

به نام خدا



تکلیف سری چهارم fpga

محدثه غفوری (9632133)

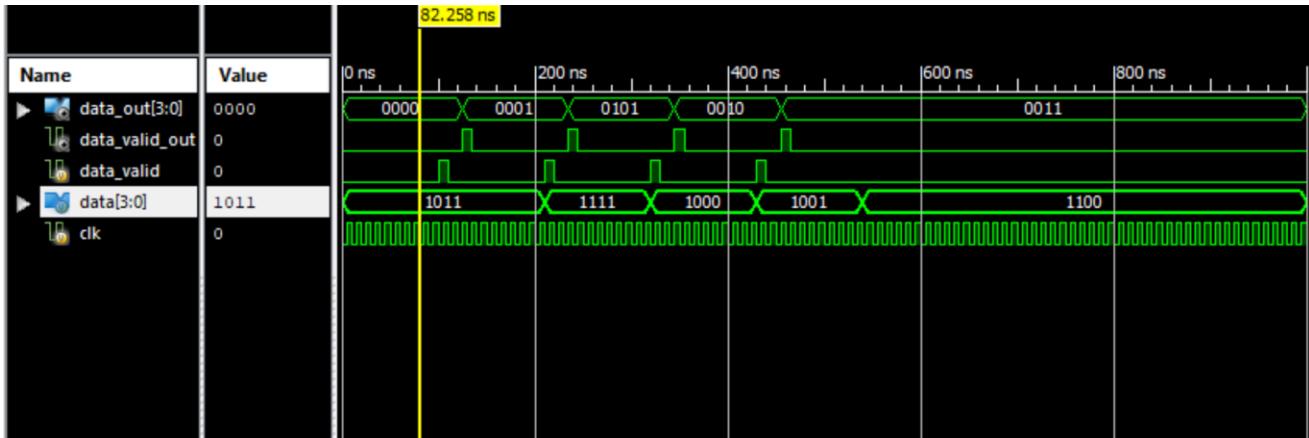
سوال 1) در مژول Q1 باید هنگام فعال شدن `data_valid` داده دریافت شود . از آنجایی که سیگنال `data_valid` تنها یک پالس فعال میشود نیاز به متغیری بنام `flag` داریم که بوسیله این سیگنال فعال شود و عملیات ارسال داده ورودی به مژول Q2 را اغزار کند . بنابراین با یک شدن سیگنال `data_valid` در لبه ی بالارونده کلک بعدی مژول Q1 با یک کردن `io_select` این داده را روی پایه ی خروجی خود که به مژول Q2 متصل است قرار میدهد و یک کلک صبر میکند تا عملیات `xor` در مژول Q2 انجام شود و در لبه ی بالارونده کلک بعدی `io_select` را صفر میکند تا پایه ی `data_out` ورودی شود و داده ی مژول Q2 را دریافت میکند و روی خروجی `data_out` قرار میدهد و با یک کردن `data_valid_out` در این لحظه نشان میدهد که این خروجی معتبر است و سپس با یک کردن پایه ی `data_valid` را خروجی میکند و منتظر فعال شدن `data_io_select` و دریافت ورودی جدید میشود . در مژول Q2 هنگامی که `io_select` یک باشد پورت `data_io` برای این مژول حکم ورودی را دارد بنابراین با دریافت در اولین لبه ی بالارونده کلکی که از مژول Q1 گرفته است ورودی را ذخیره میکند و با صفر شدن `io_select` در لبه ی بالارونده کلک بعدی توسط مژول Q1 داده ای که ذخیره شده بود با $4'b1010$ بیت به بیت `xor` میشود و روی پایه ی خروجی یعنی `data_io` قرار گرفته و تحويل به مژول Q1 داده میشود بدین ترتیب طبق خواست سوال مژول Q2 دقیقا یک کلک بعد از دریافت داده و انجام عملیات روی ان حاصل را به مژول Q1 باز میگردداند

توجه شود که برای جلوگیری از اتصال کوتاه و اسیب رسیدن به پورت مشترک این دو مژول یعنی `data_io` در صورتی که `io_select` یک باشد مژول Q2 روی این پورت مشترک چیزی قرار نمیدهد در واقع میشود اما در حقیقت این پایه حاوی اطلاعاتی است که از مژول Q1 میرسد . و همچنین با صفر بودن مژول Q1 نباید چیزی روی این پایه قرار دهد بنابراین `high impedance` میشود ولی اطلاعات ارسالی توسط مژول Q2 روی این پورت قرار دارد

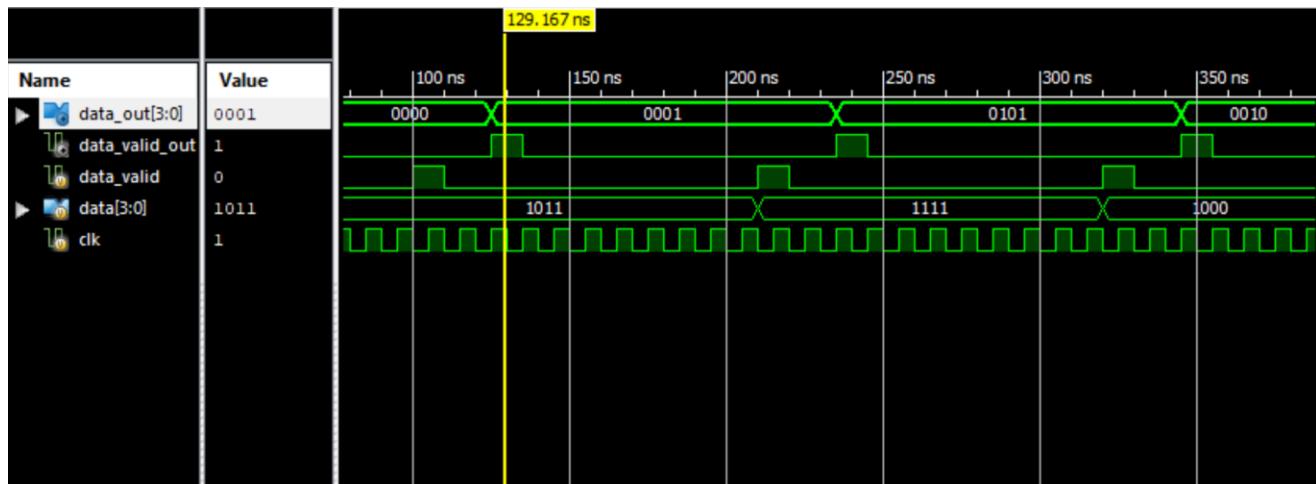
چون تنها باید یک تاپ مژول وجود داشته باشد و برای نوشتن تست بنج نیاز به فقط یک مژول داریم این دو مژول را در یک تاپ مژول قرار میدهیم (از انها در تاپ مژول ایستنس میگیریم) و برای این تاپ مژول تست بنج مینویسیم این تاپ مژول ورودی های مژول Q1 و همچنین خروجی هایی از مژول Q1 را که به مژول Q2 نمیروند را به عنوان خروجی اتخاذ میکند و مابقی خروجی های مژول Q1 به ورودی های مژول Q2 وصل میشوند

نتایج شبیه سازی و waveform ها

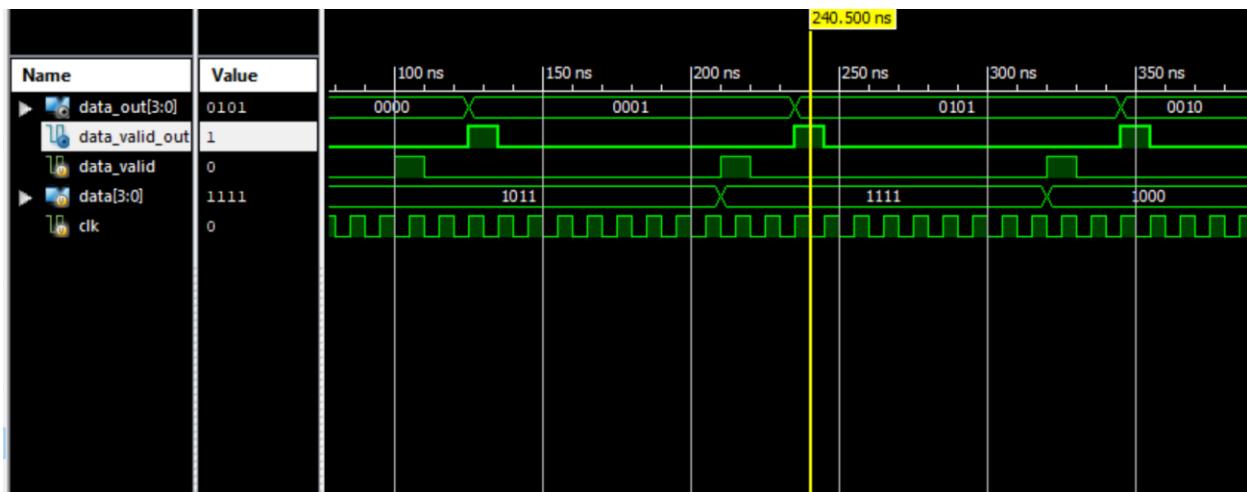
که خروجی به ازای ان نخواهیم داشت در واقع این داده دریافت نمیشود $data_valid = 0$ و $data = 4'b1011$



