به نام خدا

دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

طراحی سیستم‌های دیجیتال

پروژه‌ی پایانی درس:

ضرب‌کننده‌ی ماتریسی با استاندارد IEEE 754

استاد: **دکتر فرشاد بهاروند**

عماد زین‌اوقلی، مازیار شمسی‌پور، بردیا محمدی، محمدجواد هزاره، پویا یوسفی

۲۳ تیر ۱۴۰۰

فهرست مطالب

شرح وظایف

اعضای گروه	وظایف	درصد مشارکت
محمد جواد هزاره	طراحی Main Control Unit و تست‌بنچ، تکمیل طراحی Top module و تست‌بنچ.	20%
مازیار شمسی‌پور	طراحی Control Unit و تست‌بنچ، طراحی Top Module و تست‌بنچ. تکمیل گزارش	20%
عماد زین‌اوقلی	طراحی Memory و Arbiter و تست‌بنچ، تکمیل طراحی Matrix Multiplier و تست‌بنچ.	20%
پویا یوسفی	طراحی Golden Model و انجام عملیات سنتز. تکمیل گزارش	20%
بردیا محمدی	طراحی Matrix Multiplier و تست‌بنچ. تکمیل گزارش	20%

۱ مقدمه

۱.۱ تعریف الگوریتم

الگوریتم مورد استفاده الگوریتم ضرب ماتریسی Cannon می‌باشد در این الگوریتم با تقسیم کردن ماتریس‌های ورودی و خروجی به بلاک‌های $k * k$ که در آن k عدد ثابتی می‌باشد می‌خواهیم با داشتن تعدادی پردازنده که به صورت موازی کار می‌کنند عملیات ضرب ماتریسی را بهبود ببخشیم. به طور مثال ماتریس‌ها زیر را در نظر بگیرید:

$$A = \begin{bmatrix} A_{11} & A_{12} & \dots A_{1\mu} \\ \vdots & \ddots & \vdots \\ A_{\lambda 1} & A_{\lambda 2} & \dots A_{\lambda\mu} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} & \dots B_{1\gamma} \\ \vdots & \ddots & \vdots \\ B_{\mu 1} & B_{\mu 2} & \dots B_{\mu\gamma} \end{bmatrix} \tag{۱}$$

که در آن هر $A_{ij}B_{ij}$ یک بلاک $k * k$ می‌باشد.(توجه می‌کنیم که سایز ماتریس‌ها اگر بخش‌پذیر به k نباشد با اضافه کردن صفر آن را بخش پذیر می‌کنیم) با این اوصاف طبق قاعده‌ی ضرب بلوکی می‌دانیم که بلاک C_{ij} در ماتریس جواب از رابطه‌ی زیر محاسبه می‌شود.

$$C_{ij} = \sum_{x=0}^{\mu} A_{ix}B_{xj} \tag{۲}$$

با داشتن تعداد تعداد مشخصی ضرب کننده‌ی ماتریسی $k * k$ می‌توانیم به طور موازی با استفاده از آنها و پخش کردن C_{ij} ها بین پردازنده‌های مختلف حاصل نهایی $A \times B$ را محاسبه کنیم.
در ادامه‌ی این گزارش از علائم ریاضی‌ای استفاده می‌شود که در اینجا به شرح آنها می‌پردازیم.

۲.۱ قراردادهای ریاضی

ورودی الگوریتم مورد استفاده ماتریس‌های مستطیلی A_{mr} و B_{rn} خواهند بود و بنابراین ماتریس خروجی به صورت $A_{mr} \times B_{rn} = C_{mn}$ خواهد بود. با این حال در هر کجای گزارش که از عبارت A_{ij} (و همینطور برای B, C) استفاده شد منظور بلاک $k * k$ ستون iام و سطر jام می‌باشد. برای روشن‌تر شدن این موضوع به مثال زیر توجه می‌کنیم، فرض کنید ماتریس A به صورت زیر باشد:

$$A_{mr} = \begin{bmatrix} a_{00} & a_{01} & \dots & a_{0r} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m0} & a_{m1} & \dots & a_{mr} \end{bmatrix}$$

حال اگر این ماتریس را به بلوک‌های $k * k$ تقسیم کنیم و در صورت لزوم درایه‌های نهایی را صفر قرار دهیم ماتریسی به فرم زیر خواهیم داشت:

$$A^* = \left[\begin{array}{ccc|c} A_{00} & A_{01} & \dots & A_{0\mu-1} & \\ \vdots & \vdots & \ddots & \vdots & 0 \\ A_{\lambda-10} & A_{\lambda-11} & \dots & A_{\lambda\mu-1} & \\ \hline 0 & & & & 0 \end{array}\right]$$

که لازم است که توجه داشته باشیم که وقتی ماتریس‌ها را به فرم بلوکی می‌نویسیم مقادیر زیر را تعریف می‌کنیم:

$$\mu = \left\lceil \frac{r}{k} \right\rceil \quad (۳آ)$$

$$\lambda = \left\lceil \frac{m}{k} \right\rceil \quad (۳ب)$$

$$\gamma = \left\lceil \frac{n}{k} \right\rceil \quad (۳ج)$$

$$\theta = \lceil \frac{\lambda\gamma}{\#\text{Matrix Processors}} \rceil \quad (۳د)$$

از این نمادها به کرّات در طول گزارش استفاده خواهد شد. توجه می‌کنیم که علت اینکه سقف این حاصل تقسیم‌ها را در نظر گرفتیم همان است که اگر اندازه‌ی ماتریس‌ها بر k بخش‌پذیر نباشند با اضافه کردن صفر به انتهای آن باعث بخش‌پذیری می‌شویم.

۳.۱ نحوه‌ی عملکرد از نظر مساحت و تایمینگ

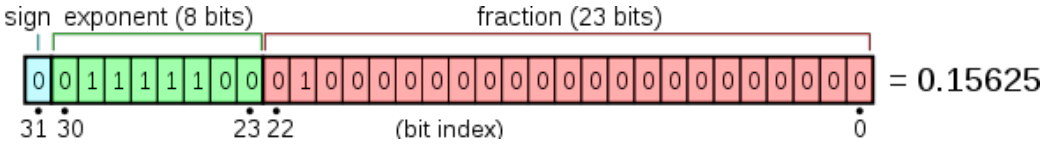
از آنجایی که هر ضرب کننده‌ی ماتریسی در حدود k^3 کلاک سایکل زمان می‌برد و محاسبه‌ی هر بلوک C_{ij} با توجه به معادله ۲ به μ بار به ضرب ماتریسی نیاز دارد. همچنین برای محاسبه‌ی تمام بلوک‌ها باید $\lambda\gamma$ بار محاسبات بالا را انجام دهیم با این حال اگر فرض کنیم که تعداد پردازنده‌ها p باشد آنگاه می‌توانیم ببینیم که تعداد کلاک سایکل‌ها تقریبا برابر با عبارت زیر است:

$$\frac{\lambda\gamma\mu k^2}{\#\text{number of PU}} = \frac{\lambda\gamma\mu k^2}{p} \tag{۴}$$

همچنین تعداد رجیسترهایی که هر واحد ضرب‌کننده‌ی ماتریس مربعی نیاز دارد از $O(k^2)$ می‌باشد. و بنابراین تعداد تمام رجیسترهایی که مورد نیاز است از $O(pk^2)$ می‌باشد.

۴.۱ استاندارد IEEE 754

محاسبات در این پروژه از استاندارد IEEE 754 - Single-precision floating-point پیروی می‌کند که به طور مختصر به شرح آن می‌پردازیم.
در این استاندارد اعداد اعشار با سه بخش exponent ، fraction ، sign مشخص می‌شوند که سهم هر یک از آنها مانند مثال زیر است:



و هر عدد طبق فرمول زیر به این نمایش در می‌آید:

$$\text{value} = (-1)^{sign} \times 2^{(E-127)} \times (1 + \sum_{i=1}^{23} b_{23-i}2^{-i}) \tag{۵}$$

۵.۱ کاربردها

محاسبات ماتریسی، در سیستمهای پردازش تصویر، در سیستمهای مخابراتی MIMO ،در سیستم های مخابراتی که از روش OFDM برای ارسال اطلاعات استفاده می کنند و همچنین در سیستم های رادار و سونار که مراحل آشکارسازی و تخمین را به کمک اطاعاتی که از آرایه ای از سنسورها جمع آوری شده انجام می دهند، کاربرد فراوانی دارند. واحد ضرب ماتریس در برنامه های پردازش سیگنال دیجیتال مانند تصویربرداری دیجیتال، پردازش سیگنال، گرافیک رایانه ای و چندرسانه ای استفاده می شود. بنابراین بسیار مهم است که مساحت کمتری اشغال کند، سریع کار کند و انرژی کمی مصرف کند. در این پروژه، به چگونگی محاسبه ی ضرب ماتریس ها و پیاده سازی آن، به عنوان یک بخش از مباحث محاسبات ماتریسی پرداخته شده است.

