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 عبداله زاده ـ ۹۷۱۰۶۱۳۲

فاطمه خاشعی ـ ۹۷۱۰۵۸۹۹

عرشیا اخوان ـ ۹۷۱۱۰۴۲۲

مجید طاهرخانی ـ ۹۷۱۰۶۱۰۸

رضا امینی ـ ۹۷۱۰۱۲۷۵

علی بالاپور ـ ۹۷۱۰۱۳۲۶

فهرست مطالب

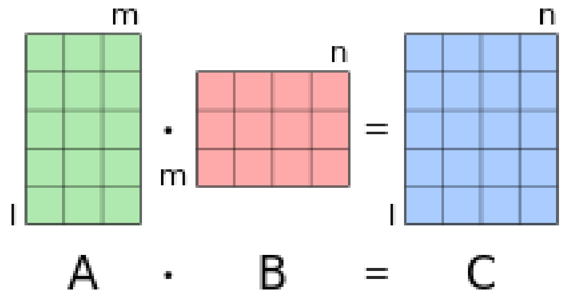
۱ مقدمه

۱.۱ چکیده

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

۲.۱ تعریف و تاریخچه مختصر

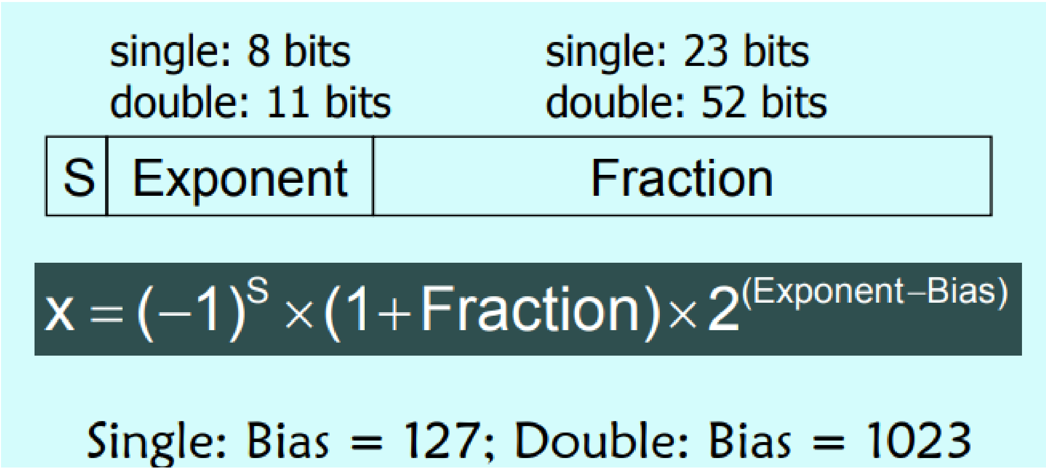
عملیات ضرب ماتریسی یکی از مهمترین عملیات های موجود در جبر خطی و ریاضیات می باشد. در ضرب ماتریسی، دو ماتریس با ابعاد n*m و m*k در یکدیگر ضرب می شوند و یک ماتریس n*k حاصل می شود. شرط عملی بودن ضرب ماتریس این است که تعداد ستون های ماتریس اول با تعداد سطر های ماتریس دوم برابر باشد. در صورتی که A و B دو ماتریس با شرایط گفته شده باشند، حاصل ضرب ماتریسی به صورت AB نشان داده می شود. [۱]



در این پروژه ما به ضرب ماتریس هایی می پردازیم که درایه هایشان، اعداد اعشاری ممیز شناور با دقت یگانه و یا Single Precision Floating Point می باشند. عملیات روی اعداد ممیز شناور (floating point) به طور گسترده ای در پردازش های علمی مورد استفاده قرار می گیرد. سازمان IEEE برای عملیات های ممیز شناور یک استاندارد کلی به نام IEEE-754 تعیین کرده است. در این پروژه، برای بدست آوردن حاصل ضرب دو ماتریس با درایه های اعشاری ممیز شناور، لازم است که از جمع و ضرب FP استفاده شود. در قسمت عملکرد، ضرب و جمع FP مورد بررسی واقع می شود. 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، تقریبا از تعداد ثابتی از ارقام significand و یک توان در پایه ثابت تشکیل می شود؛ این پایه معمولا ۲، ۱۰ و یا ۱۶ است. یعنی:

$$\text{significand} * \text{base}^{\text{exponent}}$$

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 کامپیوتراستفاده شده است. در سال ۱۹۸۵ ، استاندارد IEEE 754 برای محاسبات ممیز شناور ثبت شد. این استاندارد واگرایي شیوه های به کار رفته براي 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*n و ماتریس B دارای ابعاد n*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 سطر های ماتریس A) و دارای p ستون (تعداد ستون های ماتریس B) می باشد. هر یک از درایه های ماتریس C از رابطه زیر به دست می آیند.

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

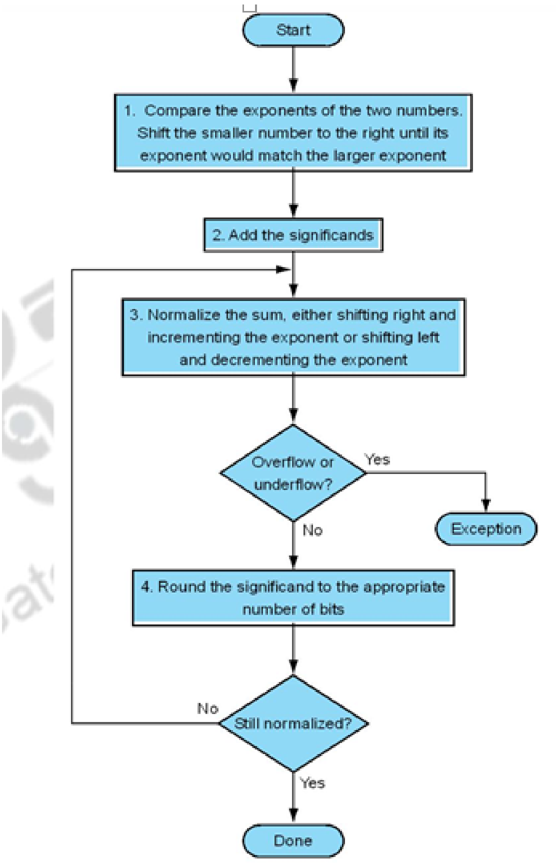
به عبارت دیگر، سطر i ام ماتریس A و ستون j ام ماتریس B به عنوان یک وکتور(بردار) در نظر گرفته می شوند و حاصل ضرب داخلی این دو وکتور در درایه ij ماتریس C قرار می گیرد. در نهایت حاصلضرب ماتریسی دو ماتریس A و B که به صورت AB نشان داده می شود، به صورت زیر است:

