APJ. 4(3):1-16, Autumn 2023

Arman Process Journal (APJ)

Homepage: armanprocessjournal.ir

An overview of Online Fault Tolerance for FPGA logic blocks

S. Gohari[*,1]

[1] Computer Engineering Department, Islamic Azad University, North Tehran Branch, Tehran, Iran

ABSTRACT

Received: 28 August 2023
Accepted: 19 November 2023

KEYWORDS:
Adaptive Computing,
Fault Tolerance,
Field-Programmable Gate Arrays (FPGA),
Reconfigurable Computing,
Reconfigurable Systems,
Reliability

[1] Corresponding author
✉ *sgohari.1984@gmail.com*

Most adaptive computing systems use reconfigurable hardware in the form of field programmable gate arrays (FPGA). In order for these systems to be fielded in harsh environments where high availability and reliability are a requirement, the programs running on FPGAs must be hardware fault tolerant, as this is the case during the lifetime of the system. may occur In this paper, we present new fault tolerance techniques for FPGA logic blocks, which are developed as part of the Self-Standing Test Areas (STAR) approach for test and diagnosis, and online configuration (we tolerate over 100 logic faults through the actual implementation on an FPGA containing a 20 x 20 array of logic blocks). A key feature is the reuse of incomplete logic blocks to increase the number of effective spares and extend mission length. To increase fault tolerance, we not only use faulty non-faulty sections or logic blocks with minor faults, but also use faulted sections of faulty logic blocks in non-faulty modes. By using and reusing faulty resources, our multilevel approach extends the number of tolerable faults beyond the number of available spare logic resources. Unlike many row, column, and piecewise methods, our multi-level approach can tolerate faults that are evenly distributed over the logic area, while also clustering faults in the same local area. Meanwhile, system operations are not interrupted for fault detection or computational fault-transitor configurations. Our fault tolerance techniques are implemented using ORAC2 series FPGAs that specify incremental dynamic runtime configuration.

NUMBER OF REFERENCES
20

NUMBER OF FIGURES
10

NUMBER OF TABLES
3

نشریه تخصصی آرمان پردازش، دوره ۴، شماره ۳، پاییز ۱۴۰۲

فصلنامه تخصصی آرمان پردازش
(APJ)
Homepage: www.armanprocessjournal.ir

مروری بر تحمل پذیری خطای آنلاین برای بلوک های منطق FPGA

سپیده گوهری[۱،*]

دانشکده مهندسی کامپیوتر، *دانشگاه آزاد اسلامی، واحد تهران شمال، تهران، ایران*

چکیده

اخیرا بیشتر سیستم های محاسباتی انطباقی از سخت افزار قابل پیکربندی مجدد به فرم آرایه های گیت قابل برنامه نویسی فیلدی (FPGA) استفاده می کنند. برای این که این سیستم ها در محیط های خشن جای میدان دادن داشته باشند در جایی که دسترس پذیری و قابلیت اعتماد بالا یک الزام هستند، برنامه های در حال اجرا روی FPGA ها باید از نظر سخت افزاری متحمل نقص باشند چون این در طول عمر سیستم ممکن است رخ دهد. در این مقاله ما تکنیک های تحمل نقص جدید برای بلوک های منطق FPGA ارائه می کنیم، که بصورت بخشی از رویکرد نواحی خود تست گردان (STAR) برای تست و تشخیص، و پیکربندی آنلاین توسعه داده می شوند (ما تحمل بالای ۱۰۰ نقص منطقی را از طریق پیاده سازی واقعی روی یک FPGA شامل یک آرایه۲۰ در ۲۰ از بلوک های منطق است). یک ویژگی کلیدی استفاده دوباره از بلوک های منطق ناقص برای افزایش تعداد زاپاس های موثر و بسط طول ماموریت می باشد. برای افزایش تحمل نقص، نه تنها از بخش های غیر خطادار معیوب یا بلوک های منطق با خطای جزئی استفاده می کنیم بلکه از بخش های خطادار بلوک های منطق معیوب در مد های غیر خطا دار استفاده می کنیم. با استفاده و کاربرد دوباره از منابع خطادار، رویکرد چند سطحی ما تعداد نقص های قابل تحمل را فراتر از تعداد منابع منطق زاپاس موجود می برد. بر خلاف خیلی از متدهای سطری، ستونی، تکه ای، رویکرد چند سطحی ما می تواند خطاهایی که به صورت برابر روی مساحت منطق توزیع شده تحمل کند، هم اینکه نقص ها در همان مساحت محلی را خوشه بندی می کند. در ضمن، عملیات سیستم به ازای تشخیص نقص یا به ازای پیکربندی های گذر دهنده-نقص محاسباتی دچار وقفه نمی شوند. تکنیک های تحمل خطای ما با استفاده از FPGA سری های ORAC2 پیاده شده که پیکربندی مدد زمان اجرای پویای افزایشی را مشخص می کند.

واژگان کلیدی:

محاسبات انطباقی،
تحمل خطا،
آرایه های گیت قابل برنامه نویسی –فیلد (FPGA)،
محاسبات قابل پیکربندی دوباره،
سیستم قابل پیکربندی مجدد،
قابلیت اعتماد،