۶.۱ مراجع مورد استفاده

References

[1] Abhishek Kumar : Scalability of Parallel Algorithms for Matrix Multiplication

[2] Patricia Ortega : Parallel Algorithm for Dense Matrix Multipication

[3] Ju-wook Jang, Seonil Choi and Viktor K. Prasanna : Area and Time Efficient Implementations of Matrix Multiplication on FPGAs

[4] Cannon's algorithm, Wiki-pedia
https://en.wikipedia.org/wiki/Cannon%27s_algorithm

۲ توصیف معماری سیستم

۱.۲ اینترفیس‌های سیستم و قرارداد استفاده از آن

به طور کلی و از نگاه بالا سخت‌افزار از یک حافظه و و کمک‌ـپردازنده‌ی ضرب ماتریسی[۱] تشکیل شده است که پردازنده برای استفاده از می‌تواند ورودی‌ها را درون حافظه قرار داده و خروجی‌ها را نیز از آن بخواند(I/O Mapped).
برای استفاده از این کمک‌ـپردازنده قراردادهایی در نحوه‌ی استفاده از مموری وجود دارد که باید به آن توجه شود.

۱.۱.۲ ساختار کلی حافظه

ساختار کلی حافظه به صورت زیر خواهد بود:

Config
Status
A_{11}
A_{12}
⋮
$A_{\lambda\mu}$
B_{11}
B_{12}
⋮
$B_{\mu\gamma}$
C_{11}
C_{12}
⋮
$C_{\lambda\gamma}$

جدول ۱: شماتیک حافظه

که در آن هر یک از A_{ij}, B_{ij}, C_{ij}ها یک بلوک $k * k$ خواهند بود و باید آن‌ها را به صورت سطری در خانه‌های پشت سر هم حافظه نوشت.

[۱] Matrix Multiplier - Co-processor

برای مثال اگر ماتریس‌های A, B به صورت زیر باشند:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

و ماتریس خروجی به صورت $C = \begin{bmatrix} 22 & 28 \\ 48 & 64 \end{bmatrix}$ خواهد بود. در صورتی که $k = 2$ و به عبارتی واحدهای درونی ضرب کننده‌های ماتریس مربعی ما توانایی ضرب بلوک‌های $2 * 2$ را داشته باشند؛ CPU باید آن را به صورت زیر در حافظه قرار دهد و همچنین بلوک‌های خروجی را از بخش‌های مشخص شده استخراج کند: : نکته حائز توجه دیگر نقطه‌ی شروع ماتریس‌های خروجی می‌باشد که تنها کافیست

Config
Status
۱
۲
۴
۵
۳
۰
۶
۰
۱
۲
۳
۴
۵
۶
۰
۰
۲۲
۲۸
۴۸
۶۴

جدول ۲: شماتیک حافظه برای مثال داده شده

توجه شود که به جز دو خانه‌ی اول حافظه بقیه‌ی خانه‌ها به صورت یکسان بین ماتریس‌های ورودی و ماتریس خروجی تقسیم شده‌است. یعنی اگر اندازه‌ی کل مموری را N در نظر بگیریم $\lceil \frac{N-2}{3} \rceil$ خانه به خروجی اختصاص پیدا می‌کند.

۲.۱.۲ خانه‌ی اول حافظه

CPU باید اولین خانه‌ی حافظه را که مربوط به کانفیگ می‌باشد به صورت زیر از اعداد پر کند:

$\underbrace{\theta}_{8\ bits}$	$\underbrace{\mu}_{8\ bits}$	$\underbrace{\gamma}_{8\ bits}$	$\underbrace{\lambda}_{8\ bits}$

که مقادیر این پارامترها در معادله ۳ مشخص شده است و همچنین باید توجه شود که مقدار θ نیز از رابطه‌ی زیر محاسبه می‌شود:

برای مثال فرض کنیم که ماتریس‌های A_{55} ، B_{55} در اختیار داشته باشیم. همچنین 4 پردازنده ضرب ماتریسی $3 * 3$ در اختیار داشته باشیم، پارامترهای مد نظر به صورت زیر محاسبه خواهند شد:

$$\lambda = \lceil \frac{m}{k} \rceil = \lceil \frac{5}{3} \rceil = 2$$

$$\gamma = \lceil \frac{m}{k} \rceil = \lceil \frac{5}{3} \rceil = 2$$

$$\mu = \lceil \frac{m}{k} \rceil = \lceil \frac{5}{3} \rceil = 2$$

$$\theta = \lceil \frac{\lambda\gamma}{\#\text{Matrix Processors}} \rceil = \lceil \frac{4}{4} \rceil = 1$$

و بنابراین خانه‌ی اول حافظه در هنگامی که کمک‌ـپردازنده دستور شروع به کار را دریافت می‌کند باید به صورت زیر باشد:

00000001	00000010	00000010	00000010

۳.۱.۲ خانه‌ی دوم حافظه

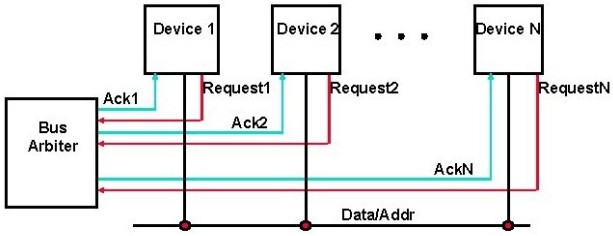
همچنین دومین خانه‌ی حافظه که مربوط به Status می‌باشد مطابق شکل زیر می‌باشد.

CPU Ready	C-P Acknowledge	...	CPU Acknowledge	C-P Ready

وظیفه‌ی CPU این است که بعد از قرار دادن ورودی‌ها و تنظیم کردن Config مقدار بیت CPU Ready را فعال کند و بعد از این که بیت Acknowledge را از طرف کمک‌ـپردازنده دریافت کرد به کارش ادامه دهد بعد از تمام شدن عملیات ضرب ماتریسی بیت C-P Ready فعال می‌شود و CPU می‌تواند بلاک‌های ماتریس خروجی را از مکانی که در مموری مربوط به خروجی‌ها می‌باشد استخراج کند.

۴.۱.۲ نحوه‌ی دسترسی به حافظه

برای دسترسی به حافظه تمامی ماژول‌های موجود در سیستم و همچنین CPU از یک Round-Robin Arbiter استفاده می‌کنند، با این تفسیر که ماژولی بخواهد خط‌های متصل به حافظ[۲]را تغییر دهد باید از Arbiter اجازه‌ی دسترسی بگرد. و اگر Arbiter سیگنال Grant مربوط به آن ماژول را فعال کرد اجازه‌ی نوشتن روی حافظه در اختیار آن ماژول قرار می‌گیرد. برای روشن‌تر شدن این موضوع خوب است به شماتیک زیر توجه کنید:



شکل ۱: Arbiter

۵.۱.۲ ریست آسنکرون

سخت‌افزار دارای یک سیگنال reset آسنکرون می‌باشد که تمام رجیسترهای درونش را صفر می‌کند.

