

به نام خدا



طراحی سیستم‌های دیجیتال

Multiplier Matrix Point Floating Single - Precision

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

استاد:

جناب آقای دکتر بهاروند

نام، نام خانوادگی و شماره دانشجویی اعضای گروه:

رضا عبدالله زاده - ۹۷۱۰۶۱۳۲

فاطمه خاشعی - ۹۷۱۰۵۸۹۹

عرشیا اخوان - ۹۷۱۱۰۴۲۲

مجید طاهرخانی

رضا امینی

علی بالاپور - ۹۷۱۰۱۳۲۶

فهرست مطالب

۲	۱	مقدمه
۲	۱.۱	چکیده
۲	۲.۱	تعریف و تاریخچه مختصر
۴	۳.۱	نحوه عملکرد کلی
۴	۱.۳.۱	ساختار کلی
۵	۲.۳.۱	جمع FP
۷	۳.۳.۱	ضرب FP
۸	۴.۱	کاربردها
۸	۱.۴.۱	نگاشت خطی
۹	۲.۴.۱	دستگاه معادلات خطی
۹	۳.۴.۱	3D Projection
۹	۴.۴.۱	یادگیری عمیق
۱۱	۲	توصیف معماری سیستم
۱۱	۱.۲	اینترفیس‌های سیستم و کلاک سیستم
۱۱	۱.۱.۲	پارامترها و ورودی‌ها
۱۲	۲.۱.۲	خروجی‌ها
۱۲	۳.۱.۲	کلاک سیستم
۱۳	۲.۲	دیاگرام بلوکی سخت‌افزار
۱۵	۳.۲	توصیف وظیفه ماژول‌های سخت‌افزار
۱۵	۱.۳.۲	Single Multiplier
۱۶	۲.۳.۲	Single Adder
۱۸	۳.۳.۲	Vector Inner Product
۱۹	۴.۳.۲	Matrix Multiplier
۱۹	۴.۲	ساختار درختی سیستم
۲۱	۳	روند شبیه‌سازی
۲۱	۱.۳	توصیف test bench
۲۱	۲.۳	adder test bench
۲۲	۳.۳	inner product test bench
۲۲	۴.۳	matrix multiplier test bench

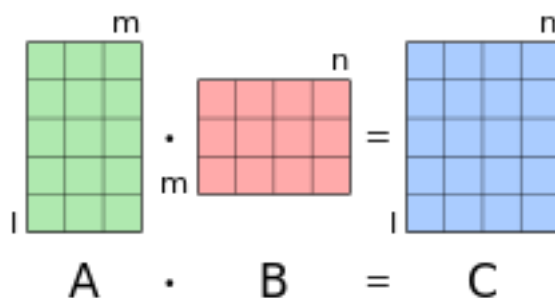
۱ مقدمه

۱.۱ چکیده

عملیات ضرب دو ماتریس با اعداد اعشاری با دقت یگانه (Single Precision Floating Point) امروزه کاربردهای بسیار زیادی در حوزه‌های مختلف علوم کامپیوتر از جمله هوش مصنوعی و یادگیری عمیق دارد. در این پروژه می‌خواهیم با استفاده از الگوریتم‌های داده شده و مدارهای ضرب و جمع Floating Point (FP)، مداری برای انجام ضرب ماتریسی ارائه دهیم.

۲.۱ تعریف و تاریخچه مختصر

عملیات ضرب ماتریسی یکی از مهمترین عملیات‌های موجود در جبر خطی و ریاضیات می‌باشد. در ضرب ماتریسی، دو ماتریس با ابعاد $n \times m$ و $m \times k$ در یکدیگر ضرب می‌شوند و یک ماتریس $n \times k$ حاصل می‌شود. شرط عملی بودن ضرب ماتریس این است که تعداد ستون‌های ماتریس اول با تعداد سطرهای ماتریس دوم برابر باشد. در صورتی که A و B دو ماتریس با شرایط گفته شده باشند، حاصل ضرب ماتریسی به صورت AB نشان داده می‌شود. [۱]

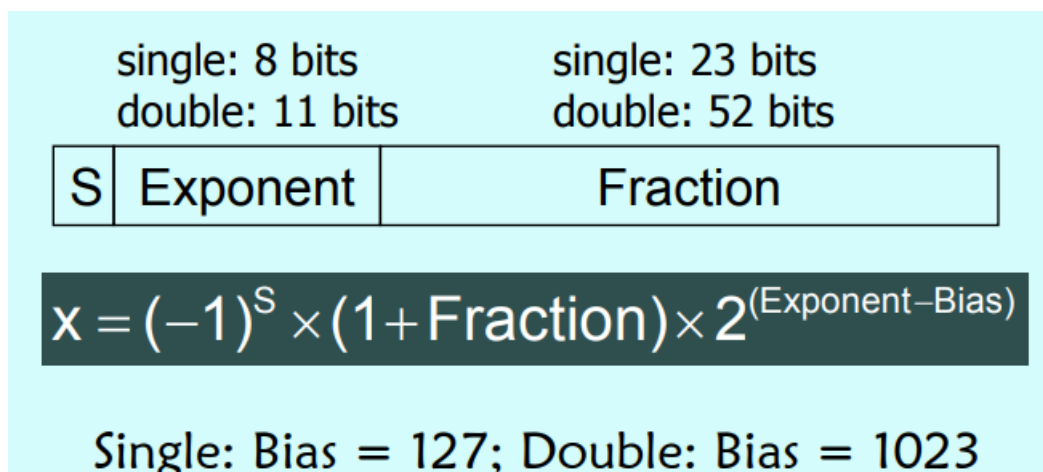


در این پروژه ما به ضرب ماتریس‌هایی می‌پردازیم که درایه‌هایشان، اعداد اعشاری ممیز شناور با دقت یگانه و یا Single Precision Floating Point می‌باشند. عملیات روی اعداد ممیز شناور (floating point) به طور گسترده‌ای در پردازش‌های علمی مورد استفاده قرار می‌گیرد. سازمان IEEE برای عملیات‌های ممیز شناور یک استاندارد کلی به نام IEEE-754 تعیین کرده است. در این پروژه، برای بدست آوردن حاصل ضرب دو ماتریس با درایه‌های اعشاری ممیز شناور، لازم است که از جمع و ضرب FP استفاده شود. در قسمت عملکرد، ضرب و جمع FP مورد بررسی واقع می‌شود. اما قبل از بررسی جمع و ضرب FP بهتر است با خود اعداد اعشاری ممیز شناور آشنا شویم.

در محاسبات عددی، از ممیز شناور، به عنوان تقریبی برای پشتیبانی از توازن بین دامنه و دقت محاسبه استفاده می‌شود. به همین دلیل، محاسبات ممیز شناور اغلب در سیستم‌هایی یافت می‌شود که اعداد بسیار کوچک و بسیار بزرگی را شامل می‌شوند که نیازمند پردازش سریع هستند. یک عدد، به طور کلی، تقریباً از تعداد ثابتی از ارقام significand و یک توان در پایه ثابت تشکیل می‌شود؛ این پایه معمولاً ۲، ۱۰ یا ۱۶ است. یعنی:

significand * base^{exponent}

منظور از ممیز شناور، این است که ممیز اعشاری عدد، می تواند شناور باشد. یعنی می توان آن را در هر قسمتی از عدد قرار داد. در نتیجه می توان ممیز شناور را نوعی نمادگذاری علمی تلقی کرد. از سیستم ممیز شناور می توان برای نشان دادن مقادیر بسیار زیاد و یا بسیار کم استفاده کرد. برای مثال می توان از این سیستم برای نمایش فاصله بین کهکشان ها یا فاصله بین عناصر موجود در هسته اتم استفاده کرد. نمایش دقیق تر این شیوه به صورت زیر می باشد:



لازم به ذکر است که ما در این پروژه، از حالت Single استفاده می کنیم.