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

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



فلوچارت الگوریتم جمع اعداد ممیز شناور

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 می کنیم. برای این کار، توان عدد کوچکتر را با توان عدد بزرگتر یکسان می کنیم. یعنی :

$$0.087 * 10^{1} + 9.95 * 10^{1}$$

سپس این دو عدد را با یکدیگر جمع می زنیم و آن را نرمال می کنیم:

$$0.087 + 9.95 = 10.037 * 10^{1}$$

$$10.037 * 10^{1} = 1.0037 * 10^{2}$$

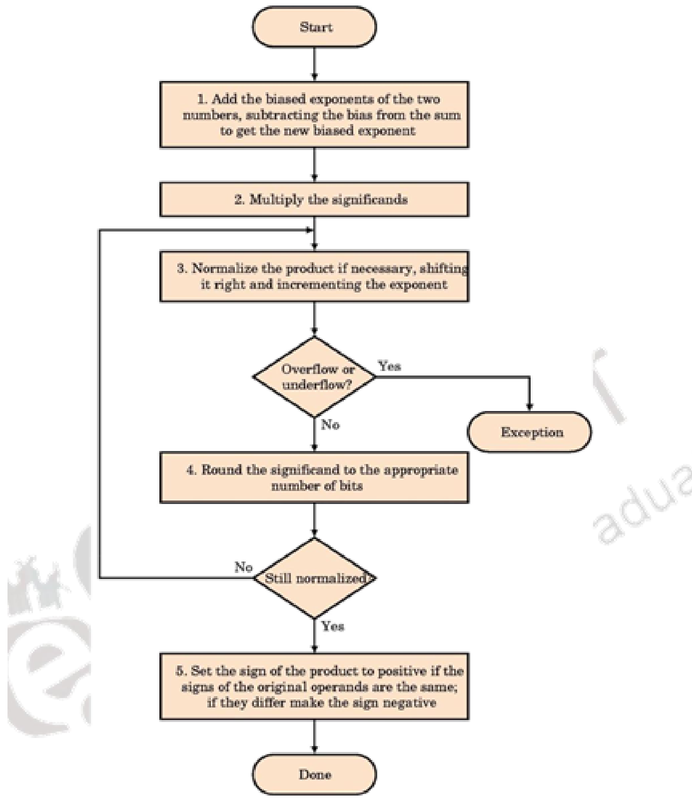
و درنهایت حاصل را گرد می نماییم.

$$1.004 * 10^{2}$$

[۵] [۶]

۳.۳.۱ ضرب FP

۱ ـ ابتدا توان ها باهم جمع می شوند.
۲ ـ مقادیر significand در همدیگر ضرب می شوند.
۳ ـ اعداد نرمال می شوند و سپس overflow ویا underflow شدن عدد حاصل، بررسی می شود.
۴ ـ اعداد گرد می شوند.
در صورتی عدد نرمال نباشد، مراحل ۳ و ۴ تکرار می شود.
۵ ـ علامت عدد تعیین می شود.
رویه کلی الگوریتم ضرب FP با دقت یگانه، در فلوچارت زیر نیز مشخص می باشد.



فلوچارت الگوریتم ضرب اعداد ممیز شناور

برای مثال می‌خواهیم حاصلضرب دو عدد اعشاری با ممیزشناور $1.110 * 10^{10}$ و $9.200 * 10^{-5}$ را بدست آوریم. ابتدا توان های هردو عدد را یکسان می کنیم:

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

سپس قسمت های اصلی اعداد را در یکدیگر ضرب می کنیم:

$$1.110 * 9.200 = 10.21200$$

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

$$1.021 * 10^6$$

[۵] [۶]

۴.۱ کاربردها

در گذشته، از ضرب ماتریسی به عنوان روشی برای راحت‌تر و واضح تر کردن محاسبات جبرخطی استفاده می شد. اما امروزه، ضرب ماتریسی یکی از مهمترین و پایه ای ترین عملیات ها در علوم ریاضیات، جبرخطی، فیزیک و علوم کامپیوتر می باشد. در ادامه به تعدادی از کاربردهای ضرب ماتریس می پردازیم.

۱.۴.۱ نگاشت خطی

نگاشت خطی یا ترادیسش خطی، یک تابع بین دو فضای برداری است که دو عملیات جمع برداری و ضرب نرده‌ای را باقی نگه می‌دارد. این تابع همچنین رابطهٔ 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 می شود. برای مثال، نگاشت خطی A، یک ماتریس m*n می باشد که برداری در فضای n بعدی را به برداری در فضای m بعدی تبدیل می کند. داریم: [۱]

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

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

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

۲.۴.۱ دستگاه معادلات خطی

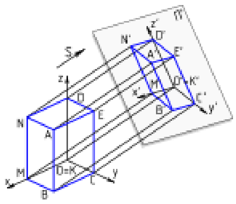
برای حل دستگاه معادلات خطی، می توان از ضرب ماتریسی استفاده کرد.[۱]