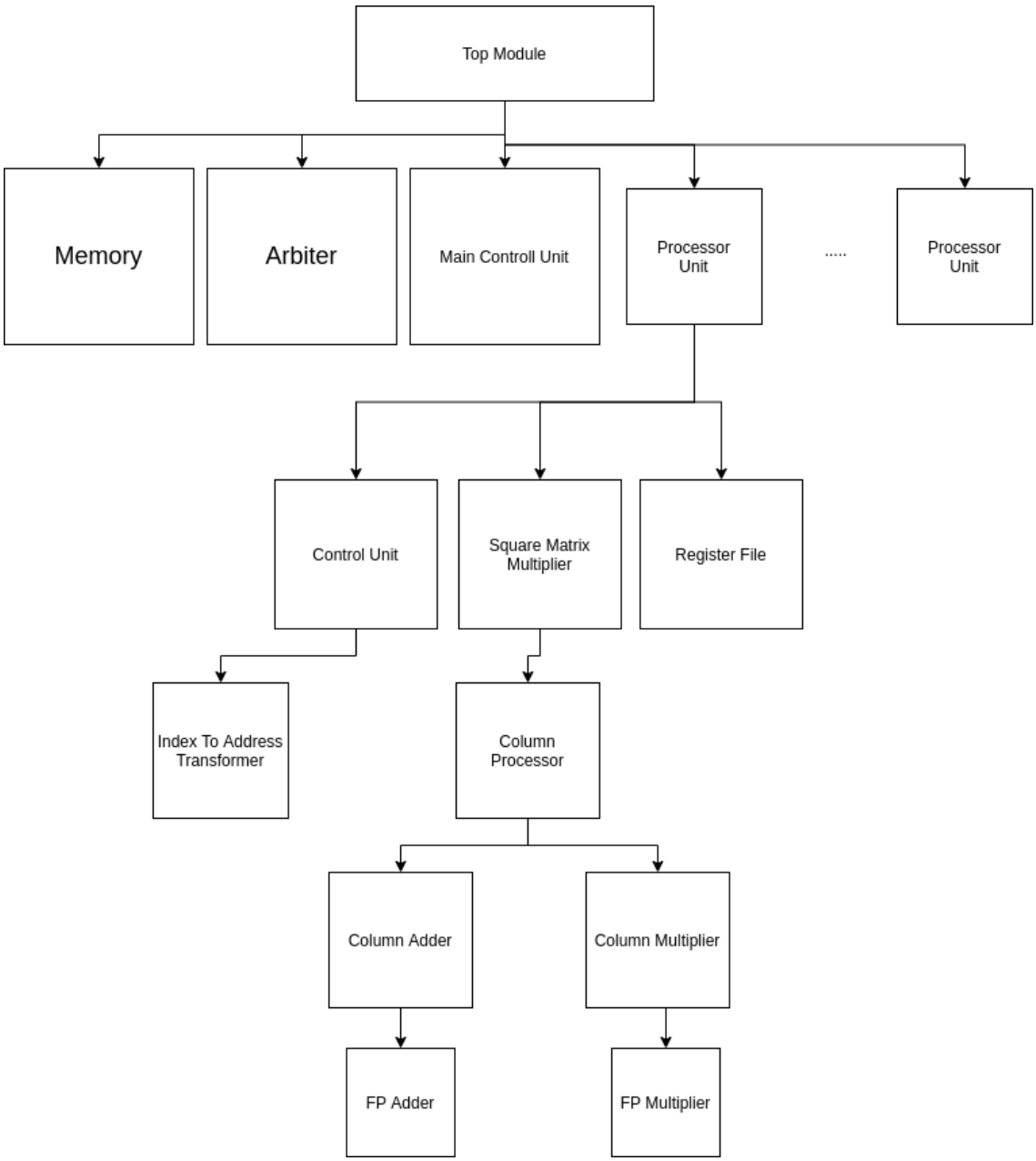
۶.۱.۲ کلاک سخت‌افزار

تمامی ماژول‌های این سخت‌افزار از جمله مموری و تمام ماژول‌های واحد حساب‌کننده‌ی ضرب ماتریسی به صورت سنکرون عمل می‌کنند و CDC در این سخت‌افزار اتفاق نمی‌افتد.

[۲] Memory Bus

ابتدا ساختار درختی سخت‌افزار طراحی شده را می‌بینیم و سپس به مفسراً درباره‌ی نقش و عملکرد هر یک از ماژول‌های مربوطه صحبت خواهیم کرد.

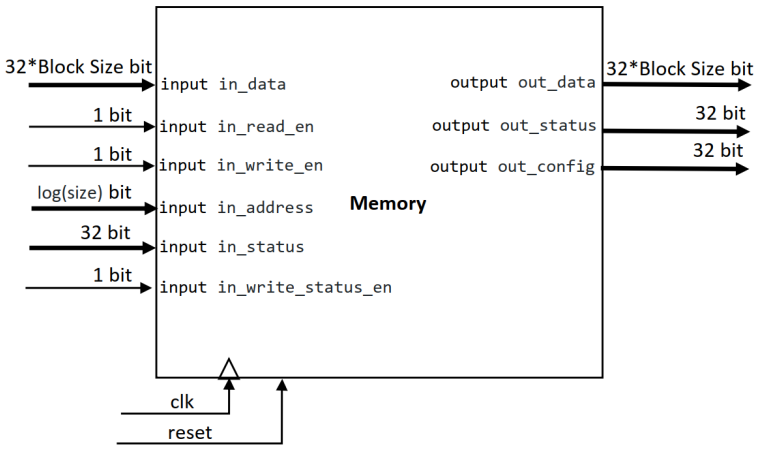
۲.۲ ساختار درختی سیستم



شکل ۲: Design Hierarchy

۳.۲ توصیف ماژول‌ها

- Memory



شکل ۳: Memory Schematic

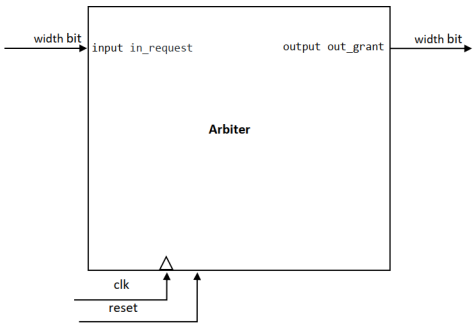
واحد حافظه سخت‌افزار که مطابق با استانداردی که در بخش‌های قبل مفسراً درباره‌ی آن توضیح دادیم می‌باشد.این واحد توانایی آدرس دهی به هر کلمه را دارد(Word Addressable). هر کلمه‌ی آن یک عدد ممیز شناور با استاندار IEEE 754 - Single Precision می‌باشد.

خواندن و نوشتن در آن این حافظه به منظور بهبود عملکرد زمانی به گونه‌ایست که در هر بار دسترسی به حافظه به تعداد k کلمه در آن نوشته یا از آن خوانده می‌شود.

هر ماژول دیگر در سخت افزار یا بیرون سخت افزار برای دسترسی به باس ورودی‌های ممووری باید از Arbiter اجازه گرفته باشد یعنی مطابق شکل ۱ باید سیگنال Request خود را فعال کند و سپس منتظر بماند که سیگنال Grant دریافت شود و بعد از آن می‌تواند عملیات‌های نوشتن و خواند روی حافظه را انجام دهد.

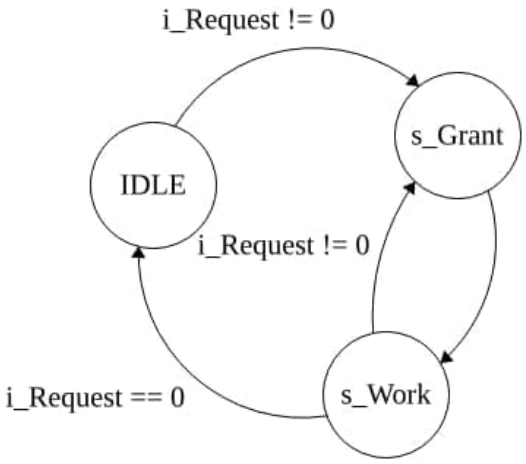
از آنجایی که Config و Status مکررا مورد نیاز ماژول‌ها در برنامه قرار می‌گیرد تصمیم گرفتیم که خواندن و نوشتن این دو (البته فقط نوشتن در Status) بدون نیاز داشتن به اجازه‌ی Arbiter و توسط Main Controller که در ادامه آن را توضیح می‌دهیم صورت بگیرد.

- Arbiter



شکل ۴: Arbiter Schematic

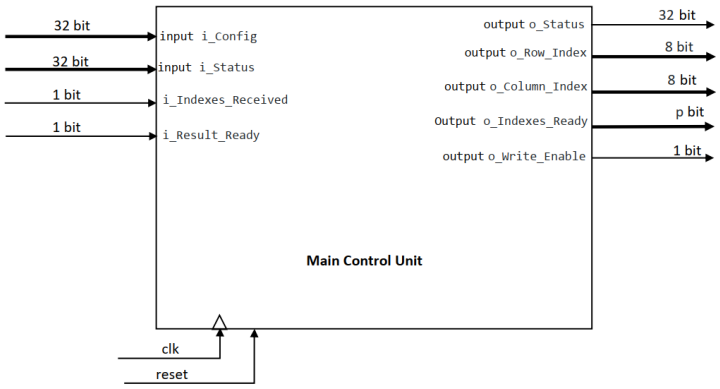
همانطور که برای Memory توضیح دادیم این واحد نقش پخش کردن اجازه‌ی دسترسی به مموری را بین ماژول‌ها دارد در شکل ۱ این موضوع مشخص است. مقدار پهنای ورودی و خروجی این ماژول متناسب با تعداد ماژول‌های دیگریست که اجازه‌ی دسترسی به حافظه را می‌خواهند. برای روشن شدن عملکرد این ماژول به FSM زیر توجه کنید:



شکل ۵: Arbiter Fsm

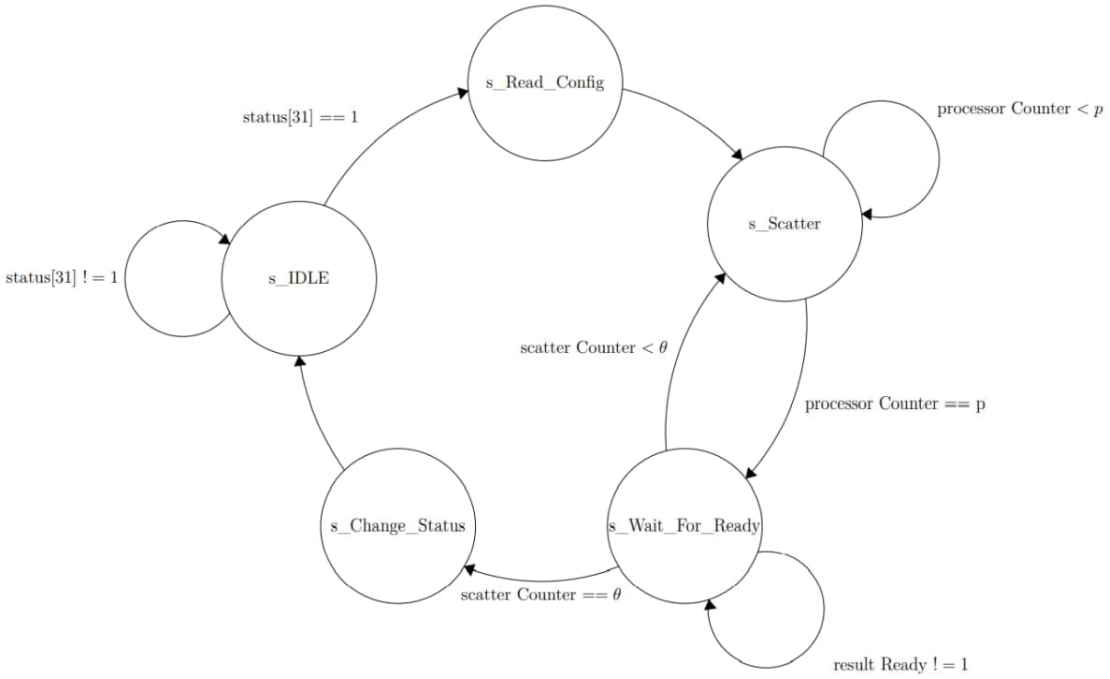
همانگونه که از این FSM مشخص می‌باشد هر گاه یکی سیگنال‌های Request فعال باشد Arbiter به حالت Grant می‌رود و برای ماژولی که درخواست داده و اولویت بالاتری دارد Grant تولید می‌کند. روشن است که سیگنال Grant، Hot-Bit می‌باشد. وقتی که سیگنال گرنت فعال شد Arbiter منتظر می‌ماند که همان ماژولی که Grant را دارد Request خود را قطع کند. بعد از آن دوباره در صورتی که Request داشتن ماژول دیگری به حالت تولید Grant می‌رود و در غیر این صورت به حالت اولیه بازگشته و منتظر می‌ماند تا Request یکی از ماژول‌ها فعال شود.

- Main Control Unit



شکل ۶: Main CU Schematic

این واحد وظیفه‌ی پخش کردن بلاک‌های C_{ij} بین پردازنده‌ها را دارد به نمودار حالت زیر توجه می‌کنیم:



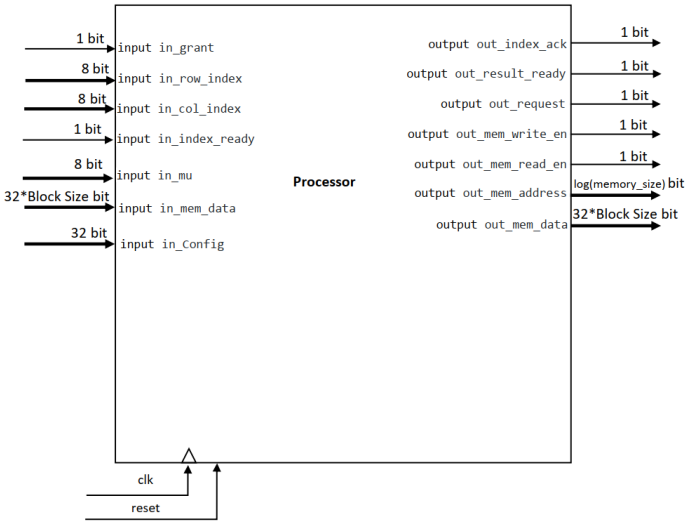
شکل ۷: Main CU Fsm

با یک شدن بیت آخر خانه‌ی status در حافظه، این واحد کار خود را آغاز کرده و به پردازنده مرکزی acknowledge می‌دهد که کار خود را آغاز کرده است.

در حالت بعدی به سراغ خواندن اطلاعات ماتریس‌های ضرب شونده از خانه config در حافظه رفته و این اطلاعات را در رجیسترهای میانی خود ذخیره می‌کند. پس از آن شروع به پخش کردن بلوک‌های ماتریس‌ها میان پردازنده‌ها کرده که این کار نیز به این صورت انجام می‌شود که منتظر سیگنال acknowledge از طرف پردازنده مورد نظر می‌ماند و اگر این سیگنال را دریافت کند به سراغ پردازنده بعدی می‌رود. این کار را تا زمانی ادامه می‌دهد که به تمامی پردازنده‌ها یک بلوک اختصاص یابد.

پس از اختصاص بلوک‌ها به تمامی پردازنده ها به حالت بعدی رفته و منتظر می‌ماند تا پردازنده‌ها کار خود را تمام کنند. با تمام شدن کار تمامی پردازنده‌ها، سیگنال result_ready یک شده و در این حالت اگر بلوکی برای اختصاص دادن باقی‌مانده بود، دوباره به حالت s_Scatter برگشته و تخصیص را انجام می‌دهد و در غیر اینصورت به حالت بعدی، یعنی s_Change_Status رفته و مطابق قرارداد، بیت اول خانه‌ی status حافظه را یک می‌کند که نشان می‌دهد کار ضرب‌ماتریس‌ها تمام شده و نتیجه نیز در حافظه نوشته شده است و پردازنده‌ی مرکزی می‌تواند خروجی را استخراج کند.

- Processor



شکل ۸: Processor Fsm

این واحد وظیفه دارد که با دریافت یک i, j و سیگنال‌های ورودی دیگر مقدار C_{ij} را محاسبه کرده و در حافظه ذخیره کند.

در واقع این واحد که متشکل از Register File ، Control Unit، Square Matrix Multiplier می‌باشد وظیفه‌ی برقراری اتصالات بین این ماژول‌ها را دارد تا کارهای زیر به درستی انجام شود:

(I) با توجه به معادله ۲ آدرس A_{ix} و B_{xj} از Control Unit بگیرد و به مموری بدهد سپس متناسب با این اندیس‌ها آدرسی برای Register File ایجاد کرده و سیگنال Write Enable آن را فعال کند تا Register File به درستی این ماتریس‌ها را از حافظه بخواند.

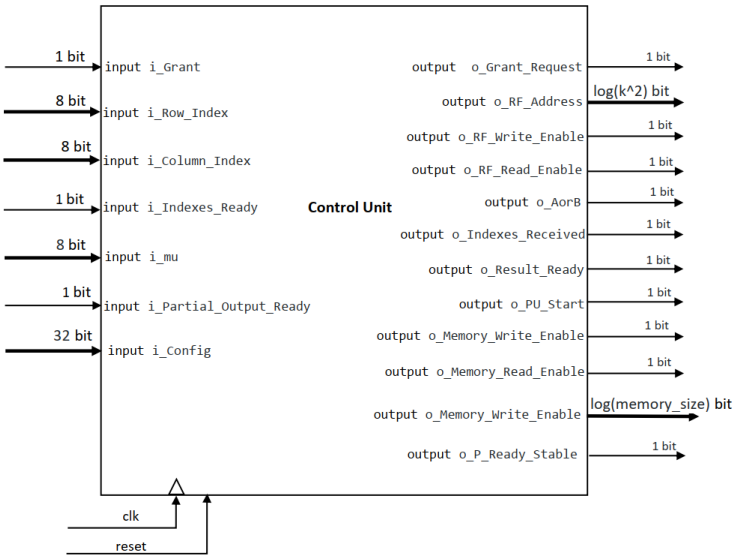
(II) بعد از اینکه Register File A_{ix} و B_{xj} شد باید سیگنال Start را از Control Unit گرفته و به Square Matrix Multiplier بدهد.

(III) داده‌های مورد نیاز Square Matrix Multiplier را در کلاک‌های مختلف از Register File خوانده تا عملیات ضرب ماتریس مربعی $A_{ix} \times B_{xj}$ به درستی انجام شود.

(IV) بعد از اتمام ضرب حاصل را با مقادیر قبلی که در رجیستر فایل برای C_{ij} بود جمع بزند.