در طول سال های متمادی، انواع مختلفی از نمایش ممیز شناور برای محاسبات عددی در کامپیوتر استفاده شده است. در سال ۱۹۸۵، استاندارد IEEE 754 برای محاسبات ممیز شناور ثبت شد. این استاندارد واگرایی شیوه های به کار رفته برای نمایش ممیز شناور را کاهش داد و امروزه اکثر کامپیوترها، از این استاندارد تبعیت می کنند. واحد اندازه گیری سرعت عملیات های ممیز شناور، FLOPS می باشد امروزه اکثر پردازنده های مدرن دارای یک واحد ممیز شناور برای انجام عملیات روی اعداد ممیز شناور می باشند. [۲]

نکته قابل توجه این است که پردازنده های نسل جدید (مخصوصا پردازنده های گرافیکی مانند سری RTX شرکت Nvidia) دارای واحد پردازشی مجزایی برای محاسبات ماتریسی می باشند. به این واحد های پردازشی، TPU یا Tensor Processing Unit گفته می شود. البته به این پردازشگرها، هسته های تنسور (Tensor Cores) هم گفته می شود. هسته های تنسور به طور خلاصه برای محاسبات ماتریسی کاربرد دارند و در زمینه هوش مصنوعی و یادگیری عمیق کاربردهای فراوانی دارند. یکی از مهمترین و کاربردی ترین وظایف چنین هسته هایی، ضرب ماتریسی می باشد. پس داشتن الگوریتم مناسب برای پیاده سازی ضرب ماتریس برای ساخت چنین واحدهای پردازشی ای، ضروری می باشد. [۳] [۴]

۳.۱ نحوه عملکرد کلی

۱.۳.۱ ساختار کلی

شرط پذیرا بودن ضرب دو ماتریس، یکسان بودن تعداد ستون‌های ماتریس اول (ماتریس سمت چپ) و تعداد سطرهای ماتریس دوم (ماتریس سمت راست) می باشد. برای مثال، دو ماتریس زیر را در نظر بگیرید.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

با توجه به این که ماتریس A دارای ابعاد $m \times n$ و ماتریس B دارای ابعاد $n \times p$ می باشد، پس ضرب این دو ماتریس پذیراست. فرض کنید حاصل ضرب ماتریسی این دو، ماتریسی به نام C می باشد که به شکل زیر است:

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

ماتریس C دارای m سطر (تعداد سطرهای ماتریس A) و دارای p ستون (تعداد ستون‌های ماتریس B) می باشد. هر یک از درایه های ماتریس C از رابطه زیر به دست می آیند.

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^n a_{ik}b_{kj},$$

به عبارت دیگر، سطر i ام ماتریس A و ستون j ام ماتریس B به عنوان یک وکتور (بردار) در نظر گرفته می شوند و حاصل ضرب داخلی این دو وکتور در درایه ij ماتریس C قرار می گیرد. در نهایت حاصل ضرب ماتریسی دو ماتریس A و B که به صورت AB نشان داده می شود، به صورت زیر است:

$$\mathbf{C} = \begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & a_{11}b_{12} + \cdots + a_{1n}b_{n2} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ a_{21}b_{11} + \cdots + a_{2n}b_{n1} & a_{21}b_{12} + \cdots + a_{2n}b_{n2} & \cdots & a_{21}b_{1p} + \cdots + a_{2n}b_{np} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & a_{m1}b_{12} + \cdots + a_{mn}b_{n2} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

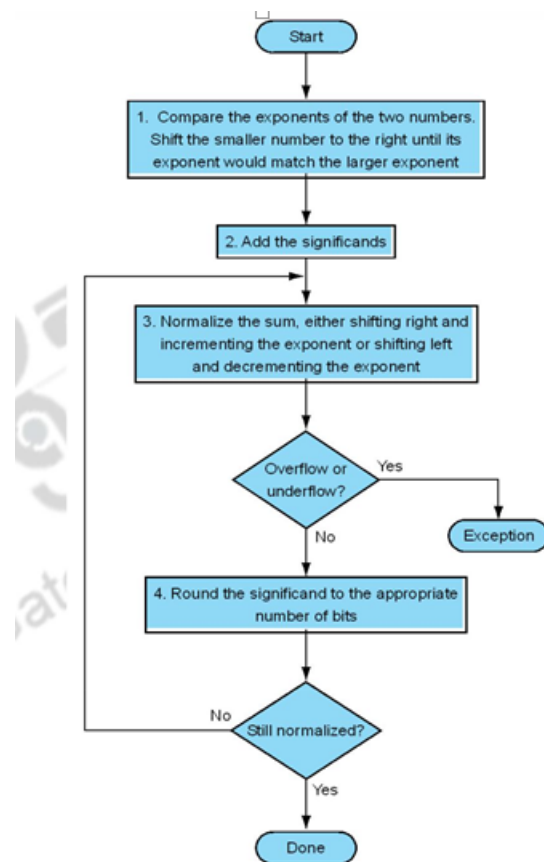
در قسمت های قبلی، اشاره شد که درایه های ماتریس، از جنس اعداد اعشاری ممیز شناور با دقت یگانه می باشند. برای محاسبه هر درایه ماتریس C ، به تعدادی ضرب و جمع بین این اعداد، نیاز هست. لازم به ذکر است که ضرب و جمع این نوع اعداد، با ضرب و جمع اعداد معمولی و عادی، تفاوت دارد. در دو بخش بعدی به بررسی ضرب و جمع FP می پردازیم.

۲.۳.۱ جمع FP

همان گونه که اشاره شد، جمع اعداد FP دشوارتر از جمع اعداد صحیح می باشد و الگوریتم خاص خود را دارد. برای جمع دو عدد a و b که به صورت ممیز شناور می باشند و $a < b$ ، داریم:

- ۱ - ابتدا توان های دو عدد را یکسان می کنیم. برای این کار، عدد کوچکتر یعنی a به سمت راست شیفت داده می شود.
- ۲ - مقادیر دو عدد با هم جمع می شوند.
- ۳ - حاصل جمع، نرمال می شود.
- ۴ - overflow و یا underflow شدن حاصل جمع را بررسی می شود.
- ۵ - حاصل جمع گرد می شود.

در صورتی که عدد نرمال نباشد، مراحل ۳ تا ۵ دوباره تکرار می شوند. رویه کلی انجام این کار در شکل زیر آمده است.



