دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر و فن‌آوری اطلاعات

پایان‌نامه کارشناسی

گرایش سخت‌افزار

عنوان

محاسبه حالت‌های پایه‌ای در شبکه‌های متابولیکی با استفاده از روش دودویی

نگارش

مهشید علی‌نوری

استاد راهنما

دکتر مرتضی صاحب‌الزمانی

تیر ۱۳۹۷

به نام خدا

تعهدنامه اصالت اثر

تاریخ:

اینجانب مهشید علی‌نوری متعهد می‌شوم که مطالب مندرج در این پایان‌نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است، مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است.

مهشید علی‌نوری

امضا

چکیده

ویژگی‌های فیزیولوژیکی و بیوشیمیایی هر سلول با استفاده از مجموعه‌ای از فرآیندهای فیزیکی و متابولیکی توصیف می‌شود. این فرایندها با شبکه‌های متابولیکی[۱] نمایش داده می‌شود. امروزه در علومی چون بیوتکنولوژی مدل‌سازی شبکه‌های متابولیکی در بسیاری از تحقیقات و آزمایش‌ها کاربرد دارد. برای انجام مدل‌سازی، نیازمند شناسایی پارامترهایی برای توصیف عملکرد شبکه هستیم. حالت‌های پایه‌ای[۲] از جمله این پارامترها هستند و از این رو محاسبه آنها در مدل‌سازی شبکه از اهمیت زیادی برخوردار است.

در این پروژه به محاسبه دودویی[۳] حالت‌های پایه‌ای می‌پردازیم. این روش حالت‌های پایه‌ای را به صورت الگوهایی دودویی از واکنش‌های شرکت‌کننده محاسبه می‌کند. در روش دودویی، محاسبات بر روی داده‌هایی به شکل ۰ و ۱ انجام می‌شوند. در نهایت حالت‌های پایه‌ای به شکل بردارهایی شامل مقادیر ۰ و ۱ نمایش داده شده و با استفاده از آنها مقادیر حقیقی محاسبه می‌شوند. در این پروژه، برای بهره‌برداری هرچه بیشتر از امکاناتی که این روش در اختیار ما قرار می‌دهد، پیاده‌سازی این روش را با استفاده از طراحی توامان سخت‌افزار و نرم‌افزار[۴] انجام می‌دهیم. پس از پیاده‌سازی همه بخش‌های الگوریتم دودویی که برخی از آنها به صورت سخت‌افزاری و برخی به صورت نرم‌افزاری انجام می‌گیرند، نتیجه را بر روی زدبورد[۵] اجرا می‌کنیم و برای اطمینان از صحت خروجی، نتیجه سیستم پیاده‌سازی شده را با نتیجه ابزارهای استاندارد همچون متاتول[۶] مقایسه می‌کنیم.

در این گزارش جزئیات مراحل طراحی و پیاده‌سازی الگوریتم دودویی برای محاسبه حالت‌های پایه‌ای و نحوه صحت‌سنجی این پیاده‌سازی شرح داده می‌شود.

---

[1] metabolic network

[2] elementary modes

[3] binary

[4] hardware software co-design

[5] ZedBoard

[6] Metatool

واژه‌های کلیدی:

شبکه‌های متابولیکی، حالت‌های پایه‌ای، روش دودویی، طراحی توامان سخت‌افزار و نرم‌افزار

صفحه

فهرست عناوین

فهرست اشکال **صفحه**

۱

فصل اول

مقدمه

۱.۱ شرح مسئله

ویژگی‌های فیزیولوژیکی و بیوشیمیایی هر سلول با استفاده از مجموعه‌ای از فرآیندهای فیزیکی و متابولیکی توصیف می‌شود. این فرایندها با شبکه‌های متابولیکی نمایش داده می‌شود. هر شبکه شامل تعدادی مسیرهای سوخت‌وساز[1] می‌باشد که هر کدام از مجموعه‌ای به هم پیوسته از واکنش‌های شیمیایی تشکیل شده است.

بازسازی شبکه‌های متابولیکی از جمله موضوعاتی است که در صنعت بیوتکنولوژی و علم میکروبیولوژی بسیار مورد توجه قرار دارد. از سال ۱۹۱۷، از میکروب‌ها برای تولید صنعتی آنزیم‌ها و دیگر بیومولکول‌ها استفاده شده است. وجود مدل‌های دقیق و پویا برای متابولیسم‌های میکروبی امکان بهینه‌سازی در این مسیر را فراهم می‌کند. روش‌هایی همچون تحلیل تعادل شار[2] به محققان این امکان را می‌دهد تا به صورت بهینه به تولید، کاهش و دیگر اعمال بر روی مولکول‌ها بپردازند[۱].

تحلیل تعادل شار روشی است که در آن از بهینه‌سازی خطی استفاده می‌شود تا مقادیر شار به لبه‌های شبکه متابولیکی تخصیص داده شوند. در این روش مسیرها را می‌توان اصطلاحا به حالت‌های پایه‌ای تجزیه کرد. این حالت‌های پایه‌ای حداقل اجزایی هستند که در مسیر متابولیک و در شرایط پایدار می‌توانند مستقل و منسجم عمل کنند. ابزارهایی همچون متاتول تحلیل‌هایی را در سطح متوسط و به صورت نرم‌افزاری انجام می‌دهند. به عبارت دیگر در شبکه‌های متابولیکی، طبیعی‌ترین معیار برای وزن یک واکنش، شار آن است. شار نرخ تبدیل یک واکنش‌دهنده به فراورده‌هایش می‌باشد. در بسیاری از روش‌های محاسباتی، با فرض اینکه واکنش‌های متابولیک در حالت تعادل هستند، مجموعه‌ای از معادلات برای شارهای متابولیک نوشته می‌شود. منظور از تعادل در واکنش‌ها این است که واکنش‌دهنده‌ها در حالت پایا باشند. در نتیجه مجموعه‌ای از معادلات جبر خطی برای شارها تولید می‌شود. به طور کلی این معادلات غیرقابل حل هستند زیرا تعداد متغیرها از تعداد معادله‌ها بیشتر است. برای کوچک‌تر کردن فضای پاسخ، محدودیت‌های بیولوژیکی و شیمیایی را در معادلات اعمال می‌کنند. این محدودیت‌ها ممکن

[1] metabolic pathway

[2] flux balance analysis

است با استفاده از داده‌های تجربی مثلا با کمک شارهای قابل اندازه‌گیری به دست بیایند. دیگر محدودیت‌ها ممکن است توسط قوانین ترمودینامیک در طبیعت مشخص شوند[۱].

پس به طور کلی، برای انجام مدل‌سازی، نیازمند شناسایی پارامترهایی برای توصیف عملکرد شبکه هستیم. از این رو محاسبه حالت‌های پایه‌ای شبکه به عنوان یک ویژگی مجزا در مدل‌سازی شبکه از اهمیت زیادی برخوردار است. حالت‌های پایه‌ای، بردارهای شار در حالت تعادل شبکه متابولیک با مجموعه حداقلی واکنش‌های فعال هستند و نرخ تحول مولکول‌ها را در یک مسیر متابولیک مشخص می‌سازند.

۲.۱ روش حل مسئله

اگرچه محاسبه حالت‌های پایه‌ای از اهمیت بسیاری برخوردار است، ولی در عمل کار محاسباتی دشواری است. تا کنون الگوریتم‌های متعددی برای این کار پیشنهاد شده‌اند. روش فضای پوچ[۱] و روش پایه کانونی[۲] از جمله برجسته‌ترین روش‌ها برای انجام این محاسبات است. البته روش فضای پوچ نسبت به روش پایه کانونی از سرعت بالاتری برخوردار است. این در حالیست که پیاده‌سازی آن حافظه زیادی مصرف می‌کند و از این بابت محاسبات را بخصوص برای شبکه‌های بزرگ با محدودیت روبرو می‌کند. در روش فضای پوچ همانند روش معروف مسیرهای فرین[۳]، فضای پاسخ مسئله به صورت یک مخروط است که حالت‌های پایه‌ای لبه‌های آن هستند که به هر یک از آنها شعاع فرین[۴] گفته می‌شود. این مخروط حاصل اعمال محدودیت‌هایی به فضای پاسخ مسئله است که در بخش قبل به آنها اشاره شد.

روش دودویی که در این پروژه به پیاده‌سازی آن پرداخته می‌شود، حالت‌های پایه‌ای را به صورت الگوهایی دودویی از واکنش‌های شرکت‌کننده محاسبه می‌کند. این روش بر پایه روش فضای پوچ شکل

---

[1] nullspace approach

[2] canonical basis approach

[3] extreme pathways

[4] extreme ray

گرفته است. در روش فضای پوچ محاسبات وابسته به مقادیر حقیقی ضرایب در شبکه متابولیکی هستند. ولی در روش دودویی، محاسبات بر روی داده‌هایی به شکل ۰ و ۱ انجام می‌شوند و در نهایت حالت‌های پایه‌ای به شکل بردارهایی شامل مقادیر ۰ و ۱ محاسبه می‌گردند.

در روش دودویی با روش‌هایی همچون جداسازی واکنش‌های رفت و برگشت در یک واکنش برگشت‌پذیر محاسبات را ساده‌سازی می‌کند و در این صورت مصرف حافظه نیز کاهش می‌یابد. برای استفاده از سرعت بالایی که اجرا بر روی سخت‌افزار در اختیار ما قرار می‌دهد، بخشی از روش دودویی را به صورت سخت‌افزاری پیاده‌سازی می‌کنیم. البته در مراحل ابتدایی و پایانی این الگوریتم، محاسبات ریاضی و جبر خطی وجود دارد که بهتر است انجام آنها با استفاده از نرم‌افزار صورت بگیرد. از آنجایی که این محاسبات تنها یک بار انجام می‌گیرند، می‌توان ادعا کرد که تاثیر قابل توجهی بر سرعت اجرای الگوریتم ندارند. بدین ترتیب قالب کلی پروژه به صورت طراحی توامان نرم‌افزار و سخت‌افزار خواهد بود.

در ادامه این گزارش، در فصل ۲، مفاهیم استفاده شده در پروژه توضیح داده خواهند شد که در این بین توصیف دقیق‌تر الگوریتم دودویی هم وجود دارد. در فصل ۳، جزئیات طراحی و پیاده‌سازی هر بخش از پروژه به همراه چالش‌ها توضیح داده خواهد شد. در فصل ۴، نتایج به دست آمده از پروژه نشان داده خواهند شد و با نتایج مورد انتظار مقایسه خواهند شد. در پایان نیز یک جمع‌بندی اجمالی از پروژه و پیشنهادهایی برای بهبود آن ارائه خواهد شد.

۲

فصل دوم

مفاهیم پایه

۱.۲ طراحی توامان سخت‌افزار و نرم‌افزار

برای پیاده‌سازی یک سیستم دیجیتال سه روش کلی وجود دارد. یک روش استفاده از زبان‌های سطح بالای نرم‌افزاری همچون C و C++ و بهره بردن از ویژگی‌هایی چون انعطاف‌پذیری در آنهاست. روش دوم استفاده از طراحی و پیاده‌سازی سخت‌افزاری و استفاده از آرایه‌های منطقی برنامه‌پذیر در میدان یا FPGAها است. در این روش، امکان موازی‌سازی و اجرای همزمان عملیات محاسباتی وجود دارد و از این رو به سرعت اجرای عملیات افزوده می‌شود. روش سوم، طراحی توامان سخت‌افزار و نرم‌افزار نام دارد که حاصل ترکیب روش اول و دوم می‌باشد. بدین ترتیب باید در مرحله طراحی سیستم تصمیم بگیریم که چه بخش‌هایی از سیستم به صورت نرم‌افزاری و چه بخش‌هایی به صورت سخت‌افزاری پیاده‌سازی شوند. به طور کلی برای حذف پیچیدگی‌هایی که پیاده‌سازی سخت‌افزاری به برخی از بخش‌های سیستم تحمیل می‌کند، از پیاده‌سازی نرم‌افزاری استفاده می‌کنیم. در کنار آن، در بخش‌هایی که پیاده‌سازی سخت‌افزاری هزینه و سربار ایجاد نمی‌کند، برای بالا بردن سرعت انجام عملیات از این شیوه بهره می‌بریم. پس از تعیین شیوه پیاده‌سازی هر بخش، لازم است تا با انتخاب زبان مناسب نرم‌افزاری و زبان توصیف سخت‌افزار به پیاده‌سازی هر یک بپردازیم.

در طراحی این پروژه از شیوه طراحی توامان سخت‌افزار و نرم‌افزار استفاده شده است. پیاده‌سازی بخش نرم‌افزاری این پروژه با استفاده از زبان C و پیاده‌سازی بخش سخت‌افزاری با استفاده از زبان VHDL انجام شده است.

VHDL یک زبان توصیف سخت‌افزار است که برای توصیف سیستم‌های دیجیتال همچون FPGAها و مدارهای مجتمع کاربرد دارد. VHDL این امکان را می‌دهد تا پیش از تبدیل شدن سیستم طراحی شده به اجزای واقعی سخت‌افزار مانند گیت و سیم، توسط ابزار سنتز کننده، بتوان آن را مدل‌سازی و شبیه‌سازی کرد. همچنین این زبان امکان توصیف همروند را فراهم می‌کند. همچنین پروژه‌های پیاده‌سازی شده توسط VHDL قابلیت استفاده شدن مجدد در دیگر پروژه‌ها را دارند.

۲.۲ مدلسیم[1]

مدلسیم یک محیط شبیه‌سازی چند زبانه برای زبان‌های توصیف سخت‌افزار از جمله VHDL، Verilog و SystemC است. این نرم‌افزار می‌تواند به صورت جداگانه و یا در کنار برنامه‌های دیگری چون Xilinx ISE و ویوادو مورد استفاده قرار بگیرد. شبیه‌سازی در این محیط می‌تواند با استفاده از واسط گرافیکی و یا اسکریپت انجام شود. در صورتی که از این ابزار به تنهایی استفاده شود، عمدتا هدف انجام شبیه‌سازی‌های پیش از مرحله سنتز می‌باشد. در واسط گرافیکی این ابزار شکل موج سیگنال‌ها، متغیرهای درون فرآیندها و درگاه‌های ورودی و خروجی قابل مشاهده هستند و می‌توان برای انجام مراحل اشکال‌زدایی نیز از آن بهره برد.

۳.۲ محیط توسعه یکپارچه ویوادو[2]

ویوادو یک مجموعه نرم‌افزار ارائه شده توسط شرکت زایلینکس[3] می‌باشد که از آن برای تحلیل و طراحی یک سیستم با استفاده از زبان‌های توصیف سخت‌افزار استفاده می‌شود. از جمله امکاناتی که این نرم‌افزار در اختیار توسعه‌دهندگان قرار می‌دهد می‌توان موارد زیر را نام برد:

- سنتز کردن طراحی
- انجام تحلیل‌های زمانی
- شبیه‌سازی پیش و پس از سنتز
- ایجاد و بسته‌بندی[4] IPها
- سنتز سطح بالا برای زبان‌های برنامه‌نویسی C، C++، SystemC.

---

[1] Modelsim

[2] Vivado IDE

[3] Xilinx

[4] packaging

ویوادو تنها محصولات FPGA زایلینکس را پشتیبانی می‌کند و تراشه‌های آن را می‌شناسد و امکان استفاده از آن برای محصولات دیگر تولیدکنندگان وجود ندارد. همچنین شبیه‌سازی در ویوادو برخلاف محیط ISE که مبتنی بر شبیه‌ساز مدلسیم است، خود دارای یک شبیه‌ساز جداگانه است. البته به هنگام مشخص کردن ابزار شبیه‌سازی در محیط این برنامه امکان استفاده از شبیه‌ساز مدلسیم وجود دارد. در مراحل پیاده‌سازی این پروژه از نسخه ۱۷/۴ این نرم‌افزار استفاده شده است[۵].

۴.۲ کیت توسعه نرم‌افزار زایلینکس[1]

SDK یک محیط توسعه یکپارچه برای تولید سکوهای[2] نرم‌افزاری و برنامه‌های کاربردی با هدف استفاده از پردازنده‌های نهفته زایلینکس از جمله زینک[3]، زینک ۷۰۰۰[4] و مایکروبلیز[5] می‌باشد. این محیط علاوه بر اینکه به صورت جداگانه در دسترس و قابل استفاده است، در محیط ویوادو هم قرار گرفته و می‌تواند با محیط طراحی سخت‌افزار ویوادو ارتباط برقرار کند. ویرایشگر این برنامه بر اساس محیط Eclipse طراحی شده و در آن می‌توان با استفاده از زبان‌های C و C++ مراحل کدنویسی، کامپایل و اشکال‌زدایی را انجام داد. در مواردی که طراحی توامان سخت‌افزار و نرم‌افزار وجود داشته باشد، توسعه بخش نرم‌افزاری و اتصال آن با طراحی سخت‌افزار را می‌توان از طریق این برنامه انجام داد[۶].

---

[1] Xilinx Software Development Kit

[2] platform

[3] Zynq

[4] Zynq-7000

[5] Microblaze

۵.۲ زدبورد

زدبورد یک بورد شامل سیستم روی یک تراشه[1] زینک ۷۰۰۰ می‌باشد. این بورد همه امکانات لازم برای ایجاد طراحی‌های مبتنی بر لینوکس، اندروید، ویندوز و دیگر سیستم‌های عامل را داراست. خانواده سیستم روی یک تراشه زینک ۷۰۰۰، با اجتماع برنامه‌پذیری نرم‌افزاری پردازنده‌های آرم و برنامه‌پذیری سخت‌افزاری FPGAها بر روی یک دستگاه، امکان استفاده همزمان از امکانات سخت‌افزار و نرم‌افزار را فراهم می‌آورد. زینک ۷۰۰۰ شامل یک پردازنده دو هسته‌ای آرم کورتکس ای ۹[2] و یک منطق برنامه‌پذیر[3] آرتیکس ۷[4] یا کینتکس ۷[5] می‌باشد. با توجه به این مشخصات، دستگاه‌های زینک ۷۰۰۰ امکان ایجاد طراحی گستره وسیعی از سیستم‌های نهفته را فراهم می‌کنند[۷].

از دیگر امکانات زدبورد موارد زیر را می‌توان نام برد[۷]:

- چندین صفحه نمایش (1080p HDMI, 8-bit VGA, 128 x 32 OLED)
- تبدیل اتصالات USB OTG و USB-UART
- امکان برنامه‌ریزی با استفاده از اتصال USB-JTAG
- پورت‌های ورودی/خروجی عام منظوره شامل ۸ عدد LED، ۷ عدد دکمه
- سیگنال ساعت با فرکانس ۳۳,۳۳ مگاهرتز برای سیستم پردازش‌گر و نوسان‌ساز ۱۰۰مگاهرتزی برای منطق برنامه‌پذیر

[1] System on a Chip (SoC)

[2] ARM Cortex-A9

[3] Programmable Logic

[4] Artix-7

[5] Kintex-7

۶.۲ پروتکل UART

UART یکی از مهم‌ترین پروتکل‌های ارتباط سریال است که به صورت ناهمگام کار می‌کند و کنترل‌کننده آن در سیستم‌های کامپیوتری به عنوان جزء اصلی ارتباط سریال شناخته می‌شود. بسیاری از میکروکنترلرها و سیستم‌های دیجیتالی و سیستم‌های نهفته نیز UART را پشتیبانی می‌کنند. طبق این پروتکل، در ارتباطی که میان فرستنده و گیرنده برقرار می‌شود، بیت‌های هر بایت داده به صورت ترتیبی انتقال پیدا می‌کنند. از آنجایی که این پروتکل ناهمگام است، سیگنال ساعت از فرستنده به گیرنده فرستاده نمی‌شود و از این رو پیش از ارسال داده باید توافقی میان آنها برقرار شود. از جمله پارامترهایی که باید در هر دو طرف اتصال به صورت یکسان تنظیم شود نرخ ارسال، نوع بیت توازن[۱] و تعداد بیت‌های پایانی[۲] است[۸].

زدبورد اتصال UART-USB را پشتیبانی می‌کند و برای انتقال داده میان این بورد و سیستم کامپیوتری می‌توان از آن استفاده کرد و داده‌ها را از سیستم پردازش‌گر[۳] بورد به سیستم کامپیوتری انتقال داد. برای برقراری ارتباط با پورت سریال در سیستم کامپیوتری می‌توان از نرم‌افزارهای ترمینال پورت سریال مانند CoolTerm استفاده کرد. در این نرم‌افزار امکان تشخیص خودکار پورت سریال و خواندن و نوشتن در آن با نرخ تبادل داده‌های مختلف وجود دارد[۸].

۷.۲ واسط AXI4

AXI بخشی از خانواده باس آمبای آرم[۴] می‌باشد که اولین بار در سال ۱۹۹۶ معرفی شد. اولین نسخه AXI در سال ۲۰۰۳، در AMBA 3.0 قرار داده شد. در سال ۲۰۱۰ دومین نسخه اصلی AXI با نام AXI4 در AMBA 4.0 استفاده شد. در حال حاضر سه نوع واسط AXI4 موجود می‌باشد[۹]:

---

[1] parity bit

[2] stop bit

[3] Processing System (PS)

[4] ARM AMBA

- AXI4: مورد استفاده برای نیازهایی با پردازش بالا و با استفاده از نگاشت حافظه
- AXI4-Lite: مورد استفاده برای ارتباطات ساده و گذردهی پایین
- AXI4-Stream: برای جریان‌های داده با سرعت بالا

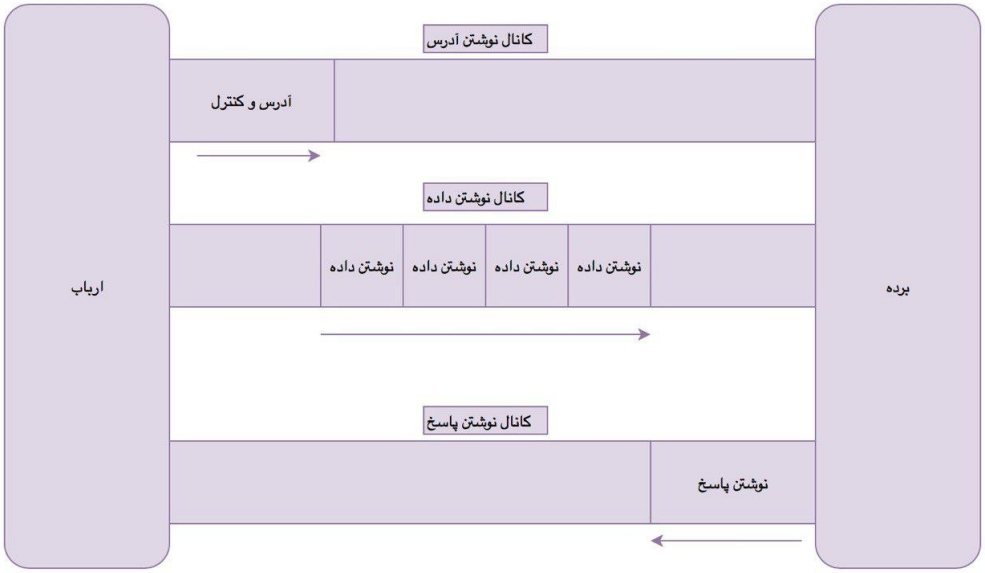
AXI4 امکانات زیادی همچون باس داده و آدرس با پهنای قابل تغییر، امکانات پیشرفته برای نهان‌سازی[1] داده‌ها و کامل کردن تراکنش‌های نامرتب را در اختیار کاربران قرار می‌دهد. اگرچه این امکانات بر قدرت کنترل کاربر می‌افزاید ولی اغلب یک پیاده‌سازی ساده‌تر که تنها تعدادی از این ویژگی‌ها را داشته باشد مطلوب‌تر است. از این رو در AXI4-lite تراکنش‌ها به صورت خیلی ساده‌تر قابل انجام هستند[۱۰].