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

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

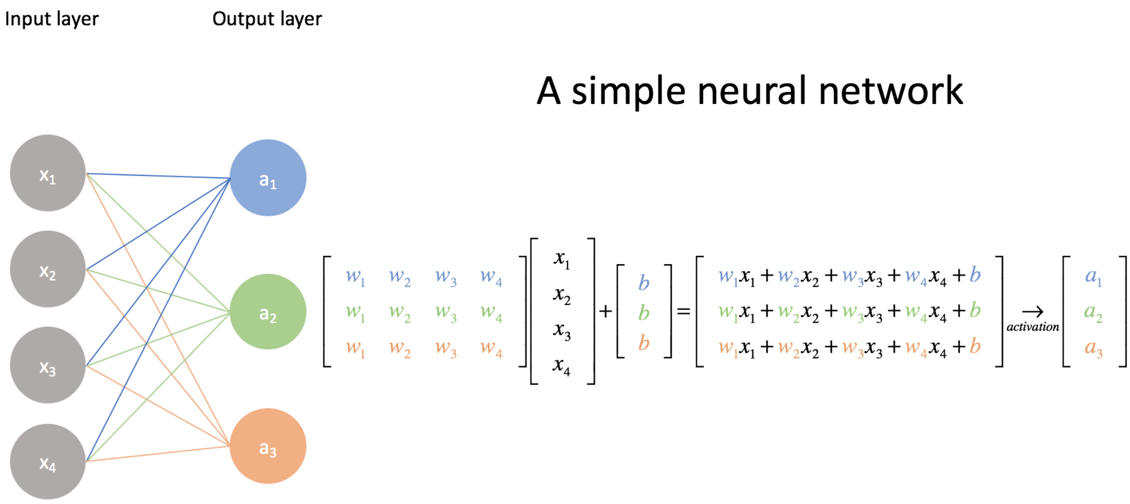
۳.۴.۱ 3D Projection

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 می شود. روش های مختلفی برای اجرای این تکنیک وجود دارد. یک روش که Orthographic projection نام دارد، از ضرب ماتریسی برای اجرای این تکنیک استفاده می کند. [۷]



۴.۴.۱ یادگیری عمیق

ضرب ماتریس یکی از مهمترین عملیات های مورد استفاده در یادگیری ماشین برای آموزش شبکه های عصبی می باشد. در هنگام آموزش مدل های عصبی عمیق، از ضرب ماتریس برای محاسبه و بروزرسانی وزن ها استفاده می شود. همچنین در مرحله تست نیز، برای بدست آوردن پاسخ نهایی، در هر لایه از عملیات ضرب ماتریسی استفاده می شود. بنابراین، درصورتی که بتوانیم مدار سریعتر و بهینه تری برای عملیات ضرب ماتریس داشته باشیم، می توانیم از الگوریتم های بهینه تری برای آموزش و تست شبکه های 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.



ساختار کلی یادگیری عمیق و استفاده از ضرب ماتریسی

۲ توصیف معماری سیستم

۱.۲ اینترفیس‌های سیستم و کلاک سیستم

سیستم به طور کلی از ۴ ماژول اصلی تشکیل شده است که این ماژول ها، به صورت تو در تو، از یکدیگر استفاده می کنند. هر ماژول واحد کنترل جداگانه خود را دارد و صرفا بین ماژول ها، سیگنال های کنترلی ای که مربوط به ماژول های دیگر است، رد و بدل می شود. این چهار ماژول اصلی به صورت زیر می‌باشند:

- matrix_multiplier
- inner_product
- adder
- multiplier

ماژول matrix_multiplier به نوعی ماژول اصلی سیستم ما می باشد 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 می شود 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 پارامتر های m و p و n و word_width به عنوان پارامترهای ورودی، به ماژول داده می شوند. این پارامترها به ترتیب نشان دهنده تعداد سطر ماتریس اول، تعدا ستون های ماتریس اول و سطرهای ماتریس دوم، تعداد ستون های ماتریس دوم و طول عدد هر درایه می باشند. همچنین به ترتیب، مقادیر پیش فرض ۱۶ ـ ۱۶ ـ ۱۶ ـ 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 حتما تعداد ستون های ماتریس اول با تعداد سطرهای ماتریس دوم برابر باشد.
ورودی های اصلی این ماژول، دو ماتریس matrix_A و matrix_B و سیگنال start و clk می باشند. با توجه به این که نمی توان یک آرایه را به عنوان ورودی، به ماژول داد، پس هر کدام از ماتریس ها 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 می گیریم. یعنی در هر ماتریس همه سطرها را پشت سرهم قرار داده و آن هارا به صورت بیت به بیت ذخیره می کنیم. همچنین می دانیم هر عدد ۳۲ بیت (یا به اندازه word_width) فضا را اشغال می کند. پس به راحتی می توان به هر عدد دسترسی داشت. سیگنال start هم تعیین کننده شروع کار کل مدار می باشد. همچنین سیگنال clk نیز کلاک کلی مدار می باشد.
در ماژول inner_product، پارامتر number_of_element نشان دهنده تعداد اعضای وکتور ها می

باشد. در این ماژول ورودی های In1, In2, clk, rst, start به ماژول داده می شود. In1 و In2 دو وکتور با سایز های برابر (element_length * number_of_elements) می باشند که مقادیر دو بردار (به طور دقیق تر سطر ماتریس اول و ستون ماتریس دوم) در آن های ذخیره شده است. هدف این ماژول ضرب داخلی این دو بردار در هم می باشد. همچنین سیگنال های 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. ورودی ها و خروجی های دو ماژول multiplier و adder با یکدیگر یکسان می باشند. ورودی های 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 می باشند. ورودی های 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 می شود. ورودی rst و clk نیز واضح می باشند.

