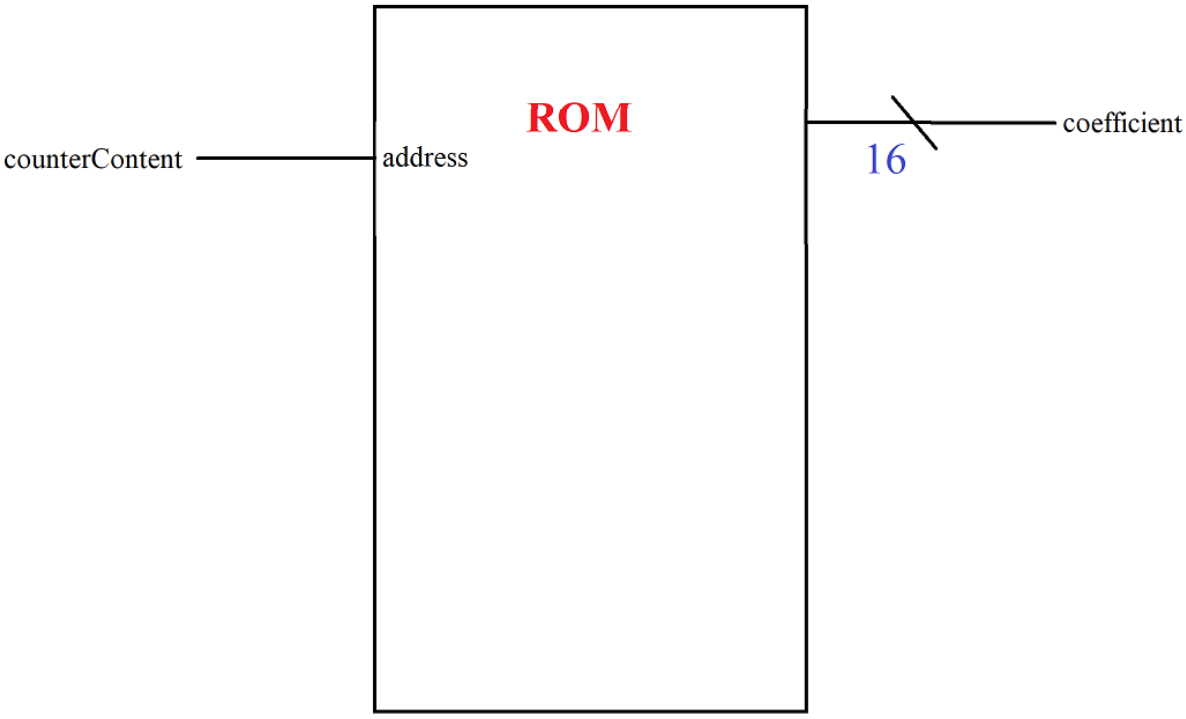
قسمت دوم: ساخت سیگنال‌های مربوط به استیت:



قسمت سوم: رجیسترها و داده:

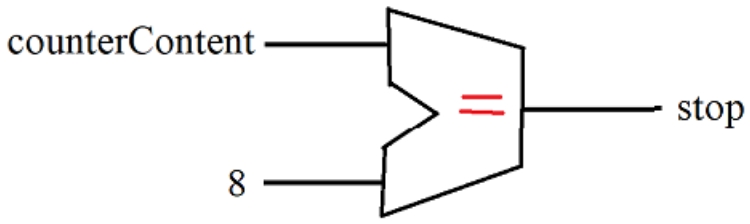


قسمت چهارم: رامی که ضرایب ثابت را نگه میدارد (در چارت ASM بصورت coeff[count] از آن استفاده شده است)



