علی نظری 99102401 بخش takehome میانترم درس DSD

پاسخ سوال ششم:

در این سوال ما باید ALU ای طراحی کنیم که 4 عمل اصلی را برای ما انجام می‌دهد. برای این طراحی، نخست باید ماژول ALU را پیاده‌سازی کنیم که کل کار داخل آن انجام می‌شود. چون که تعداد بیت اعداد ورودی مشخص نشده است، آن را به شکل parameter تعریف می کنم که مقدار دیفالت آن 4 است. یعنی ورودی‌های a و b ما هر دو 4 بیتی هستند. ورودی های ماژول هم همان چیزهایی است که در صورت سوال گفته شده است. Clock که همان کلاک موجود برای کل مدار است و reset هم که برای reset کردن کل مدار است و ورودی data_in هم برای این است که با آن هم opcode را مشخص کنیم و هم a و b را و این ورودی ها به شکل سریال داده می‌شود و باید بیت به بیت آن‌ها را ورودی بگیریم.

سپس اول از همه باید رجیستر های a و b را بسازیم که N بیتی هستند. سپس result چون قرار است حاصل ضرب را هم به ما بدهد، می دانیم که حاصل ضرب دو عدد n بیتی حداکثر 2n بیتی است. پس حداکثر سایز مورد نیاز را برای result می گذاریم. سپس تا حدی شبیه روش هافمن که در کلاس گفته شد، برخی stage ها یا مراحل را در نظر میگیریم که به ترتیب باید اجرا شوند. اول از همه خب ما بخش گرفتن opcode و a و b را داریم و هر کدام مرحله متفاوتی هستند. همچنین باید opcode های متفاوت را هم در نظر داشته باشیم و قابلیت اجرای آن ها را داشته باشیم. پس دو localparam گرفتیم که یکی State را مشخص می کند که الان بیاد چه کار کنیم و یکی هم opcode را مشخص می کند که الان چه کاری باید روی ورودی ها انجام دهیم.

ار آن جا که میدانیم assign به اصطلاح real time است و هر تغییری روی مقادیر rhs ایجاد شود، درجا مقدار lhs را هم تغییر می دهد، پس از Assign برای مشخص کردن output استفاده می کنیم. می دانیم که برای مشخص کردن even parity باید تعداد زوجی یک داشته باشیم. پس با xor کردن کل بیت ها به روشی که در کد مشخص است، می توانیم این کار را انجام دهیم. برای بحث zero هم خب باید کل بیت ها را or کنیم و اگر صفر شود، یعنی خب نتیجه zero بوده است. پس مطابق خطی که مقدار data_out را مشخص کردیم، برحسب State ای که داخل آن هستیم، میایم این کارها را انجام می دهیم و مقادیر را مشخص می کنیم یعنی اگر state همان zero باشد، می آییم کل بیت ها را or می کنیم و اگر در State مربوط به parity بودیم هم کل بیت ها را xor می کنیم و اگر در این State 2 نباشیم هم به ترتیب بیت های خروجی را تحویل می دهیم.

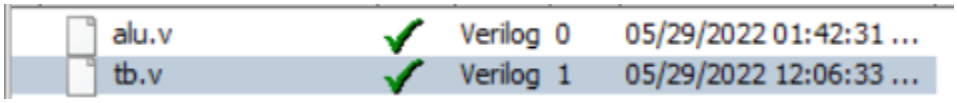
سپس با توجه به لبه بالارونده کلاک، کار خود را آغاز می کنیم. اگر ready یک شده بود که خب یعنی خروجی قبلی آماده است و به State اولیه بر می گردیم و ready را هم صفر میکنیم. اگر سیگنال reset آمده باشد، همه چیزهایی که داریم را باید به حالت اولیه برگردانیم تا مدار به درستی reset شود.

