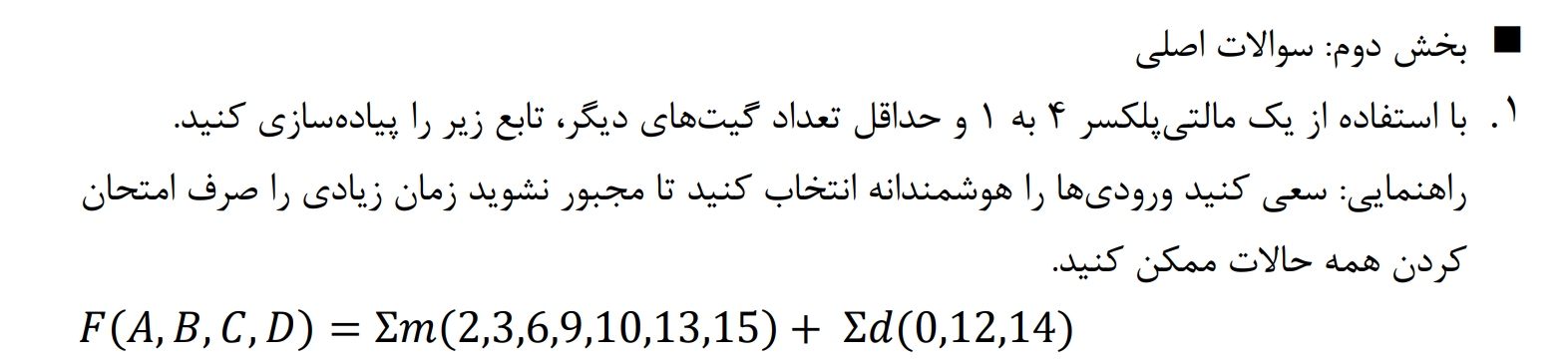
**به نام خدا**

**تمرین پنجم مدارهای منطقی**

**چمران معینی**

**۹۹۳۱۰۵۳**



می‌دانیم که هر مالتی‌پلکستر ۴:۱ دو سلکتور دارد، پس می‌توانیم دوتا از متغیرهایمان را به عنوان سلکتور انتخاب کنیم. همچنین می‌دانیم که در هر یک از چهار حالتِ سلکتور، دو متغیرِ دیگر نیز می‌توانند چهار حالتِ مختلف داشته باشند.

می‌توانیم این حالت‌های مختلف را به کمک جدول کارنو بررسی کنیم.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 10 | 11 | 01 | 00 | AB  CD |
| 0 | 0 | X | 0 | X | 00 |
| A | 1 | 1 | 0 | 0 | 01 |
| C XNOR D | 0 | 1 | 0 | 1 | 11 |
| 1 | 1 | X | 1 | 1 | 10 |
|  | C XOR D | 1 | CD’ | C |  |

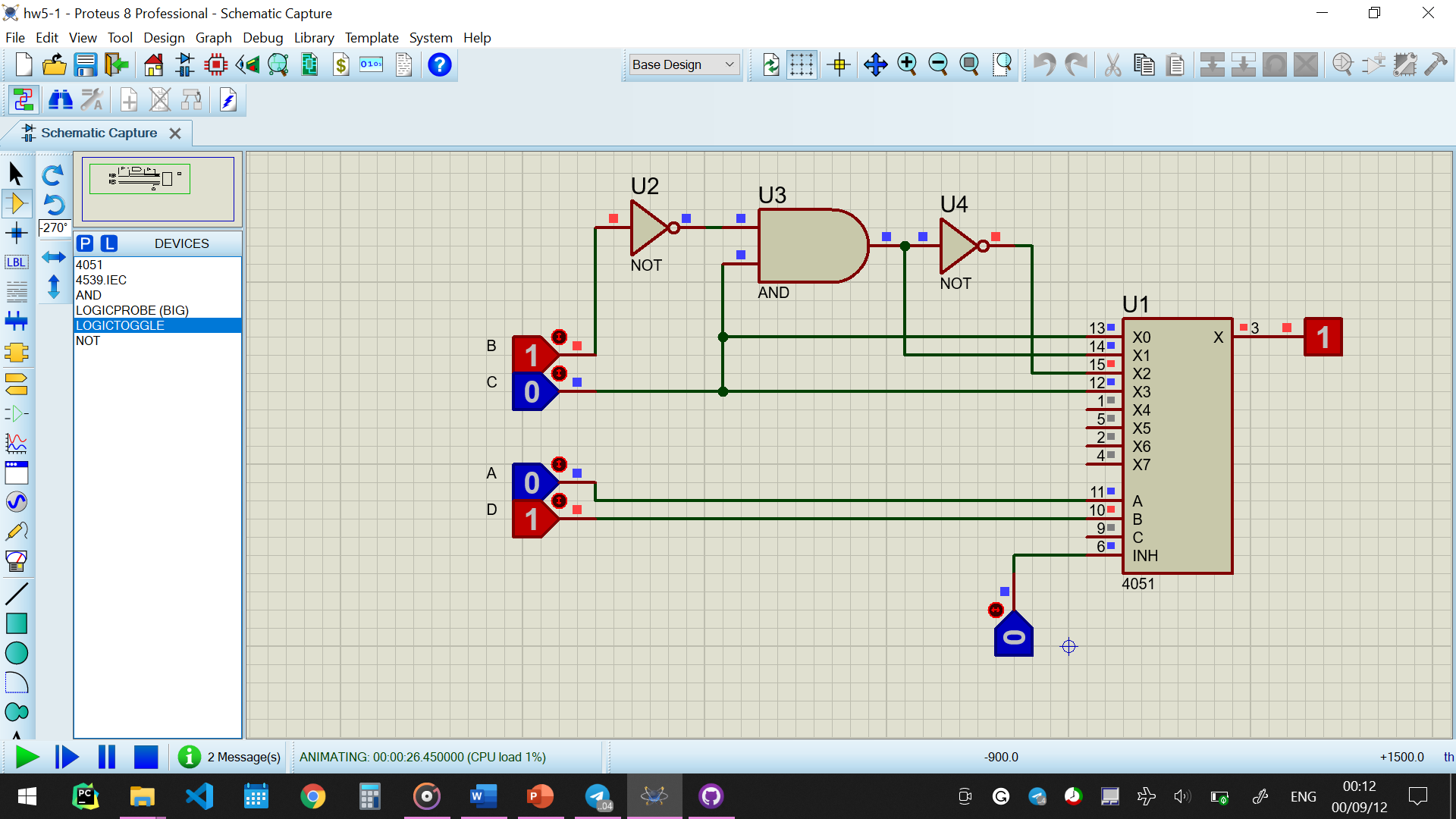
در هر یک از سطرها، و هم چنین در هر یک از سطرها، مقدار دو متغیر ثابت است که می‌توانیم آن دو متغیر را سلکتورها فرض کنیم که در یکی از چهار حالتِ خود هستند. سپس دو متغیرِ دیگر را بررسی می‌کنیم تا ببینیم در آن حالت، به واسطه‌ی چند گیت ورودیِ موردنظر را تولید می‌کنند.

