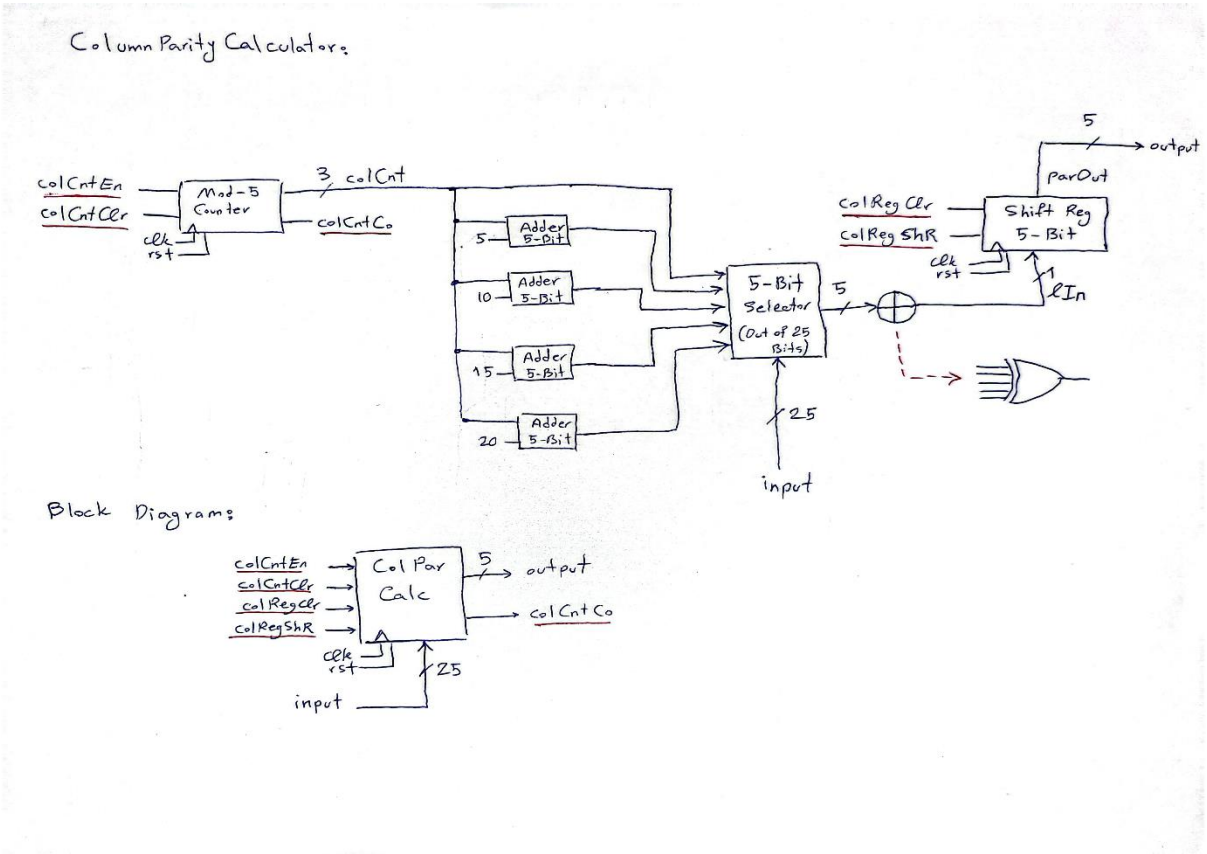
مسیر داده

برای بزرگ نشدن و خوانا بودن مسیر داده، این بخش را به چند ماژول و زیرماژول تقسیم کردیم که به صورت زیر است:

1. ماژول ColumnParityCalculator

این ماژول یک آرایه 25 بیتی را به عنوان ورودی می‌گیرد و نتیجه xor هر ستون را در محاسبه کرده و در یک شیفت رجیستر ذخیره می‌کند:



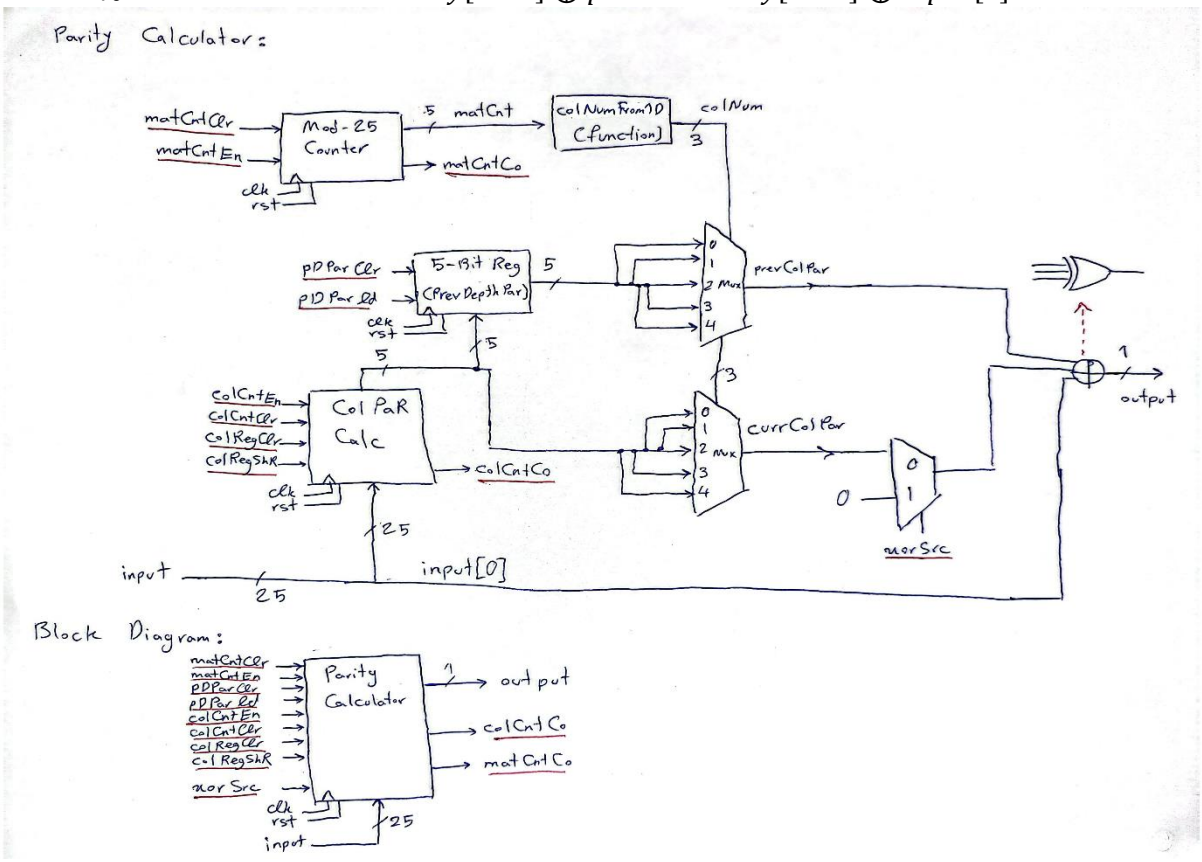
این ماژول در واقع 5 عنصر هر ستون را به صورت همزمان دریافت کرده و نتیجه xor آن‌ها را در شیفت رجیستر ذخیره می‌کند. ماژول 5-Bit Selector نیز در واقعیت از 5 عدد مولتی‌پلکسر تشکیل می‌شود. استفاده از این ماژول زمانی است که یک ورودی جدید 25 بیتی وارد مدار می‌شود و در ابتدا xor هر ستون را محاسبه می‌کنیم تا در آینده از آن استفاده کنیم. اگر مقدار خروجی counter برابر با i باشد، این مقدار نشان‌دهنده شماره ستون از سمت چپ می‌باشد و اندیس عناصر این ستون در آرایه تک بعدی به صورت زیر خواهند بود:

- i
- i + 5
- i + 10
- i + 15
- i + 20

2. ماژول ParityCalculator

در این ماژول ابتدا با استفاده از ColumnParityCalculator مقدار xor عناصر هر ستون را محاسبه می‌کنیم. مقدار xor تمامی ستون‌های عمق قبلی نیز در رجیستر 5 بیتی دیگری قرار دارد. با توجه به اینکه ورودی این ماژول در واقع خروجی یک شیفت رجیستر است، برای پیدا کردن A[i][j][k] کافی‌ست راست‌ترین بیت ورودی را در نظر بگیریم که پس از هربار شیفت دادن، تغییر می‌کند. از طرف دیگر باید شماره ستون این عنصر (i) را پیدا کنیم. برای این کار ابتدا یک Mod-25 Counter خواهیم داشت که اندیس عنصر مورد نظر را نشان می‌دهد که این مقدار با گذر از تابعی با نام colNumFrom1D، مقدار i را خروجی می‌دهد. این تابع در واقع باقی‌مانده اندیس آرایه تک بعدی بر 5 را به عنوان مقدار i برمی‌گرداند. در نهایت مقدار 1 بیت خروجی این ماژول از رابطه زیر بدست می‌آید:

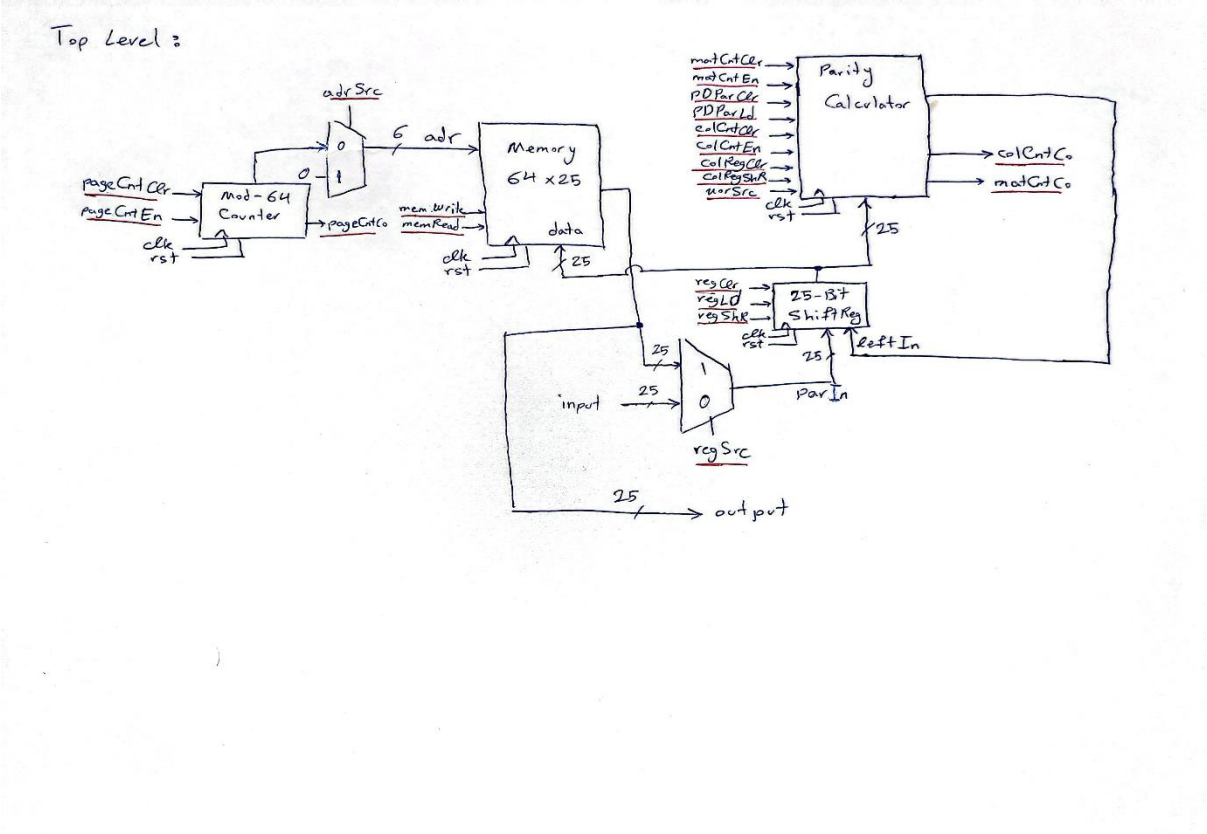
$$i = cnt \,\%\, 5 \longrightarrow res = currColParity[i-1] \oplus prevColParity[i+1] \oplus input[0]$$



