





دانشگاه گیلان

دانشکده فنی

پایان نامه کارشناسی ارشد

پیاده سازی معماری خودتعمیر برای حافظه های جاسازی

شده بر اساس استاندارد آزمون هسته ها

از:

پیام حبیبی

استاد راهنما:

دکتر راهبه نیارکی اصلی

اسفند ۱۳۹۰

دانشکده فنی

گروه مهندسی برق

گرایش الکترونیک

پیاده‌سازی معماری خودتعمیر برای حافظه‌های جاسازی شده بر
اساس استاندارد آزمون هسته‌ها

از:

پیام حبیبی

استاد راهنما:

دکتر راهبه نیارکی اصلی

اسفند ۱۳۹۰

تقدیم به:

پدر و مادر عزیز و همسر مهربانم

تشکر و قدردانی

از استاد راهنمای ارجمندم سرکار خانم دکتر راهبه نیارکی اصلی به خاطر زحمات فراوان و رهنمودهای ارزنده-
شان در انجام این پایان نامه و در تمام طول دوران تحصیل کمال تشکر و امتنان را دارم. حضور ایشان همواره
روشنی بخش مسیر تحقیق در این پژوهش بوده است.

از خانواده‌ام به ویژه پدر و مادر و برادر عزیزم که در تمام مراحل زندگی همواره مشوق و پشتیبان من بودند
سپاسگذارم و همچنین از همسر صبور و دلسوزم که در طول این پروژه زحمات و سختی‌های فراوانی را متحمل
شدند بسیار متشکرم.

پیام حبیبی

اسفند ۱۳۹۰

چکیده

پیاده‌سازی معماری خودتعمیر برای حافظه‌های جاسازی شده بر اساس استاندارد آزمون هسته‌ها

پیام حبیبی

بیشترین تعداد هسته‌های موجود در سیستم‌های روی تراشه (SOC) را حافظه‌های جاسازی شده تشکیل می‌دهند. از این رو بهره‌دهی این حافظه‌ها نقش بسزایی در بهره‌دهی کل SOC دارد. پیشرفت مداوم مجتمع‌سازی در سطح تراشه‌ها و در نتیجه افزایش چگالی حافظه‌های جاسازی‌شده، امکان بروز نقص در سلول‌های حافظه را افزایش داده است که این امر به کاهش بهره‌دهی کل تراشه منجر می‌گردد. با تعمیر حافظه‌های جاسازی‌شده می‌توان این مشکل را تا حد زیادی برطرف نمود. در صورتی که نقص پدید آمده در حافظه از نوع خطاهای سخت باشد می‌توان با جایگزین کردن سلول‌های معیوب با سلول‌های سالم که به‌صورت افزونه در کنار حافظه قرار می‌گیرند عملیات تعمیر را انجام داد.

روش‌های قدیمی تعمیر، بر استفاده از ابزارهای آزمون خودکار (ATE) و الگوریتم‌های خارج از تراشه استوار بودند. اما امروزه تعداد و پیچیدگی هسته‌های SOCها افزایش یافته و به علت تعداد زیاد I/Oهای هسته‌های حافظه، دسترسی به همه‌ی آنها از طریق I/Oهای SOC دشوارتر شده است. به منظور حل این مشکل‌ها که مشاهده‌پذیری و کنترل‌پذیری نامیده می‌شوند و همچنین به علت نیاز به انجام آزمون و تعمیر با سرعت، روش‌های گوناگون خودآزمون درون‌ساخته (BIST) و خود-تعمیری درون‌ساخته (BISR) به حافظه‌های جاسازی‌شده اعمال شدند. مدار BIST بر اساس استاندارد آزمون هسته‌ها و با استفاده از الگوریتم‌های ویژه، حافظه را می‌آزماید و آدرس محل‌های وقوع خطا را آشکار می‌سازد. تحلیلگر افزونه‌ی درون-ساخته (BIRA) که یکی از عناصر کلیدی BISR است، اطلاعات خطای ارسال شده از سوی BIST را دریافت نموده و بر اساس یک الگوریتم تحلیل، چگونگی تخصیص افزونه‌ها را تعیین می‌کند. پس از این مرحله یک پروسه‌ی پیکربندی مجدد برای غیرفعال‌سازی سلول‌های معیوب حافظه انجام می‌گیرد.

الگوریتم‌های مختلف BIRA با سه چالش مهم روبرو هستند: سطح اشغالی تراشه، نرخ تعمیر و سرعت تحلیل. در این پایان‌نامه یک مدار BIRA جدید با نرخ تعمیر بهینه با استفاده از زیرتحلیلگرهای موازی طراحی و پیاده‌سازی شده است. یکی از مشکلات اساسی روش‌های تحلیل موازی موجود این است که فضای زیادی را از سطح تراشه اشغال می‌کنند. در روش پیشنهادی تلاش شده است که این ضعف بهبود یابد. مدار طراحی شده نسبت به تحلیلگر موازی R-CRESTA ۵۰ درصد فضای کمتری از سطح تراشه اشغال می‌نماید. این برتری با پذیرش هزینه‌ی حداکثر یک تکرار آزمون به دست آمده است. با این وجود تحلیلگر پیشنهادی همچنان نسبت به تحلیلگرهای ESP و IntelligentSolveFirst از سرعت بالاتری برخوردار است.

