Computer Architecture Lab

Report #4

Implementing SRAM Controller

By: Edris Nasihatkon , Alireza Yazdanpanah

Lab partners

Ahmad Hassani 810194302

Nooshin Taghavi 810194289

در قسمت های قبلی، ماژول memory از جنس register بود که با توجه به ابعاد حافظه، این کار غیرمعقول به نظر می آید. در این فاز، از SRAM موجود در FPGA استفاده میکنیم. عملیات خواندن و نوشتن حافظه ی SRAM بر روی بورد DE2 با استفاده کلاک 50مگاهرتز، در یک کلاک انجام میگیرد. اما در پردازنده های واقعی که از کلاک سریعتر با فرکانس بیشتر استفاده میشود، نیاز به تعداد کلاک بیشتر برای خواندن و نوشتن از SRAM نیاز است. یه همین دلیل، در این فاز از پروژه، 5 کلاک را برای خواندن و نوشتن از SRAM اختصاص میدهیم تا چالشی که در پردازنده های دیگر وجود دارد را حل کنیم. چالش دیگری که با آن روبه رو هستیم، این است که داده های ما 32 بیتی هستند و در حالی که حافظه بورد DE2، 16 بیتی است.

برای حل این چالش ها، باید به این نکات دقت کنیم که اولاً وقتی در حال نوشتن یا خواندن از SRAM هستیم، باید استیج های قبلی را freeze کنیم تا دستورات از بین نروند. دوماً داده 32 بیتی را به 2 داده 16 بیتی تقسیم کرده و در 2 آدرس از خانه حافظه بنویسیم یا بخوانیم. 16 بیت کم ارزش در خانه اول و 16 بیت پرارزش در خانه بعدی آن.

برای این کار، یک ماژول SRAM (کنترلری برای حافظه) درون استیج MEM اضافه میکنیم که با توجه به سیگنال هایی که دریافت میکند، تشخیص دهد که خواندن یا نوشتن درون SRAM نیاز است. با توجه به آنها، سیگنال هایی که SRAM به آن نیاز دارد که مقدار دهی کرده و به ماژول SRAM بدهد. سپس جوابی که از SRAM دریافت میکند را (مثلاً داده خوانده شده) به صورتی که میخواهیم (یک داده 32 بیتی) به ما تحویل دهد.

حافظه 5 سیگنال 1بیتی دارد. SRAM_UB_N ، SRAM_LB_N، SRAM_WE_N، SRAM_CE_N و SRAM_OE_N. جز سیگنال SRAM_WE_N، همگی 0 اند (در پروژه ما). سیگنال SRAM_WE_N هم نشان میدهد که باید عمل نوشتن انجام دهیم یا نه و به صورت active-low میباشد. (write_enable)

حافظه یک ورودی 18بیتی SRAM_ADDR به عنوان آدرس و یک سیم دوطرفه(inout) 16 بیتی SRAM_DQ به عنوان داده خوانده شده یا داده ای که باید ذخیره شود نیز میگیرد.

در این ماژول یک شمارنده 3 بیتی میگیریم تا از 0 تا 5 بشمارد تا بفهمیم در چندمین کلاک از خواندن یا نوشتن هستیم. اگر درحال خواندن باشیم، در کلاک اول، آدرس اول که حاوی 16بیت کم ارزش را میخوانیم و در کلاک بعدی، 16 بیت مربوط به آدرس بعدی آن را میخوانیم. این دو را بهم چسبانده و یک دیتا 32بیتی روی پورت خروجی (readData) قرار میدهیم. 3 کلاک دیگر صبر میکنیم و سپس سیگنال pause که به عنوان freeze استفاده میشود را 0 میکنیم.