مولتی‌پلکسر با سلکت xorSrc به این دلیل قرار داده شده است که در عمق اول، ستون i-ام عمق اول باید با ستون 1+i-ام عمق آخر xor شود که این کار پس از پایان تمام عمق‌ها انجام می‌شود. در این حالت به جای اینکه هر عنصر این عمق را با یک ستون از عمق فعلی نیز xor کنیم (این کار در مرحله قبلی انجام می‌شود)، با مقدار 0 باید xor کنیم.

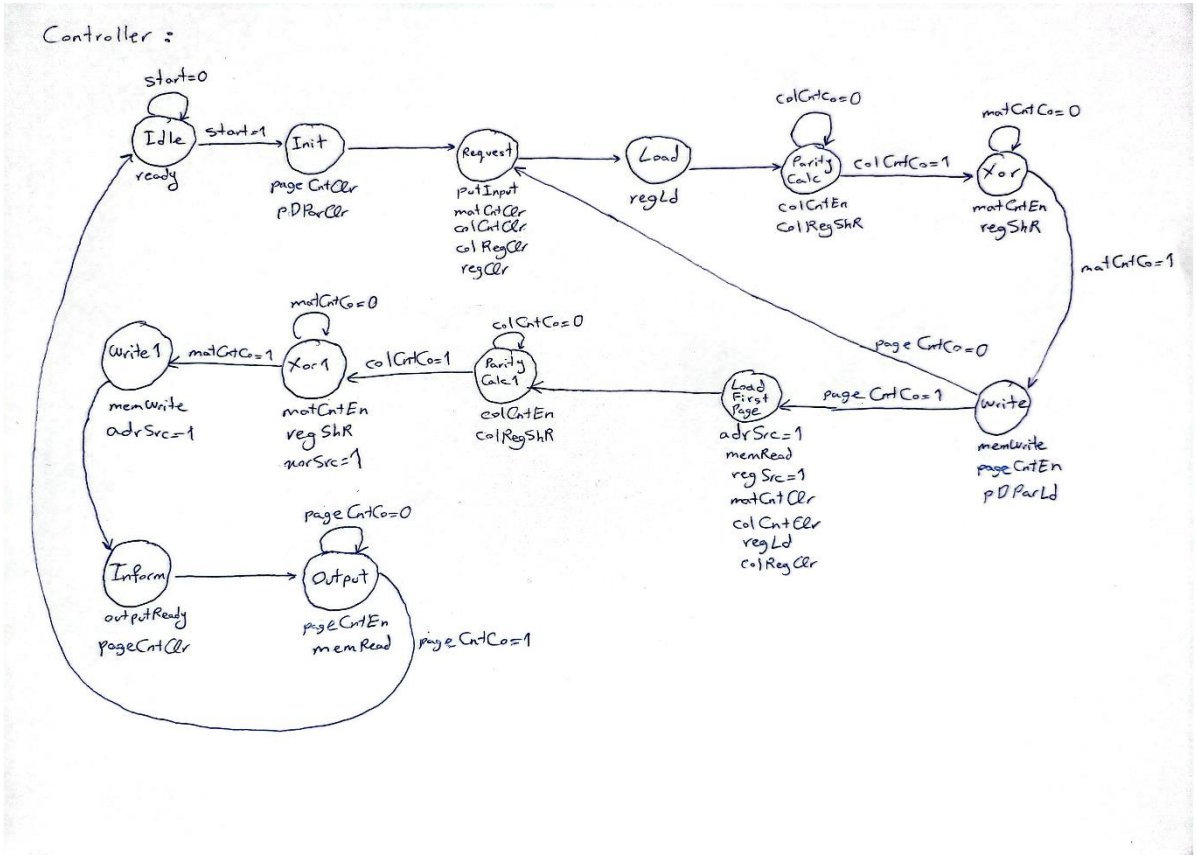
3. ماژول Main

این ماژول در واقع بخش از ماژول قبلی و مموری و رجیستر برای محاسبات اصلی استفاده می‌کند:



رجیستر قرار داده شده در این ماژول یک شیفت رجیستر است. هر بار که یک بیت از خروجی در ماژول ParityCalculator محاسبه می‌شود، به عنوان بیت MSB وارد این شیفت رجیستر شده و تمامی بیت‌ها یک واحد شیفت می‌خورند. این کار 25 بار تکرار می‌شود تا اینکه تمامی بیت‌های خروجی از رجیستر خارج شود و نتایج به صورت کامل در این رجیستر ریخته شود. این کار برای صرفه‌جویی در استفاده از رجیستر انجام شده است. در نهایت پس از 25 بار تکرار این کار، خروجی این رجیستر در مموری ذخیره می‌شود. در این ماژول تعدادی مولتی‌پلکسر قرار داده شده که کاربرد آن‌ها زمانی است که نتایج تمامی اعماق را محاسبه کرده و باید عمق اول را به دلیل xor کردن با ستون‌های عمق آخر مجددا محاسبه کنیم. به کمک این مولتی‌پلکسرها عمق اول را مجددا وارد مدار کرده و محاسبات لازم را انجام می‌دهیم.

واحد کنترل



جزئیات راه حل در بخش‌های قبل توضیح داده است. در این بخش واحد کنترل را به صورت کلی توضیح می‌دهیم. ابتدا به کمک سیگنال ready و start یک handshaking بین ماژول اصلی و تست‌بنچ صورت می‌پذیرد. پس از مقداردهی‌های اولیه، ماژول سیگنال putInput را فعال می‌کند که تست‌بنچ ورودی اول را قرار دهد. این کار و تعدادی از مراحل آینده 64 بار اجرا خواهد شد. در هر بار اجرا ابتدا ورودی را می‌خوانیم و در رجیستر ذخیره می‌کنیم و خروجی مدنظر را تولید می‌کنیم و در مموری قرار می‌دهیم. پس از تکرار این مراحل به تعداد 64 بار، باید آرایه مربوط به عمق اول را مجددا وارد مدار کنیم و xor عناصر آن با ستون‌های عمق آخر را محاسبه کنیم. در نهایت خروجی این عمق را نیز مجددا در مموری قرار می‌دهیم. سپس counter را ریست کرده و با فعال‌سازی سیگنال outputReady تست‌بنچ را از آماده شدن خروجی مطلع می‌سازیم. پس از آن در هر کلاک یک سطر از مموری را بر روی خروجی قرار می‌دهیم تا توسط تست‌بنچ دریافت شود.

تغییرات ایجاد شده نسبت به طراحی ارائه شده (میثاق محقق)

ایده کلی طراحی ثابت مانده ولی تغییراتی در نحوه پیاده سازی آن انجام گرفته است.
به چند نکته در اینجا می‌توان اشاره کرد:

نکات طراحی اشتباه:
1. در طراحی حین آزمون، به اشتباه فقط یک بیت از Parity صفحه قبلی سیو نگه داشته شده و از آن به عنوان parity(A[i+1][0…4][z-1]) استفاده شده است. این درحالیست که این مقدار به ازای i های مختلف متفاوت بوده و باید کل رجیستر 5 بیتی کپی می‌شد.
2. استفادهٔ صفحه اول از col parity های صفحه قبلش (یعنی صفحه 64ام) هندل نشده است و برای صفحه اول مقدار اولیه 0 رجیستر نگه‌دارندهٔ parity های قبلی استفاده می‌شود که تأثیری درش ندارد.
3. به دلیل کمبود وقت خیلی از سیگنال های دیتاپف نوشته نشده و کنترلر هم از این جهت ناقص می‌باشد.