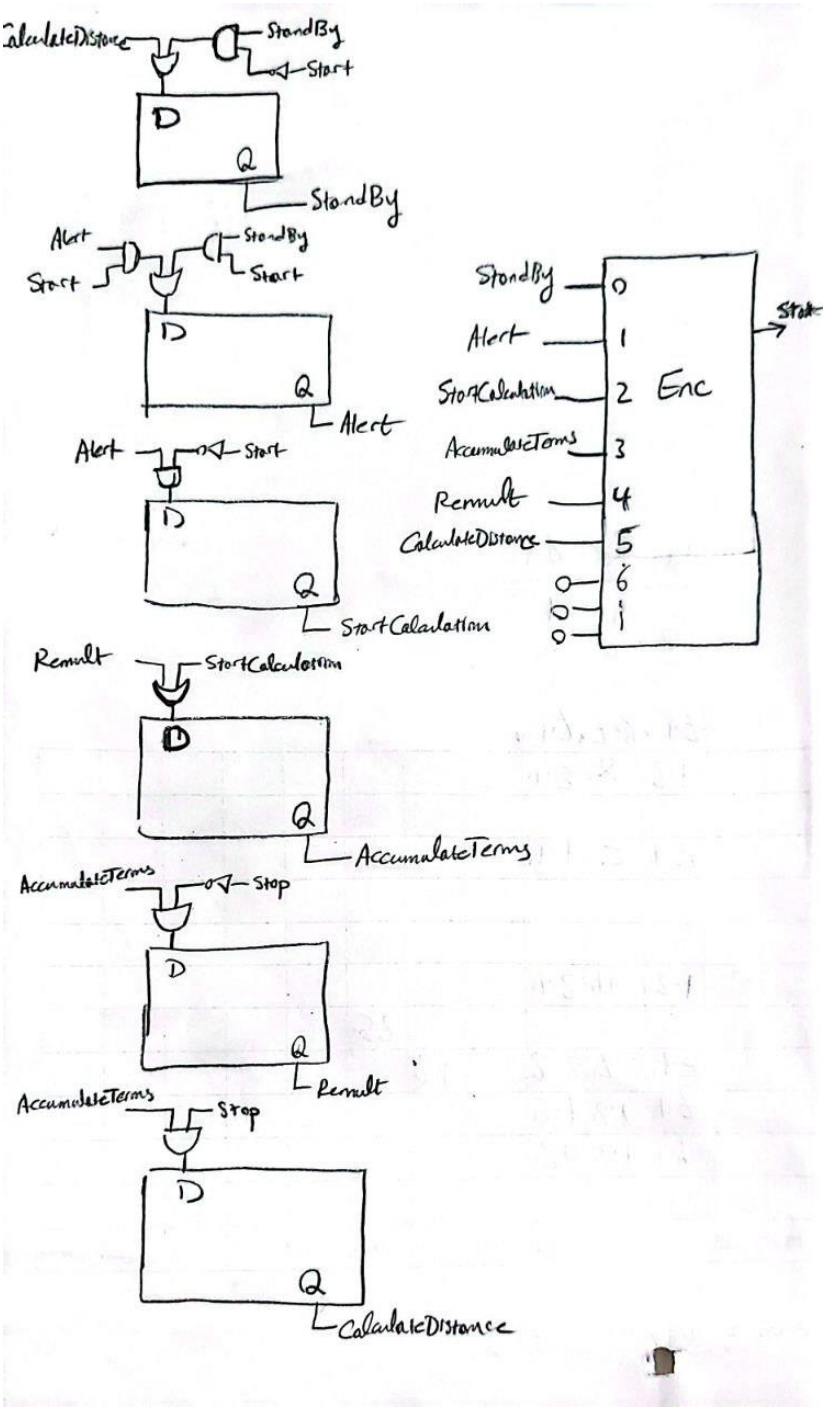
در این رام، در آدرس های متناظر به مقادیر counter مختلف، ضرایب ثابتی که باید در term ضرب شوند آورده شده است. مثلا برای counter = 0، مقدار $\frac{-1}{2 \times 1}$ گذاشته شده است، در counter = 1، مقدار $\frac{-1}{4 \times 3}$ قرار دارد و ... .

قسمت پنجم: ایجاد سیگنال stop



این سیگنال یک status signal به واحد کنترل است که باید از چرخه مصاحبه کسینوس خارج شود.

واحد کنترل را نیز به این صورت میتوانیم بسازیم:



حال باید کدهای Verilog مربوط به همین مدارها را بزنیم. فایل‌های آن همراه با این گزارش تحویل داده شده‌اند.

بخش دوم: توضیحات مختصر درباره کد وریلاگ:

کل مدار در فایل CUD.v قرار دارد که data path و control unit را کنار هم قرار داده و به هم متصل میکند. درون datapath دقیقا همان ساختاری که در شکل شماتیک کشیده ام را پیاده سازی میکنیم. نیاز داریم بجز گیت‌های ساده، باقی چیزها از جمله رجیسترها، مالتی‌پلکسرها، جمع و ضرب کننده، مقایسه‌کننده و رام را خودمان بسازیم. با توصیف‌های بسیار ساده‌ی رفتاری یا جریان‌داده میتوان این بخش ها را ساخت.