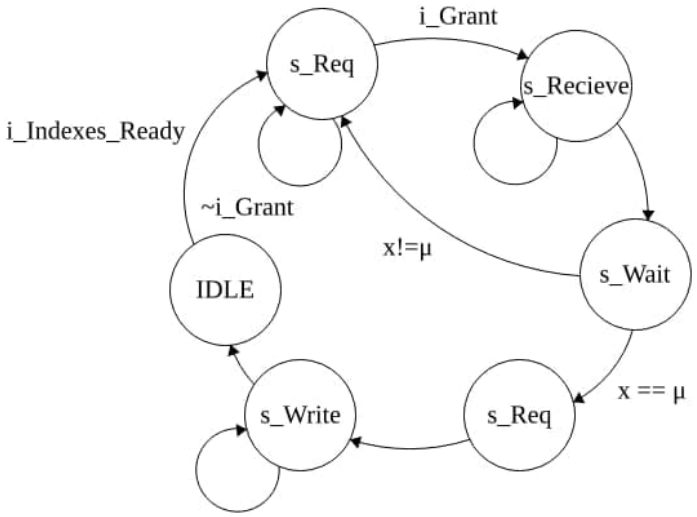
(V) بعد از اینکه عملیات‌های بالا به اندازه‌ی μ بار تکرار شد سیگنال مناسبی برای Main Control Unit ارسال کند تا در صورت نیاز Main Control Unit محاسبه‌ی بلوک دیگری را به این پراسسور اختصاص دهد.

- Control Unit



شکل ۹: CU Fsm

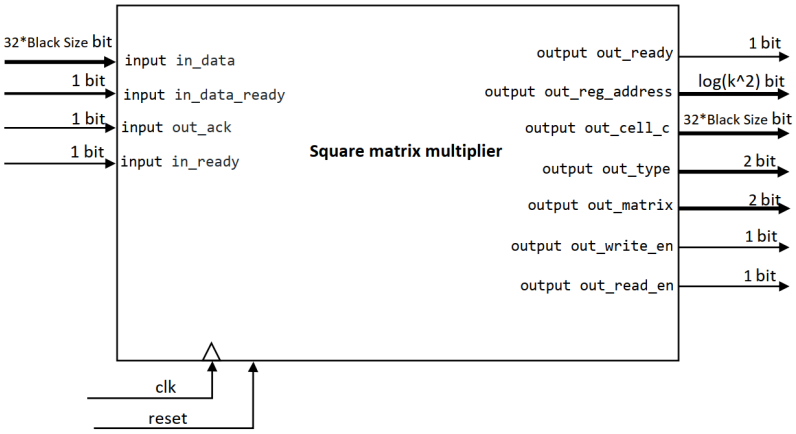
این ماژول وظیفه‌ی کنترل سیگنال‌ها هنگام محاسبه‌ی معادله ۲ را دارد. در واقع این ماژول با پیاده کردن دیاگرام حالت زیر مقدار x را تغییر می‌دهد و هر بار A_{ix}, B_{xj} جدید را از حافظه می‌خواند و سپس آن را به واحد ضرب کننده‌ی ماتریسی می‌دهد و پس از اینکه جواب نهایی حاضر شد آن را در حافظه می‌نویسد.



شکل ۱۰: CU Fsm

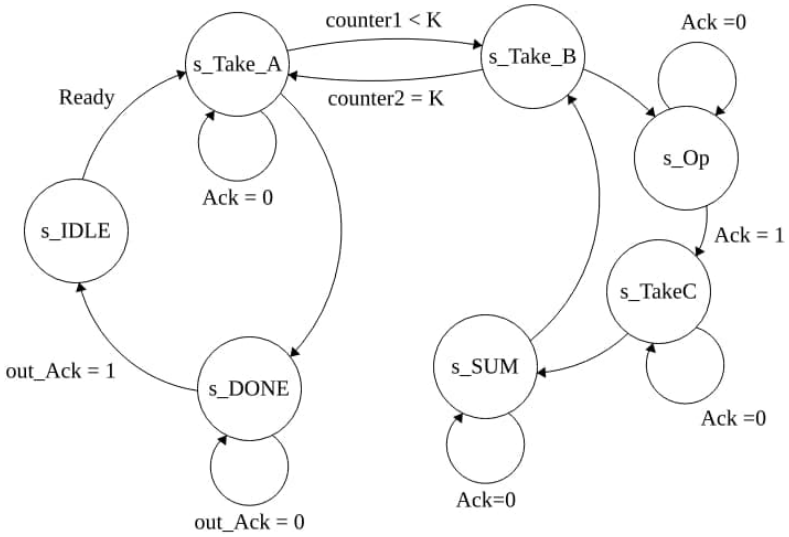
همانطور که مشخص است این واحد بعد از دریافت اندیس‌های اصلی i, j از Arbiter اجازه‌ی نوشتن در Address Bus را می‌خواهد و بعد از دریافت کردن آن به اندازه‌ای در حالت Receive می‌ماند تا A_{ix} و B_{xj} به طور کامل در رجیستر فایل نوشته شوند و این کار را به اندازه‌ی μ بار تکرار می‌کند تا نهایتا C_{ij} محاسبه شود.

- Matrix Multiplier



شکل ۱۱: Square Matrix Multiplier Schematic

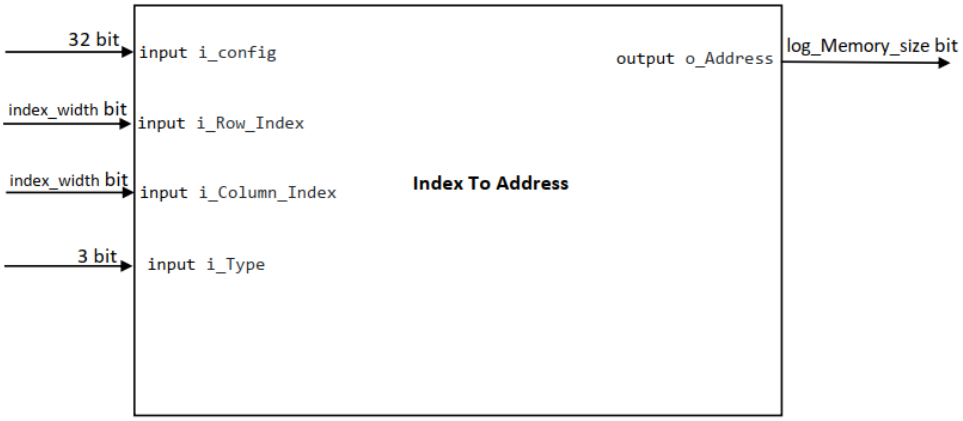
این واحد دو ماتریس $k * k$ را با توجه به نمودار حالت زیر ابتدا از رجیستر فایل می‌خواند و سپس در هم ضرب می‌کند:



شکل ۱۲: Square Matrix Multiplier Fsm

در حالت اول با دریافت بیت in_Ready که از طرف واحد کنترلی فرعی می‌آید مشخص می‌شود که باید مراحل ضرب کردن را آغاز کند. با آمدن این بیت در دو حالت Take A وTake B می‌ماند تا یک سطر و یک ستون را در یافت کند سپس در مرحله‌ی Operation این سطر و ستون را به واحدهای کوچک‌تری که مربوط به ضرب سطر و ستون می‌باشند میدهد و این کار را تا زمانی که تمام درایه‌ها را محاسبه کند ادامه می‌دهد. و نهایتا خروجی را در مرحله‌ی Done درون رجستر فایل می‌ریزد.

- Index To Address Transformer

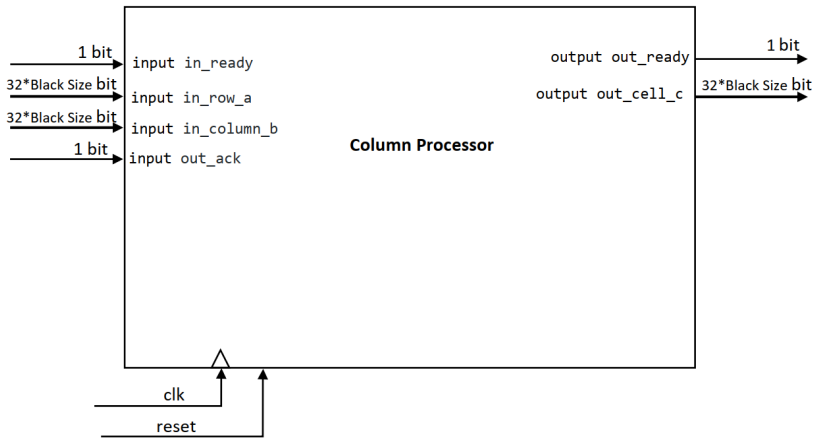


شکل ۱۳: Index To Address Transformer Schematic

این واحد با داشتن کانفیگ و همچنین ورودی‌های مشخص کننده‌ی دیگر باید بتوانند آدرس A_{ix} یا B_{xj} و یا C_{ij} را پیدا کند بیت‌های ورودی این واحد شامل اندیس سطر و ستون و ۳ بیت دیگر که مشخص می‌کند باید آدرس کدام یک از A, B, C را پیدا کند.