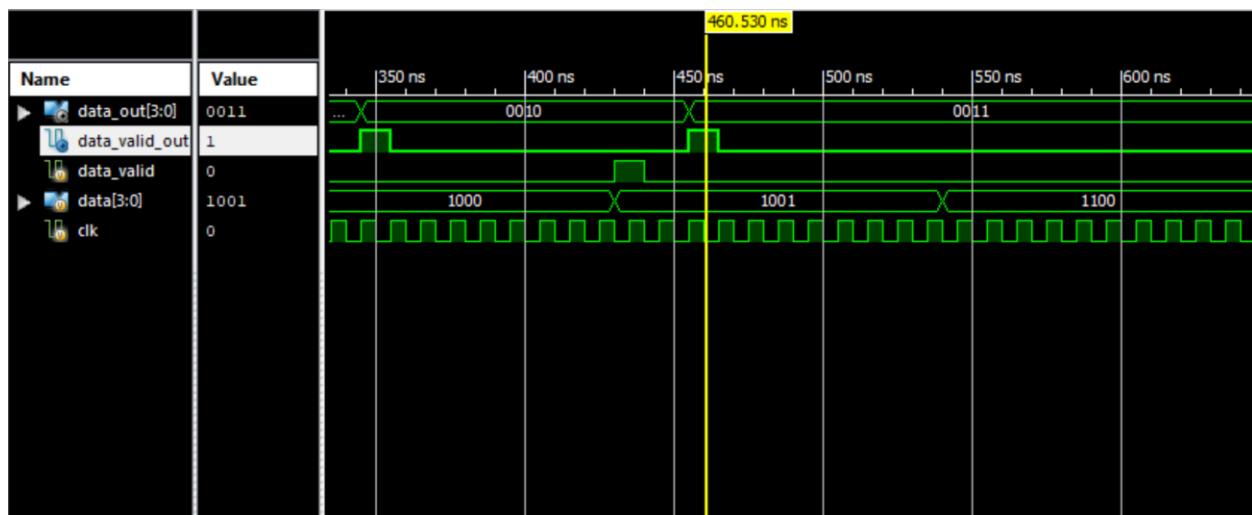
که این بار داده دریافت شده و به ازای آن خروجی معتبر داریم



data = 4'b1111 , data_valid = 1

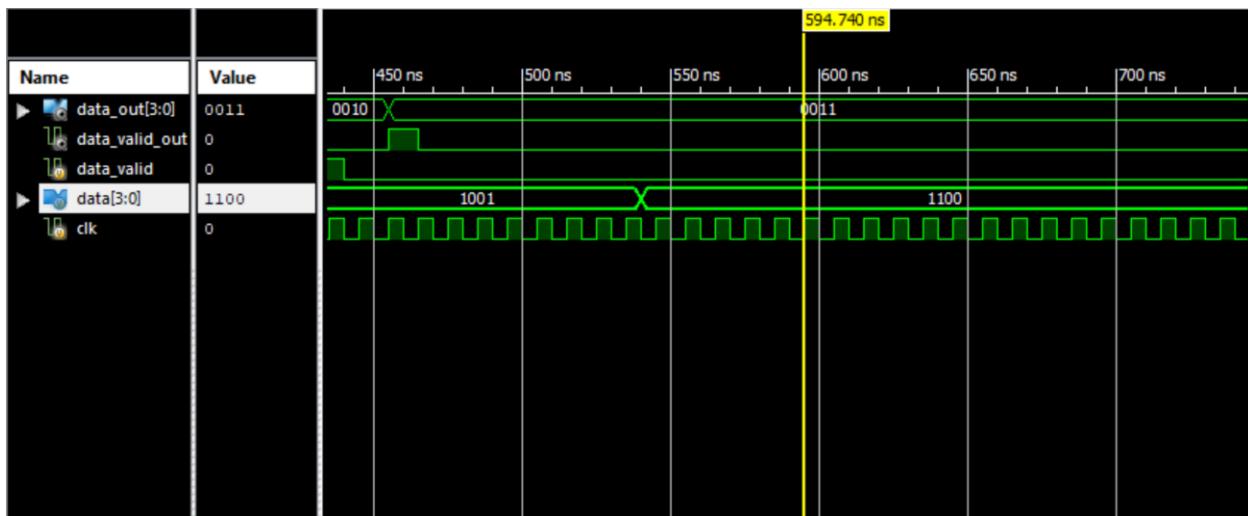


Data=1001 , data_valid=1



که این داده دریافت نشده و به ازای آن خروجی نداریم و خروجی موجود هنگام اعمال

این داده معتبر نیست



سوال 2) در این سوال برای تولید مموری ها از **block memory generator** استفاده میکنیم که برای مموری **a** از مموری با عمق **64** و عرض بیت **8** استفاده میکنیم و برای مموری **C** که حاصل ضرب خانه های دو مموری قبل دران قرار میگیرد از مموری با عمق **64** و عرض بیت **16** استفاده میکنیم (هر دو نوع مموری از نوع **simple dual port ram**) میباشند . در این نوع مموری ها با اولین پایه ادرس و کلاک اول مقداردهی های اولیه را انجام میدهیم و خانه های این بلوک های حافظه را پر میکنیم و با دومین کلاک و پایه ادرس داده های این بلوک ها را به ترتیب میخوانیم بنابراین با فعال شدن **start_initialize** پایه های **wra, web, wea** دو بلوک حافظه **a,b** را یکی یکی داده های ورودی ماذول از دو پورت ورودی **a, b** به ورودی های این مموری **a,b** میکنیم (از طریق افزایش پایه های ادرس **addrb** و **addra** و اتصال پورت های ورودی **a,b** به ورودی های این دو بلوک حافظه و فعال بودن پایه **i** نوشتن) . بلافاصله بعد از پر کردن اخرین خانه **i** حافظه ها عملیات نوشتن روی این مموری ها را متوقف میکنیم و منتظر امدن سیگنال **start_multiply** میشویم . با فعال شدن این سیگنال باید داده های داخل بلوک مموری **b** را یکی یکی با افزایش پایه های ادرس **addrbb** و **addraa** از طریق خروجی های این دو بلوک (**douta, doubt**) خوانده و حاصل ضرب این دو بلوک را در بلوک مموری **C** مینویسیم . برای فعال کردن **a,b** پایه **wec** از سیگنال کنترلی **start_multiply** استفاده میکنیم . همانند بلوک های مموری **a,b** که دو پورت **a,b** را به ورودی این بلوک ها متصل کردیم که بلافاصله با فعال شدن پایه نوشتن انها مقداردهی حافظه ها شروع شود برای بلوک **C** نیز حاصل ضرب خروجی های بلوک های **a,b** را به ورودی آن وصل میکنیم تا بلافاصله با فعال شدن پایه نوشتن ان مقداردهی به این بلوک نیز انجام گیرد

توجه شود که در هر مرحله حاصل عملیات ضرب (در واقع مقدار هر خانه **i** بلوک **C**) را بعنوان خروجی ماذول اصلی در نظر گرفته ایم که با افزایش پایه **i** ادرس **addrC** و فعال بودن **wec** مقدار ها از خروجی بلوک های **a,b** (افزایش پایه های ادرس های **addrAA, addrBB** در هر لبه **i** منفی کلاک باعث میشود قبل از نوشتن مقدار ها روی

حافظه ادرس مورد نظر اماده باشد و تداخلی بین نوشتن داده ها روی ادرس های مختلف پیش نیاید) خوانده شده و در