برای اینکه race condition رخ ندهد و بخش‌های ترکیبی در هر کلاک، مقادیر رجیستر کلاک قبل را استفاده کنند (همانگونه که در cycle accurate simulation فرض میکنیم اتفاق می‌افتد) رجیسترها را طوری توصیف میکنیم که با فاصله زمانی 1 زمان مجازی از بالا رفتن کلاک، مقدار ورودیشان را به بیرون هدایت میکنند و خروجیشان را عوض میکنند. برای این منظور از تاخیر میان‌دستوری استفاده میکنیم. برای مشاهده دقیق تر میتوانید به فایل register.v رجوع کنید.

دقت کنید که سیکل کلاک در زمان تست کردن باید بلندتر از 1 واحد زمان مجازی قرار بگیرد.

بخش سوم: تست کد وریلاگ:

یک برنامه تست بنچ با تعداد زیادی ورودی در testbench.v نوشته شده است. دستور لازم برای ران شدن و خروجی دادن آن در برنامه iverilog در فایل run.txt آمده است. در صورت انجام میبینیم که جواب های تست ها با خطای کمی تولید شده است:

Test 1:

x = 0.5, v = 1

result: 0.87744140625

expected result: 0.8775825619

error: **-0.02%**

Test 2:

x = 1.1591796875, v = -1.5

result: -0.6005859375

expected result: -0.60013719

error: **+0.075%**

Test 3:

x = 3.1416015625, v = 11.35595703125

result: -11.24462890625

expected result: -11.35595703

error: **-0.98%**

Test 4:

x = 0.78515625, v = 8.35009765625

result: 5.90771484375

expected result: 5.90583886

error: **-0.032%**

Test 5:

x = 1.23095703125, v = 8.35009765625

result: 2.78466796875

expected result: 2.78338467

error: **-0.046%**

Test 5:

x = 1.23095703125, v = 8.35009765625

result: 2.78466796875

expected result: 2.78338467

error: **-0.046%**

Test 6:

x = 2.498046875, v = -5

result: 3.974609375

expected result: 3.999865987

error: **-0.63%**

Test 7:

x = -2.498046875, v = -5

result: 3.974609375

expected result: 3.999865987

error: **-0.63%**

طبق output.txt میبینیم 21 کلاک طول میکشد تا هر ورودی بطور کامل آماده شود که البته دو سیکل آن برای شروع به کار کردن مدار است (1 و 0 کردن start) پس میتوان گفت قسمت **محاسبه** 19 کلاک طول میکشد.

بخش چهارم: برنامه پایتون مشابه

در فولدر python برنامه پایتون مربوطه نوشته شده است که با استفاده از الگوریتم گفته شده، کاری که مدار ما انجام میدهد را انجام میدهد. این برنامه با ورودی‌های رندوم، یک بار به ازای 1000 ورودی، یک بار به ازای 10000، یک بار به ازای 100000 و بار آخر به ازای 1000000 ورودی، الگوریتم را ران میکند و تایم کل مصرف شده و زمان سرشکن هربار ران کردن الگوریتم را حساب میکند.

نتیجه برنامه به این صورت شد:

```
c: 1000, t: 0.005997657775878906
average t: 5.997657775878906e-06
c: 10000, t: 0.04523658752441406
average t: 4.5236587524414065e-06
c: 100000, t: 0.46133852005004883
average t: 4.613385200500488e-06
c: 1000000, t: 6.3278725147247314
average t: 6.327872514724732e-06
```

که در آن c تعداد ورودی های رندوم است. میبینیم زمان سرشکن هربار ران کردن الگوریتم، حدود 5e-6 میباشد.

مشخصات پردازنده این بوده است:

Processor	Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
Installed RAM	16.0 GB (15.7 GB usable)

بخش پنجم: مقایسه پایتون و مدار:

چون میدانیم هر بار ران کردن الگوریتم 19 کلاک طول میکشد:

$$timeperalgorithmrun = \frac{19 \text{ clocks}}{200 \times 10^6 \text{ clocks per second}} = 10^{-7}\text{s}$$

میبینیم که ران شدن روی مدار حدود 50 برابر سریع تر است. پس برای استفاده های real time بهتر است از پیاده سازی سخت‌افزاری استفاده شود.