AXI به عنوان یک واسط، ارتباط را بین ارباب[2] و برده[3] برقرار می‌کند. در هردو نوع full و lite این واسط، پنج کانال مختلف وجود دارد که عبارتند از کانال آدرس خواندن، کانال آدرس نوشتن، کانال داده خواندن، کانال داده نوشتن و کانال پاسخ نوشتن. داده‌ها می‌توانند همزمان در هر دو جهت بین ارباب و برده منتقل شوند و اندازه انتقال داده هم می‌تواند در هر جهت متفاوت از دیگری باشد. AXI4 و AXI4-lite از جمله پروتکل‌های مبتنی بر نگاشت روی حافظه می‌باشند که در آنها تراکنش به صورت انتقال داده مربوط به یک آدرس خاص از فضای حافظه می‌باشد[۹]. شکل ۲-۱ معماری کانال خواندن و شکل ۲-۲ معماری کانال نوشتن را در AXI4 نشان می‌دهد.

[1] cache

[2] master

[3] slave

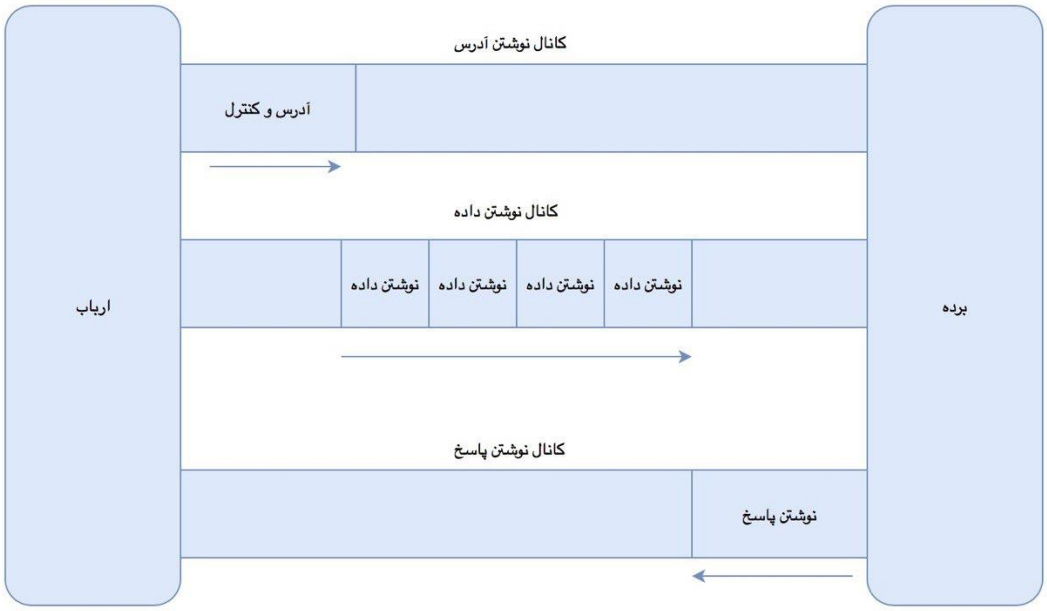


شکل ۲-۱ معماری کانال خواندن در AXI4 [۹]

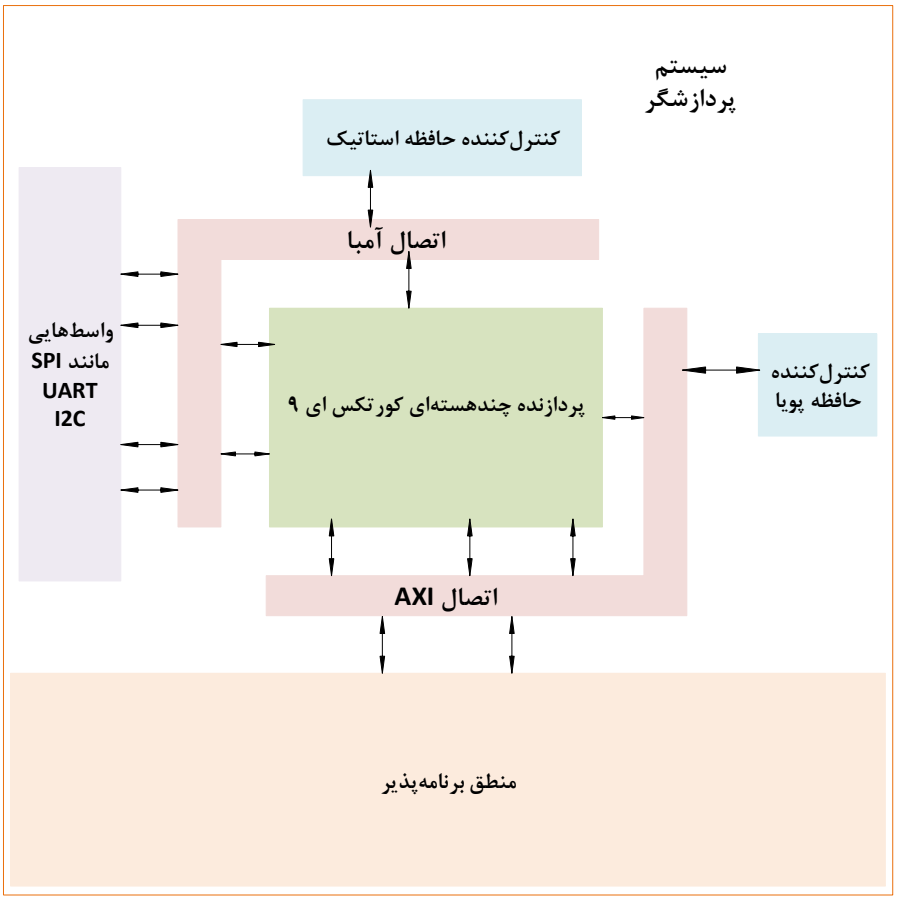
AXI4-stream یک کانال برای ارسال داده به صورت رشته[1] را ممکن می‌سازد. در این پروتکل دیگر مفهوم آدرس وجود ندارد و نیازی هم به آن نیست. در راه‌اندازی AXI4-stream نیاز است تا یک AXI DMA، پروتکل نگاشت بر روی حافظه را به رشته تبدیل کند.

همانطور که پیش‌تر گفته شد محصولات زایلینکس مانند زینک ۷۰۰۰ دو بخش سیستم پردازش‌گر و منطق برنامه‌پذیر را به گونه‌ای در اختیار طراحان قرار می‌دهند که بتوانند منطق دلخواه خود را همراه با اجرای یک نرم‌افزار بر روی هسته‌های پردازنده پیاده‌سازی کنند. این دو بخش با مجموعه‌ای از واسطه‌ها به یکدیگر متصل هستند که همان استاندارد AXI4 می‌باشد. یکی از متداول‌ترین کاربردهای این استاندارد، ایجاد یک منطق دلخواه در قسمت منطق برنامه‌پذیر توسط طراحان است. کنترل این بخش را می‌توان با استفاده از ثبات‌های نگاشت شده بر حافظه‌ای که سیستم پردازش‌گر از طریق واسط AXI4 به آنها دسترسی دارد انجام داد. در شکل ۲-۳ نحوه ارتباط دو بخش سیستم پردازش‌گر و منطق برنامه‌پذیر مشخص شده است[۱۰].

[1] stream



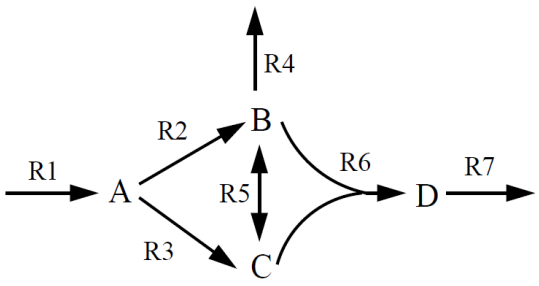
شکل ۲-۲ معماری کانال نوشتن در AXI4 [۹]



شکل ۲-۳ نمای کلی بورد و ارتباط PL و PS

۸.۲ روش دودویی در محاسبه حالت‌های پایه‌ای

در این بخش با استفاده از یک مثال به تشریح روش دودویی می‌پردازیم تا آنچه که در فصل‌های پیاده‌سازی و ارزیابی توضیح داده می‌شود روشن باشد. بدین منظور، شبکه متابولیکی نشان داده شده در شکل ۲-۴ را در نظر بگیرید[۳]. در این شکل، مدل‌سازی یک شبکه متابولیک به صورت یک گراف نشان داده است. در این گراف رئوس، متابولیت‌ها و یال‌ها واکنش‌های شیمیایی هستند. در این شبکه، متابولیت A در اثر واکنش R1 ایجاد می‌شود. این متابولیت در دو واکنش R2 و R3 مصرف می‌شود. محصول واکنش R2 متابولیت B و محصول واکنش R3 متابولیت C می‌باشد. متابولیت‌های B و C در واکنش R6 مصرف شده و متابولیت D تولید می‌شود. همچنین متابولیت B در واکنش R4 مصرف می‌شود. واکنش R5 یک واکنش دوطرفه است که در یک طرف از این واکنش متابولیت B مصرف شده و متابولیت C تولید می‌شود و در طرف دیگر واکنش عکس این عمل اتفاق می‌افتد. برای توصیف این شبکه و انجام عملیات ریاضی یک ماتریس با اندازه $m \times q$ به نام ماتریس استوکیومتری (N) تعریف می‌شود که در آن ردیف‌ها، همان متابولیت‌ها و ستون‌ها همان واکنش‌ها هستند. هر درایه n_{ij} در ماتریس، ضریب با علامت متابولیت i در واکنش j است، به گونه‌ای که علامت مثبت برای متابولیت تولید شده و علامت منفی برای متابولیت مصرف شده است.

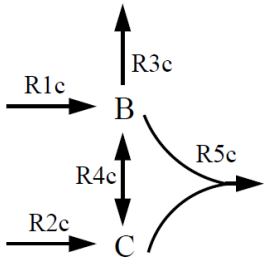


شکل ۲-۴ ساختار یک شبکه متابولیکی کوچک [۳]

ماتریس استوکیومتری معادل این شبکه به صورت زیر است:

$$\mathbf{N} = \begin{array}{c} \begin{matrix} R1 & R2 & R3 & R4 & R5 & R6 & R7 \end{matrix} \\ \begin{pmatrix} 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & -1 & -1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \end{array} \begin{matrix} A \\ B \\ C \\ D \end{matrix}$$

در ابتدای این روش نیاز است تا عمل فشرده‌سازی انجام شود تا ورودی الگوریتم آماده شود. برای فشرده کردن شبکه نشان داده شده از قوانینی مانند اینکه «متابولیت A تنها توسط یک واکنش تولید می‌شود، پس می‌توان واکنش R1 و R2 را باهم ترکیب کرد»، استفاده می‌شود. در نتیجه این تغییرات، گراف جدید به صورت شکل ۲-۵ خواهد بود:



شکل ۲-۵ ساختار شبکه متابولیکی فشرده شده [۳]

از روی گراف شکل ۲-۵، ماتریس استوکیومتری فشرده شده به شکل N_c ایجاد می‌شود:

$$\mathbf{N_C} = \begin{array}{c} \begin{matrix} R1c & R2c & R3c & R4c & R5c \end{matrix} \\ \begin{pmatrix} 1 & 0 & -1 & -1 & -1 \\ 0 & 1 & 0 & 1 & -1 \end{pmatrix} \end{array} \begin{matrix} B \\ C \end{matrix}$$

در مرحله بعد، باید یک ماتریس فضای پوچ برای ماتریس Nc محاسبه کنیم. طبق محاسبات جبری ثابت شده است که ماتریس فضای پوچ دارای ساختاری همچون $\begin{bmatrix} I \\ K' \end{bmatrix}$ است. با محاسبه ماتریس فضای پوچ، نتیجه به صورت زیر خواهد بود:

$$\mathbf{K_C} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0.5 & -0.5 & -0.5 \\ 0.5 & 0.5 & -0.5 \end{pmatrix} \begin{matrix} R1c \\ R2c \\ R3c \\ R4c \\ R5c \end{matrix}$$

در الگوریتم دودویی باید واکنش‌های برگشت‌پذیر را به دو واکنش برگشت‌ناپذیر در جهت مخالف هم تبدیل کنیم، به این معنی که علامت متابولیت‌های شرکت‌کننده در این دو واکنش عکس یکدیگر هستند. پس واکنش $R4c_b$ به عنوان واکنش برگشت برای R4c به ماتریس استوکیومتری اضافه می‌شود:

$$\mathbf{N_C} = \begin{array}{c} \begin{matrix} R1c & R2c & R3c & R4c_b & R4c & R5c \end{matrix} \\ \begin{pmatrix} 1 & 0 & -1 & 1 & -1 & -1 \\ 0 & 1 & 0 & -1 & 1 & -1 \end{pmatrix} \end{array} \begin{matrix} B \\ C \end{matrix}$$

برای ایجاد ماتریس فضای پوچ برای Nc’، طبق قضیه ۱ در [۳]، یک ردیف به بخش زیرماتریس همانی و یک ستون که تنها برای R4c و $R4c_b$ مقادیر یک دارد (ایجاد حلقه دوتایی)، اضافه می‌کنیم و در نتیجه ماتریس Kc’ ایجاد می‌شود.

$$\mathbf{K_C}' = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0.5 & -0.5 & -0.5 & 1 \\ 0.5 & 0.5 & -0.5 & 0 \end{pmatrix} \begin{matrix} R1c \\ R2c \\ R3c \\ R4c_b \\ R4c \\ R5c \end{matrix}$$

در این مرحله ماتریس آغازین برای انجام پیمایش‌های مرحله به مرحله و انجام محاسباتی که در ادامه توضیح داده خواهد شد، آماده است. در الگوریتم دودویی باید مقادیر حقیقی منفی موجود در هر ردیف را با استفاده از ترکیبات خطی مثبت ستونی که آن مقدار منفی در آن قرار گرفته است، با ستون‌های دیگر از بین ببریم. عمل جمع کردن برای ردیف‌هایی که در هرمرحله در حالت دودویی قرار دارند، با استفاده از عملگر «یا»ی منطقی انجام می‌گیرد. در Kc’، چهار ردیف اول که همان زیر ماتریس همانی هستند، در حالت دودویی قرار دارند. پس در حال حاضر ماتریس R که محاسبات روی آن قرار می‌گیرد، در پیمایش چهارم قرار دارد و R^4 به شکل زیر است:

$$\mathbf{R}^4 = \begin{pmatrix} \times & 0 & 0 & 0 \\ 0 & \times & 0 & 0 \\ 0 & 0 & \times & 0 \\ 0 & 0 & 0 & \times \\ \hline 0.5 & -0.5 & -0.5 & 1 \\ 0.5 & 0.5 & -0.5 & 0 \end{pmatrix} \begin{matrix} R1c \\ R2c \\ R3c \\ R4c_b \\ R4c \\ R5c \end{matrix}$$

ماتریس بالا به دو بخش دودویی و حقیقی تقسیم شده است. در بخش دودویی مقادیر غیر صفر با x مشخص شده‌اند. در این مرحله پنجمین ردیف را پردازش می‌کنیم. همه ستون‌هایی که مقادیر نامنفی دارند به همان شکل نگه داشته می‌شوند. ستون‌های دوم و سوم در ردیف R4c مقادیر منفی دارند. پس ستون‌های دوم و سوم را یک بار با ستون اول و بار دیگر با ستون چهارم ترکیب می‌کنیم تا در R4c

مقدار صفر ایجاد شود. برای هر جفت ستونی که لازم است باهم ترکیب شوند، باید تست مجاورت انجام دهیم. کاربرد این تست در الگوریتم دودویی، بررسی امکان اضافه شدن ستون‌های جدید به ماتریس R می‌باشد که نتیجه ترکیب شدن ستون‌های موجود در مرحله قبل هستند. در پیمایش پنجم در هر چهار حالت ممکن برای ترکیب، تست مجاورت صادق است و می‌توان نتیجه هر چهار حالت ترکیب را به ماتریس اضافه کرد. در ماتریس R5 ستون سوم در اثر ترکیب ستون‌های اول و دوم در R4 با ضرایب یک و ستون چهارم در اثر ترکیب ستون‌های اول و سوم در R4 با ضرایب یک ایجاد می‌گردند. همچنین ستون پنجم در R5 در اثر ترکیب ستون‌های دوم و چهارم در R4 به ترتیب با ضرایب دو و یک، و ستون ششم در R5 در اثر ترکیب ستون‌های سوم و چهارم در R4 به ترتیب با ضرایب دو و یک ایجاد می‌شوند[۳]. پس در پایان این مرحله، ماتریس به صورت زیر تغییر می‌کند:

$$R^5 = \begin{pmatrix} \times & 0 & \times & \times & 0 & 0 \\ 0 & 0 & \times & 0 & \times & 0 \\ 0 & 0 & 0 & \times & 0 & \times \\ 0 & \times & 0 & 0 & \times & \times \\ \times & \times & 0 & 0 & 0 & 0 \\ \hline 0.5 & 0 & 1 & 0 & 1 & -1 \end{pmatrix} \begin{matrix} R1c \\ R2c \\ R3c \\ R4c_b \\ R4c \\ R5c \end{matrix}$$

در این مرحله باید یک پیمایش دیگر برای ردیف ششم انجام دهیم. مقدار منفی در ستون آخر و برای R5c قرار دارد. ستون ششم را می‌توان با ستون‌های اول، سوم و پنجم ترکیب کرد. تست مجاورت برای ترکیب ستون ششم و سوم و همچنین ستون ششم و اول، صادق نیست. زیرا تعداد صفرهای ظاهر شده در اثر ترکیب آنها دو عدد است، در حالی که حداقل تعداد صفر در شبکه برابر است با:

تعداد ردیف‌های Nc – تعداد ستون‌های Nc = 6 – 2 – 1 =3

باتوجه به این نکته، نمی‌توان نتایجی را که تست مجاورت درباره آنها صادق نیست، به ماتریس اضافه کرد. بنابراین، نتیجه پیمایش ششم به صورت زیر است:

$$R^6 = \begin{pmatrix} \times & 0 & \times & \times & 0 & 0 \\ 0 & 0 & \times & 0 & \times & \times \\ 0 & 0 & 0 & \times & 0 & \times \\ 0 & \times & 0 & 0 & \times & \times \\ \times & \times & 0 & 0 & 0 & 0 \\ \times & 0 & \times & 0 & \times & 0 \end{pmatrix} \begin{matrix} R1c \\ R2c \\ R3c \\ R4c_b \\ R4c \\ R5c \end{matrix}$$

ماتریس R^6 نمایش دودویی حالت‌های پایه‌ای برای شبکه فشرده شده‌ای است که در آن واکنش‌های برگشت‌پذیر، تفکیک شده‌اند. در مراحل بعدی پردازش، باید ستونی را که در اثر حلقه دوتایی برای واکنش برگشت‌پذیر R4c ایجاد شده است، حذف کنیم. پس دو ردیف R4c و R4cb را با یک عملگر «یا»ی منطقی ترکیب می‌کنیم. در نهایت خروجی این الگوریتم یعنی حالت‌های پایه‌ای برای شبکه فشرده شده به صورت زیر است و هر ستون مشخص‌کننده نمایش دودویی یک بردار حالت پایه‌ای است[۳]:

$$\mathbf{EM}_{\mathrm{C}}^{B} = \begin{pmatrix} \times & \times & \times & 0 & 0 \\ 0 & \times & 0 & \times & \times \\ 0 & 0 & \times & 0 & \times \\ \times & 0 & 0 & \times & \times \\ \times & \times & 0 & \times & 0 \end{pmatrix} \begin{matrix} R1c \\ R2c \\ R3c \\ R4c \\ R5c \end{matrix}$$

۳

فصل سوم

طراحی و پیاده‌سازی

۱.۳ معماری سیستم

همانطور که پیش‌تر نیز اشاره شد، این پروژه به صورت طراحی توامان و سخت‌افزار و نرم‌افزار انجام گرفت. علت استفاده از این شیوه بهره‌مندی از مزایایی همچون سرعت بالای اجرای سخت‌افزار می‌باشد. البته باید توجه داشت که بسیاری از محاسبات ریاضی و جبر خطی لازم در روش دودویی، به صورت سخت‌افزاری قابل پیاده‌سازی نیستند و نیاز است تا با استفاده از کد نرم‌افزار پیاده‌سازی شوند. پس با ترکیب سخت‌افزار و نرم‌افزار، طراحی سیستم به صورت توامان خواهد بود. دو بخش منطق برنامه‌پذیر و سیستم پردازش‌گر بر روی زدبورد قرار دارند و به ترتیب وظیفه اجرای بخش سخت‌افزاری و نرم‌افزاری پیاده‌سازی شده را به عهده دارند. اتصال میان این دو بخش با استفاده از AXI4-lite انجام گرفته است که راه‌اندازی و استفاده از آن یکی از مراحل پروژه را تشکیل می‌دهد. در پایان اجرای الگوریتم بر روی زدبورد خروجی به واسطه پروتکل UART به سیستم کامپیوتری منتقل می‌شود و بر روی صفحه نمایش قابل مشاهده است.

۲.۳ مراحل پیاده‌سازی

الگوریتمی که در این پروژه پیاده‌سازی شده است از سه بخش اصلی تشکیل می‌شود: پیش‌پردازش، بخش اصلی و پس‌پردازش. دو بخش پیش‌پردازش و پس‌پردازش شامل تعداد زیادی محاسبات ریاضی و جبر خطی می‌شوند و از این رو قسمت نرم‌افزاری سیستم را تشکیل می‌دهند. از آنجایی که هر دو بخش گفته شده تنها یک بار اجرا می‌شوند، می‌توان از اثر آنها بر سرعت اجرا چشم‌پوشی کرد. بخش اصلی الگوریتم شامل تعداد زیادی پیمایش بدون محاسبات پیچیده ریاضی است و از این رو پیاده‌سازی این بخش به صورت سخت‌افزاری انجام می‌گیرد. شبه‌کد هر یک از این بخش‌ها که مربوط به پیاده‌سازی نرم‌افزار متلب[1] می‌باشد، به ترتیب در کدهای ۳-۱ و ۳-۲ و ۳-۳ مشخص شده است.