واژه‌های کلیدی:

Embedded Memory, SOC, Built-in Self-Test, Built-in Self-Repair, Built-in Redundancy Analyzer

فهرست مطالب

چکیده

د

Abstract

ذ

فصل اول: مقدمه

۱

- ۱-۱- مقدمه..... ۲
- ۲-۱- مفاهیم اولیه..... ۵
- ۱-۲-۱- انواع خطاها..... ۵
- ۲-۲-۱- تحمل خطا..... ۶
- ۳-۲-۱- بهره‌دهی..... ۸
- ۴-۲-۱- تعمیر سخت و تعمیر نرم..... ۱۰
- ۵-۲-۱- ساختار حافظه..... ۱۱
- ۳-۱- هدف و ساختار پایان‌نامه..... ۱۲

فصل دوم: تعمیر حافظه‌ها

۱۴

- ۱-۲- تعمیر حافظه با استفاده از افزونه‌ها..... ۱۵
- ۱-۱-۲- جایگزینی فقط با ردیف..... ۱۶
- ۲-۱-۲- جایگزینی فقط با ستون..... ۱۷
- ۳-۱-۲- جایگزینی با ردیف و ستون..... ۱۸
- ۲-۲- خودتعمیری حافظه‌های جاسازی شده..... ۱۹
- ۲-۲-۲- استاندارد IEEE 1500..... ۲۰
- ۳-۲-۲- مسئله‌ی تحلیل افزونه‌های درون ساخته..... ۲۲
- ۴-۲-۲- پیکربندی مجدد..... ۲۴
- ۳-۲- بررسی روش‌های تحلیل افزونه‌ها..... ۲۶

فصل سوم: طراحی مدار درون ساخته‌ی تحلیلگر افزونه

۳۶

- ۱-۳- مقدمه..... ۳۷
- ۲-۳- معرفی نمونه حافظه‌ی تحت بررسی..... ۳۷
- ۳-۳- معرفی سخت‌افزار مبنا جهت تعمیر نمونه‌ی تحت بررسی..... ۴۰
- ۴-۳- راهکار پیشنهادی جهت کاهش فضای اشغالی..... ۴۴
- ۵-۳- ساختار سلول زیرتحلیلگر..... ۴۸
- ۶-۳- طراحی زیر تحلیلگرها..... ۵۰
- ۷-۳- تحلیلگر نهایی..... ۵۲
- ۸-۳- انتخابگر راه‌حل تعمیر..... ۵۴

فصل چهارم: شبیه سازی و پیاده‌سازی تحلیلگر پیشنهادی و تفسیر نتایج

۵۵

- ۱-۴- مقدمه..... ۵۶
- ۲-۴- نتایج شبیه‌سازی..... ۵۶
- ۳-۴- نتایج پیاده‌سازی..... ۵۹

۶۲	۴-۴- مقایسه با روش‌های دیگر
۶۲	۴-۴-۱- فضای اشغالی
۶۶	۴-۴-۲- سرعت تحلیل
۶۸	۴-۴-۳- نرخ تعمیر
۶۸	۴-۴-۴- مقایسه‌ی نهایی