تعداد جداول	تعداد شکل ها	تعداد مراجع
۳	۱۰	۲۰

مقدمه

سیستم های محاسبات انطباقی (ACS) به سخت افزار قابل پیکربندی وابسته هستند تا عملیات سیستم با تغییرات در محیط بیرونی منطبق گردد ، و ظرفیت ماموریت با پیاده سازی توابع جدید روی پلتفرم سخت افزاری مشابه قابل توسعه باشد. این منجر به افزایش تراکم تابعی و کاهش مصرف توان می شود و این ویژگی در حوزه های بسیاری اهمیت دارد [1]. برای مثال ماموریت های فضایی یا دستگاه های موبایل. آرایه های گیت قابل برنامه نویسی-فیلد (FPGA) پیکربندی زمان اجرای پویای افزایشی دارند که باعث می شود سیستم به اجرای بدون وقفه ادامه دهد. در حالیکه بخش هایی از FPGA برای توابع منطق جدید دوباره پیکربندی می شوند. ACS ها در محیط های خشن و راه دور پیاده می شوند، آنها دارای الزامات دسترس پذیری زیاد و قابلیت اعتماد بالا هستند. نواقص حاشیه ای که موجب خرابی در تست تولید کننده نمی شوند، با گذر عمر دستگاه ممکن است فعال شوند ، یا بخاطر عوامل محیطی ممکن است ظاهر شوند. چون دخالت مستقیم انسان برای نگهداری و تعمیر در این محیط ها شدنی نیست، تکنیک های متحمل نقض FT، برای داشتن بازه عمر مطلوب الزامی هستند. ما طرح FT سنتی، بر اساس افزونگی و رای گیری ماژولار نسخه برداری شده، از نظر فضا، وزن، محدودیت های توان ACS ها بسیار گران است [2]. در این مقاله، تکنیک های FT برای بلوک های منطق قابل برنامه نویسی FPGA (BLP) را ارائه می کنیم، که بصورت بخشی از پروژه Roving STARs توسعه داده شدند. در متدهای FT، خطاها درون بخش در حال کار سیستم کشف می شوند و آنها باید هر چه سریع تر محل یابی و گذر داده شوند طوری که عملیات سیستم نرمال از سر گرفته شود. در حوزه های خود تست کننده پرسه زن (STAR)، ما FPGA را به دو بخش متمایز تقسیم می کنیم: STAR ها، که در خود تست تعبیه شده (BIST) و در تشخیص شرکت دارد و حوزه کاری که تابع سیستم عمل می کند. بعد از تکمیل تست یک حوزه، یک STAR با تکه مجاور منطق سیستم تبادل جاد می کند، نهایتاً STAR ها میان کل FPGA پرسه می زنند. نتیجه استراتژی STAR های پرسه زن این است که خطاها همیشه در یک STAR کشف می شوند و آنها بر حوزه کاری FPGA اثر وارد نمی کنند. پس عملیات سیستم را دچار وقفه نمی کنیم که منبع نقص دار را با یک بدون نقص جایگزین کنیم، چون منطق در STAR هیچ تابع سیستمی را پیاده نمی کند وقتی که خطا کشف می شود. تفاوت دیگر آن این است تشخیص نقص و پیکربندی دوباره FT محدودیت های بلادرنگ شدید ندارند. چون کار سیستم نرمال در حوزه کاری ادامه دارد، زمان بیشتری برای تشخیص دقیق و برای محاسبه پیکربندی عبور دهنده نقص خواهیم داشت[3-2]. تحمل نقص بعد از اینکه نقص ها کشف و تعیین محل می شوند انجام می گیرند. در حالیکه روش STAR پرسه زن شامل تحمل نقص برای هر دو منطق و اتصال متقابل می باشد، این مقاله فقط با تحمل نقص برای منابع منطق سر و کار دارد. نخست، تعیین می کنیم آیا کار سیستم می تواند به درستی در حضور نقص ها ادامه داشته باشد. در خیلی از مواقع ، این شدنی است و پیکربندی دوباره لازم نیست. اگر یک نقص بر کار سیستم اثر گذاشت، پیکربندی های جایگزین را تعیین می کنیم که از منابع نقص دار اجتناب می کنند. غیر از عبور دادن نقص ها ، پیکربندی دوباره می تواند برای کاهش تنزل عملکرد ناشی از اجتناب نقص استفاده شود. برای رویکرد های FT دیگر، سرعت کلاک سیستم باید مطابق با بدترین حالت پیکربندی FT اصلاح شود. در حالتی که بدترین حالت هرگز رخ نمی دهد، سیستم با یک سرعت کلاک کندتر جریمه می شود [4].

جدول ۱-علامت های اختصاری

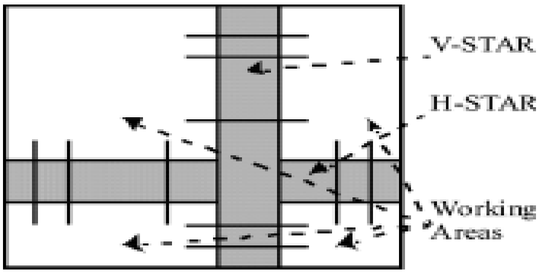
Acronym	Meaning
ASC	Adaptive Computing
BIST	Built-In Self Test
DSSM	Digital Single-side band Modulator
	Fault ByPassing Raving Configuration
FABRIC	Filip Flop
FF	FIBOnaccie number generator
FIB	Field Programmable GateArray
FPGA	Fault Tolerance
	Horizontal STAR
FT	Look- UP Table
H_STAR	Optimized Reconfigurable Array
LUT	Place and Route
ORCA	Programable Logic block
PAR	Partially Usable Block
PLB	Random Number Generator
PUB	Static Random Access Memory
PNG	Self-Testing Area
RTR	Test and Reconfiguration Contoroller

ما از مفهوم کلاک سیستم انطباقی که توسط مولد کلاک تولید شده استفاده می کنیم، دوره قابل برنامه ریزی این وقتی که سیستم برای گذر دادن نقص ها دوباره پیکربندی می شود ، می تواند تغییر داده شود. فرکانس کلاک اولیه ، به ماکزیمم مقدار مجاز در مدار بدون نقص تنظیم می شود. اگر مسیر زمان بندی بحرانی مدار در نتیجه ی پیکربندی دوباره FT تغییر کند ، نرخ کلاک بر اساس تحلیل زمان بندی پس از آدرس دهی تعدیل می شود. تحلیل زمان بندی بطور افزایشی فقط برای شبکه های سیگنالی تحت تاثیر پیکربندی دوباره اجرا می شود. بدون این تعدیل ، کلاک باید آنقدر کند باشد که برای طولانی ترین مسیر ایجاد شده توسط پیکربندی های FT کار بکند. در مقابل ، تعدیل یک دوره کلاک قابل برنامه نویسی ، جریمه های زمان بندی معرفی می کند ، این در نتیجه ی خطاهای جدید رخ می دهد [5].

