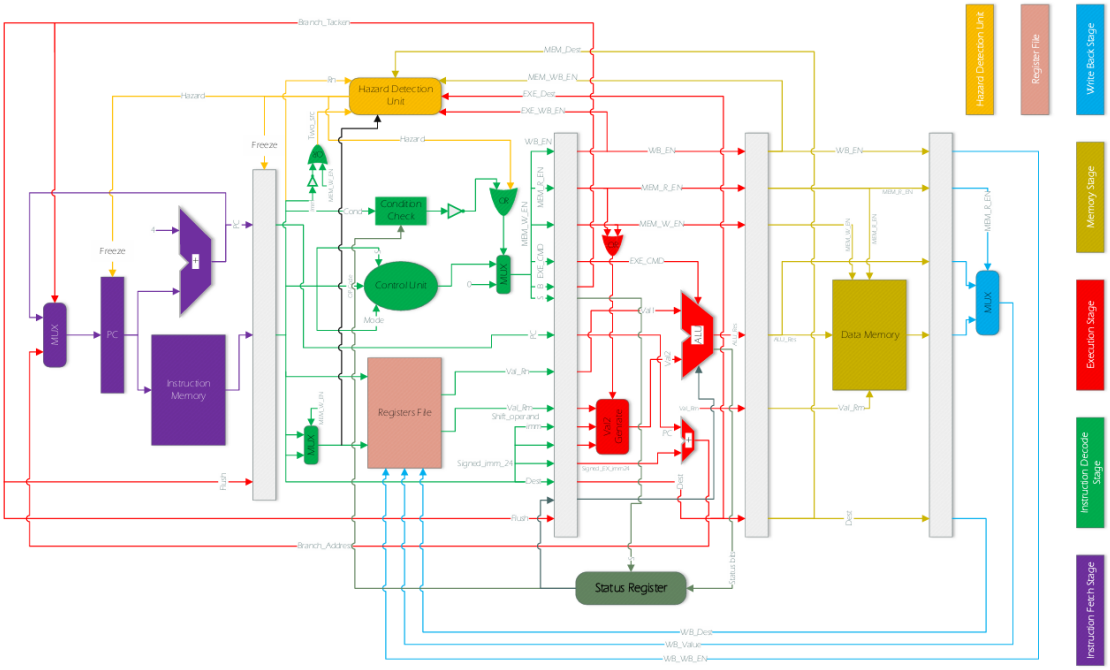
آزمایشگاه معماری کامپیوتر

نرگس سادات سیدحائری – 810100165
علی حمزه‌پور – 810100129

مقدمه

در این آزمایش، هدف اصلی طراحی و پیاده‌سازی یک پردازنده ساده مبتنی بر معماری ARM با 12 دستورالعمل است. این پردازنده باید به گونه‌ای طراحی شود که قابلیت اجرای عملیات‌های پایه را داشته باشد. برای دستیابی به این هدف، مراحل زیر به ترتیب انجام خواهند شد:

1. طراحی معماری پردازنده :ابتدا معماری اصلی پردازنده به زبان Verilog نوشته خواهد شد. این کد باید به گونه‌ای طراحی شود که قابلیت سنتز و اجرا بر روی سخت‌افزار را داشته باشد.
2. شبیه‌سازی و سنتز :پس از طراحی معماری، کد نوشته شده با استفاده از نرم‌افزار ModelSim شبیه‌سازی خواهد شد تا از صحت عملکرد آن اطمینان حاصل شود. سپس، کد با استفاده از نرم‌افزار Quartus II سنتز خواهد شد تا آماده‌سازی برای اجرا بر روی برد FPGA انجام گیرد.
3. پیاده‌سازی بر روی برد :پس از سنتز کد، برنامه بر روی برد FPGA برنامه‌ریزی خواهد شد و نتایج اجرای برنامه در ادامه گزارش می‌شود.
4. تست ماژول‌ها :برای هر یک از ماژول‌های پردازنده، یک ماژول تست جداگانه نوشته خواهد شد تا عملکرد هر بخش به صورت مجزا بررسی شود. پس از اطمینان از عملکرد صحیح هر ماژول، تمامی ماژول‌ها به یکدیگر متصل شده و کل پردازنده به صورت یکپارچه شبیه‌سازی و تست خواهد شد.
5. تست نهایی :در نهایت، یک Testbench سطح بالا طراحی خواهد شد که کد دودویی یک عملیات (مانند جمع دو عدد) را در Program ROM قرار داده و آن را خط به خط اجرا خواهد کرد تا جواب نهایی حاصل شود.



شکل پردازنده ARM

این مراحل به ما کمک می‌کند تا درک بهتری از طراحی و پیاده‌سازی پردازنده‌های ساده مبتنی بر معماری ARM داشته باشیم و با چالش‌های مرتبط با شبیه‌سازی، سنتز و اجرای کد بر روی سخت‌افزار آشنا شویم.

پیاده سازی پایپ لاین ARM:

مرحله اول: پیاده سازی IF Stage

مرحله‌ی IF اولین مرحله در خط لوله پردازنده است که در آن دستورالعمل‌ها از حافظه واکشی می‌شوند. این مرحله شامل واحدهای اصلی زیر است:

الف. شمارنده برنامه(PC)

شمارنده برنامه یا PC آدرس دستورالعمل بعدی را که باید اجرا شود، نگه‌می‌دارد. PC به صورت یک شمارنده ۳۲ بیتی عمل می‌کند که از صفر شروع به شمارش می‌کند و تا حداکثر مقدار خود افزایش می‌یابد. در صورت فعال شدن سیگنال Reset ، مقدار PC به صفر بازمی‌گردد. همچنین، اگر سیگنال freeze فعال باشد، مقدار PC به روز نمی‌شود.

مقدار PC در هر چرخه کلاک به‌روزرسانی می‌شود. این به‌روزرسانی می‌تواند به دو صورت انجام شود:

1. افزایش عادی: در حالت عادی، PC به اندازه ۴ بایت (یک دستورالعمل) افزایش می‌یابد تا به دستورالعمل بعدی اشاره کند. این افزایش توسط یک جمع‌کننده (Adder) انجام می‌شود.
2. پرش: در مواردی مانند دستورالعمل‌های پرش یا پرش شرطی، ممکن است نیاز باشد که PC به یک آدرس خاص (مثلاً آدرس هدف پرش) برود. در این حالت، مقدار PC از آدرس پرش بارگذاری می‌شود.

یک مولتی پلکسر بر اساس سیگنال کنترل‌کننده branch_taken تصمیم می‌گیرد که کدام یک از این دو مقدار را به عنوان خروجی انتخاب کند.

ب. حافظه دستورالعمل

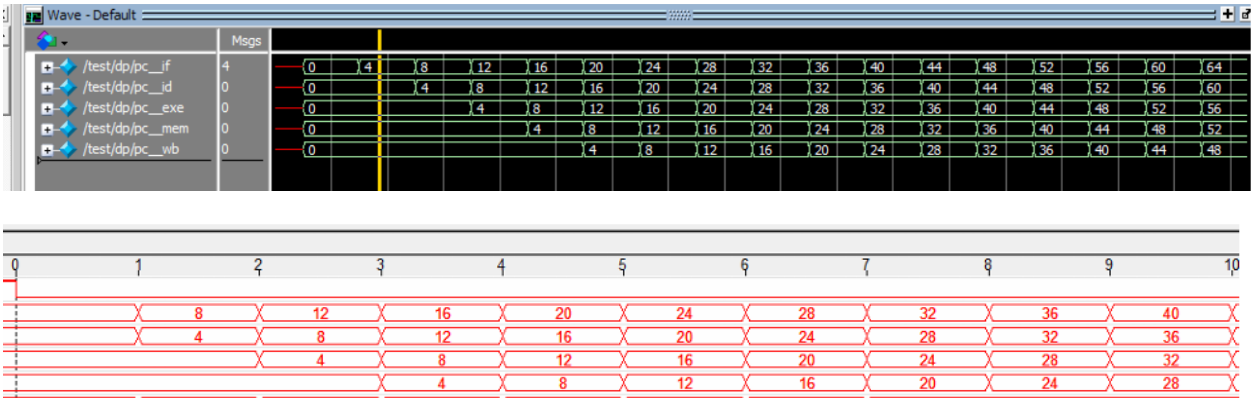
حافظه دستورالعمل (Instruction Memory) شامل دستورالعمل‌هایی است که باید اجرا شوند. در این پیاده‌سازی، حافظه دستورالعمل به صورت یک ساختار ساده با استفاده از switch-case پیاده‌سازی شده است که دستورالعمل‌های مورد نیاز را در خود نگه‌می‌دارد.