[1] MATLAB

```
%%% Variables: N: compressed stoichiometric matrix; m: number of rows in N (metaboltes);
% q: number of columns in N (reactions); rev: indices of reversible reactions (columns in N);
% irrev: indices of irreversible reactions (columns in N);
%%% Pre_Processing
%% reconfiguration
qsplit = q + length(rev);
%% initializeR : initialize R by computing a proper null space of matrix Nsplit; dimension: qsplit
% rows and qsplit-m columns
R = initializeR(N,rev);
%% splitting R in binary (R1) and real number (R2)
R1 = makeBitmap(R(1:qsplit-m, :));
R2 = R(qsplit-m+1, :);
numr = qsplit – m;
```

کد ۳-۱ شبه‌کد فاز پیش‌پردازش الگوریتم دودویی [۳]

```
%%% Main Part
for p = (qsplit-m+1):q
 new_numr = numr;
 jneg = find(R2(1, :) < 0);
 jpos = find(R2(1, :) > 0);
 for k=1:length(jneg)
  for l=1:length(jpos)
   newr = or(R1(:, jneg(k)) , R1(:, jpos(l)));   % element-wise OR
   %check for minimum number of zeros;
   if(numberOfNullBits(newr)+1 < qsplit-m+1)
    continue;
   end
   %Adjacency test
   adj = 1;
   r = 0;
   while(adj & r<= numr)
    r = r+1;
    testr = or(newr, R1(:,r));
    if(r ~= 1 & r ~= k & all(testr==newr))
     adj = 0;
    end
   end
   if(adj)
    new_numr = new_numr + 1;
    R1(:, new_numr) = newr;
    R2(:, nem_numr) = R2(1,jpos(l))*R2(:,jneg(k)) – R2(1,jneg(k))*R2(:, jpos(l));
   end
  end
 end
 %deletion of negative rays
```

```
 R1(:, jneg) = [];
 R2(:, jneg) = [];
 numr = new_numr – length(jneg);

 %transfer current row of R2 as Bitmap into R1; then delete this row in R2
 R1(p, :) = makeBitmap(R2(1, :));  % makeBitmap returns the bit mask of a matrix
 R2(1,:) = [];
end
```

کد ۳-۲ شبه‌کد فاز اصلی الگوریتم دودویی [۳]

```
%%% Post-Processing
%% step 1
remove2cycle(R1); % this sub-routine removes the 2cycles obtained from splitting rev  reactions
numr = numr – rev;
R1(rev, :) = or(R1(rev, :), R1(q+1:qsplit, :))
% the rules for back-configuration
R1(q+1:qsplit, ;) = [];
% deletes backward directions

%% step 2
fullEM = zeros(q, numr);
for(i=1:numr)
        preacs = findOneBits(R1(:, i));
        % this subroutine delivers the positions in EM i whose bit is 1
        Ne = N(:,preacs);
        v = null(Ne);
end
```

کد ۳-۳ شبه‌کد فاز پس‌پردازش الگوریتم دودویی [۳]

فاز پیش‌پردازش شامل مراحلی است که ورودی مورد نظر را برای الگوریتم فراهم می‌کند. محاسبات اصلی الگوریتم در بخش اصلی انجام می‌گیرند و در قسمت پس‌پردازش نیز خروجی نهایی الگوریتم تولید می‌شود. محاسباتی که در فاز پیش‌پردازش وجود دارند اعمال ریاضی و جبر خطی هستند که به صورت نرم‌افزاری پیاده‌سازی شده‌اند. بخش اصلی شامل یک حلقه بزرگ است که در آن محاسبات اصلی برای تبدیل کردن ماتریس اولیه به معادل دودویی و سپس محاسبه حالت‌های پایه‌ای انجام می‌گیرد. برای تسریع در انجام محاسبات، این قسمت را با استفاده از سخت‌افزار و زبان VHDL پیاده‌سازی می‌کنیم.

پس از پایان پیمایش‌های این حلقه بزرگ که خود شامل چندین حلقه دیگر است، به بخش پس‌پردازش می‌رسیم که حذف اطلاعات اضافی از خروجی سخت‌افزار و تبدیل مقادیر دودویی به مقادیر حقیقی حالت‌های پایه‌ای می‌باشد. این بخش نیز همانند پیش‌پردازش تنها یک بار انجام می‌گیرد و تاثیر چندانی در زمان اجرای الگوریتم ندارد. از این رو بخش پس‌پردازش نیز به صورت نرم‌افزاری قابل پیاده‌سازی است. پس از آماده‌سازی هر سه بخش گفته شده، لازم است تا اتصال بین نرم‌افزار و سخت‌افزار برقرار شود تا الگوریتم به صورت کامل پیاده‌سازی شود، بر روی زدبورد اجرا شود و با استفاده از معماری کاملی که در بخش قبل به آن اشاره شد در اختیار ما قرار بگیرد. در ادامه جزئیات هر یک از مراحل به همراه نحوه پیاده‌سازی و ابزار مورد استفاده در آنها توضیح داده خواهد شد.

۱.۲.۳ فاز پیش‌پردازش

این فاز با استفاده از زبان C پیاده‌سازی شده است و ورودی اصلی را برای فاز اصلی الگوریتم فراهم می‌کند. باید توجه داشت که ورودی این بخش ماتریس استوکیومتری فشرده شده می‌باشد. همانطور که پیش‌تر توضیح داده شد ماتریس استوکیومتری شامل ضرایب استوکیومتری متابولیت‌ها در واکنش‌های شیمیایی شبکه است که سطرهای آن را متابولیت‌ها و ستون‌های آن را واکنش‌ها تشکیل می‌دهند. باید توجه داشت که عمل فشرده‌سازی کار بسیار پیچیده‌ای است و تا کنون ابزارهای متفاوتی آن را پیاده‌سازی کرده‌اند. ابزاری که خروجی مورد نظر ما را برای الگوریتم دودویی فراهم می‌کند CellNetAnalyzer نام دارد[۱۴] که در فصل بعد درباره آن توضیح داده خواهد شد. کد ۳-۱ شبه‌کد مربوط به بخش پیش‌پردازش الگوریتم را نشان می‌دهد. همانطور که در آن مشخص شده است، اعمالی که در این بخش انجام می‌گیرند به شرح زیر است:

۱.۱.۲.۳ تغییر ماتریس فشرده شده

این مرحله شامل جداسازی واکنش‌های دوطرفه می‌شود. برای این کار به ازای هر واکنش برگشت‌پذیر یک ستون به ماتریس استوکیومتری اضافه می‌کنیم که مقادیر آن قرینه مقادیر واکنش اصلی است. پیاده‌سازی این بخش با استفاده از تابع reconfigure انجام می‌گیرد. در کد ۳-۱ نیز هدف از reconfigure همین امر می‌باشد. در نتیجه این تغییر، تعداد ستون‌های ماتریس به تعداد واکنش‌های برگشت پذیر افزایش پیدا می‌کند.

```
struct Matrix reconfigure(struct Matrix m, int *irrev) { … }
```

۲.۱.۲.۳ مقداردهی اولیه ماتریس R و جداسازی آن به دو بخش دودویی و حقیقی

محاسبات اصلی الگوریتم بر روی ماتریسی با نام R انجام می‌گیرد که در کد ۳-۱ خروجی تابع initializeR می‌باشد. این ماتریس در اصل از روی ماتریس کرنل ساخته می‌شود. همانطور که در [۳] اشاره شده است ماتریس اولیه برای این الگوریتم باید به صورت $[\frac{I}{K}]$ ایجاد شود. برای ساختن چنین ماتریسی ابتدا لازم است تا فضای پوچ ماتریس استوکیومتری حاصل از مرحله قبل را محاسبه کنیم. برای انجام این کار تابع null_space را پیاده‌سازی کردیم. برای پیاده‌سازی این تابع از روش گفته شده در [۱۱] استفاده شد. پس از آن باید بر روی ترانهاده ماتریس فضای پوچ، عملیات انجام دهیم تا امکان ایجاد یک ماتریس همانی فراهم شود. این بخش با استفاده از تابع rref انجام می‌گیرد. در ادامه نتیجه را به وسیله تابع tranpose ترانهاده کرده و با تابع row_perm سطرهایی که پشت سر هم می‌توانند یک ماتریس همانی تشکیل دهند را در قسمت بالایی ماتریس کنار هم قرار می‌دهیم. در آخر قسمت بالای ماتریس که خود یک ماتریس همانی می‌باشد (R1)، به عنوان بخش دودویی و بخش پایینی ماتریس به عنوان بخش حقیقی (R2) در نظر گرفته می‌شوند. در کد ۳-۱ نیز مقداردهی R1 با استفاده از تابع makeBitmap انجام می‌گیرد تا فرم دودویی ایجاد شود. اندازه ماتریس R1 در رابطه ۳-۱ مشخص شده است که در آن q تعداد واکنش‌ها، |rev| تعداد واکنش‌های برگشت‌پذیر و m تعداد متابولیت‌ها می‌باشد. بقیه سطرهای ماتریس R که در ماتریس همانی قرار نمی‌گیرند، ماتریس R2 را تشکیل می‌دهند. شبه‌کد این مرحله در کد ۳-۱ نشان داده شده است.

$$q + |rev| - m$$ رابطه ۳-۱

```
struct Matrix initialize(struct Matrix m)
{
   struct Matrix tr = transpose(m);
   struct Matrix tr_rref = rref(tr);
   struct Matrix tr_rref_tr = transpose(tr_rref);
   row_perm(tr_rref_tr);
   return tr_rref_tr;
}
```

۲.۲.۳ فاز محاسبات اصلی

ورودی‌های اصلی این بخش از الگوریتم دو ماتریس R1 و R2 هستند که در بخش قبل تشکیل دادیم. R1 یک ماتریس با مقادیر دودویی و R2 یک ماتریس با مقادیر حقیقی است. کد ۳-۲ شبه‌کد مربوط به بخش پیش‌پردازش الگوریتم را نشان می‌دهد. همانطور که در کد مشخص شده است، متغیر m معادل تعداد متابولیت‌ها، q معادل تعداد واکنش‌ها و qsplit معادل جمع تعداد واکنش‌ها و تعداد واکنش‌های برگشت‌پذیر است.

برای توصیف این بخش با استفاده از زبان VHDL ابتدا یک نمودار حالت طراحی کردیم تا با استفاده از آن بتوانیم همه حلقه‌ها و پیمایش‌های لازم را انجام دهیم. شکل ۳-۱ نمودار حالت طراحی شده را مشخص می‌کند و در هر زیربخش حالت‌های مربوط به آن توضیح داده خواهد شد. البته در سه حالت اول کارهای مقدماتی انجام می‌شود. حالت S0 مربوط به مقداردهی اولیه متغیرهاست. در حالت S0a، آرایه یک بعدی به یک ماتریس تبدیل می‌شود و به این ترتیب ماتریس R1 شکل می‌گیرد. در حالت S0b، همین عملیات برای ماتریس R2 انجام می‌شود. کد پیاده‌سازی ماشین حالت در بخش پیوست قابل مشاهده است. اعمالی که در این بخش انجام می‌گیرند به شرح زیر است:

۱.۲.۲.۳ پیمایش در حلقه اصلی به تعداد متابولیت‌ها

در این فاز یک حلقه اصلی وجود دارد که به تعداد متابولیت‌های شبکه تکرار می‌شود، زیرا به تعداد متابولیت‌ها سطر حقیقی در ماتریس R2 وجود دارد که هریک از آنها باید در یک پیمایش تغییر کرده و به حالت دودویی تبدیل شوند. حالت S1 در ماشین حالت نیز مربوط به این فاز می‌باشد. مشخص کردن هر سطر از R2 که لازم است مورد پردازش قرار بگیرد در حالت S1a انجام می‌گیرد و مقادیر سطر مورد نظر در یک متغیر جداگانه کپی می‌شود. در کد ۳-۲ این حلقه با متغیر p مشخص می‌شود و p همان شماره سطری از ماتریس R است که در هر مرحله از حلقه مورد پردازش قرار می‌گیرد.

۲.۲.۲.۳ پیدا کردن شماره درایه عناصر منفی و مثبت هر سطر

در این حلقه ابتدا تعداد عناصر مثبت و منفی اولین ردیف از ماتریس R2 شمارش می‌شود تا تعداد پیمایش‌های حلقه‌های تودردتو برای ترکیب ستون‌ها و از بین بردن عناصر منفی مشخص شود. سپس درایه‌های عناصر مثبت و منفی به صورت جداگانه مشخص می‌شوند. علت اینکه همیشه اولین ردیف

مورد هدف ما قرار می‌گیرد این است که پس از پایان یافتن هر پیمایش بر روی یک ردیف از R2 آن ردیف را حذف کرده و به R1 اضافه می‌کنیم. البته این کار در کد سخت‌افزاری به شکل دیگری صورت می‌گیرد و در هر پیمایش به شماره ردیفی از R2 که مورد نظر ماست با استفاده از یک متغیر، یک واحد اضافه می‌شود. لازم به ذکر است که پیاده‌سازی تابع find در کد ۳-۲ با استفاده از حلقه صورت می‌گیرد و از این رو باعث اضافه شدن حالت S1b به نمودار حالت مورد نظر می‌گردد. در کد ۳-۲ شماره اندیس مقادیر مثبت در jneg و شماره اندیس مقادیر مثبت در jpos ذخیره می‌شود.

۳.۲.۲.۳ ترکیب هر ستون حاوی مقدار منفی با همه ستون‌های دارای مقدار مثبت در آن سطر

پس از پیدا کردن تعداد عناصر مثبت و منفی یک حلقه تودرتو خواهیم داشت که در آن تلاش خواهد شد تا عناصر منفی را با استفاده از ترکیب با ستون‌های شامل عناصر مثبت حذف کنیم. برای این کار باید ترکیب همه حالات را در ماتریس R1 بررسی کنیم و دو شرط را چک کنیم تا مطمئن شویم که اضافه کردن چنین ترکیبی امکان‌پذیر است. منظور از ترکیب اعمال عملگر «یا»ی منطقی بر روی عناصر دو ستون مورد نظر در ماتریس R1 است که در کد ۳-۲ نیز با استفاده از تابع or انجام گرفته و در newr ذخیره شده است. در نمودار حالت، حلقه بیرونی با استفاده از حالت S2 و حلقه درونی با استفاده از حالت S3 پیاده‌سازی می‌شود. در حالت S3 نتیجه ترکیب دو ستون مورد نظر در ماتریس R1 نیز انجام می‌شود.

۴.۲.۲.۳ کنترل تعداد صفرها

شرط اولی که باید بررسی شود مربوط به تعداد صفرهای ظاهر شده در برداری است که از اعمال عملگر «یا»ی منطقی بر روی دو بردار از ماتریس R1، نتیجه می‌شود. دو بردار از R1 همان دو برداری هستند که می‌خواهیم آنها را با هم ترکیب کنیم و به این ترتیب یکی از آنها در ردیف اول R2 مقدار منفی و دیگری مقدار مثبت دارد. تعداد صفرها در بردار حاصل نباید از $qsplit - m$ کمتر باشد در غیر این صورت ترکیب مورد نظر امکان‌پذیر نیست و باید دو بردار دیگر را برای ترکیب انتخاب کنیم. شمارش تعداد صفرها در حالت S3a و کنترل برقراری شرط تعداد آنها در S3aa انجام می‌شود. در کد ۳-۲، برای شمارش تعداد صفرها تابع numberOfNullBits استفاده شده است.

۵.۲.۲.۳ انجام تست مجاورت

شرط دومی که باید مورد بررسی قرار گیرد تست مجاورت است. همانطور که در [۳] بیان شده است، تست مجاورت به صورت زیر تعریف می‌شود:

$$(r^{j+}{}_{1\ldots p} \text{ OR } r^{j-}{}_{1\ldots p}) \text{ OR } r^{k}{}_{1\ldots p} \neq (r^{j+}{}_{1\ldots p} \text{ OR } r^{j-}{}_{1\ldots p}) \qquad \text{رابطه ۳-۲}$$

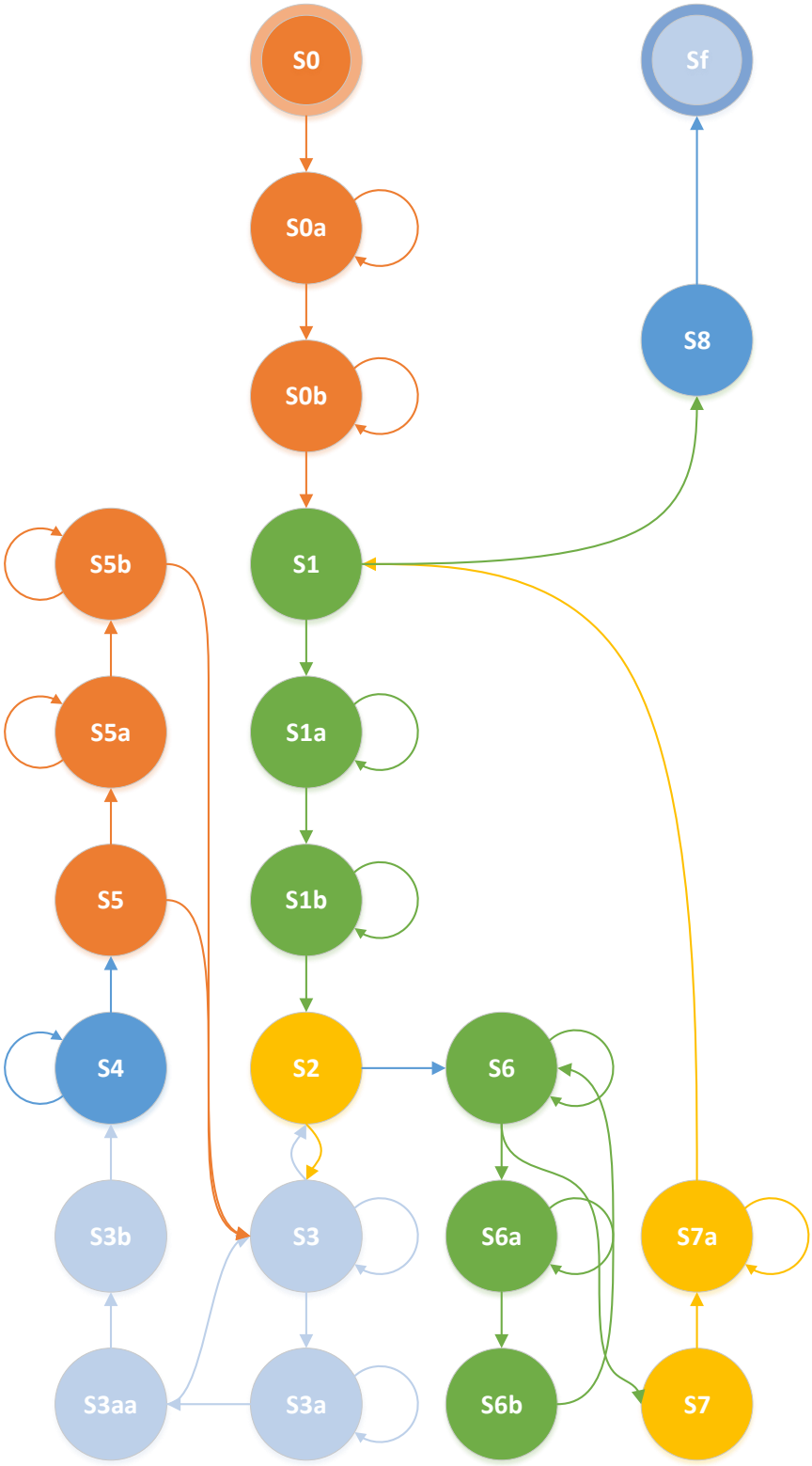
به عبارت دیگر تست مجاورت بررسی می‌کند که حاصل «یا»ی منطقی دو بردار مورد نظر از R1 که می‌خواهیم باهم ترکیب کنیم و هر بردار دیگری از R1، با حاصل «یا»ی منطقی آن دو بردار برابر نباشد. اگر این شرط نیز برقرار نباشد، نمی‌توان دو بردار مورد نظر را با هم ترکیب کرد. در حالت S3b مقدمات لازم برای چک کردن شرط مجاورت انجام می‌گیرد. در حالت S4 آزمون مجاورت انجام می‌شود. در کد ۳-۲ حلقه while مربوط به بررسی شرط مجاورت می‌باشد که در آن همه ستون‌هایی که یکی از دو ستون ایجاد کننده newr نباشند برای کنترل رابطه ۳-۲ بررسی می‌شوند.

۶.۲.۲.۳ اضافه کردن ترکیب ستون‌ها در صورت امکان

اگر هر دو شرط برقرار باشند، عمل ترکیب را انجام می‌دهیم. در ماتریس R1 عمل ترکیب معادل همان «یا»ی منطقی بین دو بردار است و در ماتریس R2 ترکیب خطی‌ای استفاده می‌شود که طی آن مقدار منفی به صفر تغییر می‌کند. در حالت S5 اعمال مقدماتی برای ترکیب ستون‌ها، در حالت S5a ترکیب ستون‌های ماتریس R1 و در حالت S5b ترکیب ستون‌های ماتریس R2 انجام می‌شود.

۷.۲.۲.۳ پاک کردن ستون‌های دارای مقدار منفی

بدین ترتیب یک ستون جدید اضافه شده و ستونی را که مقدار منفی داشته است حذف می‌کنیم. در کد ۳-۲ این کار به آسانی و با خالی کردن یک ستون انجام می‌شود ولی برای انجام آن در کد سخت‌افزاری به جای ستون‌هایی که دارای مقدار منفی هستند، ستون‌های پایانی ماتریس را قرار می‌دهیم. حالت‌های S6، S6a و S6b مربوط به انجام این عمل هستند.



شکل ۳-۱ نمودار حالت توصیف کننده بخش اصلی الگوریتم دودویی

۸.۲.۲.۳ اضافه کردن سطر مورد نظر به بخش دودویی (ماتریس R1)

در این مرحله اعمال لازم بر روی ردیف اول R2 پایان گرفته است و می‌توانیم آن را به R1 اضافه کنیم. در چنین مرحله‌ای این ردیف تنها دارای مقادیر صفر یا مثبت است. پس مقادیر مثبت را به صورت یک و مقادیر صفر را به همان صورت صفر به R1 اضافه می‌کنیم. در کد ۳-۲ از تابع makeBitmap بدین منظور استفاده می‌شود. حالت‌های S7 و S7a مربوط به این بخش می‌باشند. در پایان حالت S7a به حالت S1 برمی‌گردیم تا مرحله بعدی پیمایش و انجام عملیات بر روی سطر جدیدی از ماتریس R2 انجام بگیرد. در صورتی که همه پیمایش‌های حلقه اصلی انجام شود، در حالت S8 خروجی که در ماتریس R1 قرار دارد به آرایه یک بعدی تبدیل می‌شود و در حالت Sf، سیگنال‌های خروجی entity مقداردهی می‌شوند.