ساختار مقاله بدین شرح است. در بخش ۲ ، رویکرد STAR پرسه زن را برای BIST انلاین و تشخیص بلوک های منطق قابل برنامه نویسی بررسی می کنیم. در بخش ۳ ، تحقیق قبلی مربوط به FPGA را مرور می کنیم. در بخش ۴، رویکرد FT سه سطحی را معرفی می کنیم ، ما از آن در زمان تحمل نقص اتصال متقابل و منطق استفاده می کنیم. در بخش ۵ ، جزئیات رویکردهای FT منطق را ارائه می دهیم. ما کاربرد دوباره منطق نقص دار را برای افزایش ظرفیت کمکی موثر، استفاده از استراتژی های تخصیص کمکی برای راحتی تحمل نقص محلی ، توصیف می کنیم همچنین استراتژی های نقص –متعدد برای رسیدگی به تعداد زیادی خطا و نقص در یک حوزه محلی. در بخش ۶ ، خلاصه و نتایج خود را بیان می کنیم[6] .

STAR های پرسه زن

فرض می کنیم یک پردازنده توکار ، به نام کنترلر پیکربندی دوباره و تست (TREC) ، BIST آنلاین ، تشخیص ، و تحمل خطا را برای همه FPGA ها در ACS مدیریت می کند.



شکل ۱- FPGA با STAR های پرسه زن

TREC ، که مکانیزم های تحمل خطای خاص-پردازنده جدا دارد ، همه پیکربندی های سیستم را حفظ می کند. هدف STAR های پرسه زن کشف خطاهایی است که در عمر سیستم رخ می دهند ، شامل نقص ها در حافظه پیکربندی. برای تست آنلاین نقص های گذرا ، منطق کاربردی که در FPGA پیاده شده ، از تکنیک کشف خطای هم زمان استفاده می کند. تحمل عیب برای خطاهای گذرا ، توسط ذخیره کردن حالت سیستم به شکل دوره ای و بازیابی آخرین حالت ذخیره شده، وقتی که گذرایی رخ می دهد ، محقق می شود [7]. STAR ها حوزه های FPGA که موقتا آفلاین هستند فراهم می کنند ولی بقیه دستگاه انلاین است و کار معمول را انجام می دهد. شکل ۱ FPGA با یک STAR عمودی (V-STAR) و یک STAR افقی (H-STAR) را نشان می دهد ، برنامه کاربردی سیستم در حوزه های در حال کار قرار می گیرد. RTR جزئی از طریق واسط اسکن مرزی FPGA اجازه می دهد پیکربندی تست توسط STAR ها انجام شود بدون اینکه روی سیستم اثر بگذارد. بعد از تست ، STAR به محل جدید می رود ، در حقیقت محل های با تکه اندازه مساوی حوزه کاری با هم مبادله می شوند. موقعیت های STAR اول بطور دلخواه انتخاب می شوند. در حین نکاشت تابع سیستم ، ما افرازهای STAR را ماسک می کنیم پس از آنها اجتناب می شود. با این کار تضمین می کنیم حوزه های STAR از ابتدا در دسترس هستند. STAR ها میان FPGA پرسه می زند این با یک دنباله از پیکربندی های مجدد جزئی از پیش محاسبه شده انجام می شود و تضمین می کند کل FPGA تست خواهد شد. چون تست ما گسترده است، پوشش عیب ۱۰۰ درصد است. تاخیر نقص ما بازه مورد نیاز برای تست کل FPGA است. فرایند گشت زنی و کاربرد STAR های گشت زن برای تست و تشخیص در مقالات دیگر توصیف شدند [8-9].

بسته به موقعیت های دو STAR ، حوزه کاری ممکن است پیوسته باشد ، یا به دو یا چهار ناحیه جدا تقسیم شده باشد. همه سگمنت های سیم افقی در H-STAR و همه سگمنت های عمودی در V-STAR برای تست رزرو می شوند. سیگنال های سیستم اجازه دارند از میان STAR ها با استفاده از سگمنت های سیم افقی از میان V-STAR یا سگمنت های عمودی از میان H-STAR عبور کنند. V-STAR بین FPGA گشت می زند ولی H-STAR بالا و پایین FPGA حرکت می

کند. یک STAR هر دو BLP ها و اتصال متقابل قابل برنامه نویسی را درون حوزه آن تست می کند. در حالیکه STAR برای تست همه PLB ها استفاده می شود، هر دو STAR برای تست اتصال متقابل قابل برنامه نویسی لازم هستند. اندازه STAR ها طوری انتخاب می شوند که منابع اتصال متقابل و منطق برای پیاده سازی مدار BIST فراهم شود، که این کاملا PLBها را درون STAR تست می کند در حالیکه تعداد PLB های تخصیص داده شده به STAR را کاهش می دهد، در نتیجه ، برای منطق سیستم دسترس ناپذیر است. برای معماری ORAC 2C ، V-STAR شامل دو ستون و H-STAR شامل دو سطر PLB در امتداد منابع مسیریابی مربوطه به آنها می باشد [10].