ج. جمع‌کننده

جمع‌کننده در این مرحله برای افزایش مقدار PC به کار می‌رود. پس از هر بار واکشی دستورالعمل، مقدار PC توسط جمع‌کننده افزایش می‌یابد تا به آدرس دستورالعمل بعدی اشاره کند. این افزایش معمولاً به اندازه ۴ بایت (یک دستورالعمل) است، زیرا هر دستورالعمل ۳۲ بیتی (۴ بایت) است.

د. نتایج

در این مرحله، دستورالعمل‌ها به درستی از حافظه واکشی می‌شوند و به همراه مقدار PC به مرحله بعدی ارسال می‌شوند. برای اطمینان از صحت پیاده‌سازی، رجیسترهای میانی بین مراحل مختلف خط لوله ایجاد شده‌اند. این رجیسترها سیگنال‌های PC و instruction را نگه‌می‌دارند و انتقال آن‌ها را بین مراحل خط لوله مدیریت می‌کنند. این کار به ما امکان می‌دهد تا حرکت پله پله دستورالعمل‌ها و PC را در مراحل مختلف خط لوله مشاهده و تأیید کنیم.



مرحله دوم : پیاده سازی ID Stage

مرحله‌ی ID جایی است که دستورالعمل واکشی‌شده تفسیر می‌شود. این مرحله شامل واحد کنترل CPU است که دستورالعمل را رمزگشایی می‌کند. یک نمای کلی از آنچه در این مرحله اتفاق می‌افتد به شرح زیر است:

الف. شناسایی دستورالعمل

دستورالعمل واکشی‌شده شناسایی می‌شود. این شامل تعیین نوع دستورالعمل و عملیات‌هایی است که انجام می‌دهد. این کار با تقسیم دستورالعمل به بخش‌های مختلف بر اساس فایل ارائه‌شده به ما انجام می‌شود.

ب. رمزگشایی دستورالعمل

دستورالعمل رمزگشایی می‌شود. دستورالعمل‌ها معمولاً به شکلی ذخیره می‌شوند که نیاز به ترجمه (رمزگشایی) دارند تا به سیگنال‌های کنترل برای اجرای دستورالعمل تبدیل شوند. این رمزگشایی یک لایه انتزاعی بین سخت‌افزار و نرم‌افزار ایجاد می‌کند که امکان تغییر سیگنال‌های کنترل مورد استفاده برای اجرا را بدون شکستن سازگاری باینری فراهم می‌کند.

ج. واحد کنترل

ماژول واحد کنترل (control unit) در ترکیب با ماژول بررسی شرط (Condition Check Module) ، به ما این امکان را می‌دهد تا یک پردازنده شرطی داشته باشیم. ماژول بررسی شرط بر اساس جدول 2 در فایل ARM Instruction Set Architecture (ISA) document کار می‌کند.

و ماژول واحد کنترل از قوانین جدول 3 در فایل برای ARM_ISA جهت تعیین دستور اجرایی برای ماژول ALU و همچنین سایر سیگنال‌های مورد نیاز مانند WB_EN ، MEM_R_EN، MEM_W_EN، Bو S پیروی می‌کند. همه این

سیگنال‌ها بر اساس نوع عملیاتی که در حال اجرا است فعال یا غیرفعال می‌شوند. اگر شرایط (خروجی ماژول بررسی شرط) برقرار نباشد یا با خطر hazard مواجه شویم، همه این سیگنال‌ها به ۰ تغییر می‌کنند که نشان‌دهنده NOP (عدم انجام عملیات) است.

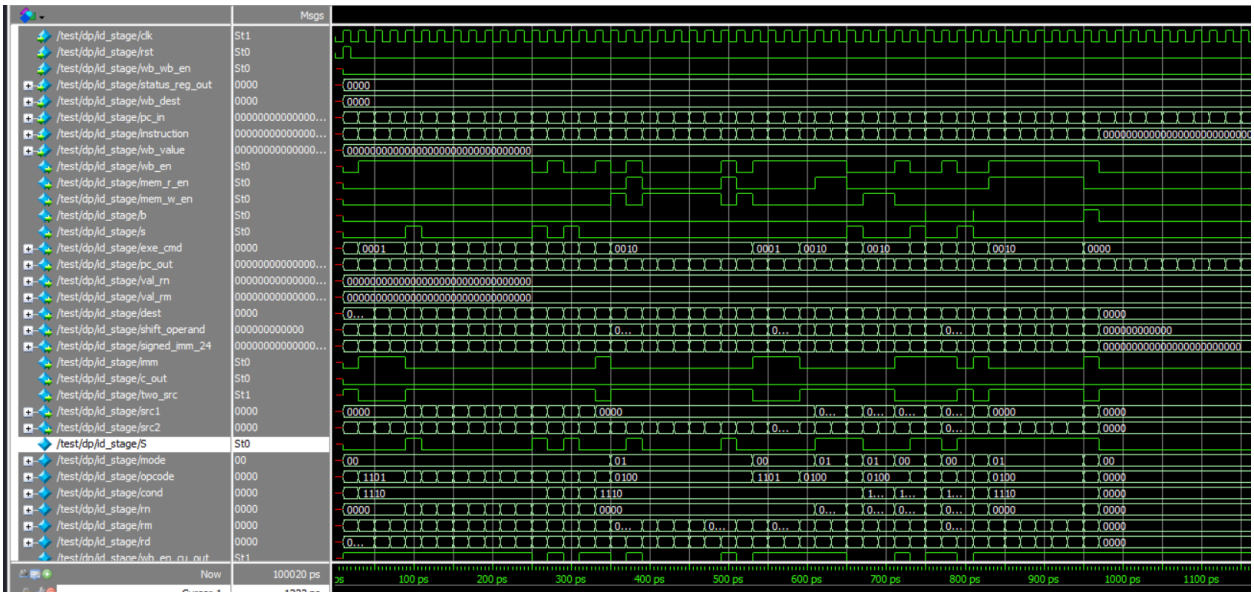
د. واکشی عملوندها

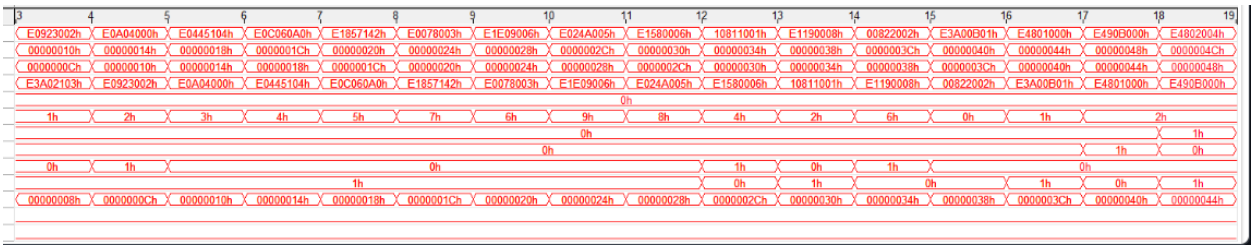
در این مرحله، عملوندهای دستورالعمل واکشی می‌شوند. این فرآیند شامل خواندن مقادیر از رجیسترها یا حافظه است. ماژول Register File به صورت همگام با کلاک نوشته می‌شود، اما خواندن به صورت ناهمگام انجام می‌گیرد. ورودی‌ها بر اساس دیکد دستورالعمل که قبلاً انجام شده، تعیین می‌شوند. با این حال، مالتی‌پلکسر برای ورودی دوم این ماژول برای دستوراتی مانند STR که از src2 استفاده می‌کنند، طراحی شده است. سایر دستورات از Rm برای ورودی دوم استفاده می‌کنند. علاوه بر این، برای بهبود عملکرد و جلوگیری از خواندن رجیسترهای غیرضروری، بررسی می‌شود که آیا رجیستر باید خوانده شود یا خیر. اگر نیازی نباشد، Register Fileبه جای مقدار، z(امپدانس بالا) برمی‌گرداند.