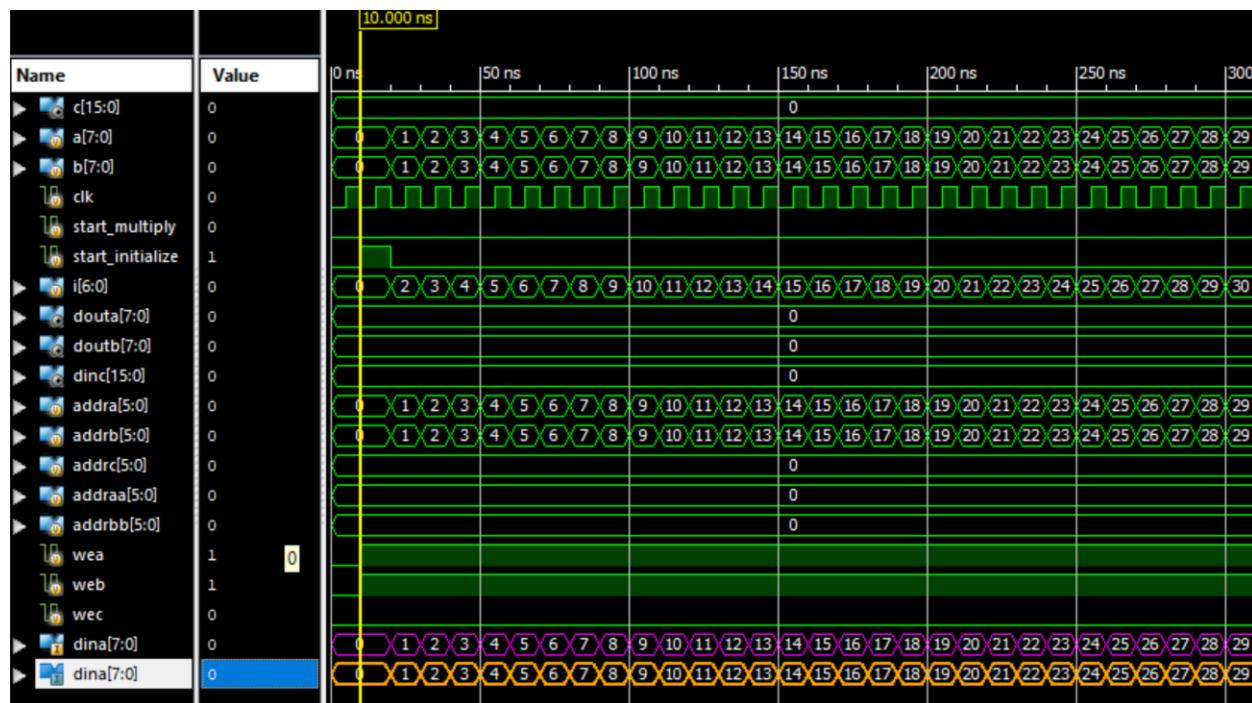
بلوک C ذخیره میشوند

سیگنال کنترلی **start_multiply** بعد انجام عملیات مقدار دهی بلوک های حافظه **a,b** فعال میشود (مقدار دهی این بلوک ها و غیر فعال کردن **wea , web** حدودا 65 کلاک به طول می انجامد) که در غیر این صورت نیز تداخلی در روند مقداردهی ندارد زیرا از خانه های اول حافظه شروع به خواندن و ضرب انجام میشود و همزمان نیز مقداردهی میتواند انجام گیرد

