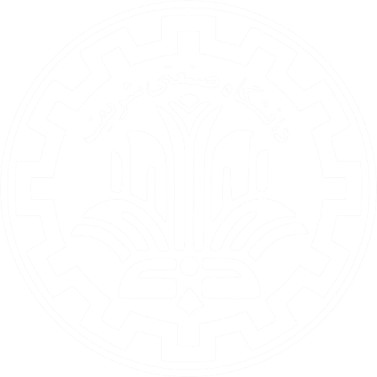
به نام خدا



**فاز سوم پروژه**

**طراحی سیستم های دیجیتال**

**استاد فلاحتی**

**اعضای تیم:**

سهیل نظری مندجین

علیرضا حقی

مهدی سعادت بخت

**نیمسال دوم سال تحصیلی ۱۴۰۰-۱۴۰۱**

**دانشگاه صنعتی شریف**

**دانشکده مهندسی کامپیوتر**

فهرست مطالب

[توضیحات پروژه 3](#_Toc110000303)

[چکیده 3](#_Toc110000304)

[مقدمه 4](#_Toc110000305)

[کاربردهای ضرب دو ماتریس 4](#_Toc110000306)

[روش‌های ضرب دو عدد 4](#_Toc110000307)

[تقسیم و غلبه 4](#_Toc110000308)

[درخت والاس 5](#_Toc110000309)

[ضرب‌کننده بران (آرایه‌ای) 6](#_Toc110000310)

[FSM and ASM 7](#_Toc110000311)

[الگوریتم ضرب ماتریس 9](#_Toc110000312)

[مرحله اول و دوم (پارتیشن‌بندی ماتریس به ساب‌بلاک‌های سایز ) 10](#_Toc110000313)

[مرحله سوم (ضرب ساب‌بلاک‌ها در هم و سپس جمع آن‌ها با هم) 10](#_Toc110000314)

[مرحله چهارم (شیفت دادن ماتریس a به راست و ماتریس b به پایین) 11](#_Toc110000315)

[ضرب ماتریس 11](#_Toc110000316)

[سنتز 12](#_Toc110000317)

[RTL Schematic 12](#_Toc110000318)

[Synthesis schematic 12](#_Toc110000319)

نحوه برخورد با اعداد اعشاری و تست برنامه................................................................................................13

# توضیحات پروژه

یک واحد پردازشی ضرب دو ماتریس را با استفاده از الگوریتم تقسیم و حل پیاده سازی کرده‌ایم. در این طراحی مولفه‌های ماتریس مطابق با استاندارد IEEE754 دریافت می‌شوند. این طراحی با استفاده از تقسیم و حل به گونه‌ای انجام شده که انجام عملیات ضرب ماتریس به صورت موازی و بهینه انجام شود.

# چکیده

با توجه به محاسبات سنگین محاسباتی ماتریسی در حوزه‌های مختلف، پردازش موازی ضرب ماتریس‌ها در راستای کمینه کردن میزان انرژی و مساحت مورد استفاده اهمیت ویژه‌ای پیدا کرده است.

در این پروژه، بنابر دغدغه‌ی الزام ضرب موازی ماتریس‌ها، به طراحی یک مدل تقریبا بهینه با استفاده از زبان برنامه‌نویسی Verilog بر اساس مساحت و تعداد ماژول‌ها رسیده‌ایم. در مدل ما از واحدهای محاسباتی استفاده شده است که قادر است در کلاک محاسبات موازی دو ماتریس را انجام دهد که در این جا بیانگر مجذور تعداد واحدهای پردازش موازی است. کدهای ما کاملا قابل سنتز است و این امر به‌وسیلهٔ نرم‌افزار سنتز Vivado صورت گرفته است. همچنین صحت کد و کارایی آن توسط آزمایش‌کننده‌ای توسط دو کد پایتون که یکی عدد رندوم تولید کرده و دیگری پاسخ ان را میابد صورت گرفته است. نکته‌ی حائز اهمیت دیگر مدل ما، پارامتریک بودن است که می‌توان به‌وسیلهٔ آن برای هر ابعاد دلخواه و هر تعداد دلخواه پردازنده آن را در نظر گرفت که در آن تعداد پردازنده مربع کامل است و ابعاد بر مجذور پردازنده‌ها بخش‌پذیر است.