برای مثال می‌خواهیم دو عدد 8.70×10^{-1} و 9.95×10^1 را با یکدیگر جمع کنیم. ابتدا توان‌های دو عدد را باهم یکسان می‌کنیم. برای این کار، توان عدد کوچکتر را با توان عدد بزرگتر یکسان می‌کنیم. یعنی:

$$0.087 \times 10^1 + 9.95 \times 10^1$$

سپس این دو عدد را با یکدیگر جمع می‌زنیم و آن را نرمال می‌کنیم:

$$0.087 + 9.95 = 10.037 \times 10^1$$

$$10.037 \times 10^1 = 1.0037 \times 10^2$$

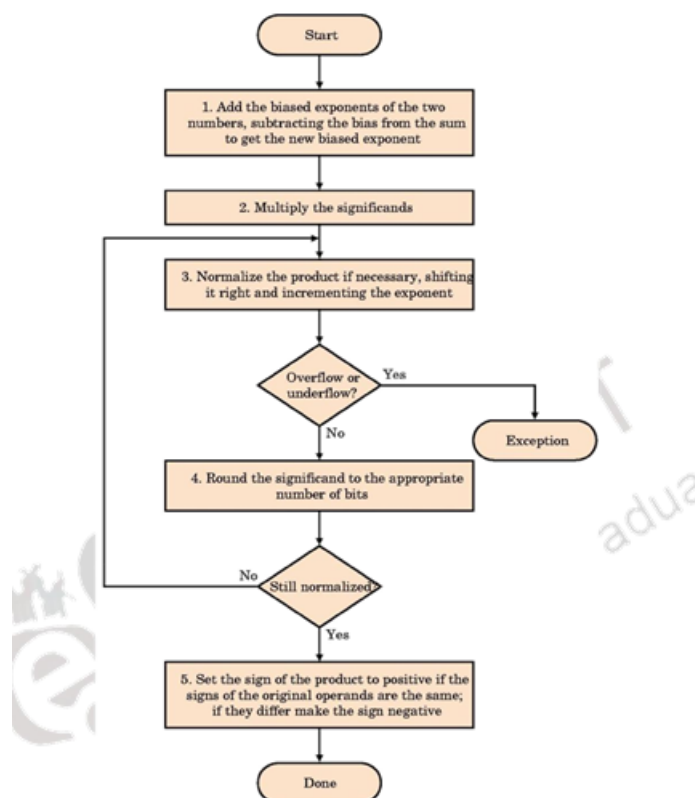
و در نهایت حاصل را گرد می‌نماییم.

$$1.004 \times 10^2$$

[۵] [۶]

۳.۳.۱ ضرب FP

- ۱ - ابتدا توان‌ها باهم جمع می‌شوند.
 - ۲ - مقادیر significand در همدیگر ضرب می‌شوند.
 - ۳ - اعداد نرمال می‌شوند و سپس overflow و یا underflow شدن عدد حاصل، بررسی می‌شود.
 - ۴ - اعداد گرد می‌شوند.
 - در صورتی عدد نرمال نباشد، مراحل ۳ و ۴ تکرار می‌شود.
 - ۵ - علامت عدد تعیین می‌شود.
- رویه کلی الگوریتم ضرب FP با دقت یگانه، در فلوچارت زیر نیز مشخص می‌باشد.



برای مثال می‌خواهیم حاصلضرب دو عدد اعشاری با ممیزشناور 1.110×10^{10} و 9.200×10^{-5} را بدست آوریم. ابتدا توان‌های هردو عدد را یکسان می‌کنیم:

$$\text{new exponent} = 10 - 5 = 5$$

سپس قسمت‌های اصلی اعداد را در یکدیگر ضرب می‌کنیم:

$$1.110 \times 9.200 = 10.21200$$

سپس حاصل را نرمال کرده و رند می‌نماییم. در نتیجه خواهیم داشت:

$$1.021 * 10^6$$

[۵] [۶]

۴.۱ کاربردها

در گذشته، از ضرب ماتریسی به عنوان روشی برای راحت‌تر و واضح‌تر کردن محاسبات جبرخطی استفاده می‌شد. اما امروزه، ضرب ماتریسی یکی از مهمترین و پایه‌ای‌ترین عملیات‌ها در علوم ریاضیات، جبرخطی، فیزیک و علوم کامپیوتر می‌باشد. در ادامه به تعدادی از کاربردهای ضرب ماتریس می‌پردازیم.

۱.۴.۱ نگاشت خطی

نگاشت خطی یا ترادیش خطی، یک تابع بین دو فضای برداری است که دو عملیات جمع برداری و ضرب نرده‌ای را باقی‌نگه می‌دارد. این تابع همچنین رابطه مستقیمی با عبارت عملگر خطی دارد که معمولاً در نگاشت‌های خطی از یک فضای برداری استفاده می‌شوند. از ضرب ماتریسی برای انجام نگاشت خطی، استفاده می‌شود. برای مثال، نگاشت خطی A ، یک ماتریس $m \times n$ می‌باشد که برداری در فضای n بعدی را به برداری در فضای m بعدی تبدیل می‌کند. داریم: [۱]

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \vdots \\ a_{31} & a_{32} & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

$$A(x) = \begin{pmatrix} a_{11}x_1 + \dots + a_{1n}x_n \\ a_{21}x_1 + \dots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + \dots + a_{mn}x_n \end{pmatrix}$$

۲.۴.۱ دستگاه معادلات خطی

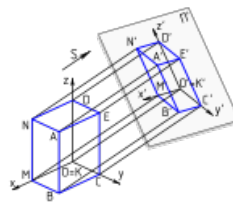
برای حل دستگاه معادلات خطی، می‌توان از ضرب ماتریسی استفاده کرد. [۱]

$$\begin{aligned} a_{11}x_1 + \dots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + \dots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + \dots + a_{mn}x_n &= b_m \end{aligned}$$

$$\mathbf{Ax} = \mathbf{b}.$$

۳.۴.۱ 3D Projection

3D Projection یا Graphical Projection تکنیک طراحی ای است که برای نشان دادن یک شی سه بعدی از روی یک شی دو بعدی (سطح) استفاده می‌شود. روش‌های مختلفی برای اجرای این تکنیک وجود دارد. یک روش که Orthographic projection نام دارد، از ضرب ماتریسی برای اجرای این تکنیک استفاده می‌کند. [۷]



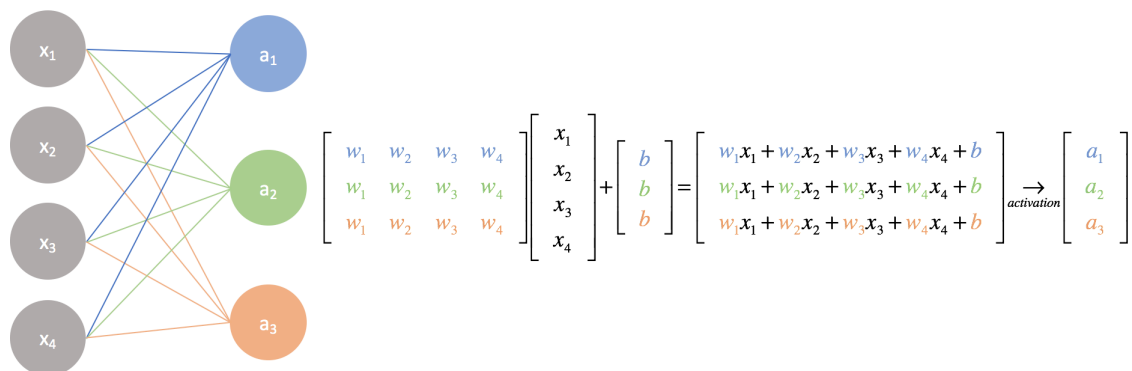
۴.۴.۱ یادگیری عمیق

ضرب ماتریس یکی از مهمترین عملیات‌های مورد استفاده در یادگیری ماشین برای آموزش شبکه‌های عصبی می‌باشد. در هنگام آموزش مدل‌های عصبی عمیق، از ضرب ماتریس برای محاسبه و بروزرسانی وزن‌ها استفاده می‌شود. همچنین در مرحله تست نیز، برای بدست آوردن پاسخ نهایی، در هر لایه از عملیات ضرب ماتریسی استفاده می‌شود. بنابراین، در صورتی که بتوانیم مدار سریع‌تر و بهینه‌تری برای عملیات ضرب ماتریس داشته باشیم، می‌توانیم از الگوریتم‌های بهینه‌تری برای آموزش و تست شبکه‌های عصبی داشته باشیم و در نهایت مدل نهایی سریع‌تری خواهیم داشت. پس اهمیت بهینه بودن الگوریتم و مدار ضرب کننده ماتریس، در هوش مصنوعی و یادگیری عمیق، بسیار زیاد است.