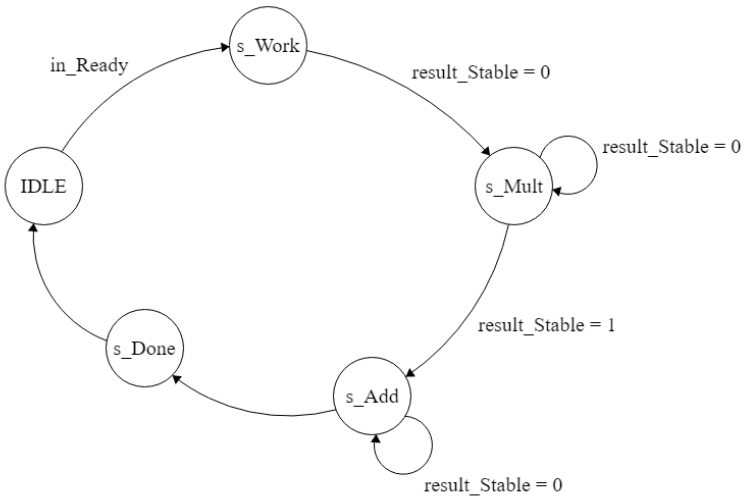
روشن است که عملکرد این واحد به قرار داد اولیه‌ای که برای حافظه و کانفیگ گذاشتیم به شدت وابسته خواهد شد و می‌توان آن را تنها با توجه به حالت‌های مختلف برای i_Type محاسبه کرد.

- Column Processor



شکل ۱۴: Column Processor Schematic

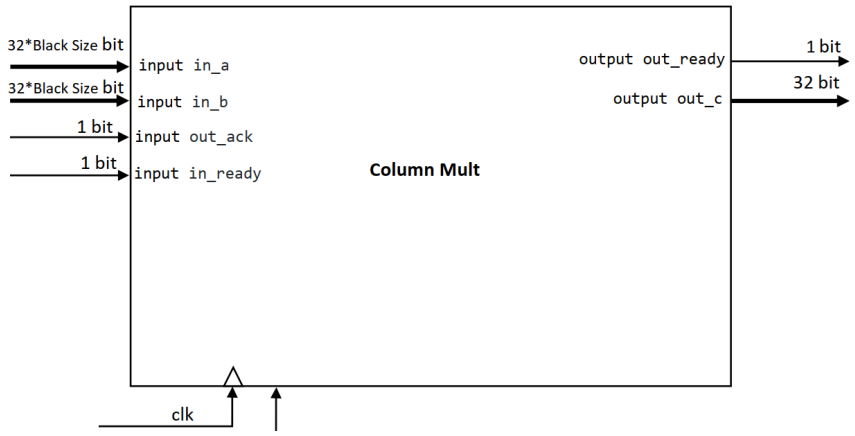
این واحد به منظور محاسبه‌ی ضرب یک سطر از ماتریس A_{ix} در یک ستون از ماتریس B_{xj} طراحی شده است.



شکل ۱۵: Column Processor Fsm

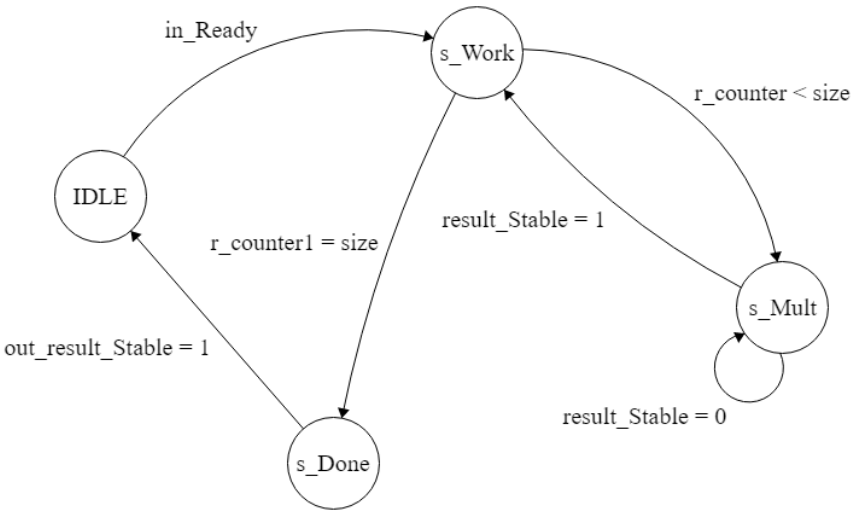
هر موقع واحد Square Matrix Multiplier داده‌های این واحد را فراهم کند بیت in_Ready را برایش فعال می‌کند سپس این واحد نیز داده‌ها را ابتدا در اختیار واحد ضرب کننده‌ی سطری می‌دهد و بعد از آماده شدن پاسخ آن داده‌ها را در اختیار واحد جمع کننده قرار می‌دهد و در نهایت این داده‌ها را در محل مناسبی در Register File می‌ریزد.

- Column Multiplier



شکل ۱۶: Column Multiplier Schematic

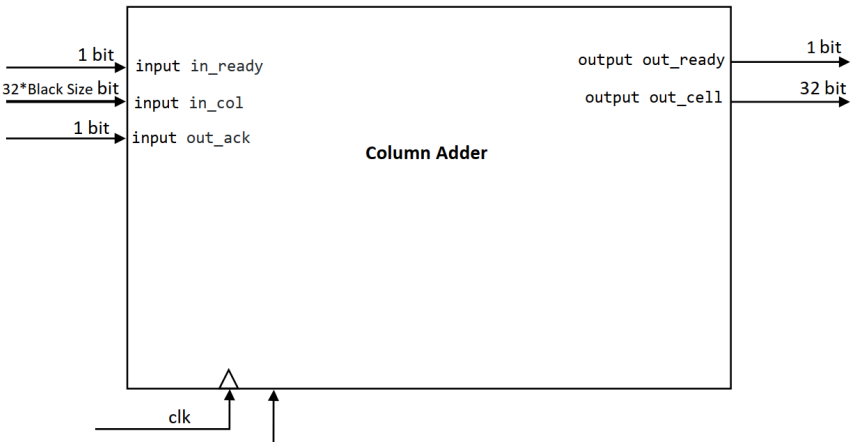
وظیفه‌ی این واحد جمع کرد این است که حاصل ضرب نقطه‌ای یک سطر از A_{ix} و یک ستون از B_{xj} را محاسبه کرده و این مقدار را نگه داشته تا column_processors بتواند از آن استفاده کند.



شکل ۱۷: Column Multiplier Fsm

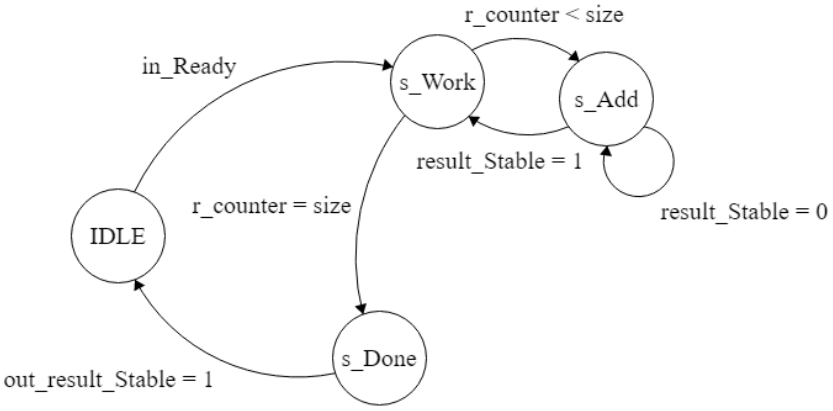
همانطور که از نمودار حالت مشخص است این واحد تا زمانی که r_Counter از k کوچک‌تر باشد ضرب کردن را تکرار می‌کند و بعد از آن به حالت s_Done می‌رود.

- Column Adder



شکل ۱۸: Column Adder Schematic

این واحد وظیفه‌ی جمع کردن یک ستون در ورودی را دارد.