# مقدمه

## کاربردهای ضرب دو ماتریس

ضرب دو ماتریس در علوم کامپیوتر کاربردهای گسترده‌ای دارد و از اساسی‌ترین دلایلی که سعی داریم تا این عمل را در پایین‌ترین سطح پردازشی، یعنی سطح ترانزیستور و چیپ انجام دهیم و از انجام آن به صورت نرم‌افزاری خودداری کنیم، کاربردهای گسترده آن است. از گسترده‌ترین استفاده‌های ضرب دو ماتریس می‌توان در گرافیک یاد کرد. هر تصویر دیجیتالی را می‌توان با یک ماتریس مدل کرد (همانطور که در حال حاضر نیز این امر صورت می پذیرد). هر خانه از این ماتریس را می‌توان حاوی اطلاعات یکی از پیکسل‌های این عکس در نظر گرفت که عدد آن‌ها نشان‌دهنده ترکیب رنگی آن پیکسل است. حال برای مثال، برای دیکود کردن یک ویدیو دیجیتالی، نیاز به ضرب ماتریس‌ها داریم. بسیار دیده‌ایم که برای مثال یک ویدیو را می‌توان از فرمت x254 به x255 دیکود کرد و این امر با توجه به بالا بودن نرخ فریم‌ها در یک ثانیه برای ویدیوهای جدید، اگر می‌خواست در سطح نرم‌افزار انجام می‌شد، زمان بسیار زیادی را از کاربر می‌گرفت. پژوهشگران MIT یکی از اولین چیپ‌های مخصوص (ASIC) و بهینه برای کد کردن ویدیوها که در تلویزیون‌های با کیفیت بالا استفاده می‌شود را با استفاده از بهینه‌سازی ضرب ماتریس‌ها خلق کرده‌اند. گرافیک در کامپیوتر با استفاده از اصطلاح پردازش دیجیتال سیگنال بیان می‌شود. از دیگر کاربردهای ضرب ماتریس‌ها در این زمینه می‌توان به تصویربرداری دیجیتال، پردازش سیگنال‌ها و چندرسانه‌ای اشاره کرد. همچنین در کاربردهای جدیدتر برای reinforce کردن یک یادگیری در بینایی ماشینی برای شتابنده‌های سخت‌افزاری هوش مصنوعی، ضرب ماتریس در مقیاس بالا استفاده می‌شود.

## روش‌های ضرب دو عدد

برای ضرب دو عدد، روش‌های متعددی وجود دارد و انتخاب هر کدام از آن‌ها، مسئله انتخاب بین سرعت و استفاده از سخت‌افزار کمتر است. به طوری که برخی از روش‌ها با وجود سرعت بالا در محاسبه، نیازمند تعداد بالایی منابع سخت‌افزاری هستند، اما برخی دیگر می‌توانند این کار را در زمانی طولانی‌تر اما با منابع سخت‌افزاری محدودتر انجام دهند. ۳ روش معروف برای این کار که به بررسی آن‌ها خواهیم پرداخت عبارتند از:

* تقسیم و غلبه (Conquer & Divide)
* درخت والاس (Tree Wallace)
* ضرب کننده بران (Multiplier Braun)

### تقسیم و غلبه

این روش بیشتر در FPGAها به کار می‌رود که همین امر باعث سرعت بیشتر می‌شود. به دلیل موازی‌سازی‌های که در این روش انجام می‌شود، مساحت به کار رفته در آن و همچنین قطعات مورد استفاده می‌توانند بیشتر شوند. این روش به روش کاراتسوبا نیز معروف است. طرز کار این الگوریتم به این صورت است که ابتدا هر کدام از دو عدد را به نیم تقسیم کرده. بر اساس تقسیم‌بندی‌ای که کردیم، عبارت نهایی پس از ضرب باید به شکل زیر باشد:

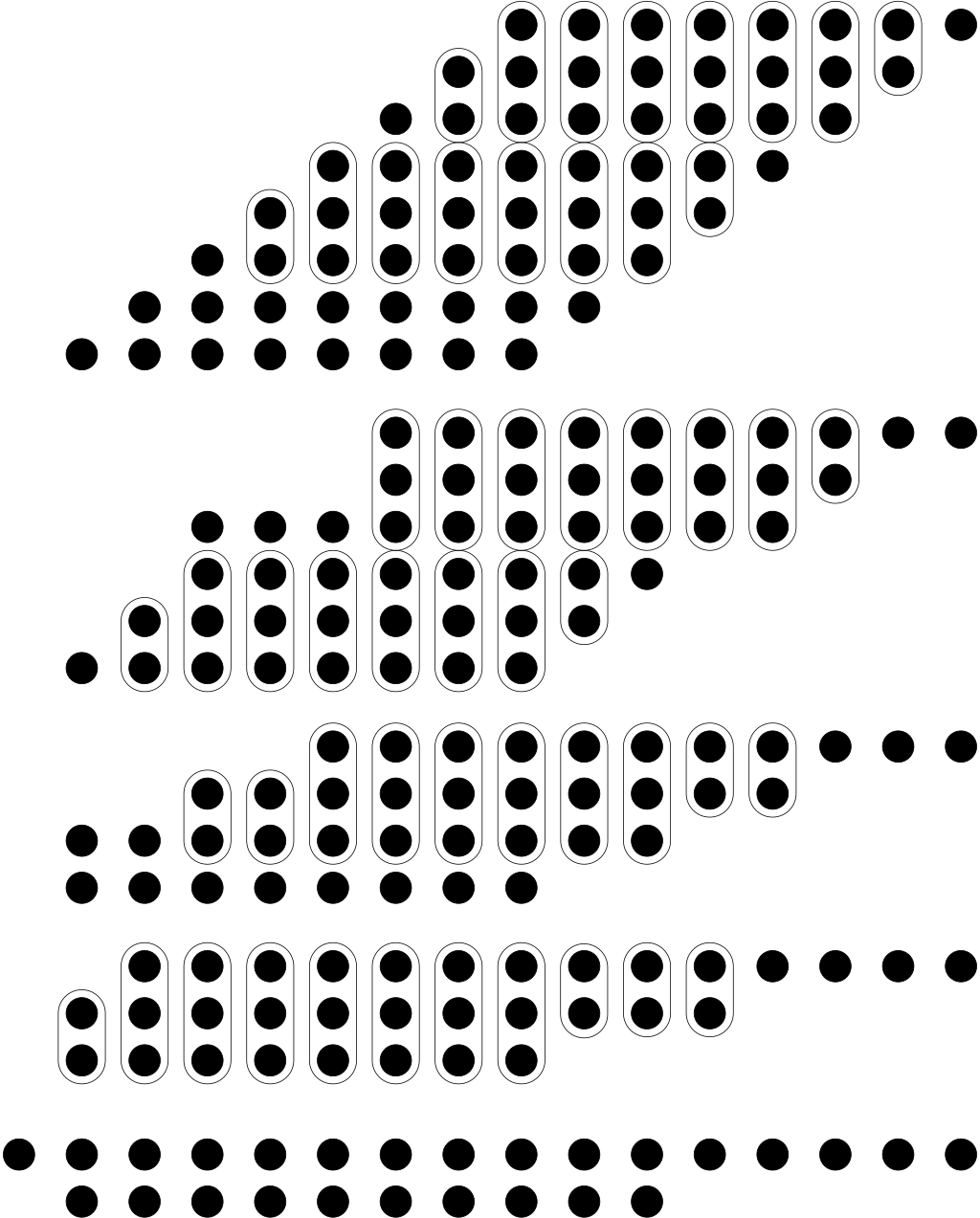
سپس ۳ عبارت زیر را محاسبه می‌کنیم:

1. 𝐴ℎ. 𝐵ℎ
2. 𝐴𝑙. 𝐵𝑙
3. (𝐴ℎ + 𝐴𝑙) . (𝐵ℎ + 𝐵𝑙)

بعد از محاسبه این ۳ عبارت، می‌توانیم به سادگی عبارت نهایی را محاسبه کنیم. پیچیدگی زمانی نهایی این محاسبات برای رسیدن به جواب نهایی، از مرتبه زمانی خواهد بود.

### درخت والاس

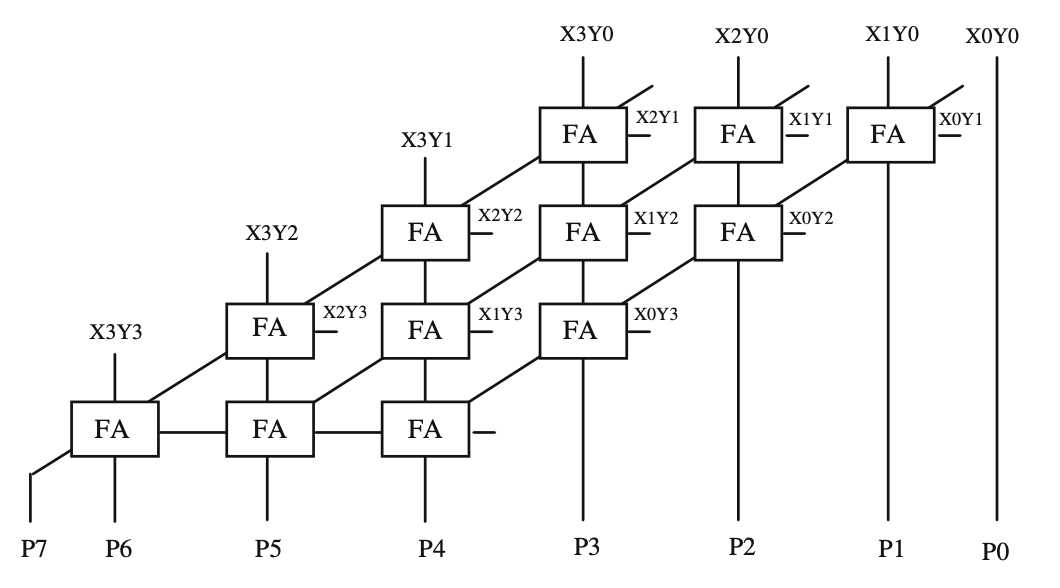
درخت والاس در هر سطح از ضرب، تعداد بیت‌های کمتری دارد ولی تاخیر به صورت موازی تقسیم نمی‌شود. با توجه به ارتباطات میانی بسیاری که درخت والاس دالاس دارد، مساحت این روش زیاد می‌شود. صورت کلی انجام این روش برمی‌گردد به جمع ریز-ضرب‌هایی که انجام می‌دهیم. شکل کلی آن در زیر قابل مشاهده است:



تصویر بالا مربوط به یک درخت والاس ۸\*۸ است که ۴ لایه از آن کاهش یافته است. هر کدام از نقطه‌ها نمایان‌گر یک بیت باارزش وزن یکسان هستند. این عمل کاهشی با استفاده از ۱۴ adder-half و ۳۸ adder-full انجام شده که adder-half ها با پکیج کردن دو نقطه نشان داده شده و adder-full ها با پکیج کردن سه نقطه. همان‌طور که در ابتدای این بخش نیز بیان شد، کاهش زمانی انجام یک عملیات (که در اینجا با کاهش لایه‌ها قابل مشاهده است)، به قیمت افزایش استفاده از منابع سخت‌افزاری است.