Input layer

Output layer

A simple neural network



۲ توصیف معماری سیستم

۱.۲ اینترفیس‌های سیستم و کلاک سیستم

سیستم به طور کلی از ۴ ماژول اصلی تشکیل شده است که این ماژول‌ها، به صورت تو در تو، از یکدیگر استفاده می‌کنند. هر ماژول واحد کنترل جداگانه خود را دارد و صرفاً بین ماژول‌ها، سیگنال‌های کنترلی‌ای که مربوط به ماژول‌های دیگر است، رد و بدل می‌شود. این چهار ماژول اصلی به صورت زیر می‌باشند:

- matrix_multiplier
- inner_product
- adder
- multiplier

ماژول matrix_multiplier به نوعی ماژول اصلی سیستم ما می‌باشد و وظیفه دارد تا دو ورودی در قالب ماتریس را دریافت کرده و حاصل ضرب ماتریسی آن را محاسبه کند. برای این کار، به ازای هر سطر و ستون، از یک ماژول inner_product استفاده می‌شود تا حاصل هر درایه، مشخص شود. همچنین در هر ماژول inner_product، از تعدادی ماژول adder و multiplier استفاده شده است که عمل جمع و ضرب FP را انجام می‌دهند.

۱.۱.۲ پارامترها و ورودی‌ها

در ماژول matrix_multiplier پارامترهای m و p و n و $word_width$ به عنوان پارامترهای ورودی، به ماژول داده می‌شوند. این پارامترها به ترتیب نشان دهنده تعداد سطر ماتریس اول، تعداد ستون‌های ماتریس اول و سطرهای ماتریس دوم، تعداد ستون‌های ماتریس دوم و طول عدد هر درایه می‌باشند. همچنین به ترتیب، مقادیر پیش فرض $۱۶ - ۱۶ - ۱۶ - ۳۲$ به این پارامترها داده شده است. البته لازم به ذکر است، همان گونه که در مقدمه بحث شد، برای این که ضرب دو ماتریس پذیرا باشد، باید حتماً تعداد ستون‌های ماتریس اول با تعداد سطرهای ماتریس دوم برابر باشد.

ورودی‌های اصلی این ماژول، دو ماتریس $matrix_A$ و $matrix_B$ و سیگنال $start$ و clk می‌باشند. با توجه به این که نمی‌توان یک آرایه را به عنوان ورودی، به ماژول داد، پس هر کدام از ماتریس‌ها را به یک وکتور با سایز $m * p * word_width$ (برای ماتریس اول) و با سایز $p * n * word_width$ (برای ماتریس دوم) به عنوان ورودی می‌گیریم. یعنی در هر ماتریس همه سطرها را پشت سرهم قرار داده و آن‌ها را به صورت بیت به بیت ذخیره می‌کنیم. همچنین می‌دانیم هر عدد ۳۲ بیت (یا به اندازه $word_width$) فضا را اشغال می‌کند. پس به راحتی می‌توان به هر عدد دسترسی داشت. سیگنال $start$ هم تعیین کننده شروع کار کل مدار می‌باشد. همچنین سیگنال clk نیز کلاک کلی مدار می‌باشد.

در ماژول inner_product، پارامتر $number_of_element$ نشان دهنده تعداد اعضای وکتور‌ها می‌باشد.

باشد. در این ماژول ورودی‌های $In1$, $In2$, clk , rst , $start$ به ماژول داده می‌شود. $In1$ و $In2$ دو وکتور با سائزهای برابر ($element_length * number_of_elements$) می‌باشند که مقادیر دو بردار (به طور دقیق تر سطر ماتریس اول و ستون ماتریس دوم) در آن‌های ذخیره شده است. هدف این ماژول ضرب داخلی این دو بردار در هم می‌باشد. همچنین سیگنال‌های rst به معنای $reset$ ماژول $inner_product$ ، clk به معنای کلاک کلی مدار و $start$ به معنای شروع فرآیند ضرب داخلی، به عنوان ورودی به این ماژول داده شده است. ورودی‌ها و خروجی‌های دو ماژول $multiplier$ و $adder$ با یکدیگر یکسان می‌باشند. ورودی‌های این دو ماژول عبارتند از: $input_a$, $input_b$, $input_a_stb$, $input_b_stb$, $output_z_ack$, clk , rst . دو ورودی $input_a$ و $input_b$ همان عملوندهای اصلی ما می‌باشند. ورودی‌های stb نشان دهنده این موضوع اند که آیا دو عملوند اصلی یعنی $input_a$ و $input_b$ آماده عملیات هستند یا خیر. در صورتی که هر دو سیگنال stb یک شوند، در این صورت عملیات ضرب یا جمع شروع می‌شود. ورودی rst و clk نیز واضح می‌باشند.

۲.۱.۲ خروجی‌ها

در ماژول $matrix_multiplier$ یکی از خروجی‌ها، ماتریس خروجی یا همان $matrix_C$ خواهد بود. این ماتریس، حاصل ضرب ماتریسی دو ماتریس ورودی $matrix_A$ و $matrix_B$ می‌باشد. همان گونه که در قسمت قبل نیز اشاره شد، با توجه به این که نمی‌توان آرایه در یک ماژول خروجی داد، پس مانند قسمت قبل عمل کرده و هر سطر را پشت سرهم قرار در یک وکتور با سائز $m*n*word_width$ قرار می‌دهیم. این ماژول دو سیگنال خروجی با نام‌های $done$ و $ready$ نیز دارد که به ترتیب نشان دهنده اتمام عملیات و آماده بودن ماژول برای دریافت ورودی می‌باشد.

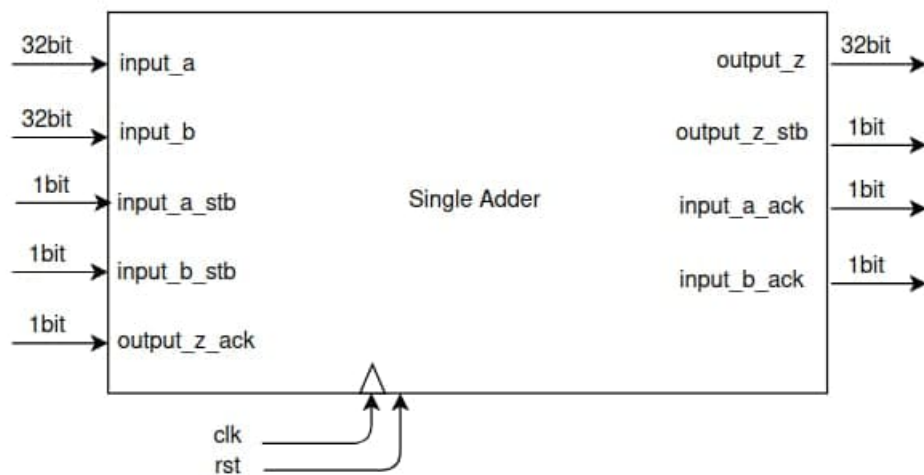
در ماژول $inner_product$ خروجی out و $done$ را داریم. $done$ نشان دهنده تمام شدن عملیات ضرب داخلی می‌باشد. out نیز یک عدد ۳۲ بیتی (به طور پیش فرض) می‌باشد که نشان دهنده حاصل ضرب داخلی است.