سپس شروع می کنیم بر حسب State های مدار کارهای لازم را انجام میدهیم. اگر در حالت parity باشیم یعنی درون Data_in بیت مربوط به parity آمده و بیت دوم بخش opcode ما را می سازد. پس اول از همه opcode[1] را برابر با data_in می گذاریم و counter را هم صفر می کنیم. این counter در واقع همانی است که ورودی های مربوط به A و b را می شمارد تا در جای مناسب، آن بیت ورودی را قرار دهیم و به این ترتیب یک دفعه خراب نمی شود کار. بعد از این کار، State را در حالت بعدی قرار می دهیم. سپس در کلاک بعدی، باید کارهای مربوط به Zero را انجام دهیم که opcode[0] را تشکیل می دهد و بعد state را هم میبریم روی حالتی که ورودی A را باید بگیریم.

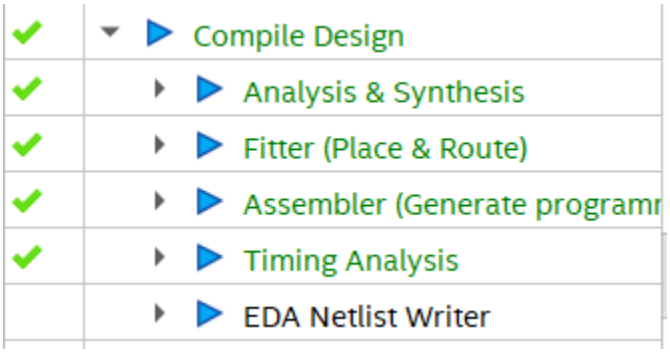
سپس در کلاک های متوالی، بیت های a را دانه دانه از Data_in می خوانیم و درون a قرار می دهیم و Counter را هم یکی زیاد می کنیم و هر زمانی که مقدار counter برابر با N شد، یعنی اینکه کل N بیت A پر شده است و باید به حالت بعدی که گرفتن B است برویم. دراین حالت گرفتن B هم دقیقا مانند قبل، می آییم دانه دانه در کلاک های متوالی بین های مربوط به B را می گیریم و داخل رجیستر مربوطه قرار می دهیم و باز هم counter را زیاد می کنیم و در نهایت هر موقع مقدار counter برابر با 2n شد، یعنی اینکه کار ما تمام شده است و دیگر ورودی هایی که می آید، برای B نیستند و الان ورودی های A و B که باید عملیاتی بین آن ها انجام دهیم، آماده هستند و نیاز است که بر حسب opcode های مختلف، بیایم کار مربوطه را انجام دهیم و نتیجه را در result بریزیم.