فصل پنجم: جمع بندی و پیشنهادات

۷۰	
۷۱	۵-۱- جمع بندی و نتیجه‌گیری
۷۲	۵-۱-۱- پیشنهادات

فصل ششم: مراجع

۷۳

فهرست شکل‌ها

- شکل (۱-۱) مثالی از یک سیستم روی تراشه [۲]..... ۲
- شکل (۲-۱) نمودار رشد فضای اشغالی حافظه نسبت به کل فضای SOC [۳]..... ۳
- شکل (۳-۱) خطاهای سخت و نرم و راه‌حل اصلاح آن [۲۰]..... ۶
- شکل (۴-۱) الف) جایگزینی سطح بالا و ب) سطح پایین..... ۷
- شکل (۵-۱) بهبود بهره‌دهی با استفاده از جایگزینی بر اساس توزیع پواسن [۲۰]..... ۹
- شکل (۶-۱) ساختار آرایه‌ی حافظه و مدارات جانبی [۲۸]..... ۱۲
- شکل (۱-۲) جایگزینی با ردیف [۳۶]..... ۱۶
- شکل (۲-۲) جایگزینی با I/O [۳۷]..... ۱۷
- شکل (۳-۲) جایگزینی با ستون [۳۷]..... ۱۸
- شکل (۴-۲) ساختار متعارف یک حافظه‌ی خودتعمیر..... ۱۹
- شکل (۵-۲) حافظه‌ی ۸×۸ تعمیرشده توسط BISR [۳۹]..... ۲۰
- شکل (۶-۲) ساختار یک سیستم با استاندارد IEEE 1500 [۴۰]..... ۲۱
- شکل (۷-۲) گراف دو قسمتی معادل خطاهای حافظه..... ۲۳
- شکل (۸-۲) جایگزینی با استفاده از برنامه‌ریزی دیکدر و حذف به روش مستقیم [۲۰]..... ۲۵
- شکل (۹-۲) جایگزینی به روش مقایسه و حذف به روش غیرمستقیم [۲۰]..... ۲۵
- شکل (۱۰-۲) هدایت مجدد با استفاده از فیوز نرم [۴۳]..... ۲۶
- شکل (۱۱-۲) جمع‌آوری خطا در ESP [۵۰]..... ۳۱
- شکل (۱۲-۲) مثالی از Local Bitmap [۵۰]..... ۳۱
- شکل (۱۳-۲) جستجوی درختی [۵۴]..... ۳۳
- شکل (۱۴-۲) مدار بررسی پوشش خطا در روش BRANCH [۵۶]..... ۳۴
- شکل (۱-۳) نمونه‌ی یک حافظه ۸×۸ معیوب با دو ردیف و دو ستون افزونه..... ۳۸
- شکل (۲-۳) اعمال رشته تعمیر R1-R5-C2-C3 به حافظه‌ی شکل (۱-۳)..... ۳۹
- شکل (۳-۳) اعمال رشته تعمیر C3-C5-R5-R3 به حافظه‌ی شکل (۱-۳)..... ۳۹
- شکل (۴-۳) ساختار تحلیلگر مبنا..... ۴۰
- شکل (۵-۳) مراحل تعمیر توسط تحلیلگر مبنا..... ۴۱
- شکل (۶-۳) تحلیلگر کاهش‌یافته مطابق با R-CRESTA..... ۴۳
- شکل (۷-۳) درخت جستجو به ازای آرایش افزونگی ۲×۲..... ۴۴
- شکل (۸-۳) ساختار مفهومی تحلیلگر پیشنهادی..... ۴۴
- شکل (۹-۳) روند تحلیل خطا در مرحله‌ی اول به ازای دریافت هشت خطا..... ۴۶
- شکل (۱۰-۳) روند تحلیل پس از تکرار آزمون به ازای دریافت هشت خطا..... ۴۷
- شکل (۱۱-۳) ساختار سلول تحلیلگر طراحی شده..... ۴۸
- شکل (۱۲-۳) نمودار RTL مقایسه‌گر آدرس..... ۴۹
- شکل (۱۳-۳) نمودار RTL سلول تحلیلگر پیشنهادی..... ۵۰
- شکل (۱۴-۳) ساختار پیشنهادی زیرتحلیلگر..... ۵۱
- شکل (۱۵-۳) نمودار RTL شمارنده..... ۵۱
- شکل (۱۶-۳) نمودار RTL دیکدر سلول تحلیلگر..... ۵۲

۵۳ شکل (۱۷-۳) ساختار کامل تحلیلگر
۵۴ شکل (۱۸-۳) دیاگرام مفهومی مدار انتخابگر تعمیر زیرتحلیلگرها
۵۷ شکل (۱-۴) شکل موج تعمیر حافظه‌ی شکل (۱-۳)
۶۰ شکل (۲-۴) floor plan مدار تحلیلگر و انتخابگر راه‌حل
۶۱ شکل (۳-۴) تصویر CPLD برنامه‌ریزی شده و وضعیت پورت‌ها پس از دریافت خطای هشتم و ارائه‌ی راه‌حل نهایی
۶۱ شکل (۴-۴) مقایسه‌ی مقدار واحدهای ذخیره‌سازی موردنیاز تحلیلگرهای مختلف، برای حافظه‌ی ۱۰۲۴×۱۰۲۴
۶۵ بهازای آرایش‌های گوناگون افزونه‌ها
۶۵ شکل (۵-۴) مقایسه‌ی مقدار واحدهای ذخیره‌سازی مورد نیاز تحلیلگرهای مختلف، برای حافظه‌های گوناگون به
۶۶ ازای آرایش افزونه‌ی ۲×۲
۶۶ شکل (۶-۴) مقایسه‌ی تعداد سیکل کلاک‌های موردنیاز تحلیلگرهای مختلف، برای حافظه‌ی ۱۰۲۴×۱۰۲۴ با دو
۶۷ ردیف و دو ستون جایگزین
۶۷ شکل (۷-۴) مقایسه‌ی سرعت تحلیلگر پیشنهادی با سایر تحلیلگرها براساس تعداد تکرار آزمون، به ازای
۶۷ پیکربندی‌های مختلف افزونه‌ها
۶۷ شکل (۸-۴) مقایسه‌ی نرخ تعمیر نرمالیزه به ازای تعداد خطای مختلف، در یک حافظه‌ی ۱۰۲۴×۱۰۲۴ با دو ردیف
۶۸ و دو ستون افزونه
۶۸ شکل (۹-۴) مقایسه‌ی نهایی تحلیلگر پیشنهادی با تحلیلگر ایده‌آل به ازای آرایش افزونه‌ی ۲×۲ در یک حافظه‌ی
۶۹ ۱۰۲۴×۱۰۲۴