در ماژول‌های $adder$ و $multiplier$ نیز خروجی‌های $input_a_ack$, $output_z$, $output_z_stb$ را داریم. $input_b_ack$ را داریم. $output_z$ همان خروجی جمع و یا ضرب می‌باشد که سائز آن به طور پیش فرض برابر ۳۲ بیت است. $output_z_stb$ نیز اتمام عملیات ضرب و جمع را نشان می‌دهد.

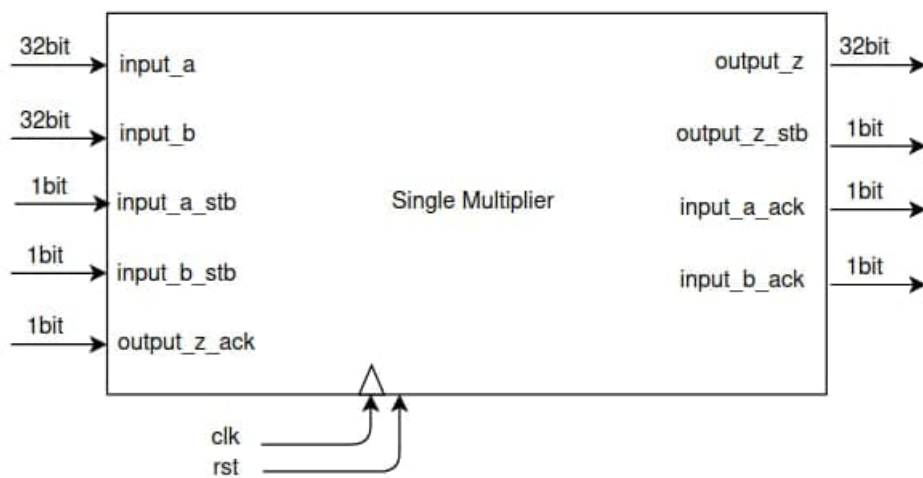
۳.۱.۲ کلاک سیستم

همان گونه که در قسمت‌های بالا نیز اشاره شد، کلاک استفاده شده در هریک از ماژول‌ها، همان کلاک اصلی سیستم می‌باشد.

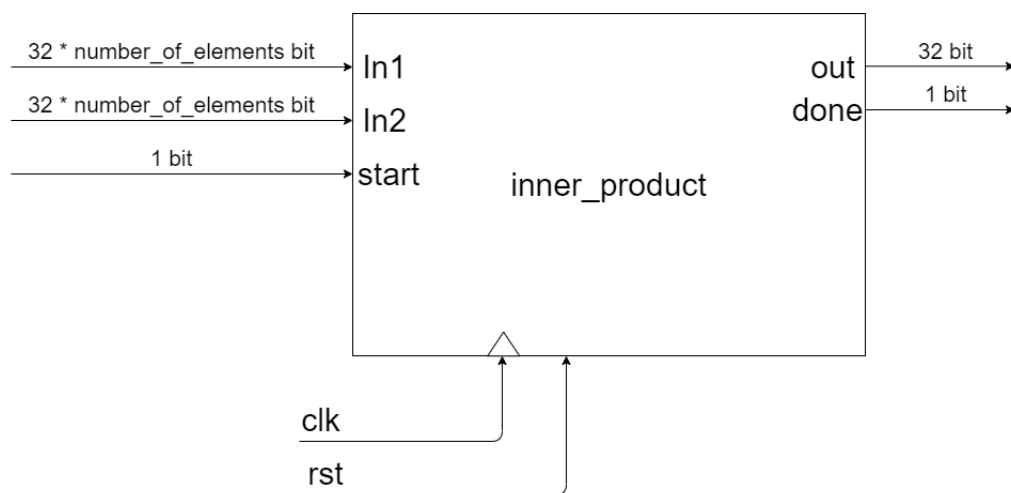
۲.۲ دیاگرام بلوکی سخت‌افزار



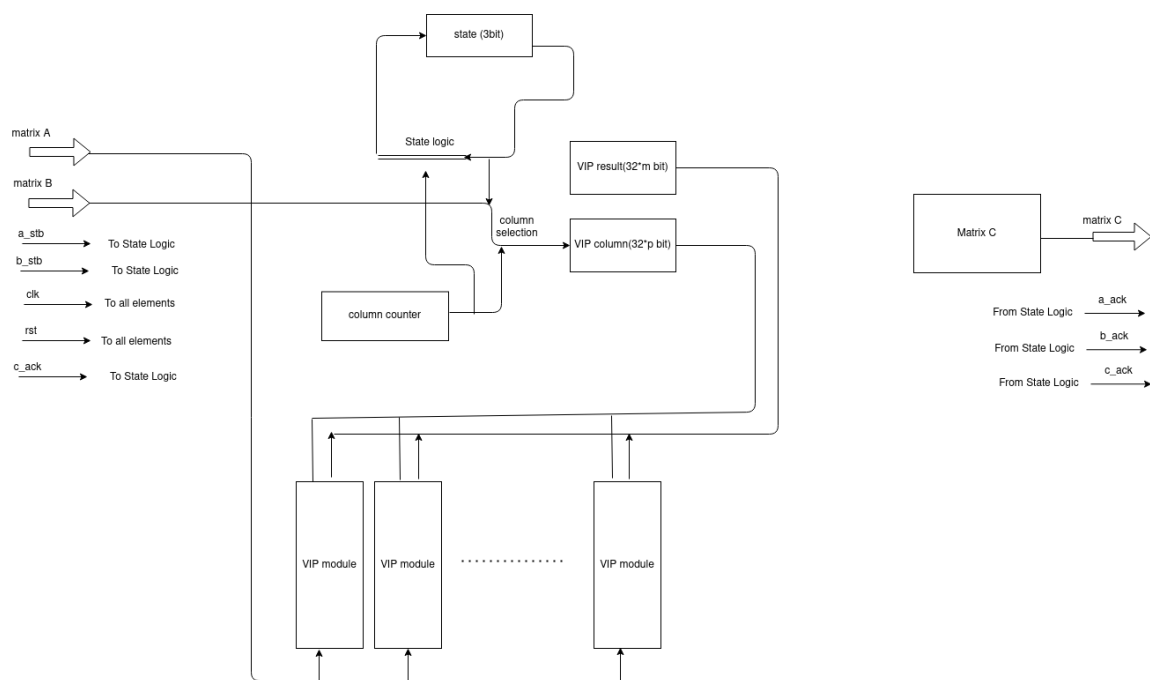
دیاگرام ماژول جمع‌کننده



دیاگرام ماژول ضرب‌کننده



دیاگرام ماژول ضرب داخلی



دیاگرام کلی ماژول ضرب کننده دو ماتریس

۳.۲ توصیف وظیفه ماژول‌های سخت‌افزار

۱.۳.۲ Single Multiplier

همان‌گونه که در قسمت قبل اشاره شد، این ماژول دو عدد ۳۲ بیتی ممیز شناور را به عنوان ورودی دریافت می‌کند و حاصلضرب این دو عدد را محاسبه می‌نماید. دو عددی ورودی a و b نام دارند و خروجی z نام دارد. در ابتدای ماژول، استیت‌های مختلف به عنوان پارامتر مشخص شده اند و تعدادی متغیر برای ذخیره سازی قسمت‌های مختلف اعداد (برای مثال توان‌های اعداد) در نظر گرفته شده است. استیت‌های ماژول به گونه زیر می‌باشند (در هر قسمت، عملیات انجام شده در هر استیت به طور خلاصه توصیف شده است):

- `get_a`

در این استیت، ماژول آماده است که ورودی اول از ماژول `top` گرفته شود و در داخل متغیر a قرار داده شود. پس از دریافت ورودی اول، سیگنال `ack` به ماژول بالاتر فرستاده می‌شود و ماژول ضرب کننده به استیت دریافت ورودی دوم می‌رود.