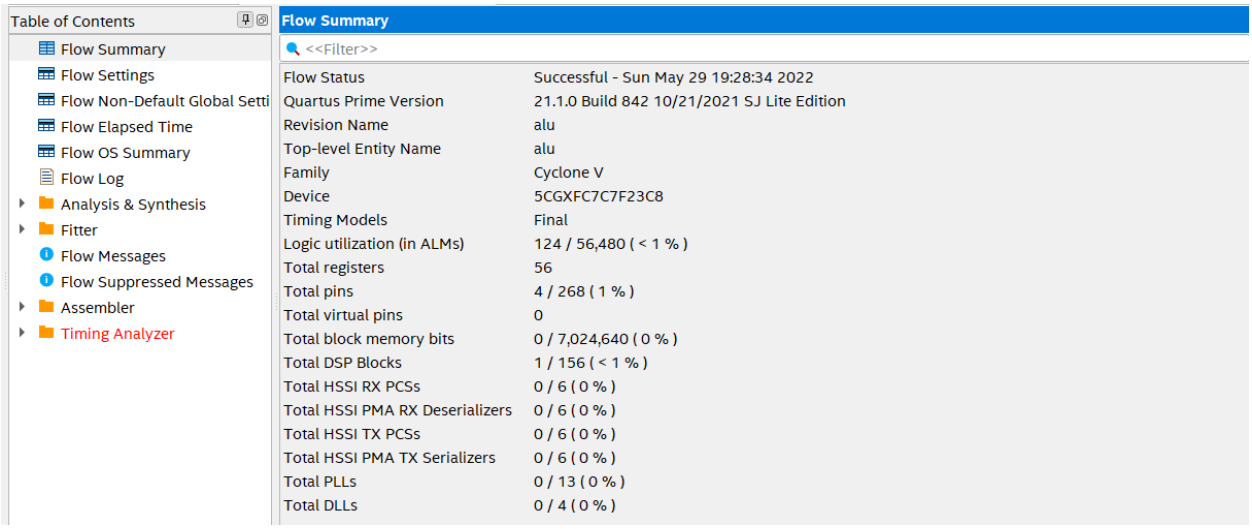
آن Exception برای وقتی است که ما تقسیم بر صفر داریم و به این ترتیب این موضوع را نشان می دهیم و خروجی را هم صفر میکنیم تا کاربر متوجه این موضوع بشود. سپس هم که بر حسب opcode و جدولی که در صورت سوال آمده، عملیات را انجام میدهیم و نتیجه را در result می ریزیم و Ready را هم یک می کنیم و باز به حالت اولیه بر میگردیم تا منتظر گرفتن ورودی های جدید باشیم.
در واقع برای state ها اینگونه عمل کردیم که حالت گرفتن parity برابر با 0، گرفتن zero برابر با 1، گرفتن a برابر با 2 و گرفتن b برابر با 3 است.
حال به توضیح تست بنچ می پردازم. در اینجا نخست reg ها و wire های لازم را ساختم و یک instance از alu گرفتم. سپس به روش سنتی کلاک را تعریف کردم. سپس داخل بلاک initial شروع کردم چندیم نمونه دادم. نخست reset را یک کردم تا مدار کامل reset شود و سپس reset را صفر کردم.
نخست عملیات جمع را تست کردم که opcode آن به شکل 10 است. پس در دو کلاک اول، باید اول یک را بدهیم و بعد صفر را تا این opcode ساخته شود. سپس باید یک را ورودی بدهیم که من 3 را به عنوان ورودی گذاشته ام و در نهایت b را باید بدهیم که من آن را یک گذاشتم. سپس خروجی محاسبه می شود و داده می شود که در شکل موج ها این موضوع را نشان میدهم که مدار دارد درست کار می کند. سپس برای جدا شدن موج ها داخل شکل موج، یک سیگنال data_in را x کردم و سپس مدار reset گشته است.
در مرجله دوم عملیات ضرب را تست کرده ام که مراحل آن دقیقا مثل جمع است فقط opcode آن همان 11 است و ورودی هم 2 و 5 داده ام و در شکل موج خروجی مربوطه را نشان میدهم.
سپس عملیات تفریق را نشان داده ام که باز هم عملیات مشخص است و opcode آن 01 است و این مورد داده شده است و 13 منهای 4 را حساب کرده ام که باز هم در شکل موج درستی محاسبات را به نمایش گذاشته ام.
در نهایت برای تقسیم، 2 حالت اصلی رخ می دهد که یا تقسیم بر صفر رخ می دهد و یا اینکه هر دو عدد صحیح هستند که برای هر دو تست مربوطه نوشته شده است. اول که opcode برای هر دو 00 است و برای حالت اول که مشکلی ندارد و 9 را تقسیم بر 3 کرده ام ولی در حالت دوم، تقسیم بر 0 را انجام داده ام که خروجی صفر می شود طبق قرارداد و آن bit مربوط به exception هم یک شده که نشان دهنده بروز مشکل به کاربر است تا بفهمد مشکلی پیش آمده است.