فهرست جدول‌ها

- جدول (۱-۴) نتایج منابع مصرف شده‌ی CPLD ۶۱
- جدول (۲-۴) لیست تخصیص بین‌های CPLD ۶۲
- جدول (۳-۴) فضای اشغالی تحلیلگرهای مختلف در پیکربندی افزونه‌ی ۲×۲ در یک حافظه‌ی ۱۰۲۴×۱۰۲۴ ۶۴

فهرست علائم اختصاری

ADC	Analog to Digital Converter
ASIC	Application-Specific Integrated Circuit
ATE	Automatic Test Equipment
BIRA	Built-in Redundancy Analyzer
BISR	Built-in Self Repair
BIST	Built-in Self Test
CAM	Content Addressable Memory
CMOS	Complementary Metal-Oxide Semiconductor
CPLD	Complex Programmable Logic Device
CRESTA	Comprehensive Realtime Exhaustive Search Test and Analysis
DAC	Digital to Analog Converter
DBL	Divided Bit Line
DFT	Design for Test
DRAM	Dynamic Random Access Memory
DWL	Divided Word Line
ECC	Error Checking and Correction
ELRM	Extended Local Repair Most
EOF	Early termination by the number of Orthogonal Faulty cells
ESP	Essential Spare Pivoting
FPGA	Field Programmable Gate Array
ITRS	International Technology Roadmap for Semiconductors
LO	Local Optimization
LRM	Local Repair Most
NP	Nondeterministic Polynomial time
NV memory	Non Volatile memory
R-CRESTA	Reduced Comprehensive Realtime Exhaustive Search Test and Analysis
RISC	Reduced Instruction Set Computer
RM	Repair Most
SOC	System On Chip
SRAM	Static Random Access Memory
TAM	Test Access Mechanism
ULSI	Ultra Large Scale Integration
VLSI	Very Large Scale Integration

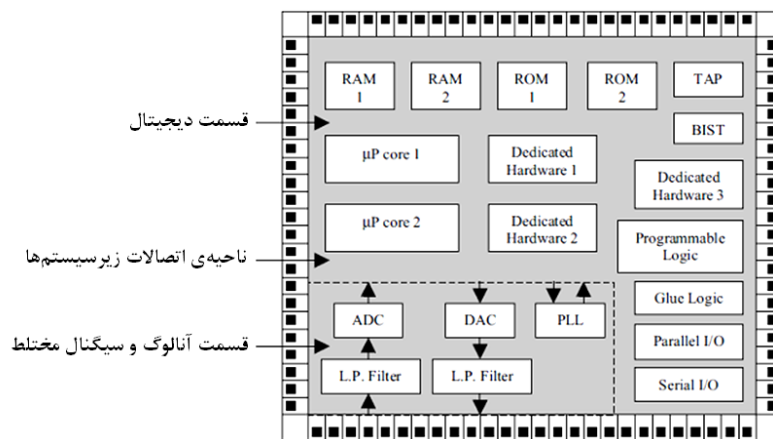
##

فصل اول:

##مقدمه

۱-۱- مقدمه

با پیشرفت فن‌آوری CMOS^۱ در قلمرو زیرمیکرون، میزان مجتمع‌سازی، کارایی و هزینه‌ی ساخت به‌طور چشمگیری افزایش می‌یابد. به‌همین دلیل در ساخت مدارهای کوچک که نیاز به مجتمع‌سازی بالا ندارند، استفاده از این فن‌آوری مقرون‌به‌صرفه نیست و تنها تراشه‌های سیستمی با کارایی بالا ارزش پیاده‌سازی با این روش را دارند. به این تراشه‌های سیستمی، سیستم‌روی تراشه^۲ گفته می‌شود [۱]. نمونه‌ای از یک سیستم‌روی تراشه در شکل (۱-۱) دیده می‌شود که شامل هسته‌های پردازنده، حافظه‌های جاسازی‌شده، DAC^۳ و ADC^۴ و غیره می‌باشد.