روش ما کل FPGA را تست می کند ، شامل منابع کمکی است. تست فقط منطق سیستم ، که در سیستم های افلاین انجام می شود ، برای تست انلاین مناسب نیست ، چون نقص های جدید در منابع کمکی یا در جای بلااستفاده ی عملیاتی سیستم رخ می دهد. اگر این نقص ها جمع شوند اثر تعیین کننده روی قابلیت اعتماد سیستم می گذارند چون این منابع بلا استفاده بعنوان جایگزین ها برای منابع نقص دار استفاده می شوند. برای تضمین یک عمل قابل اعتماد ، تست آنلاین ما همه ی منابع سیستم را چک می کند حتی کمکی ها را [11]. تکنیک BIST ما در مقاله ۷ توصیف شده و هر نقص یا چندین نقص را در یک PLB کشف می کند. وقتی عیب ها کشف می شوند ، هر گروه PLB ناقص بر اساس نتایج BIST محل یابی می شود. پیکربندی های BIST تشخیصی اضافی را می توان دانلود کرد و با آنها بخش های عیب دار خاص PLB را شناسایی کرد، مانند جدول های جستجوی ناقص (LUT) و فلیپ فلاپ های خطادار ، برای افزایش رزولوشن تشخیصی. هدف تشخیصی اکثر تکنیک های FT برای FPGA ها ، شناسایی PLB های خطا دار است ، که توسط زاپاس ها جایگزین می شوند. رزولوشن تشخیص بالای رویکرد ما اجازه می دهد شکل جدیدی از تحمل نقص را معرفی کنیم، که منابع نقص هر موقع لازم باشند دوباره استفاده می شوند. بلوک قابل کاربرد جزئی (PUB) یک PLB نقص دار با مد های شکست شناسایی شده یا زیر مدارات عیب دار می باشد. یک PUB ممکن است بعنوان یک سلول بدون نقص در بخش کاری سیستم استفاده شود ، چون نقص هایش بر عمل سیستم اثری نمی گذارند. استفاده دوباره از سخت افزار نقص دار ، ظرفیت زاپاس موثر را افزایش داده و باعث عمر ماموریت طولانی تر می شود [12]. در روش STAR های گشت زن، برای تشخیص نقص و محاسبه پیکربندی های مجدد گذر دادن خطا، دیگر نیازی به توقف عمل سیستم نیست. اما قابل ذکر است برای عوض کردن جای STAR باید زمان کافی برای پیکربندی وجود داشته باشد پس یک زمان خرابی یا توقف سیستم لازم است. در ضم در حالتی که کلاک متوقف شده ، قبل از راه اندازی دوباره اش باد وضعیت سیستم به ناحیه سیستم جدید کپی شود. در یک ORCA 2C15 FPGA (a 20 × 20 array) کلاک سیستم تقریباً به مدت $250\ \mu s$ ، برای تعویض محل STAR ، متوقف می شود [13]. کل زمان تست و پرسه زنی ORAC 2C15 ۱,۳۴ ثانیه است، وقتی که واسط اسکن مرزی آن در نرخ کلاک ماکزیمم خودش یعنی ۱۰ MHZ است. در بدترین حالت ، یک نقص به مدت ۱,۳۴ ثانیه ممکن است شناخته نشود. پس تشخیص PLB های خطا دار درون یک کاشی BIST بر اساس نتایج BIST معیوب انجام می شوند ، زمان پرسه زنی ۱,۳۴ ثانیه شامل تشخیص PLB نقص دار است. تشخیص PUB ، نیازمند حداکثر ۷,۲ ms در ORAC 2C15 است تا بتواند LUT عیب دار و فلیپ فلاپ را درون PLB عیب دار تشخیص دهد [14]. سایر تکنیک ها برای کشف نقص پیشنهاد شده اند. اما آنها با عمل نرمال سیستم در حوزه کاری FPGA سر و کار ندارند.

پیشینه پژوهش

تحمل پذیری خطای آنلاین برای بلوک ها به میزان قابل قبولی وابسته است در FPGA خطاهای آنلاین ممکن است ایجاد شود و در نتیجه FPGA نقص عملکرد بلوک اطلاعات نادرست تولید کند در برخی موارد دیگر خطاها ممکن است تا حدی تحمل پذیر باشند و تأثیر چندانی بر عملکرد نداشته باشند برای افزایش تحمل پذیری خطای آنلاین FPGA بلوک ها استفاده کرد. تکرار و توازن اطلاعات : در این روش بطور مکرر اطلاعات ورودی به بلوک FPGA تکرار می شوند در صورتیکه نتیجه تکرار ها متفاوت است از روش تصویب اکثریت استفاده می شود تا از حطاها جلوگیری شود [15-16]. رمزگذاری و تشخیص خطا : در این روش با استفاده از اطلاعات ورودی به بلوکFPGA از الگوریتم های رمزگذاری مشخصی رمزنگاری می شوند در صورتی که اطلا عات پس از رمز گذاری تغییر کنند می توان با الگوریتم های تشخیص خطا این تغییر را تشخیص داد و خطا را جلوگیری کرد. مدارهای ورودی / خروجی بطور مخصوص طراحی FPGA بلوک می شوند تا تحمل خطا را فراهم کنند این مدارها معمولا قادر به تشخیص و جبران خطاهای ساده هستند و در نتیجه عملکرد را در صورت وقوع خطا بلوک FPGA حفظ می کنند. مهمترین نکته در تحمل پذیری خطای آنلاین این است که بر اساس بلوک های مورد نیاز روش های مناسب انتخاب می شوند در طراحی و پیاده سازی بلوک های FPGA الگوریتم های مختلفی برای جلوگیری از خطاهای آنلاین و تحمل خطا متمرکز شده است. کارهایی روی تحمل عیب برای FPGA ها و دستگاه های حافظه دیگر انجام شده، ما فقط چند تلاش کلیدی را مورد اشاره قرار می دهیم.

چند تکنیک از شیفت دادن سطر یا ستون استفاده کردند. در مقاله [12]، هاتوری و همکاران ، یک ستون کمکی برای تحمل نقص ها معرفی کردند. آنها از مدار بندی انتخاب کننده تخصصی برای تنظیم

دوباره مدارات FPGA در واکنش به عیب ها استفاده کردند. مشابه روش ها برای FT در SRAM ها ، حداقل یک ستون اضافی PLB ها به FPGA افزوده می شود. اگر یک PLB عیب داشته باشد ، ستونش حذف می شود و همه توابع به ستون های بین ستون نقص دار نگاشته شده و نزدیک ترین ستون کمکی به سوی ستون کمکی شیفت داده می شوند. در [17]، دوراند و پیکوئیت از چندین ستون کمکی استفاده کردند. سربار مساحت و تحمل نقص برای هر تکنیک وابسته به تعداد ستون یا سطرهای کمکی است ، ولی اگر تعداد نقص ها در هر ستون یا سطر از ظرفیت کمکی آن ستون یا سطر بیشتر شد آنگاه کاری برای تحمل کردن متوسط نواقص باید انجام شود.

