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، جواد هزاره، پویا یوسفی

۲۳ تیر ۱۴۰۰

فهرست مطالب

شرح وظایف

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:

$$A^* = \left[\begin{array}{cccc|c} A_{00} & A_{01} & \dots & A_{0\mu-1} & \\ \vdots & \vdots & \ddots & \vdots & 0 \\ A_{\lambda-10} & A_{\lambda-11} & \dots & A_{\lambda\mu-1} & \\ \hline \multicolumn{4}{c|}{0} & 0 \end{array}\right]$$

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

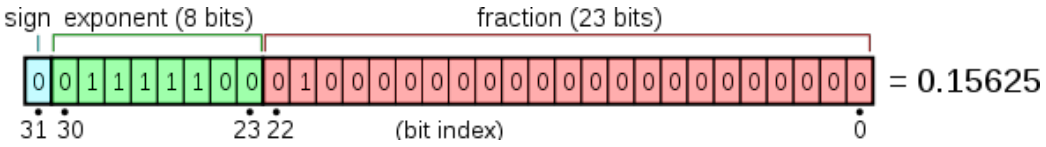
از آنجایی که هر ضرب کننده‌ی ماتریسی در حدود k^3 کلاک سایکل زمان می‌برد و محاسبه‌ی هر بلوک C_{ij} با توجه به معادله ۲ به μ بار به ضرب ماتریسی نیاز دارد. همچنین برای محاسبه‌ی تمام بلوک‌ها باید $\lambda\gamma$ بار محاسبات بالا را انجام دهیم با این حال اگر فرض کنیم که تعداد پردازنده‌ها p باشد آنگاه می‌توانیم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

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

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 sign ، fraction ، exponent مشخص می‌شوند که سهم هر یک از آنها مانند مثال زیر است:



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

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

۵.۱ کاربردها

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:

θ	μ	γ	λ
$\overbrace{\quad}^{8\ bits}$	$\overbrace{\quad}^{8\ bits}$	$\overbrace{\quad}^{8\ bits}$	$\overbrace{\quad}^{8\ bits}$

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

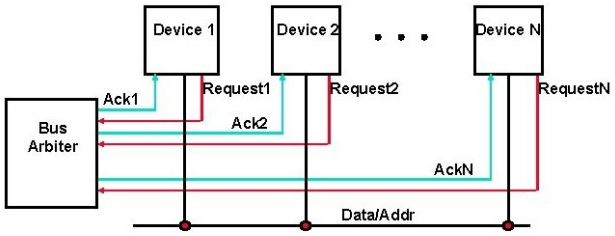
همچنین دومین خانه‌ی حافظه که مربوط به Status می‌باشد مطابق شکل زیر می‌باشد.

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.