ه. تشخیص هازارد

مرحله دیکد در خط لوله همچنین می‌تواند انواع هازاردهای داده‌ای و ساختاری را نسبت به دستورالعمل‌های قبلی تشخیص دهد. در صورت نیاز، اقداماتی مانند استفاده از result forwarding یا اقدامات بازدارنده مانند متوقف کردن خط لوله یا stall کردن انجام می‌شود. این بخش به طور کامل در آینده توضیح داده خواهد شد.

و. نتایج





مرحله سوم : پیاده سازی EXE Stage و WB Stage

مرحله‌ی (Execution) EXE و (Write Back) WB دو مرحله مهم در خط لوله پردازنده هستند که به ترتیب عملیات محاسباتی و ذخیره نتایج را انجام می‌دهند. در ادامه، این مراحل به‌صورت کامل توضیح داده می‌شوند.

مرحله EXE

مرحله‌ی EXE جایی است که عملیات محاسباتی و منطقی بر روی داده‌ها انجام می‌شود. این مرحله شامل واحدهای اصلی زیر است:

الف. ALU

ماژول ALU عملیات محاسباتی و منطقی را بر اساس سیگنال‌های EXE_CMD انجام می‌دهد. ALU دو ورودی اصلی دارد: 1. مقدار رجیستر Rn 2. توسط ماژول val2generate تولید می‌شود. این ورودی می‌تواند یک مقدار ثابت (immediate) یا نتیجه یک عملیات شیفت باشد.

این ماژول علاوه بر نتیجه عملیات، بیت‌های وضعیت (Status Flags) مانند(Carry) C ، (Overflow) V، N (Negative)و (Zero) Z را نیز تولید می‌کند. این بیت‌ها برای دستورات شرطی و تصمیم‌گیری‌های بعدی استفاده می‌شوند.

ب. Val2Generate

ماژول Val2Generate وظیفه محاسبه اپرند دوم ALU را بر عهده دارد. این ماژول دارای چهار ورودی اصلی است : MemInstکه از OR شدن EN_R_MEM و EN_W_MEM به‌دست می‌آید و نشان‌دهنده دستورات مرتبط با حافظه است، Immکه بیت ۲۵ دستور است و مشخص‌کننده نوع اپرند دوم است، ShifterOperandکه ۱۲ بیت سمت راست دستور است و ValRm که مقدار رجیستر دوم است. خروجی این ماژول یک مقدار ۳۲ بیتی است که به عنوان ورودی دوم ALU استفاده می‌شود.

این ماژول در چند حالت مختلف عمل می‌کند. اگر دستور مربوط به حافظه باشد، خروجی مقدار Extend-Sign شده ۱۲ بیت ShifterOperand است. اگر Imm برابر با ۱ باشد، ۸ بیت سمت راست ShifterOperand به صورت دایره‌ای شیفت می‌خورد. در حالت دیگر، اگر Imm برابر با ۰ باشد، ShifterOperandنوع شیفت مانندLSL ، LSR، ASR، RORو مقدار شیفت را مشخص می‌کند و ValRm بر اساس آن شیفت داده می‌شود. این ماژول به‌طور موثر ورودی دوم ALU را بر اساس نوع دستور و پارامترهای آن محاسبه می‌کند.

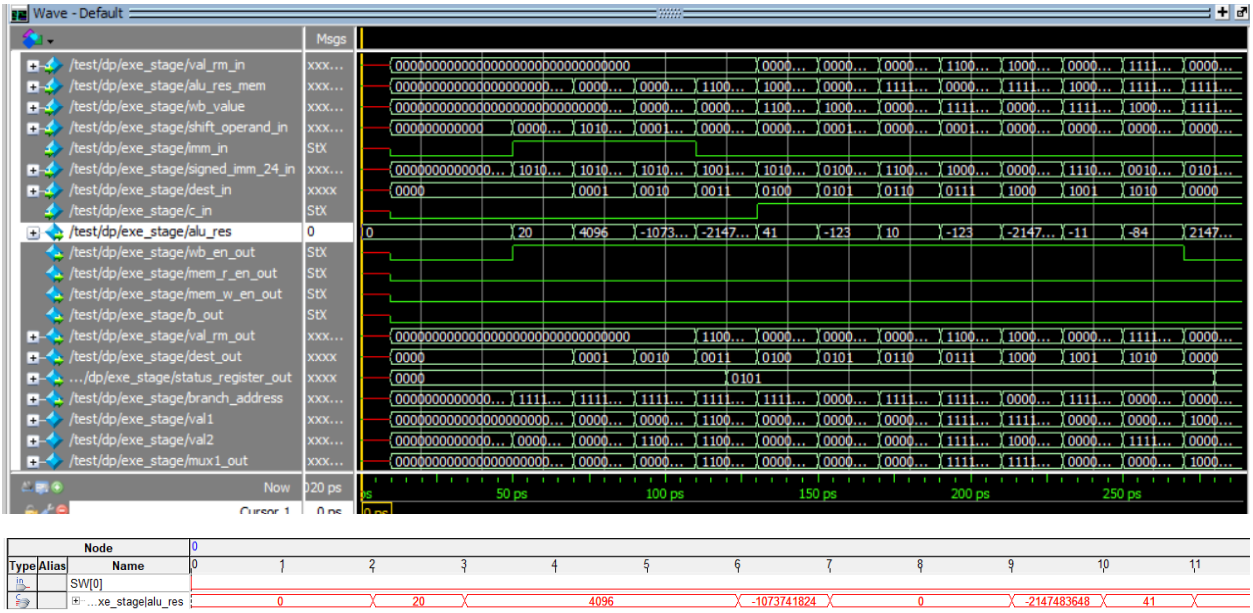
ج. ثبات وضعیت

ثبات وضعیت برای نگهداری بیت‌های وضعیت (C, V, N, Z) استفاده می‌شود. این بیت‌ها از خروجی ALU دریافت می‌شوند و برای دستورات شرطی و تصمیم‌گیری‌های بعدی مورد استفاده قرار می‌گیرند.

د. پیدا کردن آدرس پرش

ماژول Adder یک جمع‌کننده ۳۲ بیتی است که برای محاسبه آدرس Branch مورد استفاده قرار می‌گیرد. این ماژول دو ورودی اصلی دارد: مقدار PC + 4 که آدرس دستورالعمل بعدی در حالت عادی است و مقدار Extend Sign شده ۲۴ بیت سمت راست دستور Branch که نشان‌دهنده آفست پرش است. خروجی این ماژول حاصل جمع این دو مقدار است که به مرحله اول پایپلاین یعنی IF بازمی‌گردد. این آدرس محاسبه‌شده به عنوان آدرس هدف پرش استفاده می‌شود و به PC منتقل می‌شود تا دستورالعمل بعدی از آدرس جدید واکشی شود. این ماژول نقش کلیدی در اجرای دستورات پرش و تغییر مسیر اجرای برنامه دارد.

ه. نتایج



مرحله WB

مرحله WB یا Write Back آخرین مرحله در خط لوله پردازنده است که در آن نتایج عملیات به رجیسترهای عمومی بازنویسی می‌شوند. این مرحله شامل واحدهای اصلی زیر است:

در این مرحله، یک مالتی‌پلکسر (MUX) وجود دارد که تصمیم می‌گیرد داده‌های خروجی از کدام منبع به رجیسترهای عمومی ارسال شوند. اگر سیگنال MEM_R_EN فعال باشد، داده از حافظه (Memory) خوانده می‌شود و به رجیسترها ارسال می‌شود. اگر MEM_R_EN غیرفعال باشد، داده از مرحله EXE یعنی خروجی ALU به رجیسترها ارسال می‌شود. این انتخاب بر اساس سیگنال‌های کنترل‌کننده‌ای مانند WB_EN و MEM_R_EN انجام می‌شود.

نتایج عملیات، اعم از داده‌های محاسبه‌شده توسط ALU یا داده‌های خوانده‌شده از حافظه، در رجیسترهای عمومی بازنویسی می‌شوند. این کار بر اساس آدرس مقصد (Destination Register) که در مرحله ID تعیین شده است، انجام می‌شود. این مرحله اطمینان حاصل می‌کند که نتایج نهایی عملیات به درستی در رجیسترهای مورد نظر ذخیره می‌شوند و برای دستورات بعدی قابل استفاده هستند.

مرحله چهارم : پیاده سازی MEM Stage

پردازنده در مرحله Memory با سیستم حافظه تعامل دارد. در این مرحله، پردازنده برای دستورات load داده‌ها را از حافظه خوانده و آن‌ها را در یک رجیستر ذخیره می‌کند. برای دستوراتstore ، پردازنده داده‌ها را در حافظه می‌نویسد. در ابتدا، این مرحله با استفاده از ماژول Data Memory پیاده‌سازی شد، اما بعداً به یک رویکرد پیشرفته‌تر با استفاده از SRAM و Cache تغییر یافت. ماژول Data Memory تنها یک آرایه ساده از رجیسترها است که آدرس و مقدار را به عنوان ورودی دریافت می‌کند و سپس یا با لبه پایین‌رونده کلاک داده را می‌نویسد یا به صورت ناهمگام آن را می‌خواند. تنها نکته قابل ذکر این است که از آنجا که این حافظه به صورت بایت آدرس‌دهی شده است، برای نوشتن مقادیر ۳۲ بیتی، دو صفر به انتهای آدرس اضافه می‌شود.

مرحله پنجم : پیاده سازی Hazard Unit

در این مرحله، ماژول تشخیص و مدیریت مخاطره‌ها (Hazard Unit) به پردازنده اضافه می‌شود. این ماژول وظیفه تشخیص و رفع مخاطره‌های داده‌ای، به ویژه مخاطره‌های نوع(RAW (Read After Write ، را بر عهده دارد. مخاطره RAWزمانی رخ می‌دهد که یک دستور نیاز به خواندن از رجیستری دارد که هنوز توسط دستور قبلی نوشته نشده است. برای رفع این مشکل، Hazard Unit آدرس رجیسترها(src1 , src2) در مرحله ID را با مقصدهای دستورات در مراحل EXE و MEM مقایسه می‌کند. اگر یکی از منابع با مقصد دستورات در این مراحل برابر باشد و سیگنال EN_WBفعال باشد، مخاطره تشخیص داده می‌شود و سیگنال hazard فعال می‌شود.

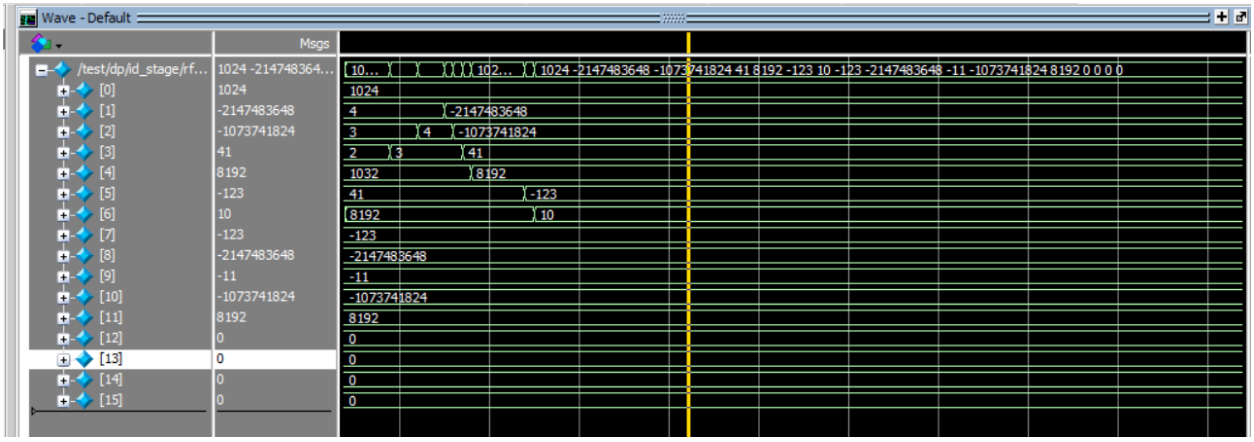
هنگامی که مخاطره تشخیص داده می‌شود، دستورات درون IF و رجیسترهای پس از آن متوقف می‌شوند و یک حباب در خط لوله ایجاد می‌شود. این کار با صفر کردن سیگنال‌های کنترلی خروجی از واحد کنترل انجام می‌شود. برای پیاده‌سازی این مرحله، ورودی‌های مورد نیاز Hazard Unit از مراحل مختلف خط لوله تأمین می‌شوند و خروجی آن به سیگنال Freeze در رجیستر PC و رجیسترهای پس از IF متصل می‌شود. این مرحله آخرین بخش از پیاده‌سازی پردازنده ARM است و قسمت‌های بعدی شامل ویژگی‌های اضافی است که به ARM اضافه می‌شوند.

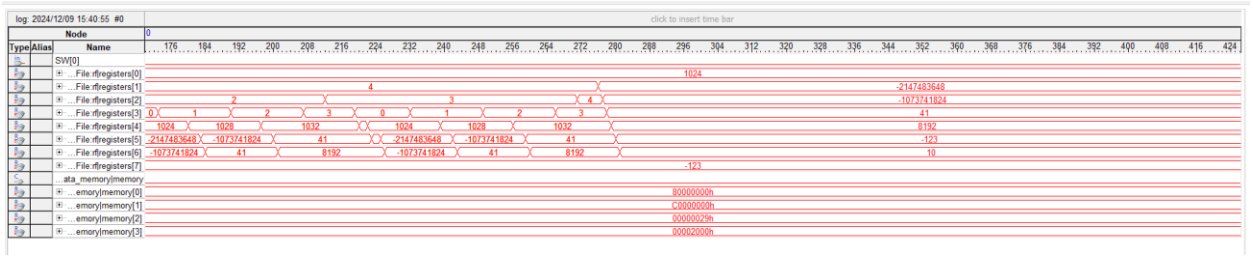
بخش امتیازی

در بعضی از دستورات src1 وجود ندارد اما پیاده‌سازی خواسته‌شده آن را تشخیص نمی‌دهد. این دستورات شامل MOV, MVN, NOP, B هستند. برای حل آن می‌توان به ماژول واحد کنترل خروجی جدیدی اضافه کرد تا مشخص کرد که آیا دستور فعلی شامل src1 هست یا نه. Hazard unit نیز تنها در حالتی که دستور دارای src1 باشد، شروط آن را بررسی کند.(تغییرات مربوط در این کامیت قابل مشاهده است.)

نتایج نهایی پیاده‌سازی ARM

پس از پایان مرحله‌ی 5، برنامه محک را با بر پردازنده اجرا کردیم و مشاهده کردیم که اعداد به درستی مرتب شدند و برنامه بدون مشکل به خروجی مدنظر رسید. خروجی modelsim و signaltab آن را می‌توانید در زیر مشاهده کنید. همچنین اطلاعات compile آن نیز قابل مشاهده است.





Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	ARM
Top-level Entity Name	ARM
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	7,753 / 33,216 (23 %)
Total combinational functions	4,066 / 33,216 (12 %)
Dedicated logic registers	5,853 / 33,216 (18 %)
Total registers	5853
Total pins	418 / 475 (88 %)
Total virtual pins	0
Total memory bits	396,288 / 483,840 (82 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

مرحله ششم : پیاده سازی Forwarding Unit

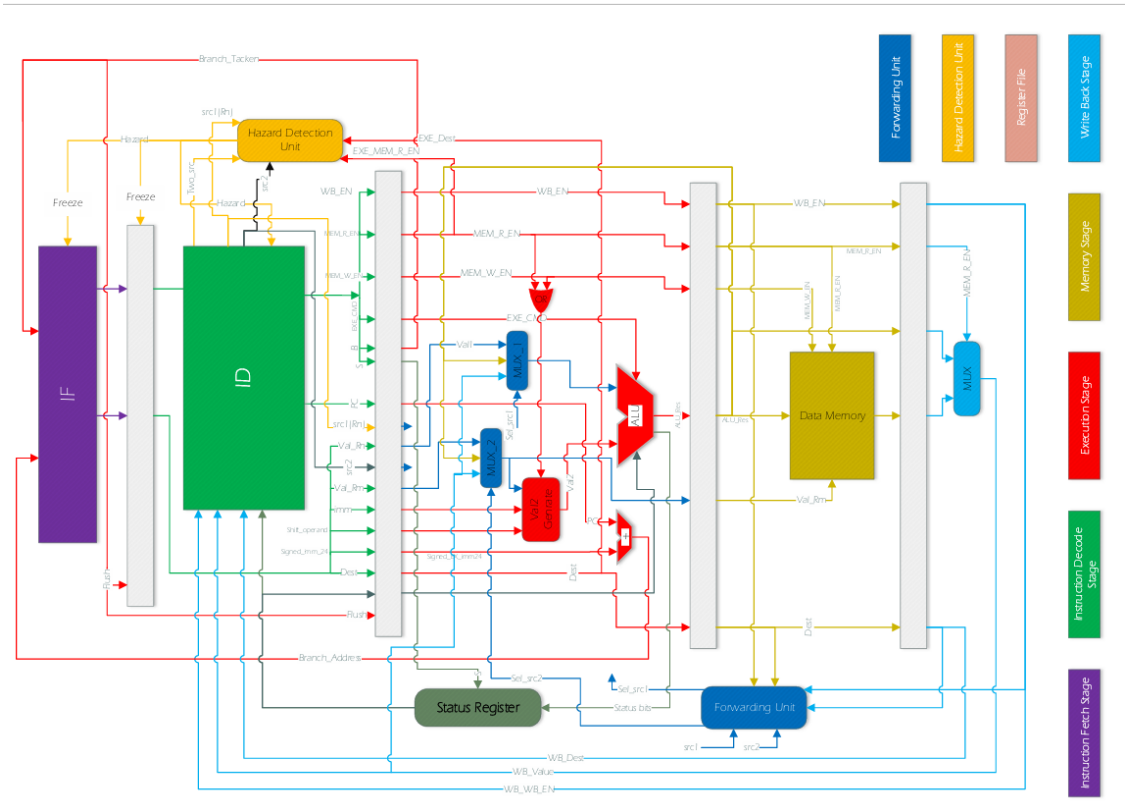
ماژول Forwarding Unit برای مدیریت تداخل‌های داده‌ای (data hazards) در پایپ‌لاین پردازنده طراحی شده است.

این ماژول با تشخیص شرایطی که در آن دستورات بعدی به داده‌های تولید شده توسط دستورات قبلی نیاز دارند، از استال کردن (stall) پایپ‌لاین جلوگیری می‌کند. در صورت فعال بودن سیگنال forward_en، این ماژول داده‌های مورد نیاز را مستقیماً از مراحل بعدی پایپ‌لاین (MEM یا WB) به مراحل قبلی مانند EXE منتقل می‌کند.

این ماژول دو سیگنال خروجی به نام‌های sel_src1 وsel_src2تولید می‌کند که به Mux های مرحله EXE متصل هستند. این Mux ها بین سه منبع داده تصمیم‌گیری می‌کنند: مقدار رجیستر از مرحله ID ، مقدار به‌روز شده توسط ALU از مرحله MEM ، و مقدار نهایی از مرحله WB. اگر Forwarding Unit فعال باشد، داده‌های صحیح از مراحل بعدی پایپ‌لاین انتخاب می‌شوند. در غیر این صورت، مقدار رجیستر از مرحله ID استفاده می‌شود.

در صورت عدم وجود Forwarding Unit ، تمامی تداخل‌های داده‌ای توسط Hazard Unit تشخیص داده شده و با استال کردن پایپ‌لاین مدیریت می‌شدند. با این حال، Forwarding Unit این امکان را فراهم می‌کند که به جای استال کردن، داده‌ها مستقیماً از مراحل بعدی به مراحل قبلی منتقل شوند. تنها استثنا دستور LDR است که به دلیل عدم پایداری داده‌های خوانده شده از حافظه، نیاز به استال کردن پایپ‌لاین دارد.

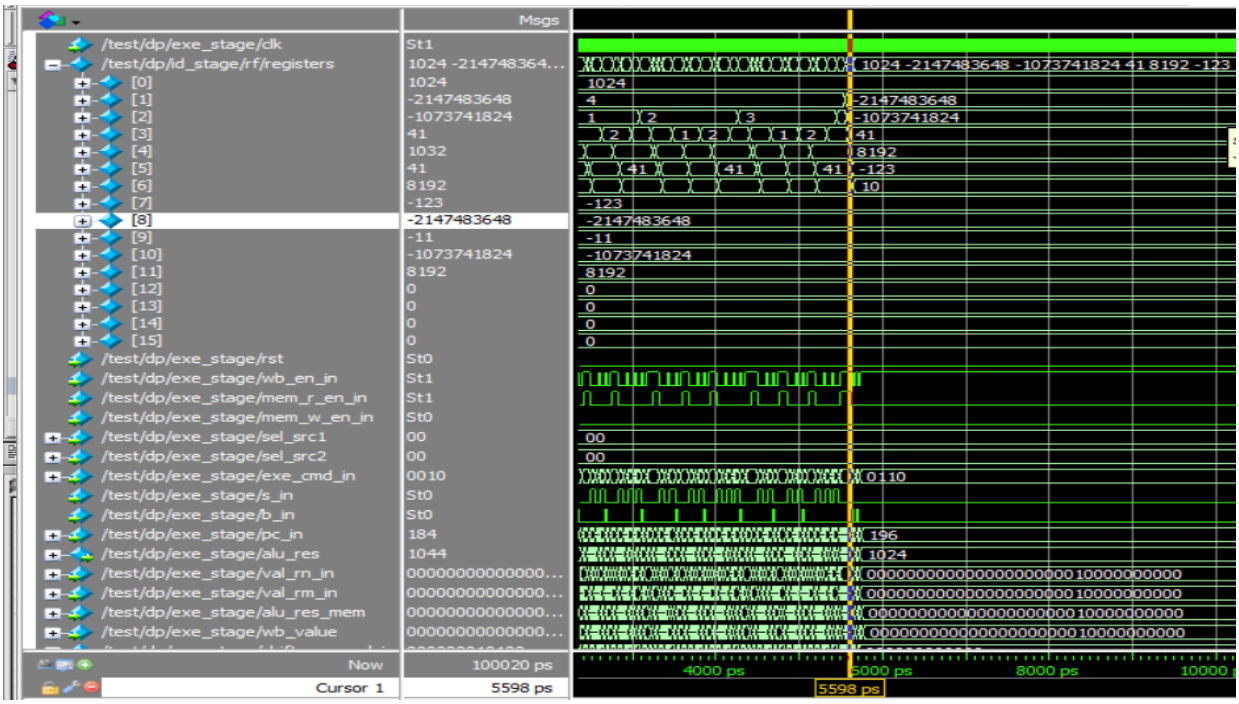
این مکانیزم باعث بهبود عملکرد پردازنده و کاهش تاخیرهای ناشی از تداخل‌های داده‌ای می‌شود.



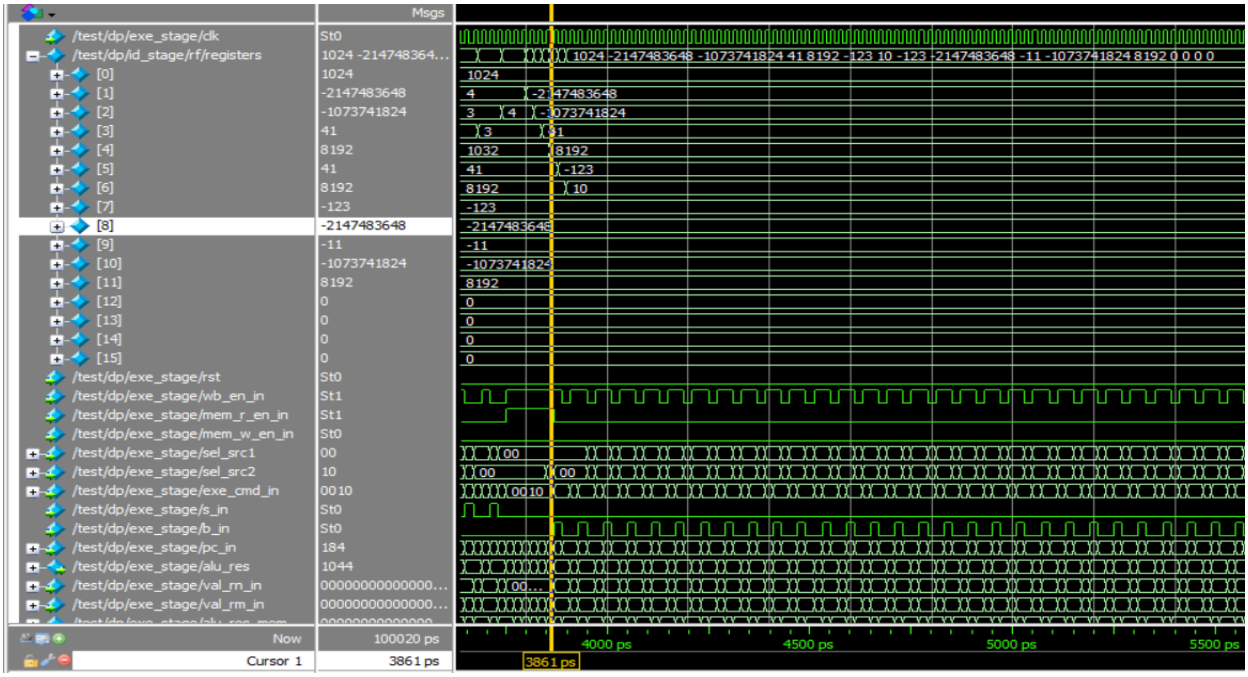
شکل پردازنده با قابلیت ارسال رو به جلو

حال برنامه را با دو حالت فعال بودن و غیرفعال بودن forwarding تست میکنیم تا میزان افزایش کارایی را بدست آوریم.

خروجی بدون استفاده از Forwarding:



خروجی با استفاده از Forwarding:



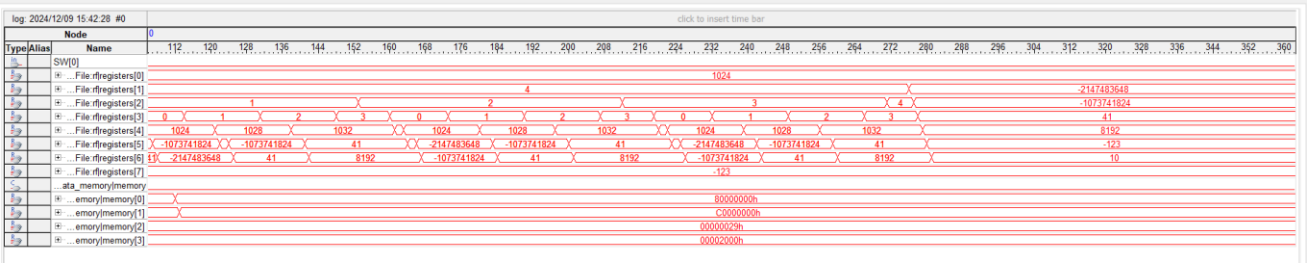
همان‌طور که مشاهده می‌شود، زمان اجرای برنامه در حالت غیر فعال بودن Forwarding 5598نانوثانیه و در حالت فعال‌سازی 3861 نانوثانیه طول کشیده است. با محاسبه بهبود عملکرد، به نتیجه زیر می‌رسیم:

$$\frac{5600 - 3860}{5600} \times 100 \cong 31.1\% \ \ improvement$$

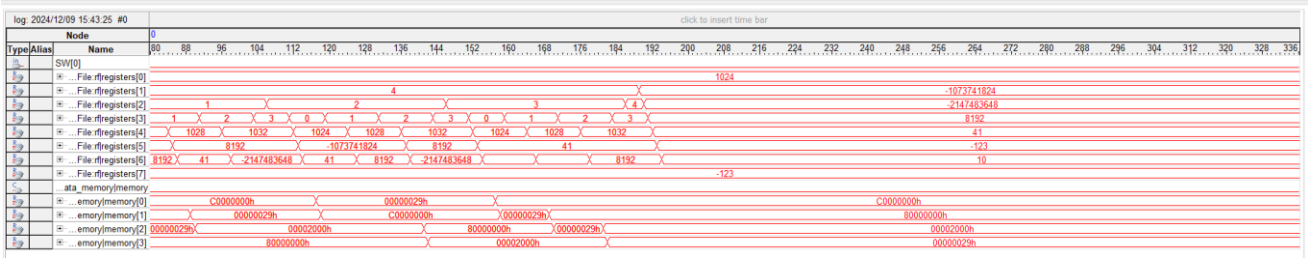
این بهبود 31.1 درصدی نشان‌دهنده تأثیر مثبت استفاده از Forwarding در کاهش زمان اجرای برنامه و افزایش کارایی پردازنده است.

نتیجه سنتز:

بدون استفاده از Forwarding:



با استفاده از Forwarding:



محاسبه هزینه سخت افزاری:

Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	ARM
Top-level Entity Name	ARM
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	7,753 / 33,216 (23 %)
Total combinational functions	4,066 / 33,216 (12 %)
Dedicated logic registers	5,853 / 33,216 (18 %)
Total registers	5853
Total pins	418 / 475 (88 %)
Total virtual pins	0
Total memory bits	396,288 / 483,840 (82 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

$$\frac{9564 - 7753}{7753} \times 100 \cong 23\% \ \ increase\ in\ hardware\ cost$$

محاسبه کارایی بر هزینه:

در این قسمت نسبت افزایش کارایی و افزایش هزینه را محاسبه می‌کنیم:

$$\frac{5600 - 3860}{9564 - 7753} = 0.96$$

به این معنا که به ازای هر المان سخت‌افزاری که اضافه شده است، 1ps کاهش زمان داشتیم(که معادل 0.1 کلاک است.)

بخش امتیازی:

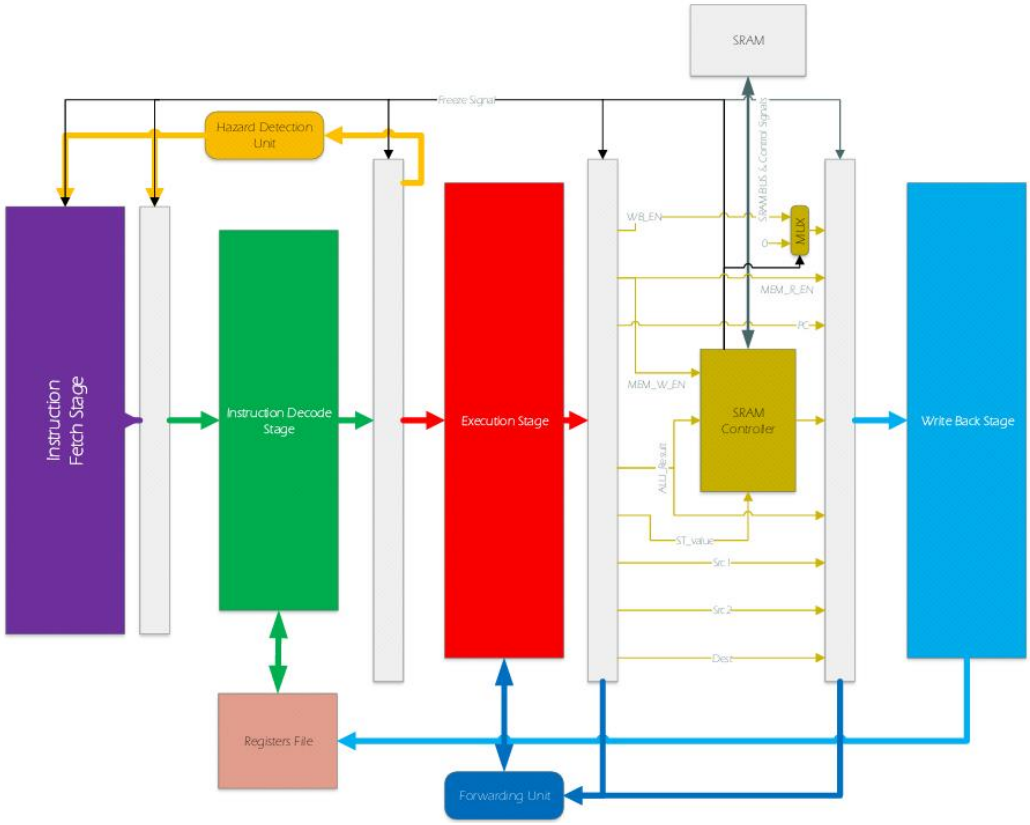
برای بهبود کارایی پردازنده علاوه بر روش ارسال به جلو(Forwarding) ، می‌توان از روش‌های دیگری نیز استفاده کرد. یکی از این روش‌ها، استفاده از اجرای نامنظم دستورات (Out-of-Order Execution) است. در این روش، پردازنده دستورات را به ترتیبی که وابستگی‌های داده‌ای و کنترلی اجازه می‌دهند اجرا می‌کند، نه لزوماً به ترتیبی که در برنامه نوشته شده‌اند. این کار باعث می‌شود که دستورات مستقل از یکدیگر به صورت موازی اجرا شوند و از زمان‌های توقف (استال) جلوگیری شود.

روش دیگر، استفاده از دستورات SIMD (Single Instruction, Multiple Data) است. این دستورات به پردازنده اجازه می‌دهند تا یک عملیات را روی چندین داده به طور همزمان انجام دهد. این روش به ویژه برای برنامه‌هایی که نیاز به پردازش موازی داده‌ها دارند، مانند پردازش تصویر یا محاسبات ماتریسی، بسیار مفید است.

همچنین، استفاده از Cacheو Branch Predictionنیز می‌تواند کارایی پردازنده را افزایش دهد .با کاهش Cache زمان دسترسی به حافظه اصلی، سرعت اجرای برنامه‌ها را افزایش می‌دهد .نیز با پیش‌بینی Branch Prediction صحیح شاخه‌های شرطی، از توقف‌های ناشی از اشتباه در پیش‌بینی شاخه‌ها جلوگیری می‌کند.

این روش‌ها باید با توجه به معیارهای قابلیت پیاده‌سازی، هزینه، و بهبود کارایی ارزیابی شوند. هر یک از این روش‌ها می‌توانند به تنهایی یا در ترکیب با یکدیگر، کارایی پردازنده را به طور قابل توجهی افزایش دهند.

مرحله‌ی هفتم: پیاده سازی SRAM

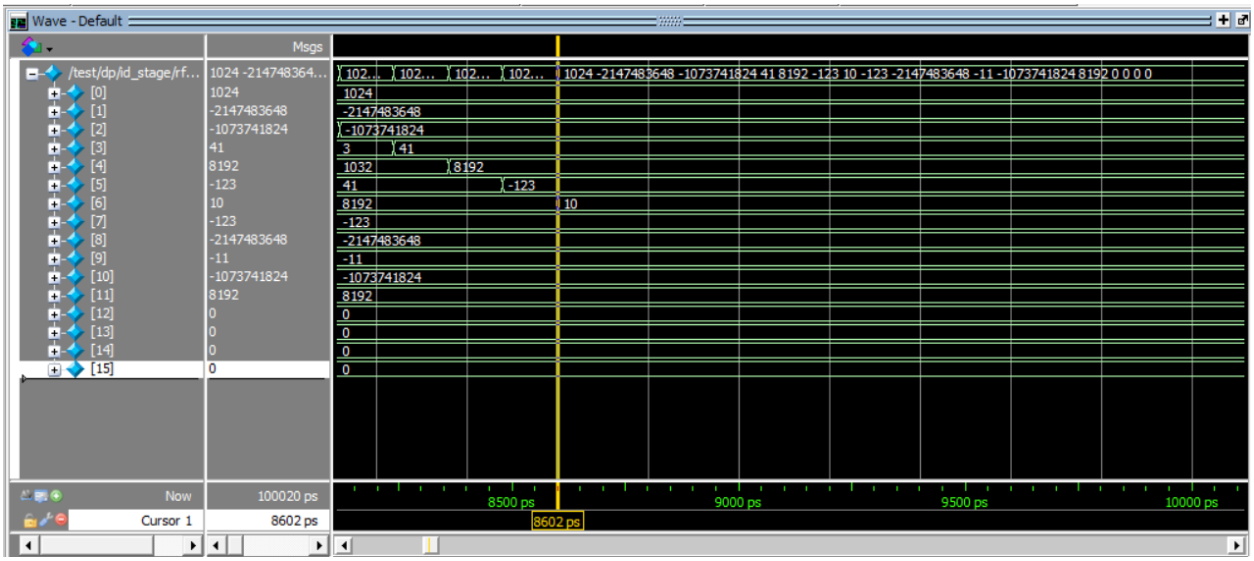


شکل پردازنده با استفاده از SRAM

الف. تغییرات در سطح RTL

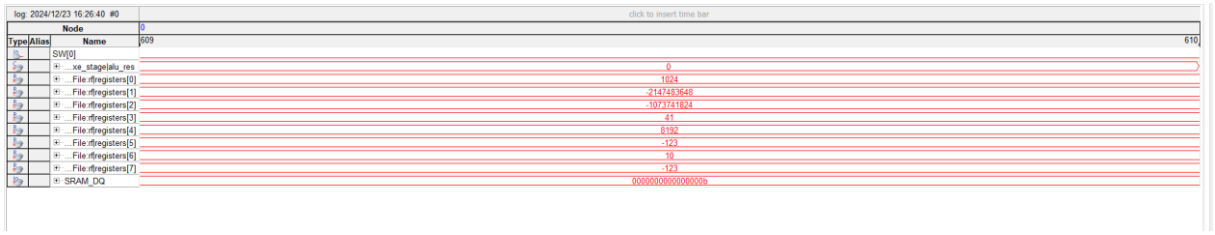
در سطحRTL ، تغییرات اصلی شامل طراحی یک ماژول کنترل‌کننده برای حافظه SRAM است که دسترسی به حافظه را با کلاک پردازنده همگام می‌کند. این ماژول خطوط داده برای خواندن و نوشتن را از هم جدا کرده و اطمینان حاصل می‌کند که عملیات حافظه به درستی انجام می‌شود. همچنین، به دلیل تفاوت در اندازه کلمات (۱۶ بیتی در SRAM و ۳۲ بیتی در پردازنده)، تغییراتی در نحوه خواندن و نوشتن داده‌ها اعمال شده است، به طوری که هر کلمه ۳۲ بیتی به دو بخش ۱۶ بیتی تقسیم و در آدرس‌های متوالی ذخیره می‌شود. علاوه بر این، برای مدیریت تاخیر دسترسی به حافظه، مکانیزم‌هایی برای متوقف کردن خط لوله (Freeze) هنگام دسترسی به حافظه اضافه شده است تا پردازنده بتواند به درستی با حافظه خارجی کار کند.

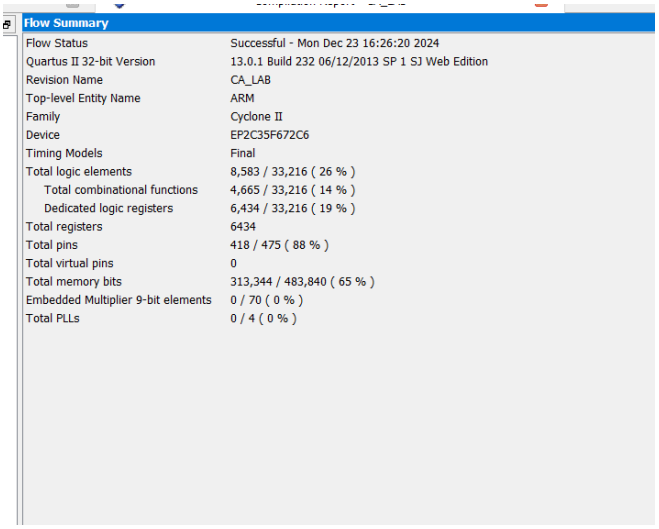
ب. نتایج و مقایسه میزان کارایی



$$\frac{3860 - 8600}{8600} \times 100 = 55\% \; decrease \; in \; performance$$

ج. نتایج سنتز و مقایسه میزان هزینه سخت افزار





Flow Summary	
Flow Status	Successful - Mon Dec 23 16:26:20 2024
Quartus II 32-bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	CA_LAB
Top-level Entity Name	ARM
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	8,583 / 33,216 (26 %)
Total combinational functions	4,665 / 33,216 (14 %)
Dedicated logic registers	6,434 / 33,216 (19 %)
Total registers	6434
Total pins	418 / 475 (88 %)
Total virtual pins	0
Total memory bits	313,344 / 483,840 (65 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

$$\frac{8583 - 9564}{9564} \times 100 \cong 10\% \ decrease\ in\ hardware\ cost$$

دلیل کاهش آن می‌تواند این باشد که از sram خود FPGA استفاده می‌کنیم و دیگر المان‌های سخت‌افزاری حافظه در کد قبلی ما حذف شدند.

بخش امتیازی

استفاده از حافظه خارجی مانند SRAM به دلیل تاخیر دسترسی بیشتر، کارایی پردازنده را کاهش می‌دهد. برای بهبود کارایی، می‌توان از روش‌هایی مانند پیش‌بینی دستورات (Branch Prediction) استفاده کرد تا دستورات پرش به‌طور موثرتری مدیریت شوند و تاخیر ناشی از دستورات شرطی کاهش یابد. همچنین، افزایش موازی‌سازی در خط لوله با استفاده از تکنیک‌هایی مانند اجرای سوپراسکالر (Superscalar) می‌تواند تعداد دستورات اجرا شده در هر چرخه کلاک را افزایش دهد.

مرحله هشتم: پیاده سازی Cache

حافظه کش یک نوع حافظه کوچک و فرار در کامپیوتر است که دسترسی پرسرعت به داده‌ها را برای پردازنده فراهم می‌کند. این حافظه با ذخیره‌سازی برنامه‌ها، اپلیکیشن‌ها و داده‌های پرکاربرد، سرعت اجرای عملیات‌ها را به طور قابل توجهی افزایش می‌دهد. به عنوان سریع‌ترین نوع حافظه در کامپیوتر، حافظه کش معمولاً روی مادربرد تعبیه شده یا مستقیماً در پردازنده یا RAM قرار می‌گیرد.

در بخش‌های قبلی آزمایش، یک کنترلر SRAM پیاده‌سازی شد تا به جای استفاده از یک حافظه بسیار کوچک درون پردازنده، از یک حافظه بزرگ‌تر خارجی استفاده کنیم. اما این پیاده‌سازی یک مشکل اساسی به همراه داشت: تاخیر بسیار زیاد در خواندن و نوشتن داده‌ها در حافظه خارجی. برای هر عملیات حافظه، پردازنده مجبور بود ۶ کلاک صبر کند و در این مدت نمی‌توانست کار دیگری انجام دهد.

برای رفع این مشکل، در این بخش از آزمایش، یک حافظه نهان (Cache) پیاده‌سازی شد. این حافظه کش به عنوان یک لایه میان‌افزار بین پردازنده و حافظه اصلی عمل می‌کند و دسترسی به داده‌های پرکاربرد را تسریع می‌بخشد.

کنترلر کش (Cache Controller)

برای اضافه کردن حافظه کش به معماری پردازنده، یک کنترلر کش (Cache Controller) طراحی شد. این کنترلر به صورت ترکیبی (Combinational) پیاده‌سازی شده است و وظیفه مدیریت دسترسی به حافظه کش و هماهنگی با کنترلر SRAM را بر عهده دارد. کنترلر کش فرمان‌های کنترلی لازم را به SRAM Controller ارسال می‌کند و دسترسی به داده‌ها را بهینه‌سازی می‌کند.

ساختار حافظه کش

در آزمایشگاه ما، یک طراحی کش دوطرفه (Two-Way Set Associative Cache) پیاده‌سازی شده است. در این طراحی، هر کلمه در کش ۳۲ بیت است و هر بلوک شامل دو کلمه می‌باشد. همچنین، هر ردیف شامل دو بلوک و در کل ۶۴ ردیف وجود دارد که در مجموع ۸۱۹۲ بیت یا ۱ کیلوبایت حافظه کش را تشکیل می‌دهد. این طراحی با تقسیم آدرس ورودی به بخش‌های offset ، index و tag، امکان دسترسی کارآمد به داده‌ها را فراهم می‌کند. کش

با استفاده از بیت‌های index ردیف مناسب را پیدا کرده و با مقایسه tag کش با tag آدرس، تشخیص می‌دهد که آیا داده مورد نظر در کش وجود دارد (Hit) یا خیر(Miss). همچنین، سیاست‌های دسترسی به داده برای مدیریت کارآمد بلوک‌های کش و بازیابی سریع داده‌های پرکاربرد پیاده‌سازی شده‌اند.

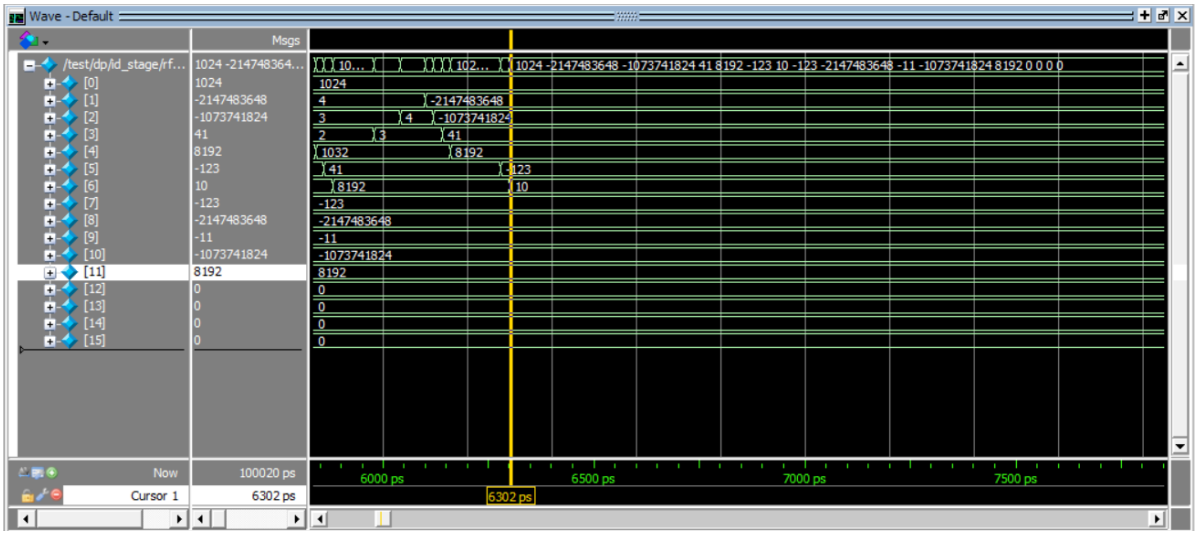
سیاست جایگزینی

برای مدیریت موثر کش، از سیاست کمترین استفاده اخیر (Least Recently Used - LRU) برای جایگزینی داده‌ها استفاده می‌شود. این سیاست اطمینان می‌دهد که داده‌هایی که اخیراً استفاده شده‌اند در دسترس باقی می‌مانند، در حالی که داده‌های قدیمی‌تر در صورت نیاز جایگزین می‌شوند. در ابتدا، تمام بیت‌های valid در کش روی صفر تنظیم می‌شوند که نشان‌دهنده عدم وجود داده معتبر است.

هنگام خواندن داده از کش، بیت LRU به‌روزرسانی می‌شود تا نشان دهد کدام way اخیراً استفاده شده است. برای جایگزینی داده جدید، اگر بیت LRU برابر ۰ باشد، ۲way جایگزین می‌شود؛ در غیر این صورت، ۱way جایگزین می‌شود. پس از جایگزینی، بیت LRU به‌روزرسانی می‌شود.

در صورت نوشتن داده در کش، اگر داده قبلاً وجود داشته باشد، فقط بیت valid به‌روزرسانی می‌شود. در غیر این صورت، داده جدید نوشته شده و بیت valid به ۱ تنظیم می‌شود. از روش Write Throughاستفاده شده است، یعنی داده هم در کش و هم در حافظه اصلی به‌روزرسانی می‌شود تا همگام‌سازی حفظ شود.

خروجی برنامه با استفاده از cache :



$$\frac{6300 - 8600}{8600} \times 100 = 26\% \; increase \; in \; performance$$