شکل (۱-۱) مثالی از یک سیستم روی تراشه [۲]

دو علت برای تبدیل حافظه‌های جاسازی شده به یک عنصر کلیدی در SOCها وجود دارد. دلیل اول این است که فن‌آوری CMOS امکانات مناسبی جهت مجتمع‌سازی حافظه‌های بزرگ با مدارات دیگر ارائه می‌دهد. این فن‌آوری قابلیت پیاده‌سازی تراشه‌های ULSI^۵ با بیش از 10^9 عنصر روی یک تراشه را دارا می‌باشد. پردازش داده‌ها و ذخیره‌سازی آن‌ها از اساسی‌ترین مولفه‌های مدارهای دیجیتال هستند. بنابراین پیاده‌سازی حافظه‌ها روی تراشه از اولویت بالایی برخوردار است. دلیل دیگر این است که پهنای باند حافظه یکی از چالش‌های جدی در عملکرد سیستم‌ها است. محدودیت سرعت I/Oها باعث ایجاد شکاف بین سرعت ریزپردازنده‌ها و حافظه‌ها می‌شود. این شکاف سرعت باعث افزایش تقاضا برای مجتمع‌سازی حافظه‌ها با ریزپردازنده‌ها روی یک تراشه شده است.

^۱Complementary Metal-Oxide Semiconductor

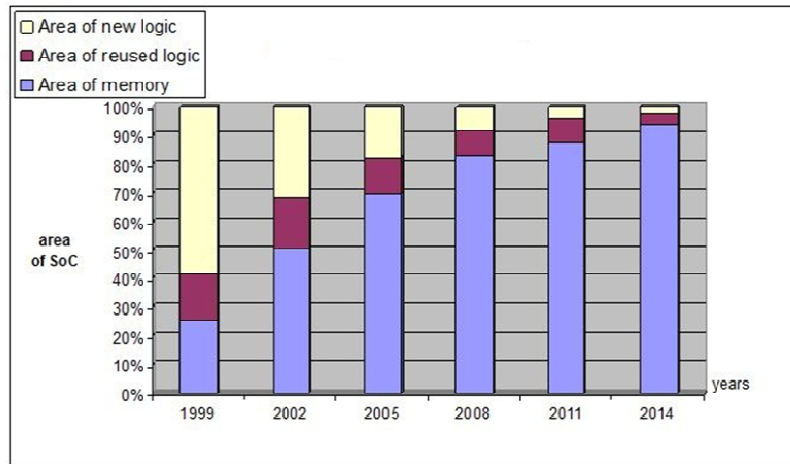
^۲System On Chip

^۳Digital to Analog Converter

^۴Analog to Digital Converter

^۵Ultra Large Scale Integration

معماری RISC^۱ شدیداً وابسته به پهنای باند حافظه است به گونه‌ای که ریزپردازنده‌های RISC با کارایی بالا، دارای فرکانس بیش از ۲۵ تا ۵۰ مگاهرتز، با حافظه‌ی Cache روی تراشه تجهیز می‌شوند [۲]. بر اساس پیش‌بینی ITRS 2001^۲ [۳]، تا سال ۲۰۱۴ بیش از ۹۰ درصد فضای اشغالی SOCها را حافظه‌ها به خود اختصاص خواهند داد [۷-۴]. شکل (۲-۱) نموداری از رشد مساحت حافظه‌ها نسبت به مساحت SOC را نشان می‌دهد. هم‌اکنون در تراشه‌هایی مانند Strong ARM1100 ۹۰ درصد از فضای تراشه به حافظه اختصاص یافته است [۸]. همچنین ۱۳۵ میلیون ترانزیستور از ۱۵۲ میلیون ترانزیستور تراشه‌ی Compaq Alpha EV7 مربوط به حافظه‌ها تراشه است.



شکل (۲-۱) نمودار رشد فضای اشغالی حافظه نسبت به کل فضای SOC [۳]

با کاهش طول گیت به زیر ۲۰ نانومتر در سیستم‌های VLSI^۳ چالش‌های جدیدی در زمینه‌ی طراحی پدید می‌آید. با پیچیده‌تر شدن پروسه‌ی ساخت، نقص‌های حین ساخت افزایش می‌یابند و این در حالی است که به علت پیچیده‌تر شدن سیستم، آزمون آن بسیار دشوارتر می‌گردد. مشکلاتی مانند نویز هم‌شنوایی^۴، افت ولتاژ، اتلاف توان، توزیع گرمایی سبب ایجاد انواع خطاها در حافظه و کاهش قابلیت اطمینان^۵ می‌شود [۹]. علاوه‌براین کوچک شدن اندازه مشخصه^۶، تراشه را مستعد نقص‌های غیرقابل پیش‌بینی می‌نماید که ناشی از ذرات آلفا بوده و خطای

¹Reduced Instruction Set Computer

²International Technology Roadmap for Semiconductors

³Very Large Scale Integration

⁴Cross talk

⁵Reliability

⁶Feature size

نرم^۱ نامیده می‌شوند.