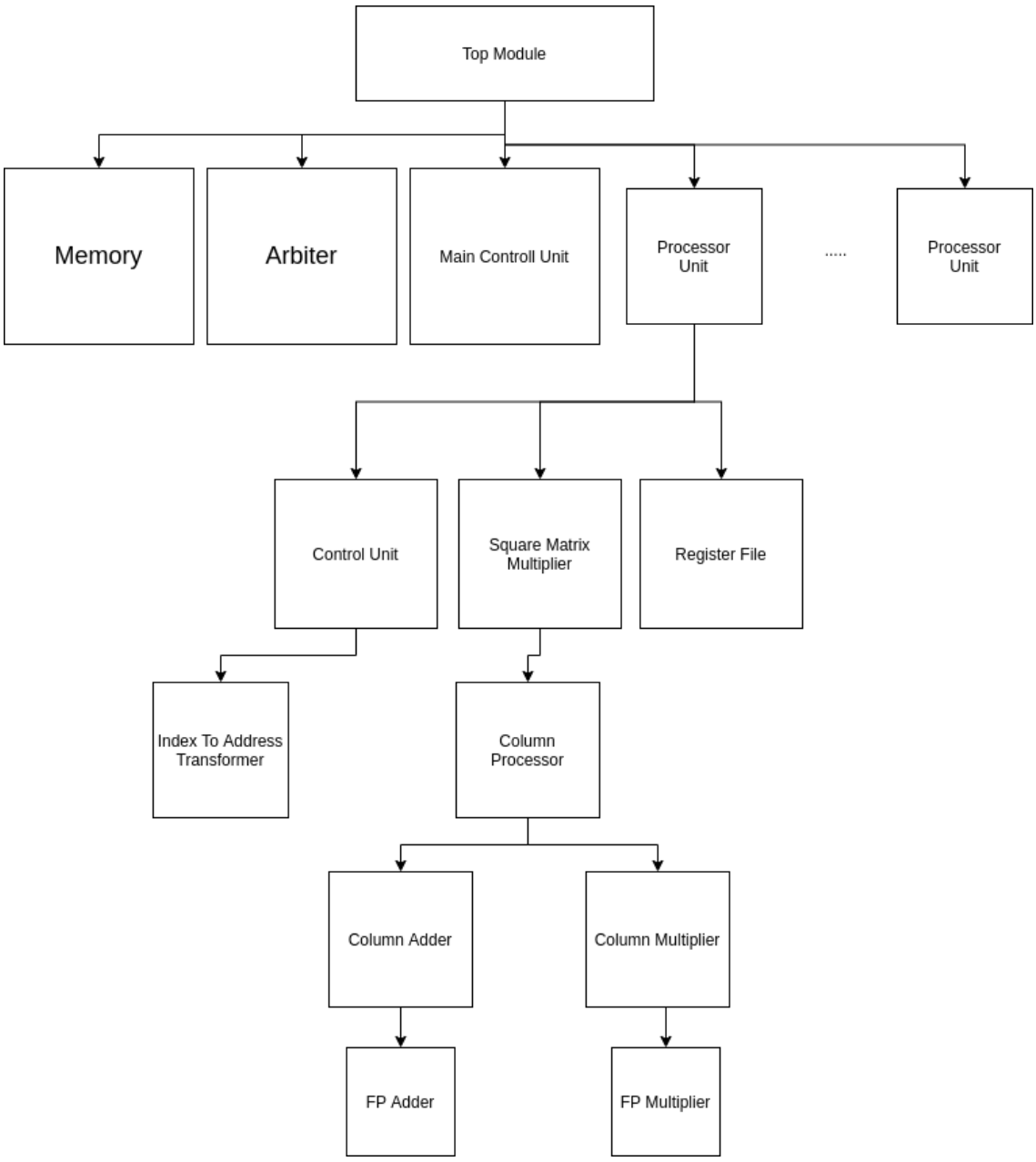
۶.۱.۲ کلاک سخت‌افزار

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.

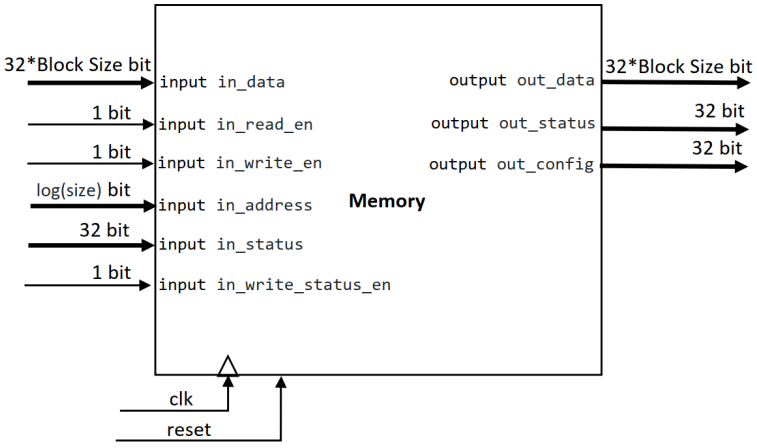
۲.۲ ساختار درختی سیستم



شکل ۲: Design Hierarchy

۳.۲ توصیف ماژول‌ها

- Memory



شکل ۳: Memory Schematic

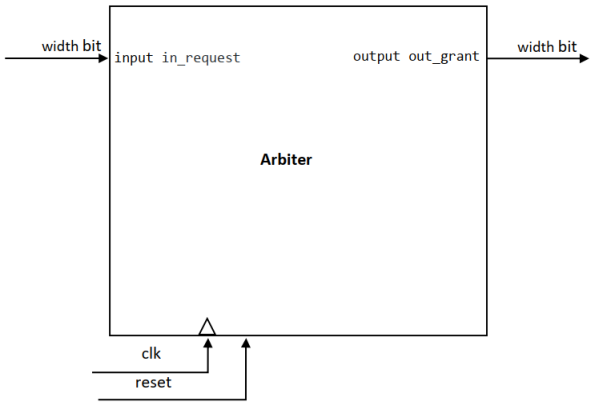
واحد حافظه سخت‌افزار که مطابق با استانداردی که در بخش‌های قبل مفسراً درباره‌ی آن توضیح دادیم می‌باشد.این واحد توانایی آدرس دهی به هر کلمه را دارد(Word Addressable). هر کلمه‌ی آن یک عدد ممیز شناور با استاندار IEEE 754 - Single Precision می‌باشد.

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.