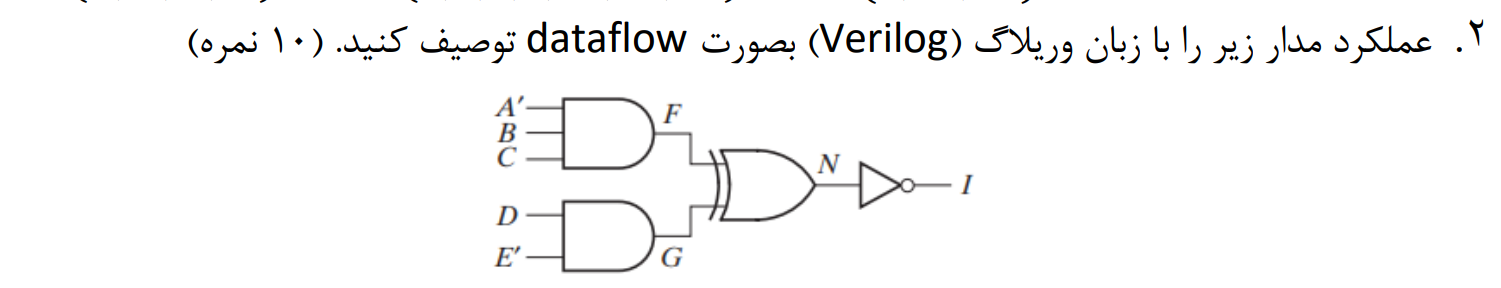
همین کار را برای شش (تعداد حالات ممکن برای انتخابِ دو سلکتور از چهار متغیر) حالتِ دیگر نیز بررسی می‌کنیم.

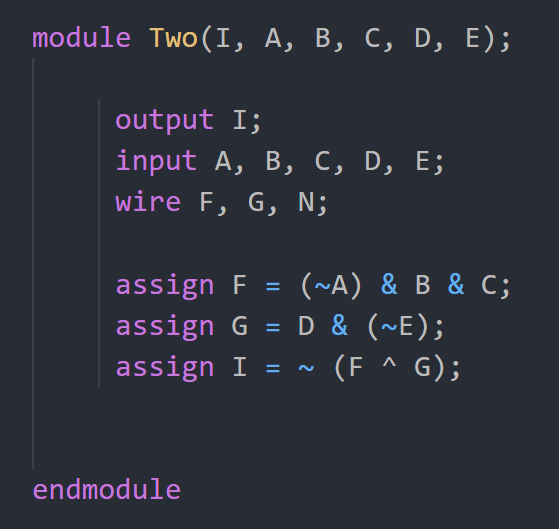
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 10 | 11 | 01 | 00 | AC  BD |
| C | 0 | 1 | 1 | X | 00 |
| A XOR C | 1 | 0 | 1 | 0 | 01 |
| A | 1 | 1 | 0 | 0 | 11 |
| C | X | X | 1 | 0 | 10 |
|  | D | B+D’ | B’+D’ | 0 |  |

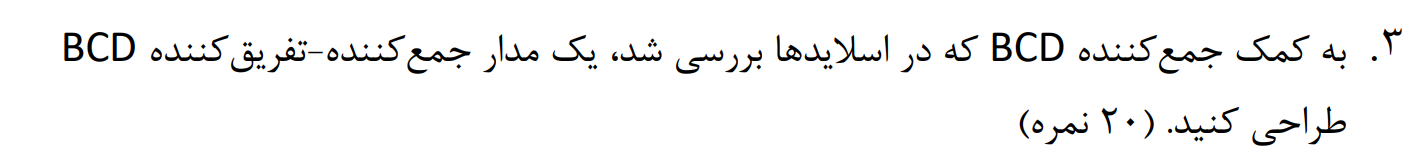
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 10 | 11 | 01 | 00 | AD  BC |
| AD | 0 | 1 | 0 | X | 00 |
| A’+D’ | 1 | 0 | 1 | 1 | 01 |
| A’+D | X | 1 | 0 | 1 | 11 |
| A | X | 1 | 0 | 0 | 10 |
|  | C | (B’C)’ | B’C | C |  |

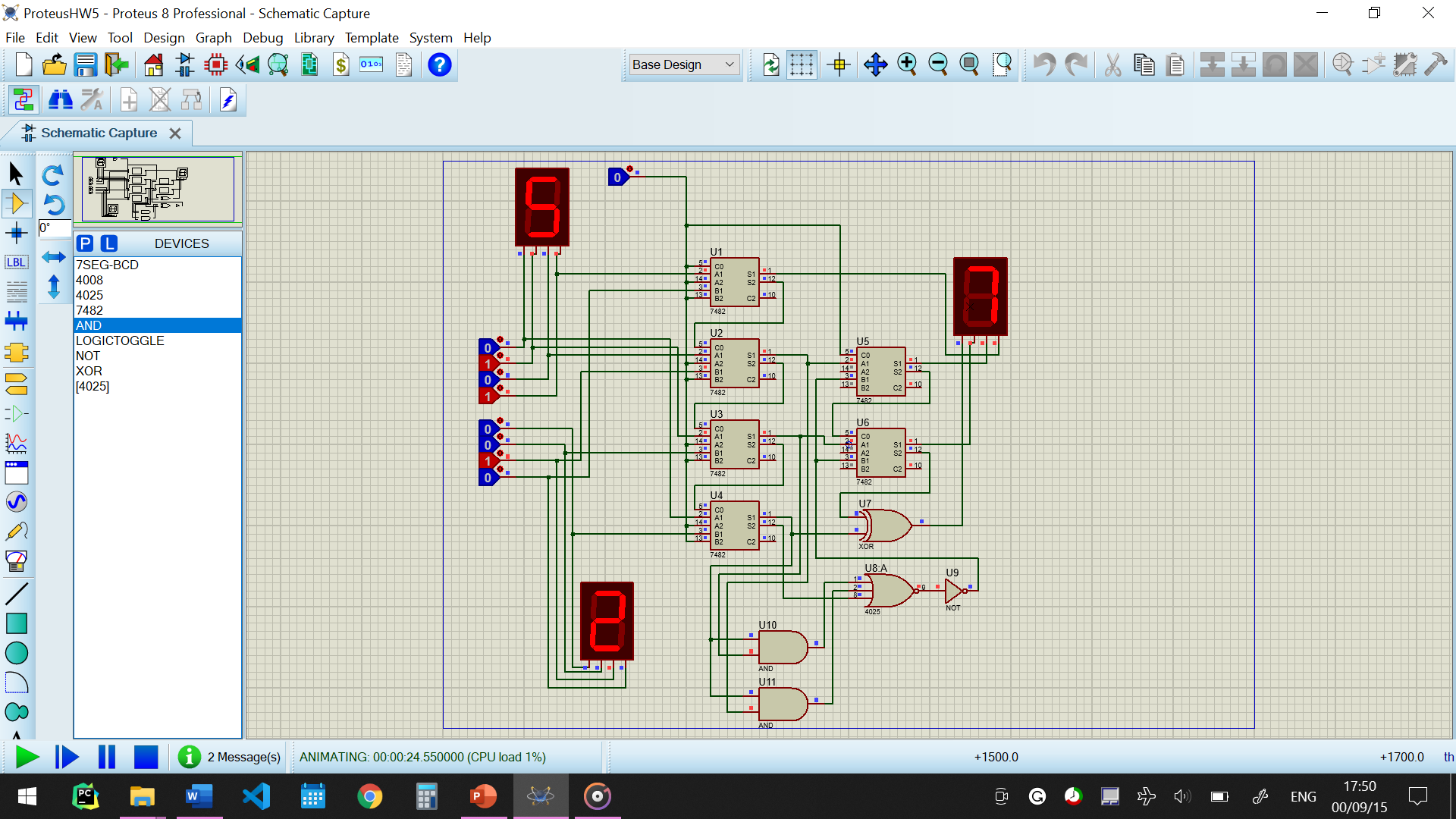
با مقایسه‌ی این شش حالت، می‌بینیم بهترین حالت هنگامی‌ست که AD به عنوان سلکتور انتخاب شود، در این حالت تنها به دو گیت نات و یک گیت اند نیاز داریم.