- `get_b`

این استیت مشابه استیت بالا می‌باشد. پس از دریافت ورودی دوم و ارسال سیگنال `ack`، ماژول به استیت `unpack` می‌رود.

- `unpack`

در این استیت، همان‌گونه که از اسم آن مشخص است، اعداد ورودی به چند قسمت تقسیم می‌شوند تا فرآیند ضرب عدد اعشاری، تسهیل شود. به ازای هر عدد اعشاری ورودی، ۲۳ بیت کم ارزش به عنوان `mantissa` جدا می‌شود. ۸ بیت بعدی هر عدد نیز جدا شده و عدد ۱۲۷ از آن کم می‌شود تا توان عدد یا `exponent` محاسبه و در متغیر مخصوص خود، ذخیره شود. و در نهایت، بیت پرارزش نیز به عنوان علامت عدد در نظر گرفته شده و در متغیر مخصوص خود، ذخیره می‌شود. لازم به ذکر است که همه این عملیات‌ها، بر روی هر دو عدد انجام می‌شود.

- `special_cases`

در این استیت، موارد خاصی که ممکن است به عنوان عدد ورودی، به ماژول داده شود، بررسی می‌گردد و اقدامات لازم برای ضرب چنین موارد خاصی، اتخاذ می‌گردد. برای مثال در این استیت چک می‌شود که آیا یکی از اعداد ورودی مقدار بینهایت دارد یا خیر. در صورت داشتن چنین مقداری، خروجی باید مقدار بینهایت باشد.

- `normalise_a`

همان‌گونه که در مقدمه نیز ذکر شده است، فرآیند ضرب دو عدد اعشاری با فرمت ممیز شناور، با ضرب اعداد صحیح تفاوت دارد. در این استیت و استیت بعد، دو عدد ورودی، با توجه به این که کدام عدد شرایط لازم را ندارد، با شیفต์ دادن و کم کردن توان، نرمال می‌شوند.

- `normalise_b`

عملکرد این استیت، مشابه استیت قبل می‌باشد.

- multiply_0
در این استیت، مقدار توان و علامت حاصل ضرب، یعنی متغیر z مشخص می‌شود و ماژول آماده محاسبه قسمت اصلی حاصلضرب می‌شود.
- multiply_1
در این استیت، قسمت mantissa حاصلضرب با استفاده از متغیرهای کمکی ایجاد شده در استیت قبلی، محاسبه می‌شود.
- normalise_1 2
در این دو استیت، مقدار حاصلضرب نرمال می‌شود.
- round
در این استیت، مقدار حاصلضرب، با توجه به شرایطی که دارد، گرد می‌شود تا آماده ماژول آماده تحویل دادن خروجی گردد.
- pack
در این استیت، ۲۳ بیت کم ارزش mantissa محاسبه شده، در ۲۳ بیت کم ارزش متغیر خروجی اصلی قرار می‌گیرد. همچنین ۸ بیت بعدی که نشان دهنده توان عدد حاصلضرب می‌باشد، نیز در جایگاه خود قرار می‌گیرد. و در نهایت، علامت عدد تعیین می‌شود.
- put_z
در این استیت، که استیت نهایی نیز می‌باشد، حاصلضرب در متغیر خروجی ماژول قرار گرفته و سیگنال stb نیز فعال می‌شود. پس از اتمام این استیت، ماژول به استیت اولیه بازمی‌گردد.

۲.۳.۲ Single Adder

همان‌گونه که در قسمت قبل اشاره شد، این ماژول دو عدد ۳۲ بیتی ممیز شناور را به عنوان ورودی دریافت می‌کند و حاصل جمع این دو عدد را محاسبه می‌نماید. این ماژول شباهت بسیار زیادی به ماژول ضرب کننده دارد. دو عددی ورودی a و b نام دارند و خروجی z نام دارد. در ابتدای ماژول، استیت‌های مختلف به عنوان پارامتر مشخص شده اند و تعدادی متغیر برای ذخیره سازی قسمت‌های مختلف اعداد (برای مثال توان‌های اعداد) در نظر گرفته شده است. استیت‌های ماژول به گونه زیر می‌باشند (در هر قسمت، عملیات انجام شده در هر استیت به طور خلاصه توصیف شده است):

- get_a
در این استیت، ماژول آماده است که ورودی اول از ماژول top گرفته شود و در داخل متغیر a قرار داده شود. پس از دریافت ورودی اول، سیگنال ack به ماژول بالاتر فرستاده می‌شود و ماژول جمع کننده به استیت دریافت ورودی دوم می‌رود.
- get_b
این استیت مشابه استیت بالا می‌باشد. پس از دریافت ورودی دوم و ارسال سیگنال ack، ماژول به استیت unpack می‌رود.

- **unpack**
در این استیت، همانگونه که از اسم آن مشخص است، اعداد ورودی به چند قسمت تقسیم می‌شوند تا فرآیند جمع عدد اعشاری، تسهیل شود. به ازای هر عدد اعشاری ورودی، ۲۳ بیت کم ارزش به عنوان mantissa جدا می‌شود. ۸ بیت بعدی هر عدد نیز جدا شده و عدد ۱۲۷ از آن کم می‌شود تا توان عدد یا exponent محاسبه و در متغیر مخصوص خود، ذخیره شود. و در نهایت، بیت پرارزش نیز به عنوان علامت عدد در نظر گرفته شده و در متغیر مخصوص خود، ذخیره می‌شود. لازم به ذکر است که همه این عملیات‌ها، بر روی هر دو عدد انجام می‌شود.
- **special_cases**
در این استیت، موارد خاصی که ممکن است به عنوان عدد ورودی، به ماژول داده شود، بررسی می‌گردد و اقدامات لازم برای ضرب چنین موارد خاصی، اتخاذ می‌گردد. برای مثال در این استیت چک می‌شود که آیا یکی از اعداد ورودی مقدار بینهایت دارد یا خیر. در صورت داشتن چنین مقداری، خروجی باید مقدار بینهایت باشد.
- **align**
در این استیت، با توجه به این که کدام یک از دو عدد ورودی بزرگتر است، توان عدد کوچکتر برابر با عدد بزرگتر قرار داده می‌شود تا توان‌های دو عدد باهم برابر شوند. همچنین عملیات شیفت mantissa عدد کوچکتر نیز در این استیت انجام می‌شود. پس از هم توان کردن دو عدد ورودی، به مرحله اصلی جمع می‌رویم.
- **add_0 add_1**
در این استیت، عملیات جمع انجام می‌شود. ابتدا علامت‌های دو عدد چک می‌شوند و سپس با توجه به علامت‌های دو عدد، عمل جمع mantissa انجام می‌شود. در نهایت به استیت نرمال کردن حاصل جمع، می‌رویم.
- **normalise_1 normalize_2**
در این دو استیت، حاصل جمع دو عدد ورودی، با شیفت دادن و کم کردن توان، نرمال می‌شوند.
- **round**
مقدار جمع، با توجه به شرایطی که دارد، گرد می‌شود تا آماده ماژول آماده تحویل دادن خروجی گردد.
- **pack**
در این استیت، ۲۳ بیت کم ارزش mantissa محاسبه شده، در ۲۳ بیت کم ارزش متغیر خروجی اصلی قرار می‌گیرد. همچنین ۸ بیت بعدی که نشان دهنده توان عدد حاصل جمع می‌باشد، نیز در جایگاه خود قرار می‌گیرد. و در نهایت، علامت عدد تعیین می‌شود.
- **put_z**
در این استیت، که استیت نهایی نیز می‌باشد، حاصل جمع در متغیر خروجی ماژول قرار گرفته و سیگنال stb نیز فعال می‌شود. پس از اتمام این استیت، ماژول به استیت اولیه بازمی‌گردد.