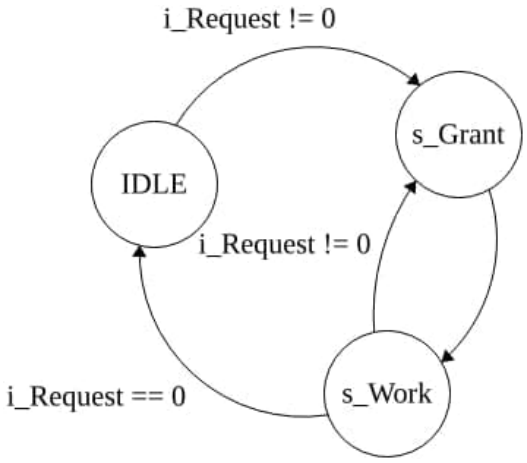
از آنجایی که Config و Status مکررا مورد نیاز ماژول‌ها در برنامه قرار می‌گیرد تصمیم گرفتیم که خواندن و نوشتن این دو (البته فقط نوشتن در Status) بدون نیاز داشتن به اجازه‌ی Arbiter و توسط Main Controller که در ادامه آن را توضیح می‌دهیم صورت بگیرد.

- Arbiter



شکل ۴: Arbiter Schematic

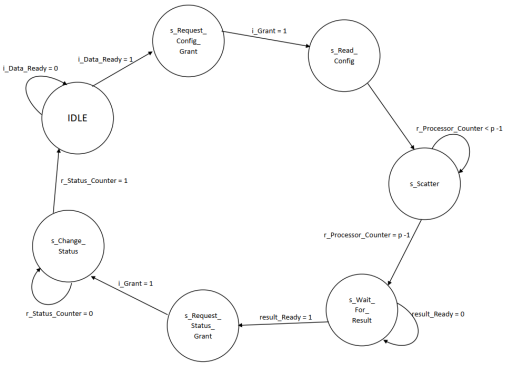
همانطور که برای Memory توضیح دادیم این واحد نقش پخش کردن اجازه‌ی دسترسی به مموری را بین ماژول‌ها دارد در شکل ۱ این موضوع مشخص است. مقدار پهنای ورودی و خروجی این ماژول متناسب با تعداد ماژول‌های دیگریست که اجازه‌ی دسترسی به حافظه را می‌خواهند. برای روشن شدن عملکرد این ماژول به FSM زیر توجه کنید:



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



شکل ۶: Main CU Fsm

همان طور که از این نمودار حالت مشخص است هرگاه کلمه‌ی status در حافظه نشان دهنده Cpu_Ready باشد از حالت اولیه خارج می‌شویم و از Arbiter درخواست می‌کنیم که حافظه را در اختیار ما بگذارد، سپس با داشتن کانفیگ می‌توانیم بین Processorهای مختلف اندیس‌ها را پخش کنیم. این کار به اندازه‌ی θبار انجام می‌دهیم تا نهایتا همه‌ی C_{ij}ها توسط پردازنده‌ها محاسبه شده و در مموری ذخیره شود. سپس با درخواست از Arbiter دسترسی به حافظه را بدست می‌آوریم و کلمه‌ی status را تغییر می‌دهیم تا CPU متوجه به پایان رسیدن عملیات شود.

- Processor

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. شماتیک این ماژول به صورت زیر می‌باشد.

شماتیییک

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 ممووری 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.

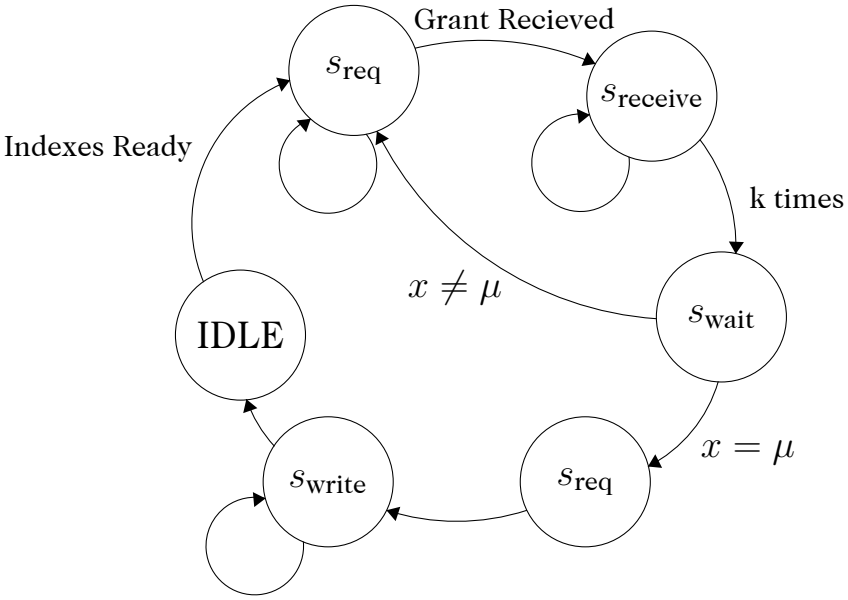
(IV) بعد از اتمام ضرب حاصل را با مقادیر قبلی که در رجیستر فایل برای C_{ij} بود جمع بزند.