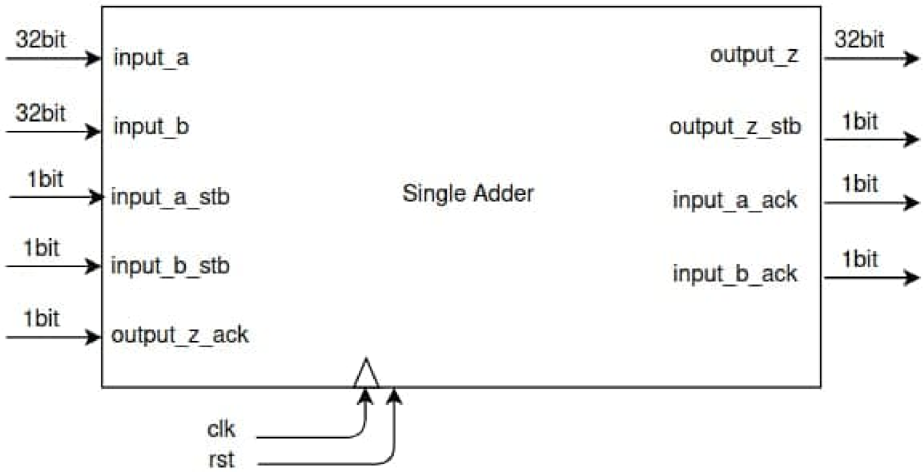
۲.۱.۲ خروجی‌ها

در ماژول matrix_multiplier یکی از خروجی ها، ماتریس خروجی یا همان matrix_C خواهد بود. این ماتریس، حاصل ضرب ماتریسی دو ماتریس ورودی matrix_A و matrix_B می باشد. همان گونه که در قسمت قبل نیز اشاره شد، با توجه به این که نمی توان آرایه در یک ماژول خروجی داد، پس مانند قسمت قبل عمل کرده و هر سطر را پشت سرهم قرار در یک وکتور با سایز m*n*word_width قرار می دهیم. این ماژول دو سیگنال خروجی با نام های done و ready نیز دارد که به ترتیب نشان دهنده اتمام عملیات و آماده بودن ماژول برای دریافت ورودی می باشد.
در ماژول inner_product خروجی out و done را داریم. done نشان دهنده تمام شدن عملیات ضرب داخلی می باشد. out نیز یک عدد ۳۲ بیتی (به طور پیش فرض) می باشد که نشان دهنده حاصل ضرب داخلی است.
در ماژول های multiplier و adder نیز خروجی های input_a_ack output_z_stb output_z input_b_ack را داریم. output_z همان خروجی جمع و یا ضرب می باشد که سایز آن به طور پیش فرض برابر ۳۲ بیت است. output_z_stb نیز اتمام عملیات ضرب و جمع را نشان می دهد.

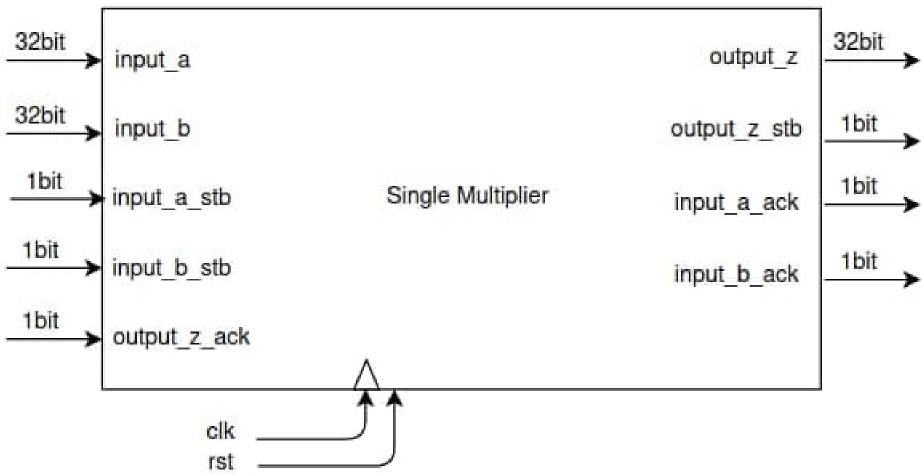
۳.۱.۲ کلاک سیستم

همان گونه که در قسمت های بالا نیز اشاره شد، کلاک استفاده شده در هریک از ماژول ها، همان کلاک اصلی سیستم می باشد.

