

به نام خدا



طراحی سیستم‌های دیجیتال

گزارش پروژه

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

استاد:

فرشاد بهاروند

اعضای گروه:

عماد زین‌اوقلی، مازیار شمسی‌پور، بردیا محمدی

جواد هزاره، پویا یوسفی

نیم سال دوم ۹۹-۰۰

فهرست

۲	شرح وظایف
۳	مقدمه
۳	تعریف الگوریتم
۴	قراردادهای ریاضی
۴	نحوه‌ی عملکرد از نظر مساحت و تایمینگ
۵	استاندارد IEEE 754
۵	مراجع مورد استفاده
۶	توصیف معماری سیستم
۶	اینترفیس‌های سیستم و قرارداد استفاده از آن
۸	کلاک سخت‌افزار
۸	دیاگرام‌های بلوکی سخت‌افزار
۹	توصیف ماژول‌ها
۱۳	ساختار درختی سیستم
۱۴	روند شبیه‌سازی و نتایج حاصل
۱۴	توصیف TestBench‌ها
۱۴	توصیف روند کلی شبیه‌سازی
۱۴	توصیف Golden Model
۱۴	مقایسه‌ی خروجی‌های نهایی با Golden Model



شرح وظایف



مقدمه

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:

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

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

$$C_{ij} = \sum_{x=0}^{\mu} A_{ix} B_{xj} \quad (2)$$

با داشتن تعداد تعداد مشخصی ضرب کننده‌ی ماتریسی $k * k$ می‌توانیم این به طور موازی با استفاده از آنها حاصل نهایی $A \times B$ را محاسبه کنیم.



قراردادهای ریاضی

توجه می‌کنیم که در ادامه‌ی این گزارش و توضیحات لازمه در نظر می‌گیریم که ماتریس‌های ورودی A_{mr} و B_{rn} خواهند بود و بنابراین ماتریس خروجی به صورت $A_{mr} \times B_{rn} = C_{mn}$ خواهد بود. همچنین لازم است که توجه داشته باشید که وقتی ماتریس‌ها را به فرم بلوکی می‌نویسیم مقادیر زیر را تعریف می‌کنیم:

$$\mu = \left\lceil \frac{r}{k} \right\rceil \quad (آ۳)$$

$$\lambda = \left\lceil \frac{m}{k} \right\rceil \quad (ب۳)$$

$$\gamma = \left\lceil \frac{n}{k} \right\rceil \quad (ج۳)$$

از این نمادها به کرات 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 بخش پذیر نباشد با اضافه کردن صفر به انتهای آن باعث بخش پذیری می‌شویم.

نحوه‌ی عملکرد از نظر مساحت و تایمینگ

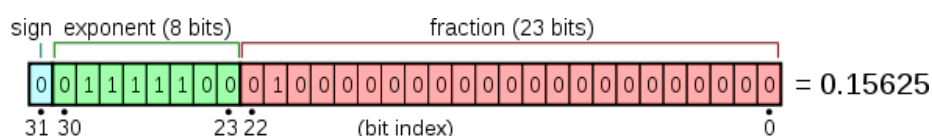
از آنجایی که هر ضرب کننده‌ی ماتریسی در حدود k^3 کلاک سایکل زمان می‌برد و محاسبه‌ی هر بلوک C_{ij} با توجه به معادله ۲ به μ بار به ضرب ماتریسی نیاز دارد. همچنین برای محاسبه‌ی تمام بلوک‌ها باید $\lambda\gamma$ بار محاسبات بالا را انجام دهیم با این حال اگر فرض کنیم که تعداد پردازنده‌ها p باشد آنگاه می‌توانیم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

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



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:

$$\text{value} = (-1)^{\text{sign}} \times 2^{(E-127)} \times (1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}) \quad (5)$$

مراجع مورد استفاده

References

- [1] Abhishek Kumar : Scalability of Parallel Algorithms for Matrix Multiplication
- [2] Patricia Ortega : Parallel Algorithm for Dense Matrix Multiplication
- [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



توصیف معماری سیستم

اینترفیس‌های سیستم و قرارداد استفاده از آن

به طور کلی سخت‌افزار از یک حافظه و بخش محاسبه‌ی ضرب ماتریسی تشکیل شده است که پردازنده می‌تواند ورودی‌ها را درون حافظه قرار داده و خروجی‌ها را نیز از آن بخواند. (I/O Map). با این حال قراردادهایی در نحوه‌ی استفاده از مموری وجود دارد که باید به آن توجه شود. ساختار کلی حافظه به صورت زیر خواهد بود:

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

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. برای مثال اگر ماتریس A به صورت زیر باشد:

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

و در صورتی که $k = 2$ و به عبارتی بلوک‌ها $2 * 2$ باشند CPU باید آن را به صورت زیر در حافظه قرار دهد:



Config
Status
۱
۲
۴
۵
۳
۰
۶
۰
⋮

به عبارتی وظیفه‌ی بلوک کردن ماتریس و همچنین صفر قرار دادن خانه‌های اضافی به عهده‌ی CPU خواهد بود. همچنین CPU باید اولین خانه‌ی حافظه را که مربوط به کانفیگ می‌باشد به صورت زیر از اعداد پر کند:

θ	μ	γ	λ
----------	-------	----------	-----------

که هر کدام ۸ بیت خواهند بود و مقادیر این پارامترها در معادله ۳ مشخص شده است و البته باید توجه داشتید که مقدار θ نیز از رابطه‌ی زیر محاسبه می‌شود:

$$\theta = \frac{\lambda \gamma}{\# \text{Matrix Processors}} \quad (۶)$$

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

MP Ready	CPU Acknowledge	...	MP Acknowledge	CPU 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 ضرب کننده‌ی ماتریسی دریافت کرد به کارش ادامه دهد بعد از تمام شدن عملیات ماتریسی بیت MP Ready فعال می‌شود و CPU می‌تواند بلاک‌های ماتریس خروجی را از مکانی که در مموری مربوط به خروجی‌ها می‌باشد استخراج کند.

با این تفاسیر تنها ورودی لازم به سخت‌افزار ریست آسنکرون می‌باشد. و با استفاده از مموری سخت‌افزار می‌تواند ورودی‌ها را از حافظه خوانده و آنها را محاسبه کند.



کلاک سخت‌افزار

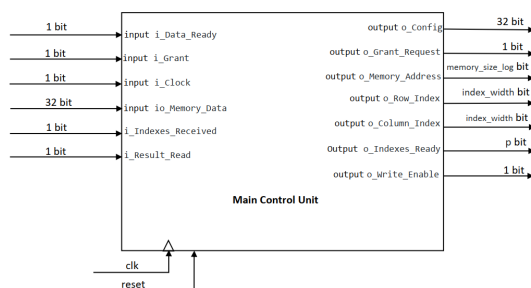
تمامی ماژول‌های این سخت‌افزار از جمله مموری و تمام ماژول‌های واحد حساب‌کننده‌ی ضرب ماتریسی به صورت سنکرون عمل می‌کنند و CDC در این سخت‌افزار اتفاق نمی‌افتد.

دیاگرام‌های بلوکی سخت‌افزار

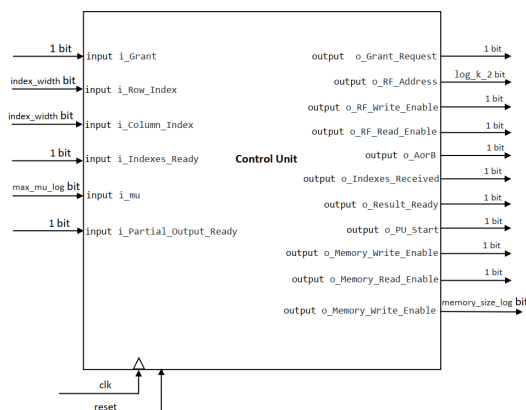
این سخت‌افزار شامل ماژول‌های زیر می‌باشد:

- Memory
- Arbiter
- Main Control Unit
- Control Unit
- Processor Unit
- Matrix Multiplier
- Matrix Adder
- Index To Address Transformer

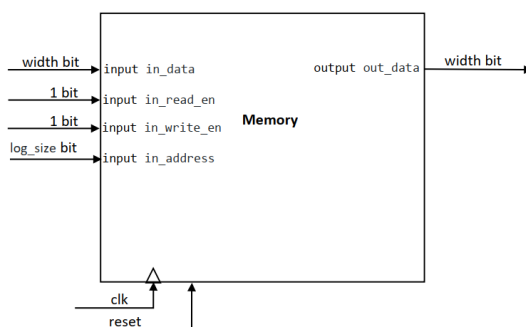
در اینجا به توصیف ورودی خروجی هر یک از آن‌ها و نحوه‌ی اتصال آنها می‌پردازیم و در بخش بعد نحوه‌ی عملکرد هر یک را توضیح می‌دهیم:



شکل ۱: Main Control Unit



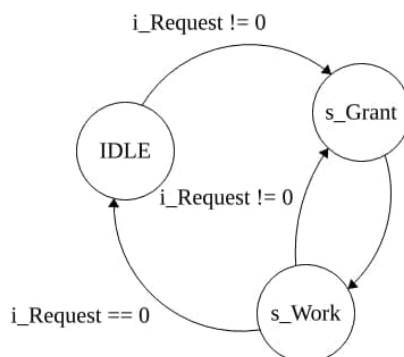
شکل ۲: Control Unit



شکل ۳: Memory

توصیف ماژول‌ها

- **Memory:** واحد مموری سخت‌افزار که مطابق با استاندارد ی که ابتدا آورده شده است. این واحد شامل یک باس خروجی داده است که ماژول‌ها می‌توانند از آن استفاده کنند. همچنین دارای یک باس ورودی و باس آدرس می‌باشد که Arbiter تعیین می‌کند که کدام ماژول حق استفاده از این باس‌ها و همچنین حق استفاده از enable‌های خواندن و نوشتن را دارد.
- **Arbiter:** این واحد نقش پخش کردن اجازه‌ی دسترسی به مموری را بین ماژول‌ها دارد، به FSM زیر توجه کنید:
نحوه‌ی عملکرد این ماژول به این صورت است که یک صف بااهمیت از ماژول‌هایی



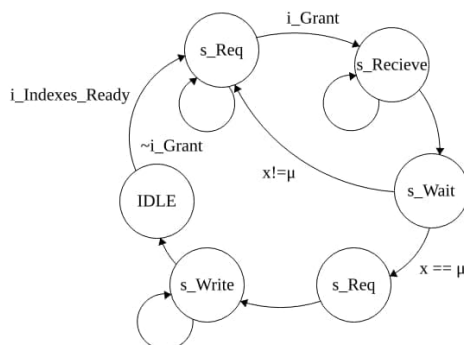
شکل ۴: Arbiter Fsm

که آن وصل هستند را نگه می‌دارم و در صورتی که ورودی Request آن یک باشد به بااهمیت‌ترین ماژول متصل به خود اجازه‌ی دسترسی به حافظه را می‌دهد. سپس آن ماژول می‌تواند از مموری استفاده کند.

- 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 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



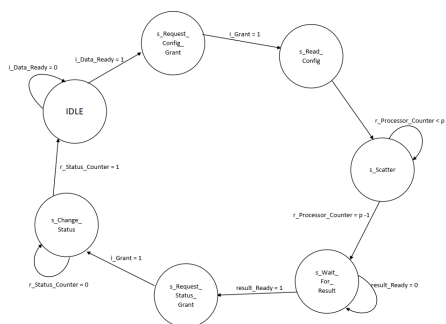
شکل ۵: CU Fsm

که در حالت Receive به 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.

- 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

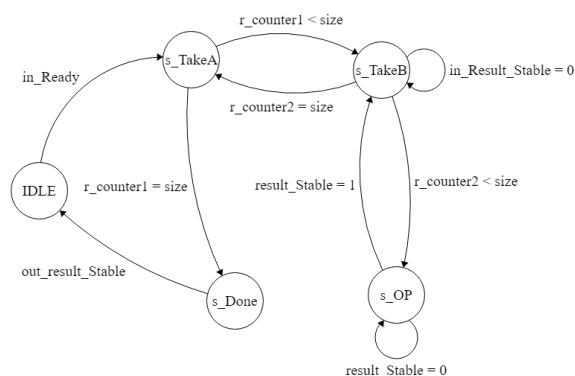
همان طور که از این نمودار حالت مشخص است هرگاه کلمه‌ی status در حافظه نشان دهنده Cpu_Ready باشد از حالت اولیه خارج می‌شویم و از Arbiter درخواست می‌کنیم که حافظه را در اختیار ما بگذارد، سپس با داشتن کانفیگ می‌توانیم بین Processorهای مختلف اندیس‌ها را پخش کنیم. این کار به اندازه‌ی θ بار انجام می‌دهیم تا نهایتاً 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 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 رجیستر فایل می‌ریزد و یا از آن می‌خواند و Matrix Multiplier با استفاده از داده‌های موجود در رجیستر فایل ضرب ماتریسی را انجام می‌دهد و نهایتاً 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