نتایج شبیه سازی و waveform ها

برای سری داده هی **a** با شروع از 0 تا 63 و **b** با شروع از 0 تا 63

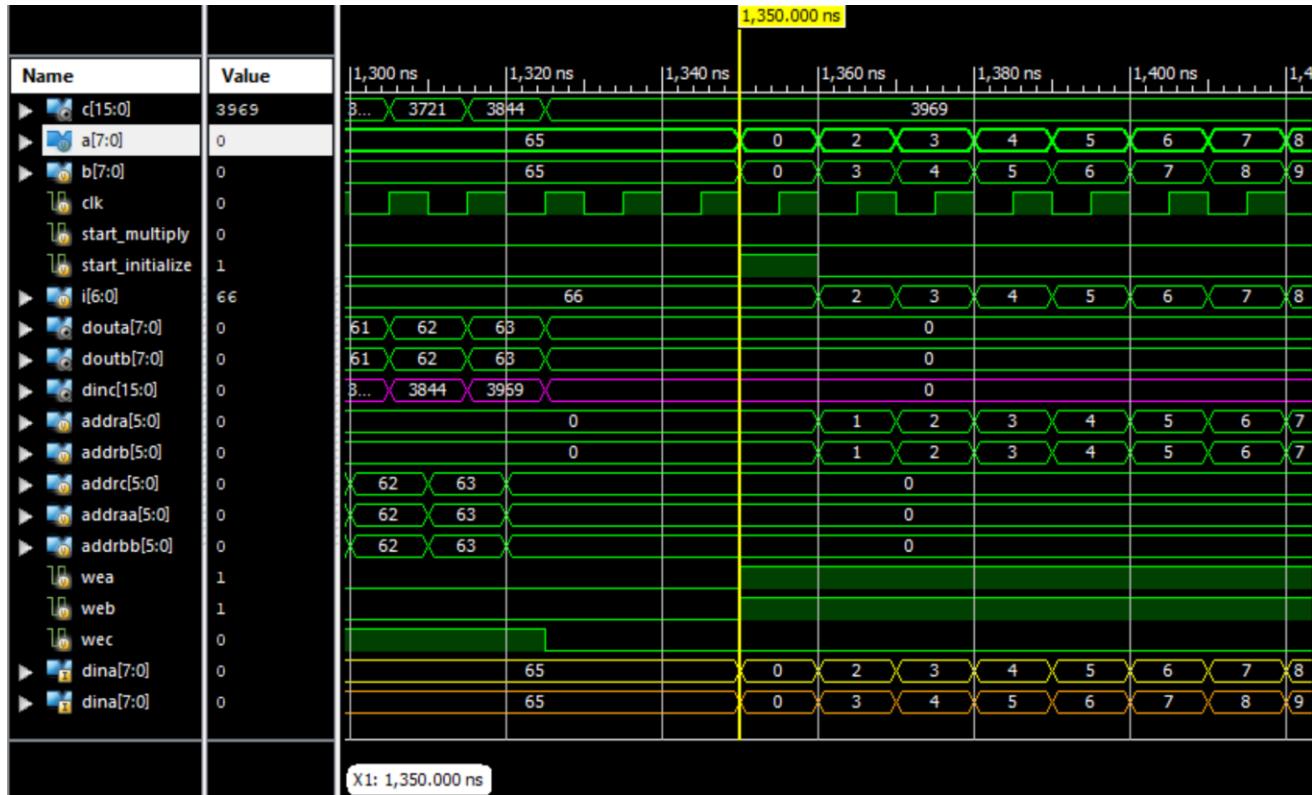
نوشتن داده ها روی بلوک های **a , b**



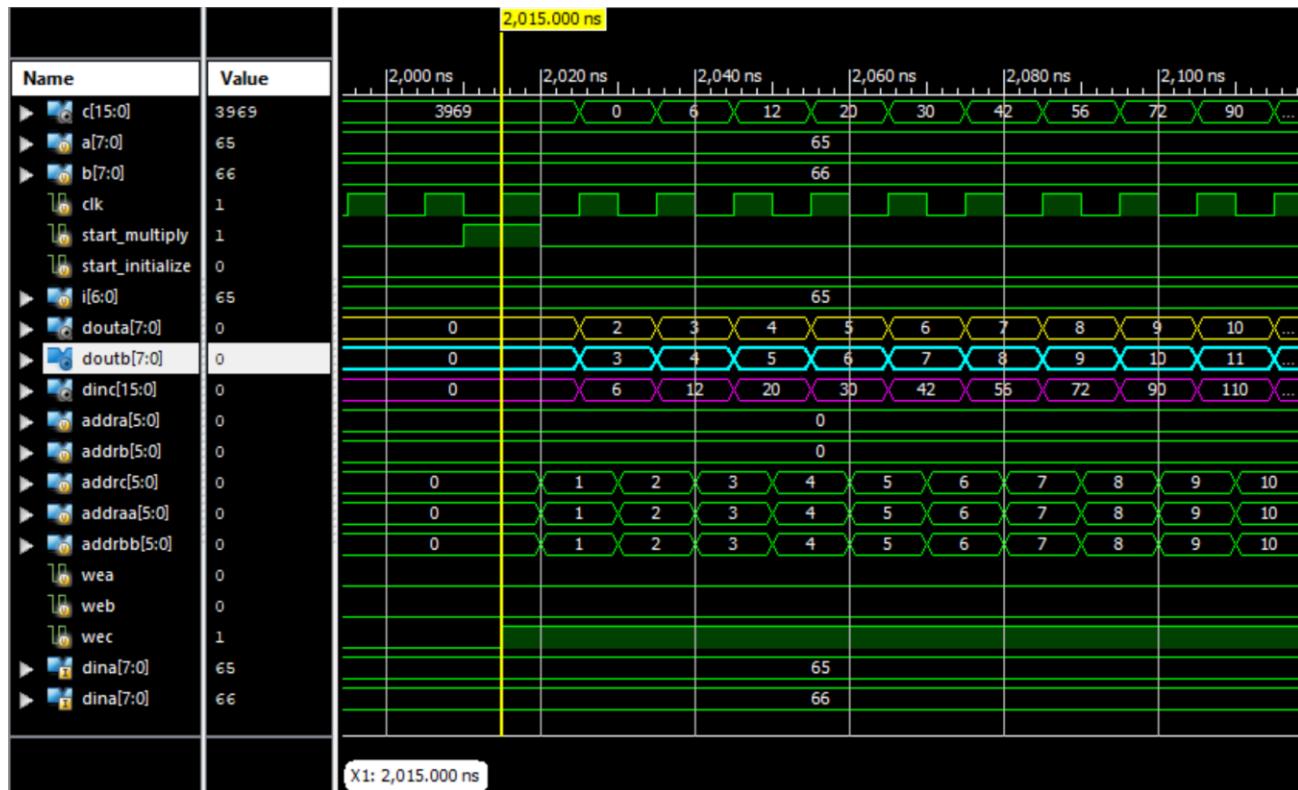
نوشتن حاصل ضرب ها روی بلوک C



سری داده‌ی بعدی با $b=0,3,4,5,6,\dots$ و $a=0,2,3,4,\dots$



نوشتن حاصل ضرب ها روی بلوك C



سوال 3) این سوال پیاده سازی سوال دو با کد نویسی بوسیله ارایه های دو بعدی است . بنابراین دو ارایه **a** ، **b** را تعریف میکنیم که تعداد انها 64 تا و طول هر عضو آن 8 تاست . با فعال شدن **start_initialize** فلگ **cnt** یک میشود و عملیات مقدار دهی به خانه های دو ارایه اغاز میشود . این عملیات بوسیله ی یک کانتر **cnt** انجام میشود که در هر لبه ی بالارونده ی کلاک با زیاد شدن این کانتر مقدار پورت های ورودی **a**، **b** به خانه های ارایه های **aprim** ، **bprim** منتقل میشود و با اتمام مقدار دهی به خانه های این ارایه ها **cnt** صفر میشود و منتظر فعال شدن سیگنال **start_multiply** می مانیم تا با یک شدن ان فلگ **start_multi** یک شود و بوسیله ی کانتر **cnt1** که برای کنترل عملیات ضرب استفاده میشود ارایه های دو بعدی **cprim** که 64 عضو بطول 8 دارد شروع به پر شدن کند و حاصل ضرب خانه های **cprim** در **aprim** ، **bprim** ریخته شود . با اتمام قرار دادن حصل عملیات های ضرب در خانه های **cprim** کانتر **cnt1** صفر شده و منتظر سری ورودی های بعدی میشویم که با یک شدن **start_initialize** وارد میشوند

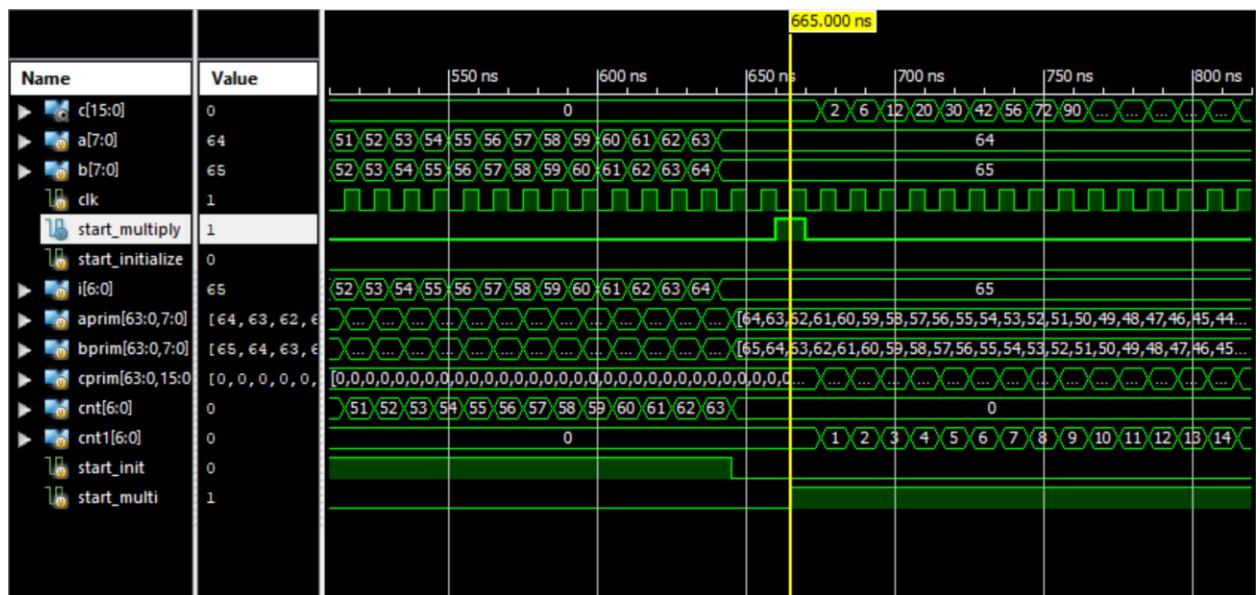
با توجه به اینکه **lut** ها منابع ارزشمندی در **fpga** هستند **XST** در شرایطی که بتواند بجای استفاده از انها از منابع دیگری استفاده میکند . در همین سوال نیز با وجود اینکه کد نویسی با ارایه ها انجام شده است اما **XST** بجای استفاده از **LUT** ها از بلاک مموری ها برای ذخیره ای اطلاعات استفاده کرده است که در تصویر زیر نیز و شماتیک سوال کاملا قابل مشاهده است

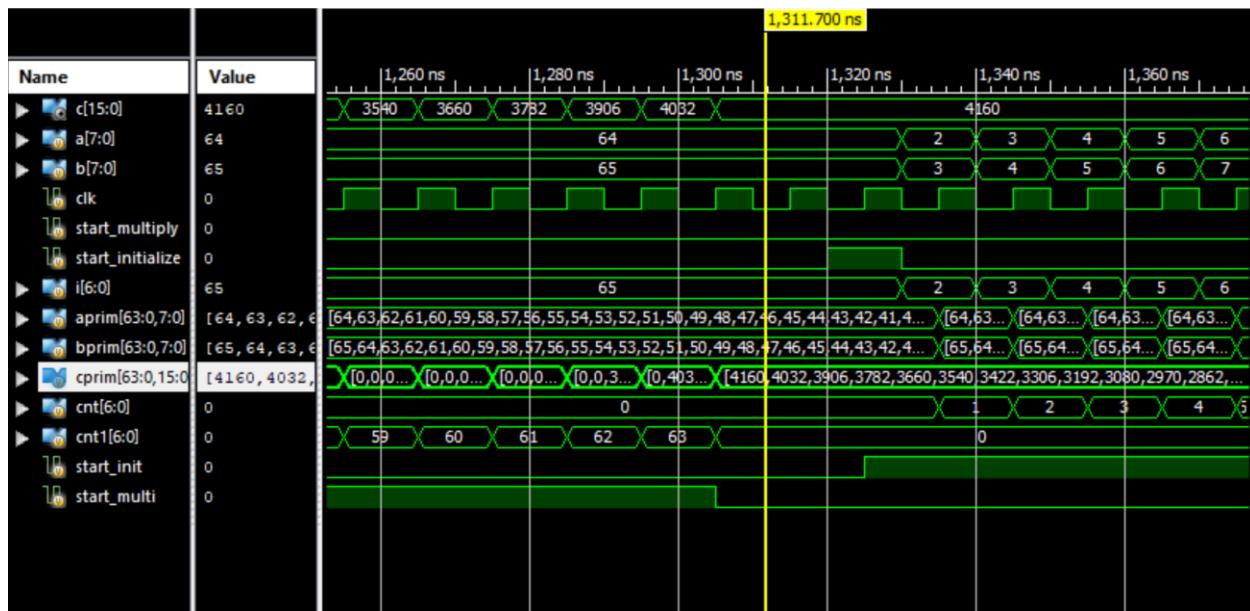
Device Utilization Summary			
Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	21	11,440	1%
Number used as Flip Flops	21		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	40	5,720	1%
Number used as logic	15	5,720	1%
Number using O6 output only	8		
Number using O5 output only	0		
Number using O5 and O6	7		
Number used as ROM	0		
Number used as Memory	24	1,440	1%
Number used as Dual Port RAM	24		
Number using O6 output only	24		
Number using O5 output only	0		
Number using O5 and O6	0		
Number used as Single Port RAM	0		
Number used as Shift Register	0		

نتایج شبیه سازی و waveform ها

برای سری داده $b=2,3,4,\dots$ و $a=1,2,3,\dots$

c[15:0]	4160
a[7:0]	64
b[7:0]	65
clk	0
start_multiply	0
start_initialize	0
i[6:0]	65
aprim[63:0:7:0]	[64, 63, 62, 61, 60, 59, 58, 57, 56, 55, 54, 53, 52, 51, 50, 49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
bprim[63:0:7:0]	[65, 64, 63, 62, 61, 60, 59, 58, 57, 56, 55, 54, 53, 52, 51, 50, 49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
cprim[63:0:15:0]	[4160, 4032, 3906, 3782, 3660, 3540, 3422, 3306, 3192, 3080, 2970, 2862, 2756, 2652, 2550, 2450, 2352, 2256, 2162, 2070, 1976, 1880, 1784, 1688, 1592, 1496, 1400, 1304, 1208, 1112, 1016, 920, 824, 728, 632, 536, 440, 344, 248, 152, 64, 0]
cnt[6:0]	0
cnt1[6:0]	0
start_init	0
start_multi	0