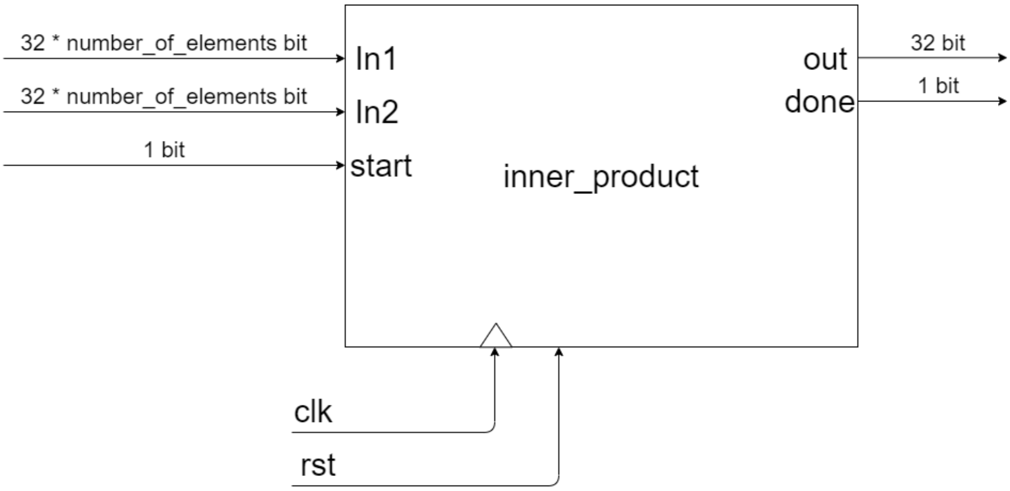
۲.۲ دیاگرام بلوکی سخت‌افزار



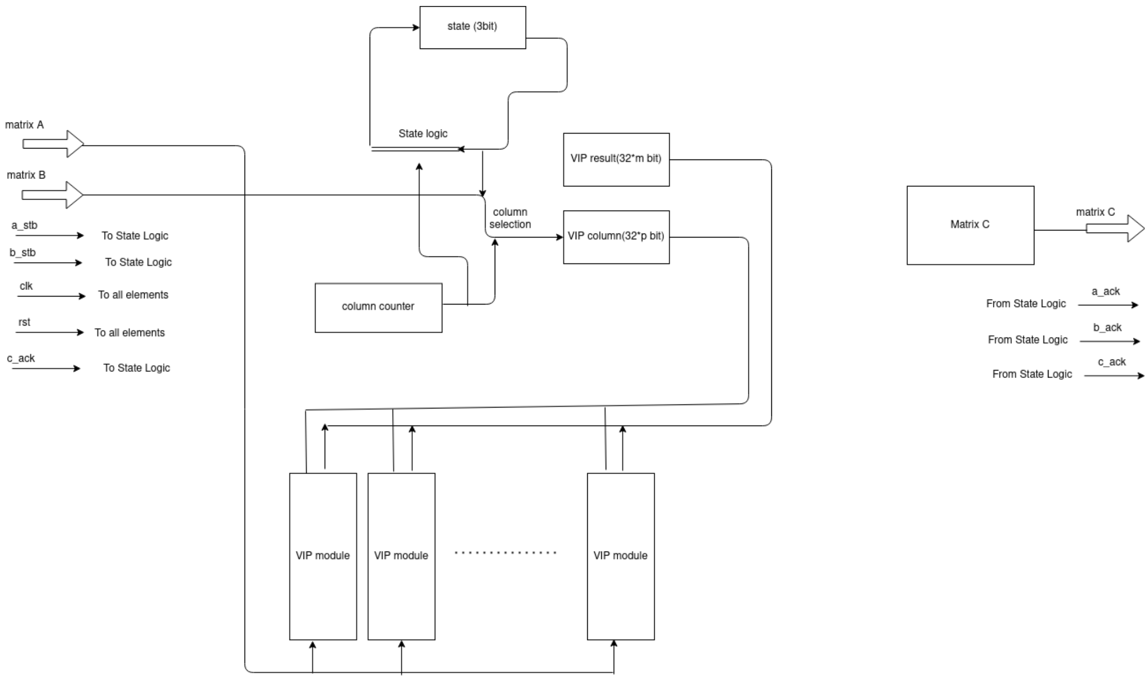
دیاگرام ماژول جمع‌کننده



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 می نماید. دو عددی ورودی a و b نام دارند و خروجی z نام دارد. در ابتدای ماژول، استیت های مختلف به عنوان پارامتر مشخص شده اند و تعدادی متغیر برای ذخیره سازی قسمت های مختلف اعداد (برای مثال توان های اعداد) در نظر گرفته شده است. استیت های ماژول به گونه زیر می باشند (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 می شود 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 می رود.

- 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 می شود 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 می گردد 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 شیفت 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 حاصلظرب 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، مقدارحاصلضرب 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 می شود.

- 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، استیت های مختلف به عنوان پارامتر مشخص شده اند و تعدادی متغیر برای ذخیره سازی قسمت های مختلف اعداد (برای مثال توان های اعداد) در نظر گرفته شده است. استیت های ماژول به گونه زیر می باشند (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 می شود 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 می رود.

- 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 می شود 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 می گردد 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 چینین 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 علامت های دوعدد چک می شوند 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 حاصل‌جمع می‌باشد، نیز در جایگاه خود قرار میگیرد. و در نهایت، علامت عدد تعیین می شود.

- 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 درایه ها در هم و سپس جمع این مقادیر استفاده می‌شود. ابتدا طبق قواعد handshake ورودی های 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 به تک و خروجی دادن انجام می‌شود. پس از اتمام کار عدد ۳۲بیتی 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 کننده ها داریم که بصورت هم زمان تمام درایه های دو وکتور را بطور متناظر در هم ضرب میکنند و حاصل را در درایه ی نظیر در آرایه mul_out میریزیم سپس یک متغیر به نام inner_product_result داریم که در ابتدا برابر صفر است و با تمام خانه های آرایه mul_out جمع میشود تا حاصل نهایی را خروجی دهد.

برای انجام این فرآبندها 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 row_i_stb و column_i_stb و out_o_ack صفر باشد به استیت بعدی می‌رویم.

- state_mult_elements
در این مرحله فرایند ضرب را شروع می‌کنیم (stb هارا یک می‌کنیم، این سیگنال ها در در تمامی ضرب کننده ها یکی است) و ورودی ها وارد ضرب کننده ها می‌شوند.

- state_wait_for_mult
در این مرحله انقدر منتظر می‌مانیم تا تمام stb های خروجی در تمام ضرب کننده ها یک شود و وقتی این اتفاق افتاد و یعنی خروجی آماده بود و خروجی درون خانه های آرایه temp1_vector ریخته شد، به مرحله بعد می‌رویم.

- state_add_elements
ما یک adder داریم و به ترتیب به عنوان ورودی inner_product_result و یکی از درایه های mul_out را ورودی می‌دهیم به آن و خروجی را در inner_product_result می‌ریزیم، و در متغیر index شماره خانه‌ی فعلی از mul_out را نگه می‌داریم. در این حالت stb هارا یک می‌کنیم و ورودی ها به adder می‌روند و به مرحله ی 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 inner_product_result 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 اخر است که s_out_o_stb را یک می‌کنیم و به حالت صفر برمی‌گردیم. (state_idle)

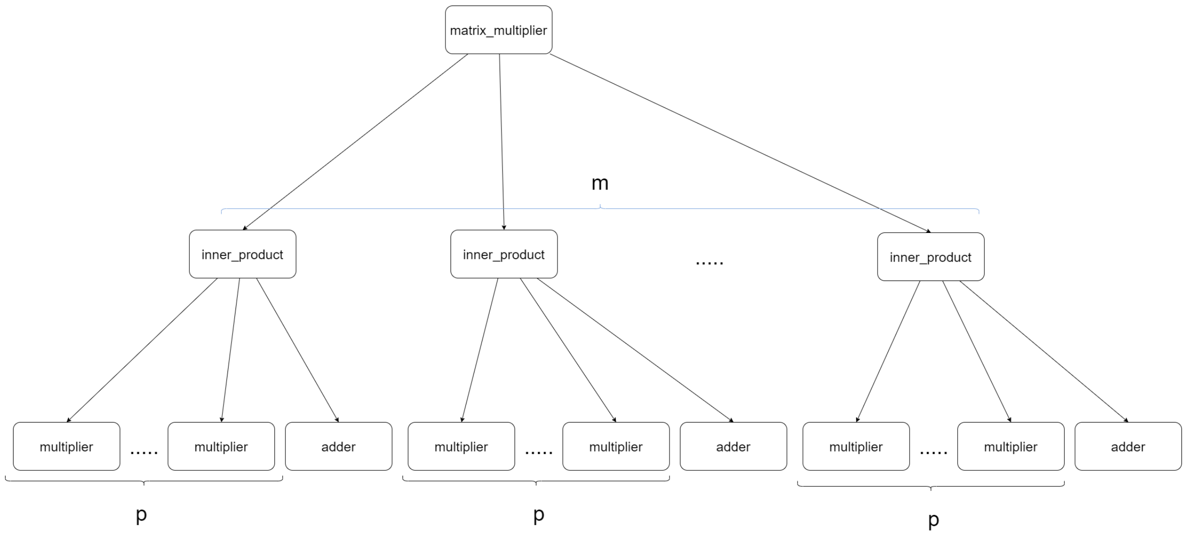
۴.۳.۲ Matrix Multiplier

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 سطر های 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:



ساختار درختی مدار

۳ روند شبیه‌سازی

۱.۳ توصیف test bench

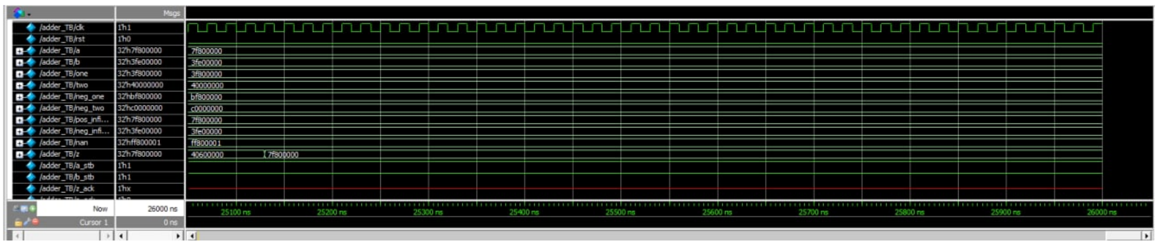
برای تست هر یک از ماژول‌های adder, inner product, matrix multiplier یک تست بنچ آماده شده است. تست بنچ هر کدام در فولدر اصلی قرار داده‌شده است.

۲.۳ test bench ماژول adder

در تست این ماژول سعی شده همه‌ی حالات ممکنی 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 غیرعدد 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 01111111000000000000000000000001 +  00111111100000000000000000000010  = 10110100000000000000000000000000

test4: -1 + (-2)
result of 01111111000000000000000000000001 +  10111111100000000000000000000010  = 10110100000000000000000000000000

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

result of 00111111100000000000000000000001 +  11111111100000000000000000000001  = 11111111110000000000000000000000

test6: infinity + NAN
result of 01111111100000000000000000000000 +  11111111100000000000000000000001  = 11111111110000000000000000000000

test7: -infinity + NAN
result of 00111111111000000000000000000000 +  11111111100000000000000000000001  = 11111111110000000000000000000000

test8: infinity + 1
result of 01111111100000000000000000000000 +  00111111100000000000000000000001  = 11111111110000000000000000000000

result of 01111111100000000000000000000000 +  00111111100000000000000000000001  = 01111111100000000000000000000000

test9: -infinity + 1
result of 00111111111000000000000000000000 +  00111111100000000000000000000001  = 01111111100000000000000000000000

result of 00111111111000000000000000000000 +  00111111100000000000000000000001  = 01000000011000000000000000000000

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

result of 01111111100000000000000000000000 +  01111111100000000000000000000000  = 01111111100000000000000000000000

test11: -infinity + (-infinity)
result of 00111111111000000000000000000000 +  00111111111000000000000000000000  = 01111111100000000000000000000000

result of 00111111111000000000000000000000 +  00111111111000000000000000000000  = 01000000011000000000000000000000

test12: infinity + (-infinity)
result of 01111111100000000000000000000000 +  00111111111000000000000000000000  = 01000000011000000000000000000000

result of 01111111100000000000000000000000 +  00111111111000000000000000000000  = 01111111100000000000000000000000
```

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

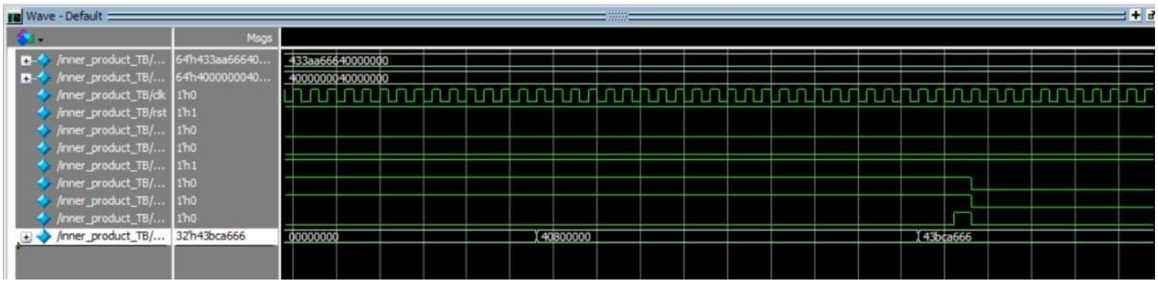


۳.۳ test bench ماژول inner product

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.

```
VSIM 84> run
# this is a test for inner product module
# output is 01000011101111001010011001100110 and it is 1

VSIM 85>
```

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

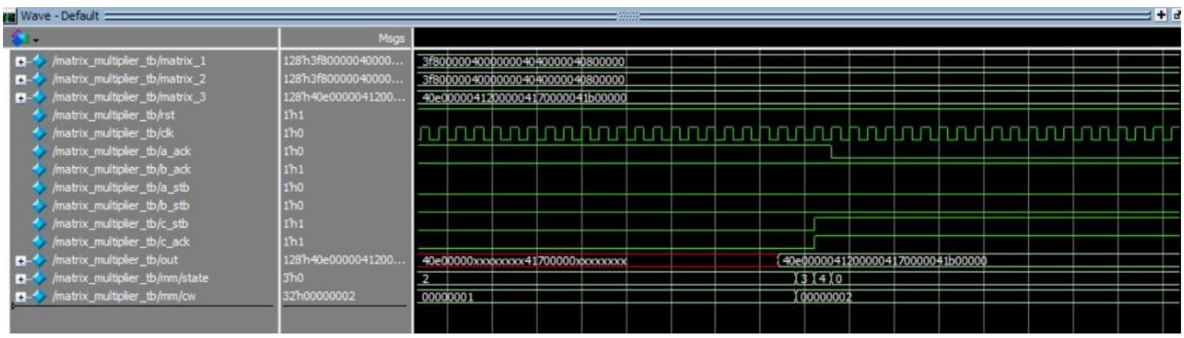


۴.۳ test bench ماژول matrix multiplier

در ابتدا دو ماتریس ۲ در ۲ از فایل خوانده شده و نتیجه ی ضرب با نتیجه درست مقایسه شده است