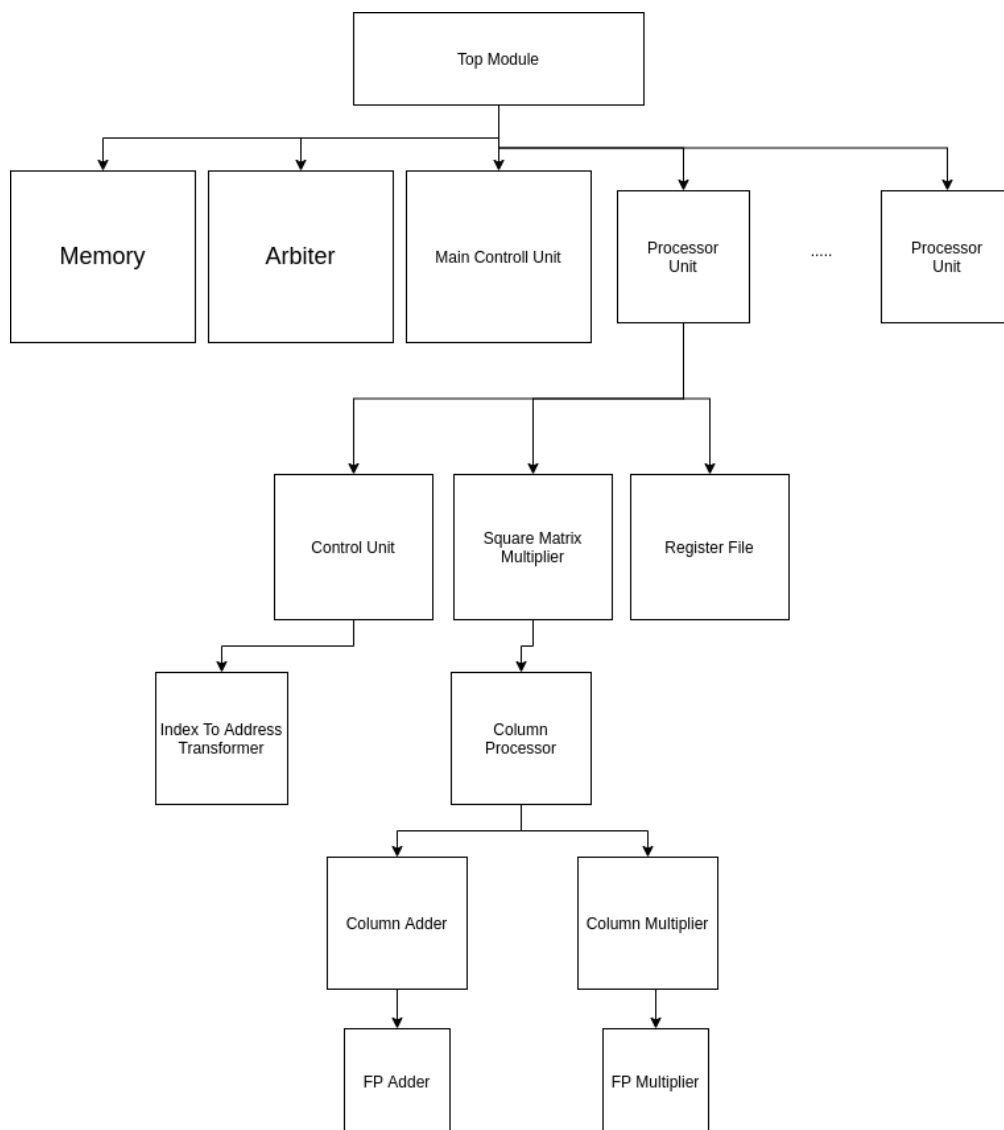
شکل ۷: Multiplier Fsm

واحد کنترلی می‌آید 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 کننده دیگر باید بتواند آدرس A_{ix} یا B_{xj} 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.



ساختار درختی سیستم



شکل ۸: Design Hierarchy



روند شبیه‌سازی و نتایج حاصل

توصیف TestBench ها

توصیف روند کلی شبیه‌سازی

توصیف Golden Model

مقایسه‌ی خروجی‌های نهایی با Golden Model