سری داده های $a=2,3,4,\dots$, $b=3,4,5,\dots$



سوال 4) در این سوال ساخت بلوک های حافظه برای a, b دقیقاً مانند سوال دو میباشد. بوسیله هی پایه ادرس های $addra, addrb$ مقادیر پورت های ورودی مازول را به خانه های حافظه منتقل میکنیم و بوسیله هی پایه ادرس های $addrAA, addrBB$ مقادیر خانه های حافظه را یکی یکی خوانده و عملیات ضرب را روی خروجی انها انجام میدهیم و

در انتها تمامی حاصل ضرب های بلوک های حافظه را باهم جمع میکنیم و در یک متغیر میریزیم و انرا بعنوان خروجی نمایش میدهیم

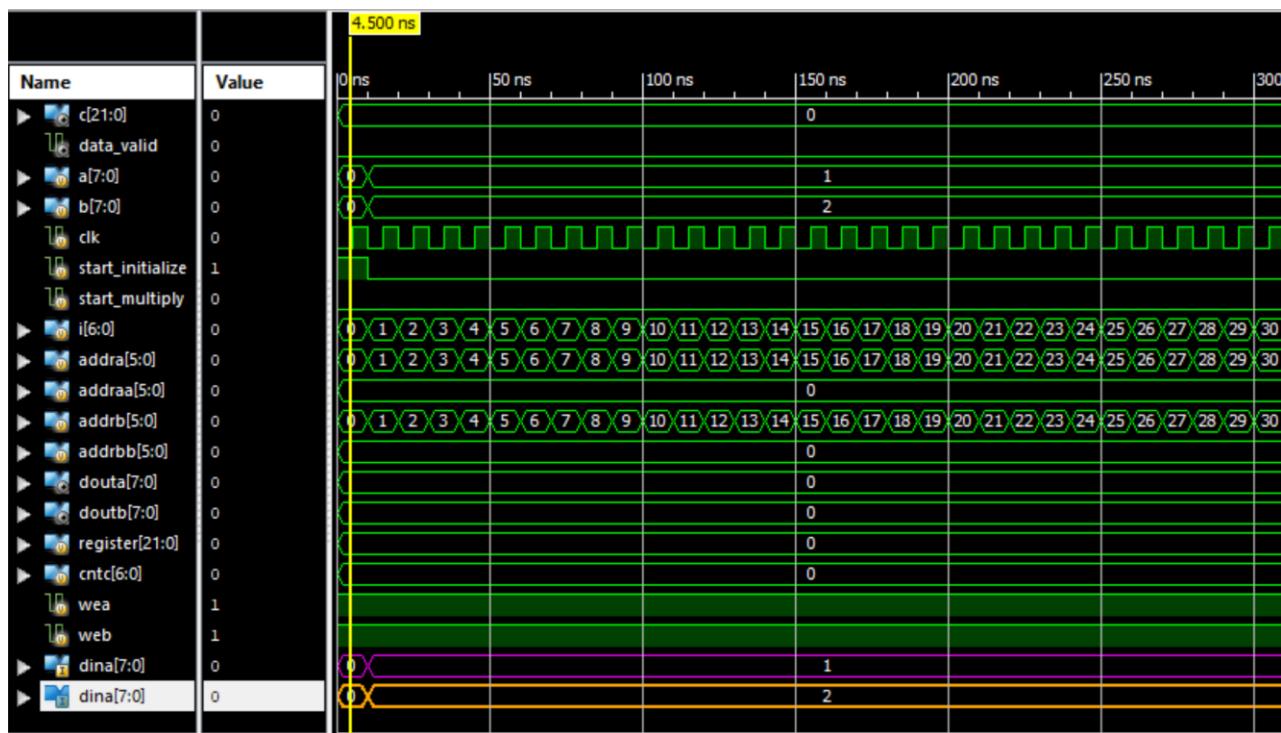
برای تعیین تعداد بیت های مورد نیاز برای پورت خروجی **c** میدانیم تمامی خانه های حافظه **a**, **b** هشت بیتی هستند یعنی ماکسیمم مقدار 255 را به خود میگیرند که حاصل ضرب انها ماکسیمم $255 \times 255 = 64 \times 255 \times 255$ است که چون تعداد این خانه ها 64 است ماکسیمم حالت عدد بدست امده برای **c** عدد 21:0 را بعنوان خروجی داریم

یک بافر بنام **register** تعریف میکنیم و در هر لبه ای بالارونده کلاک مقدار حافظه های **a**, **b** را که از **douta**, **doutb** خواندیم به همراه مقادیر قبلی ضرب شده جمع کرده و هر بار با ریختن در این بافر مقدار انرا **update** میکنیم و در نهایت پس از جمع زدن تمامی حاصل ضرب ها مقدار این بافر را در خروجی ریخته و با یک کردن **data_valid** داده ای معتبر را نمایش میدهیم و برای مقدار دهی های بعد بافر و کانتر ها و فلگ ها را صفر میکنیم

