گزارش تمرین اول و دوم

طراحی کامپیوتری سیستم‌های دیجیتال

دکتر مدرسی

اعضای گروه:

پریا خوش‌تاب 810198387

پرنیان فاضل 810198516

نکات طراحی:

- از آنجایی که N حداکثر 5 بیت می باشد و ازای بزرگ ترین N یعنی 31 مقدار خروجی fib(31) یک عدد 63 بیتی می شود و همچنین به یک بیت کمکی (flag)برای مشخص کردن وضعیت هر عنصر استک نیاز داریم، بنابراین تمام محاسبات را در فضای 64 بیتی انجام می دهیم.
- در صورتی که این بیت کمکی یعنی بیت 64ام 0 باشد، یعنی این عنصر عدد(پارامتر تابع) می باشد و اگر 1 باشد یعنی این عنصر مقدار خروجی تابع می باشد.
- به ازای توابع فیبوناچی که هنوز محاسبه نشده‌اند یک عدد پیش فرض 64 بیتی که بیت پر ارزش آن برابر 1 و بقیه بیت های آن 0 است در نظر میگیریم و بعدا مقدار درست آن را به جایش قرار می‌دهیم.
- در هر مرحله از این الگوریتم با توجه به بیت flag دو عدد بالای استک و همچنین کوچکتر یا بزرگتر بودن عدد هر استک فریم از مقدار 1، چند حالت ایجاد می شود که با توجه به هر حالت اقدامات لازم را انجام میدهیم. این قسمت را در استیت q16 در استیت ماشین هم می‌توان مشاهده کرد.
- ما استک را طوری پیاده سازی کردیم که علاوه بر مقدار بالای استک، یک سیگنال is_empty خروجی دهد که نشانگر این است که آیا استک خالی است یا خیر. همچنین همانطور که در درس اعلام شد ما مجاز به استفاده از سیگنال ورودی top بودیم و بنابراین سیگنال ما 3 سیگنال ورودی pop، push و top دارد که در لبه بالارونده‌ی کلاک عملیات مربوطه انجام می‌شود. توجه شود که عملکرد استک پیاده سازی شده در پروژه به غیر از این سیگنال خروجی is_empty و سیگنال ورودی top، هیچ تفاوت دیگری با استک معمولی ندارد.
- یک سیگنال خروجی به نام done در نظر میگیریم که زمانی که fib(N) کامل محاسبه شود، آن را 1 می کنیم که 1 بودن این سیگنال به معنای قرار داشتن مقدار صحیح fib(N) در رجیستر res می باشد.

اصلاحات طراحی:

- تمام محاسبات را به جای فضای 31 بیتی در فضای 64 بیتی انجام می‌دهیم تا سرریز رخ ندهد.
- همانطور که در نکات طراحی گفته شد هر 3 عملیات مربوط به استک با لبه‌ی بالارونده کلاک انجام میشود اما در فاز قبلی فرض شده بود که سیگنال top بدون نیاز به کلاک و در همان لحظه مقدار بالای استک را روی خروجی stack_out قرار می‌دهد.
- برای سادگی، به جای استفاده از یک comparator مشترک همراه با mux برای مقایسه عدد N با 1 و مقایسه reg1_out با 1 از 2 comparator جدا استفاده می‌کنیم.
- به جای divider از یک shifter یک بیت استفاده می‌کنیم.
- برای طراحی بهتر به جای استفاده از چند جمع‌کننده، تفریق‌کننده و ضرب‌کننده جدا از یک ماجول ALU استفاده می‌کنیم.
- در کنترلر در مواردی که لبه‌ی بالارونده‌ی کلاک برای انجام عملیات مربوطه نیاز بود اما در نظر گرفته نشده بود، استیت اضافه کردیم، برای مثال push و pop را در استیت های جداگانه انجام دادیم(q8 و q10) و لود کردن در رجیستر و pop/top را هم در استیت های جداگانه انجام دادیم(q3).

بخش TCL:

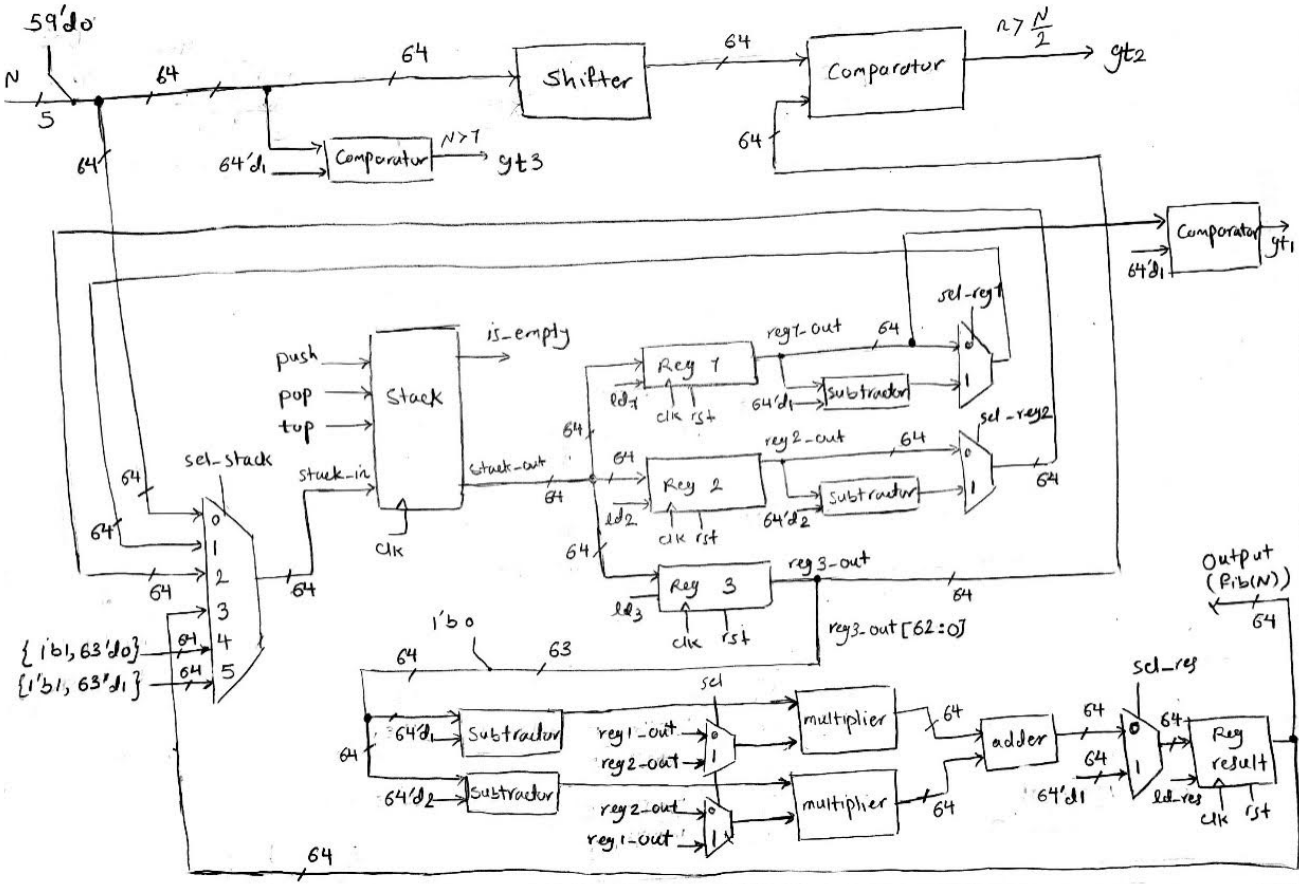
برای این بخش موارد زیر را انجام دادیم:

1. مقدار set TB در ابتدای script را برابر با نام فایل "Fibonacci_TB" قرار دادیم.
2. مقدار set run_time در ابتدای script را برابر "us 1000" قرار دادیم.
3. در قسمت Add verilog files تمام فایل‌های با پسوند v. را قرار دادیم، برای مثال:

```
vlog +acc -incr -source +define+SIM $hdl_path/Datapath.v
```

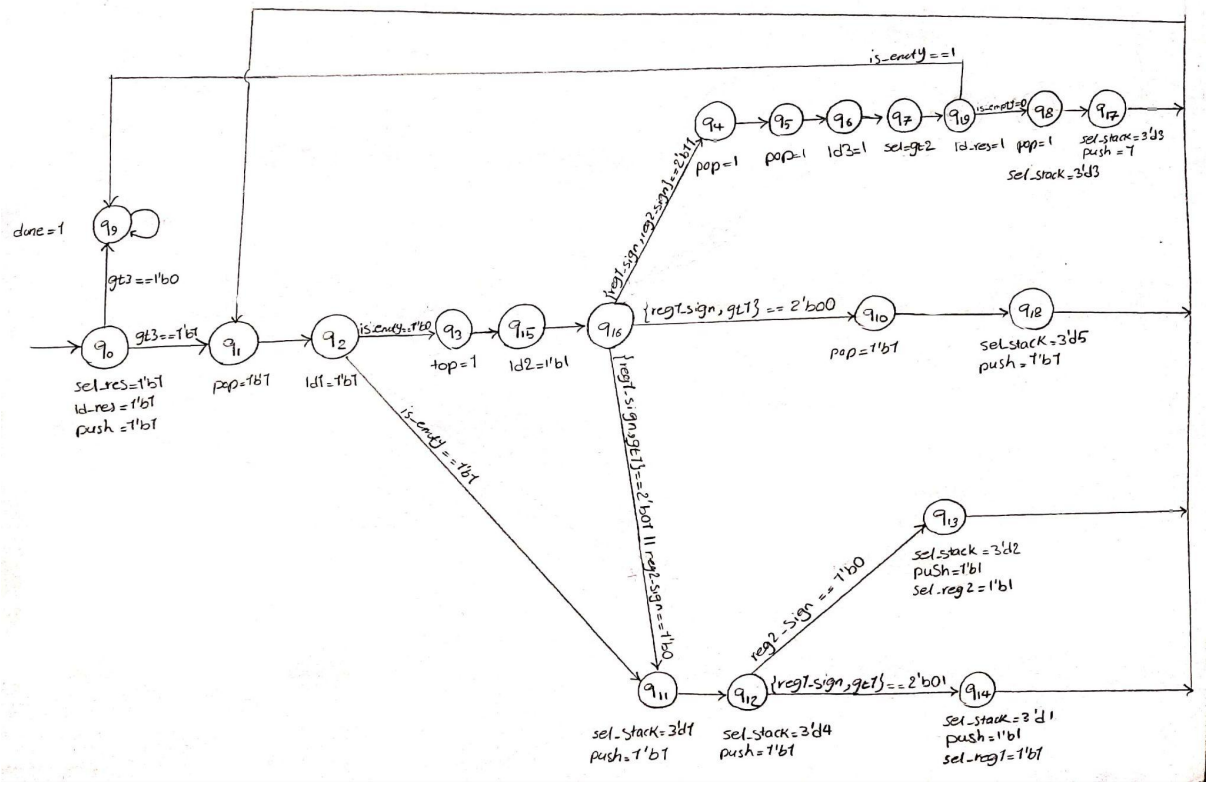
در نهایت با دستور do sim_top.tcl میتوانیم script را اجرا کنیم.

Datapath:



همانطور که دیده می‌شود، در واقع قسمت اصلی مسیر داده 3 استک Reg2 ، Reg1 و Reg3 است که درصورت نیاز برای محاسبه از این رجیستر ها استفاده میشود و در هر مرحله با توجه به بیت کمکی flag، سیگنال های ورودی دیتاپت از طریق کنترلر مقدار می‌گیرند مثل selector ها و load رجیستر ها.

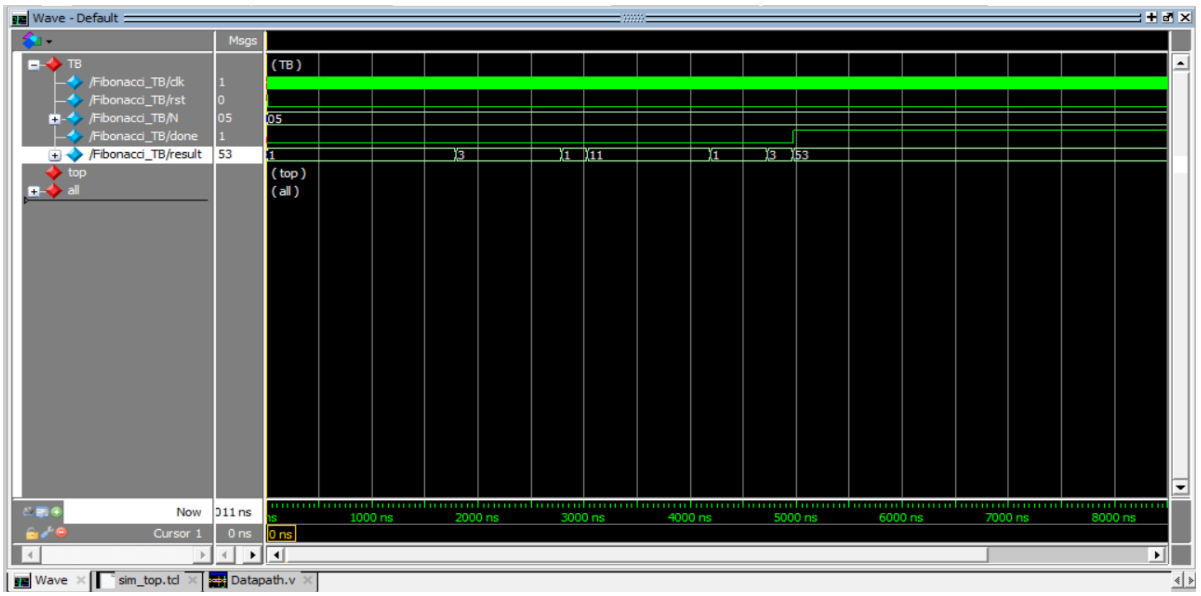
Controller:



قسمت اصلی استیت ماشین بالا در استیت q16 می‌باشد که با توجه به بیت کمکی flag تصمیم میگیریم کدام سیگنال ها را مقداردهی کرده و به عنوان سیگنال‌های ورودی مسیرداده، خروجی دهیم. برای مثال در حالتی که بیت کمکی هر دو عدد بالای استک برابر 1 باشند یعنی میتوانیم عملیات محاسبه فیبوناچی را برای این استک‌فریم انجام دهیم پس به استیت q4 میرویم و سیگنال های مربوط به این حالت در استیت‌های q4 تا q7 مقداردهی می‌شوند.

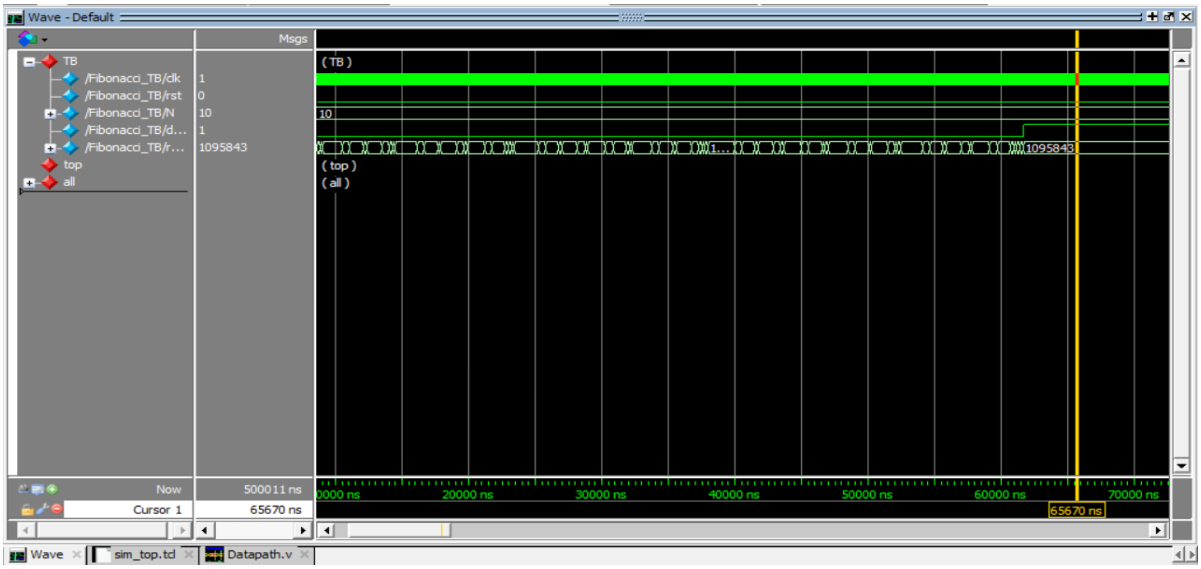
چند تست:

➢ **N = 5:**



در اینجا همانطور که دیده می‌شود، هنگامی که مقدار done برابر 1 می‌شود، مقدار خروجی نهایی result برابر 53 شده و درست است.

➢ **N = 10:**



در این تست همانطور که دیده می‌شود زمانی که مقدار done برابر 1 می‌شود، مقدار خروجی نهایی result برابر 1095843 می‌شود و درست است.