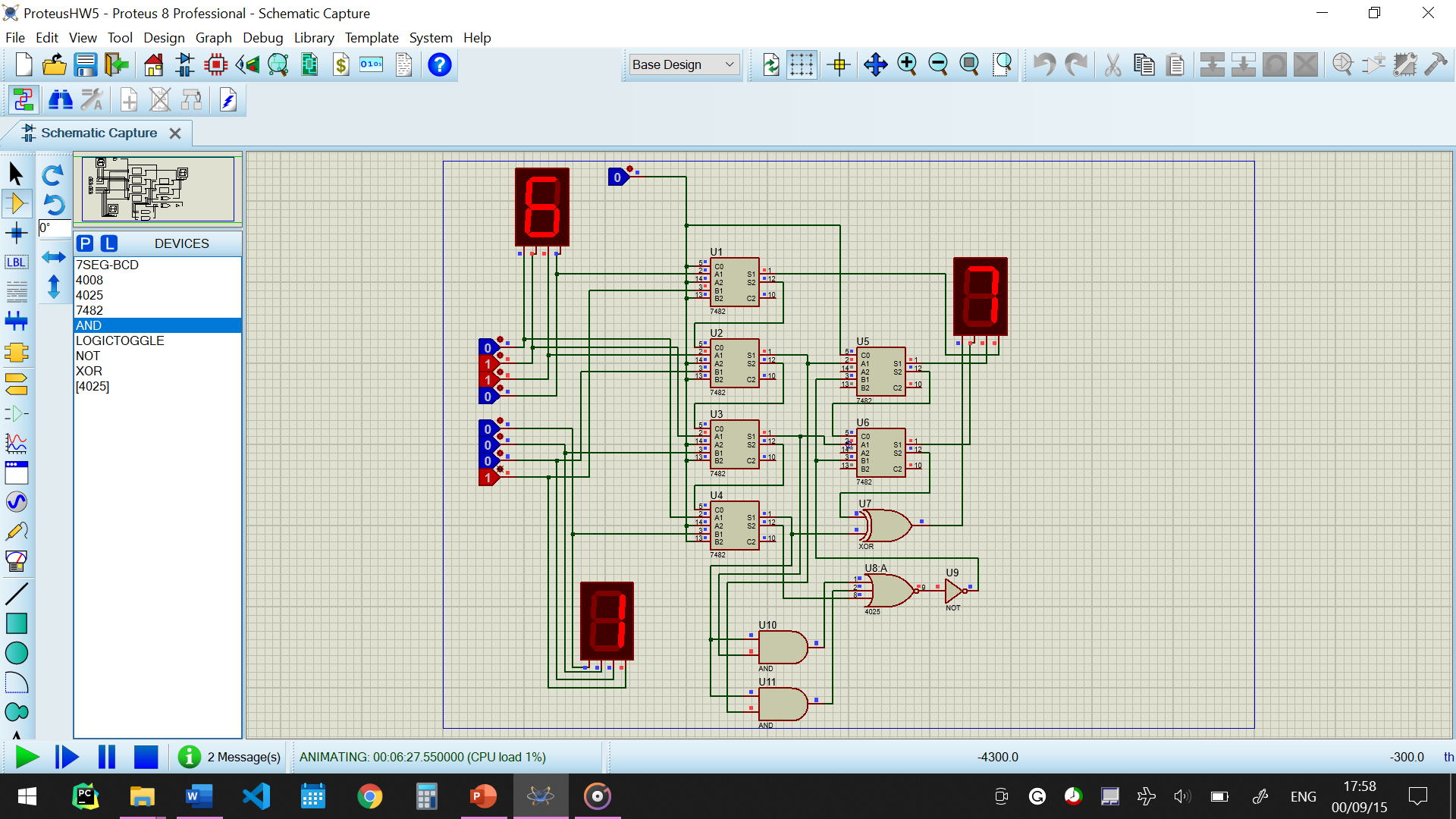












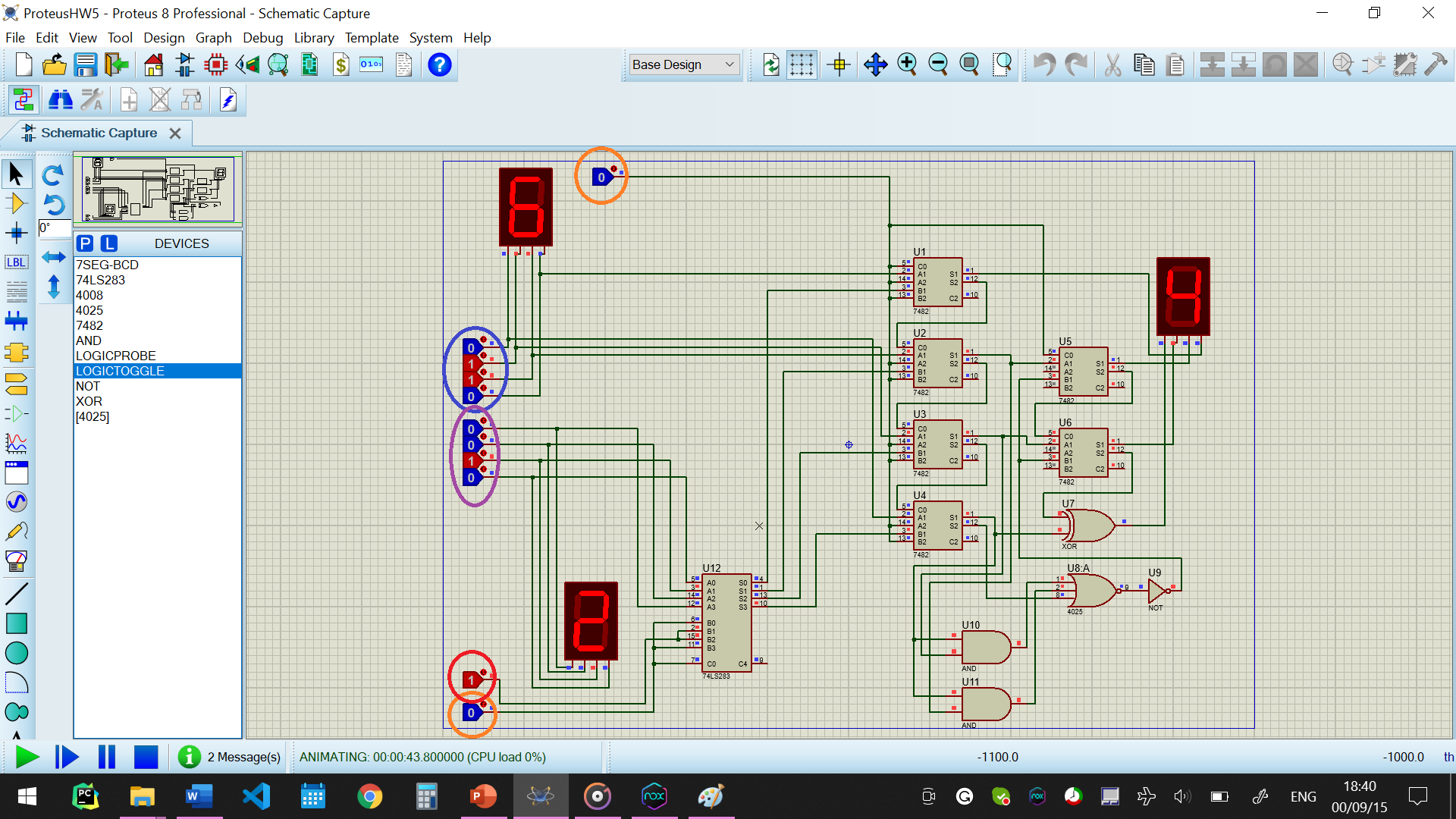
این یک مدار جمع‌کننده‌ی BCD است، مانند همان چیزی که در اسلایدها بررسی شد، با این تفاوت که چون FA مناسب برای جمع دو تک بیتی در پروتئوس پیدا نکردم، از FA هایی برای جمع دو عددِ دوبیتی استفاده شده که از s1 های‌شان به عنوان Cout استفاده شده و به A2 B2 های‌شان هم مقدار صفر داده شده است تا مشابه فول‌ادرهایی که در اسلایدها برای جمع دو تک‌بیتی داشتیم، عمل کنند.