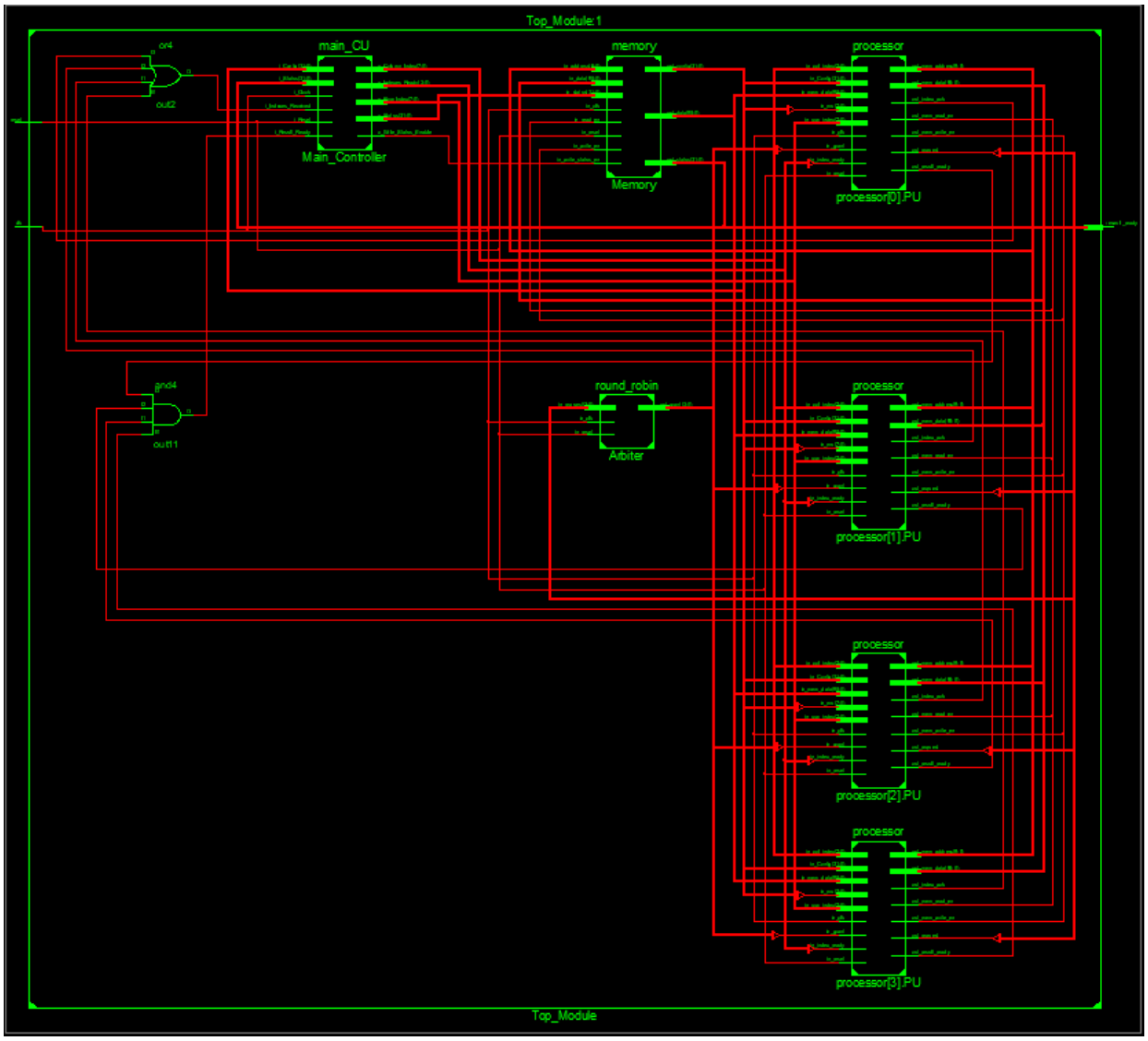


شکل ۱۹: Column Adder Fsm

همانطور که می‌توان از این نمودار حالت دید نحوه‌ی عملکرد این واحد به گونه‌ایست که با دریافت سیگنال in_ready در حالت جمع کردن بلاک‌ها قرار می‌گیرد و این کار را تا زمانی که r_counter از k کمتر باشد ادامه می‌دهد.

۴.۲ شماتیک کلی سخت‌افزار

در این بخش بعد از سنتز به وسیله‌ی XilinX شماتیک کلی سخت افزار را استخراج کردیم که به صورت زیر می‌باشد.



شکل ۲۰: شماتیک نهایی Top Module

۳ روند شبیه‌سازی و نتایج حاصل

۱.۳ توصیف TestBench

برای تمامی واحدهای طراحی شده در این پروژه، تست‌بنچ در سطح واحد در نظر گرفته شده است تا بتوانیم از صحت عملکرد تمامی ماژول‌ها اطمینان حاصل کنیم، در اینجا به توصیف تست‌بنچ اصلی می‌پردازیم که Top Module را مورد تست قرار می‌دهد و عملکرد آن را بررسی می‌کند. در واقع این تست‌بنچ تست Integrated می‌باشد.از آنجایی که سخت‌افزار طراحی شده تنها به کلاک و ریست احتیاج دارد به جز این دو که باید از نوع reg تعریف شوند بقیه‌ی netها از نوع wire خواهند بود.

برای بررسی عملکرد سخت‌افزار یک فایل memory_init.txt در پوشه‌ی test وجود دارد که به حافظه مقادیر اولیه مناسبی می‌دهد. برای تولید این مقادیر اولیه با کمک کد پایتون MemoryInit.py ورودی رندم تولید می‌کنیم که البته Config و Status در آن به درستی تنظیم شده اند.

۲.۳ توصیف روند کلی شبیه‌سازی

درون تست‌بنچ به خانه‌ی Status حافظه درون یک Always Block گوش می‌دهیم و هرگاه بیت ابتدایی آن یک شد یعنی خروجی سخت‌افزار حاضر می‌باشد. بنابراین روند کلی شبیه‌سازی به این صورت خواهد بود که ابتدا درون تست‌بنچ پارامترها را با مقدارهای مناسب مقدار دهی می‌کنیم سپس حافظه را مقداردهی اولیه می‌کنیم و بعد از آن منتظر می‌مانیم که سخت‌افزار بیت Status را تغییر دهد. نهایتا می‌توانیم با استفاده از Memory Visual موجود در نرم‌افزار Modelsim خروجی‌های برنامه را مشاهده کنیم.

۳.۳ توصیف Golden Model

در مدل طلایی برای ضرب و جمع اعشاری IEEE۷۵۴ از سایت http://weitz.de/ieee استفاده کردیم. برای ضرب ماترسی ابتدا دو ماتریس که درایه‌هایشان به طور تصادفی از اعداد بین دو عدد $0x3A83126F = 0.001$ و $0x42C80000 = 100.0$ انتخاب می‌شوند درست کردیم و سپس با الگوریتم معمولی ضرب ماتریسی و ماژول‌های ضرب و جمع اعشاری این دو ماتریس را در هم ضرب کردیم. در نهایت مقادیر دو ماتریس ورودی در فایل memory_tb_init.txt قرار می‌دهیم و ماتریس جواب را در فایل processor_tb_check.txt را بارگذاری می‌کنیم تا با مقادیر تولید شده توسط سخت افزار مقایسه کنیم.

۴.۳ مقایسه‌ی خروجی‌های نهایی با Golden Model

در پوشه‌ی tests تست‌های متعددی وجود دارد در اینجا خروجی یکی از آنها را بررسی می‌کنیم:

```
01020202
80000000
427b16bf
41634983
39de16c8
3a123639
42b29233
3cdd5c9a
3d191360
3936f467
39a95300
3be8e51a
3d8675b1
00000000
3def51f0
38f4a5d8
00000000
3e292484
4062cdb5
00000000
3d290e88
3d3891aa
3b1ff03a
39eb155b
41c85036
39326407
00000000
00000000
00000000
40c76460
409a4d67
```

شکل ۲۱: چند خط ابتدایی یکی از تست‌بنچ‌ها

این تست بنچ برای تعداد پردازنده مساوی با ۴، و بلاک‌های ۳ در ۳ طراحی شده است و همچنین سایز ماتریس‌هایی که می‌خواهند در هم ضرب شوند ۵ در ۵ است. و بنابراین می‌توانیم ببینیم که کانفیگ به درستی تعیین شده است.