حال کد را با modelsim کامپایل می کنیم:



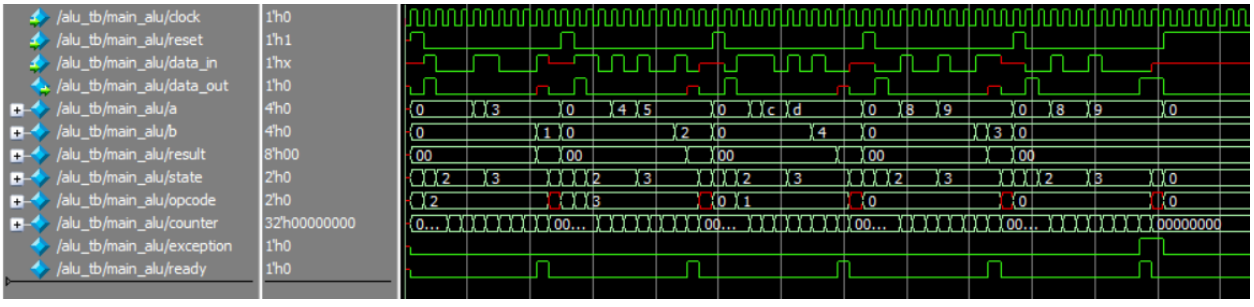
سپس کد را با کوارتوس سنتز می کنیم:





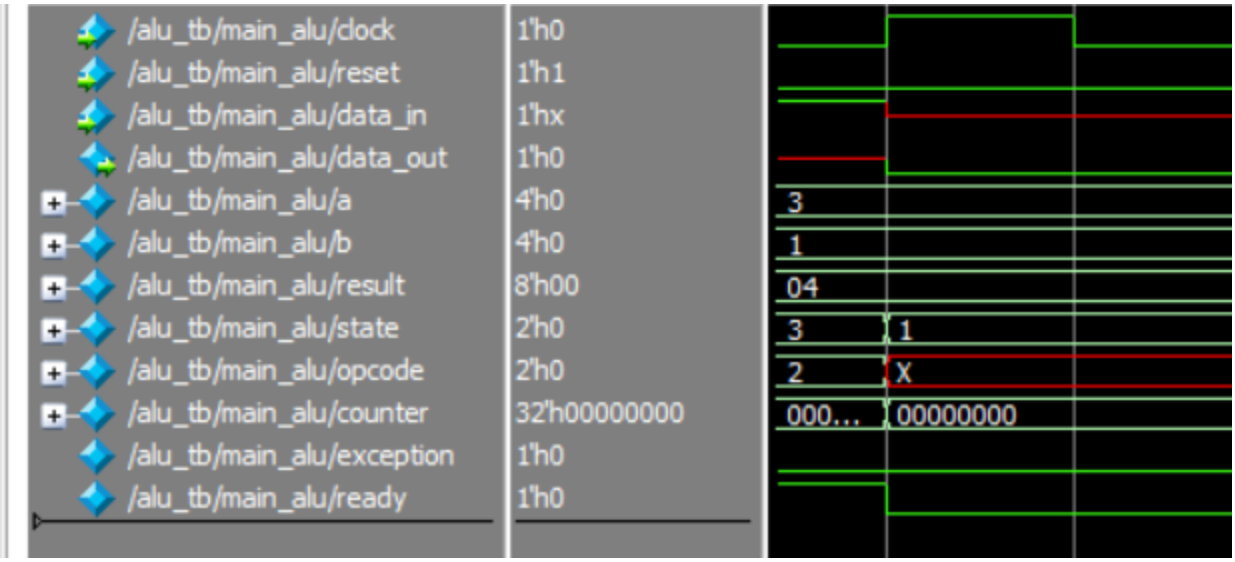
پس کامپایل و سنتز با موفقیت انجام شده است.