حال قصد داریم عملگر تفریق را نیز به قابلیت‌های این مدار اضافه بکنیم. می‌دانیم که اگر به تفریق این دو عدد ده واحد اضافه کنیم، چیزی که در خروجی می‌بینیم تغییر نخواهد کرد، یعنی:

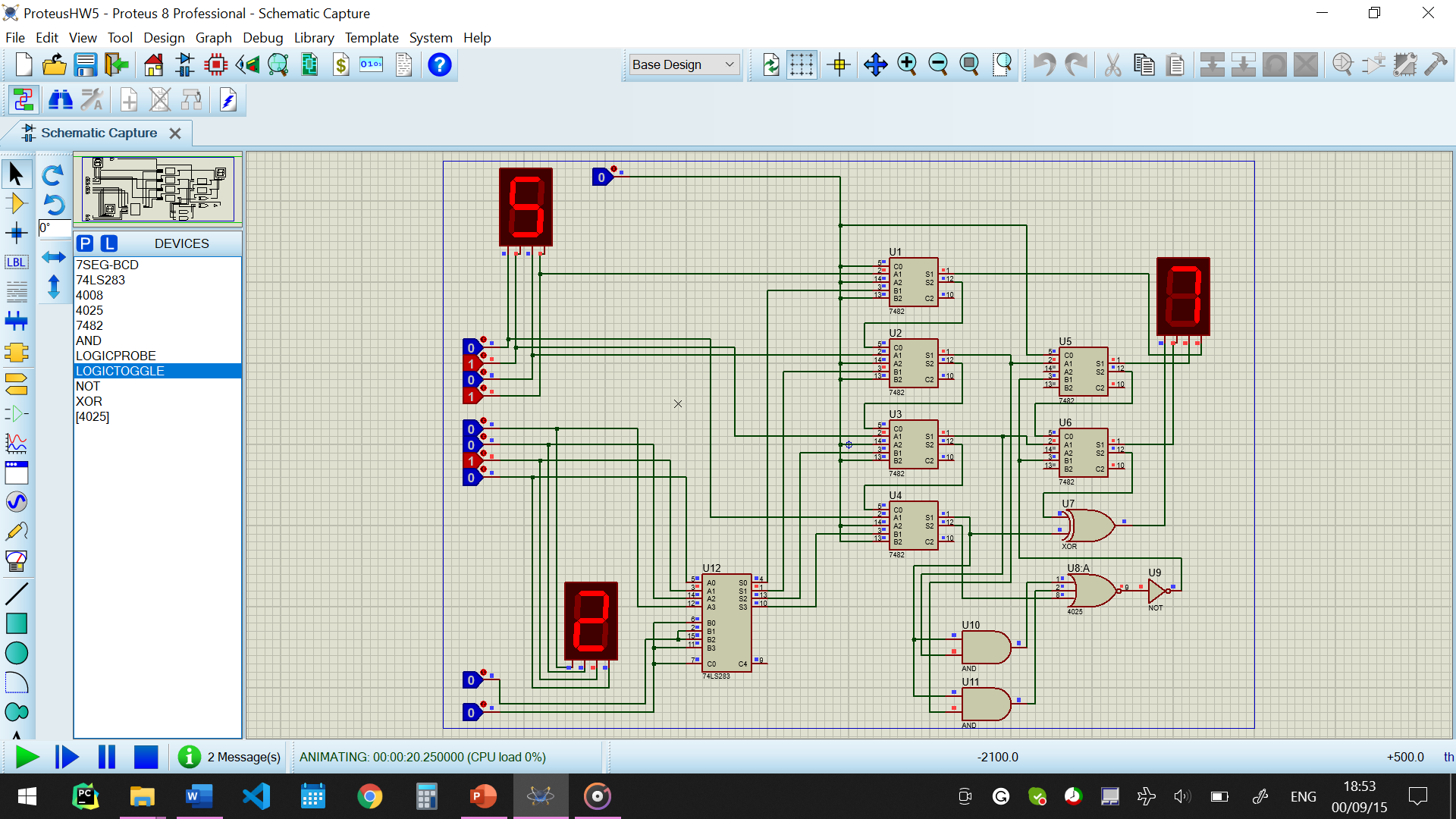
*پس می‌توانیم در ورودی‌مان یه سلکتورِ تک‌بیتی اضافه کنیم. اگر صفر بود، یعنی حاصل برابر جمع خواهد بود و همان فرآیندِ قبلی را ادامه می‌دهیم. اگر سلکتور برابرِ یک بود، به جای B از 10 – B استفاده می‌کنیم.*