خروجی حاصل از اجرای تست‌بنچ با پارامترهای متناسب و سایز حافظه ۱۰۲۴ به صورت زیر خواهد بود:

```
720 00000000
719 00000000
718 00000000
717 00000000
716 00000000
715 3f978ddc
714 3f2714ca
713 00000000
712 40c7a9cc
711 40f037a0
710 00000000
709 00000000
708 00000000
707 424142f6
706 43833f01
705 3da233bc
704 439254be
703 40fd3a9a
702 40233121
701 00000000
700 3e660b3f
699 40374b8b
698 00000000
697 408a7426
696 3ffe908a
695 00000000
694 41304aba
693 3ef7d5fa
692 43566e80
691 409828e6
690 3d9d1e92
689 42fcb644
688 446a1e6b
687 3eaa1571
686 42393ee1
685 4316b7db
684 415b16f6
683 00000000
682 00000000
681 00000000
680 00000000
```

شکل ۲۲: خروجی نهایی سخت افزار

که باید توجه داشت که آدرس‌های انتهایی بالاتر از آدرس‌های ابتدایی قرار دارند که این برعکس چیزیست که در خروجی مدل طلایی می‌بینیم.

```
41db16f6
4396b7db
42b93ee2
3f2a1571
44ea1e6b
437cb644
3e1d1e92
411828e6
43d66e80
3f77d5fa
41b04aba
00000000
407e908a
410a7426
00000000
40b74b8b
3ee60b3f
00000000
40a33121
417d3a9a
441254be
3e2233bc
44033f01
42c142f6
00000000
00000000
00000000
417037a0
4147a9cc
00000000
3fa714ca
40178ddc
00000000
00000000
00000000
00000000
```

شکل ۲۳: خروجی مدل طلایی

علارغم تفاوت‌هایی که در دقت محاسبات اعشاری وجود دارد(گلدن مدل با پایتون حساب می‌شود و تفاوت‌های پیاده‌سازی ضرب اعشاری وجود دارد) می‌توانیم ببینیم که به جز در تعداد اندکی از بیت‌ها باهم همخوانی دارند.

۴ پیاده‌سازی و نتایج حاصل

با استفاده از نرم افزار ISE سنتز انجام گرفت که نتایج آن در ادامه گزارش آمده است. به کمک سنتز موفق تعداد ثبات های استفاده شده، تعداد هاLUT و حداقل تعداد کلاک به طوری که سیستم دچار مشکلات زمان بندی نشود را به دست می آید. خلاصه‌ی نتایج سنتز به صورت زیر است:

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	45564	184304	24%
Number of Slice LUTs	139840	92152	151%
Number of fully used LUT-FF pairs	43125	142279	30%
Number of bonded IOBs	3	338	0%
Number of BUFG/BUFGCTRLs	1	16	6%
Number of DSP48A1s	28	180	15%

شکل ۲۴: خلاصه‌ی نتایج به دست آمده از سنتز

به کمک جدول، پی می بریم که تعداد فلیپ فلاپ های استفاده شده، ۴۵۵۶۴ می باشد که ۲۴ درصد ظرفیت سیستم را استفاده کرده است. از طرفی، تعداد LUT ها ۱۳۹۸۴۰ تا می باشند.که این ۱۵۱ درصد ظرفیت سیستم است. دلیل زیاد بودن تعداد LUTها استفاده از مموری به صورت غیر از sramاست که این روش بهینه نمی باشد. همچنین به همین دلیل، امکان پیاده‌سازی نمی‌باشد و در مرحله Map، به مشکل بر خورد می کنیم.

#LUT	45564
#FF	139840

پس از سنتز موفق، Timing Summary و فایل گزارش سنتز آماده می شود که برای این سیستم،Timing Summary به صورت زیر می باشد.

```
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 26.732ns (Maximum Frequency: 37.409MHz)
   Minimum input arrival time before clock: 13.795ns
   Maximum output required time after clock: 3.732ns
   Maximum combinational path delay: No path found

=========================================================================

Process "Synthesize - XST" completed successfully
```

Find: synthesized Next Previous Options

Console Errors Warnings Find in Files Results

شکل ۲۵: خلاصه‌ی نتیجه‌ی تایمینگ

حداقل پریود برابر با 26.732 نانوثانیه است می باشد که باعث می شود حداکثر فرکانس سیستم به

مقدار 37.409 MHz برسد.

Minimum period	26.732ns
Maximum Frequency	37.409MHz

از طریق فایل گزارش سنتز، می توان تعداد کلاک‌های سیستم، تعداد سطح‌های منطقی مدار و Critical Path یا تاخیر مدار را در قسمت Timing Report ملاحظه کرد.

این سیستم، ۲۲ سطح منطقی دارد. همان طور که قبلا اشاره شد، به دلیل استفاده از Memory تعداد Fanout های زیادی شکل می گیرد و همین باعث می شود تاخیر مدار زیاد شود.

```
2651  ================================================================================
2652  Timing Report
2653
2654  NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
2655        FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
2656        GENERATED AFTER PLACE-and-ROUTE.
2657
2658  Clock Information:
2659  ------------------
2660  -----------------------------------+------------------------+-------+
2661  Clock Signal                       | Clock buffer(FF name)  | Load  |
2662  -----------------------------------+------------------------+-------+
2663  clk                                | BUFGP                  | 45580 |
2664  -----------------------------------+------------------------+-------+
2665
2666  Asynchronous Control Signals Information:
2667  ----------------------------------------
2668  No asynchronous control signals found in this design
2669
2670  Timing Summary:
2671  ---------------
2672  Speed Grade: -3
2673
2674     Minimum period: 26.732ns (Maximum Frequency: 37.409MHz)
2675     Minimum input arrival time before clock: 13.795ns
2676     Maximum output required time after clock: 3.732ns
2677     Maximum combinational path delay: No path found
2678
2679  Timing Details:
2680  ---------------
2681  All values displayed in nanoseconds (ns)
2682
2683  ================================================================================
2684  Timing constraint: Default period analysis for Clock 'clk'
2685    Clock period: 26.732ns (frequency: 37.409MHz)
2686    Total number of paths / destination ports: 10371196561141 / 62877
2687  --------------------------------------------------------------------------------
```

شکل ۲۶: گزارش سنتز

```
2683  ==========================================================================
2684  Timing constraint: Default period analysis for Clock 'clk'
2685    Clock period: 26.732ns (frequency: 37.409MHz)
2686    Total number of paths / destination ports: 10371196561141 / 62877
2687  --------------------------------------------------------------------------
2688  Delay:               26.732ns (Levels of Logic = 22)
2689    Source:            processor[0].PU/control_unit/r_x_5 (FF)
2690    Destination:       Memory/out_data_33 (FF)
2691    Source Clock:      clk rising
2692    Destination Clock: clk rising
2693
2694    Data Path: processor[0].PU/control_unit/r_x_5 to Memory/out_data_33
2695                                  Gate     Net
2696      Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
2697      ----------------------------------------  ------------
2698       FDCE:C->Q             7   0.447   1.002  processor[0].PU/control_unit/r_x_5
2699       LUT4:I1->O            2   0.205   0.616  processor[0].PU/control_unit/mux13
2700       DSP48A1:B5->M9        2   3.364   0.616  processor[0].PU/control_unit/index
2701       DSP48A1:C9->P0        1   2.687   0.580  processor[0].PU/control_unit/index
2702       LUT5:I4->O            1   0.205   0.000  processor[0].PU/control_unit/index
2703       MUXCY:S->O            1   0.172   0.000  processor[0].PU/control_unit/index
2704       XORCY:CI->O           2   0.180   0.617  processor[0].PU/control_unit/index
2705       LUT2:I1->O            1   0.205   0.000  processor[0].PU/control_unit/index
2706       XORCY:LI->O           2   0.136   0.617  processor[0].PU/control_unit/index
2707       LUT2:I1->O            1   0.205   0.000  processor[0].PU/control_unit/index
2708       XORCY:LI->O           1   0.136   0.580  processor[0].PU/control_unit/index
2709       LUT3:I2->O            1   0.205   0.000  processor[0].PU/control_unit/index
2710       XORCY:LI->O           1   0.136   0.580  processor[0].PU/control_unit/index
2711       LUT5:I4->O            1   0.205   0.827  Memory_Address<1>LogicTrst15 (Memo
2712       LUT6:I2->O        17505   0.203   6.167  Memory_Address<1>LogicTrst16 (Mem
2713       begin scope: 'Memory:in_address<1>'
2714       LUT2:I1->O           20   0.205   1.093  Madd_n8642_xor<1>11_204 (Madd_n864
2715       LUT6:I5->O            1   0.205   0.827  Mmux_GND_327_o_memory[1023][31]_wi
2716       LUT6:I2->O            1   0.203   0.827  Mmux_GND_327_o_memory[1023][31]_wi
2717       LUT6:I2->O            1   0.203   0.808  Mmux_GND_327_o_memory[1023][31]_wi
2718       LUT6:I3->O            1   0.205   0.827  Mmux_GND_327_o_memory[1023][31]_wi
2719       LUT6:I2->O            1   0.203   0.000  Mmux_n8645592_F (N200962)
2720       MUXF7:I0->O           1   0.131   0.000  Mmux_n8645592 (n8645<62>)
2721       FD:D                      0.102          out_data_62
2722      ----------------------------------------
2723      Total                     26.732ns (10.148ns logic, 16.584ns route)
2724                                         (38.0% logic, 62.0% route)
2725
2726  ==========================================================================
```

شکل ۲۷: ادامه‌ی گزارش سنتز