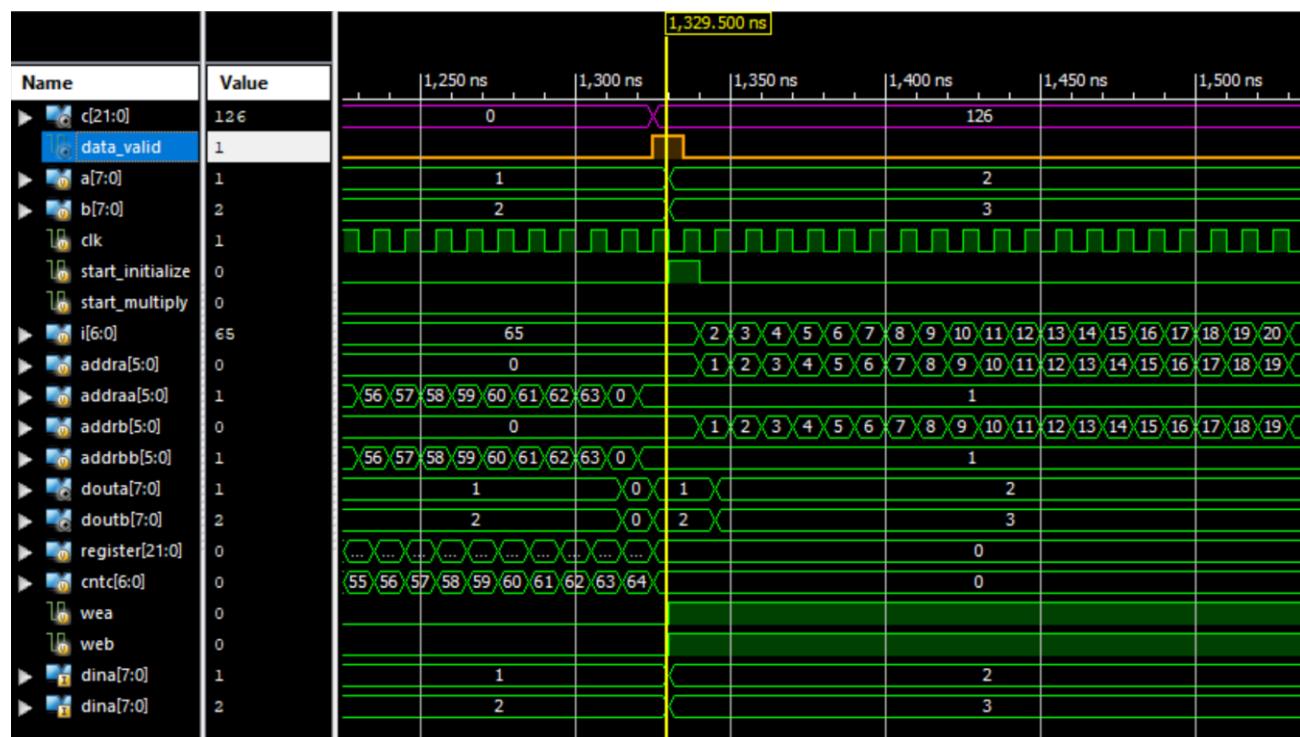
نتایج شبیه سازی و waveform ها

برای سری داده ورودی $b=0,2,2,2,2,\dots$ و $a=0,1,1,1,1,\dots$

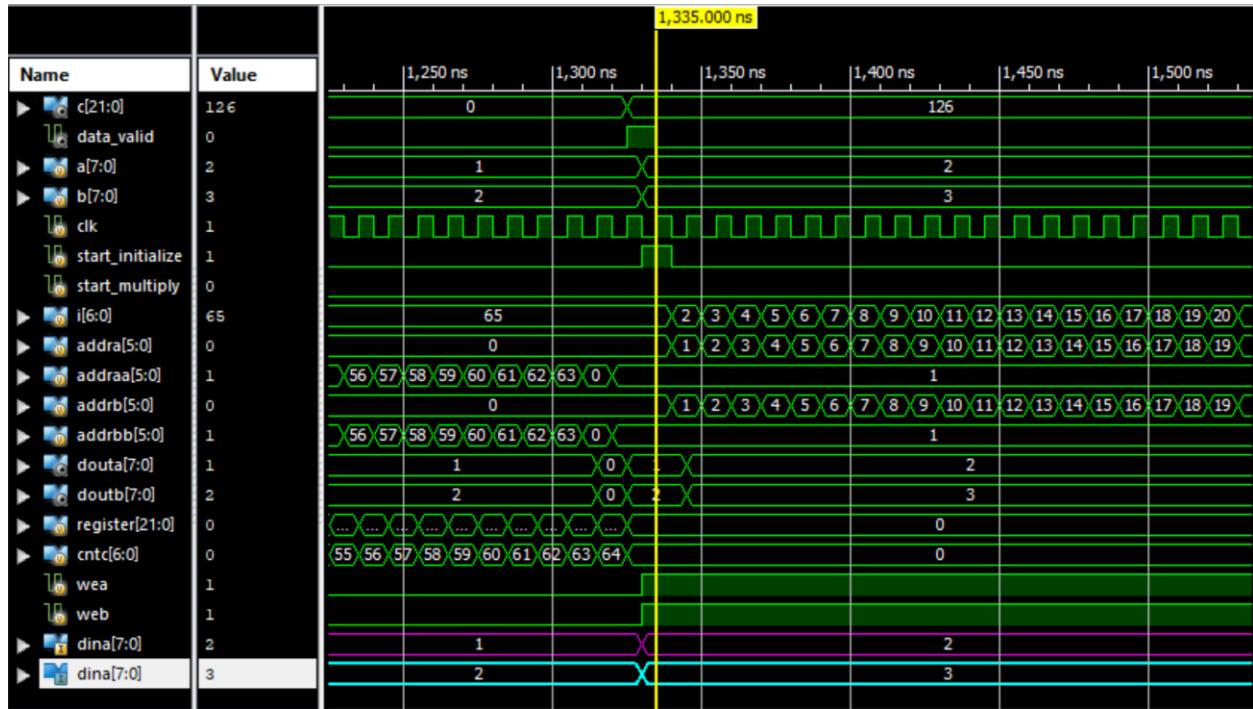
مقدار دهی به بلوک های حافظه



حاصل عملیات (63*2) در 63 خانه 2 قرار دارد



برای سری داده $b=3,3,3,\dots$ و $a=2,2,2,\dots$



حاصل عملیات (64×6) در هر 64 خانه 6 قرار دارد



سوال 5) در ساخت فیفو موردنظر کلاک مربوط به نوشتن اطلاعات روی انرا به وسیله‌ی خروجی **copy_clk250** گرفته شده از **DCM** تامین میکنیم همچنین برای خواندن اطاعات از فیفو از کلاک **copy_clk100** گرفته شده از **DCM** استفاده میکنیم . از انجایی که تا قبل از اماده شدن کلاک هایی که عملیات خواندن و نوشتن فیفو را کنترل میکنند نمیتوان داده ای در فیفو نوشت یا از آن خواند پایه‌ی ریست فیفو را به **locked** متصل میکنیم تا قبل از اماده شدن کلاک ها فیفو ریست و خالی باشد و پایه‌ی ریست آن فعال باشد اما با اماده شدن کلاک ها و یک شدن **locked** حال ریست خارج شده و اماده‌ی دریافت داده شود . همچنین با توجه به سوال ورودی فیف به پورت **data_in** مازول متصل است و تنها درصورتی که **data_valid** فعال باشد داده داخل فیفو نوشته میشود بنابراین **wr_en** فیفو را به **rd_en** متصل میکنیم و تا زمانی که داده داخل آن وجود دارد و فیفو خالی نیست از آن داده میگیریم (**rd_en** به **empty** ~متصل میکنیم)

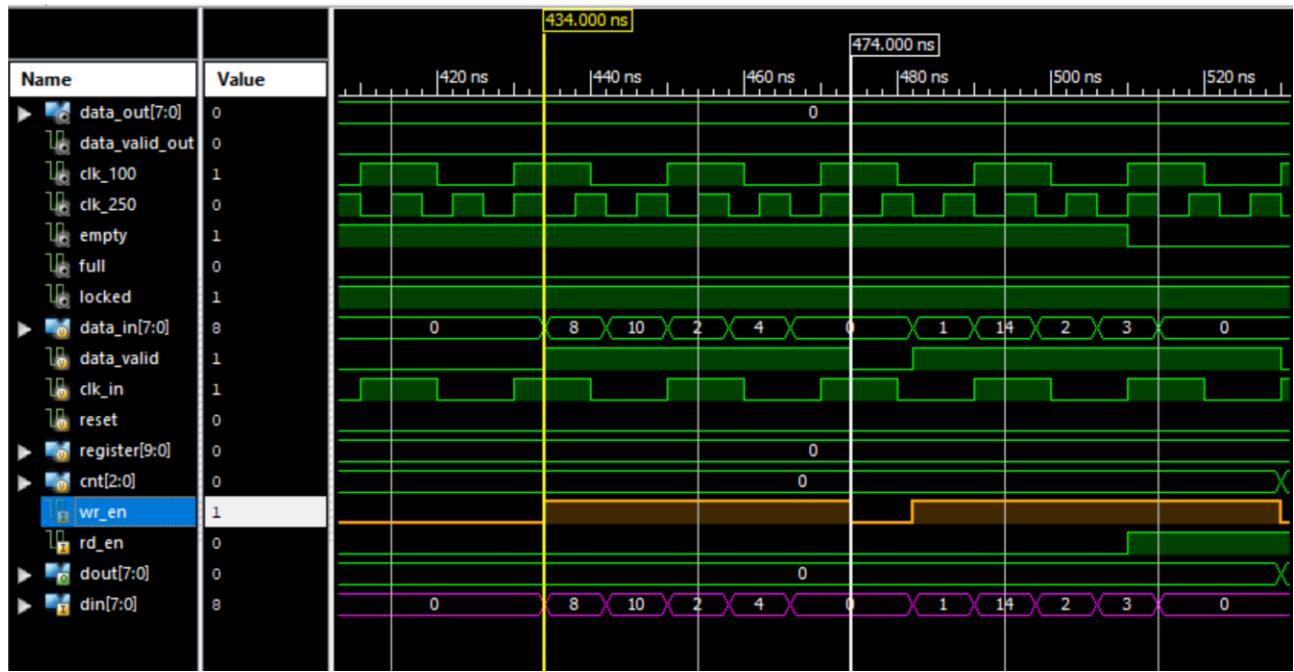
از انجایی که پورت‌های دیتا کانت خواندن و نوشتن که مربوط به خود فیفو هستند دقیق نمیباشند عملیات دریافت چهار داده از فیفو را با متغیری بنام **cnt** در لبه‌های بالارونده کلاک خواندن فیفو (**copy_clk100**) انجام میدهیم و در چهار لبه این کلاک از فیفو داده دریافت کرده و بعد از دریافت داده‌ها در لبه‌ی بعدی میانگین این داده‌ها محاسبه و به خروجی منتقل شده و **data_valid_out** یک میشود . در اینجا از بافر **register** برای جمع زدن این داده‌ها استفاده میکنیم .