تمامی مراحلی که در بالا گفته شد با استفاده از زبان VHDL پیاده‌سازی شده است و برای شبیه‌سازی آن از نرم‌افزار مدلسیم و ویوادو استفاده شده است. برای انجام محاسبات لازم است تا عملیات ریاضی همچون ضرب و تفریق بر روی اعداد اعشاری انجام بگیرد. در زبان VHDL اعداد حقیقی و ممیز شناور قابل سنتز نیستند. از این رو برای استفاده از اعداد اعشاری باید از کتابخانه fixed_pkg_2008 استفاده کنیم. این کتابخانه در نسخه VHDL 2008 پشتیبان می‌شود و در نسخه‌های دیگر کاربرد ندارد. برای استفاده از این کتابخانه ابتدا باید فایل مربوط به آن را به پروژه اضافه کنیم. این کار را با استفاده از دستورات زیر انجام می‌دهیم:

```
add_files –norecurse <path to package file>/fixed_pkg_2008.vhd
set_property library ieee [get_files <path to package file>/fixed_pkg_2008.vhd]
```

در قسمت کتابخانه‌های مورد استفاده در کد نیز باید خط زیر را اضافه کنیم:

```
use ieee.fixed_pkg.all;
```

در بدنه کد تعریف سیگنال‌ها، متغیرها و پورت‌ها به صورت زیر انجام می‌گیرد:

```
signal s1: ufixed(21 downto -2);
variable v1 : sfixed(10 downto -4);
port (….
P1: in ufixed(12 downto -2);
….);
```

استفاده از این سیگنال‌ها و متغیرها مشابه دیگر سیگنال‌ها و متغیرهاست.

در استفاده از مقادیر ممیز ثابت باید به دو نکته توجه داشت:

۱. تفاوت میان sfixed و ufixed که اولی برای مقادیر علامت‌دار (signed) و دومی برای مقادیر بدون علامت (unsigned) است.

۲. اندازه مشخص شده نشان‌دهنده تعداد بیت‌های مورد استفاده برای بخش صحیح و بخش اعشاری است. به طور مثال در سیگنال S1 که در بالا تعریف شد، مشخص شده است که بازه صفر تا ۲۱ یعنی ۲۲ بیت به بخش صحیح و بازه ۱- تا ۲- نیز به بخش اعشاری اختصاص داده شده است[۱۲].

برای انجام محاسبات نیز باید در نظر داشت که اندازه نتیجه به اندازه عملوندها بستگی دارد و به طور مثال برای عمل جمع، این اندازه با بیشینه اندازه دو متغیر بعلاوه یک، هم در قسمت صحیح و هم قسمت اعشاری برابر است. همچنین برای عمل ضرب این اندازه برابر است با مجموع اندازه دو عملوند. از این رو باید به اندازه سیگنال یا متغیری که نتیجه عملیات در آن ذخیره می‌شود توجه کرد[۱۲].

لازم به ذکر است که در ابزار ویوادو اگرچه کتابخانه اعداد ممیز ثابت سنتزپذیر است ولی در فاز شبیه‌سازی قابل استفاده نیست. از این رو برای شبیه‌سازی پیش از سنتز از ابزار مدلسیم استفاده شد. برای شبیه‌سازی پیش از سنتز یک تست‌بنچ[1] نوشته شد که در آن سیگنال‌ها و پورت‌ها به صورت sfixed تعریف شدند و شبیه‌سازی با استفاده از مدلسیم انجام گرفت. ولی برای شبیه‌سازی پس از سنتز استفاده از آن تست‌بنچ امکان‌پذیر نبود زیرا همانطورکه پیش‌تر اشاره شد، ویوادو قابلیت شبیه‌سازی این کتابخانه را ندارد. به همین خاطر برای تعریف entity مورد نظر که دارای پورت‌هایی از نوع sfixed است، باید از تعریف پس از سنتز entity استفاده کنیم. به همین خاطر پس از پایان سنتز باید توصیف نت‌لیست[2] با استفاده از کد VHDL را با استفاده از دستور زیر تولید کنیم:

write_vhdl <path to target file>/name.vhd

[1] test bench

[2] netlist

در فایل خروجی دستور بالا، کل نت‌لیست به صورت کد VHDL نوشته شده است. بسته به تعداد LUTهای مورد استفاده، اندازه این فایل تغییر می‌کند و حتی در حدود یک میلیون خط کد قرار می‌گیرد. در توصیف entity جدید که در این فایل مشاهده می‌کنیم، می‌بینیم که متغیرهای generic به attribute تبدیل شده‌اند. همچنین هر یک از پورت‌های sfixed به یک std_logic_vector تبدیل شده است. با توجه به اینکه در entity ما یک آرایه از sfixedها وجود دارد، مشاهده می‌کنیم که به تعداد اندازه آن آرایه در پورت‌های entity پس از سنتز، std_logic_vector ایجاد شده است. پس در توصیف تست‌بنچ نیز باید به همین ترتیب پورت‌های sfixed را با استفاده std_logic_vector مقداردهی کنیم.

در صورتی که نیاز باشد برای شبیه‌سازی پس از سنتز نیز اشکال‌زدایی انجام دهیم، می‌توانیم متغیرهای کنترلی تعریف کنیم ولی باید توجه داشت که عمل سنتز بهینه‌سازی‌هایی انجام می‌دهد که منجر به حذف برخی سیگنال‌های غیرضروری می‌شود. برای حل این مسئله می‌توان از ویژگی keep استفاده کرد. این ویژگی ابزار سنتز را ملزم می‌کند تا سیگنالی را که دارای این ویژگی است نگه دارد و آن را در نت‌لیست قرار دهد. مقادیری که برای این ویژگی وجود دارد، TRUE و FALSE هستند و در حالت پیش فرض مقدار آن FALSE است[۱۲].

۳.۲.۳ فاز پس‌پردازش

در پایان مرحله قبل، ماتریس R1 نمایش دودویی حالت‌های پایه‌ای یک شبکه متابولیکی، پس از تنظیمات لازم تولید شد. نمایش دودویی حالت‌های پایه‌ای شبکه متابولیکی مورد نظر در میان ستون‌های این ماتریس قرار دارد ولی باید فاز پس‌پردازش را انجام دهیم تا به بردارهای حالت‌های پایه‌ای دست پیدا کنیم. این فاز نیز به صورت نرم‌افزاری و با استفاده از زبان C پیاده‌سازی شده است و در پایان به صورت بخش نرم‌افزاری که بر روی سیستم پردازش‌گر زدبورد اجرا می‌شود مورد استفاده قرار می‌گیرد. کد ۳-۳ شبه‌کد مربوط به بخش پس‌پردازش الگوریتم را نشان می‌دهد. عملیاتی که در این بخش انجام می‌گیرد به شرح زیر است.

۱.۳.۲.۳ حذف حلقه‌های دوتایی

در این مرحله باید تغییراتی در ماتریس R1 انجام گیرد و حلقه‌های دوتایی از آن حذف شود. ستون‌های این حلقه‌های دوتایی در اصل با اضافه کردن عکس واکنش‌های برگشت‌پذیر به ماتریس استوکیومتری

تولید می‌شوند. پس از مقداردهی اولیه ماتریس R تا پایان فاز محاسبات اصلی، این حلقه‌های دوتایی به صورت ستون‌هایی مشاهده می‌شود که تنها در سطرهای مربوط به یک واکنش برگشت‌پذیر خاص و جهت عکس آن دارای مقدار یک است. حذف این ستون‌ها با استفاده تابع remove2Cycles انجام شده است. در کد ۳-۳ نیز تابع remove2cycles همین عملکرد را پیاده‌سازی می‌کند.

```
struct Matrix remove2Cycles(struct Matrix m, int rev_count){ … }
```

باید توجه داشت که پیش از انجام این مرحله لازم است تا ترتیب قرارگیری واکنش‌ها به حالت اولیه‌ای که در ماتریس استوکیومتری قرار داشت، مرتب شود. این ترتیب در تابع row_perm در فاز پیش‌پردازش جابجا شد تا بتوان شکل $\begin{bmatrix} I \\ K \end{bmatrix}$ را به وجود آورد. بازمرتب‌سازی در این مرحله با استفاده از تابع row_perm_post انجام می‌شود.

```
void row_perm_post(struct Matrix m, int *perm){ … }
```

۲.۳.۲.۳ انجام عمل عکس تنظیمات بر روی ماتریس استوکیومتری

در این مرحله عمل back-configuration را انجام دهیم. از آنجایی که ما بردارهای شار را برای ماتریس استوکیومتری تغییریافته محاسبه کردیم باید در نظر داشته باشیم که برای مشخص کردن بردارهای شار ماتریس استوکیومتری اصلی کدام یک از بردارهای شار را انتخاب کنیم. طبق قانون گفته شده در [۳] اگر N ماتریس استوکیومتری اصلی و N' ماتریس استوکیومتری شبکه تغییریافته باشد، داریم:

$$N' = [N - N_{Rev}]$$

که در آن Nrev همه ستون‌هایی از N است که به واکنش‌های برگشت‌پذیر مربوط می‌شود. اگر بردار شار شبکه اصلی را با v و بردار شار شبکه تغییریافته را با v' نشان دهیم، داریم: Nv = N'v'. اگر برای هر $i \in Rev$ حداقل یکی از دو ضریب $v'_{(i, +1)}$ و $v'_{(i, -1)}$ صفر باشند، برای نگاشت کردن v' به v خواهیم داشت:

$v_i = v'_i$ if i Irrev

$v_i = v'_{(i,+1)}$ if $i \in$ Rev and $v'_{(i,-1)} = 0$

$v_i = -v'_{(i,-1)}$ if $i \in$ Rev and $v'_{(i,+1)} = 0$

به این عمل back_configuration می‌گویند. پس باید با استفاده از عملگر «یا»ی منطقی vهای موردنظر را مقداردهی کنیم. در پایان این مرحله معدل دودویی حالت‌های پایه‌ای شبکه آماده می‌شوند. برای انجام این کار تابع bc را پیاده‌سازی کردیم و خروجی این تابع بعنوان ورودی برای آخرین مرحله از

الگوریتم مورد استفاده قرار خواهد گرفت. در کد ۳-۳ برای انجام این بخش، از تابع or استفاده می‌کند و پس از آن سطر اضافه‌ای را که مربوط به جهت برگشت واکنش است، پاک می‌کند.

```
struct Matrix bc(struct Matrix m, int rev_count){ … }
```

۳.۳.۲.۳ تبدیل مقادیر دودویی به مقادیر حقیقی

در این بخش از فاز پس‌پردازش، باید با استفاده از الگوی دودویی ضرایب حقیقی حالت‌های پایه‌های را محاسبه کنیم. در مرحله قبل بردارهای ماتریس خروجی نمایش دودویی حالت‌های پایه‌ای بودند. پس هر بردار به ازای برخی واکنش‌ها دارای مقدار ۱ و به ازای برخی دیگر دارای مقدار ۰ است. از این رو تابع bin2real وظیفه دارد برای هر بردار ماتریس ضرایب، درایه‌هایی را که مقدار یک دارند پیدا کند و با استفاده از محاسبه فضای پوچ برای ماتریس ضرایب مقادیر غیر صفر بردارها را محاسبه و به صورت یک عدد حقیقی نمایش دهد. لازم به ذکر است که منظور از ماتریس ضرایب یک بردار شار، آن ستون‌هایی از ماتریس استوکیومتری اولیه است که بردار شار در سطر متناظر با آن واکنش‌ها مقدار یک داشته باشد. در کد ۳-۳ اندیس مقادیر غیر صفر به ازای هر بردار حالت پایه‌ای در preacs ذخیره می‌شود. Ne ماتریس ضرایب را نشان می‌دهد و v، فضای پوچ ماتریس ضرایب را نشان می‌دهد. مقادیر محاسبه شده برای همه بردارهای v خروجی تابع bin2real می‌باشد.

```
struct Matrix *bin2real(struct Matrix em, struct Matrix n){ … }
```

پس از محاسبه مقدار حقیقی در بردارهای حالت پایه‌ای باید درایه هریک از آنها را در جای درست خود در ماتریس پایانی قرار دهیم و هر مقدار حقیقی را در درایه درست آن در ماتریس قرار دهیم. این کار در تابع finalize انجام می‌شود.

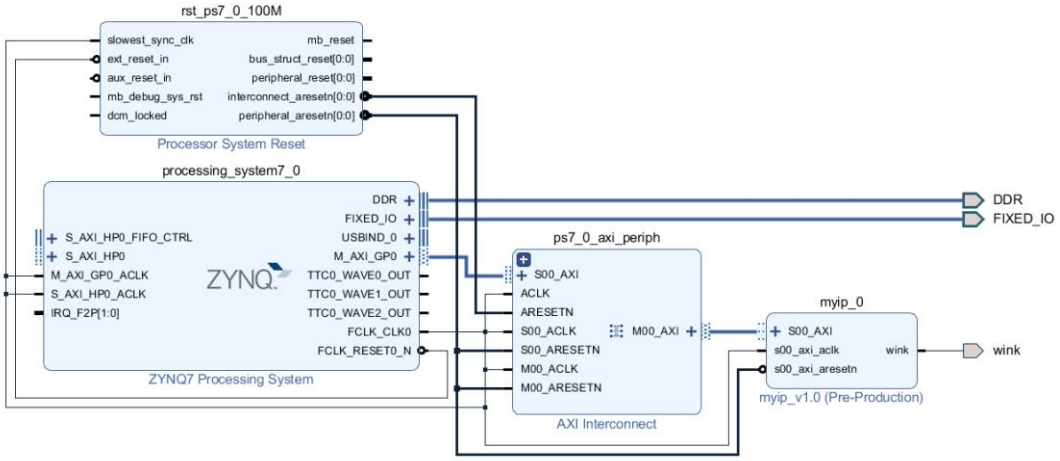
```
struct Matrix finalize(struct Matrix *real_ems, struct Matrix em){ … }
```

۴.۲.۳ اتصال سخت‌افزار و نرم‌افزار

تا قبل از این مرحله پیاده‌سازی کل الگوریتم انجام گرفته است. فاز اصلی آن به صورت سخت‌افزاری و فاز پیش‌پردازش و پس‌پردازش به صورت نرم‌افزاری پیاده‌سازی شدند. در این مرحله لازم است تا بین این سه بخش اتصال برقرار کنیم. همانطورکه پیش‌تر هم اشاره شد پیاده‌سازی سخت‌افزاری بر روی منطق برنامه‌پذیر بورد اجرا می‌شود و پیاده‌سازی نرم‌افزاری بر روی سیستم پردازش‌گر و برای اتصال این دو می‌توان از استاندارد AXI4 استفاده کرد. محیط ویوادو هنگام اضافه کردن IP جدید به سیستم طراحی

شده گزینه اضافه کردن واسط AXI4 را به صورت جداگانه در اختیار ما قرار می‌دهد و می‌توان بین سه نوع آن انتخاب کرد. در این پروژه ابتدا سعی بر آن بود که از دو نوع واسط AXI4، یک AXI4-Lite برای تولید و استفاده از وقفه و یک AXI4 (آنچه در ویوادو AXI4 full نامیده می‌شود)، به عنوان واسط اصلی استفاده شود. البته AXI4 full پیچیدگی بسیار فراتری نسبت به دو نوع دیگر دارد و اتصال آن به اجزای سیستم نیازمند دخیل شدن اجزای جانبی زیادی است. از این رو استفاده از آن باعث بیشتر شدن پیچیدگی این پروژه می‌گردید و حتی برای شبیه‌سازی و آزمودن درستی عملکرد باس با استفاده مایکروبلیز مشاهده شد که منابعی همچون حافظه رم را خیلی بیشتر از حالت‌های دیگر نیاز دارد. علت اصلی این انتخاب این بود که اگر بخواهیم در برنامه‌های آتی برای این پروژه، حالت‌های پایه‌ای شبکه‌های خیلی بزرگ را محاسبه کنیم که در آنها تعداد بردارهای حالت پایه‌ای بالغ بر هزار عدد است و هرکدام شامل چند ده یا چند صد بیت هستند، بتوان انتقال داده‌های خروجی را به راحتی انجام داد. در مرحله بعد تلاش شد برای پاسخگویی به این نیاز از AXI4 Stream استفاده شود ولی باتوجه به اضافه شدن DMA و اجزای جانبی غیرضروری در این پروژه در نهایت از AXI4-lite استفاده شد که نیازمندی‌های این پروژه را نیز برطرف می‌نماید.

پس از اضافه کردن واسط AXI4-lite یک پوشه به پروژه اضافه شد که در آن دو فایل توصیف IP به زبان VHDL قرار دارد که یکی از این دو توصیف، پیاده‌سازی برده در AXI4-lite و دیگری توصیف IP است که در آن از AXI4-lite نمونه تولید شده است. این کدها با استفاده از گزینه edit IP packager قابل تغییر هستند و می‌توان برای خواندن و نوشتن در ثبات‌های آن، کد را تغییر داد و اتصال بین پیاده‌سازی سخت‌افزار و پردازنده نرم‌افزاری را برقرار کرد. در شکل ۳-۲ دیاگرام بلوکی سیستم طراحی شده در محیط ویوادو نشان داده شده است. دو بخش AXI Interconnect و PS Reset به صورت اتوماتیک به دیاگرام اضافه می‌شوند.



شکل ۳-۲ نمودار بلوکی سیستم طراحی شده در محیط ویوادو

در تکه کد زیر، نحوه خواندن از ثبات‌ها در بدنه IP نشان داده شده است. slv_regها همان ثبات‌هایی هستند که امکان انتقال داده را فراهم می‌کنند. همانطور که در کد مشخص شده است، مقدار بیت صفر در ثبات slv_reg20 به ورودی hw_call اعمال می‌شود و به این ترتیب مشخص می‌گردد که کار نرم‌افزار در فاز پیش‌پردازش پایان یافته است و سخت‌افزار می‌تواند با داده‌های معتبر به انجام محاسبات بپردازد. مقادیر سطرهای بخش حقیقی ماتریس خروجی در فاز پیش‌پردازش، در ثبات‌های slv_reg40 تا slv_reg47 قرار گرفته اند.

top_init : top port map (S_AXI_ACLK, hw_call, pre_mat1, pre_mat2, pre_mat3, pre_mat4, pre_mat5, pre_mat6, pre_mat7, pre_mat8, slv_reg0(0), em_cols, em_rows, em_data);

```
hw_call <= slv_reg20(0);
pre_mat1 <= slv_reg40(25 downto 0);
pre_mat2 <= slv_reg41(25 downto 0);
pre_mat3 <= slv_reg42(25 downto 0);
pre_mat4 <= slv_reg43(25 downto 0);
pre_mat5 <= slv_reg44(25 downto 0);
pre_mat6 <= slv_reg45(25 downto 0);
pre_mat7 <= slv_reg46(25 downto 0);
pre_mat8 <= slv_reg47(25 downto 0);
```

در تکه کد زیر، نحوه نوشتن در ثبات‌ها در بدنه IP نشان داده شده است. خروجی‌های entity اصلی ما که شامل سیگنال‌های sw_call، em_cols و em_rows هستند به ترتیب در ثبات‌های slv_reg0، slv_reg1 و slv_reg2 نوشته شده‌اند. سیگنال خروجی em_data به صورت ۶تایی جدا شده و در ۶ بیت کم‌ارزش ثبات‌ها قرار می‌گیرند.

```
sw_call <=   slv_reg0(0);  --sw_call
 slv_reg0(31 downto 1) <= (others => '0');

 slv_reg1(11 downto 0) <= std_logic_vector(to_unsigned(em_cols, 12));    --em_cols
 slv_reg1(31 downto 12) <= (others => '0');

 slv_reg2(2 downto 0) <= std_logic_vector(to_unsigned(em_rows, 3));    --em_rows
 slv_reg2(31 downto 3) <= (others => '0');

 slv_reg3(5 downto 0) <= em_data(1 to 6);    --1st em vector
 slv_reg3(31 downto 6) <= (others => '0');

 slv_reg4(5 downto 0) <= em_data(7 to 12);   --2nd em vector
 slv_reg4(31 downto 6) <= (others => '0');

 slv_reg5(5 downto 0) <= em_data(13 to 18);  --3rd em vector
 slv_reg5(31 downto 6) <= (others => '0');

 slv_reg6(5 downto 0) <= em_data(19 to 24);  --4th em vector
 slv_reg6(31 downto 6) <= (others => '0');

 slv_reg7(5 downto 0) <= em_data(25 to 30);  --5th em vector
 slv_reg7(31 downto 6) <= (others => '0');

 slv_reg8(5 downto 0) <= em_data(31 to 36);  --6th em vector
 slv_reg8(31 downto 6) <= (others => '0');
```

در کد زیر نیز نحوه فراخوانی توابع پیش‌پردازش، پس‌پردازش و همچنین استفاده از ثبات‌ها در کد نرم‌افزاری مشخص شده است. در این کد برای نوشتن در ثبات‌های اتصال AXI4-lite در کد C از تابع Xil_Out32 و برای خواندن از ثبات‌ها از تابع Xil_In32 استفاده شده است که مقدار هر ثبات را به صورت یک عدد صحیح می‌خواند. البته باید توجه داشت که پس از پایان اجرای توابع پیش‌پردازش، مقدار ۱ در آدرس مربوط به ثبات slv_reg20 نوشته می‌شود تا زمان شروع عملیات سخت‌افزار را اعلام کند. همچنین پیش از خواندن ثبات‌هایی که مقادیر ماتریس حالت‌های پایه‌ای و اندازه این ماتریس را مشخص می‌کنند، باید با کنترل ثباتی که مقدار sw_call در آن ذخیره شده است، از معتبر بودن مقادیر داخل ثبات‌های نامبرده اطمینان حاصل کنیم. در این کد برای تبدیل مقادیر اعشاری به نمایش دودویی، از تابع to_binary استفاده می‌کنیم. همچنین برای به دست آوردن نمایش دودویی حالت‌های پایه که به صورت عدد صحیح در ثبات‌ها ذخیره شده‌اند از تابع int2bin استفاده شده است.

```
int main()
{
   init_platform();
   struct Matrix pre = pre_process_sample0();
   int *binary = to_binary(pre);
   for(int i = 0; i < (pre.row - pre.col) * pre.col; i++){
      Xil_Out32(base + ((40 + i) * 4 ) , binary[i]);
   }
   Xil_Out32(base + 20 * 4 , 1);
   int sw_call = 0;
   struct Matrix mp;
   mp.row = 6;
   mp.col = 6;
   float **mp_p = malloc(mp.row * sizeof(float *));
   for (int i = 0; i < mp.row; i++)
      mp_p[i] = malloc(mp.col * sizeof(float));
   mp.mat = mp_p;
   printf("waiting for hardware computations...\n");
   while (1)
   {
      sw_call = Xil_In32(base);
      if (sw_call == 1)
         break;
   }
   printf("hardware computations done.\n");
   int em;
   char *em_binary;
   for (int i = 0; i < mp.col; i++)
   {
      em = Xil_In32(base + ((i + 3) * 4));
      em_binary = int2bin(em);
      int cell;
      for (int j = 26; j < 32; j++)
      {
         cell = em_binary[j] - 48;
          mp_p[j - 26][i] = cell;
      }
   }
   struct Matrix result = post_process_sample0(mp);
   printf("post process completed.\n");
   printf("EFMS %d x %d:\n",result.row, result.col);
   print_mat_t(result.mat, result.row, result.col);
   cleanup_platform();
   return 0;
}
```

در پایان فاز اصلی، خروجی حالت‌های پایه‌ای به صورت یک std_logic_vector تعریف می‌شود و به ثبات‌های مشخصی از AXI4-lite متصل می‌شود. پس از اینکه مقدار sw_call که نشان‌دهنده پایان کار سخت‌افزار است برابر با یک شد، لازم است تا نرم‌افزار مقادیر قرار گرفته در ثبات‌های مشخص را بخواند. پس از خواندن مقادیر هر بردار و تشکیل دوباره ماتریس حالت‌های پایه‌ای، توابع پیاده‌سازی شده در فاز پس‌پردازش فراخوانی می‌شوند تا خروجی نهایی تولید شود.

