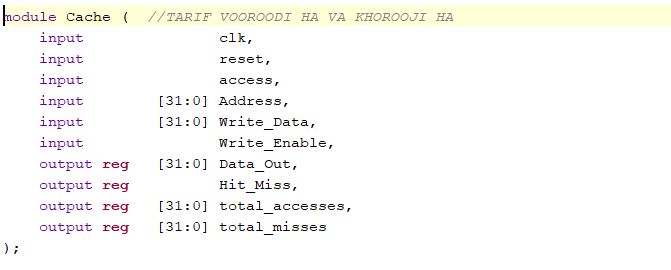
به نام خدا

بهراد نیک زاد- طاها عابدینی

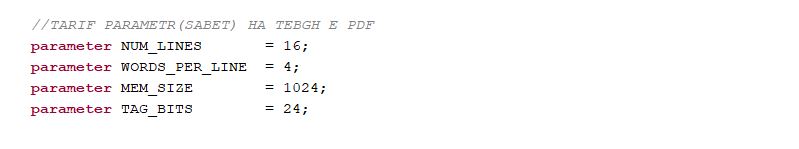
موضوع: CACHE

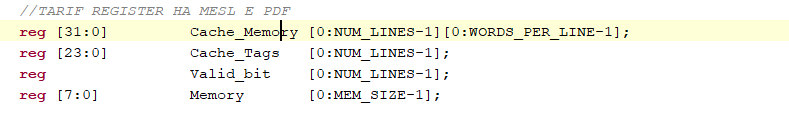
در اینجا می خواهیم به بررسی کد های مربوط به پروژه بپردازیم و نحوه عملکرد و ساختار آن ها را توضیح دهیم.

حالا شروع به توضیح کد می کنیم.

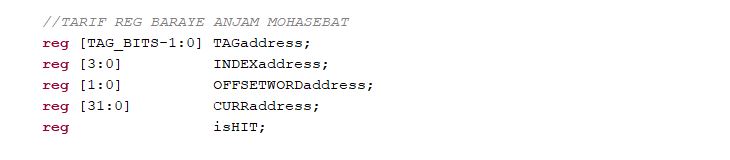


این بخش از کد مربوط به تعریف ورودی ها و خروجی ها می باشد. برای مثال clk سیگنال کلاک برای هماهنگی عملیات ها می باشد یا مثلا reset سیگنالی برای تنظیم ورودی ها به حالت اولیه می باشد. همچنین ورودی هایی تعریف کردیم مانند Address که منظور یک آدرس 32 بیتی برای دسترسی به حافظه می باشد. و در آخر دو تا شمارنده برای تعداد دسترسی ها و تعدادی که miss شده اند تعریف کرده ایم.

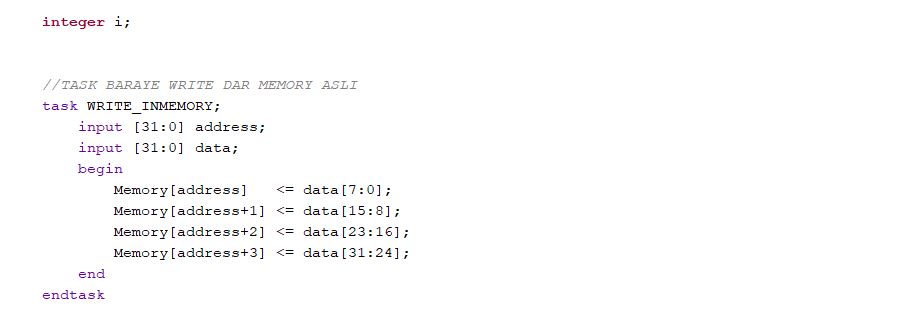
این بخش از کد مربوط به تعریف پارامتر ها طبق پی دی اف می باشد. در این تکه کد ما تعداد خط ها و کلمه های هر خط را همراه با مقدار فضای مورد نیاز همراه با تعداد بیت های تگ برای شناسایی تعریف کرده ایم.



در اینجا به تعریف رجیستر ها طبق پی دی اف پرداخته ایم. کش مموری ما یک آرایه دو بعدی می باشد که ابعاد آن 16\*4 می باشد. یعنی 16 خط و در هر خط 4 کلمه داریم. همچنین هر خانه از حافظه ما 32 بیتی فرض شده است. در ادامه نیز برای تگ های هر خط آرایه تعریف کرده ایم و همچنین بیت اعتبار هم تعریف کردیم. در آخر نیز آرایه ای برای حافظه اصلی تعریف کردیم که هر خانه آن طبق دستورالعمل 8 بیت می باشد.