۳.۳.۲ Vector Inner Product

وظیفه این ماژول اعمال ضرب داخل دو بردار داده شده است. برای انجام این کار از الگوریتم ساده‌ی ضرب تک تک درایه‌ها در هم و سپس جمع این مقادیر استفاده می‌شود. ابتدا طبق قواعد handshake ورودی‌های a و b شامل دو بردار از مقادیر floating-point (به تعداد `number_of_elements` که برابر تعداد درایه‌های سطری یا ستونی است) که ۳۲ bit هستند، ورودی گرفته می‌شود و سپس اعمال ضرب تک تک به تک و خروجی دادن انجام می‌شود. پس از اتمام کار عدد ۳۲ بیتی حاصل از جمع به عنوان خروجی داده می‌شود. در این ماژول به طور دقیق مقدار

$$a_i * b_i = c_i$$

به کمک single Multiplier محاسبه می‌شود و در یک بردار c ذخیره می‌شود. سپس به کمک ماژول single Adder هر کدام از c_i ها با یک مقدار accumulator جمع شده و در نهایت پس از پایدار شدن به عنوان خروجی داده می‌شود.

در ابتدا در یک generate block یک حلقه به اندازه `number_of_elements` برای ضرب کننده‌ها داریم که بصورت هم زمان تمام درایه‌های دو وکتور را بطور متناظر در هم ضرب میکنند و حاصل را در درایه‌ی نظیر در آرایه `temp1_vector` میریزیم سپس یک متغیر به نام `temp_res` داریم که در ابتدا برابر صفر است و با تمام خانه‌های آرایه `temp1_vector` جمع میشود تا حاصل نهایی را خروجی دهد. برای انجام این فرآیندها ۶ state وجود دارد که در ابتداری ماژول با اعداد ۰ تا ۵ مقداردهی شده‌اند. state های ماژول به گونه زیر می باشند (در هر قسمت، عملیات انجام شده در هر state به طور خلاصه توصیف شده است.):

- `state_idle`
در ابتدا در اینجا قرار داریم و اگر ورودی `start` یک و ورودی `vip_done` صفر باشد به استیت بعدی می‌رویم.
- `state_mult_elements`
در این مرحله فرایند ضرب را شروع می‌کنیم (`stb` هارا یک می‌کنیم، این سیگنال‌ها در در تمامی ضرب کننده‌ها یکی است) و ورودی‌ها وارد ضرب کننده‌ها می‌شوند.
- `state_wait_for_mult`
در این مرحله انقدر منتظر می‌مانیم تا تمام `stb` های خروجی در تمام ضرب کننده‌ها یک شود و وقتی این اتفاق افتاد و یعنی خروجی آماده بود و خروجی درون خانه‌های آرایه `temp1_vector` ریخته شد، به مرحله بعد می‌رویم.
- `state_add_elements`
ما یک adder داریم و به ترتیب به عنوان ورودی `temp_res` و یکی از درایه‌های `temp1_vector` را ورودی می‌دهیم به آن و خروجی را در `temp_res` می‌ریزیم، و در متغیر `index` شماره خانه‌ی فعلی از `temp1_vector` را نگه می‌داریم. در این حالت `stb` هارا یک می‌کنیم و ورودی‌ها به adder می‌روند و به مرحله‌ی بعد برای گرفتن خروجی می‌رویم.

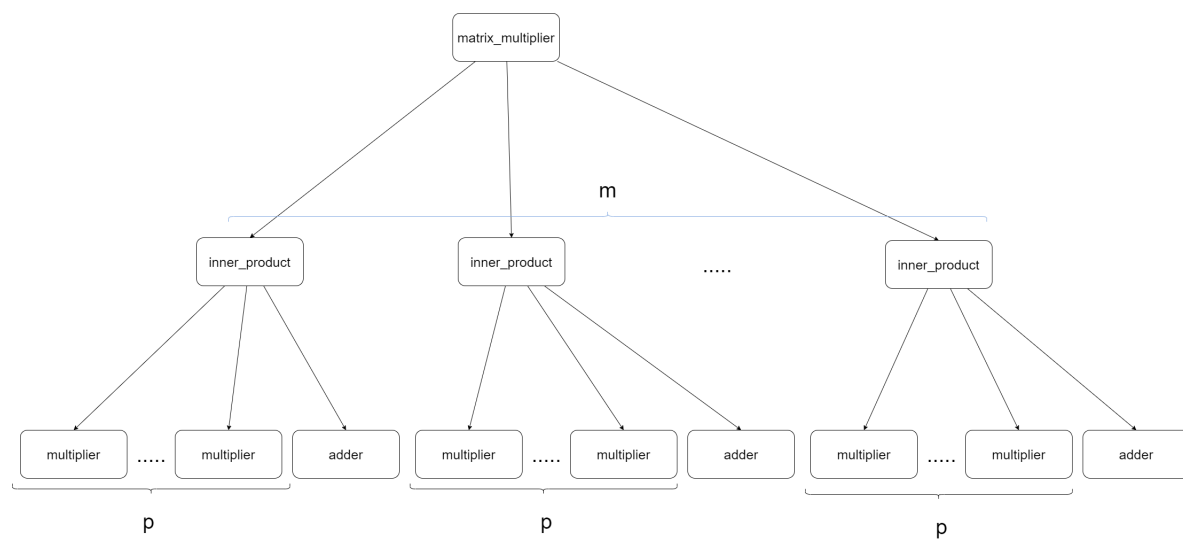
- state_wait_for_add
در حالت انقدر می‌مانیم تا stb خروجی یک شود و وقتی یک شد خروجی را می‌گیریم و در temp_res می‌ریزیم و سپس index را یکی زیاد می‌کنیم (در اول قسمت قبل شرط پایان چک می‌شود یعنی وقتی که index برابر ۱ + number_of_elements شود).
- state_out_is_ready
مرحله ی آخر است که vip_done را یک می‌کنیم و به حالت صفر برمی‌گردیم. (state_idle)

۴.۳.۲ Matrix Multiplier

این ماژول، ماژول اصلی مدار را تشکیل می‌دهد و هدف آن محاسبه ضرب دو ماتریس داده شده است. برای انجام این کار از یک بخش ترکیبی برای برای محاسبه حاصل ضرب در هر ستون و یک بخش ترتیبی برای تعمیم بخش ترکیبی به تمام ستون ها استفاده می‌شود و الگوریتم اجرای آن الگوریتم معمول محاسبه ضرب داخلی ردیف های ماتریس نخست و ستون های ماتریس دوم است. دو ورودی شامل دو ماتریس A و B در قالب دو بردار به ابعاد تعداد درایه های هر ماتریس ضرب در تعداد بیت هر درایه با پیروی از قوانین handshake ورودی داده می‌شود و سپس با محاسبه ضرب این دو ماتریس، مقدار درقالب یک بردار مشابه ورودی ها به عنوان ماتریس حاصل ضرب خروجی داده می‌شود. ابتدا به کمک ماژول Product Inner Vector حاصل ضرب داخلی تمام سطرهای ماتریس A را در ستون مشخص i از ماتریس B محاسبه می‌کنیم. بردار محاسبه شده را در ستون i خروجی C ذخیره می‌کنیم و سپس با افزایش i در کلاک سایکل های بعدی تمام ماتریس C را بدست می‌آوریم.

۴.۲ ساختار درختی سیستم

لازم به ذکر است که حروف نوشته شده، نشان دهنده تعداد تکرار هر نمونه ماژول در ماژول پدر می‌باشد. m نشان دهنده تعداد سطرهای ماتریس اول و p نشان دهنده تعداد ستون‌های ماتریس اول یا همان تعداد سطرهای ماتریس دوم می‌باشد. ساختار درختی سیستم از قرار زیر می‌باشد:



۳ روند شبیه‌سازی

۱.۳ توصیف test bench

۲.۳ adder test bench

در تست این ماژول سعی شده همه‌ی حالات ممکن که برای جمع دو عدد وجود دارد تست شود برای مثال جمع دو عدد مثبت، جمع دو عدد منفی، جمع یک عدد مثبت و یک عدد منفی و ... همچنین سعی شده حالت‌های خاص نیز پوشش داده شود برای مثال جمع دو عدد بی‌نهایت و یا جمع عدد و غیر عدد در تست‌ها لحاظ شده‌اند. در شکل زیر خروجی تست بنچ برای ورودی‌های متفاوت مشاهده می‌شود.

```
this is a test for floating point adder
result of 00000000000000000000000000000000 + 00000000000000000000000000000000 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

test1: 1 + 1
result of 00111111100000000000000000000001 + 00111111100000000000000000000001 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

result of 00111111100000000000000000000001 + 00111111100000000000000000000001 = 01000000000000000000000000000001

test2: 1 + (-2)
result of 00111111100000000000000000000001 + 10111111100000000000000000000010 = 01000000000000000000000000000001

result of 00111111100000000000000000000001 + 10111111100000000000000000000010 = 10110100000000000000000000000000

test3: -1 + 2
result of 01111111100000000000000000000001 + 00111111100000000000000000000010 = 10110100000000000000000000000000

test4: -1 + (-2)
result of 01111111100000000000000000000001 + 10111111100000000000000000000010 = 10110100000000000000000000000000

test5: 1 + NAN
result of 00111111100000000000000000000001 + 11111111100000000000000000000001 = 10110100000000000000000000000000

result of 00111111100000000000000000000001 + 11111111100000000000000000000001 = 11111111100000000000000000000000

test6: infinity + NAN
result of 01111111100000000000000000000000 + 11111111100000000000000000000001 = 11111111100000000000000000000000

test7: -infinity + NAN
result of 00111111110000000000000000000000 + 11111111100000000000000000000001 = 11111111100000000000000000000000

test8: infinity + 1
result of 01111111100000000000000000000000 + 00111111100000000000000000000001 = 11111111100000000000000000000000

result of 01111111100000000000000000000000 + 00111111100000000000000000000001 = 01111111100000000000000000000000

test9: -infinity + 1
result of 00111111110000000000000000000000 + 00111111100000000000000000000001 = 01111111100000000000000000000000

result of 00111111110000000000000000000000 + 00111111100000000000000000000001 = 01000000011000000000000000000000

test10: infinity + infinity
result of 01111111100000000000000000000000 + 01111111100000000000000000000000 = 01000000011000000000000000000000

result of 01111111100000000000000000000000 + 01111111100000000000000000000000 = 01111111100000000000000000000000

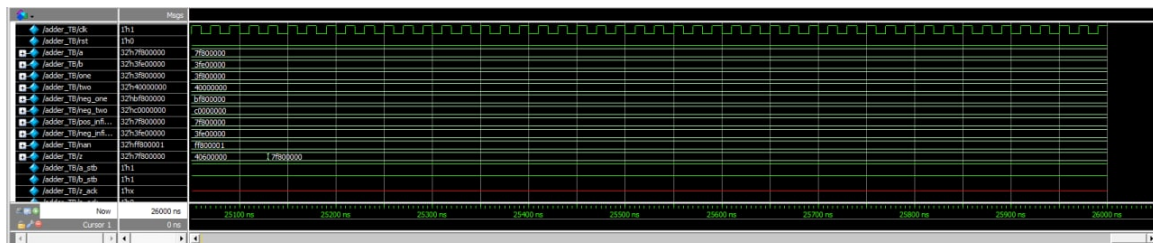
test11: -infinity + (-infinity)
result of 00111111110000000000000000000000 + 00111111110000000000000000000000 = 01111111100000000000000000000000

result of 00111111110000000000000000000000 + 00111111110000000000000000000000 = 01000000011000000000000000000000

test12: infinity + (-infinity)
result of 01111111100000000000000000000000 + 00111111110000000000000000000000 = 01000000011000000000000000000000

result of 01111111100000000000000000000000 + 00111111110000000000000000000000 = 01111111100000000000000000000000
```

wave form به دست آمده نیز در شکل بعدی قابل مشاهده است.

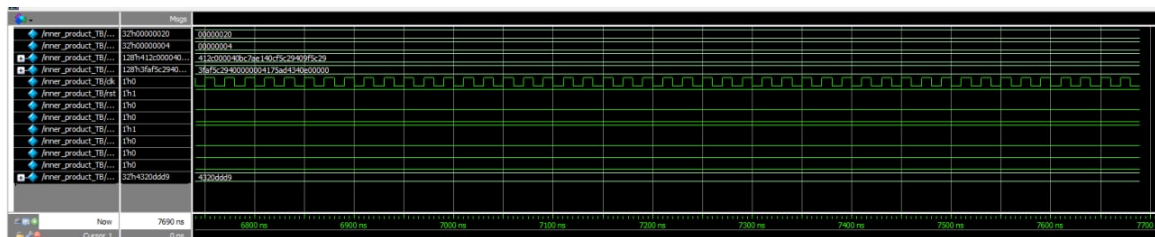


۳.۳ test bench مازول inner product

در این تست بنچ با ضرب دو بردار چهار درایه ای برخی حالات لازم را ایجاد و بررسی کردیم. در شکل زیر خروجی تست مشاهده می‌شود.

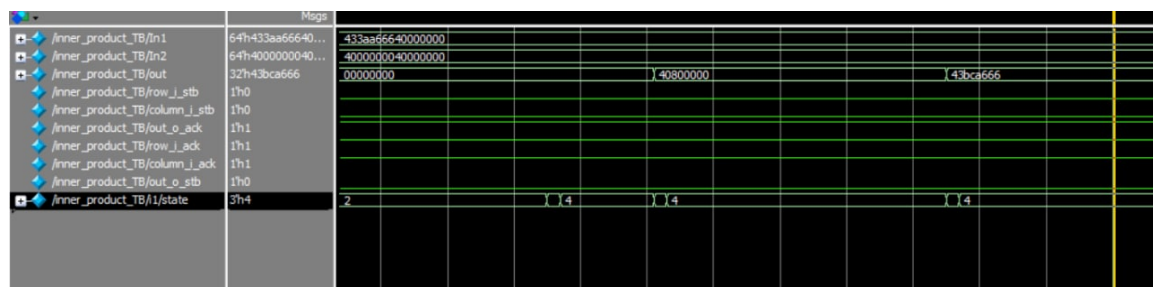
```
this is a test for inner product module
output is 01000011001000001101110111011001
```

wave form به دست آمده نیز در شکل قابل مشاهده است.



۴.۳ test bench مازول matrix multiplier

wave form به دست آمده نیز در شکل قابل مشاهده است.



مراجع

- [1] Wikipedia. Matrix multiplication, 2021.
- [2] Wikipedia. Floating point *arithmetic*, 2021.
- [3] Wikipedia. Tensor processing unit, 2021.

- [4] Mahmoodi, Ahmad. floating point operations, 2011.
- [6] eedwards. Floating point numbers.
- [7] Wikipedia. 3d projection, 2021.