اگر در حال نوشتن باشیم، در کلاک اول، 16 بیت کم ارزش را روی پورت SRAM_DQ گذاشته. آدرس خانه ای که میخواهیم ذخیره شود را روی SRAM_ADDR قرار میدهیم. سیگنال SRAM_WE_N را 0 میکنیم. پس 16 بیت دیتا در آن خانه نوشته میشود. در کلاک بعدی، 16 بیت پرارزش را در خانه بعدی مانند بالا قرار میدهیم. 3 کلاک صبر کرده و سیگنال pause را برمیداریم (0 میکنیم)

آدرس ورودی از برنامه هم اینگونه تبدیل به آدرس SRAM میشود: اول 1024 واحد از آن کم میشود. چون 4 تا 4 تا زیاد میشد، 2 رقم آخر آن را برمیداریم. سپس در انتهای آن، اگر آدرس 16بیت کم ارزش را بخواهیم (خانه اول)، 0 و اگر آدرس 16بیت پرارزش دیتا را بخواهیم، 1 اضافه میکنیم. 18 بیت اول آن را میگیریم و به SRAM میدهیم. اینگونه آدرس دریافتی را به آدرس قابل استفاده برای SRAM تبدیل میکنیم.

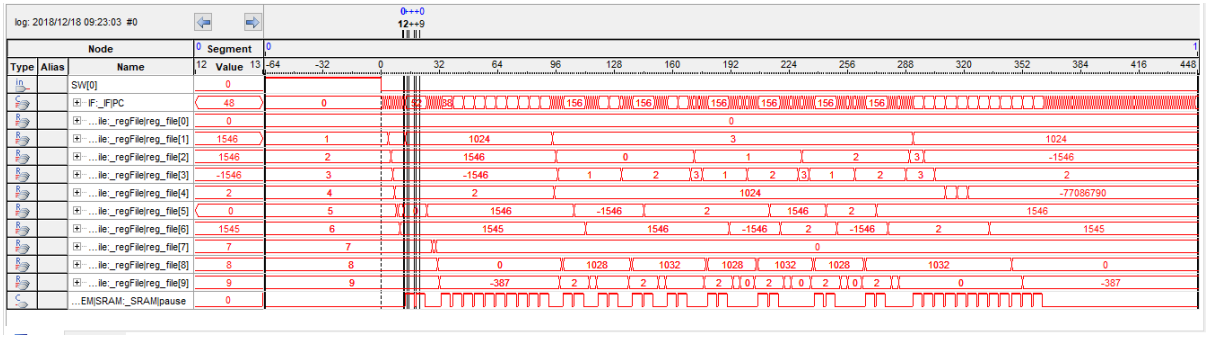
چون خواندن و نوشتن در 5 کلاک صورت میگیرد، وقتی کلاک کمتر از 5 باشد، باید سیگنال freeze به استیج های قبلی داده شود و در کلاک 5ام، آن را برداریم.

on Report - MIPS

Flow Summary

Flow Status	Successful - Tue Dec 18 09:20:53 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	MIPS
Top-level Entity Name	MIPS
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	8,887 / 33,216 (27 %)
Total combinational functions	4,482 / 33,216 (13 %)
Dedicated logic registers	7,294 / 33,216 (22 %)
Total registers	7294
Total pins	418 / 475 (88 %)
Total virtual pins	0
Total memory bits	181,760 / 483,840 (38 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

شکل 1 – نتایج کامپایل و سنتز



شکل 2 – Signal Tap

CPI (Clock per Ins)	Total Logic Element	Total Combinational Functions	Dedicated Logic Registers	Runtime
5.85	8887 27%	4482 13%	7294 22%	363 Clock

همانطور که انتظار میرود، زمان اجرای برنامه حدوداً دوبرابر شده(از 164 به 363) زیرا قبل آن دستورات store و load یک کلاک طول میکشید ولی الان 5 کلاک طول میکشد که باعث میشود برنامه freeze های بیشتری را ببیند.

همچنین با حذف Memory و جایگزینی آن با SRAM خود FPGA، تعداد رجیسترهای مورد نیاز برای سنتز کمتر شده. زیرا قبل آن از رجیستر به عنوان حافظه استفاده میکردیم.

طبق این نتایج trade-off بین هزینه سخت افزاری و زمانی داریم. که با کاهش سخت افزار مورد نیاز، زمان اجرا را بالا بردیم که باعث پایین آمدن کارایی پردازنده میشود. (trade-off همیشگی بین مساحت و سرعت در پردازنده)