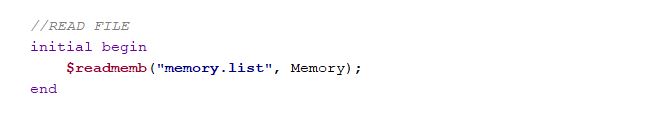


این تکه کد برای تعریف رجیستر ها برای انجام محاسبات است. TAGaddress برای ذخیره تگ ورودی می باشد و INDEX برای شناسایی خط می باشد و OFFSET برای تشخیص کلمه در خط می باشد. در آخر نیز آدرس جدید را برای بار بعدی می سازیم. یک فلگ نیز تعریف کردیم برای hit که ببینیم hit بوده است یا خیر.

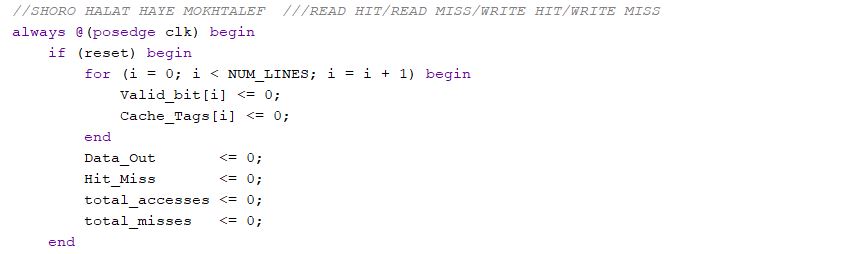


در این تکه کد ابتدا یک متغیر تعریف کردیم که در حلقه ها استفاده کنیم.

تسک ما برای نوشتن یک کلمه 32 بیتی در حافظه اصلی می باشد. دو ورودی دارد که آدرس شروع در حافظه اصلی و دیتا آن می باشد. هدف ما این است که 32 بیت را به 4 تا 8 بیت بشکنیم و آن را به صورت متوالی ذخیره کنیم زیرا حافظه اصلی ما بر حسب بایت است. و همچنین همانطور که بالاتر دیدید memory ما 8 بیتی می باشد.

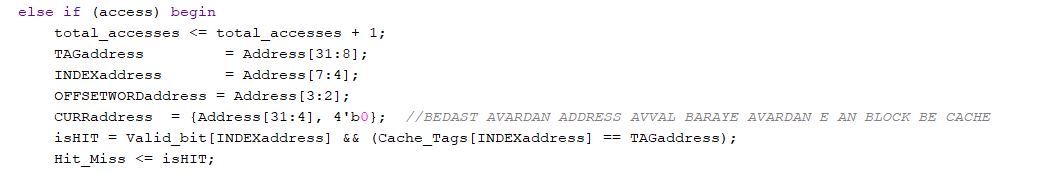


این بخش از کد همانطور که مشخص است برای خواندن memory.list و تبدیل آن در memory می باشد که باید 1024 تا مقدار 8 بیتی در آن باشد.



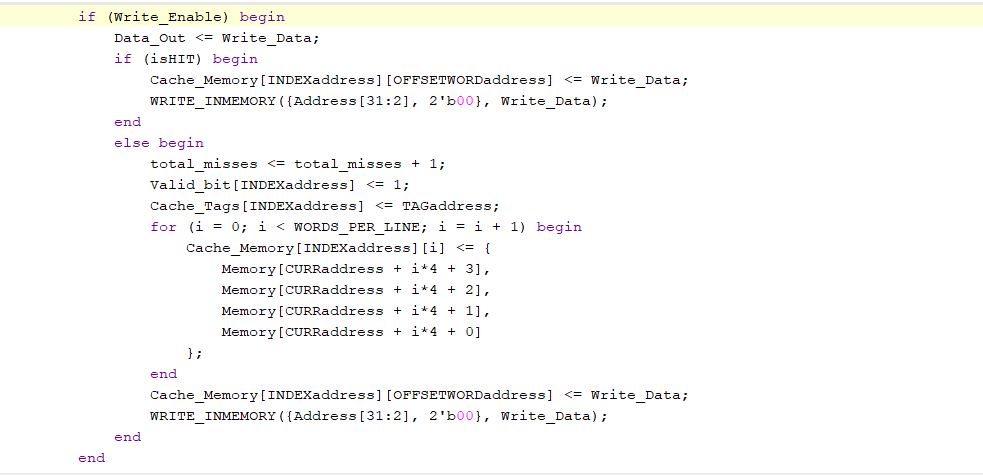
خط اول مربوط به تعریف بلاک حساس به لبه بالارونده می باشد که تمام عملیات های ما درآن انجام می شود.

سپس وارد کد اصلی می شویم که ابتدا به reset می پردازیم. در صورت فعال بودن سیگنال reset حافظه ما تنظیم مجدد می شود که شامل 0 شدن بیت های اعتبار و تنظیم شدن تگ به خط 0 و همچنین شمارنده های ما و دیتا ما نیز 0 می شوند.



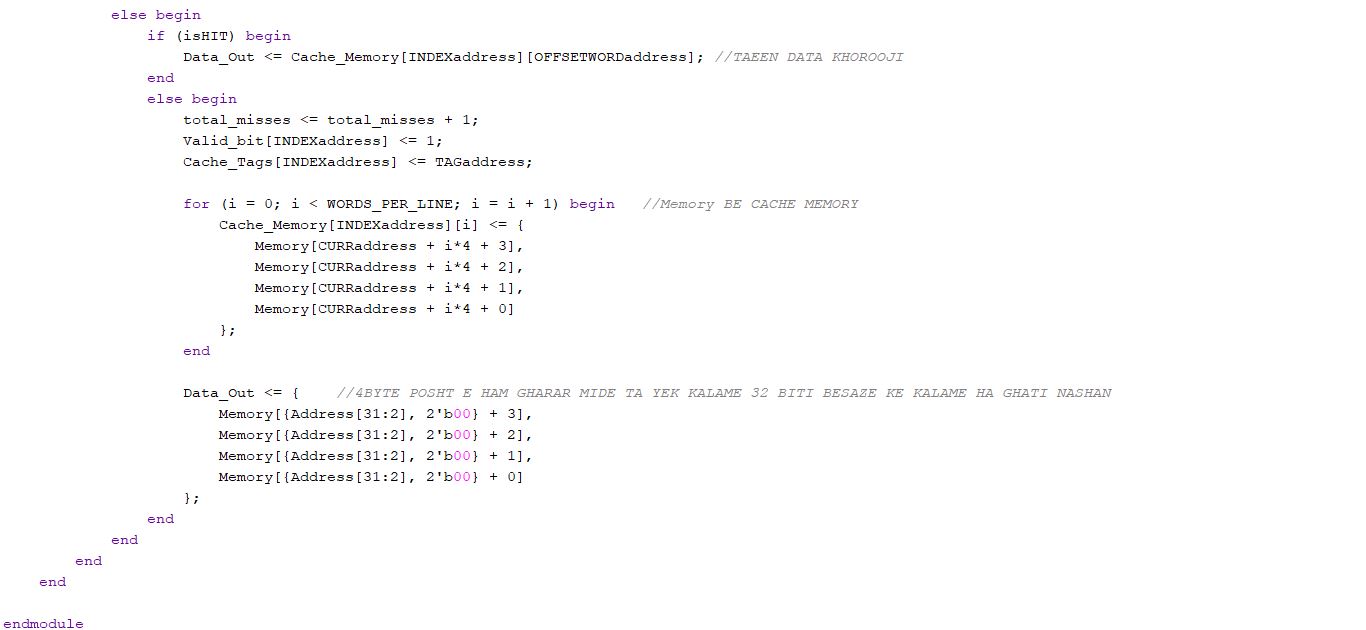
در ادامه به بررسی عملیات های خواندن و نوشتن در صورت اجازه دسترسی می پردازیم. ابتدا تعداد دسترسی ها را افزایش داده و سپس تگ های آدرس را استخراج می کنیم و خط و کلمه مورد نظر را مشخص می کنیم. درآخر نیز آدرس را آماده برای بعدی می کنیم.

در دو خط آخر نیز به بررسی hit بودن یا نبودن می پردازیم. اگر تگ ذخیره شده در خط با تگ آدرس برابر باشد هیت 1 می شود و سپس مقدار hit\_miss هم برابر با فلگ ما می شود.



این بخش از کد باری مدیریت عملیات write می باشد. دیتا خروجی را برابر با دیتا نوشته شده می خواهیم قرار دهیم. اگر هیت بود داده مورد نظر را در حافظه دو بعدی که پیش تر ساختیم قرار می دهیم و به کمک تسک بالا در حافظه اصلی نیز آن را می نویسیم.

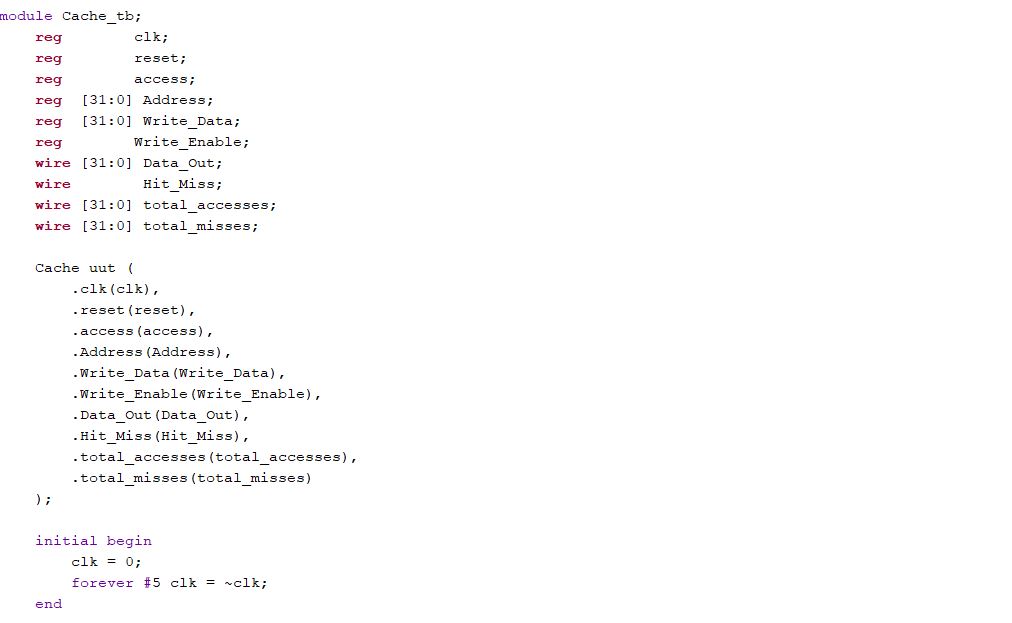
اگر اینطور نبود تعداد میس ها افزایش پیدا می کند و بیت اعتبار 1 می شود و تگ جدید ذخیره می شود. سپس خط اصلی ما در حافظه memory ذخیره می شود که شامل 4 کلمه 4 بایتی یا همان 32 بیتی می باشد. سپس داده اصلی در حافظه ذخیره می شود و در حافظه اصلی نیز آن داده نوشته می شود.



در آخر نیز به بررسی حالت read می پردازیم. ابتدا فعال بودن نوشتن را 0 می کنیم. حالا اگر هیت بود دیتای خروجی را برابر با دیتای خوانده شده از cache memory قرار می دهیم. و اگر میس بود دوباره تعداد میس ها را افزایش می دهیم و خط ما معتبر میشه و همچنین تگ جدید را ذخیره می کنیم.

در آخر نیز خط کامل ما مشابه قبل که شامل 4 کلمه است از حافظه اصلی بارگزاری می شود و دیتای خروجی ما مستقیم از حافظه اصلی خوانده می شود.

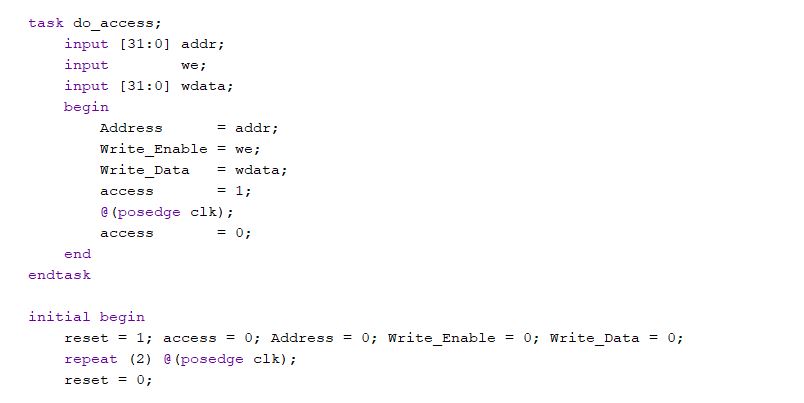
حالا به test bench می رسیم و آن را بررسی و تحلیل می کنیم.



در ابتدا ماژول Cache-tb را را تعریف می کنیم و سیگنال های گفته شده در پی دی اف را تعریف می کنیم.

سپس از آن instance می گیریم. باعث می شود سیگنال های ورودی و خروجی تست بنچ به پورت های ماژول Cache متصل شوند. این اتصال در واقع باعث می شود که تست بنچ سیگنال های ورودی را به ماژول دهد و خروجی را بررسی کند.

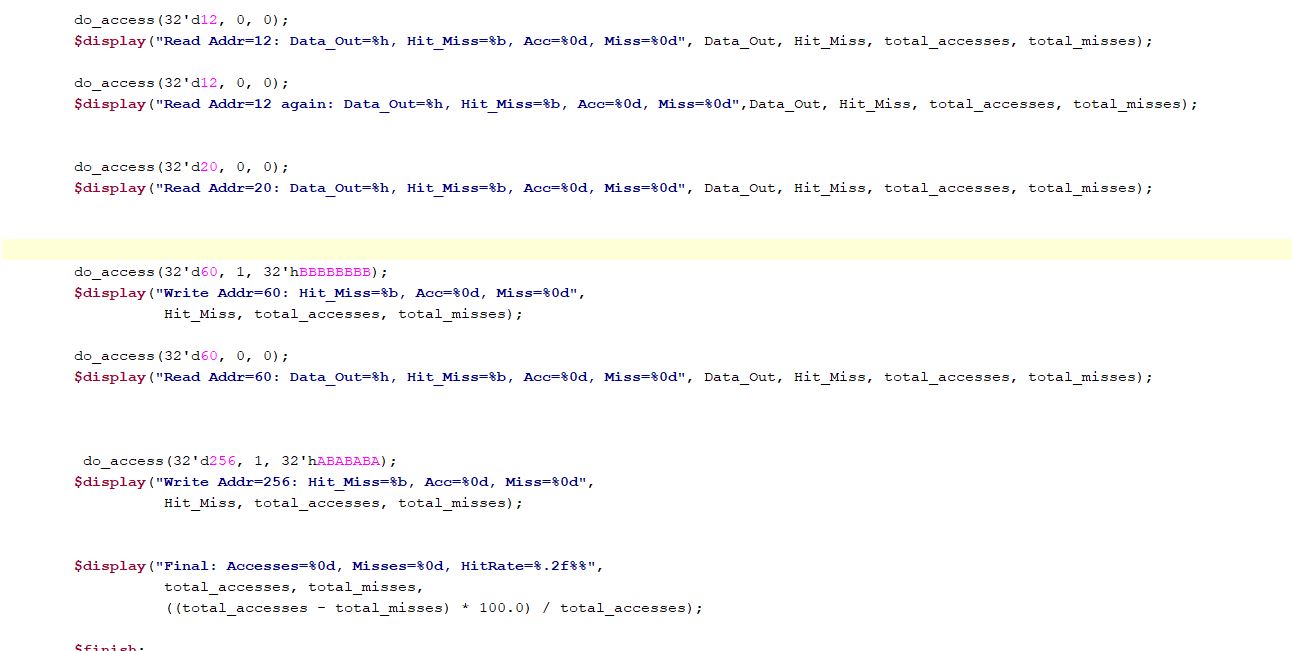
در آخر سیگنال کلاک را تولید می کنیم. کلاک در ابتدا صفر است و هر 5 نانوثانیه تغییر می کند.



حالا تسک مورد نظر را پیاده سازی می کنیم. هدف از این تسک انجام عملیات Read یا write می باشد.ورودی های ما آدرس و خواندن یا نوشتن و دیتا برای نوشتن می باشند.

حالا مقدایر را با مقادیر ورودی برابر می کنیم. سپس دسترسی را 1 می کنیم و بعداز یک سیکل (بعد از لبه بالا رونده) آن را 0 می کنیم.

در آخر نیز ماژول را ریست می کنیم و متغیر ها را تنظیم می کنیم. حالا دو سیکل کلاک صبر می کنیم و ریست را 0 می کنیم تا ماژول وارد حالت عادی بشود.

حالا به تست ها می رسیم. در اینجا ابتدا می خواهیم آدرس 12 را بخوانیم. باید عدد آدرس را 32 بیتی در نظر بگیریم.

عدد 12 در باینری 00001100 است. (8 بیتی) ایندکس ما 0000 است و آفست ما 11 است.

این دسترسی به خط 0 و کلمه 3 اشاره می کند. این تست miss می شود.

در تست بعدی همین آدرس را می خوانیم. چون آدرس تگ ها مطابقت دارن و خط 0 در دسترسی قبلی بارگزاری شده هیت می شود.

تست بعدی آدرس 20 است که مانند تست اول miss می شود.

تست بعدی نوشتن آدرس 60 می باشد. آدرس را می نویسیم و miss می شود.

دوباره همان آدرس را می خواهیم بخوانیم. چون در دسترسی قبلی بارگزاری شده hit می شود.

در آخر نیز آدرس 256 را write می کنیم و miss می دهد.