همچنین برای انتقال کلاک‌های تولید شده توسط **DCM** انها را بوسیله دستور **assign** به پورت‌های خروجی **clk_250** و **clk_100** متصل میکنیم

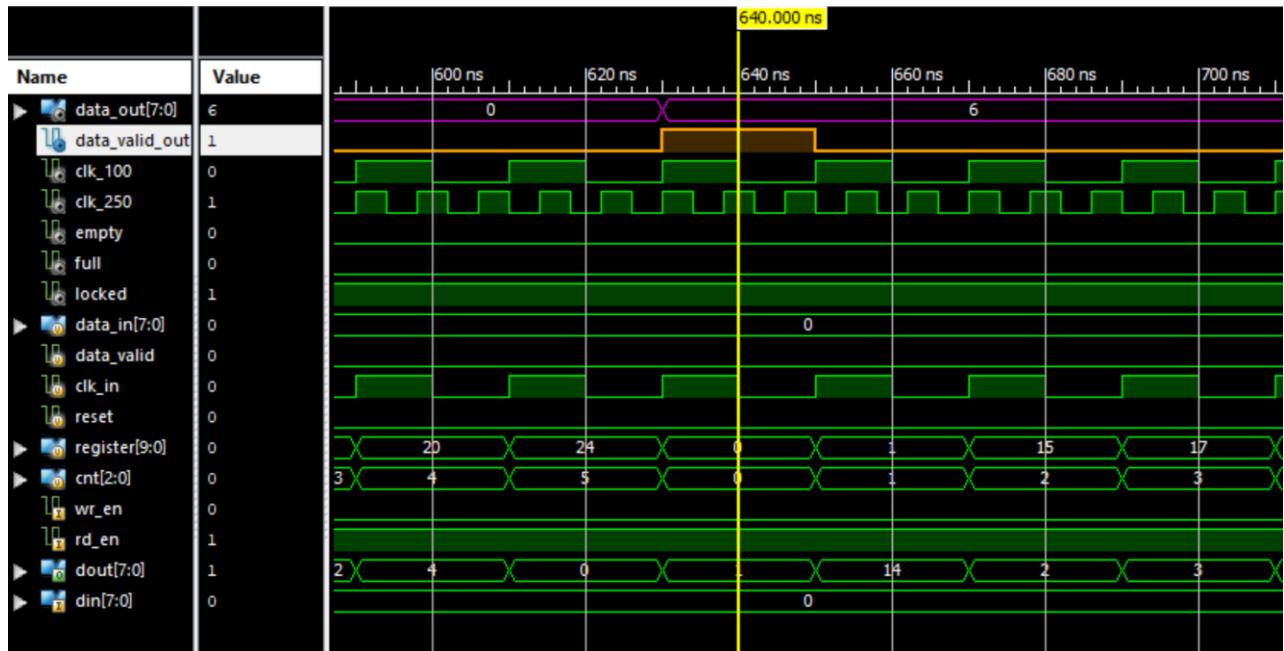
در تست بنج طرح تا قبل فعال شدن کلاک‌ها داده‌ای وارد نمیکنیم (**wait(locked)**) و پس از آن برای اماده شدن فیفو باید به میزانی تاخیر وارد کنیم و در نهایت در لبه‌های منفی کلاک نوشتن فیفو (**clk_250**) داده‌های ورودی را همراه با فعال کردن **data_valid** وارد میکنیم و در انتهای هر سری داده چندین صفر به انتهای آن اضافه میکنیم که هنگام خواندن از فیفو خالی نشود و شرط خواندن از روی آن (**empty** صفر باشد) برقرار بماند

نتایج شبیه سازی و waveform ها

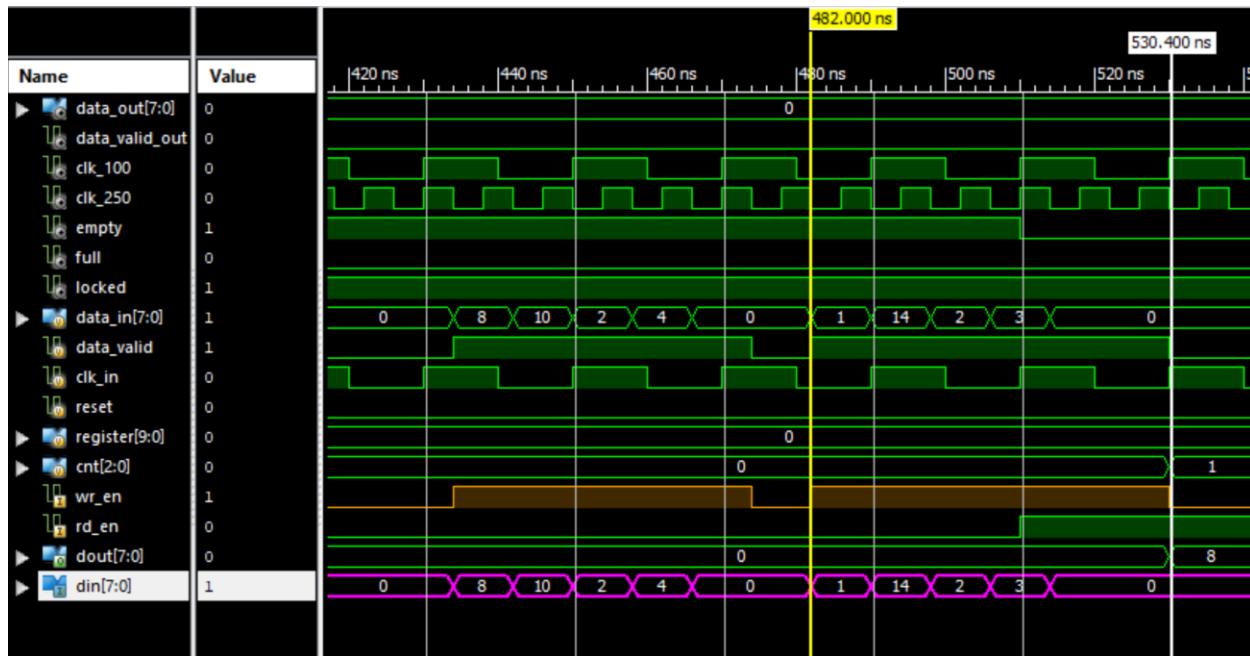
سری داده‌ی ورودی 8 و 10 و 4



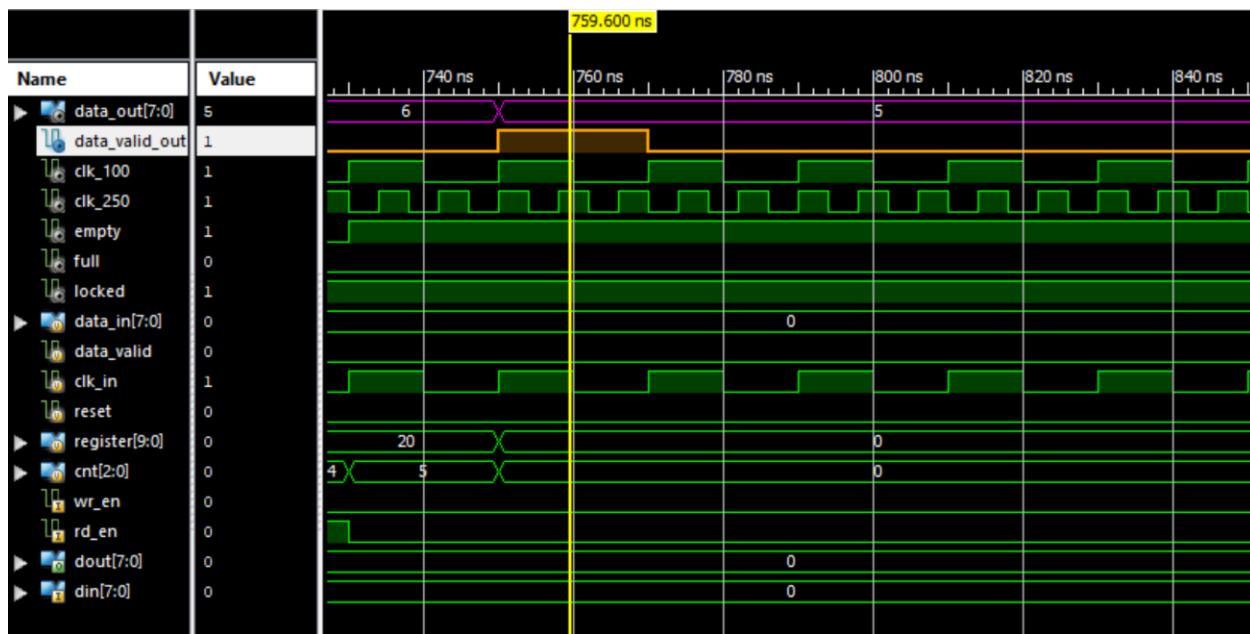
خروجی این سری داده ها



سری داده‌ی ۱، ۲ و ۳



خروجی این سری داده



سوال اختیاری) در این سوال کافی است به مدت یک ثانیه تعداد تغییرات سطح پالس ورودی را اندازه گرفته و بر چهار تقسیم کنیم (زیرا هر چهار تغییر سطح معادل دو پالس و هر دو پالس نماینده یک دور ماشین است) . روش شمارش تعداد تغییرات سطح مستقل از ورودی متقارن میباشد و برای هر نوع پالس ورودی به میزان یک ثانیه زمان برای اعلام تعداد دور موتور نیاز است