۵.۲.۳ پیاده‌سازی بر روی زدبورد

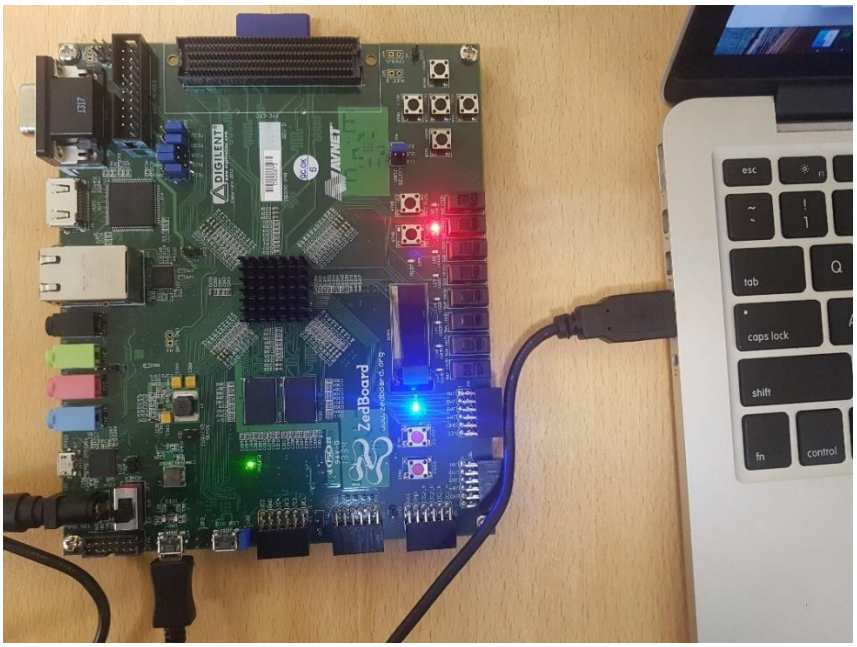
آخرین مرحله پیاده‌سازی پروژه، اجرای سیستم بر روی زدبورد است. لازمه انجام این کار در بخش پیاده‌سازی سخت‌افزار انجام implementation و تولید رشته بیت می‌باشد. البته باید توجه داشت که پس از مرحله سنتز باید محدودیت‌های لازم برای نمایش خروجی‌ها بر روی بورد و یا دریافت ورودی‌ها از روی بورد را مشخص کنیم. به طور مثال هنگامی که کار سخت‌افزار به اتمام می‌رسد و مقدار sw_call یک می‌شود، یک LED بر روی بورد روشن شود و برای این کار باید در فایل constraints.xdc دستورات زیر وارد شود. البته این کار را با استفاده از محیط گرافیکی نرم‌افزار ویوادو نیز می‌توان انجام داد.

```
set_property PACKAGE_PIN T21 [get_ports sw_call]
set_property IOSTANDARD LVCMOS33 [get_ports sw_call]
```

پس از تولید رشته بیت و اعمال دستور export hardware و ساختن پروژه نرم‌افزاری در محیط SDK باید فایل BOOT.bin را تولید کنیم. برای ساختن این فایل باید سه فایل را به ترتیب در پنجره مربوط به ایجاد فایل بوت در محیط SDK، اضافه کنیم تا در پارتیشن‌های این فایل قرار بگیرند. اولین فایل در این قسمت فایل elf. مربوط به بارگذاری بوت مرحله اول[1] است. فایل دیگر همان رشته بیت تولید شده برای سخت‌افزار است و آخرین فایل elf. مربوط به کد نرم‌افزاری است. پس از تولید فایل بوت، آن را بر روی کارت حافظه SD ذخیره می‌کنیم. پس از نصب کارت حافظه در درگاه بورد و روشن کردن بورد، برنامه شروع به اجرا می‌کند. البته باید توجه داشت که جامپرهای روی بورد به گونه‌ای تنظیم شده باشند که برنامه‌ریزی بورد را از طریق خواندن محتوای کارت حافظه امکان‌پذیر سازند. در کد نرم‌افزاری ارتباط

[1] first stage boot loader

UART راه‌اندازی شده است و با استفاده از توابع print می‌توانیم داده مورد نظر را از طریق UART و به صورت سریال به یک سیستم کامپیوتری بفرستیم. در صورتی که نرم‌افزار پورت سریال را اجرا کنیم و تنظیمات UART را به صورت دقیق اعمال کنیم می‌توانیم خروجی نهایی را بر روی صفحه نمایش کامپیوتر مشاهده کنیم. در شکل ۳-۳ وضعیت بورد و اتصال آن با کامپیوتر نشان داده شده است.



شکل ۳-۳ وضعیت بورد و نحوه اتصال آن به کامپیوتر

۴

فصل چهارم

صحت‌سنجی و بررسی نتایج

۱.۴ ابزار مورد استفاده

برای بررسی صحت سیستم پیاده‌سازی شده، لازم است تا یک شبکه استاندارد و ترجیحا فشرده شده را به عنوان ورودی به سیستم بدهیم و خروجی سیستم را با مقدار اصلی حالت‌های پایه‌ای آن مقایسه کنیم. ورودی‌ای که در این پروژه مورد آزمون قرار گرفت شبکه tricarboxylic-acid-cycle می‌باشد. این شبکه جزئی از یک شبکه شناخته شده است و اندازه آن برای بررسی دقیق خروجی سیستم پیاده‌سازی شده مناسب است. در بخش ۲,۴ درباره جزئیات این شبکه توضیح داده خواهد شد. برای یافتن شبکه فشرده شده مطلوب، ماتریس استوکیومتری آن را به عنوان ورودی به یکی از رابط‌های برنامه‌نویسی نرم‌افزار[۱] برنامه CellNetAnalyzer با نام CompressNetwork می‌دهیم. ساختاری که به عنوان خروجی آن تولید می‌شود، شبکه فشرده شده‌ای است که به صورت یک ماتریس با اندازه کوچکتر توصیف می‌شود. با استفاده از نتیجه فشرده شده و اعمال آن به عنوان ورودی ماتریس استوکیومتری برنامه متاتول یا EFMTool، می‌توان ماتریس حالت‌های پایه‌ای را در ساختار خروجی آن‌ها مشاهده کرد. در ادامه توضیحاتی درباره جزئیات هر یک از ابزار مورد استفاده در آزمون پروژه ارائه می‌شود.

۱.۱.۴ ابزار متاتول [۱۳]

متاتول یک برنامه پیاده‌سازی شده برای اکتاو[۲] و متلب است که کار محاسبه ماتریس فضای پوچ، حالت‌های پایه‌ای و دیگر ویژگی‌های ساختاری شبکه‌های متابولیکی را انجام می‌دهد. آخرین نسخه این برنامه که در این پروژه نیز از آن استفاده شده است نسخه ۵/۱ می‌باشد. برای استفاده از این برنامه ابتدا باید فایل‌های آن که به صورت متن‌باز قابل دسترس هستند دانلود شوند. سپس از طریق برنامه متلب وارد پوشه مربوط به برنامه می‌شویم. اولین دستوری که باید اجرا شود فایل ورودی با فرمت استاندارد را دریافت می‌کند و محتوای آن را در فیلدهای مختلف یک ساختمان داده ذخیره می‌کند:

---

[1] Application Programming Interface (API)

[2] Octave

```
ex= parse('example.dat');
```

در مرحله بعد با استفاده از دستور زیر می‌توان محاسبات لازم را انجام داد و خروجی مورد نظر را دریافت کرد:

```
ex= Metatool(ex);
```

متغیر ex دارای تعدادی فیلد می‌باشد که توضیح هریک به شرح زیر است:

st: ماتریس استوکیومتری که در آن ردیف‌ها معادل متابولیت‌ها و ستون‌ها معادل واکنش‌ها هستند

irrev_react: برداری که در آن به ازای هر واکنش برگشت‌ناپذیر مقدار آن ۱ و به ازای هر واکنش برگشت‌پذیر مقدار آن ۰ است

kn: ماتریس کرنل (فضای پوچ) ماتریس استوکیومتری

sub: ماتریس زیرمجموعه‌ها که ردیف‌ها معادل زیرمجموعه‌های متابولیکی و ستون‌ها معادل واکنش‌ها در st هستند

rd: ماتریس استوکیومتری فشرده شده

irrev_rd: برداری که در آن به ازای هر واکنش برگشت‌ناپذیر مقدار آن ۱ و به ازای هر واکنش برگشت‌پذیر مقدار آن ۰ است

rd_ems: حالت‌های پایه‌ای مربوط به ماتریس فشرده شده rd

irrev_ems: برداری که در آن به ازای هر حالت پایه‌ای برگشت‌ناپذیر مقدار آن ۱ و به ازای هر حالت پایه‌ای برگشت‌پذیر مقدار آن ۰ است

int_met: نام متابولیت‌های داخلی

ext_met: نام متابولیت‌های خارجی

react_name: نام واکنش‌ها

در میان موارد نامبرده شده در قسمت بالا، آنهایی که به صورت ضخیم مشخص شده‌اند، به محض خواندن فایل ورودی ایجاد می‌شوند و بقیه موارد، نتیجه محاسبات می‌باشند.

البته ابزار متاتول این امکان را به ما می‌دهد تا بدون داشتن فایل استاندارد و تنها با داشتن ماتریس استوکیومتری شبکه متابولیکی و برگشت‌پذیری یا برگشت‌ناپذیری واکنش‌های آن، حالت‌های پایه‌ای را محاسبه کنیم. برای انجام این کار از دستورات زیر می‌توان استفاده کرد.

```
net.st = [1 1 -1 0; 0 0 1 -1];
net.irrev_react = [1 0 1 1];
net = metatool(net);
```

۲.۱.۴ ابزار CellNetAnalayzer [۱۴]

یکی دیگر از ابزارهایی که با متلب و برای تحلیل شبکه‌های متابولیکی استفاده می‌شود، CellNetAnalyzer یا به اختصار CNA است. از این ابزار می‌توان با استفاده از خط فرمان به واسطه APIهای طراحی شده آن و یا با استفاده از واسط گرافیکی استفاده کرد. از جمله امکاناتی که APIهای این ابزار در اختیار کاربران قرار می‌دهد، نوشتن و اصلاح ساختار شبکه‌های متابولیکی و فراخوانی توابع مورد نظر بدون استفاده از واسط گرافیکی است. البته باید توجه داشت که پیش از استفاده از هر یک از توابع مورد نظر که در CNA پیاده‌سازی شده‌اند، لازم است تا برنامه با استفاده از دستور startcna مقداردهی اولیه شود. این دستور همه پوشه‌های مورد نیاز را به مسیرهای متلب اضافه می‌کند. در صورتی که بخواهیم از خط فرمان استفاده کنیم دستور startcna(1) مقداردهی اولیه را برای ما انجام می‌دهد. یک شبکه متابولیکی در CNA با استفاده از یک ساختار با نام cnap مشخص می‌شود. این متغیر دارای اجزای مختلفی است که برخی از آنها باید به صورت مستقیم توسط کاربر مقداردهی شوند و برخی دیگر پس از اجرای توابع خود CNA مقدار می‌گیرند.

در این پروژه برای فشرده‌سازی شبکه‌های متابولیکی از CNAcompressMFNetwork استفاده شد. این تابع با استفاده از ماتریس استوکیومتری شبکه و برگشت‌پذیری واکنش‌ها، به چهار روش شبکه‌ها را فشرده می‌کند:

- از بین بردن واکنش‌های بلوکه شده
- از بین بردن متابولیت‌های وابسته
- یکی کردن زیرمجموعه‌های آنزیمی
- یکی کردن نقاط خفگی (متابولیت‌هایی که طی یک واکنش تولید و طی چند واکنش مصرف می‌شوند و یا برعکس)

این نوع فشرده‌سازی یکسان بودن شبکه فشرده شده را با شبکه اصلی در زمینه‌هایی چون مجموعه حالت‌های پایه‌ای تضمین می‌کند.

فراخوانی این تابع به صورت زیر انجام می‌گیرد:

```
[redsmat,irrev,reacidx,metidx,cnapcomp]= CNAcompressMFNetwork(cnap);
```

redsmat در خروجی‌های این تابع، همان ماتریس شبکه فشرده شده است. irrev یک آرایه با مقادیر ۰ و ۱ است که بازگشت‌ناپذیر بودن واکنش‌ها را مشخص می‌کند. reacidx لیست نام‌های واکنش‌ها و metidx لیست نام‌های متابولیت‌هاست.

برای ساختار cnap که ورودی تابع را تشکیل می‌دهد مقادیر cnap.stoichMat، cnap.numr و cnap.reacMin لازم است توسط کاربر مقداردهی شوند. این مقادیر به ترتیب معادل ماتریس استوکیومتری شبکه، تعداد واکنش‌های شبکه و برداری است که برگشت‌پذیر بودن یا نبودن واکنش‌ها را مشخص می‌کند.

۳.۱.۴ ابزار EFMTool [۱۵]

EFMTool ابزار دیگری برای محاسبه حالت‌های پایه‌ای شبکه‌های متابولیکی است. پیاده‌سازی این نرم‌افزار توسط زبان جاوا انجام شده و در نهایت با متلب کامل شده است.

در این ابزار همانند مدلسازی‌ای که در [۳] نیز به آن اشاره شده است، محاسبه حالت‌های پایه‌ای با استفاده از روش شمارش شعاع‌های فرین یک مخروط چند وجهی انجام می‌گیرد.

آخرین نسخه منتشر شده برای EFMTool، ۴/۷/۱ می‌باشد. برای استفاده از این ابزار باید تابع CalculateFluxModes در متلب فراخوانی شود. این تابع پارامترهای ورودی را به شکل‌های مختلفی قبول می‌کند. با استفاده از دستور help CalculateFluxModes تمامی حالاتی را که می‌توان به این تابع ورودی داد، می‌توان مشاهده کرد.

یکی از ساده‌ترین راه‌های مشخص کردن ورودی، مشخص کردن ماتریس استوکیومتری و بازگشت‌پذیری واکنش‌هاست. در این ابزار بر خلاف متاتول واکنش‌های برگشت‌پذیر با مقدار ۱ و واکنش‌های برگشت‌ناپذیر با مقدار ۰ مشخص می‌شوند. به طور مثال داریم:

```
stru.stoich = [1 1 -1 0; 0 0 1 -1];
stru.reversibilities = [0 1 0 0];

mnet = CalculateFluxModes(stru);
```

از دیگر روش‌های اعمال پارامترهای ورودی توصیف فرمول واکنش‌هاست که به صورت زیر امکان‌پذیر است:

```
stru.metaboliteNames = {'A', 'B', 'C', 'D', 'E', 'P'};
stru.reactionNames = {'R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10'};
rformulas = {
'--> A'
'<--> B'
'P -->'
'E -->'
'A --> B'
'A --> C'
'A --> D'
'B <--> C'
'B --> P'
'C + D --> E + P'
};
mnet = CalculateFluxModes(rformulas)
```

خروجی این تابع نیز یک ساختار است که از اجزای زیر تشکیل شده است:

metaboliteNames: نام متابولیت‌ها که اگر کاربر آنها را وارد نکند به ترتیب از M1 نام‌گذاری می‌شود

reactionNames: نام واکنش‌ها که اگر کاربر آنها را وارد نکند به ترتیب از R1 نام‌گذاری می‌شود

reactionFormulas: فرمول واکنش‌ها که بر حسب نام متابولیت‌ها نوشته می‌شود

reactionLowerBounds: کران پایین واکنش‌ها که برای واکنش‌های برگشت‌ناپذیر ۰ و برای واکنش‌های برگشت‌پذیر منفی بی‌نهایت است

reactionUpperBounds: کران بالای واکنش‌ها که معمولا برای همه واکنش‌ها بی‌نهایت است

stoich: ماتریس استوکیومتری شبکه متابولیکی

efms: حالت‌های پایه‌ای برای شبکه توصیف شده

در این پروژه از این ابزار برای کنترل مشابهت نتایج خروجی استفاده شد.

۲.۴ شبکه متابولیکی tricarboxylic-acid-cycle–glyoxylate-shunt

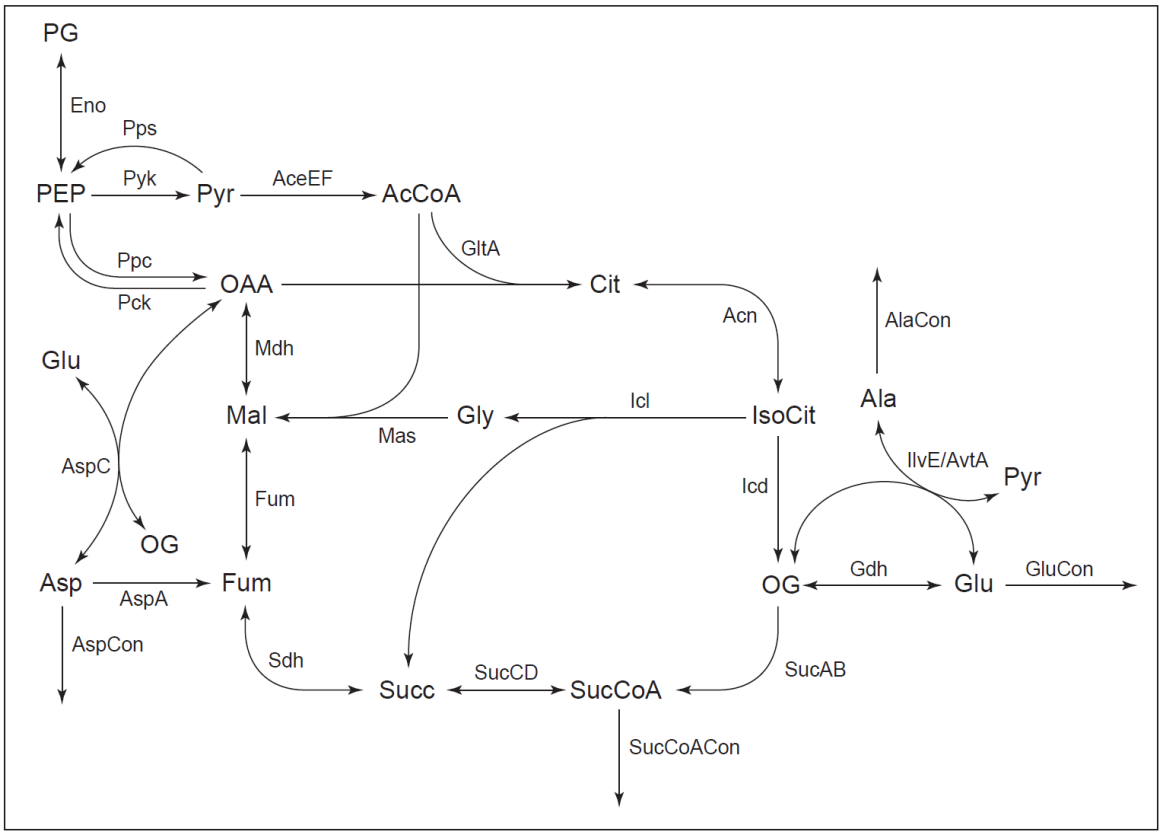
این شبکه یک بخش خاص از یک شبکه بزرگتر به نام E coli می‌باشد. E coli یکی از گونه‌های خاص باکتری ایشریشیا[1] می‌باشد. این باکتری به راحتی در محیط آزمایشگاهی قابل رشد دادن است و به همین خاطر به یک مدل ارگانیسم اصلی برای انجام تحقیقات بیوتکنولوژی و میکروبیولوژی تبدیل شده و در حال حاضر در مقایسه با دیگر گونه‌ها اطلاعات بیشتری از آن در اختیار است.

به دلیل گستردگی بیش از اندازه شبکه E coli آن را به بخش‌های متعددی تقسیم کرده‌اند و محاسبات لازم را به صورت مجزا بر روی هر یک از این بخش‌ها انجام داده و در نهایت نتیجه کلی را به شبکه E coli تعمیم می‌دهند. بخشی از این شبکه که مورد هدف این پروژه می‌باشد، شامل حلقه تری‌کربوکسیلیک‌اسید (مجموعه‌ای از واکنش‌ها در ارگان‌های هوازی برای آزادسازی انرژی ذخیره شده)، حلقه گلیوکسیلات (شکل دیگری از حلقه تری‌کربوکسیلیک‌اسید که در گیاهان و باکتری‌ها وجود دارد) و واکنش‌های مجاور سوخت‌وساز آمینواسید می‌شود. تعداد متابولیت‌های داخلی این شبکه برابر با ۱۶ و تعداد واکنش‌های آن برابر با ۲۴ می‌باشد که ۹ تا از آنها واکنش‌های برگشت‌پذیر هستند. نمودار این شبکه در شکل ۴-۱ نشان داده شده است. همانطور که در شکل مشخص شده است، واکنش‌های برگشت‌پذیر با پیکان‌های دوطرفه نمایش داده شده‌اند و از جمله آنها می‌توان به واکنش‌های AspC، Mdh و Acn اشاره کرد. توصیف این شبکه مطابق با استانداردهای متاتول در بخش پیوست پایان‌نامه آورده شده است.