حال کد را simulate می کنیم:



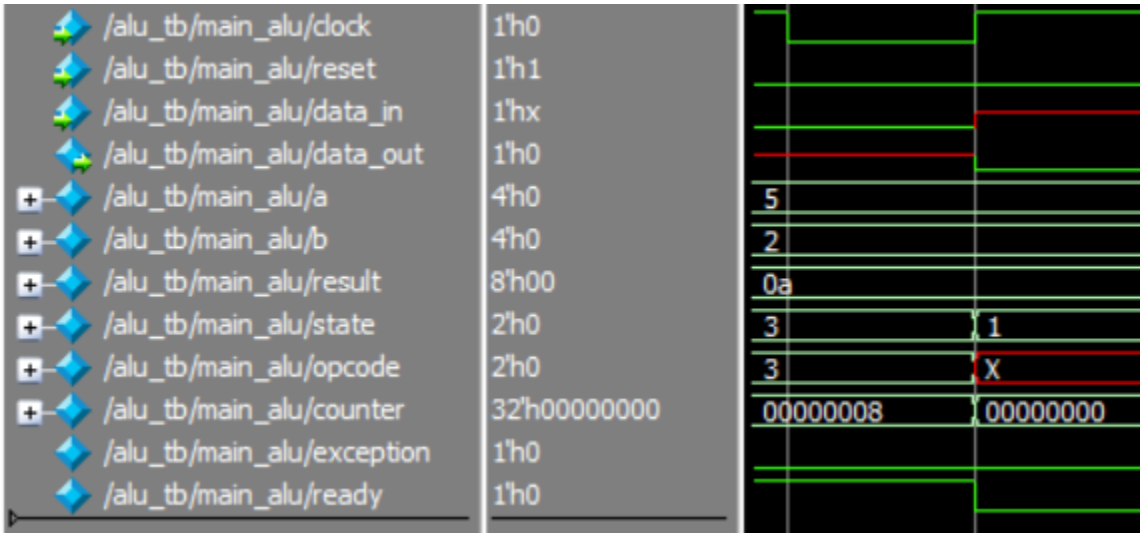
حال بخش‌های مختلف آن را بررسی می کنیم.

نخست عملیات جمع انجام می شود. همانطور که مشخص است، در 2 کلاک opcode گرفته شده است و در 8 کلاک بعدی، 2 عدد 4 بیتی گرفته شده است.



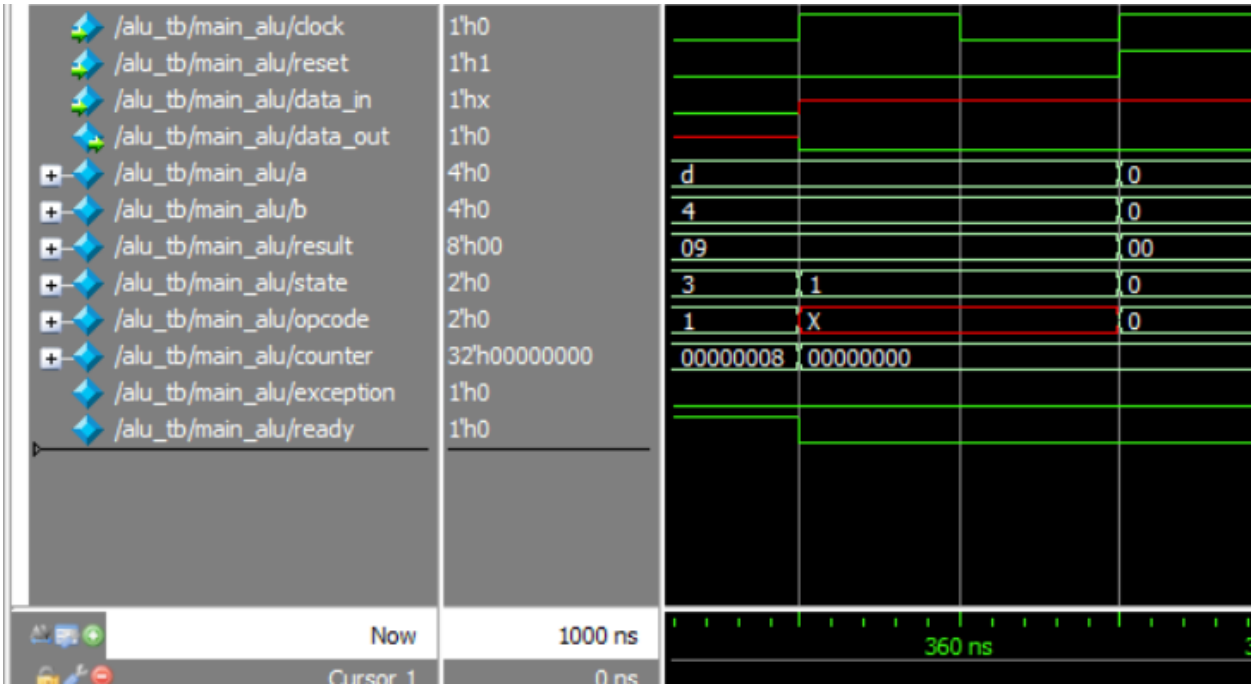
از عکس زیر هم مشخص است که در آخر تست بخش جمع که data_in را x کردیم، ورودی a به درستی 3 و ورودی b به درستی 1 است و نتیجه هم به درستی 4 گزارش شده است.