برای بهبود تحمل عیب توسط افزایش تعداد کل نقص ها ، ناراشیمهان و همکاران یک تکنیک FT برای FPGA ها یا آرایه های مجتمع اندازه-میفر، توسعه دادند. آنها از یک الگوریتم شیفت بیشتر از یک برای پیکربندی دوباره PLB های نقص دار استفاده کردند. روش آنها انعطاف پذیر بود چون محدود به یک خطا در هر سطر، ستون یا کاشی نبود. تکنیک آنها مشابه ماست آنها از منابع بلا استفاده برای کمکی ها یا تحمل خطا استفاده می کنند. پس سربار مساحت وجود ندارد. روش آنها برای کاربرد آفلاین بود ، و اطلاعاتی روی براوردهای سرعت عملیاتی مشخص نمی شد. کلی و آیوی از افزونگی برای عبور دادن نقص ها در کاربردهای نگاشت شده بهFPGAها استفاده کردند. تکنیک FT آنها به شیفت دادن متکی است. آنها از یک سوئیچ پیکربندی مجدد برای تنظیم دوباره استفاده می کنند و آنها از ابزارهای محل و مسیر نرمال (PAR) برای نگاشت مدارات به FPGA ها استفاده می کنند. شبکه ماتریس سوئیچ ، تکنیک آنها را منعطف تر از تکنیک های ستون و سطر در ۱۲ و ۱۳ می کند. یک الگوریتم پیکربندی سوئیچ برای تنظیم سوئیچ های خاص حول PLB های عیب دار استفاده می شوند. متد آنها نیازمند منابع بلا استفاده یا کمکی برای تحمل نقص ها است ، تحمل خطا به منابعی که در اول در دسترس بودند و قابلیت اعتماد وابسته است. برای استعمال FPGA به اندازه ۸۰ درصد ، ۲۰ درصد منابع باید برای زاپاس ها در دسترس باشند. در مرجع ۱۷ ، کودپا و کوربا از FPFA های مبتنی بر SRAM Xilinx استفاده کردند تا ظرفیت های FT FPGA ها را نشان دهند. آنها بطور تصادفی از PLB های نقص دار استفاده کردند. الگوریتم آنها پوشش نقص را تعیین می کند و نرخ بازیابی نقص برای آن تعریف می شود این در یک FPGA مبتنی بر SRAM پیاده شده است. بازیابی خطا در کار آنها با متدهای افزونه ماژولار امکان پذیر بود. مشابه کلی و آلوی ، متد آنها نیازمند منابع کمکی است ، و تحمل خطا به تعداد زاپاس ها وابسته است. دوت و هانچک یک روش برای افزایش بهره FPGA توسعه دادند. آنها از منابع مسیریابی رزرو شده برای جایگزینی عاملیت PLB های نقص دار استفاده کردند. یک سطر یا ستون PLB برای کمکی رزرو می شود. اگر یک PLB در هر ستون یا سطر نقص دار بود ، عاملیت همه PLB ها در ستون یا سطر از نقص دار به PLB کمکی به سوی PLB کمکی شیفت داده می شد. منابع مسیریابی کمکی ، سربار مسیریابی مجدد تعویض مدار به روز شده را حذف کردند. این روش از نظر زمان پیکربندی دوباره انعطاف پذیر است. چون قبلا منابع کمکی تخصیص داده شده اند زمان پیکربندی با تعداد نقص ها رابطه خطی دارد. فقط در هر ستون تعداد خطاهای متحمل شده محدود است. [18] امرت و بیتا یک تکنیک FT برای پیکربندی مجدد بصورت افزایشی حول PLB های عیب دار توسعه دادند. آنها بیشتر از نقص های محلی را تحمل می کنند. آنها از تطبیق گرید مینیمم ماکسیمم استفاده کردند، تا محل های PLB عیب دار را به منابع PLB کمکی نگاشت دهند. توابع PLB بین PLB نقص دار و محل کمکی تطبیق داده شده اش به سوی محل کمکی شیفت داده می شوند. روش شیفت دادن ، تنزل عملکرد ناشی از تطبیق دادن توابع PLB با کمکی ها را کاهش داد. این روش از Xilinx FPGA استفاده نمود. لاکاماروجو و تیسر ایده شیفت دادن منطق را توسط ایده شیفت دادن منطق درون یک PLB اصلاح کردند آنها مقدار پیکربندی مجدد مورد نیاز برای تحمل نقص منطق را کاهش دادند.

لاک و همکارانش سیستم های FT با سر بار کم را توسعه دادند. روش آنها افراز کردن یک طراحی به تعدادی سلول بود. هر سلول تعدادی PLB کمکی داشت. در جایی که نقص در یک سلول دیده می شد یک PLB کمکی برای تعویض PLB نقص دار جایگزین می شد. وقتی نقص شناخته می شد پیکربندی مربوط به آن دانلود می شد. در این روش هر سلول نیازمند یک یا چند منبع کمکی بود. بدترین حالت تحل خطا توسط تعداد کمکی ها در سلول با کمترین کمکی ها تعیین می شد چون توانایی برای کشیدن کمکی ها از سلول های دیگر وجود نداشت. همه کارهای قبلی به جز لاک و همکارانش، به پیکربندی های خطا بصورت آنلاین متکی بودند. این نیازمند محاسبات سریع است تا دسترسی پذیری برای سیستم FT حفظ شود. پیکربندی های از پیش کامپایل شده لاک و همکارانش ، زمان خرابی سیستم را بطور قابل توجهی کاهش می دهند. نسبت به سربار مساحت ، با استثناهای کم روش های قبلی نیازمند حداقل ۱۰ درصد سربار مساحت برای تکنیک تحمل نقص بودند. در روش ما ، ما از منابع منطق نقص دار استفاده کردیم که هیچ سربار مساحتی ندارند همچنین منابعی که بلا استفاده هستند. ما نیاز به تخصیص از قبل نداریم، ولی کمکی های در دسترس بیشتری لازم است. تکنیک STAR BIST گشت زن ما به دو ستون و سطر FPGA برای کشف و تشخیص عیب ها نیاز دارد. نسبت به سرعت عملیاتی ، چون ما مدار گذر دادن خاص نداریم و نیازی هم به کمکی های از پیش تخصیص داده شده نیست ، عمل سیستم به سرعت های

بهینه نزدیک می شود. روش ما ماکزیمم سرعت عملیاتی را کاهش می دهد. نسبت به مصرف توان ، سربارهای منطق قابل پیکربندی مجدد نیازمند توان بیشتری از بخش های شمارنده غیر قابل پیکربندی دوباره هستند ، ولی تعداد آنها کم است. روش ما الزامات توان را افزایش می دهد اما به خاطر پیکربندی جزئی STAR ها در زمان تست و به دلیل فعالیت تست همزمان با عمل نرمال ، روش ما سهم قابل توجهی در این حوزه ایفا می کند. افزایش توان بخاطر STAR های گشت زن از مرتبه 2/N است با فرض اینکه کلاک سیستم و کلاک تست در همان فرکانس کار کنند، و خیلی کمتر وقتی که کلاک سیستم در فرکانس بالاتر از کلاک تست باشد [18].