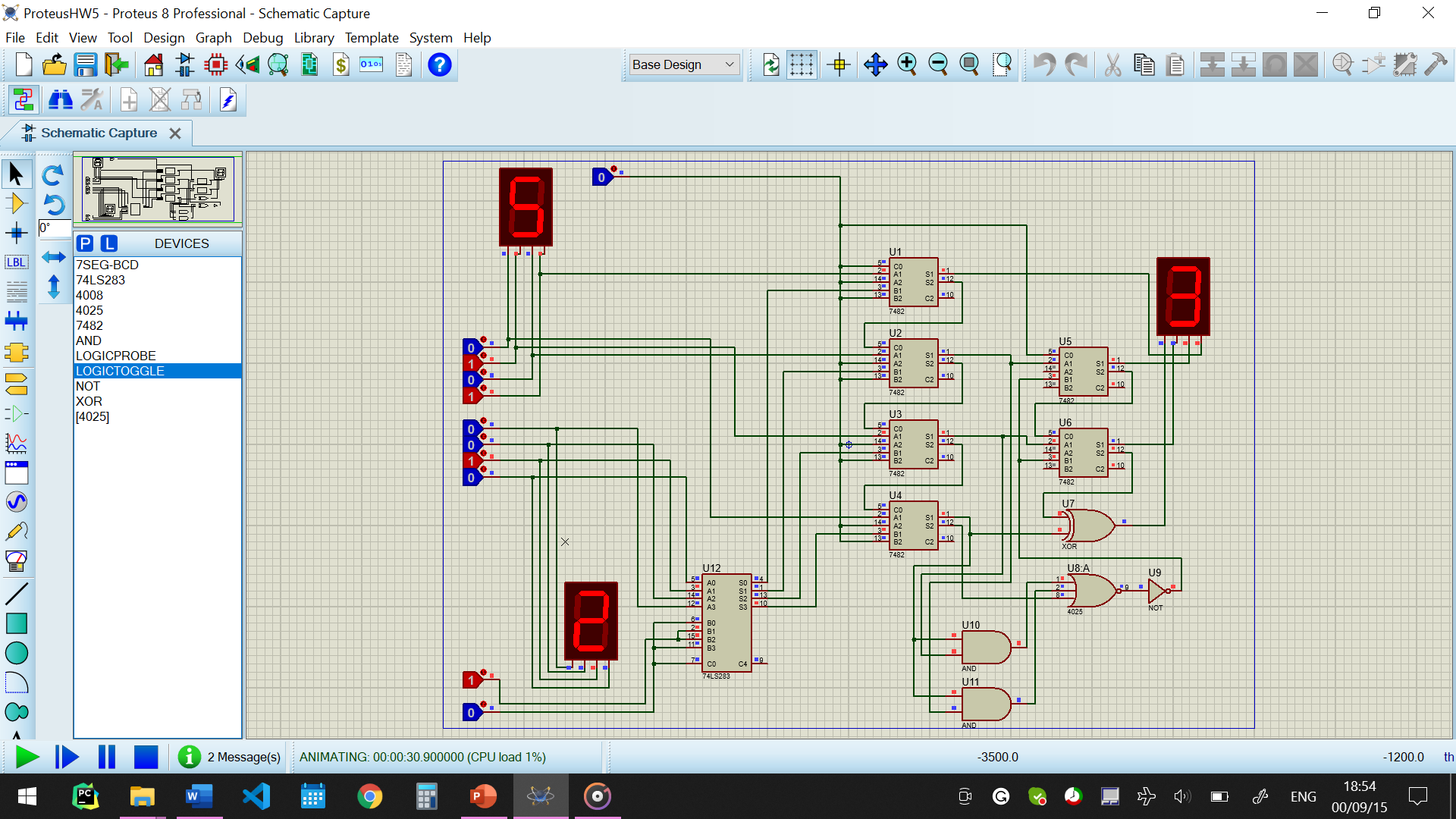
*برای به دست آوردنِ 10 – B کافی‌ست که از یک ادر استفاده کنیم و عدد B را با مکملِ دوی 1010 (دهِ باینری) که برابر است با 0110 جمع بکنیم.*