### ضرب‌کننده بران (آرایه‌ای)

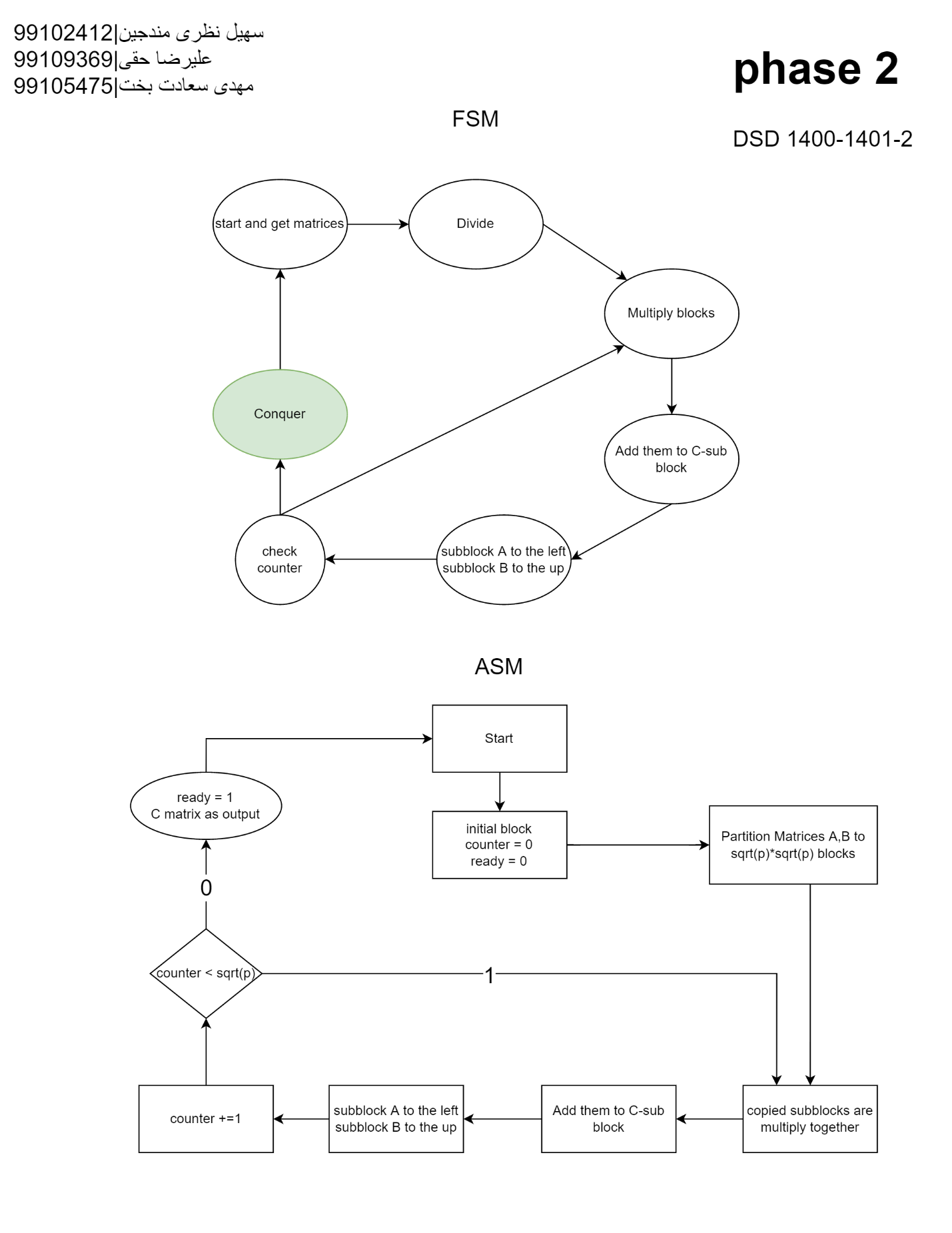
این روش برخلاف روش قبل، توزیع تاخیر یکسانی برای تمام سطوح دارد. این روش در اصل همان ضرب آرایه‌ای است که از دوران مدرسه آن را آموخته‌ایم. شکل آن به صورت زیر است:



این روش از بدترین پیچیدگی زمانی با مرتبه زمانی برخوردار است. اما مزیت استفاده از این روش، کاهش پیچیدگی پیاده‌سازی و استفاده از منابع سخت‌افزاری خواهد بود.

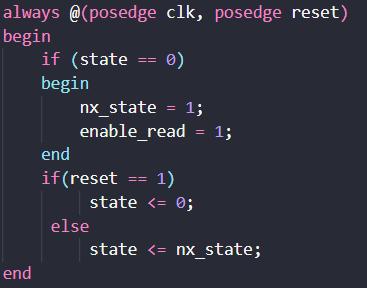
## FSM and ASM

در این بخش ما شمایی کلی از حالت و وضعیت‌های محاسباتی ممکن برای ماژول ضرب‌کننده نشان می‌دهیم که از ۷ حالت تشکیل شده است:

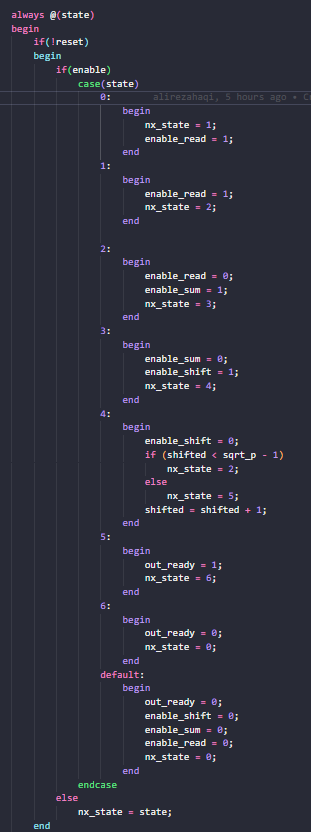


با استفاده از طراحی FSM انجام شده یک ماژول کنترلر تعریف کرده‌ایم که به وسیله آن استیت‌های مختلف ماشین ما تغییر کرده و سه بیت کنترلی برای خواندن، جمع کردن و شیفت دادن در اختیار داریم که میان استیت‌ها مقادیرشان را تنظیم می‌کنیم. نکتهٔ دیگر این است که ماشین کنترلی مد نظر از نوع Moore است. بنابراین به ورودی برای حرکت بین حالات حساس نیست.