روش FT چند سطحی

در روش FT ما ، ما از پیکربندی های از پیش کامپایل شده برای گذر دادن PLB های نقص دار استفاده می کنیم، ولی اگر پیکربندی های از پیش کامپایل شده در دسترس نبودند ، هر تنظیمی که لازم باشد تولید می کند ولی سیستم همچنان در حال اجراست. مزیت روش ما این است که تعداد زیادی از نقص های اتصال متقابل و منطق را تحمل می کند. سطح ۱ شامل ترک یک STAR پارک شده بر روی ناحیه ای است که خطا کشف شده بوده. در حالیکه STAR در آنجا پارک بوده سیستم کار خود را انجام می داده چون نقص ها در ناحیه کاری قرار ندارند. پارک STAR ادامه دارد در حالیکه نقص ها شناسایی می شوند. برای ماکزیمم تفکیک تشخیص در وضعیت های خاص ، مثلا داشتن چندین PLB عیب دار درون STAR ، یا وقتی LUT عیب دار یا فلیپ فلاپ را درون یک PLB شناسایی می کنیم، پیکربندی های اضافی مورد نیاز هستند [19]. نقص های شناسایی شده بر تکه بعدی منطق کاری که در موقعیت STAR پارک شده محل یابی شده تاثیر می گذارد. منابع مسیریابی و منطق بلا استفاده در ناحیه کاری بصورت کمکی ها برای گذر دادن نقص ها استفاده می شوند. سوال اینجاست منطق کاربردی طوری نگاشت شده که بعد از محل یابی دوباره ، روی منابع نقص دار که تازه شناسایی شدند قرار بگیرد. اگر نقص ها فقط بر منابع کمکی اثر بگذارند، پس اقدامی لازم نیست. سوال بعدی، آیا منابع نقص دار برای کار سیستم لازم هستند. اگر نقص با پیکربندی های فعلی سازگار نباشد، آنگاه قدم به قدم پیکربندی دوباره می کنیم تا قبل از اینکه ناحیه کاری در مرحله بعدی پرسه STAR تعیین محل شود از نقص اجتناب گردد. چند پیکربندی پرسه زن گذردهنده نقص (FABRICS) از پیش کامپایل می شوند ولی اکثر نقص ها نیازمند FABRIC های جدید هستند که بصورت انلاین محاسبه شده باشند. این محاسبه وقتی انجام می شود که STAR روی نقص ها پارک کرده باشد. مزیت پارک کردن STAR این است که تست ادامه دارد در حالیکه STAR پارک کرده. فرض کنید H STAR ، STAR پارک کرده است. بعد از اینکه نقص ها شناسایی شدند و در حالیکه TREC در حال محاسبه FABRIC است، V STAR می تواند به کار گشت زنی و تست ادامه دهد. این فرایند همه ی PLBها و همه اتصال متقابل عمودی را تست می کند، ولی منابع اتصال متقابل افقی معمولا توسط H STAR تازه پارک کرده تست می شوند. پس فرایند تست جزئی حتی وقتی یک STAR پارک کرده فعال است. سطح ۲ وقتی رخ می دهد که STAR از پارک درآمده و شروع به گشت می کند. در اینجا از FABRIC های تازه محاسبه شده یا از پیش کامپایل شده استفاده می کنیم. با کمک ظرفیت پیکربندی دوباره جزئی FPGA ها ، اندازه این FABRIC ها در مقایسه با حافظه لازم برای ذخیره کردن پیکربندی های FPGA کامل، نسبتا کوچک حفظ می شود. یک FABRIC جای یک منبع نقص دار را با کمکی عوض می کند. ایده ال این است همیشه در همسایگی نقص یک منبع کمکی وجود داشته باشد [19]. سطح ۳ وقتی است که همه منابع کمکی در ناحیه کاری برای تحمل خطا استفاده شده باشند. در اینجا از چند PLB که توسط دو STAR فراهم شده برای گذر دادن نقص ها استفاده می کنیم. این سطح به سرقت STAR معروف است. با سرقت STAR از دیگری ، کار پرسه زنی با هر دو STAR ادامه می یابد. در جایی که STAR کاملا مصرف می شود و STAR دیگری سرقت می کنیم ، می توانیم فقط با یک STAR جزئی گشت بزنیم. منابعی که از STAR گرفته می شوند دیگر کمکی نیستند [20]. برای روش STAR های پرسه زن ، تاخیر تست تابعی از زمان لازم برای تنظیم FPGA است. برای تست و تحمل خطا ، هر چه سریع تر FPGA را دوباره پیکربندی کنیم ، هر نقص را سریع تر کشف می کنیم. پیکربندی دوباره جزئی این فرایند را با کاهش زمان پیکربندی مورد نیاز ، وقتی که تست بارگذاری برای یک محل STAR خاص انجام می شود و وقتی که یک STAR را به یک محل جدید تخصیص می دهیم ، سرعت می بخشد. در روش FT ما ، ما از پیکربندی های از پیش کامپایل شده برای گذر دادن PLB های نقص دار استفاده می کنیم، ولی اگر پیکربندی های از پیش کامپایل شده در دسترس نبودند ، هر تنظیمی که لازم باشد تولید می کند ولی سیستم همچنان در حال اجراست. مزیت روش ما این است که تعداد زیادی از نقص های اتصال متقابل و منطق را تحمل می کند. سطح ۱ شامل ترک یک STAR پارک شده بر روی ناحیه ای است که خطا کشف شده بوده. در حالیکه STAR در آنجا پارک بوده سیستم کار خود را انجام می داده چون نقص ها در ناحیه کاری قرار ندارند. پارک STAR ادامه دارد در حالیکه نقص ها شناسایی می شوند. برای ماکزیمم تفکیک تشخیص در وضعیت های خاص ، مثلا داشتن چندین PLB عیب دار درون STAR ، یا وقتی LUT عیب دار یا فلیپ فلاپ را درون یک PLB شناسایی می کنیم ، پیکربندی های اضافی مورد نیاز هستند. نقص های شناسایی شده بر تکه بعدی منطق کاری که در موقعیت STAR پارک شده محل یابی شده تاثیر می