پیشرفت‌های زیادی در تکنیک‌هایی همچون طراحی آزمون‌پذیر^۲ و خودآزمون درون‌ساخته^۳ صورت گرفته است. روش‌های آزمون در گذشته بر استفاده از ابزارهای خارج از تراشه یا ATE^۴ استوار بودند [۱۰-۱۳]. اما در مدارهای مجتمع کنونی کارآیی خود را از دست داده‌اند. زیرا روش‌های مبتنی بر ATE تنها خطاهای ناشی از نقص‌های تراشه^۵ را تشخیص می‌دادند اما قادر به آشکارسازی خطاهای تصادفی یا خطاهای ناشی از تغییر دما مثل خطاهای نرم نبودند [۱۴]. از سوی دیگر در یک SOC حافظه‌هایی با اندازه‌ها و معماری‌های گوناگون به کار رفته است و باید از ابزارهای آزمون و تعمیر با پیکربندی‌های متفاوت استفاده شود که این مسئله خود منجر به تحمیل هزینه‌های مضاعفی شود. با افزایش مجتمع‌سازی SOCها امکان دسترسی به پایه‌های حافظه‌ی تحت آزمون کاهش می‌یابد. بنابراین به مرور زمان برای رفع این مشکلات، کاهش هزینه، افزایش سرعت و افزودن قابلیت تعمیر با سرعت^۶، روش‌های خارج از تراشه جای خود را به رهیافت‌های درون‌ساخته و روی تراشه دادند.

وقوع نقص یا خطا در سلول‌های حافظه، به کاهش بهره‌دهی^۷ آن منجر می‌شود. با توجه به این که حافظه بخش عمده‌ی SOC را تشکیل می‌دهد، کاهش بهره‌دهی آن سبب کاهش بهره‌دهی کل مدار می‌شود. جهت رفع این مشکل و همچنین برای دستیابی به قابلیت اطمینان^۸ بالا، سیستم باید به صورت دینامیک دارای قابلیت تحمل خطا باشد تا بتواند در مرحله‌ی کارکرد، خطاها را رفع کند. یک روش معمول برای حل این مشکل، جایگزین کردن سلول‌های معیوب با سلول‌های سالم یا به اصطلاح افزونه‌ها^۹ می‌باشد. این کار توسط مدار خودتعمیر درون‌ساخته^{۱۰} که روی تراشه در کنار حافظه قرار می‌گیرد، انجام می‌شود. در [۱۵] با استفاده از این روش ۱۱/۵۳٪ بهبود بهره‌دهی به دست آمده است.

¹Soft error

²Design for Test (DFT)

³Built-in Self-Test (BIST)

⁴Automatic Test Equipment

⁵Defects

⁶at-speed

⁷Yield

⁸Reliability

⁹Redundancy

¹⁰Built-in Self-Repair

۲-۱- مفاهیم اولیه

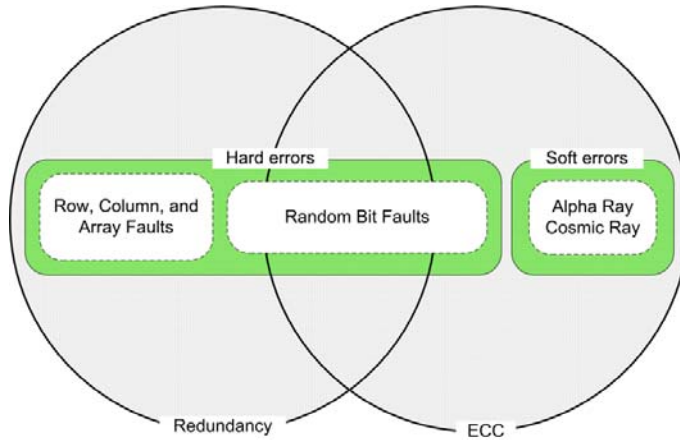
۱-۲-۱- انواع خطاها

به دنبال افزایش ظرفیت، کاهش اندازه مشخصه و کاهش ولتاژ کاری حافظه‌های جاسازی شده انواع گوناگونی از نقص‌ها و خطاها پدید می‌آیند. از جمله این خطاها می‌توان خطاهای سخت/نرم، خطاهای حاشیه ولتاژ/زمان و خطاهای مرتبط با سرعت^۱ را نام برد. روش‌های اصلاح این خطاها با توجه به نوع خطا متفاوت است. طرح ارائه شده در این پروژه برای تعمیر خطاهای سخت مورد استفاده قرار می‌گیرد. خطاهای سخت که اساساً در اثر نقص‌های طی پروسه‌ی ساخت پدید می‌آیند سبب عملکرد نادرست قسمتی از یک تراشه می‌شوند. شکل (۱-۳) انواع خطاهای سخت و روش اصلاح آن‌ها را در کنار خطاهای نرم نشان می‌دهد. خطاهای سخت را می‌توان بر اساس تعداد سلول‌های معیوب و نوع توزیع آن‌ها به سه دسته تقسیم کرد که عبارتند از خطاهای تک بیتی (معیوب بودن تنها یک سلول)، خطاهای ردیف/ستون (معیوب بودن تمام سلول‌های یک ردیف یا ستون) و خطاهای آرایه‌ای (معیوب بودن تمام سلول‌های یک آرایه).