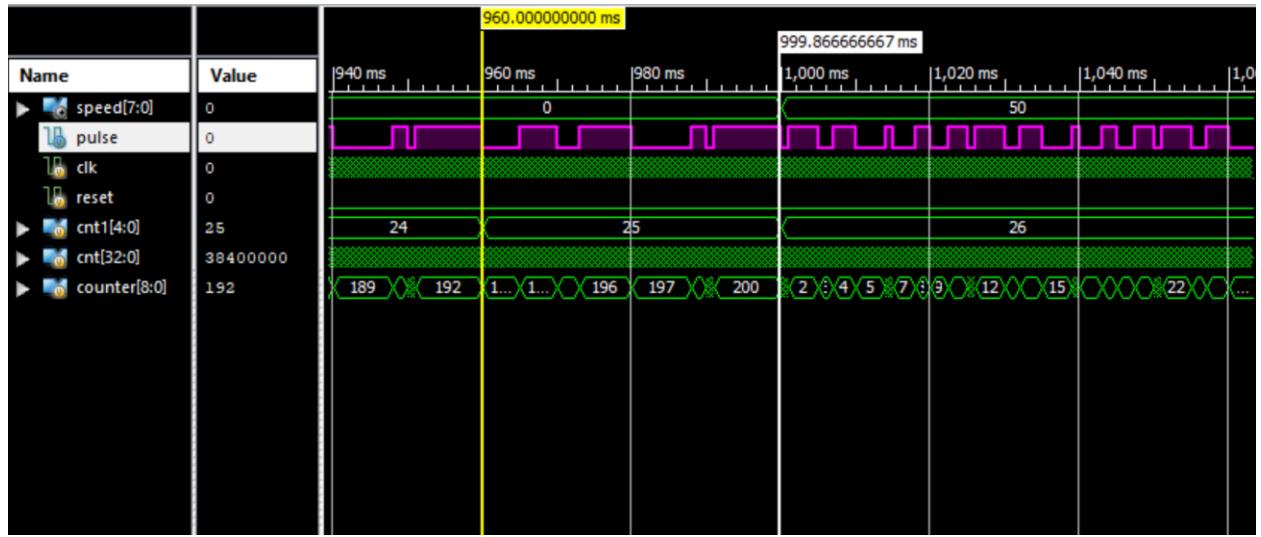
از انجایی که کلاک ورودی مژول **MHz40** است و دوره تنابوب آن **25ns** است کاتر **cnt** را برای شمارش یک ثانیه اتخاذ میکنیم بدین ترتیب که اگر 7×10^4 تا لبه‌ی بالارونده کلاک ورودی مژول را شمارش کنیم با توجه به دوره تنابوب کلاک میزان زمان **25ns * 40000000 = 25ns** معادل با گذر زمان یک ثانیه میباشد . در این مدت زمان تمامی تغییرات لبه‌ی پالس را (لبه‌ی بالارونده و لبه‌ی پایین رونده) را شمارش میکنیم تا تعداد تغییرات سطح پالس را چه از صفر به یک و چه از یک به صفر بشماریم (این تعداد تغییرات سطح را در متغیر **counter** قرار میدهیم) و هنگامی که یک ثانیه گذشت این تعداد تغییرات سطح را به ۴ تقسیم کرده و بعنوان خروجی قرار میدهیم و برای یک ثانیه‌ی بعدی تمامی کاترها را صفر میکنیم

برای تولید تست بنج در حالت اول تعداد دور **50** دور بر ثانیه باید **100** تا پالس باید تولید شود که این **100** پالس را به **25** بخش تقسیم میکنیم که هر بخش حاوی **4** پالس است که این بخش‌ها بصورت متناوب تکرار میشود و این **4** پالس‌ها با وجود نامتقارن بودن متناوب است . کل این **100** پالس در مدت یک ثانیه بطول می‌انجامد بنابراین هر کدام از این **25** بخش $1/25 = 0.04$ ثانیه یا **40** میلیون نانو ثانیه طول دارد . بنابراین بوسیله تاخیرهای مختلف در هر بخش مجموعا **4** پالسی تولید میکنیم که **40** میلیون نانو ثانیه مجموع صفر و یک بودن‌های هر بخش باید بطول بینجامد

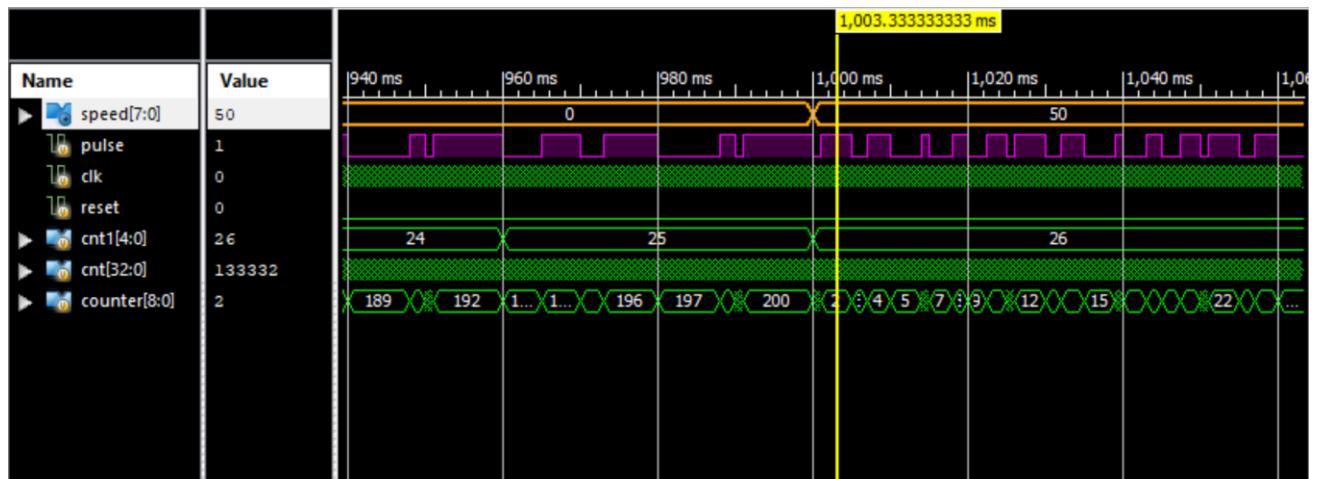
برای حالت دوم تعداد دور **100** دور بر ثانیه باید **200** تا پالس باید تولید شود که میخواهیم این **200** پالس را به **40** بخش حاوی **5** پالس تقسیم کنیم . $1/40 = 0.025$ ثانیه یا **25** میلیون نانو ثانیه طول دارد . بنابراین با تاخیرهای در هر بخش **5** پالس تولید میکنیم که مجموعا **25** نانو ثانیه طول میکشد که این بخش‌ها میتناوب اند اما پالس‌های درون آنها نا متقارن

نتایج شبیه سازی و waveform ها

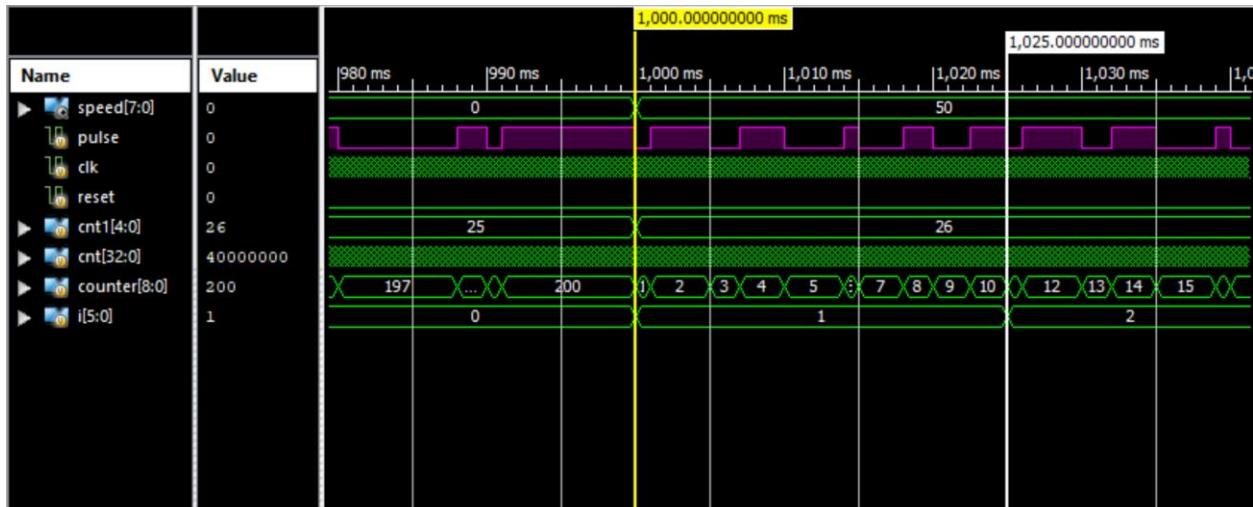
تولید پالس‌های نامتقارن



تعداد دور



تولید پالس های نامتقارن سری دوم



تعداد دور