گذارد. منابع مسیریابی و منطق بلا استفاده در ناحیه کاری بصورت کمکی ها برای گذر دادن نقص ها استفاده می شوند. سوال اینجاست منطق کاربردی طوری نگاشت شده که بعد از محل یابی دوباره ، روی منابع نقص دار که تازه شناسایی شدند قرار بگیرد. اگر نقص ها فقط بر منابع کمکی اثر بگذارند ، پس اقدامی لازم نیست. سوال بعدی ، آیا منابع نقص دار برای کار سیستم لازم هستند. اگر نقص با پیکربندی های فعلی سازگار نباشد ، آنگاه قدم به قدم پیکربندی دوباره می کنیم تا قبل از اینکه ناحیه کاری در مرحله بعدی پرسه STAR تعیین محل شود از نقص اجتناب گردد. چند پیکربندی پرسه زن گذردهنده نقص (FABRICS) از پیش کامپایل می شوند ولی اکثر نقص ها نیازمند FABRIC های جدید هستند که انلاین محاسبه شده باشند. این محاسبه وقتی انجام می شود که STAR روی نقص ها پارک کرده باشد. مزیت پارک کردن STAR این است که تست ادامه دارد در حالیکه STAR پارک کرده. فرض کنید H STAR ، STAR پارک کرده است. بعد از اینکه نقص ها شناسایی شدند و در حالیکه TREC در حال محاسبه FABRIC است ، V STAR می تواند به کار گشت زنی و تست ادامه دهد. این فرایند همه ی PLBها و همه اتصال متقابل عمودی را تست می کند ، ولی منابع اتصال متقابل افقی معمولا توسط H STAR تازه پارک کرده تست می شوند. پس فرایند تست جزئی حتی وقتی یک STAR پارک کرده فعال است. سطح ۲ وقتی رخ می دهد که STAR از پارک درآمده و شروع به گشت می کند. در اینجا از FABRIC های تازه محاسبه شده یا از پیش کامپایل شده استفاده می کنیم. با کمک ظرفیت پیکربندی دوباره جزئی FPGA ها ، اندازه این FABRIC ها در مقایسه با حافظه لازم برای ذخیره کردن پیکربندی های FPGA کامل ، نسبتا کوچک حفظ می شود. یک FABRIC جای یک منبع نقص دار را با کمکی عوض می کند. ایده ال این است همیشه در همسایگی نقص یک منبع کمکی وجود داشته باشد. سطح ۳ وقتی است که همه منابع کمکی در ناحیه کاری برای تحمل خطا استفاده شده باشند. در اینجا از چند PLB که توسط دو STAR فراهم شده برای گذر دادن نقص ها استفاده می کنیم. این سطح به سرقت STAR معروف است. با سرقت STAR از دیگری ، کار پرسه زنی با هر دو STAR ادامه می یابد. در جایی که STAR کاملا مصرف می شود و STAR دیگری سرقت می کنیم ، می توانیم فقط با یک STAR جزئی گشت بزنیم. منابعی که از STAR گرفته می شوند دیگر کمکی نیستند. برای روش STAR های پرسه زن ، تاخیر تست تابعی از زمان لازم برای تنظیم FPGA است. برای تست و تحمل خطا ، هر چه سریع تر FPGA را دوباره پیکربندی کنیم ، هر نقص را سریع تر کشف می کنیم. پیکربندی دوباره جزئی این فرایند را با کاهش زمان پیکربندی مورد نیاز ، وقتی که تست بارگذاری برای یک محل STAR خاص انجام می شود و وقتی که یک STAR را به یک محل جدید تخصیص می دهیم ، سرعت می بخشد.

FT برای PLB ها

به دلیل گشت زدن ، منطق سیستم محل ثابتی ندارد. شکل ۲ نشان می دهد چطور PLB فیزیکی یکسان میان چهار تابع سلول منطق مختلف ، زمان-مشترک است. این به موقعیت های STAR ها بستگی دارد. تکنیک های FT ما با رفتار پویا مواجه هستند.

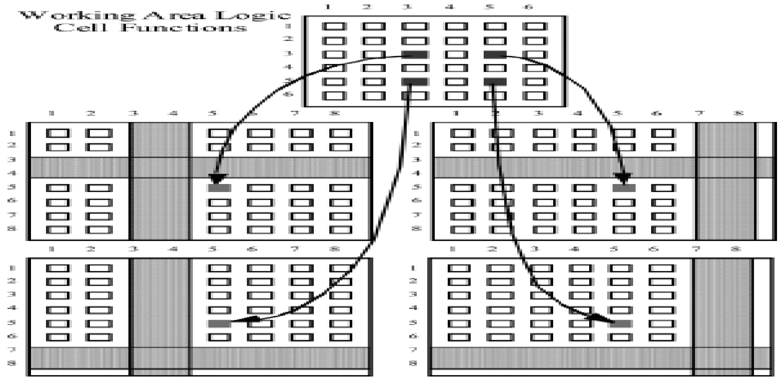


Fig. 2. Four logic cells time sharing one PLB.

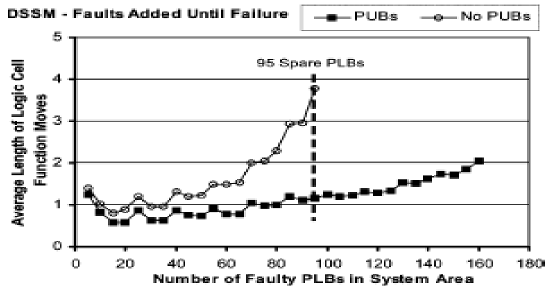
شکل ۲. تسهیم زمانی سلول های منطق

هر جا لازم باشد PLB ها را بعنوان PUB ها استفاده می کنیم. ما از منبع نقص دار در یک مد غیر عیب دار استفاده می کنیم. مثلا اگر یک LUT دارای یک پشته سلول در ۰ باشد ، می توانیم از این برای پیاده سازی تابعی استفاده کنیم که نیازمند یک ۰ در آن سلول است. اگر این ممکن نباشد ، سعی می کنیم از منطق بدون نقص در PLB برای تابع های دیگر استفاده کنیم. این روش اجازه می دهد خطاهای منطق بیشتری را کنترل کنیم. اگر استفاده دوباره ممکن نباشد ما FABRIC ها را محاسبه می کنیم. با ترکیب استفاده مجدد PLB های ناقص با گذر دادن نقص از طریق FABRIC ، تکنیک ما می تواند گروه های نقص بزرگتری را در یک ناحیه فشرده رسیدگی کند. در حالی که تکنیک

