کاهش اثرات سالمندی در حافظه نهان دستورالعمل با استفاده از روش کدگذاری مجموعه دستورالعمل

سید میلاد ابراهیمی‌پور[۱]، بهنام قوامی[۲]، محسن راجی[۳]

[۱] دانشجوی کارشناسی ارشد، گروه مهندسی کامپیوتر، دانشگاه شهید باهنر کرمان ، کرمان،
miladebrahimi@eng.uk.ac.ir

[۲] دانشیار، گروه مهندسی کامپیوتر، دانشگاه شهید باهنر کرمان ، کرمان،
ghavami@uk.ac.ir

[۳] استادیار، دانشکده مهندسی برق و کامپیوتر، دانشگاه شیراز ، شیراز،
mraji@shirazu.ac.ir

چکیده

با پیشرفت تکنولوژی و کاهش ابعاد ترانزیستورها، چالش‌های جدیدی در حوزه قابلیت اطمینان تراشه‌های دیجیتال بوجود آمده است. از جمله این چالش‌ها می‌توان به سالمندی ترانزیستورها اشاره کرد که باعث کاهش کارآیی و تخریب عملکرد مدار می‌شود. سالمندی ترانزیستورها موجب کاهش حاشیه نویز ایستا در سلول‌های حافظه می‌گردد. تاکنون روش‌های زیادی به منظور کاهش اثرات سالمندی در حافظه‌ها ارائه شده است، اما تمامی این روش‌ها سربار مساحت زیادی را به سیستم تحمیل می‌کنند. در این مقاله، یک روش کدگذاری مجموعه دستورالعمل آگاه از سالمندی به منظور کاهش اثرات سالمندی در حافظه نهان دستورالعمل ارائه شده است. در روش پیشنهادی بخش‌های مختلف یک دستورالعمل به گونه‌ای کدگذاری مجدد می‌شوند که اثرات ناشی از سالمندی در حافظه نهان دستورالعمل کاهش یابد. نتایج حاصل از شبیه‌سازی نشان می‌دهد که روش پیشنهادی به‌طور میانگین حدود ۳۴/۳۵٪ اثرات ناشی از سالمندی را در یک حافظه نهان بهبود می‌بخشد.

کلمات کلیدی

قابلیت اطمینان، سالمندی، حاشیه نویز ایستا، حافظه نهان

۱- مقدمه

نیاز به کارآیی بیشتر، طراحان مدارهای دیجیتال را بر آن داشت تا اندازه ساخت ترانزیستورها را به ابعاد کمتر از چندین نانومتر کاهش دهند. این پیشرفت‌ها باعث افزایش نمایی تعداد ترانزیستورها، افزایش فرکانس و کاهش ولتاژ کاری مدار شده‌اند[1]. با پیشرفت تکنولوژی و کاهش ابعاد ترانزیستورها، طراحی و ساخت تراشه‌های الکترونیکی با مشکلات جدیدی مواجه شده است. از جمله‌ی این موارد می‌توان به کاهش قابلیت اطمینان[۱] مدارهای دیجیتال اشاره کرد[2]. پدیده سالمندی[۲] ترانزیستور در اثر ناپایداری حاصل از دما و بایاس[۳] (BTI)، یکی از مهم‌ترین عواملی است که قابلیت اطمینان سیستم را به خطر می‌اندازد[۳و۴].

پدیده BTI شامل دو مرحله استرس و بازیابی است. همان‌طور که در شکل (۱) مشخص شده در مرحله استرس و هنگامی که ترانزیستور فعال است، چگالی جریان[۴] در طول زمان کاهش یافته و منجر به افزایش ولتاژ آستانه ترانزیستور می‌شود. از سوی دیگر در مرحله بازیابی و هنگامی که ترانزیستور خاموش است، اثرات ناشی از مرحله استرس کاهش می‌یابد. با این

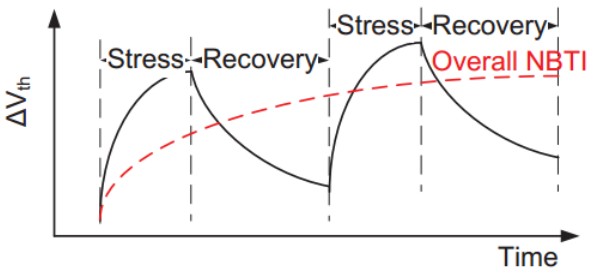
وجود، نکته مهمی که باید به آن توجه کرد این است که همانطور که در شکل (۱) قابل مشاهده است، مرحله بازیابی نمی‌تواند به صورت کامل اثرات مرحله استرس را خنثی کند و در طولانی مدت ولتاژ آستانه ترانزیستور افزایش می‌یابد. در نتیجه، پدیده BTI موجب افزایش ولتاژ آستانه و همچنین افزایش تاخیر مدارهای ترکیبی شده و به طور قابل ملاحظه‌ای عملکرد و کارآیی مدار را در طول دوره کارکرد آن تحت تاثیر قرار می‌دهد و در نهایت می‌تواند منجر به نقض محدودیت‌های زمانی مدار شود[۵]. با توجه به موارد ذکر شده می‌توان نتیجه گرفت که پدیده BTI می‌تواند موجب کاهش میانگین زمان تا بروز خطا[۵] (MTTF) و همچنین فرسودگی[۶] سریع‌تر سیستم شود.

اگرچه پدیده BTI می‌تواند باعث افزایش تاخیر مدارهای ترکیبی شود، اما این پدیده به طور موثری موجب کاهش حاشیه زمانی ایستا[۷] (SNM) در یک سلول حافظه SRAM شده [۶ و ۷] و تاثیر آن بر تاخیر دسترسی[۸] حافظه‌های SRAM قابل چشم‌پوشی است[۸ و ۹]. SNM یک سلول SRAM نشان‌دهنده میزان مقاومت آن سلول در مقابل نویز ولتاژی است. به عبارت دیگر، SNM نشان‌دهنده بیشینه میزان نویز ولتاژی است که می‌تواند توسط یک سلول حافظه SRAM تحمل شود بدون اینکه مقدار ذخیره شده در آن سلول تغییر یابد[۱۰].

نرخ دوره کاری[۹] (DCR) یک سلول حافظه SRAM برابر است با درصدی از طول عمر پیش‌بینی شده سلول حافظه که درآن مقدار یک ذخیره شده است. اگر سلول SRAM برای یک مدت زمان طولانی مقداری را در خود ذخیره کند، DCR نزدیک به صفر یا یک، تنزل SNM ناشی از BTI افزایش می‌یابد [۱۱]. در نتیجه، به منظور کمینه کردن تنزل SNM ناشی از BTI، میزان DCR آن سلول باید به ۰/۵ نزدیک شود [۱۲].

از آن‌جایی که حافظه نهان دستورالعمل[۱۰] (ICache) یکی از اجزا اصلی پردازنده‌های نهفته می‌باشد که از سلول‌های SRAM ساخته شده و همچنین به دلیل این‌که سلول‌های این حافظه مقدار ثابتی را برای مدت زمانی طولانی در خود ذخیره می‌کنند، تاثیر BTI بر روی حافظه‌های ICache تشدید می‌شود. بنابراین، مقاوم‌سازی حافظه‌های ICache در مقابل پدیده BTI یکی از مهم‌ترین مراحل به منظور بهبود قابلیت اطمینان پردازنده‌های نهفته می‌باشد. در این مقاله، یک روش کدگذاری مجموعه دستورالعمل[۱۱] (ISE) به منظور بهبود قابلیت اطمینان حافظه‌های ICache در مقابل پدیده BTI ارائه شده است. در روش ارائه شده، ابتدا جایگزینی‌های متوالی دستورالعمل‌ها در یک حافظه ICache بررسی شده و سپس در قالب گرافی به نام گراف جایگزینی دستورالعمل‌ها نمایش داده می‌شود. در مرحله بعد با استفاده از گراف استخراج شده، کدگذاری بهینه دستورات به گونه‌ای استخراج می‌شود که اثرات ناشی از BTI در سلول‌های حافظه ICache به کم‌ترین مقدار ممکن برسد.

در ادامه این مقاله و در بخش دوم به بررسی تاثیر BTI بر تنزل SNM در سلول‌های SRAM پرداخته شده و همچنین کارهای پیشین مورد بررسی قرار می‌گیرند. در بخش سوم روند استفاده شده به منظور استخراج تنزل SNM در حافظه‌های ICache معرفی شده و در بخش چهارم روش ISE پیشنهادی به منظور کاهش اثرات ناشی از BTI در حافظه‌های ICache توضیح داده خواهد شد. در بخش پنجم نتایج حاصل از شبیه‌سازی ارائه شده و در نهایت در بخش هفتم نتیجه‌گیری ارائه می‌شود.



شکل (۱) غلبه مرحله استرس بر مرحله بازیابی [۵]

۲- پیش‌زمینه‌ها

در این بخش ابتدا اثرات BTI بر سلول حافظه SRAM مورد بررسی قرار گرفته و سپس کارهای پیشین بررسی می‌شوند.

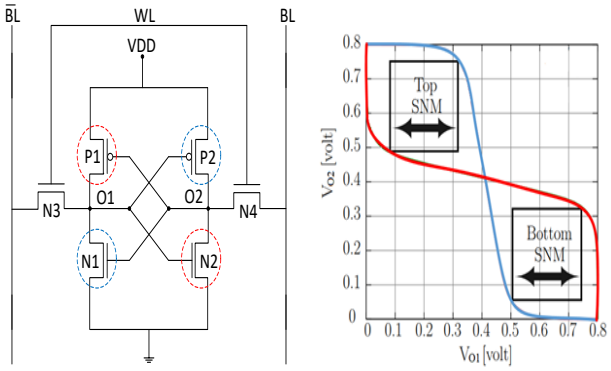
۲-۱- تاثیر BTI بر حافظه‌های SRAM

SNM یک سلول SRAM نشان‌دهنده میزان مقاومت آن سلول در مقابل نویز ولتاژی است. بسته به میزان SNM، این نویز می‌تواند مقدار ذخیره شده در سلول حافظه SRAM را تخریب کند. به عبارت دیگر هرچه میزان SNM یک سلول SRAM بیشتر باشد، مقاومت آن سلول در مقابل نویز ولتاژی افزایش یافته و نویز ولتاژی قوی‌تری نیاز است تا بتواند موجب تخریب داده ذخیره شده در سلول حافظه شود.

شکل (۲-الف) یک سلول حافظه SRAM با شش ترانزیستور را نشان می‌دهد. SNM این سلول با استفاده از نمودار پروانه‌ای که نشان‌دهنده مشخصه انتقالی یک سلول SRAM است، محاسبه می‌شود. همان‌طور که در شکل (۲-ب) مشخص شده است، SNM شامل SNM بالایی و SNM پایینی است که میزان هر کدام برابر است با طول مربعی که بین دو منحنی قرار می‌گیرد. در نهایت، SNM یک سلول SRAM برابر است با کمینه میزان SNM بالایی و SNM پایینی آن سلول:

$$SNM_{SRAM\ cell} = \min\left(SNM_{top}, SNM_{bottom}\right) \quad (۱)$$

میزان تنزل SNM ناشی از BTI در یک سلول SRAM، تابعی از DCR آن سلول است. در حالت DCR = 0، سلول مقدار صفر را در خود نگه



الف) ساختار سلول **ب) نمودار پروانه‌ای**

شکل (۲) ساختار سلول حافظه SRAM و نمودار پروانه‌ای سلول به منظور محاسبه حاشیه نویز ایستا

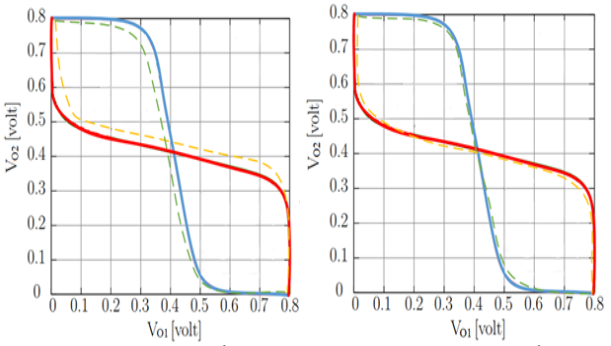
می‌دارد. در این حالت $O_1='0'$ و $O_2='1'$، ترانزیستورهای P_2 و N_1 در حالت استرس و ترانزیستورهای P_1 و N_2 در حالت بازیابی قرار می‌گیرند. بنابراین همان‌طور که در شکل (۳-الف) مشخص شده است، پدیده BTI موجب افزایش ولتاژ آستانه ترانزیستورهای P_2 و N_1 و کاهش SNM سلول SRAM شده و در نتیجه آن قابلیت اطمینان سلول کاهش می‌یابد. از سوی دیگر و در حالتی که DCR = 1 باشد، سلول مقدار یک را در خود نگه می‌دارد. در این حالت $O_1='1'$ و $O_2='0'$، ترانزیستورهای P_1 و N_2 در حالت استرس و ترانزیستورهای P_2 و N_1 در حالت بازیابی قرار می‌گیرند.

در حالت DCR = 0.5، همه‌ی ترانزیستورها به یک اندازه دچار استرس می‌شوند که در نتیجه‌ی آن همان‌طور که در شکل (۳-ب) مشخص شده تنزل SNM سلول متقارن خواهد بود. از مقایسه شکل (۳-الف) و شکل (۳-ب) می‌توان نتیجه گرفت که تنزل SNM و در نتیجه نرخ سالمندی سلول در حالت DCR = 0.5 کمتر از حالات DCR = 0 و DCR = 1 می‌باشد.

۲-۲- کارهای پیشین

تاکنون روش‌های متعددی به منظور کاهش اثرات سالمندی در حافظه‌ها ارائه شده است. روش "چرخش بیت[۱۲]" [۱۴ و ۱۳]که در آن از مدارهای شیفت چرخشی به چپ یا راست استفاده می‌شود، یکی از روش‌هایی است که به صورت گسترده برای کاهش تاثیر BTI در بایگانی ثبات[۱۳] استفاده می‌شود. اما از آن‌جایی که این روش نیازمند استفاده از شیفت‌دهنده‌ها در درگاه‌های ورودی و خروجی بایگانی ثبات می‌باشد، استفاده از این روش موجب ایجاد سربار مساحت و توان مصرفی می‌شود. همچنین کارآیی این روش هنگامی که داده ذخیره شده در ثبات شامل صفرها و یا یک‌های متوالی است، کاهش می یابد. روش معکوس کردن بیت[۱۴] [۱۷-۱۵] روش دیگری است که در آن سعی شده تا با معکوس کردن مقدار ذخیره شده در سلول SRAM به صورت متناوب، اثرات ناشی از سالمندی در حافظه کاهش یابد. این روش نیز مانند روش چرخش بیت نیازمند مدارات اضافی است که موجب ایجاد سربار مساحت و توان مصرفی در حافظه می‌شود.

کدگذاری مجموعه دستورالعمل‌ها از جمله روش‌های موثری است که در آن از طریق کاهش فعالیت سوئیچینگ[۱۵] سلول‌های SRAM به منظور کاهش توان مصرفی در حافظه‌های ICache استفاده می‌شود [۲۰-۱۸]، اما باید در نظر داشت که کاهش فعالیت سویچینگ یک سلول SRAM موجب افزایش اثرات ناشی از BTI در آن سلول می‌شود. در [۲۲ و ۲۱] یک روش



الف) DCR = 0 **ب) DCR = 0.5**

شکل (۳) تاثیر سالمندی بر روی حاشیه نیز ایستا سلول SRAM

افزایش اثرات ناشی از BTI در آن سلول می‌شود. در [۲۲ و ۲۱] یک روش کدگذاری مجموعه دستورالعمل‌ها به منظور مقابله با اثرات سالمندی ناشی از BTI در واحد دیکد پردازنده ارائه شده است. اما،از آن‌جایی که هدف این روش تنها کاهش تاثیر BTI در واحد رمزگشا[۱۶] پردازنده است، تنها بخش آپکد[۱۷] دستورالعمل‌ها مورد بررسی قرار گرفته است و سایر قسمت‌های دستورالعمل‌ها مورد بررسی قرار نگرفته است. بنابراین، از این روش نمی‌توان به منظور کاهش تاثیر BTI بر حافظه‌های ICache استفاده کرد. در [۲۳] با استفاده از تکنیک ISE سعی شده تا اثر BTI در حافظه‌های ICache بهینه شود. به این منظور، یک الگوریتم بهینه‌سازی مبتنی بر شبیه‌سازی تبرید[۱۸] ارائه شده است که در هر مرحله از آن کدی با کمترین میزان تنزل ناشی از BTI انتخاب می‌شود. نکته قابل توجه این است که در این روش به دلیل این‌که در هر مرحله از فرآیند بهینه‌سازی به صورت تصادفی کد باینری دستورات با یکدیگر جابجا می‌شود، اجرای الگوریتم بسیار زمان‌بر خواهد بود.

۳- بررسی تنزل SNM ناشی از سالمندی در حافظه ICache

قبل از تشریح روش پیشنهادی، در این بخش روند استفاده شده به منظور تخمین تنزل SNM ناشی از سالمندی در حافظه‌های ICache معرفی می‌شود. به این منظور، در ابتدا با استفاده از شبیه‌سازی HSPICE برای هر ترانزیستور موجود در حافظه SRAM، تنزل ولتاژ آستانه ناشی از BTI به ازای DCRهای مختلف محاسبه می‌گردد. سپس با استفاده از نتایج بدست آمده و همچنین با استفاده از شبیه‌سازی HSPICE، میزان SNM سلول SRAM به ازای DCRهای مختلف محاسبه شده و در جدولی ذخیره می‌شود.

از سوی دیگر با استفاده از یک شبیه‌ساز سطح معماری مانند gem5 [۲۴]، به ازای بارهای کاری مختلف از مجموعه محک SPEC CPU2006 [۲۵]، ردیابی دستورات در حافظه ICache استخراج شده و DCR هر سلول حافظه محاسبه می گردد. سپس، DCR هر سطر از حافظه از رابطه (۲) محاسبه می گردد:

$$DCR_i = \max_{1\le j\le m}\left(DCR_{ij}\right) \qquad (۲)$$

در این رابطه، m نشان‌دهنده تعداد ستون‌ها و DCR_i و DCR_{ij} به ترتیب نشان‌دهنده DCR سطر i و DCR سلول j ام از سطر i می‌باشند و سرانجام DCR حافظه ICache از رابطه (۳) محاسبه می‌شود:

$$DCR_{ICache} = \max_{1\le i\le n}\left(DCR_i\right) \qquad (۳)$$

در این رابطه، n نشان‌دهنده تعداد سطرها می‌باشد. در نهایت، DCR حافظه ICache از طریق درون‌یابی نقاط موجود در جدول SNM که به DCR_{ICache} استخراج شده از رابطه (۳) نزدیک هستند، استخراج می‌گردد.

٤- کدگذاری مجموعه دستورالعمل آگاه از سالمندی

در این مقاله، بدون نقض کلیت و به منظور توضیح روش کدگذاری مجموعه دستورالعمل‌ها از معماری مجموعه دستورالعمل[۱۹] ARM (ISA) استفاده

شده است، اما روش پیشنهادی کلی بوده و برای سایر ISAها نیز قابل استفاده

قسمت‌های مختلف یک دستور را می‌توان به دو دسته کلی بخش‌های وابسته به کدگذاری، مانند آپکد و کد عملکرد[۲۰]، و بخش‌های مستقل از کدگذاری، مانند بخش بی واسطه[۲۱]، تقسیم کرد. مانند بسیاری از ISAها، هر دستور در معماری ARM دارای سه بخش وابسته به کدگذاری به نام‌های آپکد، کد عملکرد و همچنین بخش ثبات می‌باشد. بخش‌های آپکد و ثبات به ترتیب مشخص کننده عمل اصلی دستورالعمل و ثبات مورد استفاده دستور را می‌باشند. کد عملکرد نیز مشخص می‌کند که از میان صور گوناگون دستور مشخص شده در آپکد، کدام یک باید اجرا شود. بنابراین، مسئله کدگذاری مجموعه دستورالعمل‌ها را می‌توان به سه مسئله بهینه‌سازی یافتن کدگذاری آگاه از سالمندی بخش آپکد، بخش کد عملکرد و همچنین بخش ثبات تقسیم‌بندی کرد. حل مسئله یافتن کدگذاری آگاه از سالمندی برای بخش آپکد و همچنین بخش کد عملکرد یکسان است. اما، در مسئله یافتن کدگذاری بهینه بخش ثبات باید به این نکته توجه کرد که در قالب هر دستورالعمل چندین بخش ثبات مانند بخش ثبات عملوند مبداء و بخش ثبات عملوند مقصد قرار دارد.

بخش‌های ثبات یک دستورالعمل، کد باینری ثبات‌های قابل مشاهده توسط برنامه‌نویس را ذخیره می‌کند. اگر هریک از این بخش‌ها به صورت مجزا بهینه‌سازی شوند، یک ثبات یکسان ممکن است به شکل‌های گوناگونی کدگذاری شود که می‌تواند روند اجرای دستورات را تغییر داده و منجر به خروجی نادرست برنامه شود. در نتیجه، برای حل این مشکل در حل مسئله یافتن کدگذاری بخش ثبات باید تمامی بخش‌های ثبات یک دستورالعمل را در نظر گرفت.

در ادامه، از واژه "کد" به جای آپکد، کد عملکرد و همچنین کد ثبات استفاده شده است. M دستورالعمل را در نظر بگیرید که هر دستورالعمل i دارای کد E_i می‌باشد. هدف نهایی الگوریتم، یافتن مجموعه‌ای از کدهای جدید E به گونه‌ای است که اولاً تنزل SNM در حافظه ICache کمینه شود و ثانیاً هیچ دو دستوری دارای کد یکسان نباشند:

$$\forall_{i,j\in M|i\neq j}\ E_i \neq E_j \qquad (۴)$$

کد E_i به عنوان یک کد آگاه از سالمندی انتخاب می‌شود، اگر وتنها اگر، DCR بیت‌های مربوط به این کد در حافظه ICache به ۰/۵ نزدیک باشد. از آن‌جایی که با افزایش فعالیت سوئیچینگ هر بیت احتمال نزدیک شدن DCR آن بیت به ۰/۵ افزایش می‌یابد، در الگوریتم پیشنهادی سعی شده تا فعالیت سوئیچینگ هر بیت افزایش یابد. بنابراین، تابع هدف الگوریتم پیشنهادی را می‌توان به صورت رابطه (۵) بیان کرد:

$$\max\left(Switching\ Activity_{total}\right)= \max\left(\sum_{\forall\left(I_i,I_j\right)\in M} R_{i,j}\ HD\left(I_i,I_j\right)\right) \qquad (۵)$$

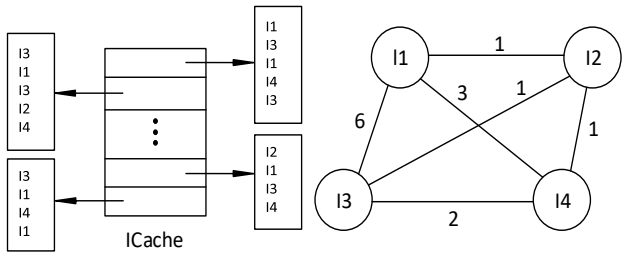
که در این رابطه، $Switching\ Activity_{total}$ نشان‌دهنده فعالیت سوئیچینگ حافظه ICache، I_i مشخص کننده دستورالعمل i می‌باشد که $\left(I_i,I_j\right)=\left(I_j,I_i\right)$ و $R_{i,j}$ نشان دهنده تعداد دفعاتی است که دستور I_i با دستور I_j و یا دستور I_j با دستور I_i جایگزین شده است. همچنین، $HD\left(I_i,I_j\right)$ فاصله همینگ بین دستورات I_i و I_j را نشان می‌دهد.

روش پیشنهادی شامل دو مرحله است. در مرحله اول، ابتدا جایگزینی‌های متوالی دستورالعمل‌ها در یک حافظه ICache بررسی شده و سپس در قالب گرافی به نام گراف جایگزینی دستورالعمل‌ها نمایش داده می‌شود. در مرحله بعد با استفاده از گراف استخراج شده، کدگذاری آگاه از سالمندی دستورات به گونه‌ای استخراج می‌شود که اثرات ناشی از BTI در سلول‌های حافظه ICache به کم‌ترین مقدار ممکن برسد.

۴-۱- گراف جایگزینی دستورالعمل

مسئله پیدا کردن بهترین کدگذاری دستورات از جمله مسائل ان‌پی سخت[۲۲] به شمار می‌رود[۲۶]. بنابراین، پیدا کردن بهترین کدگذاری مجموعه دستورالعمل‌ها از طریق ایجاد کردن تمامی حالات ممکن در مدت زمان معقول غیر ممکن است. در این مقاله به منظور حل این مسئله در مدت زمان قابل قبول، از یک روش مبتنی بر گراف استفاده می‌کنیم. با استفاده از گراف جایگزینی دستورالعمل می‌توان جایگزینی بین دستورات را استخراج کرد و با استفاده از آن کدگذاری مجموعه دستورالعمل‌ها را به گونه‌ای انجام داد که فعالیت سوئیچینگ سلول‌های SRAM موجود در حافظه ICache افزایش یابد.

گراف جایگزینی دستورالعمل، یک گراف وزن‌دار است که به صورت G(V, E) با مجموعه گره‌های V و مجموعه یال‌های E تعریف می‌شود. در این گراف هر گره $v_i \in V$ نشان‌دهنده یک کد مشخص و یال $e\left(v_i,v_j\right)\in E$ نشان‌دهنده تعداد جایگزینی‌های گره v_i با گره v_j و بالعکس است. شکل (۴-الف) نشان دهنده جایگزینی دستورات در یک حافظه ICache را نشان می‌دهد. شکل (۴-ب) نشان‌دهنده گراف جایگزینی دستورالعمل استخراج شده از شکل (۴-الف) می‌باشد.



شکل (٤) الف) جایگزینی دستورات در یک حافظه ICache
ب) گراف جایگزینی دستورالعمل

۴-۲- تخصیص کد آگاه از سالمندی

الگوریتم (۱)، شبه کد الگوریتم تخصیص کد آگاه از سالمندی را نشان می‌دهد. در این الگوریتم، مجموعه X نشان‌دهنده گره‌هایی از V است که به آن‌ها کد جدیدی تخصیص داده شده است. در الگوریتم پیشنهادی، ابتدا تعداد جایگزینی‌های هر گره v_i از طریق رابطه (۶) محاسبه می‌شود:

$$Total\_Weight\left(v_i\right)= \sum_{\forall v_j \in Graph} e\left(v_i,v_j\right) \qquad (۶)$$

Algorithm 1. Aging-aware Code Assignment

1. Inputs : instruction replacement graph $G(V, E)$
2. Output : Optimized ISE

---

3. X : *Set of nodes that assigned new code*
4. *Available_Codes : Set of available codes that has not been assigned to any node*
5. **for each** $node_{v_i}$ **in** *Graph G* **do**
6. **Compute** $Total\_Weight(v_i)$
7. **end for**
8. **Add** $node_{v_i}$ **with max** $Total\_Weight$ **to** X
9. **for each** $node_{v_i}$ **in** *Graph G* **do**
10. **if** ($node_{v_i}$ *is not in* X)
11. $Total\_Weight\_X(v_i)$
12. **end if**
13. **end for**
14. **Select** $node_{v_i}$ **with max** $Total\_Weight\_X$
15. **for each** $mapping\ code_i$ **in** *Available_Codes* **do**
16. $Switching\ Activity(v_i, code_i)$
17. **end for**
18. **select** $code_i$ **with max** $Switching\ Activity$
19. **Change** $node_{v_i}$ **code to** $code_i$
20. **Add** $node_{v_i}$ **to** X
21. **if(size(X) != size(v))**
22. **goto line 9**

گره v_i با بیشترین میزان $Total\_Weight$ انتخاب شده و به مجموعه X اضافه می‌شود. سپس برای هر گره v_i که عضوی از X نمی‌باشد، تعداد جایگزینی‌های آن با هریک از گره‌های موجود در X از طریق رابطه (۷) محاسبه می شود:

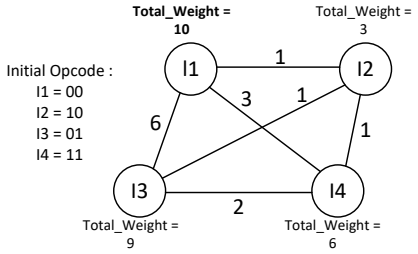
$$Total\_Weight\_X(v_i) = \sum_{\forall v_j \in X} e(v_i, v_j) \qquad (۷)$$

در ادامه، گره‌ای با بیشترین میزان $Total\_Weight\_X$ برای تخصیص کد جدید انتخاب می‌شود. به منظور تخصیص کد جدید به ازای تمامی نگاشت‌های ممکن از میان کدهای باینری موجود، کدی انتخاب می‌شود که فعالیت سوئیچینگ گره را نسبت به گره‌های موجود در X به حداکثر برساند:

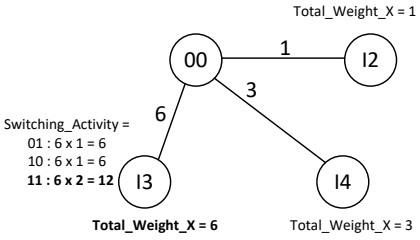
$$Switching\ Activity(v_i, code_i) = \sum_{\forall v_j \in X} e(v_i, v_j)\, HD(code_i, v_j) \qquad (۸)$$

در این رابطه $Switching\ Activity(v_i, code_j)$ نشان‌دهنده میزان فعالیت سوئیچینگ گره v_i به ازای نگاشت $code_j$ را نشان می‌دهد. در این مرحله کد $code_j$ که دارای بیشترین مقدار $Switching\ Activity(v_i, code_j)$ است به عنوان کد جدید v_i انتخاب شده و به مجموعه X اضافه می‌شود. این فرآیند تا زمانی ادامه می‌یابد که به همه‌ی گره‌ها کد جدیدی تخصیص یابد.

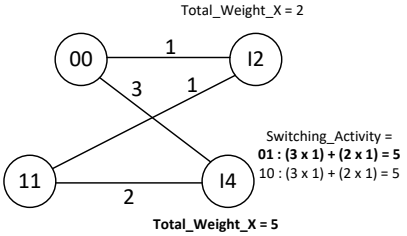
شکل (۶) نتایج حاصل از هر مرحله از الگوریتم پیشنهادی را بر روی گراف شکل (۴–ب) نشان می‌دهد. همان‌طور که در شکل مشخص است، در مرحله اول از اجرای الگوریتم (شکل (۶–الف))، بدلیل اینکه گره I1 دارای بیشترین تعداد جایگزینی می‌باشد (بیشترین میزان Total_Weight)، این گره برای تخصیص کد انتخاب شده و کد "00" به آن تخصیص داده می‌شود. در مرحله دوم (شکل (۶–ب))، تعداد جایگزینی هر دستور با دستور I1 محاسبه شده (Total_Weight_X) و سپس بدلیل اینکه دستور I3 دارای بیشترین میزان جایگزینی با دستور I1 است برای تخصیص کد انتخاب می‌شود. در نهایت بدلیل اینکه کد "11" باعث ایجاد بیشترین میزان Switching Activity می‌شود، کد "11" به این گره تخصیص داده می‌شود. در مرحله سوم (شکل (۶–ج))، تعداد جایگزینی‌های هر دستور با دستورات I1 و I3 محاسبه شده و در نهایت گره I4 به منظور تخصیص کد انتخاب شده و کد "01" به آن تخصیص داده می‌شود. در مرحله آخر از اجرای الگوریتم نیز کد "10" به گره I2 تخیصی داده می‌شود (شکل (۶–د)).



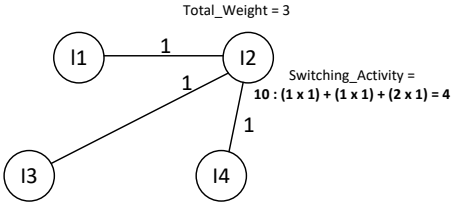
الف) مرحله اول اجرای الگوریتم



ب) مرحله اول اجرای الگوریتم

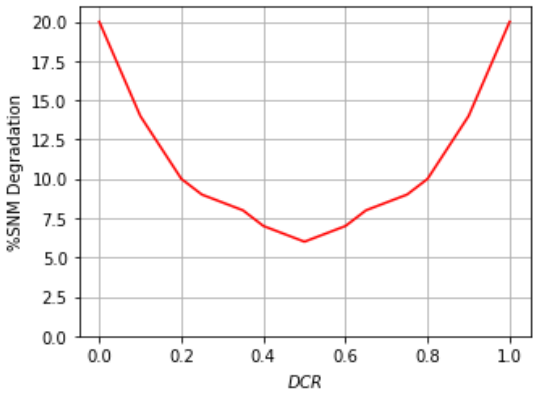


ج) مرحله سوم اجرای الگوریتم



د) مرحله چهارم اجرای الگوریتم

شکل (٦) نتایج حاصل از اجرای هر مرحله از الگوریتم تخصیص کد آگاه از سالمندی بر روی گراف شکل (٤–ب)



شکل (۷) تنزل SNM ناشی از سالمندی در یک سلول حافظه SRAM به ازای مقادیر مختلف DCR

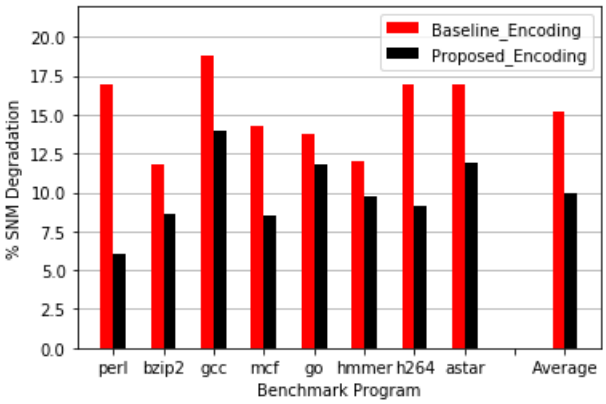
۵- نتایج شبیه‌سازی

روش پیشنهادی با استفاده از برنامه‌های مجموعه محک SPEC CPU2006 بر روی یک پردازنده ARM مورد ارزیابی قرار گرفته است.

در ابتدا، با استفاده از شبیه‌سازی‌های HSPICE معرفی شده در بخش ۳، جدول میزان SNM براساس مقادیر مختلف DCR استخراج می‌شود. شکل (۷)، تنزل SNM یک سلول SRAM به ازای مقادیر مختلف DCR را بعد از گذشت ۱۰ سال نشان می‌دهد. همان‌طور که در شکل قابل مشاهده است، با نزدیک شدن DCR به ۰/۵ میزان تنزل SNM سلول SRAM کاهش می‌یابد. سپس با استفاده از شبیه‌ساز سطح معماری gem5، ردیابی دستورات در حافظه ICache استخراج می‌شود و در نهایت با استفاده از روش معرفی شده در بخش ۳، SNM حافظه ICache محاسبه می‌گردد.

به منظور ارزیابی روش پیشنهادی، بارهای کاری موجود در مجموعه محک SPEC CPU2006 به دو مجموعه آموزش و ارزیابی تقسیم شده‌اند. چهار بارکاری از مجموعه محک به عنوان مجموعه آموزش برای آموزش الگوریتم پیشنهادی استفاده شده و سایر بارهای کاری موجود در مجموعه محک به عنوان مجموعه ارزیابی با هدف بررسی میزان کارآیی روش پیشنهادی مورد استفاده قرار گرفته‌اند. در ابتدا، بارهای کاری موجود در مجموعه آموزشی با استفاده از شبیه‌ساز gem5 اجرا شده و ردیابی دستورات مربوط به آن‌ها استخراج می‌شوند. سپس با استفاده از نتایج حاصل از ردیابی دستورات، گراف جایگزینی دستورالعمل استخراج می‌شود. در مرحله بعد، روش ارائه شده در بخش ۴-۲ بر روی گراف جایگزینی دستورالعمل اعمال شده و کدگذاری جدید دستورات استخراج می‌شود.

به منظور ارزیابی روش پیشنهادی، کدگذاری جدید دستورات برروی بارهای کاری موجود در مجموعه محک مجموعه ارزیابی اعمال شده و SNM مربوط به حافظه ICache محاسبه می گردد. شکل (۸) میزان تنزل SNM ناشی از BTI در حافظه ICache را به‌ازای کدگذاری اولیه و همچنین کدگذاری آگاه از سالمندی، برای هریک از بارهای کاری موجود در مجموعه ارزیابی نشان می‌دهد. همان‌طور که در شکل قابل مشاهده است، روش پیشنهادی به طور میانگین تنزل SNM ناشی از BTI را حدود ۳۴/۳۵٪ بهبود می‌دهد. همچنین برای بسیاری از بارهای کاری موجود در



شکل(۸) تنزل SNM ناشی از سالمندی در حافظه نهان دستورالعمل

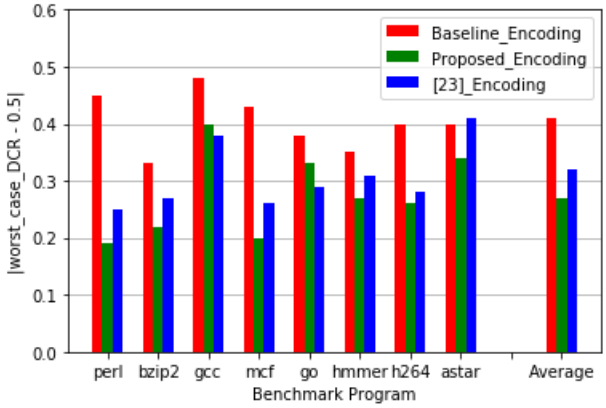
مجموعه ارزیابی، تنزل SNM ناشی از سالمندی به طور چشمگیری بهبود یافته است.

شکل (۹) میزان DCR حافظه ICache را به ازای کدگذاری اولیه، کدگذاری آگاه از سالمندی روش پیشنهادی و همچنین کدگذاری استخراج شده از روش ارائه شده در [۲۳] را نشان می‌دهد. همان‌طور که در شکل قابل ملاحظه است، روش پیشنهادی به طور چشمگیری DCR حافظه ICache را بهبود می‌دهد. همچنین برای بسیاری از بارهای کاری موجود در مجموعه ارزیابی، DCR بدست آمده از روش پیشنهادی در مقایسه با روش ارائه شده در [۲۳]، به ۰/۵ نزدیکتر است.

باید توجه داشت که از آن‌جایی که روش کدگذاری مجموعه دستورالعمل‌ها یک روش زمان طراحی۲۳ است. سربار مساحت ناشی از روش پیشنهادی محدود به سربار ناشی از پیاده‌سازی جدید واحد دیکد پردازنده است و همان‌طور که در [۲۲] نشان داده شده، سربار مساحت ناشی از این روش در مقایسه با روش‌های پیشین بسیار اندک است.

٦- نتیجه گیری

با کاهش روزافزون ابعاد ترانزیستورها، سالمندی و تنزل تدریجی خصوصیات ترانزیستورها که باعث کاهش کارآیی و تخریب عملکرد مدار می‌شود، به یک چالش اساسی در زمینه قابلیت اطمینان مدارهای مجتمع تبدیل شده است. در سلول‌های حافظه SRAM پدیده BTI به طور موثری موجب کاهش حاشیه زمانی ایستا در یک سلول حافظه SRAM می‌شود. در این



شکل (۹) میزان DCR حافظه نهان دستورالعمل به ازای کدگذاری اولیه، روش پیشنهادی و همچنین کدگذاری استخراج شده از [۲۳]

مقاله یک روش کدگذاری مجموعه دستورالعمل به منظور بهبود تنزل SNM ناشی از سالمندی در حافظه نهان دستورالعمل ارائه شد. نتایج حاصل از شبیه-سازی نشان می‌دهد که روش ارائه شده می‌تواند تنزل SNM در یک حافظه نهان دستورالعمل را حدود ۳۴/۳۵٪ بهبود دهد.

مراجع

[1] J. Fang, S. Gupta, S.V. Kumar, S.K. Marella, V. Mishra, P. Zhou, and S.S. Sapatnekar, "*Circuit reliability: from physics to architectures"*, IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 243–246, 2012.

[2] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. When, "*Reliable on-chip systems in the nano-era: lessons learnt and future trends*", In Design Automation Conference (DAC), pp. 1–10, 2013.

[3] W. Wang, S. Yang, S. Bhardwaj, R. Vattikonda, S. Vrudhula, F. Liu, and Y. Cao, "*The impact of NBTI on the performance of combinational and sequential circuits",* In Design Automation Conference (DAC), pp. 364–369 , 2007

[4] Plamondon, R., Lorette, G., *"Automatic Signature Verification and Writer Identification - The State of the Art"*, Pattern Recognition, Vol. 22, pp. 107-131, 1989.

[5] W. Wang, S. Yang, S. Bhardwaj, R. Vattikonda, S. Vrudhula, F. Liu, and Y. Cao, "*The impact of NBTI on the performance of combinational and sequential circuits",* In Design Automation Conference (DAC), pp. 364–369 , 2007

[6] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel, "*Reliability-aware design to suppress aging*", in Design Automation Conference (DAC), 2016.

[7] S. Arasu, M. Nourani, J. M. Carulli, and V. K. Reddy, "*Controlling aging in timing-critical paths*", IEEE Design and Test, vol. 33, no. 4, pp. 82–91, 2016.

[8] S. Khan et al., "*Bias temperature instability analysis of FinFET based SRAM cells*," in Proc. Design, Autom. Test Eur. Conf. Exhibit., 2014, pp. 1–6

[9] M. Namaki-Shoushtari, A. Rahimi, N. Dutt, P. Gupta, and R. K. Gupta, "*ARGO: Aging-aware GPGPU register file allocation*," in Proc. 9th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth., Oct. 2013, Art. no. 30

[10] N. Rohbani, M. Ebrahimi, S.G. Miremadi, and M.B. Tahoori, "*Bias Temperature Instability Mitigation via Adaptive Cache Size Management*", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2016.

[11] Kunitake et al. "*Signal probability control for relieving nbti in sram cells*". In International Symposium on Quality Electronic Design (ISQED), 2010.

[12] Kumar et al. "*Impact of nbti on sram read stability and design for reliability*". In International Symposium on Quality Electronic Design (ISQED), 2006.

[13] H. Amrouch, T. Ebi and J. Henkel, "*Stress Balancing to Mitigate NBTI Effects in Register Files*," in Dependable Systems and Networks (DSN), 2013.

[14] S. Wang, T. Jin, C. Zheng and G. Duan, "*Low power aging-aware register file design by duty cycle balancing*," in Design, Automation and Test in Europe (DATE), 2012.

[15] T. Siddiqua and S. Gurumurthi, "*Recovery boosting: A technique to enhance NBTI recovery in SRAM arrays*," in Annual Symposium on VLSI, 2010.

[16] S. Kothawade, K. Chakraborty and S. Roy, "*Analysis and mitigation of NBTI aging in register file: An end-to-end approach*," in International Symposium on Quality Electronic Design (ISQED), 2011.

[17] H. Amrouch, T. Ebi and J. Henkel, "*Stress Balancing to Mitigate NBTI Effects in Register Files*," in Dependable Systems and Networks (DSN), 2013.

[18] Helkala et al. "*Variable length instruction compression on transport triggered architectures*". In Embedded Computer Systems, 2014.

[19] Chattopadhyay et al." *Power-efficient instruction encoding optimization for embedded processors*". In VLSI Design, 2007.

[20] Guo et al. Shifted "*gray encoding to reduce instruction memory address bus switching for low-power embedded systems*". Journal of Systems Architecture, 2010.

[21] Oboril et al. "*Arise: Aging-aware instruction set encoding for lifetime improvement*". In ASP-DAC, 2014

[22] F. Oboril, and M.B. Tahoori, "*Exploiting Instruction Set Encoding for Aging-Aware Microprocessor Design*", ACM Trans. Des. Autom. Electron. Syst. 21, 1, Article 5 (November 2015), 26 pages.

[23] A. Gebregiorgis, F. Oboril, M.B. Tahoori, and S. Hamdioui, "*Instruction Cache Aging Mitigation Through Instruction Set Encoding*", International Symposium on Quality Electronic Design (ISQED) 2016.

[24] Binkert N.; at al. "*The gem5 simulator*". ACM SIGARCH Computer Architecture News, vol.39 (2), 2011. www.gem5.org

[25] J. Henning. "*Performance Counters and Development of SPEC CPU2006*". Computer Architecture News. March 2007

[26] Kim et al. "*Opcode encoding for low-power instruction fetch"*. Electronics Letters Vol. 35, Issue. 13, 1999.

پانویس ها

[1] Reliability
[2] Aging
[3] Negative Bias Temperature Instability
[4] Current Density
[5] Mean Time To Failure
[6] Wear out
[7] Static Noise Margin
[8] Access Latency
[9] Duty Cycle Ratio
[10] Instruction Cache
[11] Instruction Set Encoding
[12] Bit Rotation
[13] Register File
[14] Bit Flipping
[15] Switching Activity
[16] Decode
[17] Opcode
[18] Simulated Annealing
[19] Instruction Set Architecture
[20] Function Code
[21] Immediate Field
[22] NP Hard
[23] Design-Time Technique