تعداد حالت‌های پایه‌ای محاسبه شده برای این شبکه پس از اعمال مراحل فشرده‌سازی ۱۶ عدد است که هر کدام یک بردار ۱۵تایی هستند. این شبکه به عنوان شبکه اصلی برای آزمون و بررسی نتایج سیستم پیاده‌سازی شده در این پروژه استفاده می‌شود. در بخش بعدی به بیان نتایج سیستم و مقایسه آنها با نتایج حاصل از ابزار متاتول بر روی ورودی این شبکه می‌پردازیم.

---

[1] Escherichia



شکل ۴-۱ نمودار شبکه **tricarboxylic-acid-cycle** [۱۶]

۳.۴ مقایسه و بررسی نتایج

در این بخش به بررسی نتایج سیستم برای دو شبکه، یکی شبکه نمونه در [۳] و دیگری شبکه نشان داده شده در شکل ۴-۱ می‌پردازیم.

۱.۳.۴ ماتریس استوکیومتری شبکه

نمودار شبکه نمونه در شکل ۲-۴ قابل مشاهده است. ماتریس استوکیومتری که این شبکه را توصیف می‌کند در شکل ۴-۲ مشخص شده است. این شبکه در ابتدا ۴ متابولیت و ۷ واکنش دارد که یکی از این واکنش‌ها برگشت‌پذیر است.

cnap2.stoichMat

	1	2	3	4	5	6	7
1	1	-1	-1	0	0	0	0
2	0	1	0	-1	-1	-1	0
3	0	0	1	0	1	-1	0
4	0	0	0	0	0	1	-1

شکل ۴-۲ ماتریس استوکیومتری شبکه نمونه

نمودار شبکه tricarboxylic-acid-cycle نیز در شکل‌های ۴-۳ و ۴-۴ نشان داده شده است. این شبکه دارای ۱۶ عدد متابولیت و ۲۴ عدد واکنش است که ۹ تا از آنها برگشت‌پذیر می‌باشد.

cnap1.stoichMat

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	-1	-1	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	-1	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0	0	0	0
6	0	0	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0	0	0
9	0	0	0	1	-1	0	0	0	0	0	0	0
10	0	0	0	-1	0	-1	0	0	0	0	0	0
11	0	0	1	0	0	0	0	0	0	0	0	0
12	0	0	-1	0	0	0	0	0	0	-1	1	0
13	0	1	-1	0	0	0	-1	0	0	0	0	0
14	0	-1	1	0	-1	0	1	0	0	0	0	0
15	1	-1	0	0	0	0	0	0	0	0	0	-1
16	-1	0	0	0	0	0	0	0	0	1	-1	1

شکل ۴-۳ بخش اول ماتریس استوکیومتری شبکه tricarboxylic-acid-cycle

cnap1.stoichMat

	13	14	15	16	17	18	19	20	21	22	23	24
1	0	-1	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	1	0	0
3	-1	0	0	0	0	0	0	0	0	-1	1	-1
4	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	1	-1	0	0	0
6	0	0	0	0	0	0	1	-1	0	0	0	0
7	0	0	0	0	0	1	-1	0	0	0	0	0
8	0	0	-1	0	0	-1	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1	-1	1
10	0	0	0	0	1	0	0	0	0	0	0	0
11	0	0	0	0	-1	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	1	-1	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	1	0	0	1	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	-1
16	0	0	0	1	0	0	0	0	0	0	0	0

شکل ۴-۴ بخش دوم ماتریس استوکیومتری شبکه tricarboxylic-acid-cycle

۲.۳.۴ نتیجه فشرده‌سازی

با استفاده از تابع فشرده‌سازی در CellNetAnalyzer و وارد کردن دستوراتی مشابه دستور زیر، ماتریس فشرده شده را به دست می‌آوریم.

```
st2 = [1,-1,-1,0,0,0,0
    0,1,0,-1,-1,-1,0
    0,0,1,0,1,-1,0
    0,0,0,0,0,1,-1];

rev2 = [0,0,0,0,-Inf,0,0];
cnap2.stoichMat = st2;
cnap2.numr = 7;
cnap2.reacMin= rev2';

cnap = CNAgenerateMFNetwork(cnap2,1);
[redsmat,irrev,reacidx,metidx,cnapcomp]= CNAcompressMFNetwork(cnap);
disp(redsmat);
```

همانطور که در قسمت پایین مشاهده می‌شود شبکه به ۲ متابولیت و ۵ واکنش کاهش می‌یابد.

```
Size of original network: 4 internal metabolites, 7 reactions
Size of compressed network: 2 internal metabolites, 5 reactions
```

redsmat

2x5 double

	1	2	3	4	5
1	-1	0	1	0	1
2	-1	1	0	-1	-1

شکل ۴-۵ ماتریس فشرده شده شبکه نمونه

نتیجه فشرده‌سازی برای شبکه دوم نیز به صورت زیر است و در پایان تعداد متابولیت‌ها ۶ عدد و تعداد واکنش‌ها ۱۵ عدد است که از میان آنها ۵ واکنش برگشت‌پذیر وجود دارد.

```
Size of original network: 16 internal metabolites, 24 reactions
Size of compressed network: 6 internal metabolites, 15 reactions
```

6x15 double

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	1	-1	-2	0	0	0	1	-1
2	1	0	0	0	-1	-1	0	-1	-1	-1	1
3	0	0	-1	0	1	0	0	0	0	0	0
4	-1	1	0	0	0	1	0	0	0	0	0
5	0	-1	0	0	0	0	1	0	1	0	0
6	0	0	1	0	0	1	-1	0	0	0	0

شکل ۴-۶ ماتریس فشرده شده شبکه tricarboxylic-acid-cycle

۳.۳.۴ نتیجه فاز پیش‌پردازش

در این بخش ورودی مطلوب برای فاز اصلی محاسبه می‌شود که هدف از آن ارائه یک ماتریس مقداردهی شده با فرم $[\begin{matrix} I \\ K \end{matrix}]$ است. ترتیب عملیاتی که برای تولید خروجی مطلوب در این بخش قرار می‌گیرد به شرح زیر است که در آن r1 تا r5 ستون‌های ماتریس فشرده شده هستند و irrev آرایه‌ای است که واکنش‌ناپذیر بودن هر واکنش را مشخص می‌کند.

```
float *m_p[] = {r1, r2, r3, r4, r5};
struct Matrix m_rec = reconfigure(m, irrev);
struct Matrix m_null = null_space(m);
struct Matrix m_init = initialize(m_null);
```

ماتریس مورد نظر برای شبکه نمونه و شبکه tricarboxylic-acid-cycle به ترتیب به صورت شکل‌های ۴-۷ و ۴-۸ است.

```
1.0     0.0     0.0     0.0
0.0     1.0     0.0     0.0
0.0     0.0     1.0     0.0
0.0     0.0     0.0     1.0
0.5     0.5     -0.5    0.0
0.5     -0.5    -0.5    1.0
```

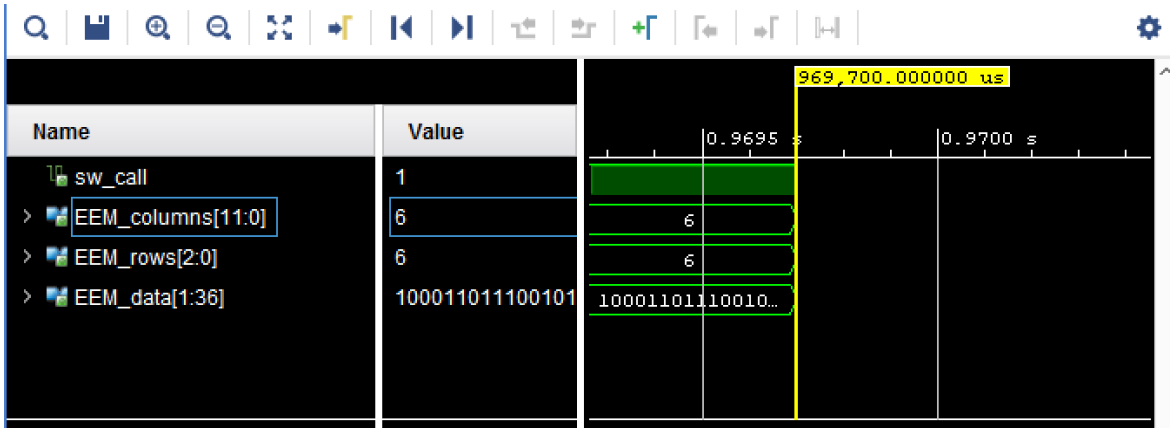
شکل ۴-۷ خروجی فاز پیش‌پردازش برای شبکه نمونه

```
1.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
0.0     1.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
0.0     0.0     1.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
0.0     0.0     0.0     1.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
0.0     0.0     0.0     0.0     1.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
0.0     0.0     0.0     0.0     0.0     1.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
0.0     0.0     0.0     0.0     0.0     0.0     1.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
0.0     0.0     0.0     0.0     0.0     0.0     0.0     1.0     0.0     0.0     0.0     0.0     0.0     0.0
0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     1.0     0.0     0.0     0.0     0.0     0.0
0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     1.0     0.0     0.0     0.0     0.0
0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     1.0     0.0     0.0     0.0
0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     1.0     0.0     0.0
0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     1.0
1.0     1.0     1.0     1.0     1.0     -1.0    0.0     1.0     0.0     1.0     0.0     0.0     0.0     0.0
1.0     -1.0    1.0     0.0     1.0     -1.0    0.0     1.0     0.0     0.0     0.0     1.0     0.0     0.0
0.0     1.0     -1.0    0.0     -1.0    1.0     0.0     -1.0    0.0     0.0     0.0     0.0     0.0     0.0
1.0     2.0     0.0     0.0     -1.0    1.0     0.0     1.0     1.0     0.0     0.0     0.0     1.0     0.0
1.0     0.0     1.0     1.0     1.0     -1.0    0.0     1.0     0.0     0.0     1.0     0.0     0.0     0.0
1.0     0.0     1.0     0.0     1.0     -1.0    0.0     1.0     0.0     0.0     0.0     0.0     0.0     1.0
```

شکل ۴-۸ خروجی فاز پیش‌پردازش برای شبکه tricarboxylic-acid-cycle

۴.۳.۴ نتیجه فاز اصلی

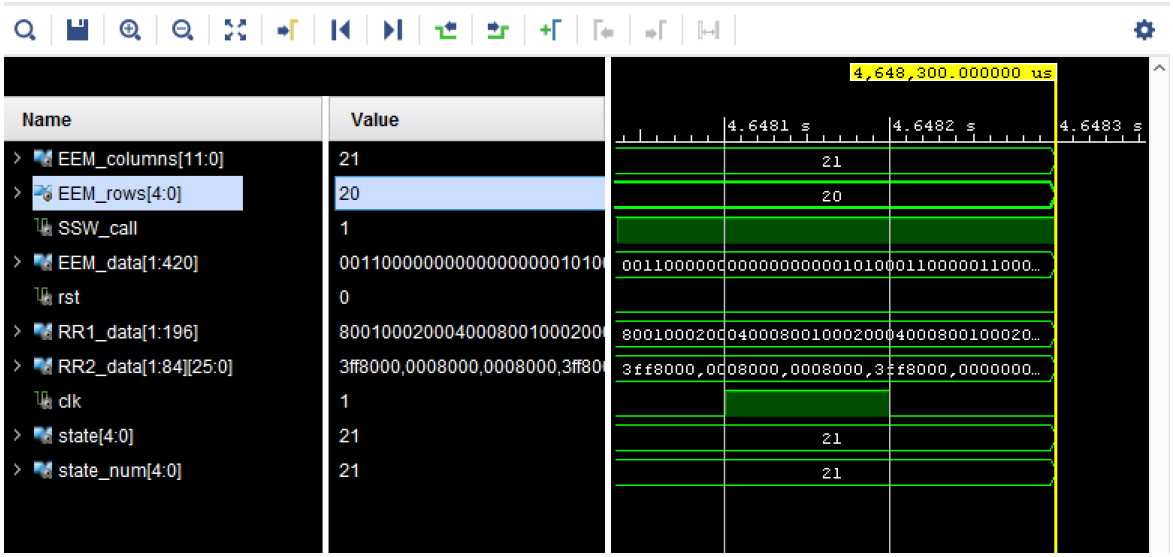
در این مرحله خروجی فاز پیش‌پردازش را می‌توان به عنوان ورودی و مقادیر اولیه به ماژول سخت‌افزاری داد تا فرم دودویی حالت‌های پایه‌ای را محاسبه نماید. شکل‌های ۴-۹ و ۴-۱۱ به ترتیب خروجی شبیه‌سازی پس از سنتز را برای دو شبکه نامبرده نشان می‌دهد. تعداد ستون‌های محاسبه شده در این فاز به تعداد واکنش‌های برگشت‌پذیر از تعداد حالت‌های پایه‌ای بیشتر است. خروجی این بخش به صورت یک std_logic_vector می‌باشد که حاصل یک بعدی شدن ستونی ماتریس R1 است.



شکل ۴-۹ خروجی شبیه‌سازی فاز اصلی برای شبکه نمونه

```
1  100011
2  011100
3  010110
4  000101
5  101000
6  110010
```

شکل ۴-۱۰ تبدیل خروجی شبیه‌سازی شبکه نمونه به آرایه دوبعدی



شکل ۴-۱۱ خروجی شبیه‌سازی فاز اصلی برای شبکه **tricarboxylic-acid-cycle**

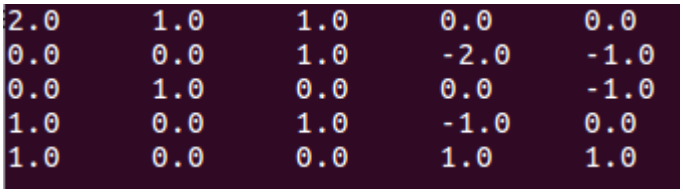
```
1  1000000000000110111
2  0100010000110100110
3  0100000100000100111
4  0001000000000100010
5  0100100000000100111
6  0000010001110100110
7  0000001000000000000
8  0110000000000100111
9  0000000010000000100
10 0000000010000100000
11 0000000000100000010
12 0000000000100010000
13 0000000000010000100
14 0000000000001000001
15 1000010000000001100
16 0001010000010100110
17 0010010000000000100
18 0100000000100101100
19 0000110000000000000
20 0000010100000000100
21 1100000000000101111
```

شکل ۴-۱۲ تبدیل خروجی شبیه‌سازی شبکه **tricarboxylic-acid-cycle** به آرایه دوبعدی

۵.۳.۴ نتیجه فاز پس‌پردازش

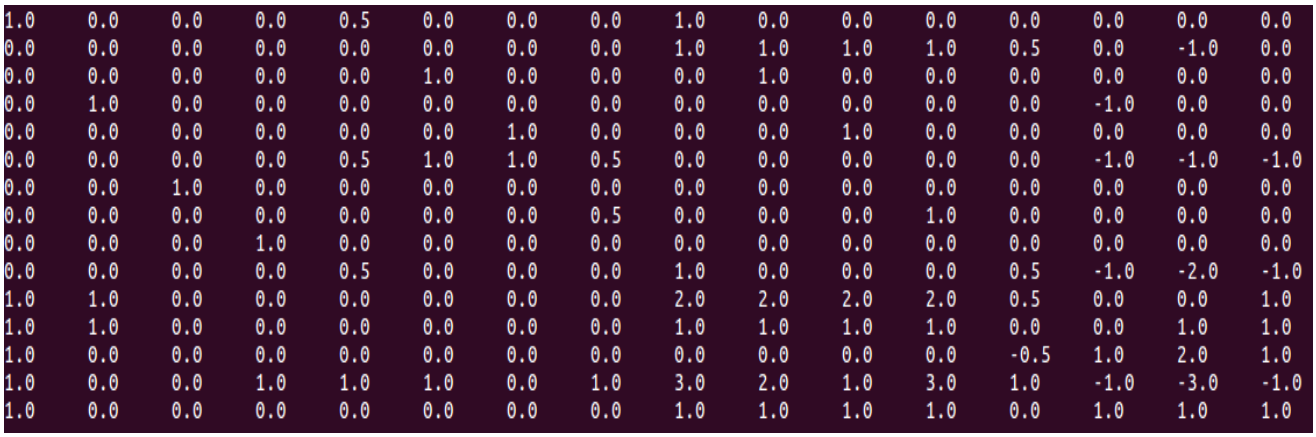
در شکل‌های ۴-۱۳ و ۴-۱۴ خروجی فاز پس‌پردازش نمایش داده شده است. بردارهای ماتریس نشان داده شده در این دو شکل، مقدار دقیق حالت‌های پایه‌ای شبکه‌های فشرده شده می‌باشند. این خروجی حاصل اعمال توابع پس‌پردازش بر روی ماتریس حاصل از فاز اصلی پروژه می‌باشد. ترتیب عملیاتی که در این بخش انجام می‌شود به صورت زیر است که در آن em ماتریس تولید شده پس از تبدیل آرایه یک بعدی به آرایه دوبعدی است.

```
struct Matrix em_r2c = remove2Cycles(em, rev_count);
struct Matrix em_bc = bc(em_r2c, rev_count);
struct Matrix *spaces = bin2real(em_bc, n);
struct Matrix final = finalize(spaces, em_bc);
```



2.0	1.0	1.0	0.0	0.0
0.0	0.0	1.0	-2.0	-1.0
0.0	1.0	0.0	0.0	-1.0
1.0	0.0	1.0	-1.0	0.0
1.0	0.0	0.0	1.0	1.0

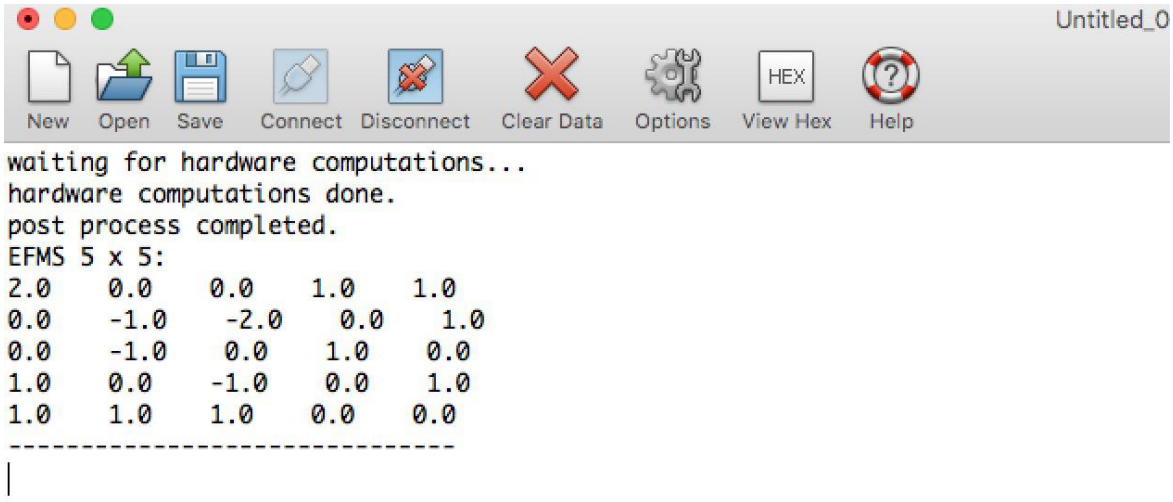
شکل ۴-۱۳ خروجی فاز پس‌پردازش برای شبکه نمونه



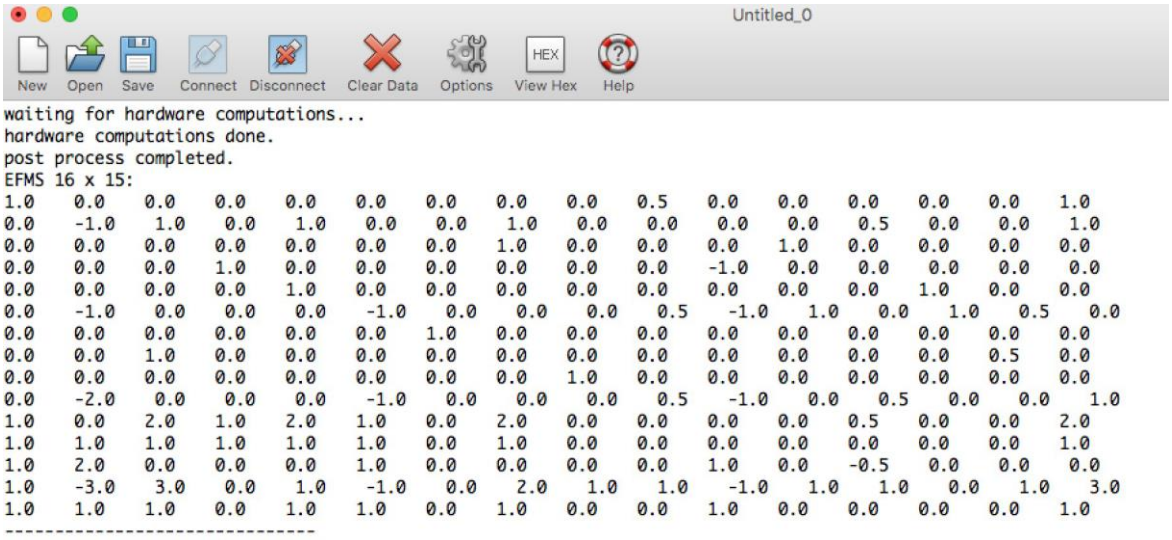
1.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	0.5	0.0	-1.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.5	1.0	1.0	0.5	0.0	0.0	0.0	0.0	0.0	-1.0	-1.0	-1.0
0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.5	-1.0	-2.0	-1.0
1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	2.0	2.0	2.0	0.5	0.0	0.0	1.0
1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.5	1.0	2.0	1.0
1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	3.0	2.0	1.0	3.0	1.0	-1.0	-3.0	-1.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0

شکل ۴-۱۴ خروجی فاز پس‌پردازش برای شبکه tricarboxylic-acid-cycle

شکل‌های ۴-۱۵ و ۴-۱۶ خروجی نهایی را بر روی صفحه نمایشگر کامپیوتر پس از انتقال سریال از سمت زدبورد برای دو شبکه مورد آزمون نشان می‌دهد.