*با منطقِ گفته شده، مدار مطلوب به این شکل خواهد بود:*

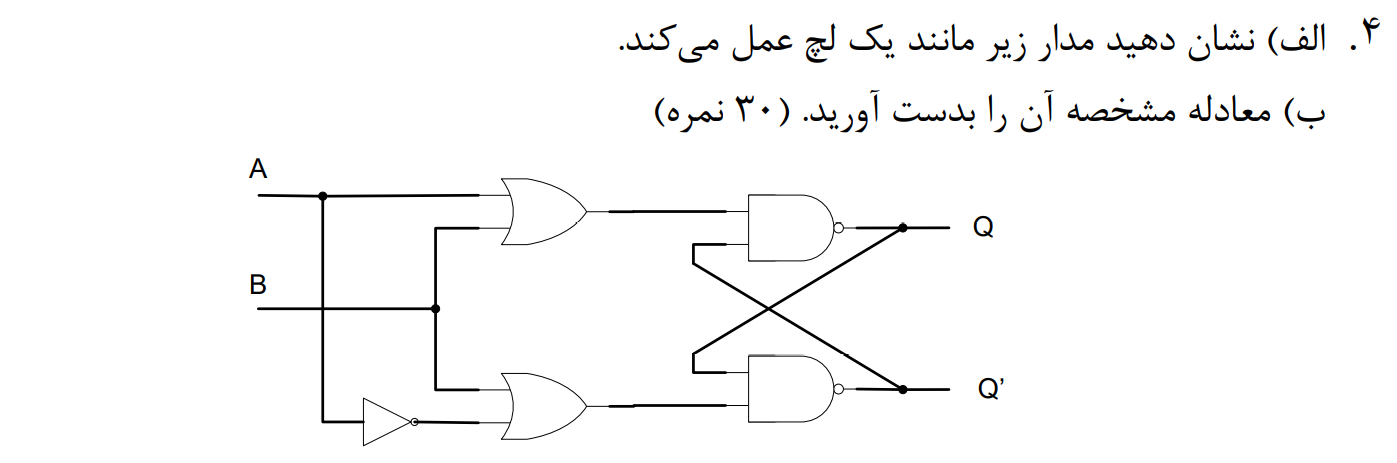


دو 0 ای که با دایره‌ی نارنجی مشخص شده‌اند، صفرهای منطقی هستند. 1 ای که با دایره‌ی قرمز مشخص شده است، سلکتوری‌ست که اگر 1 باشد، حاصلمان A – B خواهد شد و اگر 0 باشد، حاصل‌مان A + B خواهد شد. دایره‌های آبی و بنفش هم به ترتیب اعداد A و B‌ را نشان می‌دهند که هر ارقامِ باارزش‌تر بالاتر قرار گرفته‌اند.





اگر قصد داشته باشیم بیشتر از دو عدد را هم با یکدیگر جمع و تفریق کنیم، می‌توانیم از Cin ادری که کم‌ارزش‌ترین ارقام اعداد را جمع می‌کند، به عنوان Cin استفاده کنیم، برای Cout هم از خروجی گیتِ NOT استفاده می‌کنیم.

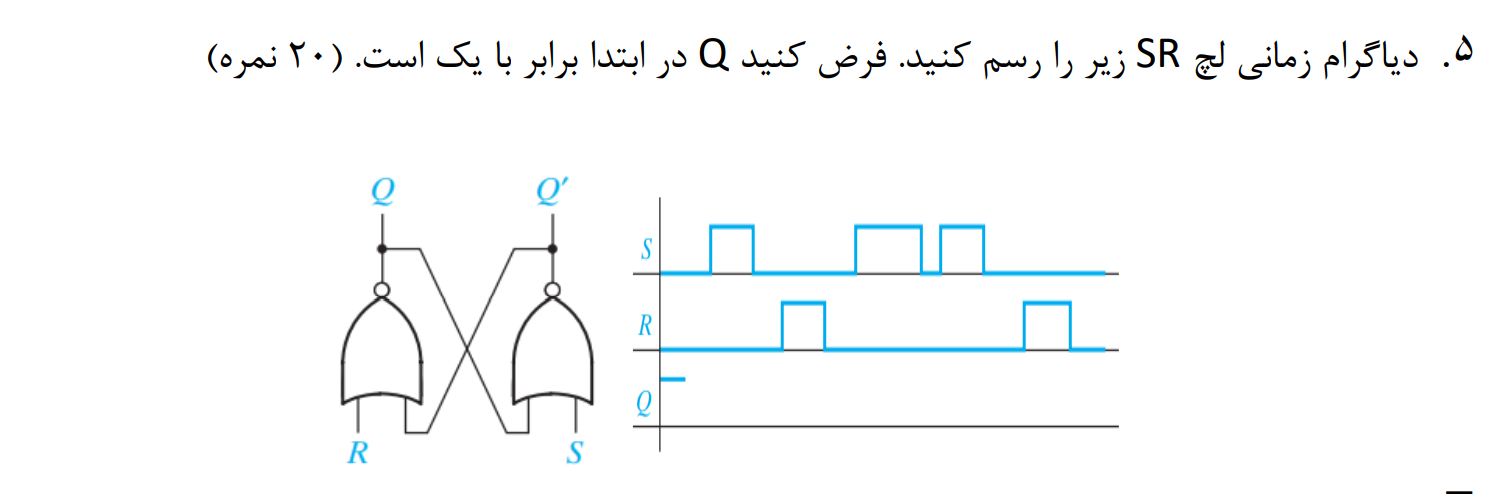


الف)

در این مدار، هنگامی که مقدار B برابر با صفر باشد، مقدار Q برابر با A’ می‌شود به سرعت، و هنگامی که مقدار B برابر با ۱ شود، مدار وارد حالتِ HOLD می‌شود و هرچقدر که مقدار A را تغییر دهیم، مقدار Q و Q’ همان مقدار قبلی‌شان خواهند ماند.

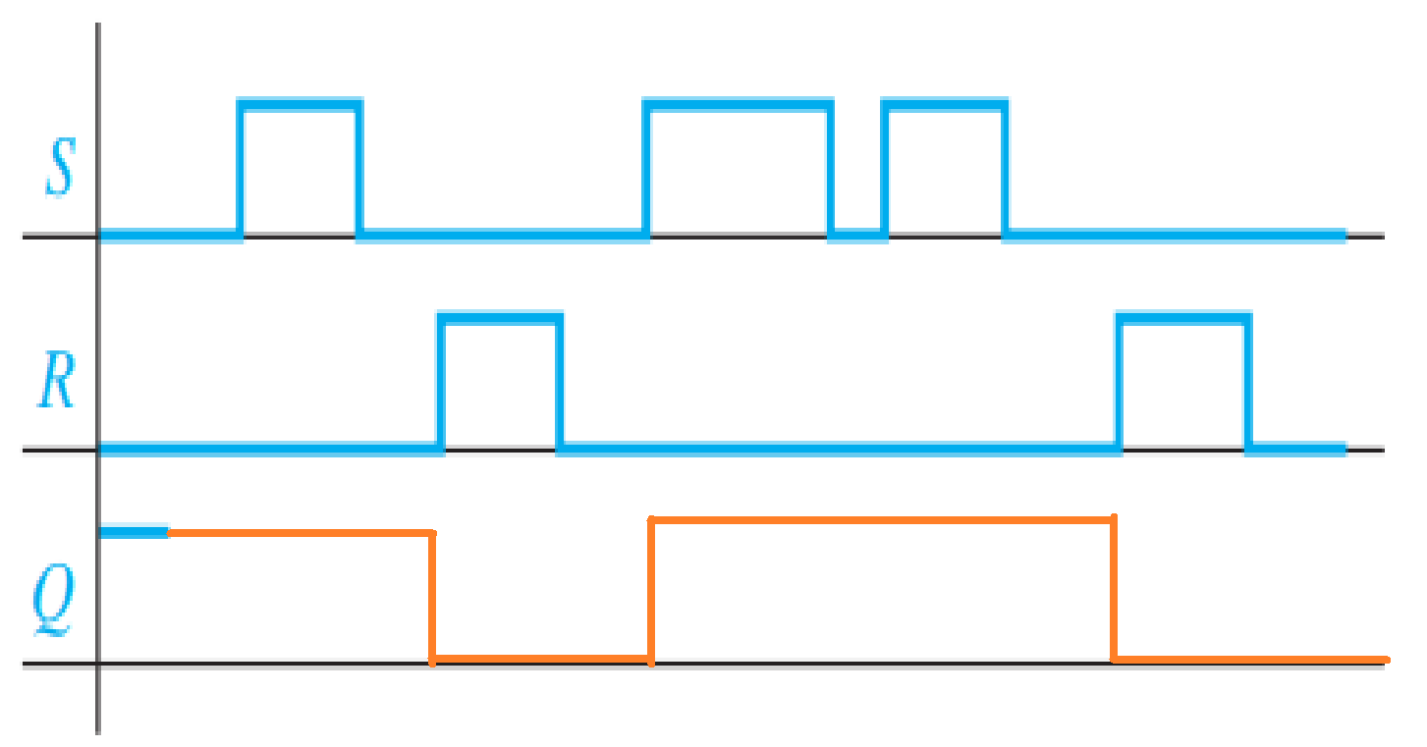
ب)