خطاهای نرم خطاهایی هستند که در اثر آن‌ها اطلاعات ذخیره شده در گره‌های سلول‌های حافظه و یا سایر گره‌های مدار از دست می‌رود. این خطاها در اثر نویز به خصوص ذرات آلفا و یا پرتوهای کیهانی پدید می‌آیند. از آن‌جا که در اثر خطاهای نرم حافظه دچار خرابی نمی‌شود، با نوشتن مجدد اطلاعات این مشکل برطرف می‌شود. جهت اصلاح خطاهای نرم از روشی موسوم به ECC^۲ استفاده می‌شود [۱۶-۱۹]. به این ترتیب که طی مرحله‌ی نوشتن در حافظه، مدار کدگذاری از روی بیت‌های داده‌ی ورودی چک‌بیت‌هایی را تولید کرده و با استفاده از آن به آشکارسازی و رفع خطاها می‌پردازد. هرچند ECC برای اصلاح هر دو نوع خطا استفاده می‌شود اما اغلب برای تعمیر خطاهای نرم به کار گرفته می‌شود و افزونه‌ها به‌عنوان راه‌حل اصلی تعمیر خطاهای سخت به‌شمار می‌روند [۲۰] که در این پروژه نیز به آن پرداخته شده است.

^۱Speed relevant

^۲Error Checking and Correction



شکل (۳-۱) خطاهای سخت و نرم و راه‌حل اصلاح آن [۲۰]

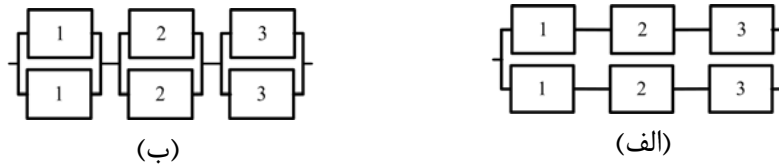
۱-۲-۲- تحمل خطا^۱

به قابلیت یک سیستم در تداوم اجرای عملیات مورد نظر علی‌رغم بروز نقص، تحمل خطا گفته می‌شود. داشتن چنین ویژگی در کاربردهای حساس مثل موارد پزشکی و هوافضا از اهمیت فراوانی برخوردار می‌باشد. سیستم با تحمل خطا قادر است به روش سخت‌افزاری یا نرم‌افزاری خطاهای غیرقابل پیش‌بینی را برطرف سازد. سیستمی با این ویژگی دارای قابلیت اطمینان بالاتری نیز خواهد بود. اگر قابلیت اطمینان یک عنصر سیستم ۹۹/۹۹٪ باشد، قابلیت اطمینان کل سیستم متشکل از ۱۰۰۰۰ قطعه از این عنصر تنها ۳۶/۷۹٪ خواهد بود [۲۱]. بنابراین طراحی کلیه اجزای سیستم به گونه‌ای که دارای تحمل خطا باشند به بهبود قابلیت اطمینان کل سیستم کمک بسزایی می‌نماید. از آن‌جا که حافظه‌های جاسازی شده عنصر غالب در SOCها می‌باشند، در این پروژه تلاش شده است که با افزودن ویژگی خودتعمیری، تحمل خطا در حافظه افزایش یافته و در نتیجه قابلیت اطمینان آن نیز بهبود یابد.

با وجود تلاش طراحان و سازندگان در ارائه‌ی محصول بی‌نقص، بروز خطا در سیستم اجتناب‌ناپذیر است. عامل برخی از این خطاها عوامل محیطی بوده که از کنترل سازنده خارج است. راه‌های گوناگونی جهت دستیابی به یک سیستم دارای تحمل خطا وجود دارد که یکی از موارد رایج آن استفاده از عناصر جایگزین یا افزونه‌ها می‌باشد. افزونه‌ها به دو دسته‌ی اصلی افزونه‌ی فضایی و افزونه‌ی زمانی تقسیم می‌شوند. در جایگزینی فضایی عناصر یا داده‌هایی مضاعف به سیستم افزوده می‌شود که در صورت عدم وقوع خطا بدون استفاده خواهند ماند. این نوع

^۱Fault tolerance

افزونه خود به سه دسته سخت‌افزاری، نرم‌افزاری و اطلاعاتی تقسیم می‌گردد. در جایگزینی زمانی، محاسبات یا داده‌های ارسالی تکرار می‌شوند و نتیجه حاصل با نتیجه ذخیره شده پیشین مقایسه می‌گردد. منظور از افزونه در این پایان‌نامه جایگزین فضایی نوع سخت‌افزاری می‌باشد. حافظه از آرایه‌ای از سلول‌های حافظه تشکیل می‌شود که در ردیف‌ها و ستون‌هایی کنار یکدیگر قرار گرفته‌اند. بنابراین می‌توان جایگزین‌هایی به صورت سلولی، ردیفی، ستونی و بلوکی را برای آن در نظر گرفت. جهت انتخاب یکی از این جایگزین‌ها برای حافظه‌ی خودتعمیر، دو سیستم شکل (۴-۱) را در نظر می‌گیریم.