تکه کد ریست و استارت در استیت صفر:



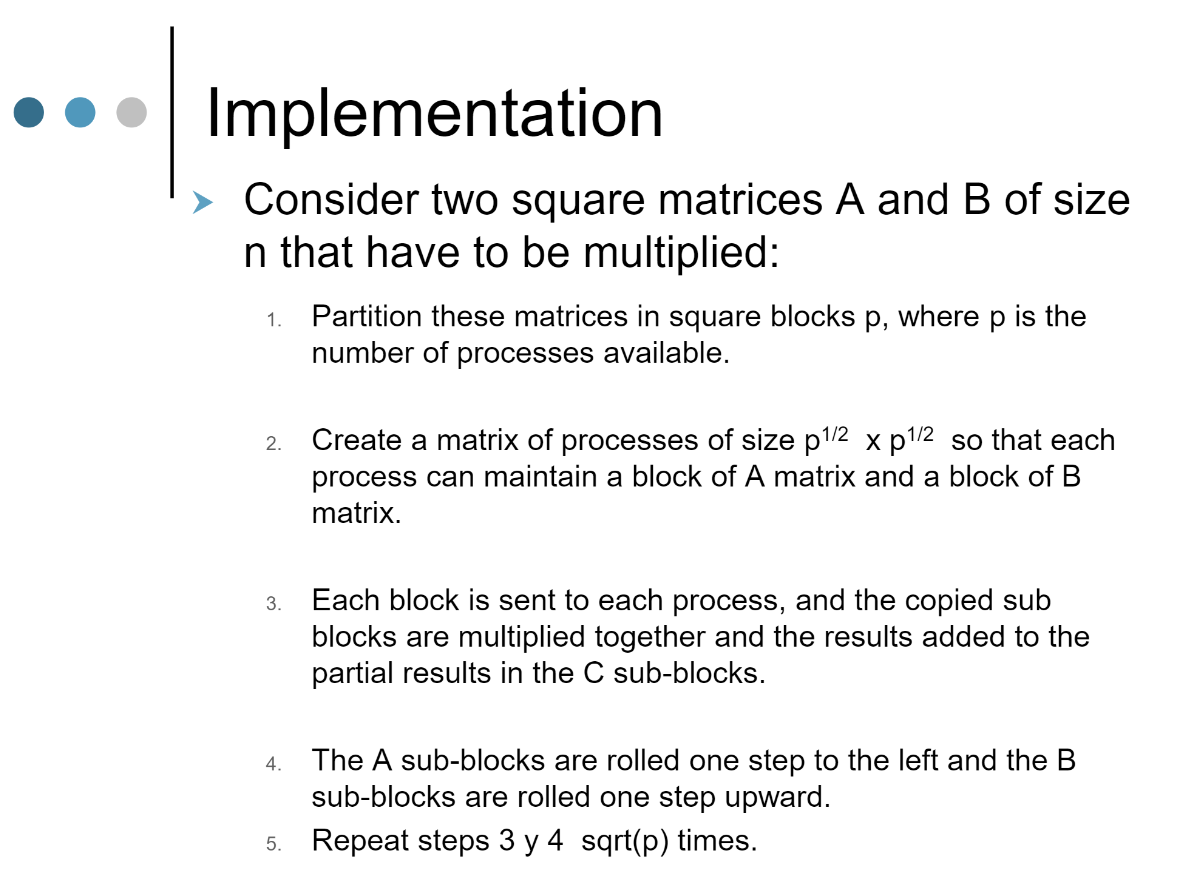
تکه کد انتقال میان استیت‌ها و ست کردن بیت‌های کنترلی:



# الگوریتم ضرب ماتریس

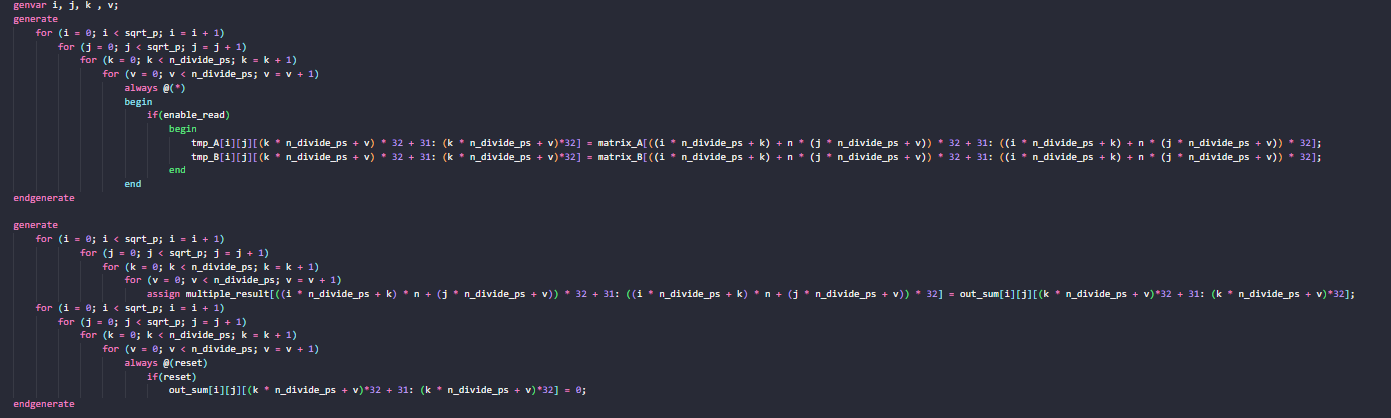
الگوریتم کلی بدین شکل است که هر دو ماتریس n \* n را به ماتریس با ابعاد تقسیم می‌کنیم. سپس در طی p مرحله با استفاده از شیفت و ضرب اجزای متناظر ماتریس‌های به‌دست آمده و در نهایت جمع آن‌ها جواب نهایی حاصل می‌شود. در عکس پایین نیز شبه‌کدی برای الگوریتم ارائه شده است ( را تعداد پردازنده‌ها است).