```
VSIM 70> run
# loading rom 3
# loading rom 1
# loading rom 2
# 01000000111000000000000000000000 == 01000000111000000000000000000000 is 1
# 01000001001000000000000000000000 == 01000001001000000000000000000000 is 1
# 01000001011100000000000000000000 == 01000001011100000000000000000000 is 1
# 01000001101100000000000000000000 == 01000001101100000000000000000000 is 1
```

wave form به دست آمده در شکل قابل مشاهده است.



در تست بعدی، دو ماتریس ۶ در ۶ از ورودی خوانده می‌شود و عملیات ضرب ماتریسی بر روی این دو ماتریس اجرا می‌شود. مقادیر این دو ماتریس به شکل زیر می‌باشد:

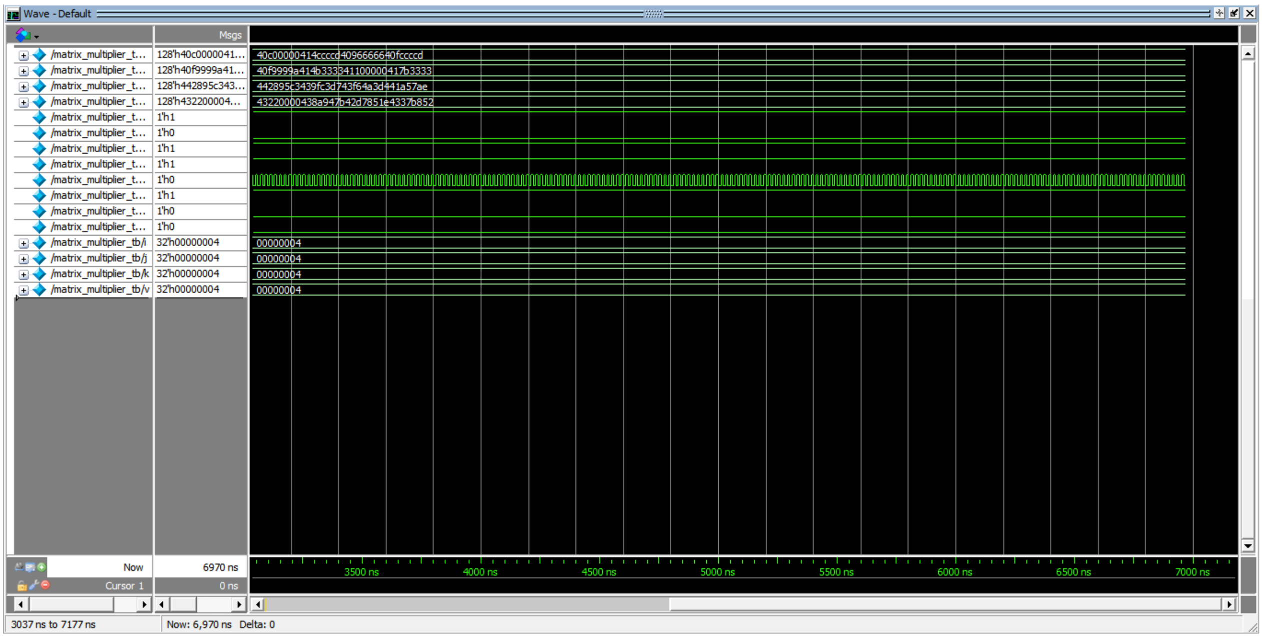
$$A = \begin{pmatrix} 6.0 & 12.8 & 4.7 & 7.9 & 11.5 & 15.5 \\ 9.8 & 11.7 & 6.2 & 0.4 & 3.9 & 10.9 \\ 4.2 & 17.8 & 2.5 & 2.4 & 17.9 & 11.4 \\ 5.2 & 13.1 & 4.5 & 14.6 & 14.8 & 19.1 \\ 8.8 & 0.3 & 7.6 & 1.4 & 14.5 & 9.9 \\ 13.8 & 6.6 & 4.3 & 3.7 & 2.8 & 1.4 \end{pmatrix}$$

$$B = \begin{pmatrix} 7.8 & 12.7 & 9.0 & 15.7 & 14.2 & 17.6 \\ 18.8 & 7.9 & 9.0 & 4.3 & 9.3 & 12.7 \\ 9.8 & 2.6 & 0.9 & 8.5 & 8.0 & 3.9 \\ 5.1 & 7.1 & 13.5 & 18.2 & 4.1 & 10.3 \\ 10.5 & 0.9 & 1.9 & 0.2 & 10.3 & 0.4 \\ 11.6 & 4.1 & 12.3 & 18.2 & 3.5 & 18.1 \end{pmatrix}$$

حاصل ضرب دو ماتریس بالا، یعنی AB برابر است با:

$$C = \begin{pmatrix} 674.34 & 319.53 & 492.58 & 617.37 & 446.93 & 653.01 \\ 526.59 & 284.05 & 345.96 & 463.31 & 377.53 & 548.22 \\ 724.33 & 280.35 & 406.88 & 418.47 & 479.29 & 547.95 \\ 782.36 & 376.52 & 628.9 & 792.52 & 510.82 & 777.45 \\ 422.99 & 197.47 & 256.96 & 412.61 & 378.29 & 387.74 \\ 338.37 & 273.11 & 259.96 & 374.97 & 340.65 & 408.04 \end{pmatrix}$$

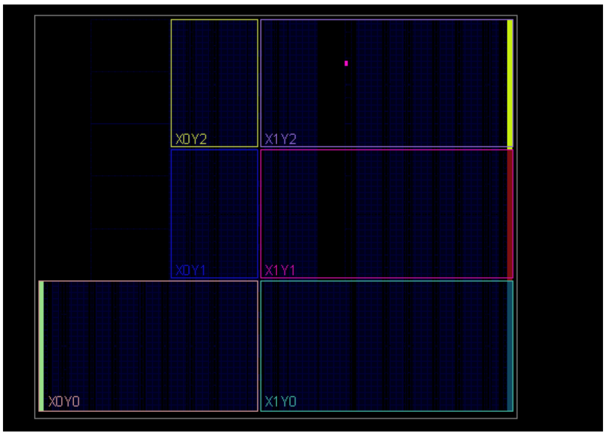
فرم عدد اعشاری ممیز شناور دو ماتریس A و B در ماتریس‌های matrix_1.txt و matrix_2.txt در فولدر test_files ذخیره شده است. برای تست کردن ضرب ماتریس، این دو فایل را به مدار داده و خروجی تولید می‌گردد. wave form اجرایی مدار به صورت زیر می‌باشد:



۴ گزارش پیاده‌سازی و شبیه‌سازی

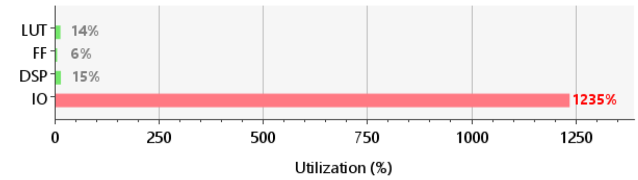
با استفاده از ابزار Vivado و روی دستگاه zynq-7000 xc7z020iclg400 سنتز و implementation سیستم انجام شد که نتایج آن در ادامه، قابل مشاهده می‌باشد.

۱.۴ مساحت



شماتیک کلی

Resource	Utilization	Available	Utilization %
LUT	7456	53200	14.02
FF	6292	106400	5.91
DSP	32	220	14.55
IO	1544	125	1235.20



Utilization Summary

۲.۴ تعداد FF و LUTها

تعداد FF و LUT ها در تصویر ۱.۴ قابل مشاهده می‌باشد. جزییات بیشتر، در تصویر زیر قرار دارد.

Name	Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	DSPs (220)	Bonded IOB (125)	BUFGCTRL (32)
matrix_multiplier	7456	6292	132	32	1544	3
genblk1[0].vip (inner_product)	1775	1395	1	8	0	0
genblk1[1].vip (inner_product_0)	1692	1395	1	8	0	0