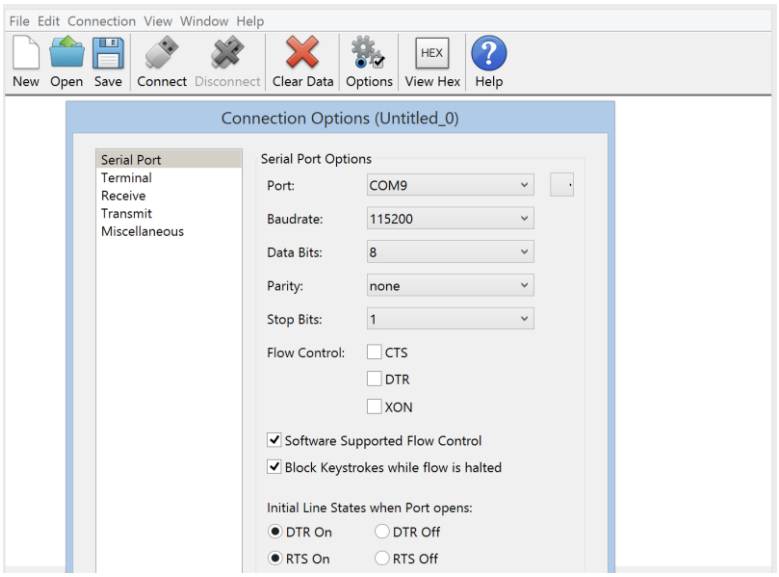


شکل ۴-۱۵ خروجی نهایی سیستم برای شبکه نمونه



شکل ۴-۱۶ خروجی نهایی سیستم برای شبکه tricarboxylic-acid-cycle

برای دریافت این خروجی از برنامه CoolTerm استفاده شد و همانطور که در شکل ۴-۱۷ نشان داده شده است با اعمال تنظیمات لازم همچون نرخ ۱۱۵۲۰۰ اتصال سریال برقرار می‌شود.

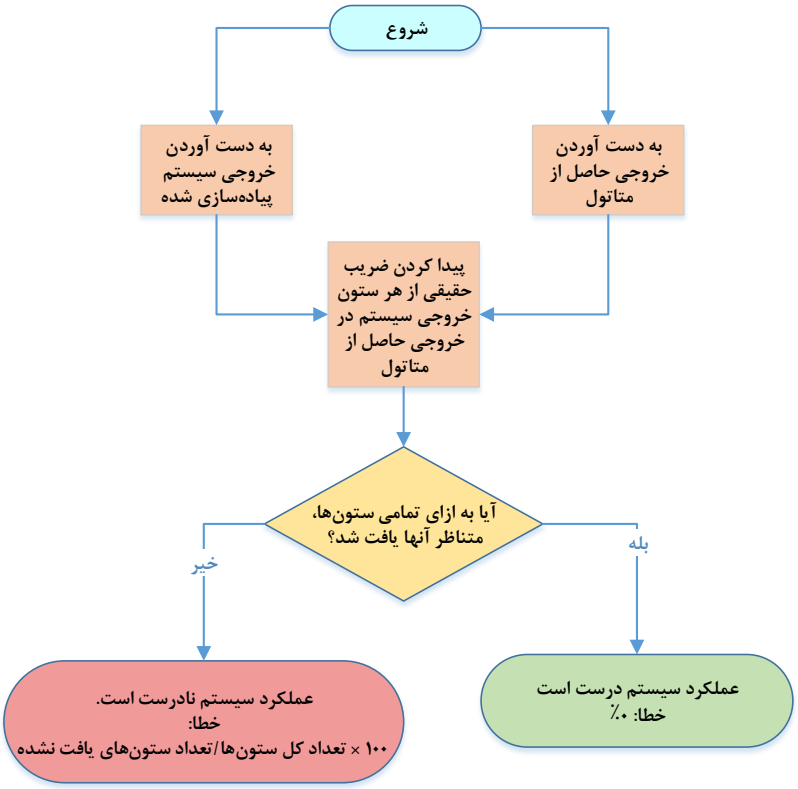


شکل ۴-۱۷ محیط برنامه CoolTerm

۶.۳.۴ مقایسه و نتیجه‌گیری

برای اطمینان از درستی سیستم، مقادیر حالت‌های پایه‌ای شبکه فشرده شده توسط سیستم را با خروجی‌هایی که نرم‌افزارهایی چون متاتول و EFMTool تولید می‌کنند مقایسه می‌کنیم. در صورتی که به ازای هر یک از بردارهایی که در ماتریس خروجی سیستم ما تولید می‌شوند، خود آن بردار و یا ضریبی

از آن را در خروجی ابزار نامبرده مشاهده کنیم، می‌توانیم از درستی عملکرد سیستم اطمینان حاصل کنیم. در شکل ۴-۱۸ نمودار بلوکی نحوه مقایسه نتایج نشان داده شده است.



شکل ۴-۱۸ نمودار بلوکی مقایسه نتیجه سیستم

همانطور که در شکل ۴-۱۹ مشخص شده است تعداد حالت‌های پایه‌ای برای شبکه نمونه ۵ عدد است که هر کدام دارای ۵ عنصر هستند که نشان دهنده تعداد واکنش‌هاست. حالت‌های پایه‌ای ستون‌های ماتریس rd_ems را تولید می‌کنند و این با اندازه خروجی سیستم ما که در شکل ۴-۱۳ نشان داده شده است، مطابقت دارد.

1x1 struct with 19 fields

Field	Value
st	[1,0,-1,-1,-1;0,1,0,1,-1]
irrev_react	[1,1,1,0,1]
all_int	1
kn	*5x3 double*
crel	*[ ]*
sub	*5x5 double*
irrev_rd	[1,1,1,0,1]
blocked_react	*[ ]*
sub_irr_viol	*[ ]*
rd	[1,0,-1,-1,-1;0,1,0,1,-1]
rd_met	[1;2]
req_reacts	*[ ]*
rd_ems	*5x5 double*
rd_cb	[NaN;NaN;NaN;NaN;...
err	0
wrd	[0,-1,-1,1,-1;1,0,-1,0,1]
irrev_wrd	[1,1,1,1,0]
wkr	*5x3 double*
irrev_ems	*1x5 logical*

شکل ۴-۱۹ خروجی ابزار متاتول برای شبکه نمونه

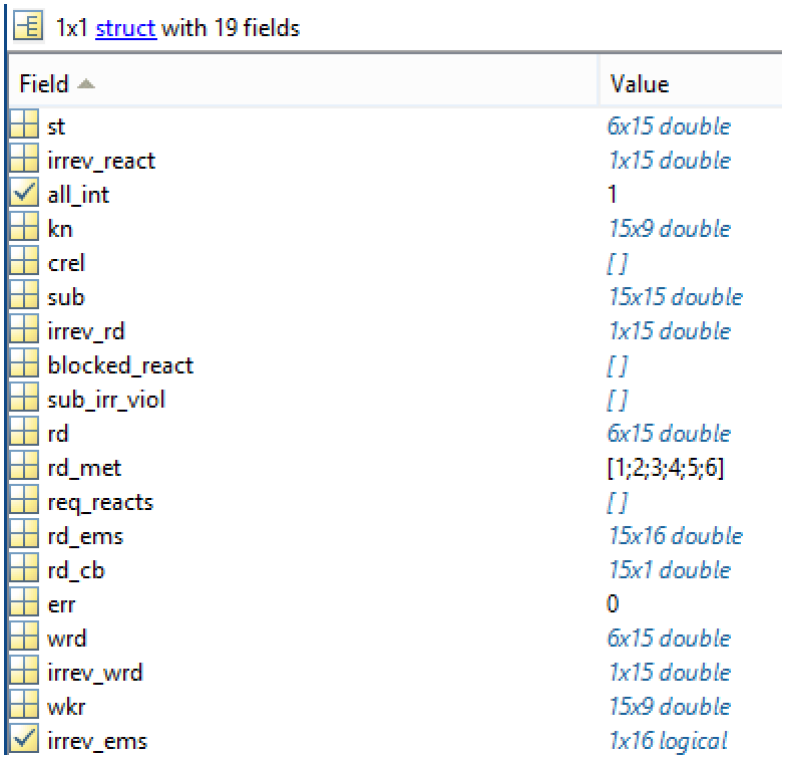
محتوای ماتریس rd_ems در شکل ۴-۲۰ نشان داده شده است. همانطور که مشاهده می‌شود، خود یا ضریبی از بردارهای شکل ۴-۱۳ در این ماتریس وجود دارد.

net.rd_ems

	1	2	3	4	5
1	0	0	2	1	2
2	1	1	0	0	2
3	1	0	0	1	0
4	0	0.5000	1	0	2
5	-1	-0.5000	1	0	0

شکل ۴-۲۰ حالت‌های پایه‌ای محاسبه شده با متاتول برای شبکه نمونه

خروجی سیستم برای شبکه tricarboxylic-acid-cycle نیز یک ماتریس با تعداد ستون ۱۶ تولید می‌کند که هر کدام ۱۵ عنصر دارد. پس تعداد حالت‌های پایه‌ای محاسبه شده برای این شبکه ۱۶ می‌باشد و این تعداد را می‌توان در اندازه ماتریس rd_ems در شکل ۴-۱۴ مشاهده کرد.



شکل ۴-۲۱ خروجی ابزار متاتول برای شبکه tricarboxylic-acid-cycle

محتوای ماتریس rd_ems در شکل ۴-۲۲ نشان داده شده است. همانطور که مشاهده می‌شود، خود یا ضریبی از بردارهای شکل ۴-۱۴ در این ماتریس وجود دارد. به طوریکه ستون‌های متناظر از شکل‌های ۴-۱۴ و ۴-۲۲ به ترتیب زیر هستند:

- ستون ۱ از خروجی سیستم متناظر با ستون ۲ از خروجی متاتول
- ستون ۲ از خروجی سیستم متناظر با ستون ۱۴ از خروجی متاتول
- ستون ۳ از خروجی سیستم متناظر با ستون ۱۵ از خروجی متاتول
- ستون ۴ از خروجی سیستم متناظر با ستون ۸ از خروجی متاتول
- ستون ۵ از خروجی سیستم متناظر با ستون ۱۱ از خروجی متاتول
- ستون ۶ از خروجی سیستم متناظر با ستون ۹ از خروجی متاتول
- ستون ۷ از خروجی سیستم متناظر با ستون ۱۶ از خروجی متاتول
- ستون ۸ از خروجی سیستم متناظر با ستون ۱۰ از خروجی متاتول
- ستون ۹ از خروجی سیستم متناظر با ستون ۶ از خروجی متاتول
- ستون ۱۰ از خروجی سیستم متناظر با ستون ۵ از خروجی متاتول

- ستون ۱۱ از خروجی سیستم متناظر با ستون ۴ از خروجی متاتول
- ستون ۱۲ از خروجی سیستم متناظر با ستون ۳ از خروجی متاتول
- ستون ۱۳ از خروجی سیستم متناظر با ستون ۷ از خروجی متاتول
- ستون ۱۴ از خروجی سیستم متناظر با ستون ۱۲ از خروجی متاتول
- ستون ۱۵ از خروجی سیستم متناظر با ستون ۱۳ از خروجی متاتول
- ستون ۱۶ از خروجی سیستم متناظر با ستون ۱ از خروجی متاتول

net.rd_ems

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0
2	0	0	1	1	1	1	1	0	0	0	0	0	1	0	0	0
3	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
5	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
6	1	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
8	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
9	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
10	1	0	0	0	0	1	1	0	0	0	1	1	2	0	0	0
11	-1	1	2	2	2	2	1	0	0	0	0	0	0	1	0	0
12	-1	1	1	1	1	1	0	0	0	0	0	0	-1	1	0	0
13	-1	1	0	0	0	0	-1	0	0	0	0	-1	-2	0	0	0
14	1	1	3	1	2	3	2	1	1	2	2	1	3	0	0	0
15	-1	1	1	1	1	1	0	0	0	0	0	-1	-1	0	0	0

شکل ۴-۲۲ حالت‌های پایه‌ای محاسبه شده با متاتول برای شبکه tricarboxylic-acid-cycle

۵

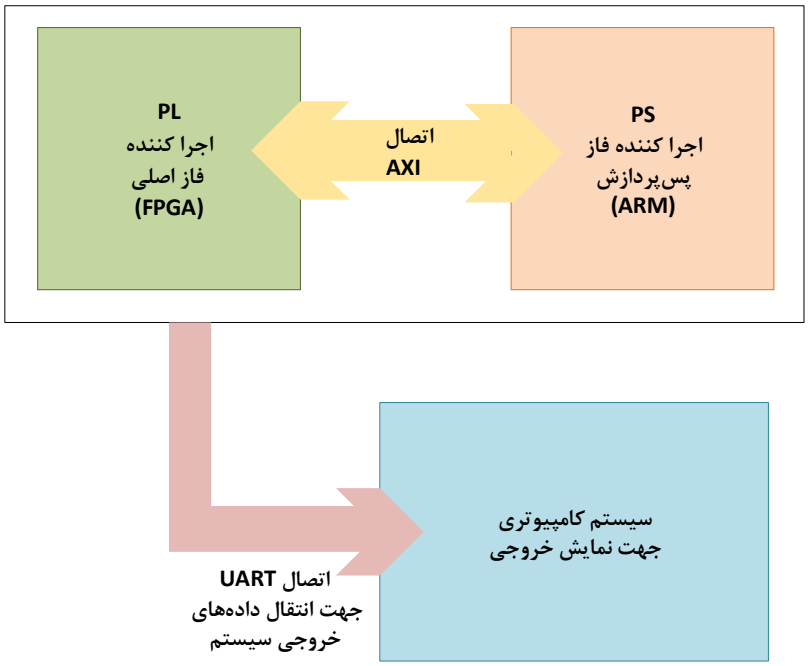
فصل پنجم

جمع‌بندی و کارهای آینده

۱.۵ جمع‌بندی

هدف از انجام این پروژه، طراحی و پیاده‌سازی یک سیستم ترکیبی سخت‌افزار و نرم‌افزار بود که به واسطه آن روش دودویی برای محاسبه حالت‌های پایه‌ای شبکه‌های متابولیکی انجام گرفت. الگوریتم پیاده‌سازی شده دارای سه بخش پیش‌پردازش، بخش اصلی و پس‌پردازش می‌شود که در سیستم ما بخش اصلی که شامل پیمایش‌های متعدد است به صورت سخت‌افزاری و با استفاده از زبان VHDL پیاده‌سازی شد. پیاده‌سازی دو قسمت پیش‌پردازش و پس‌پردازش به صورت نرم‌افزاری و با استفاده از زبان C انجام گرفت و با استفاده از پردازنده آرم موجود بر روی زدبورد اجرا شدند. از طریق واسط AXI4 ارتباط میان پردازنده نرم‌افزار و واحد منطق برنامه‌پذیر برقرار شد و داده‌های مورد نظر به صورت حافظه‌محور بین بخش سخت‌افزار و نرم‌افزار منتقل شدند. پس از انجام عملیات لازم توسط پردازنده، خروجی نهایی به واسطه رابط سریال به کامپیوتر منتقل و نمایش داده شد.

در شکل ۵-۱ شمای کلی سیستم پیاده‌سازی شده که حالت‌های پایه‌ای را با استفاده از روش دودویی محاسبه می‌کند، نشان داده شده است.



شکل ۵-۱ معماری کلی سیستم پیاده‌سازی شده

۲.۵ کارهای آینده

از جمله کارهایی که درجهت بهبود و کاربردی شدن هرچه بیشتر این پروژه می‌توان به آنها پرداخت می‌توان به موارد زیر اشاره کرد:

- افزایش اندازه شبکه‌های مورد آزمون
- محاسبه حالت‌های پایه‌ای برای شبکه فشرده نشده
- بررسی دقیق میزان بهبود سرعت محاسبات در پیاده‌سازی توامان سخت‌افزار و نرم‌افزار

اصلی‌ترین مزیت این پروژه پیاده‌سازی آن به صورت طراحی توامان سخت‌افزاری و نرم‌افزاری است. از طرفی بزرگترین چالش این پروژه نیز در همین بخش قرار می‌گیرد. در الگوریتم دودویی که طی این پروژه پیاده‌سازی شد، محاسباتی نیاز است که انجام آنها به صورت نرم‌افزاری و با استفاده از کتابخانه‌های موجود، از جهت کیفیت پیاده‌سازی به صرفه‌تر است. از این رو در این بخش دو پروژه را معرفی می‌کنیم که طراحی توامان را ساده‌تر می‌کنند.

۱.۲.۵ استفاده از پروژه PYNQ

PYNQ یک پروژه متن‌باز زایلینکس می‌باشد که کار طراحی سیستم‌های نهفته را با استفاده از سیستم زینک زایلینکس ساده می‌کند. در این پروژه، طراحان با استفاده از زبان پایتون و کتابخانه‌های آن طراحان می‌توانند از مزایای منطق برنامه‌پذیر و ریزپردازنده در زینک استفاده کنند و بدین ترتیب سیستم‌های نهفته با توانایی بالاتری را تولید کنند. از جمله امکاناتی که طی استفاده از این پروژه به سیستم‌های طراحی شده اضافه می‌گردد به موارد زیر می‌توان اشاره کرد[۱۷]:

- اجرای موازی سخت‌افزاری
- الگوریتم‌های تسریع شده توسط سخت‌افزار
- پردازش سیگنال بی‌درنگ
- پهنای باند بالا در IOها

بورد PYNQ-Z1 اولین بوردی است که PYNQ را پشتیبانی می‌کند که می‌توان کد پایتون را به صورت مستقیم روی آن اجرا کرد و طراحی توامان را به سادگی انجام داد. با استفاده از زبان پایتون، توسعه‌دهندگان می‌توانند از کتابخانه‌های سخت‌افزاری بر روی منطق برنامه‌پذیر استفاده کنند[۱۷].

به طور کلی این پروژه PYNQ، برای گروه‌های مختلفی از طراحان و توسعه‌دهندگان قابل استفاده خواهد بود: توسعه‌دهندگان نرم‌افزاری که می‌خواهند از قابلیت‌های زینک و سخت‌افزار برنامه‌پذیر استفاده کنند بدون آنکه از ابزارهای طراحی سخت‌افزار استفاده کنند؛ طراحان سخت‌افزاری که می‌خواهند طراحی آنها پاسخگوی نیازهای عده زیادی از افراد باشد؛ و طراحانی که می‌خواهند یک واسط نرم‌افزاری خوب برای طراحی زینک ارائه دهند. این کتابخانه‌ها سرعت اجرای نرم‌افزار بر روی بورد PYNQ-Z1 را بالا می‌برند و امکان سفارشی‌سازی واسط‌ها و سکوهای سخت‌افزاری را در اختیار کاربران قرار می‌دهند. علاوه بر زبان پایتون، امکان برنامه‌نویسی با زبان‌های C و C++ و استفاده از محیط توسعه نرم‌افزار زایلینکس برای این بورد وجود دارد. از مشخصات فنی این بورد موارد زیر را می‌توان نام برد[۱۷]:

- پردازنده: دوهسته‌ای آرم کورتکس ای ۹
- FPGA: ۱/۳ میلیون گیت با قابلیت پیکربندی مجدد
- حافظه: ۵۱۲ مگابایت DDR3/Flash
- ابعاد: ۸۷ میلیمتر x ۱۲۲ میلیمتر

۲.۲.۵ استفاده از پروژه Reconfigure

یکی دیگر از پروژه‌ها در این زمینه، پروژه Reconfigure می‌باشد که به ما اجازه می‌دهد تمام کد سخت‌افزاری و نرم‌افزاری خود را به زبان Go تهیه کرده و سیستم خود را به صورت یکجا برنامه‌ریزی کنیم. این زبان توسط شرکت Google ارائه شده است و کارآیی بالایی در برنامه‌نویسی همروند دارد. مهم‌ترین هدفی که این پروژه دنبال می‌کند ارتقای سرعت پردازش در اجرای موازی با استفاده از FPGA است. اگرچه با استفاده از پردازنده‌های سریع‌تر و پردازش چندهسته‌ای ارتقای کارایی سیستم‌ها صورت گرفته است، می‌دانیم که در زمینه‌هایی چون پردازش مه‌داده‌ها در هوش مصنوعی، اقتصاد، اینترنت اشیا و رسانه به امکاناتی فراتر از آنچه پردازنده‌های معمول در اختیار ما می‌گذارند نیاز داریم و در چنین شرایطی FPGAها می‌توانند پاسخگوی نیازهای ما باشند. سرعت در FPGAها بسته به نوع الگوریتم می‌تواند در بازه ده تا صدبرابر نسبت به پردازنده‌های معمول بیشتر باشد. با آنکه با فرکانسی پایین‌تر نسبت به پردازنده‌های امروزی کار می‌کنند ولی این توانایی را دارند تا بسیاری از کارها در زمانی بسیار کمتر انجام دهند. همچنین اگر بتوانیم در سطح گسترده از FPGAها استفاده کنیم هزینه‌ها بطور چشمگیری کاهش می‌یابند زیرا برای مثال یک FPGA می‌تواند همزمان کارهای چندین سرویس‌دهنده

را به انجام برساند. امکان موازی‌سازی و بازبرنامه‌پذیری در هنگام نیاز نیز از دیگر قابلیت‌های FPGAهاست. ولی به دلیل نیاز به مهارت‌ها و ابزار خاص، استفاده از FPGA تا حدی محدود شده است. Reconfigure این امکان را به ما می‌دهد تا بتوانیم نیازهای خود را با همان ابزار و مهارت‌های توسعه نرم‌افزاری بر روی FPGAها برنامه‌ریزی کنیم. علت آنکه در Reconfigure از زبان Go استفاده شده است امکاناتی در این زبان است که عبارتند از توابعی که به صورت همروند قابل اجرا با دیگر توابع هستند، کانال‌هایی که این توابع با استفاده از آنها امکان همگام‌سازی و ارتباط با یکدیگر را پیدا می‌کنند و امکان مدیریت اجرای موازی توابع. همانطور که پیش‌تر نیز توضیح داده شد، برای اتصال FPGA به یک حافظه اشتراکی از واسط AXI استفاده می‌شود. این واسط پردازنده‌های چندهسته‌ای را نیز پشتیبانی می‌کند. با این حال سطح موازی‌سازی ممکن است فراتر از حدی باشد که مدیریت AXI به راحتی قابل انجام باشد. از این رو در Reconfigure یک پروتکل جدید با نام SMI برای این امر ایجاد شده است[۱۸].

۶ منابع و مراجع

[1] Sciencedirect.com. (2018). *Metabolic network - an overview | ScienceDirect Topics*. [online] Available at: https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/metabolic-network. [Accessed 20 Jul. 2018].

[2] Terzer, Marco, and Jörg Stelling. "Elementary flux modes–state-of-the-art implementation and scope of application." *BMC Systems Biology* 1.1 (2007): Pl.

[3] Gagneur, Julien, and Steffen Klamt. "Computation of elementary modes: a unifying framework and the new binary approach." *BMC bioinformatics* 5.1 (2004).

[4] Wagner, C. "Nullspace approach to determine the elementary modes of chemical reaction systems." *The Journal of Physical Chemistry B* 108.7 (2004): 2425-2431.

[5] En.wikipedia.org. (2018). *Xilinx Vivado*. [online] Available at: https://en.wikIPedia.org/wiki/Xilinx_Vivado. [Accessed 20 Jul. 2018].

[6] Xilinx.com. (2015). *Xilinx Software Development Kit (SDK) User Guide*. [online] Available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/SDK_Doc/index.html. [Accessed 20 Jul. 2018].

[7] Reference.digilentinc.com. (2018).*ZedBoard [Reference.Digilentinc]*. [online] Available at: https://reference.digilentinc.com/reference/programmable-logic/ZedBoard/start. [Accessed 20 Jul. 2018].

[8] Tutorial by Cytron. (2012). *UART - Universal Asynchronous Receiver and Transmitter*. [online] Available at: https://tutorial.cytron.io/2012/02/16/uart-universal-asynchronous-receiver-and-transmitter. [Accessed 20 Jul. 2018].

[9] Xilinx.com. (2017). *Vivado AXI Reference Guide*. [online] Available at:

https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf. [Accessed 20 Jul. 2018].

[10] Griffin, Rich. "Designing a custom AXI-lite slave perIPheral." (2014).

[11] Gregthatcher.com. (2013). *Null Space Calculator*. [online] Available at: http://www.gregthatcher.com/Mathematics/NullSpaceCalculator.aspx. [Accessed 20 Jul. 2018].

[12] Xilinx.com. (2013). *Vivado Synthesis*. [online] Available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_2/ug901-vivado-synthesis.pdf. [Accessed 20 Jul. 2018].

[13] Weiss, D. (2008). *FSU Jena | Lehrstuhl fr Bioinformatik*. [online] Pinguin.biologie.uni-jena.de. Available at: http://pinguin.biologie.uni-jena.de/bioinformatik/networks/metatool/metatool5.1/metatool5.1.html. [Accessed 20 Jul. 2018].

[14] Www2.mpi-magdeburg.mpg.de. (2018). *CellNetAnalyzer Manual.* [online] Available at: https://www2.mpi-magdeburg.mpg.de/projects/cna/manual_cellnetanalyzer.pdf. [Accessed 20 Jul. 2018].

[15] Csb.ethz.ch. (2018). *Documentation*. [online] Available at: http://www.csb.ethz.ch/tools/software/efmtool/documentation.html. [Accessed 20 Jul. 2018].

[16] Schuster, Stefan, Thomas Dandekar, and David A. Fell. "Detection of elementary flux modes in biochemical networks: a promising tool for pathway analysis and metabolic engineering." *Trends in Biotechnology* 17.2 (1999): 53-60.

[17] Pynq.readthedocs.io. (2017). *Getting Started — Python productivity for Zynq (Pynq) v1.0*. [online] Available at: http://pynq.readthedocs.io/en/latest/getting_started.html. [Accessed 20 Jul. 2018].

[18] Docs.reconfigure.io. (2017). *How does it Work? — Reconfigure.io 0.17.5 documentation*. [online] Available at: http://docs.reconfigure.io/overview.html#go-compilation-stages. [Accessed 20 Jul. 2018].

پیوست

پ۱ توصیف شبکه تست tricarboxylic-acid-cycle–glyoxylate-shunt

-ENZREV
Eno Acn SucCD Sdh Fum Mdh AspC Gdh IlvEAvtA

-ENZIRREV
Pyk AceEF GltA Icd SucAB Icl Mas AspCon AspA Pck Ppc Pps GluCon
AlaCon SucCoACon

-METINT
Ala Asp Glu Gly Mal Fum Succ SucCoA OG IsoCit Cit OAA AcCoA CoA
Pyr PEP

-METEXT
Sucex Alaex Gluex ADP ATP AMP NH3 Aspex FADH2 FAD NADPH NADP NADH CO2 NAD
PG

-CAT
Eno : PG = PEP .
Pyk : PEP + ADP = Pyr + ATP .
AceEF : Pyr + NAD + CoA = AcCoA + CO2 + NADH .
GltA : OAA + AcCoA = Cit + CoA .
Acn : Cit = IsoCit .
Icd : IsoCit + NADP = OG + CO2 + NADPH .
SucAB : OG + NAD + CoA = SucCoA + CO2 + NADH .
SucCD : SucCoA + ADP = Succ + ATP + CoA .
Sdh : Succ + FAD = Fum + FADH2 .
Fum : Fum = Mal .
Mdh : Mal + NAD = OAA + NADH .
Icl : IsoCit = Succ + Gly .
Mas : Gly + AcCoA = Mal + CoA .
AspC : OAA + Glu = Asp + OG .
AspCon : Asp = Aspex .
AspA : Asp = Fum + NH3 .
Gdh : OG + NH3 + NADPH = Glu + NADP .
Pck : OAA + ATP = PEP + ADP + CO2 .
Ppc : PEP + CO2 = OAA .
Pps : Pyr + ATP = PEP + AMP .
GluCon : Glu = Gluex .
IlvEAvtA : Pyr + Glu = Ala + OG .
AlaCon : Ala = Alaex .
SucCoACon : SucCoA = Sucex + CoA .

پ۲ کد پیاده‌سازی سخت‌افزار

در این بخش کدهای مربوط به توصیف ماشین حالتی که بخش اصلی الگوریتم دودویی را پیاده‌سازی می‌کند آورده شده است.

```
library work;
use work.my_package.all;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.fixed_pkg.all;


entity main is
   generic(m : in integer range 0 to 15 := 6;   --metabolites
           q : in integer range 0 to 25 := 15;  --reactions not splitted
           qsplit : in integer range 0 to 30 := 20 ;
           --reactions splitted = q + revs
           R_rows : in integer range 0 to 30 := 20;      --qsplit
           R_columns : in integer range 0 to 16 := 14;   --qsplit-m
           R1_rows : in integer range 0 to 16 := 14;     --qsplit-m
           R2_rows : in integer range 0 to 15 := 6;      --m
           max_column : in integer range 0 to 3000 := 100);
           --max of columns in R after adding combination
   port(clock, reset : in std_logic;
      R1_data : in std_logic_vector(1 to R1_rows*R_columns);
      R2_data_postfix : in std_vec_array(1 to R2_rows*R_columns);
      SW_call : out std_logic := '0';
      EM_columns : out integer range 0 to 3000 := 0;
      EM_rows : out integer range 0 to R_rows := 0;
      EM_data: out std_logic_vector(1 to 480) := (others => '0')
   );
end entity;
architecture arch of main is

--fixed point equal for zero
signal zero : sfixed(10 downto -15);

--fixed point signal of R2 data in R2_data_postfix
signal R2_data: fixedp_array(1 to R2_rows*R_columns);

--all states
type state_type is (S0, S0a, S0b, S1, S1a, S1b, S2, S3, S3a, S3aa, S3b, S4, S5, S5a, S5b, S6, S6a,
S6b, S7, S7a, S8, S_f);
```

```
signal state : state_type := S0;                --initial state

signal state_num : integer range 0 to 23 := 0;
 --to monitor sequential value of states

--to force keeping the signal in optimization
attribute keep : string;
attribute keep of state_num : signal is "true";
--attribute keep of null_check : signal is "true";

--customizing fsm encoding
attribute fsm_encoding : string;
attribute fsm_encoding of state : signal is "sequential";

begin
   zero <=  to_sfixed(0.0, zero);

   --initialize R2_data with fixed_point values read from R2_data_postfix
   for_label: for i in 1 to (R2_rows*R_columns) generate
      R2_data(i) <= to_sfixed(R2_data_postfix(i), 10 , -15);
   end generate for_label;

   process(clock, zero, state_num)

   --counters for different loops
   variable l1_counter : integer range 1 to 200 := 1;  --boundary: R1_rows
   variable l2_counter : integer range 1 to 200 := 1;  --boundary: R2_rows
   variable l3_counter : integer range 1 to max_column := 1;
   --boundary: valid_column
   variable l4_counter : integer range 1 to 200 := 1;
   --boundary: R1_valid_row
   variable l5_counter : integer range 1 to 200 := 1;
   --boundary: R2_rows
   variable l6_counter : integer range 1 to max_column := 1;
   --boundary: jneg_size
   variable l7_counter : integer range 1 to max_column := 1;
   --boundary: valid_column
   variable l8_counter : integer range 1 to max_column := 1;
   --boundary: valid_column

   --counter for additional loops that implements matrix iteration
   variable la_counter : integer range 1 to 200 := 1;
   --boundary: R_columns
   variable lb_counter : integer range 0 to 200 := 1;
   --boundary: R1_data size
```

```
variable lc_counter : integer range 1 to 200 := 1;
--boundary: R_columns
variable ld_counter : integer range 1 to 200 := 1;
--boundary: R2_data size
variable lf_counter : integer range 1 to max_column := 1;
--boundary: valid_column
variable lg_counter : integer range 1 to 200 := 1;
--boundary: R1_valid_row
variable lh_counter : integer range 1 to 200 := 1;
--boundary: R1_valid_row
variable li_counter : integer range 1 to 200 := 1;
--boundary: R1_valid_row
variable lj_counter : integer range 1 to 200 := 1;
--boundary: R1_valid_row
variable lk_counter : integer range 1 to 200 := 1;
--boundary: R2_rows
variable ll_counter : integer range 1 to 200 := 1;
--boundary: R1_valid_row
variable lm_counter : integer range 1 to 700 := 1;
--boundary: EM_data size

--variables the same as pseudo code
variable new_numr : integer range 0 to 200 := qsplit - m;
variable numr : integer range 0 to 200 := qsplit - m;
variable p : integer range 0 to 200 := q-m;
--main loop counter, bounday: m
variable k : integer range 0 to max_column := 0;
--jneg loop counter, bounday: jneg_size
variable l : integer range 0 to max_column := 0;
--jpos loop counter, bounday: jpos_size
variable r : integer range 0 to max_column:= 0;
--test loop counter, bounday: valid_columns
variable adj : std_logic := '0';
--test result
variable nullbits : integer range 0 to 200 := 0;
--number of zeros in newr
variable newr : std_logic_vector(1 to R_rows) := (others => '0');
--new column to be added
variable testr : std_logic_vector(1 to R_rows):= (others => '0');
--test column

--variables to store R1 and R2 with max size
variable R1_matrix : bit_matrix(1 to R_rows, max_column downto 1) :=
(others =>(others => '0'));
variable R2_matrix : fixedp_matrix(1 to R2_rows, max_column downto 1)
:= (others =>(others => zero));
```

```
--jneg and jpos row_vectors and their size
variable jpos : int_array (1 to max_column) := (others => 0);
--row_vector of indices in R2 row with positive value
variable jneg : int_array (1 to max_column) := (others => 0);
--row_vector of indices in R2 row with negative value

variable jneg_size : integer range 0 to 200 := 0;    --size of jneg
variable jpos_size : integer range 0 to 200 := 0;    --size of jpos

--S1 variables
variable wanted_row : integer range 0 to R2_rows := 0;
--the row number of R2 which is being changed
variable wanted_R2 : fixedp_array(1 to max_column) :=
(others => zero); --the row_vector of R2 which is being changed

--last valid state of the result matrix
variable valid_column : integer range 0 to max_column :=
R_columns;    --total number of valid columns in R2 and R1
variable R1_valid_row : integer range 0 to R_rows := R1_rows;
--total number of valid rows in R1

variable mul1, mul2 : sfixed(21 downto -30);
--result of subtraction and multiply in R2 new values, not necessary
variable newR2element : sfixed(22 downto -30);

variable moving_col : integer range 0 to max_column := 0;
variable target_col : integer range 0 to max_column := 0;
variable or_col1, or_col2 : integer range 0 to max_column := 0;

begin
   if rising_edge(clock) then
      if (reset = '1') then
         state <= S0;
      else
         case state is
            when S0 =>
               state_num <= 0;
               -- initializing some variables
               p := q-m;
               k := 0;
               l := 0;
               r := 0;
               l1_counter := 1;
               l2_counter := 1;
               l3_counter := 1;
```

```
            l4_counter := 1;
            l5_counter := 1;
            l6_counter := 1;
            l7_counter := 1;
            l8_counter := 1;
            la_counter := 1;
            lb_counter := 1;
            lc_counter := 1;
            ld_counter := 1;
            lf_counter := 1;
            li_counter := 1;
            lj_counter := 1;
            lk_counter := 1;
            ll_counter := 1;
            lm_counter := 1;
            wanted_row := 0;
            valid_column := R_columns;
            R1_valid_row := R1_rows;
            new_numr := qsplit - m;
            numr := qsplit - m;
            jneg_size := 0;
            jpos_size := 0;
            state <= S0a;

        when S0a =>
            state_num <= 1;
            -- turn the row vector of R1 into a 2D matrix
            if(l1_counter < R1_rows or l1_counter = R1_rows)
            then
                R1_matrix(l1_counter, la_counter) :=
                R1_data(lb_counter);
                la_counter := la_counter + 1;
                lb_counter := lb_counter + 1;
                if(la_counter > R_columns) then
                    la_counter := 1;
                    l1_counter := l1_counter + 1;
                    state <= S0a;
                else
                    state <= S0a;
                end if;
            else
                l2_counter := 1;
                lc_counter := 1;
                ld_counter := 1;
                state <= S0b;
            end if;
```

```
            when S0b =>
               state_num <= 2;
               -- turn the row vector of R2 into a 2D matrix
               if(l2_counter < R2_rows or l2_counter = R2_rows)
               then
                  R2_matrix(l2_counter, lc_counter) :=
                  R2_data(ld_counter);
                  lc_counter := lc_counter + 1;
                  ld_counter := ld_counter + 1;
                  if(lc_counter > R_columns) then
                     lc_counter := 1;
                     l2_counter := l2_counter + 1;
                     state <= S0b;
                  else
                     state <= S0b;
                  end if;
               else
                  p := q-m;
                  state <= S1;
               end if;

            when S1 =>
               state_num <= 3;
               --main loop
               p := p+1;
               if((p < q) or (p = q)) then
      -- determine the row of R2 which is being changed
                  wanted_row := wanted_row + 1;
                  lf_counter := 1;
                  state <= S1a;
                  k := 0;
               else
                  l8_counter := 1;
                  ll_counter := 1;
                  lm_counter := 1;
                  state <= S8;
               end if;

            when S1a =>
               state_num <= 4;
   --copy values of wanted row from R2 into wanted_R2 to find neg and  pos values
               wanted_R2(lf_counter) := R2_matrix(wanted_row,
               lf_counter);
               lf_counter := lf_counter + 1;
               if(lf_counter > valid_column) then
```

```
            -- initiazlization for S1b
            jneg_size := 0;
            jpos_size := 0;
            jneg := (others => 0);
            jpos := (others => 0);
            l3_counter := 1;
            state <= S1b;
        else
            state <= S1a;
        end if;

    when S1b =>
        state_num <= 5;
        --find_neg and find_pos implementation
        if((l3_counter < valid_column) or
        (l3_counter = valid_column))  then
            if(wanted_R2(l3_counter) > zero) then
                jpos_size := jpos_size + 1;
                jpos(jpos_size) := l3_counter;
            elsif(wanted_R2(l3_counter) < zero) then
                jneg_size := jneg_size + 1;
                jneg(jneg_size) := l3_counter;
            end if;
            l3_counter := l3_counter + 1;
            state <= S1b;
        else
            state <= S2;
        end if;

    when S2 =>
        state_num <= 6;
        --go to the inner loop for jpos
        k := k+1;
        if((k < jneg_size) or (k = jneg_size)) then
            lg_counter := R1_valid_row + 1;
            l := 0;
            state <= S3;
        else
            if(jneg_size > 0) then
                lj_counter := 1;
                l6_counter := 1;
                state <= S6;
            else
                state <= S7;
            end if;
        end if;
```

```
            when S3 =>
               state_num <= 7;
 --bitwise or of two columns of R1; one negative and one positive
               if(lg_counter > R1_valid_row) then
                  l := l+1;
                  lg_counter := 1;
                  state <= S3;
               elsif((l < jpos_size) or (l = jpos_size)) then
                  or_col1 := jneg(k);
                  or_col2 := jpos(l);
                  newr(lg_counter) :=
                  R1_matrix(lg_counter,jneg(k)) or
                  R1_matrix(lg_counter,jpos(l));
                  lg_counter := lg_counter + 1;
                  if(lg_counter > R1_valid_row) then
                     l4_counter := 1;
                     nullbits := 0;
                     state <= S3a;
                  else
                     state <= S3;
                  end if;
               else
                  state <= s2;
               end if;

            when S3a =>
               state_num <= 8;
--count the number of null bits in newr (the result of bitwise or in S3)
               if((l4_counter < R1_valid_row) or (l4_counter =
               R1_valid_row)) then
                  if(newr(l4_counter) = '0') then
                     nullbits := nullbits + 1;
                  end if;
                  l4_counter := l4_counter + 1;
                  state <= S3a;
               else
                  state <= S3aa;
               end if;

            when S3aa =>
               state_num <= 9;
               --null_check <= nullbits;
               --check the minimum number of zeros
               if(nullbits+1 < qsplit-m-1) then
                  lg_counter := R1_valid_row + 1;
```

```
                state <= S3;
            else
                state <= S3b;
            end if;

        when S3b =>
            state_num <= 10;
            --initialization for adjacency test
            adj := '1';
            r := 0;
            lh_counter := R1_valid_row + 1;
            state <= S4;

        when S4 =>
            state_num <= 11;
--adjacency test: r+ or r- != r+ or r- or (other R columns)
            if(lh_counter > R1_valid_row) then
                r := r+1;
                lh_counter := 1;
                state <= S4;
            else
                if((r < numr or r = numr) and (adj = '1')) then
                    testr(lh_counter) := newr(lh_counter) or
                    R1_matrix(lh_counter, r);
                    lh_counter := lh_counter + 1;
                    if(lh_counter > R1_valid_row) then
                        if ((r /= jpos(l)) and (r /= jneg(k))
                        and (testr = newr)) then
                            adj := '0';
                        else
                            adj := '1';
                        end if;
                        state <= S4;
                    else
                        state <= S4;
                    end if;
                else
                    state <= S5;
                end if;
            end if;

        when S5 =>
            state_num <= 12;
            --initializtion for combination loop in S5a
            if(adj = '1') then
                new_numr := new_numr + 1;
```

```
            valid_column := valid_column + 1;
            li_counter := 1;
            state <= S5a;
        else
            lg_counter := R1_valid_row + 1;
            state <= S3;
        end if;

    when S5a =>
        state_num <= 13;
        --combine and add the result of combination to R1
        R1_matrix(li_counter, new_numr) :=
        newr(li_counter);
        li_counter := li_counter + 1;
        if(li_counter > R1_valid_row) then
            l5_counter := wanted_row;
            state <= S5b;
        else
            state <= S5a;
        end if;

    when S5b =>
        state_num <= 14;
        --combine and add the result of combination to R2
        if((l5_counter < R2_rows) or (l5_counter =
            R2_rows)) then
            mul1 := R2_matrix(wanted_row,
            jpos(l))*R2_matrix(l5_counter, jneg(k));
            mul2 := R2_matrix(wanted_row,
            jneg(k))*R2_matrix(l5_counter, jpos(l));
            newR2element := mul1 - mul2;
            if((Is_Negative(newR2element) = true) and
              (newR2element(10) = '0')) then
                R2_matrix(l5_counter, new_numr) := '1' &
                newR2element(9 downto -15);
            else
                R2_matrix(l5_counter, new_numr) :=
                newR2element(10 downto -15);
            end if;
            l5_counter := l5_counter + 1;
            state <= S5b;
        else
            lg_counter := R1_valid_row + 1;
            state <= S3;
        end if;
```

```
            when S6 =>
               state_num <= 15;
--copy the last columns of R1 in the place of columns with negative rows
               if((l6_counter < jneg_size) or
                 (l6_counter = jneg_size)) then
                  moving_col := valid_column - l6_counter + 1;
                  target_col := jneg(l6_counter);
                  R1_matrix(lj_counter, target_col) :=
                  R1_matrix(lj_counter, moving_col);
                  R1_matrix(lj_counter, moving_col) := 'U';
                  lj_counter := lj_counter + 1;
                  if(lj_counter > R1_valid_row) then
                     lk_counter := 1;
                     state <= S6a;
                  else
                     state <= S6;
                  end if;
               else
                  state <= S7;
               end if;
            when S6a =>
               state_num <= 16;
 --copy the last columns of R2 in the place of columns with negative rows
               R2_matrix(lk_counter, jneg(l6_counter)) :=
               R2_matrix(lk_counter,
                           valid_column - l6_counter + 1);
                     R2_matrix(lk_counter,
                 valid_column - l6_counter + 1) := zero;
               lk_counter := lk_counter + 1;
               if(lk_counter > R2_rows) then
                  state <= S6b;
               else
                  state <= S6a;
               end if;

            when S6b =>
               state_num <= 17;
         --some reset initiation for copy loops(deletion of neg rays)
               lj_counter := 1;
               l6_counter := l6_counter + 1;
               state <= S6;

            when S7 =>
               state_num <= 18;
               --edit required after delete process
               numr := new_numr - jneg_size;
```

```
            new_numr := new_numr - jneg_size;
            valid_column := valid_column - jneg_size;
        --initialization to copy the last editted row of R2 into R1;
            R1_valid_row := R1_valid_row + 1;
            l7_counter := 1;
            state <= S7a;

        when S7a =>
            state_num <= 19;
            --copy the new binary row of R2 into R1;
            if((l7_counter < valid_column) or
               (l7_counter = valid_column)) then
               if(R2_matrix(wanted_row, l7_counter) >
                  zero)then
                  R1_matrix(R1_valid_row, l7_counter) := '1';
               elsif(R2_matrix(wanted_row, l7_counter) = zero)
                                then
                  R1_matrix(R1_valid_row, l7_counter) := '0';
               end if;
               l7_counter := l7_counter + 1;
               state <= S7a;
            else
               state <= S1;
            end if;

        when S8 =>
            state_num <= 20;
            --turn the 2D R2_matrix into a vector
            if((l8_counter < valid_column)
              or (l8_counter = valid_column)) then
               EM_data(lm_counter) <=
               R1_matrix(ll_counter, l8_counter);
               lm_counter := lm_counter + 1;
               ll_counter := ll_counter + 1;
               if(ll_counter > R1_valid_row) then
                  l8_counter := l8_counter + 1;
                  ll_counter := 1;
                  state <= S8;
               else
                  state <= S8;
               end if;
            else
               state <= S_f;
            end if;

        when S_f =>
```

```
                state_num <= 21;
                --set the value of output signals and call software
                EM_columns <= valid_column;
                EM_rows <= R1_valid_row;
                SW_call <= '1';

            when others =>
                state <= S0;
            end case;
        end if;
    end if;
  end process;
end architecture;
```

Amirkabir University of Technology
(Tehran Polytechnic)

Computer Engineering and Information Technology Department

B.Sc. Thesis

Title

Computation of Elementary Modes in Metabolic Networks Using Binary Approach

By
Mahshid Alinoori

Supervisor
Dr. Morteza Saheb Zamani

July 2018