حال نوبت به ضرب می رسد که در بخشی که در انتهای آن data_in شده x شکل موج به شکل زیر است:



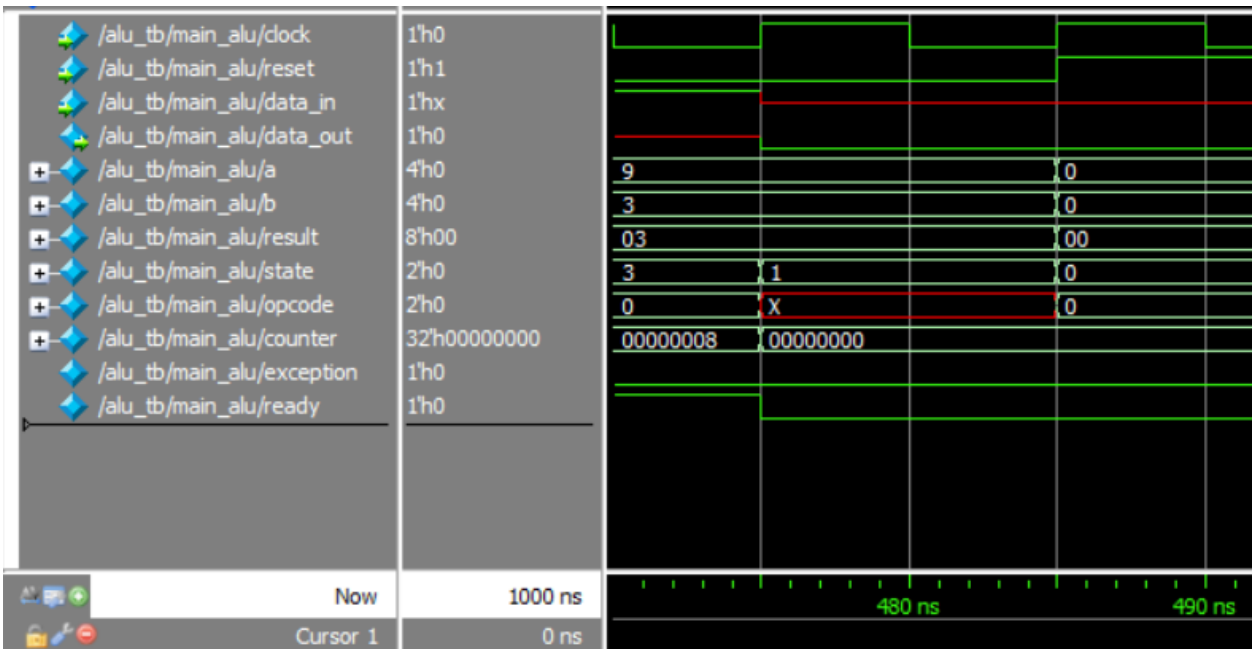
که a شَده 5 و b شده است 2 که در نتیجه خروجی شد 10 شده است و باز هم گرفتن opcode 2 کلاک و گرفتن دو عدد 8 کلاک روی هم زمان گرفته است.