شکل (۴-۱) جایگزینی سطح بالا و (ب) سطح پایین

در شکل (۴-۱-الف) که نمونه‌ای از جایگزینی سطح بالا می‌باشد کل سیستم متشکل از اجزای ۱ و ۲ و ۳ به-عنوان جایگزین در کنار سیستم اصلی قرار گرفته است در حالی که در شکل (۴-۱-ب) که نشان‌دهنده‌ی جایگزینی سطح پایین می‌باشد به ازای هر یک از اجزای سیستم یک جایگزین قرار داده شده است. اگر قابلیت اطمینان سیستم‌های الف و ب را به ترتیب با R_a و R_b نشان دهیم مقادیر زیر را خواهیم داشت [۲۱]:

$$R_a = 1 - (1 - R_1 R_2 R_3)^2 \quad (۱-۱)$$

$$R_b = (1 - (1 - R_1)^2)(1 - (1 - R_2)^2)(1 - (1 - R_3)^2) \quad (۲-۱)$$

برای مقایسه‌ی قابلیت اطمینان دو سیستم این دو مقدار را از هم کم می‌کنیم. با فرض این‌که قابلیت اطمینان همه‌ی اجزا برابر با R باشد خواهیم داشت:

$$R_b - R_a = 6R^3(1 - R)^2 \quad (۳-۱)$$

بنابراین R_b همواره از R_a بزرگتر خواهد بود یعنی در صورتی که جایگزینی را در سطح عناصر کوچکتر سیستم انجام دهیم به قابلیت اطمینان بالاتری دست خواهیم یافت.

۱-۲-۳- بهره‌دهی

بهره‌دهی یک تراشه به صورت نسبت اجزای سالم و قابل قبول به کل اجزای ساخته شده در پروسه ساخت تعریف می‌شود. یکی از مشکلاتی که با رشد مجتمع‌سازی حافظه‌های جداسازی شده پدید می‌آید، کاهش بهره‌دهی حافظه و در نتیجه کاهش بهره‌دهی کل سیستم است. افزونه‌ها به‌طور گسترده به‌عنوان یک راه‌حل موثر در بهبود بهره‌دهی و کاهش هزینه به‌کار گرفته شده‌اند. برای تحقیق این مسئله ابتدا باید مدلی از توزیع خطا در دست داشته باشیم.

یک تراشه حافظه با N عنصر را در نظر می‌گیریم، این عناصر می‌توانند یک سلول، ردیف، ستون و یا حتی یک آرایه حافظه باشند. اگر احتمال وقوع نقص در یک عنصر p باشد و حافظه دارای N عنصر باشد، آنگاه احتمال معیوب بودن k عنصر از حافظه را می‌توان با توزیع دوجمله‌ای زیر نمایش داد [۲۰]:

$$p(k) = \binom{N}{k} p^k (1-p)^{N-k} \quad (۴-۱)$$

اغلب N بسیار بزرگ و p بسیار کوچک است، با فرض $\lambda = Np$ خواهیم داشت:

$$p(k) = \frac{N(N-1)\dots(N-k+1)}{k!} p^k (1-p)^{N-k} \quad (۵-۱)$$

$$= 1 \cdot \left(1 - \frac{1}{N}\right) \dots \left(1 - \frac{k+1}{N}\right) \frac{\lambda^k}{k!} \left(1 - \frac{\lambda}{N}\right)^N \left(1 - \frac{\lambda}{N}\right)^{-k}$$

اگر $N \rightarrow \infty$ به عبارت (۶-۱) می‌رسیم که یک توزیع پواسن است:

$$p(k) = \frac{\lambda^k}{k!} \left(1 - \frac{\lambda}{N}\right)^N = \frac{\lambda^k e^{-\lambda}}{k!} \quad (۶-۱)$$

مدل پواسن به علت سادگی ریاضی، برای محاسبه بهره‌دهی حافظه مورد استفاده قرار می‌گیرد [۲۳، ۲۲، ۱۱].

ثابت می‌شود که λ برابر با میانگین تعداد خطاها می‌باشد. بنابراین می‌توان آن را به صورت زیر نشان داد:

$$\lambda = AD \quad (۷-۱)$$

در این رابطه A مساحت تراشه و D چگالی خطا می‌باشد. احتمال بی‌خطا بودن یک تراشه با قرار دادن $k = 0$

به دست می‌آید و این مقدار با بهره‌دهی حافظه بدون جایگزین برابر است: