گزارش تمرین سوم طراحی سیستم‌های دیجیتال

طراحی پردازندهٔ چندچرخه‌ای

ریحانه خیاط‌زاده ماهانی ۴۰۲۱۰۵۹۶۵

ماژول CPU :

ماژول CPU ماژول top level و اصلی است که تمام submodule ها و بخش‌های مختلف پردازنده یعنی حافظهٔ اصلی، رجیستر فایل، واحد کنترلی و ALU را در آن instantiate و ایجاد کردیم و اتصالات لازم را هم برقرار کرده‌ایم.

ماژول ALU :

در این ماژول دو ورودی signed ۱۶ بیتی گرفته شده و خروجی signed ۱۶ بیتی داده میشود. همچنین ۱۶ بیت بالا برای باقی‌مانده و قسمت بالای عملیات ضرب نیز تولید میشود ولی در دستورات استفاده‌ای از آن نداریم. Submodule های مختلف برای عملیات‌های جمع، ضرب و تقسیم در این ماژول instantiate میشوند و اتصالات هم برقرار میشود. چند state هم برای روند اجرای محاسبات وجود دارد که برای ضرب و تقسیم که چند کلاک نیاز دارند استفاده میشوند ولی عمل جمع در یک کلاک انجام میشود که حالا به بررسی روند پیاده‌سازی عملیات‌های مختلف میپردازیم.

۱. عملیات جمع و تفریق:

برای عملیات تفریق کافیست ورودی دوم را نات کرده و رقم نقلی ورودی را ۱ دهیم تا two’s complement آن وارد شود.

* **ماژول Carry Select Adder :**

در این ماژول ورودی‌های ۱۶ بیتی به ۴ تا بلوک ۴ بیتی تقسیم شده و بجز بلوک اول که با carry اولیه درست شروع به جمع می‌کند سایر ۳ بلوک دیگر هرکدام دو جمع ۴ بیتی انجام میدهند یکبار با carry صفر و یکبار با یک، و درنهایت پس از به‌دست آمدن مقدار درست carry از بلوک اول تا آخر، مقدار خروجی نهایی carry و sum هر بلوک مشخص میشود.

* **ماژول Ripple Carry Adder :**

هر کدام از جمع‌های بلوک‌های ماژول CSA، با استفاده از instantiate کردن این ماژول انجام میشوند که یک جمع‌کنندهٔ کلاسیک ٤ بیتی شامل ٤ تا full adder است که رقم نقلی را به صورت موج از بیت اول تا آخر انتقال میدهد.

* ماژول **Full Adder** :

جمع‌کنندهٔ سادهٔ دو بیت و یک بیت نقلی برای ماژول Ripple Carry Adder است.

۲. عملیات ضرب:

این عملیات در ۸ کلاک انجام میشود پس ALU با مشاهدهٔ این opcode سیگنال start ماژول ضرب‌کننده را فعال میکند، تا زمانیکه ضرب‌کننده خروجی done ۱ بدهد یعنی عملیات تمام شده و خروجی‌ها داخل result ریخته میشوند.

* ماژول **Karatsuba** :

این ماژول مطابق الگوریتم داده‌شده، نیاز به ۳ ضرب ۸ بیتی دارد که برای این کار ۳ ماژول ۸ shift & add بیتی ایجاد کرده‌ایم که به صورت موازی باهم کار میکنند و در ابتدای الگوریتم سیگنال start شان فعال میشود و صبر میکنیم تا done همه فعال شود تا خروجی نهایی را تولید کنیم. برای این کار نیاز به ٤ تا حالت داریم.

* ماژول **Shift and Add** :

هر کدام از ضرب‌های ۸ بیتی بخش های بالا و پایین دو عدد با استفاده از این ماژول در ۸ کلاک و ٤ حالت انجام میشوند. در هر کلاک، اگر بیت صفر عدد دوم ۱ باشد، عدد اول به حاصل‌ضرب اضافه میشود و سپس هر دو عدد شیفت داده میشوند.

۳. عملیات تقسیم:

عملیات تقسیم در ١٦ کلاک انجام شده و مشابه ضرب، ALU پس از مشاهدهٔ این opcode منتظر میماند تا عملیات تمام شود.

* ماژول **Restoring Division** :

مطابق الگوریتم توضیح داده شده در داک، در هر سایکل باقی‌ماندهٔ فعلی یکی به چپ شیفت داده شده و یک بیت از مقسوم وارد میشود، سپس اگر نتیجه مثبت بودم مقسوم‌علیه از باقی‌مانده کم میشود. این روند ١٦ بار تکرار شده تا خارج قسمت و باقی‌ماندهٔ نهایی در به ترتیب در ١٦ بیت پایین و بالای result قرار بگیرد.

ماژول Main Memory :

برای ایجاد حافظه از یک آرایه ۲ بعدی با ۱۰۲٤ خانهٔ ١٦ بیتی استفاده کردیم (در داک ۲ توان ١٦ خانه گفته شده بود که برای راحتی ۱۰۲٤ خانه گرفتیم) که در ابتدا مقدار همهٔ خانه‌هایش را صفر میکنیم و دو سیگنال کنترلی برای خواندن و نوشتن نیز در نظر میگیریم که برای load و store استفاده کنیم.

ماژول Register File :

مطابق داک ٤ رجیستر ١٦ بیتی ایجاد میکنیم که در ابتدا مقدارشان صفر است و در لبهٔ پایین‌روندهٔ هر کلاک ۲ رجیستر خوانده‌شده در خروجی رجیسترفایل قرار گرفته و در صورت فعال بودن سیگنال write enable در لبهٔ بالارونده هر کلاک در یک رجیستر نوشته میشود.

ماژول Control Unit :

این ماژول روند اجرای دستورات را با استفاده از حالت های اصلی Fetch/Decode/Execute/Memory/Writeback کنترل میکند و در هر مرحله، سیگنال های مربوط به enable خواندن و نوشتن، هدایت دیتای خوانده و نوشته‌شده و روند مراحل را تنظیم میکند. همچنین برای عملکرد صحیح دسترسی به حافظه و رجیسترفایل، حالت‌های میانی RF Access و Memory Access هم داریم که مقادیر درستی خوانده شوند، برای اجرای درست مرحلهٔ Execute نیز از آنجایی که محاسبات ALU ممکن است بیش از یک کلاک زمان ببرند حالت میانی ALU Wait را هم قرار داده‌ایم که حالت‌ها ساده‌تر شوند.

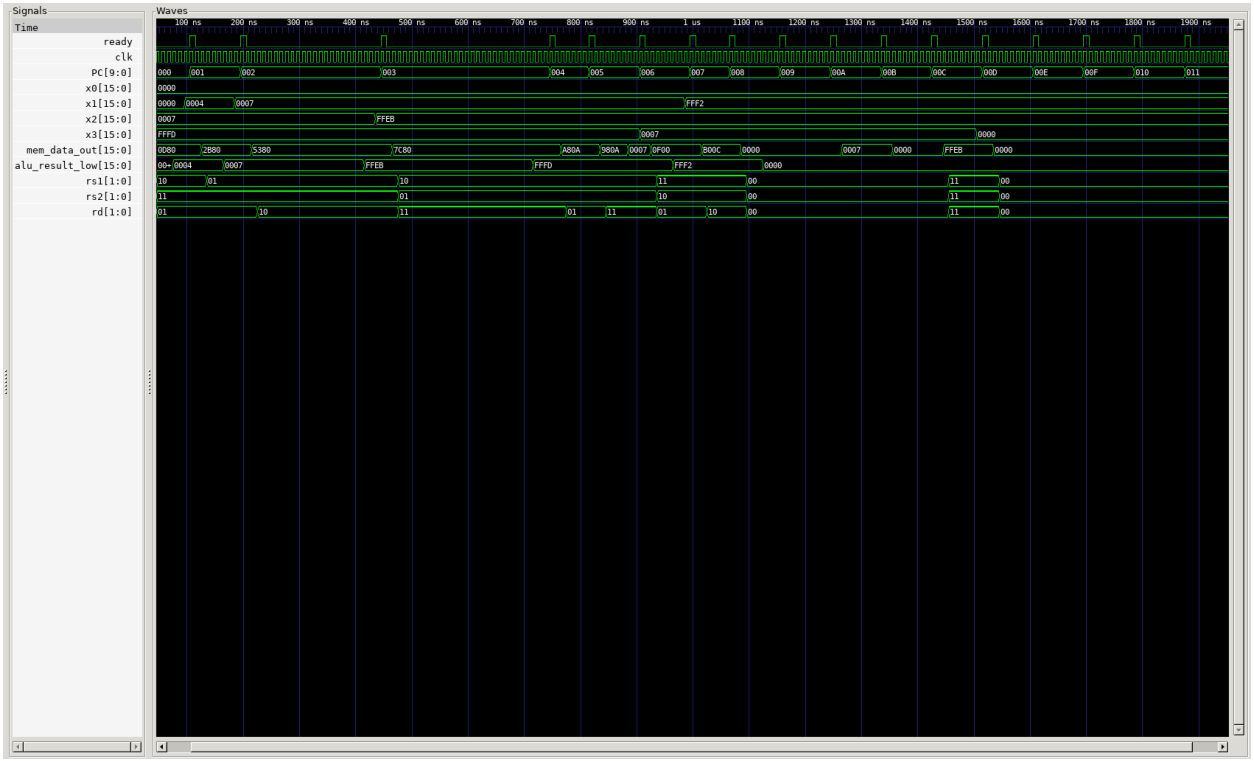
ماژول تست‌بنچ

برای تست این کد مجموعه‌ای از دستورات را در حافظه نوشته‌ایم و در رجیستر فایل نیز مقدار اولیه‌ای برای رجیسترها ایجاد کرده و سپس اجرای دستورات را تست کرده و خروجی را در ترمینال با display چاپ کرده و همچنین خروجی waveform را در فایل VCD با اسم cpu_waveform قرار داده‌ایم.

خروجی تست‌بنچ:

```
VCD info: dumpfile cpu_waveform.vcd opened for output.
x1=     4, x2=     7, x3=    -3, Mem[10]=   123, Mem[12]=     0
x1=     7, x2=     7, x3=    -3, Mem[10]=   123, Mem[12]=     0
x1=     7, x2=   -21, x3=    -3, Mem[10]=   123, Mem[12]=     0
x1=     7, x2=   -21, x3=    -3, Mem[10]=   123, Mem[12]=     0
x1=     7, x2=   -21, x3=    -3, Mem[10]=     7, Mem[12]=     0
x1=     7, x2=   -21, x3=     7, Mem[10]=     7, Mem[12]=     0
x1=   -14, x2=   -21, x3=     7, Mem[10]=     7, Mem[12]=     0
x1=   -14, x2=   -21, x3=     7, Mem[10]=     7, Mem[12]=   -21
TEST COMPLETE
```

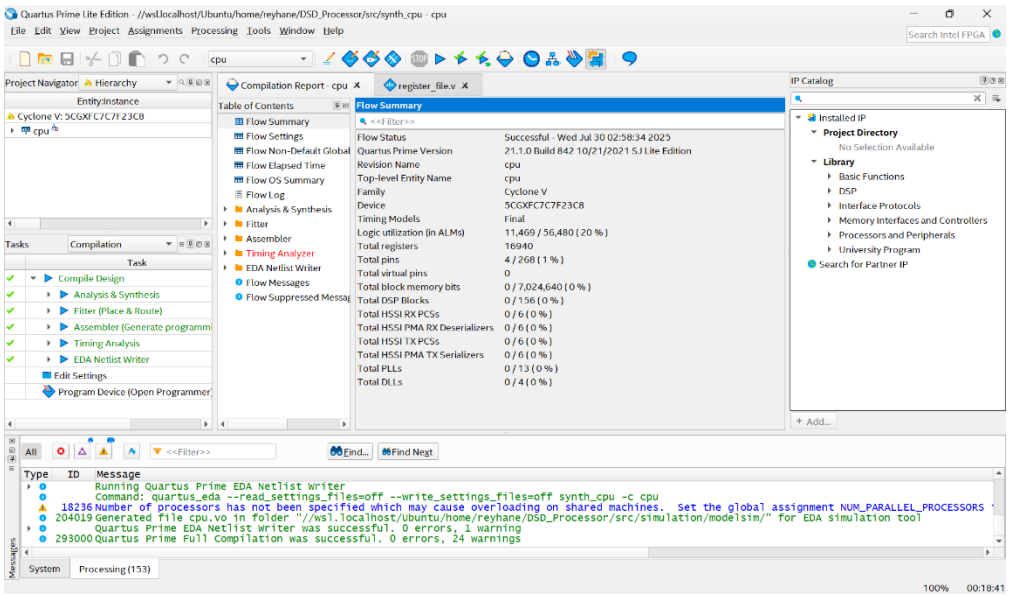
Waveform :



سنتز

برای سنتز نیز از ابزار Quartus استفاده کردیم و فایل های

کامپایل :



فایل های vo. و sft. و xrf. خروجی در پوشهٔ synth قرار گرفته اند. همچنین فایل پی‌دی‌اف شماتیک بخش های مختلف مدار هم که توسط کوارتوس تولید کردیم در این پوشه قرار دارد که اینجا هم ماژول اصلی را قرار داده‌ام.