عملیات‌های ضرب دو ماتریس به شکل تقسیم و حل به ۵ مرحله تقسیم می‌شود که به شکل زیر است.

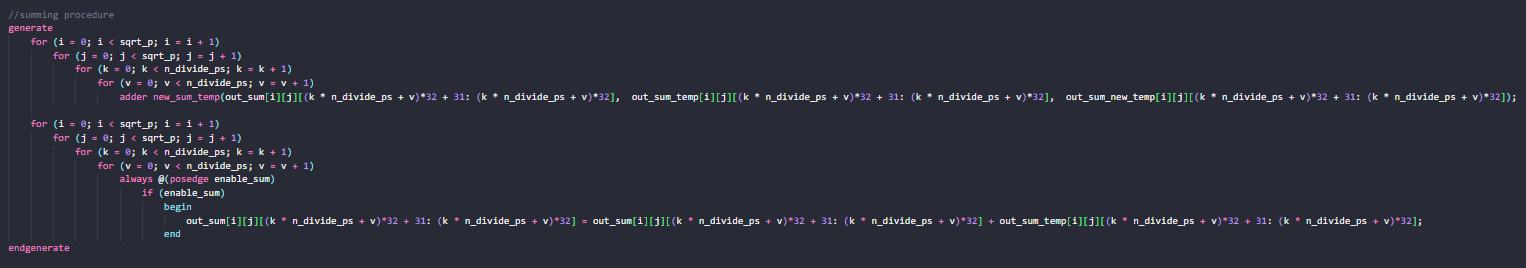


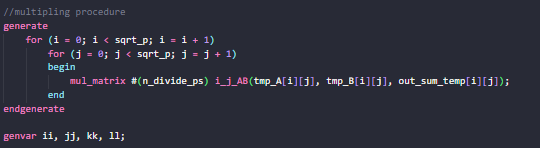
ما در یک ماژول به اسم array-divider این مراحل را با استفاده از جنریتورها پیاده‌سازی کرده‌ایم. تکه‌کد مراحل مختلف به ترتیب آمده‌اند. در این بخش به‌واسطهٔ جنریتورها توانستیم برنامهٔ کاملا کارا و پارامتریک که هیچ وابستگی به متغیر و عدد ثابتی ندارد را پیاده‌سازی کنیم که هر چند منجر به پیچیدگی‌های قابل ملاحظه‌ای در پیاده‌سازی کد شد.