حال به بخش تفریق می رسیم که شکل پایان آن که data_in شده x به شکل زیر است:



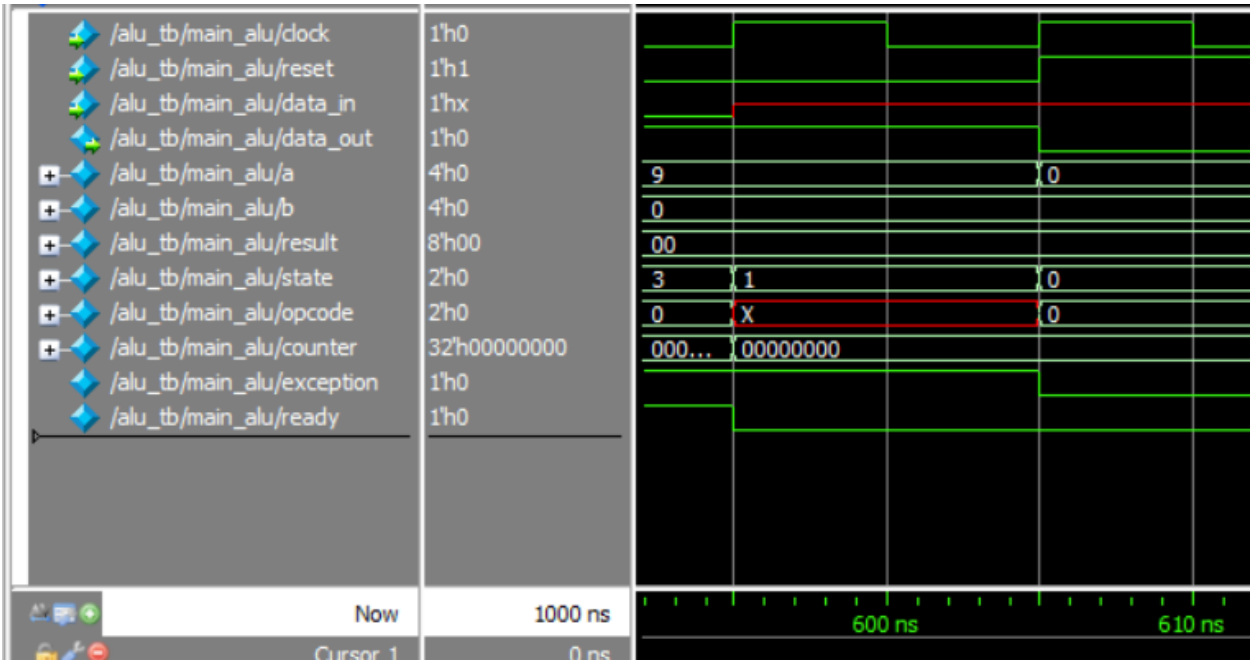
در اینجا هم 13 منهای 4 شده است و نتیجه به درستی 9 شده است و خب باز هم گرفتن opcode دو کلاک و گرفتن هر کدام از دو عدد، 4 کلاک زمان گرفته است.

حال به بخش تقسیم میرسیم که شکل آن برای حالت غیر صفر مانند زیر است:



که باز هم a شده 9 و b شده است 3 و حاصل هم خب 3 می شود به درستی و کلاک ها اینجا هم رعایت شده اند.

حال تقسیم بر صفر را چک می کنیم:



در اینجا a همان 9 مانده است ولی مقسوم علیه شده صفر و در نتیجه بیت exception شده یک تا نشان دهد مشکلی پیش آمده و خروجی هم 0 شده و جوابی نشان نمی دهد. پس در این حالت هم به درستی کار می کند و کلاک ها هم رعایت شده است. پس حالت های مهم مدار تست شده است و بقیه حالت ها برای هر 4 عمل اصلی، بقیه موارد به همین شکل است.