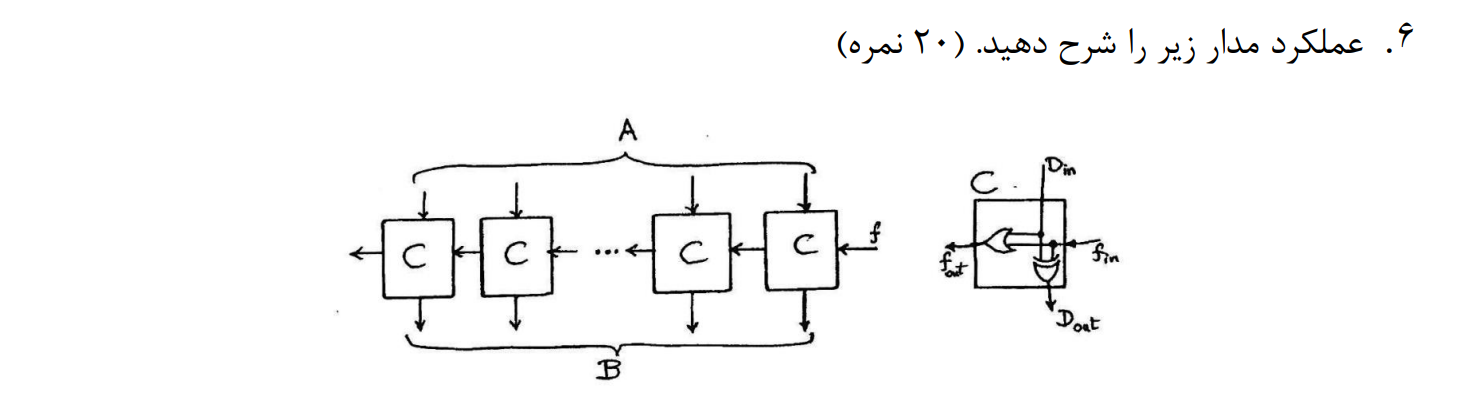
|  |  |  |  |
| --- | --- | --- | --- |
| Q’ | Q | B | A |
| 0 | 1 | 0 | 0 |
| HOLD | HOLD | 1 | 0 |
| 1 | 0 | 0 | 1 |
| HOLD | HOLD | 1 | 1 |



این که در ابتدا مقدار Q برابر با یک است، به این معناست که مدتی پیش، مقدار S یک بوده است.

هنگامی که دوباره مقدار S برابر با یک و دوباره صفر می‌شود، تغییری در مقدار Q ایجاد نمی‌شود، اما هنگامی که R برابر با یک می‌شود، مقدار Q برابر با صفر خواهد شد و این بار Q’ برابر یک می‌شود. بعد از صفر شدن R نیز، Q همچنان همان مقدار را در خود نگه خواهد داشت، تا زمانی که دوباره مقدار S برابر با یک می‌شود که این باعث می‌شود دوباره Q برابر با یک شود و این مقدار را تا زمانی که R برابر با صفر است، در خود نگه می‌دارد و نهایتا هنگامی که دوباره R برابر با یک می‌شود، دوباره مقدارQ برابر با صفر می‌شود و تا زمانی که مقدار S برابر با صفر است، این مقدار را در خود نگه می‌دارد.





هنگامی که **f برابر با یک** باشد، هر یک از بیت‌های B برابر با متضادِ بیت متناظر از A خواهد بود، به بیانِ دیگر مدارمان مکملِ یکِ ورودی‌اش را خروجی می‌دهد.

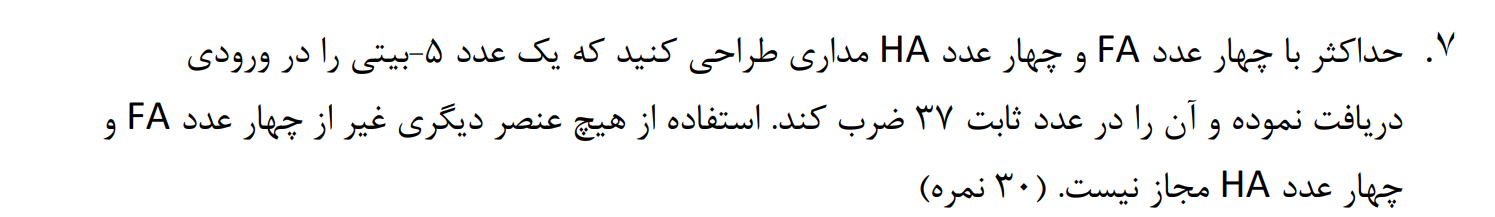
هنگامی که **f برابر با صفر** باشد، تا جایی که بیت‌های A (از شروع به پایان / از راست به چپ) برابر با ۰ باشند، هر یک از بیت‌های B برابر با بیت متناظر از Aخواهند بود، اولین بیتِA که برابر با ۱ بشود نیز، خروجیِ متناظرش برابر آن خواهد بود و مقدارِ یک را خواهد داشت، اما از آن‌جا به بعد تمام بیت‌های B برابر با مقدار متضاد بیتِ متناظر از A خواهند شد. می‌بینیم که این همان روشی‌ست که برای یافتنِ مکمل دوی اعداد استفاده می‌کردیم، که از سمت راست شروع می‌کردیم و تمام صفر‌ها و اولین یک را رد می‌کردیم و بعد از آن تمام ۱ ها را ۰ ، و تمام ۰ ها را ۱ می‌کردیم. پس در این حالت، مدار مکملِ دوی ورودی‌اش را خروجی می‌دهد.

اما چطور و چگونه؟

در حالتی که **f برابر با یک** باشد، fout در اولین C مان هم برابر یک خواهد شد. به این ترتیب، fin بعدی و پس از آن تمام fin ها و تمام fout ها برابر با یک می‌شوند. هنگامی که همه‌ي fin ها برابر با یک هستند، مقدار B برابر با A xor 1 یا همان A’ خواهد شد، به این ترتیب مکمل یک تولید می‌شود.

در حالتی که **f برابر با صفر** باشد، تا جایی که بیت‌های A هم برابر با صفر هستند، هم 0 xor 0 برابر با صفر است و هم 0 and 0 ، در نتیجه تمام خروجی‌ها و fin و fout ها نیز 0 خواهند بود. هنگامی که اولین بیتِ A برابر با یک شود، آن‌گاه 1 xor 0 هم برابر با یک می‌شود و از آن‌جا که

1 and 0 برابر با 1 است، از این‌جا به بعد تمام fin ها و fout ها برابر با ۱ خواهند شد و مطابقِ همان منطقِ بندِ قبلی، هر یک از بیت‌های B برابر با NOT بیتِ متناظرِ A خواهد شد.

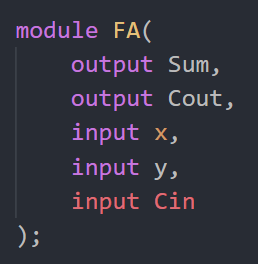


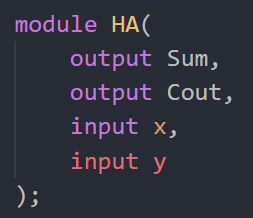
باید ۳۷ را به شکلِ جمعی از توان‌های دو در بیاوریم:

می‌دانیم که با ضرب شدنِ دو در یک عددِ باینری، یک صفر به سمتِ راستِ عددمان اضافه می‌شود، یعنی حاصل ضرب هر عدد ۵-بیتی در ۳۷ را می‌توان این‌گونه به دست آورد:

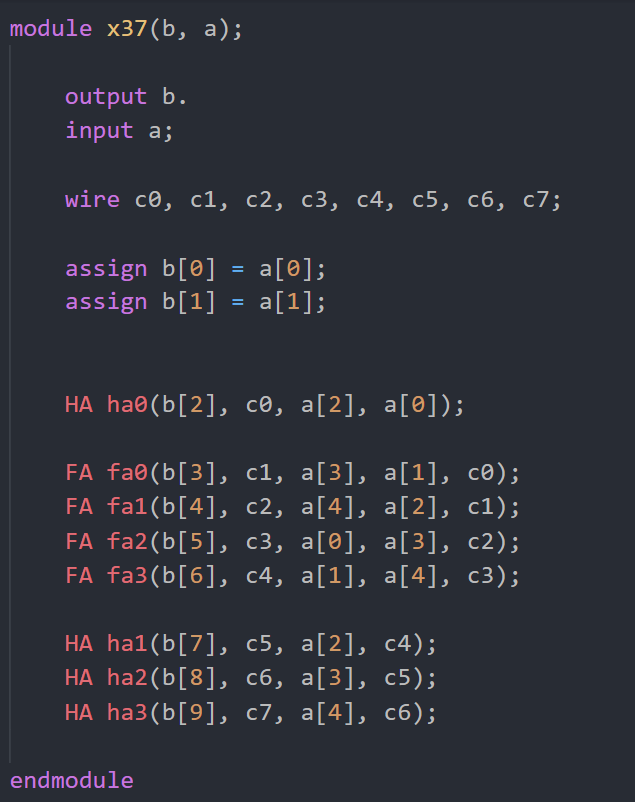
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A[0] | A[1] | A[2] | A[3] | A[4] |  |  |  |  |  |  |
|  |  | A[0] | A[1] | A[2] | A[3] | A[4] |  |  |  |
|  |  |  |  |  | A[0] | A[1] | A[2] | A[3] | A[4] |
| A[0] | A[1] | A[2]  + A[0] | A[3] + A[1]  + Cout(A[2] + A[0]) | A[4] + A[2]  + Cout(A[3] + A[1]) | A[3] + A[0]  + Cout(A[4] + A[2]) | A[3] + A[0]  + Cout(A[4] + A[2]) | A[2]  + Cout | A[3] | A[4] | ANSWER |

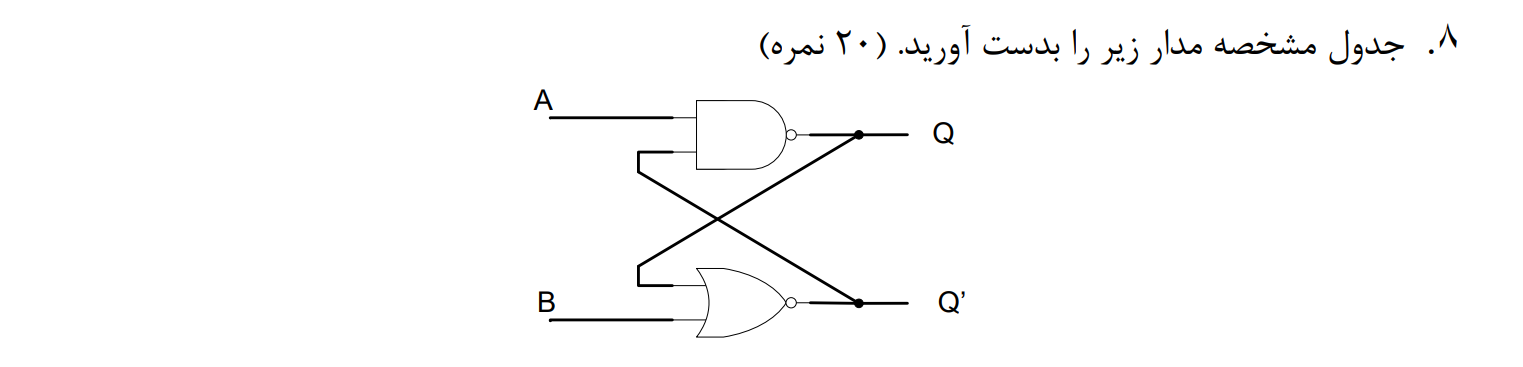
تفاوت FA و HA در این است که در HA دو بیت را با یکدیگر جمع می‌کنیم و یک بیت کری و یک بیتِ سام را در خروجی می‌دهیم، اما در FA علاوه بر این‌ها، همچنین می‌توانیم یک Cin هم بگیریم، یعنی تواناییِ جمعِ سه بیت را داریم، پس برای بخش‌های بنفشِ جدول که جمع دو متغیرند، از HA و برای بخش‌های آبی که جمع سه متغیرند، از FA استفاده می‌کنیم.

فرض می‌کنیم که HA و FA به این ترتیب ورودی بگیرند:



اگر عدد ورودی را A بنامیم و خروجی را B بنامیم، کد وریلاگ مدار طراحی شده، به این شکل خواهد بود:





|  |  |  |  |
| --- | --- | --- | --- |
| Q’ | Q | B | A |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |