طرح کلی

پروژه از سه بخش اصلی تشکیل شده است که بخش اول مربوط به پیاده‌سازی یک Full Adder یک بیتی و سپس تعمیم آن به چهار بیت است. بخش دوم مربوط به پیاده‌سازی باس با نگاه به معماری مانو بوده که در آن AR, PC, DR, AC, IR, TR شش رجیستر ۱۶ بیتی سیستم را تشکیل داده و در کنار آن OUTR و INPR که OUTR رجیستری ۸ بیتی و INTPR لاین‌های ورودی است برای ورودی و خروجی در نظر گرفته شده‌اند. لازم به ذکر است که می‌توان INPR را بصورت داده مستقیم و یا یک رجیستر همانند OUTR در نظر گرفت که در اینجا با توجه به اینکه در تصویر بخش ۷.۴ سیگنالی کنترلی برای آن وجود ندارد و همچنین در سند پروژه بطور مستقیم به رجیستری بودن آن اشاره نشده است، در حالی که برای OUTR این موضوع مورد توجه قرار گرفته است **INPR را بصورت یک سیگنال ۸ بیتی و نه یک رجیستر** در نظر می‌گیریم.

در این سیستم علاوه بر این رجیسترها و سیستم IO یک واحد حافظه نیز در اختیار داریم که همانطور که در فایل پروژه گفته شده است برای جلوگیری از زمان زیاد شبیه‌سازی تعداد کلمات قابل نگهداری توسط آن به ۳۲ کاهش یافته است در حالیکه در سیستم اصلی تعداد برابر ۴۰۹۶ است. برای این ماژول نیز در بخش سوم پروژه پیاده‌سازی صورت گرفته است. همچنین یک عدد فلیپ‌فلاپ به نام E بعنوان Cout از واحد محاسبات بیرون آمده است که می‌تواند در نقش یک پرچم (flag) نیز مورد استفاده قرار بگیرد. پیاده‌سازی این ALU شانزده بیتی نیز در بخش سوم عنوان شده است. با توجه به اینکه در بخش دوم هنوز با نحوه پیاده‌سازی ALU, E flip-flop و حافظه مموری آشنایی نداریم، ترجیح میدهم ترتیب انجام قسمت‌ها را بصورت زیر تغییر دهم:

۱. در ابتدا بخش اول پروژه تحت عنوان پیاده‌سازی 4-bit adder انجام شده است.

۲. در قسمت دوم به ترتیب به سراغ پیاده‌سازی E flip-flop، و بعد از آن RAM و واحد محاسبات رفته‌ام.

۳. در نهایت به بخش دوم بازگشته و با پیاده‌سازی نمونه‌های باس و پس از آن رجیستر در نهایت سیستم کلی را شبیه‌سازی کرده‌ام.

تمامی قسمت‌های پروژه اعم از فایل‌های نمونه و قسمت‌هایی که برای آشنایی عنوان شده بود بطور کامل شبیه‌سازی شده‌اند و فایل نهایی ISE آن‌ها در پوشه تحویلی موجود است و نتایج هر قسمت قابل بررسی است.

همانطور که در تعریف پروژه نیز عنوان شده است، در پیاده‌سازی سیستم بطور کامل دیدی ماژولار داریم بطوری که هر قسمت پایه‌ای ابتدا طراحی شده است و سپس از آن برای بسط سیستم بزرگتر استفاده شده است. در این زمینه می‌توان به پیاده‌سازی ALU شانزده بیتی با شانزده ALU تک بیتی و یا مدل کردن رجیسترهای مدل بخش دوم با استفاده از یک مدل پایه‌ای اشاره کرد. این دید امکان ساخت سریع‌تر و کم‌خطاتر سیستم‌های بزرگ را مهیا می‌سازد.

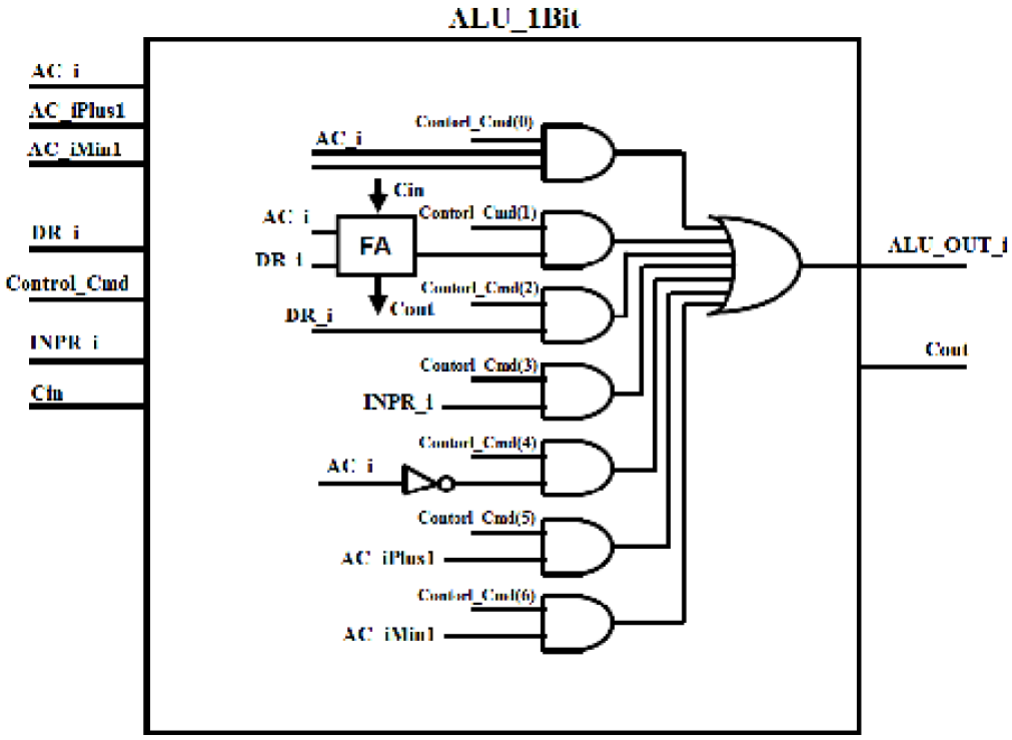
در این قسمت نکته دیگری که در پروژه بصورت فرض گرفته شده است عنوان می‌شود و آن **استفاده از شیفت معمول بجای شیفت چرخشی است.** در واقع در هنگام طراحی ALU با توجه به اینکه می‌خواستیم شانزده عدد ALU تک بیتی را کنار هم قرار دهیم امکان پیاده‌سازی هر دو روش را داشتیم، اما از شیفت معمول استفاده کردیم که در آن در شیفت به راست مقدار MSB به صفر و در شیفت چپ نیز مقدار LSB به صفر تغییر می‌کند. در صورت استفاده از شیفت چرخشی که با تغییر دو خط از کد قابل پیاده‌سازی است، در شیفت راست مقدار MSB برابر با مقدار قبلی بیت LSB شده و در شیفت به چپ نیز عکس این اتفاق رخ می‌داد.

بخش اول

در بخش اول با استفاده از یک FullAdder که نمونه آن در فایل سند آمده بود و استفاده از چهار نمونه از آن یک 4-bit Adder ساخته شده است. کد مربوط به VHDL آن در فایل زیپ اولیه وجود دارد. برای اینکار تنها یک component جدید را معرفی کرده و با چهار با port map همانند آنچه که برای دو بیت انجام گرفته بود، خروجی Cout هر قسمت را به Cin بعدی متصل کرده و این کار را تا رسیدن به Cout اصلی ادامه داده‌ایم.

بخش دوم

در این بخش همانگونه که در بالا توضیح داده شده بود، ابتدا کدهای مربوط به فلیپ‌فلاپ E و حافظه رم را مطابق نمونه پیاده‌سازی و ایرادات آن را رفع کرده‌ایم. سپس برای پیاده‌سازی واحد ALU دقیقا از شکل زیر کمک گرفته‌ایم.

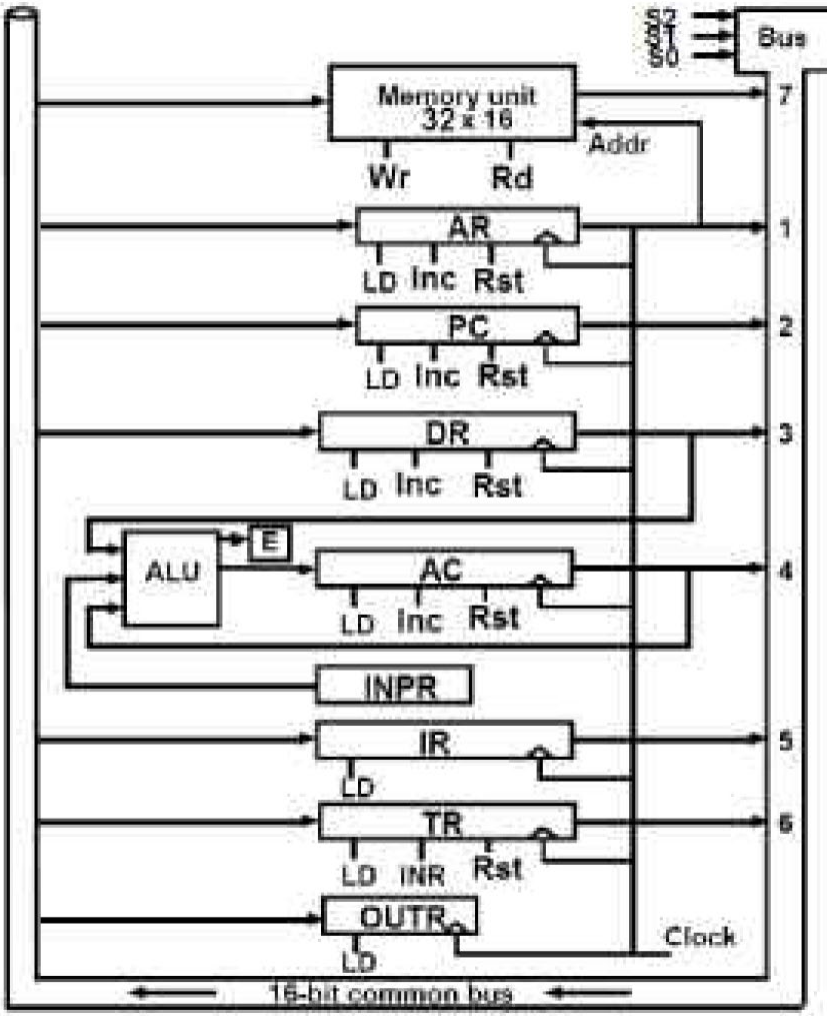


در این قسمت با توجه به اینکه Control_Cmd دارای ۷ لاین است قادر به انجام ۷ دستور در ALU هستیم که البته با توجه به مقداری که Cin می‌تواند بعنوان ورودی داشته باشد این مقدار افزایش نیز می‌یابد. این افزایش خاص دستورات محاسباتی مانند افزایش به مقدار یک یا همان increment است. در اینجا با توجه به اینکه مدار منطقی به طور کامل نشان داده شده است و از قبل Full Adder را پیاده سازی کرده بودیم یک نمونه از آن را در اینجا نیز می‌آوریم و ورودی و خروجی‌ها را مطابق شکل مشخص می‌کنیم. سیگنال‌های میانی که خروجی گیت‌های AND هستند نیز در اینجا تعریف شده و مورد استفاده قرار می‌گیرند. همه این خروجی‌ها که با توجه به سیگنال Control_Cmd تنها یکی از آنها فعال است (این فعال بودن می‌توان صفر یا یک باشد و به معنی یک بودن حتمی نیست) خروجی اصلی در ALU_Out_i ظاهر خواهد شد. حال با استفاده از Cin و Cout های متوالی و نمونه‌سازی ۱۶ عدد از این واحد محاسباتی تک بیتی می‌توانیم ALU شانزده بیتی مورد نظر را پیاده‌سازی کنیم.

برای اینکار ورودی Cin هر یک از ورودی Cout قبلی تامین می‌شود و در ورودی دادن به AC_i, AC_iPlus1, AC_iMin1 دوباره این توالی را مشاهده می‌کنیم. با توجه به این موارد و دید سیستمی که در مباحث قبل پیدا کرده بودیم، می‌توان ساختار ALU مورد نظر را که در نهایت قرار است در یک بخش از معماری مانو مورد استفاده قرار بگیرد طراحی کنیم.

بخش سوم

در این بخش ابتدا همانند نمونه یک BUS برای چهار ورودی می‌سازیم که سیگنال کنترلی آن دو بیت هستند، اما برای پیاده‌سازی Data path مانو نیاز به سه بیت کنترلی داریم چرا که هفت ورودی در BUS داریم که باید از میان آن‌ها خروجی مورد نظر را انتخاب کنیم.



در اینجا برای پیاده‌سازی کامل تمامی ماژول‌ها را بطور جداگانه تولید کرده‌ایم (اگر طراحی BUS سه بیتی را هم در این فرآیند لحاظ کنیم) و تنها نیاز به نمونه‌سازی و قرار دادن موارد بطور کافی در کنار هم داریم. لازم به ذکر است که کد مربوط به رجیستر ۱۶ بیتی نیز همانگونه که در فایل پروژه بود پیاده‌سازی شده است و نمونه کوچکتر ۸ بیتی آن که برای OUTR مورد استفاده قرار می‌گیرد نیز بدون سیگنال Inc در همان فایل نهایی وجود دارد.

در نهایت نیاز به ۱۰ نمونه‌سازی داریم که شش عدد آن مربوط به رجیسترها، یکی برای E flip-flop، یکی برای ALU شانزده بیتی، یکی برای رجیستر ۸ بیتی OUTR و در نهایت یکی هم برای BUS داده است. البته با توجه به نحوه چینش رجیسترها این BUS برای انتقال آدرس نیز مورد استفاده قرار می‌گیرد و کارکرد Address/data bus را دارد، اما سیگنال‌های کنترلی از بیرون به ما خواهند رسید. در این بین برای اتصال این ماژول‌ها به یکدیگر نیاز به سیگنال‌های میانی داریم که به تعداد مناسب تعریف شده‌اند و مورد استفاده نیز قرار می‌گیرند.

طراحی سیستم با دید سیستمی نسبتا سر راست بوده چرا که نحوه اتصال‌ها تقریبا مشابه هم بوده و سعی شده که براحتی قابلیت گسترش بیشتر برای سیستم فراهم شود. تنها رجیستری که بصورت مستقیم از BUS ورودی نمی‌گیرد AC است که برای ورودی دادن به آن باید ابتدا داده را به DR منتقل کنیم و سپس از آن روی AC قرار بدهیم. تعدادی از رجیسترها هم برای ذخیره و دانستن آدرس مورد استفاده قرار می‌گیرند و داده‌ها هیچگاه روی آن‌ها قرار نخواهند گرفت. قابل ذکر است که یکی از ورودی‌های BUS نیز (ورودی صفر) استفاده نشده است، هرچند بدلیل عملکرد نسبتا کامل همین سیستم نیازی به اضافه کردن آن نخواهد بود.