نکات طراحی متفاوت:
1. در طراحی آزمون برای استیت‌هایی که توسط counter ای تکرار می‌شدند دو استیت در نظر گرفته شده ولی اینجا با چرخه زدن روی خود استیت حل شده است.
این انتخاب به خاطر تردید در امکان وجود مشکل در enable کردن counter در همان استیتی که چک می‌شود بود و مشکلی ایجاد نمی‌کند.
2. در طراحی آزمون برای بخش ParityCalc که همه خانه‌های صفحه را پیمایش کرده و مقدار جدیدش را می‌گذارد، از دو counter استفاده شده که مثل دو چرخه درهم 5*5 اجرا می‌شود. یعنی به ازای هر خانه سطر پایین ماتریس، ابتدا کل ستون اش تغییر کرده و سپس به بعدی می‌رود ولی در اینجا هر 25 خانه به ترتیب پیمایش می‌شوند.
این انتخاب به دلیل سعی در موازی سازی تغییر هر 5 المان ستون‌ها با هم بوده که با شکست مواجه شد و جز داشتن یک counter بیشتر فرقی با پیمایش 25 خانه‌ای ندارد.
3. در آزمون برای تغییر یک بیت رجیستر از module ای به نام BitLoader استفاده شده که خروجی رجیستر، اندیس i و مقدار مورد نظر را گرفته و محتوای جدید را خروجی می‌دهد که باید با multiplexer ای وارد رجیستر بشود.
در طراحی کنونی از shift register ها استفاده شده و نیازی به این module نیست.

تغییرات ایجاد شده نسبت به طراحی ارائه شده (پاشا براهیمی)

طراحی نهایی در واقع ترکیبی از طراحی من و میثاق است. در بخش‌هایی تفاوت وجود دارد که در ادامه ذکر می‌شود.
در ابتدا در ماژول ColumnParityCalculator مشکلاتی وجود داشت که رفع شد. ورودی adderهای این ماژول باید مضارب 5 باشند در صورتی که در طراحی من ورودی همه adderها برابر با 5 بود. رجیستر قبل از xor حذف شد و به جای آن ماژول 5-Bit Selector قرار گرفت زیرا نیازی به این رجیستر نبود. رجیستر انتهایی این ماژول هم تبدیل به شیفت رجیستر شد.

در ماژول بخش Top-Level یک ColumnParityCalculator وجود داشت که همین مورد در ماژول ParityCalculator نیز وجود دارد با این تفاوت که مورد اول برای محاسبه xor ستون‌های عمق قبلی است و مورد دوم برای محاسبه xor ستون‌های عمق فعلی؛ در نتیجه با توجه به تکراری بودن این ماژول، از Top-Level حذف شد و فقط رجیستر آن به ParityCalculator منتقل شد. در واقع الان پس از محاسبه یک عمق کامل، مقدار xor ستون‌های عمق فعلی، به رجیستر اضافه شده منتقل می‌شود که مقدار xor ستون‌های عمق قبلی را نشان دهد. همچنین، Mod-25 Counter نیز به داخل این ماژول منتقل شد و رجیستر Result نیز حذف شد و رجیستر اصلی این ماژول نیز به صورت شیفت رجیستر قرار گرفت تا مقدار خروجی نیز در همین رجیستر ذخیره شود. این کار نیز برای صرفه‌جویی در استفاده از رجیستر انجام شد. سپس یک ممووری نیز در ماژول قرار دادیم تا بتوانیم بخش xor شدن عمق اول با عمق آخر را هندل کنیم. در نتیجه این مورد، تعدادی مولتی‌پلکسر نیز به ماژول اضافه شد.

در ماژول ParityCalculator نیز از لحاظ منطق تغییر خاصی صورت نگرفت و فقط کمی بهینه‌سازی انجام شد، از کامپوننت‌های معقول‌تری نظیر مولتی‌پلکسر به جای Partial Register Reader (خواندن اندیس یا اندیس‌های خاصی از رجیستر به طوری که اندیس مورد نیاز یکی از ورودی‌های رجیستر است) استفاده شد و همچنین یک مولتی‌پلکسر نیز برای هندل کردن xor شدن عمق اول با عمق آخر قرار دادیم. همچنین، همانطور که پیش‌تر گفته شد، Mod-25 Counter نیز به این ماژول منتقل شد.

کنترلر طراحی شده در زمان امتحان به دلیل کمبود وقت، کمی نقص داشت که در این بخش تکمیل شد.

در کل بیشترین تغییرات مربوط به بهینه‌سازی و استفاده از کامپوننت‌های معقول‌تر بود و طراحی در زمان امتحان به جز موارد زیر با کمی تصحیح قابل اجرا می‌بود:

- هندل نشدن xor عمق اول با عمق آخر
- نقص‌های کنترلر