## مرحله اول و دوم (پارتیشن‌بندی ماتریس به ساب‌بلاک‌های سایز )

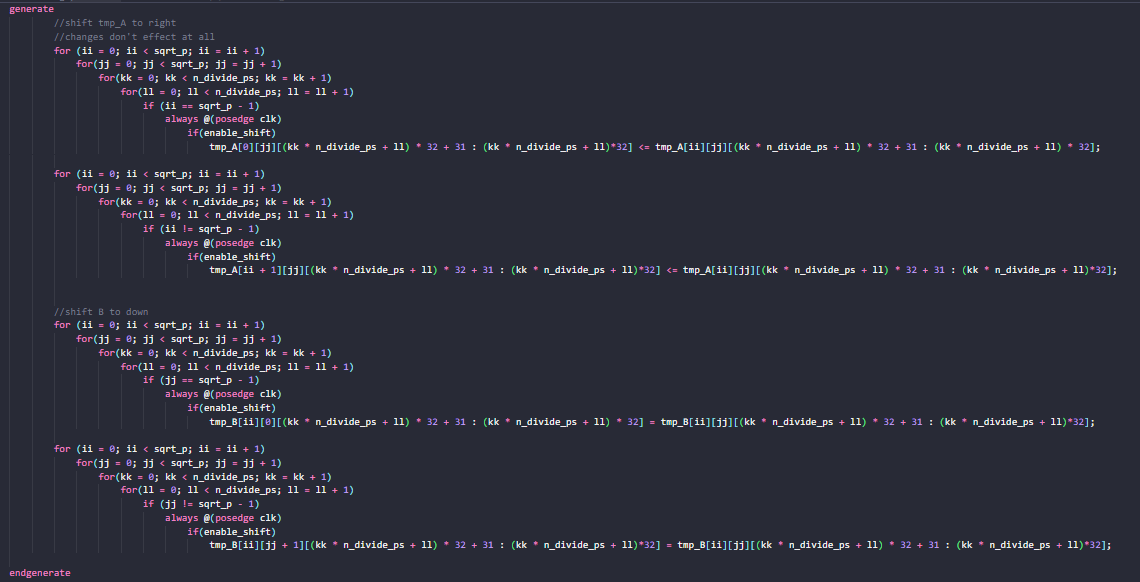


## مرحله سوم (ضرب ساب‌بلاک‌ها در هم و سپس جمع آن‌ها با هم)





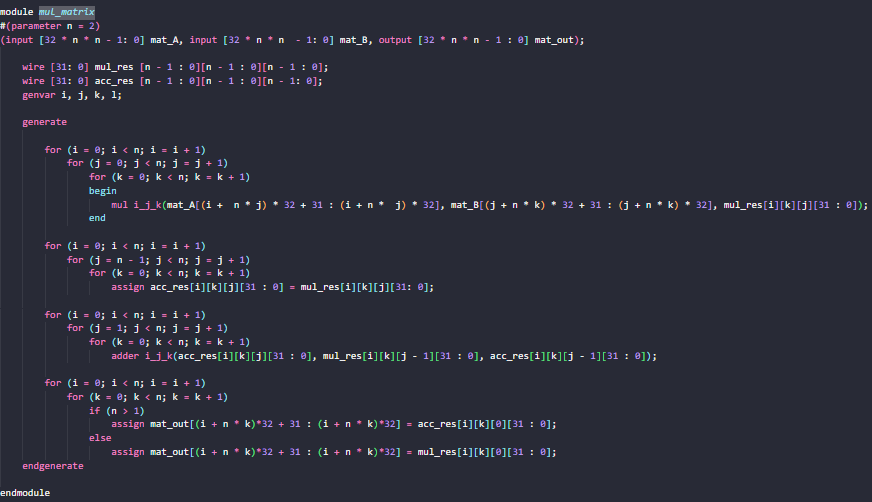
## مرحله چهارم (شیفت دادن ماتریس a به راست و ماتریس b به پایین)



مرحله پنجم نیز در تصاویر نشان داده است. همان‌طور که می‌بینید داخل یک حلقه به اندازه قرار دارند.

# ضرب ماتریس

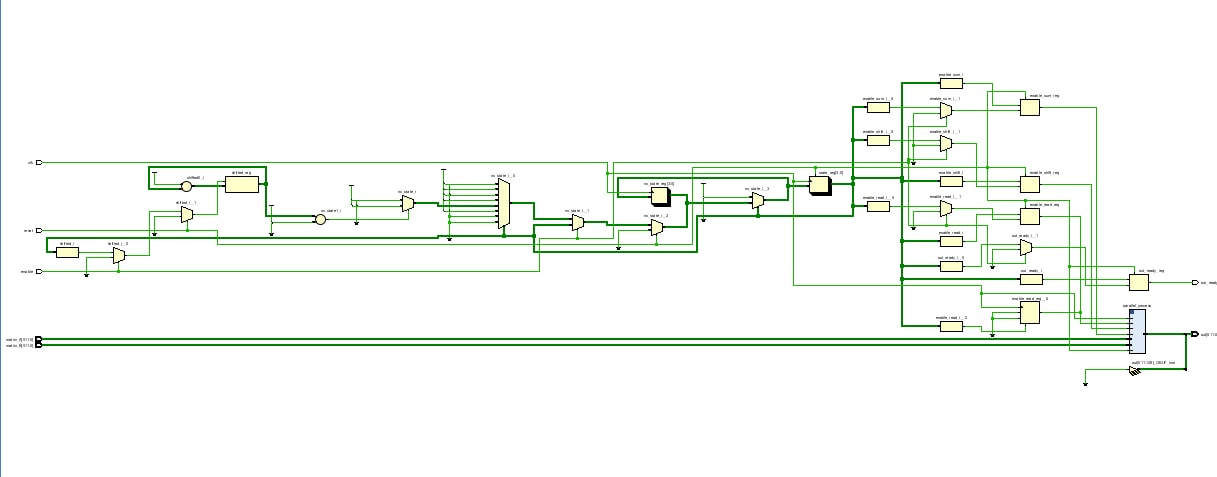
یکی دیگر از بخش‌های مورد نیاز برای پیاده‌سازی ضرب موازی ماتریس‌ها، ضرب دو ماتریس با ابعاد دلخواه است که در آن برای ضرب دو ماتریس از واحد ضرب و واحد جمع استفاده شده است که delay که در این عملیات وجود دارد برابر است با n برابر delay یک واحد جمع‌کننده به‌علاوهٔ یک واحد ضرب‌کننده. هر چند اگر بهینه‌تر پیاده‌سازی کرد، می‌توان این delay را به بار delay جمع‌کننده و یک delay ضرب‌کننده رساند که با توجه به بزرگ‌تر شدن n می‌توان به شکل قابل توجهی سرعت کلاک را کمتر کرد و در غیر این‌صورت می‌توان سرعت کلاک را پایین نگه داشت ولی تعداد دورهایی که کلاک در استیت ضرب کردن است بنابر آزمایشات زمانی به‌دست آورد.