(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.

یکی دیگر از وظیفه‌های Processor این است که در صورتی که Grant را در اختیار نداشت به وسیله‌ی یک Tri-state Buffer خروجی‌های این مجموعه که به سمت حافظه می‌رود را z کند.

```
assign out_mem_data = (in_grant) ? reg_out_data : 'bz;
assign out_mem_write_en = (in_grant) ? cu_mem_write : 1'bz;
assign out_mem_read_en = (in_grant) ? cu_mem_read : 1'bz;
assign out_mem_address = (in_grant) ? cu_mem_address : 'bz;
```

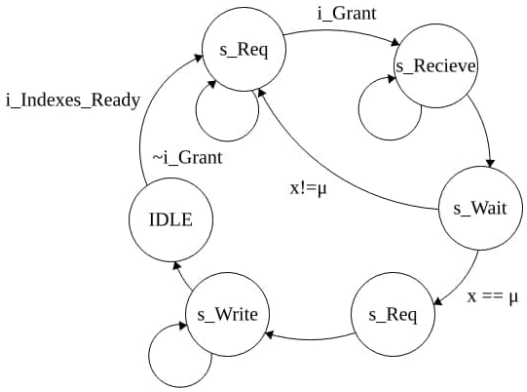
- Control Unit



این ماژول وظیفه‌ی محاسبه‌ی حاصل جمع زیر را دارد:

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

در واقع این ماژول با پیاده کردن دیاگرام حالت زیر اندیس‌ها 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. توجه می‌کنیم که در حالت Receive به



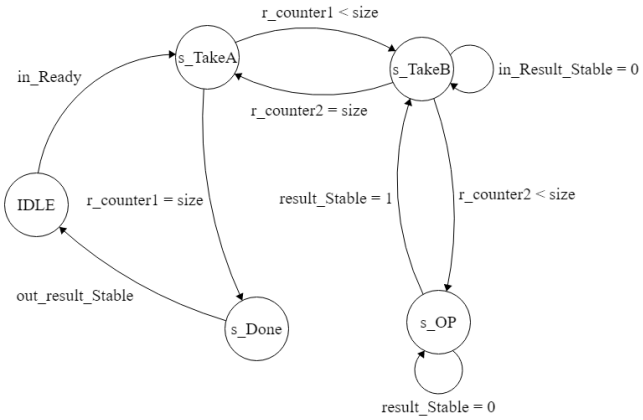
شکل ۷: CU Fsm

اندازه‌ی $2k^2$ باید صبر کنیم تا تمام بیت‌های مورد نیازمان نوشته شود همچنین این حالت به دو حالت درونی Receive A و Receive B تقسیم بندی می‌شود.

- Processor Unit: این واحد متشکل از Control Unit و Matrix Multiplier و Register File می‌باشد و وظیفه‌ی برقراری ارتباط بین آنها را دارد. به طور خلاصه این واحد دستورات کنترل

یونیت را به مموری می‌فرستد و داده‌ها را درون رجیسترفایل می‌ریزد و یا از آن می‌خواند و Matrix Multiplier با استفاده از داده‌های موجود در رجیستر فایل ضرب ماتریسی را انجام می‌دهد و نهایتا خروجی را درون رجیستر فایل می‌ریزد. سپس با استفاده از دستورات واحد کنترلی مقادیر موجود در رجیستر فایل به حافظه انتقال پیدا می‌کند.

- Matrix Multiplier: این واحد وظیفه‌ی ضرب ماتریسی دو ماتریس $k * k$ را دارد به نمودار حالت زیر توجه کنید: در حالت اول با دریافت بیت in_Ready که از طرف واحد کنترلی می‌آید



شکل ۸: Multiplier Fsm

مشخص می‌شود که باید مراحل ضرب کردن را آغاز کند و این ماژول با استفاده از رجیسترهای میانی زمان پایان عملیات ضرب را متوجه می‌شود.

- Index To Address Transformer 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.

۳ روند شبیه‌سازی و نتایج حاصل

۱.۳ توصیف TestBench ها

۲.۳ توصیف روند کلی شبیه‌سازی

۳.۳ توصیف Golden Model

۴.۳ مقایسه‌ی خروجی‌های نهایی با Golden Model