genblk1[2].vip (inner_product_1)	1480	1395	1	8	0	0
genblk1[3].vip (inner_product_2)	1644	1395	1	8	0	0

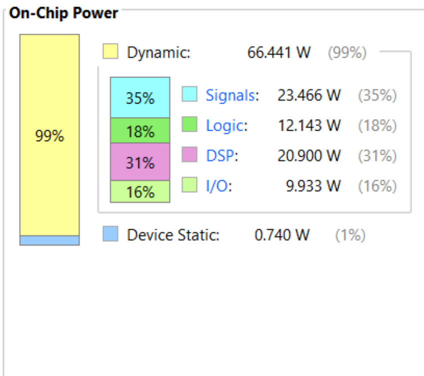
Detailed Utilization

۳.۴ زمان‌بندی، کلاک و حداکثر فرکانس

متاسفانه نتوانستیم با ابزار Vivado مقادیر زمان بندی و کلاک و فرکانس مدار را بدست آوریم.

۴.۴ توان

مشخصات توان مدار، در تصاویر زیر مشخص می‌باشد.



Power Report 1

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**67.181 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**125.0°C**
Thermal Margin:	-699.8°C (-60.3 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Power Report 2

۵ نتیجه‌گیری

در این گزارش ابتدا به عملیات ضرب ماتریسی و نحوه عملکرد و ساختار آن پرداختیم. سپس اعداد ممیز شناور با دقت یگانه را بررسی کردیم و عملیات ضرب و جمع بر روی این اعداد را شرح دادیم. در قسمت آخر مقدمه نیز به تعدادی از کاربردهای ضرب ماتریسی در علوم کامپیوتر، جبرخطی، هوش مصنوعی و حوزه‌های دیگر، اشاره کردیم. همچنین به اهمیت سریع و بهینه بودن الگوریتم و مدار ضرب ماتریسی در هوش مصنوعی و یادگیری عمیق اشاره شد. درنهایت، به این نتیجه رسیدیم که ضرب ماتریسی یکی از مهمترین عملیات‌های جبر خطی می‌باشد و کاربردهای بسیاری در علوم مختلف دارد.
در قسمت توصیف معماری سیستم، ابتدا به توصیف اینترفیس‌ها و ورودی و خروجی و پارامترهای مدار پرداختیم. سپس در مورد کلاک مدار و یکسان بودن کلاک کل مدار با هریک از ماژول‌ها بحث شد. دیاگرام بلوکی هر ماژول به همراه ماژول اصلی ترسیم شد در ادامه این قسمت گنجانده شد. پس از آن، به توصیف عملکرد هر یک از ماژول‌های مدار پرداختیم و هر مدار را بر اساس state هایی که واحد کنترل ماژول داشت، بررسی کردیم. و در انتهای این بخش، ساختار درختی مدار ترسیم شد.
در قسمت شبیه سازی، ابتدا به توصیف test bench ها پرداخته شد. پس از آن، برای هر یک از ماژول های اصلی یعنی adder, inner product, matrix multiplier تست بنچ همراه با نتایج حاصله ارائه گردید.
در قسمت پیاده سازی نیز، مدار سنتز شده قرار گرفت و به توضیح مساحت مدار، تعداد فلیپ فلاپ ها و LUT ، زمان بندی، کلاک و حداکثر فرکانس و همچنین توان مدار سنتز شده پرداخته شد. در این گزارش به معرفی و تحلیل مدار ضرب کننده طراحی شده پرداختیم و این شبیه سازی و پیاده سازی مدار را بررسی کردیم. در طراحی مدار سعی شد که از تمامی ویژگی‌ها و امکانات سخت‌افزاری وریلاگ استفاده شود. در نهایت لازم است که از آقای امیرمهدی نامجو (دانشجوی ترم پیش همین درس) به خاطر در اخیار قرار دادن قالب LaTeX گزارش کار، تشکر و سپاسگزاری نماییم.

مراجع

[1] Wikipedia. Matrix multiplication, 2021.

[2] Wikipedia. Floating point arithmetic, 2021.

[3] Wikipedia. Tensor processing unit, 2021.

[5] Mahmoodi, Ahmad. floating point operations slides, 2011.

[6] eedwards. Floating point numbers.

[7] Wikipedia. 3d projection, 2021.