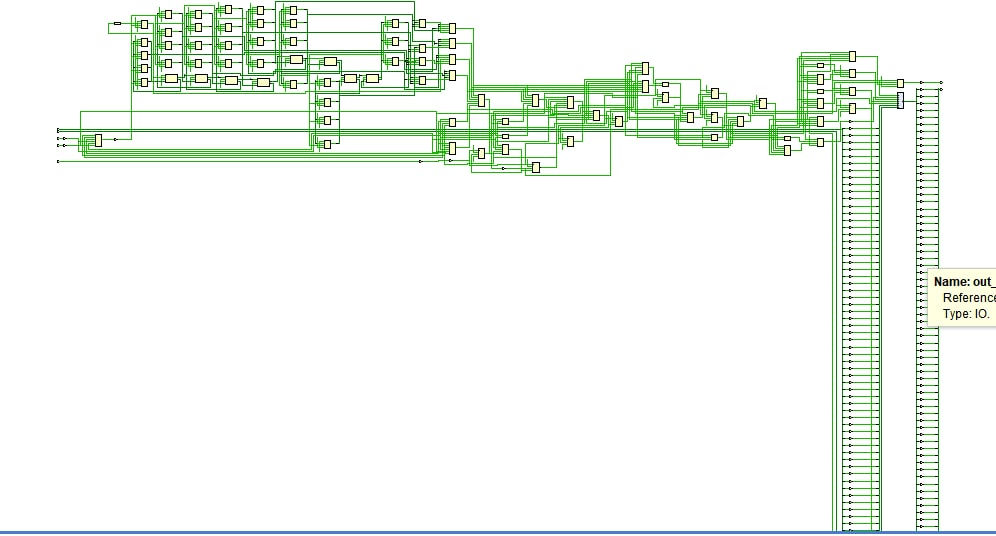
# سنتز

ما با استفاده از ابزار سنتز Vivado طراحی خود در وریلاگ را سنتز کرده و کدهای ما کاملا سنتزپذیر بوده و خروجی‌های آن به شکل زیر می‌باشد.

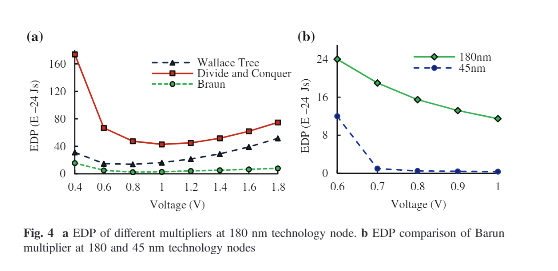
## RTL Schematic



## Synthesis schematic



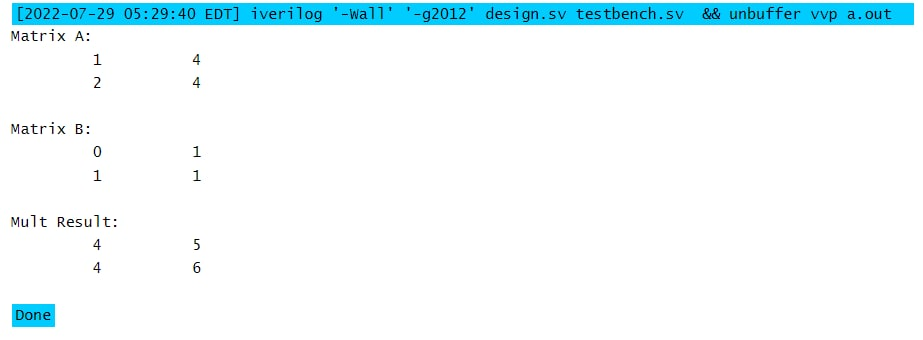
ما با استفاده از روش تقسیم و حل و همچنین موازی‌سازی عملیات‌های پیدا کردن هر یک از درایه‌های ماتریس خروجی در توان مصرفی نهایی قطعه خود صرفه جویی بسیاری انجام داده‌ایم. طبق آزمایش‌ها و محاسباتی که بر روی این قضیه انجام شده است، جدول‌های زیر برای سه الگوریتم مختلف انجام این کار نشان داده شده است.



**EDP**

The proposed work uses energy-delay product (EDP), where energy the total energy consumption of cores and delay is the amount of time for executing applications.

نتیجه یکی از تست های امتحان شده با استفاده از طراحی ما:



صحت سنجی به کمک گلدن مادل:



**نحوه برخورد با Floating point:**فایل های تحت عنوان جمع کننده و ضرب کننده خودکار اعداد اعشاری به ما داده شده بود که متاسفانه این فایل ها مشکلاتی داشتند. ابتدا از همه با توجه به ترکیبی بودن کل فرایند ضرب ماتریس در برنامه ما کلاک داشتن و ترتیبی بودن آن ها کار را خراب میکرد دوما به هیچ عنوان قابل سنتز نبودند و سوم ایرادات ریز سینتکسی داشتند که قادر به حل آن مشکلات در زمان کوتاه باقی مانده نبودیم فلذا مجبور شدیم خودمان یک جمع کننده و ضرب کننده اعداد اعشاری بالا بیاوریم ک آن ها را در دو ماژول ترکیبی جدا پیاده سازی کرده و با استفاده از وریلاتور و تست های CPP درستی آن ها را مشخص کردیم.

متاسفانه با توجه به زمان اندک باقی مانده تا ارائه نتوانتسیم تست خودکار برنامه را راه بیندازیم ولی با توجه به پاسخگویی های دستی برنامه و همچنین پاسخ های درست آن در استیت های قبلی و همچنین درستی تست ها میتوان از پاسخ ارسالی برنامه اطمینان حاصل کرد.