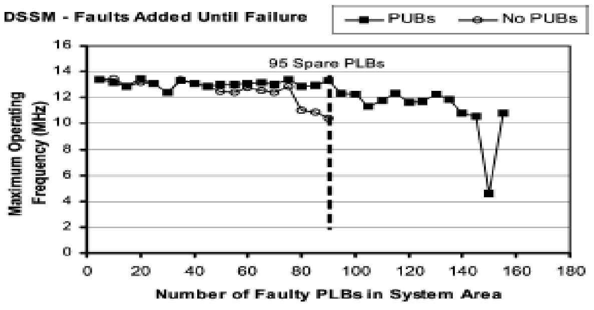
های دیگر محدود به تعداد خاصی کمکی در یک زیر مجموعه از FPGA هستند. وقتی این کمکی های محلی مصرف می شوند آن تکنیک ها شکست می خورند. مثلا شیفت دادن سطر یا ستون در کارهای هاتوری ، دوراند ، و دوت ، اجازه می دهد که با سخت افزار تعبیه شده خیلی سریع عمل تنظیم دوباره انجام شود ولی آنها یکی دو نقص در هر ستون را تحمل می کنند. روش کاشی کاری در رفرنس های ۲۳ و ۲۴ ، FABRIC هایی برای تکنیک ما ایجاد می کند. اما، روش کاشی کاری محدودتر از روش ماست. مثلا اگر تعداد کل نقص ها درون یک کاشی از تعداد کمکی ها بیشتر شود این روش شکست می خورد حتی بقیه FPGA بدون نقص باشد. در ضمن منابع کمکی ما در هر جای ناحیه کاری قابل استفاده هستند. هر چند که به پردازنده همزمان جداگانه برای محاسبه و مدیریت تنظیمات جایگزینی نیاز داریم. تطبیق در کار ما حداقل است یعنی تعداد زیادی نقص منطق در یک ناحیه را می توان تحمل کرد. در کار ما از تطبیق دادن گریدی استفاده شده که کمتر محدودیت اعمال می کند. در ضمن محاسبه FABRIC ها بصورتی که کار سیستم آنلاین است انجام می شود.

استفاده دوباره از PLB های نقص دار

روش ما مشخص می کند LUT عیب دار یا FF های نقص دار درون یک PLB عیب دار هستند ، یا اینکه این مد عملیاتی را با شکست مواجه می کند. قابلیت کاربرد دوباره یک PLB عیب دار در قالب رابطه سازگاری بین عیب در یک PLB و تابع سلول منطق دلخواه برای آن PLB تعریف می شود. اگر PLB نقص دار کار را درست انجام دهد می گوییم نقص با تابع سازگار است. هر نقصی با تابع تهی یک PLB کمکی سازگار است. برای یکنواختی، PLB بدون خطا که دارای نقص تهی است در نظر گرفتیم ، پس این با هر تابعی سازگار است. کاربرد پذیری یک PLB ناقص باید بطور جدا برای هر چهار تابع سلول منطق بررسی شود. یک نمونه از سازگاری ، نقص در جایی از PLB است که در تابع سلول منطق محل یابی دوباره استفاده نمی شود ، مثل یک LUT یا FF بلا استفاده ، یا نقصی که روی عمل اثر می گذارد در تابع سلول منطق محل یابی دوباره استفاده نمی شود مثلا ضرب. مثال دیگر ، یک LUT با پشته سلول حافظه در ۰ را تصور کنید. اگر LUT خطا دار یک تابع منطق ترکیباتی را پیاده کند که سلول حافظه نقص دار با منطق ۰ تنظیم شده ، آنگاه نقص بدون نیاز به هیچ پیکربندی و هیچ کمکی تحمل می شود. اگر مقدار پشته و مقدار مورد نیاز مکمل باشند ، یا اگر LUT نقص دار بصورت یک RAM ساتفاده شود آنگاه نقص با تابع سیستم سازگار نیست. اگر LUT/RAM نقص دار یک بیت آدرس یا ورودی بلا استفاده داشته باشد راه دیگر تحمل ، استفاده از نصف LUT است که حاوی سلول حافظه نقص دار نباشد. مثلا اگر LUT/RAM ۴ ورودی داشته باشیم ، و سلول منطق فقط سه ورودی باشد ، می توانیم از نصف سلول های حافظه برای پیاده کردن یک LUT سه ورودی یا یک RAM با سه بیت آدرس استفاده کنیم. این نیاز به پیکربندی مجدد محلی دارد که نصفه ی بدون عیب RAM انتخاب شود. یک خطای ناسازگار ممکن است با یک تابع دلخواه سازگار باشد اگر PLB عیب دار دارای بخش های بلا استفاده باشد. مثلا نقص ممکن است بر LUT ها اثر بگذارد.



شکل ۳-طول جابجایی های تابع منطق

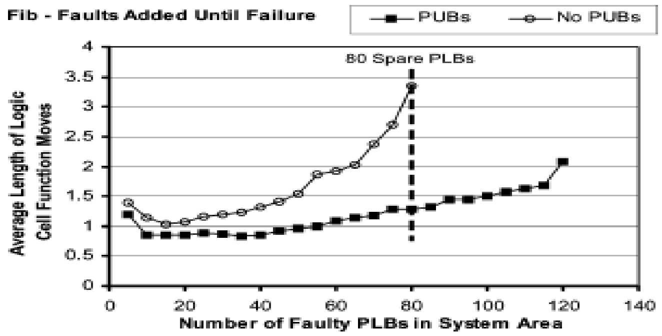


شکل ۴-ماکزیمم فرکانس عملیاتی سیستم

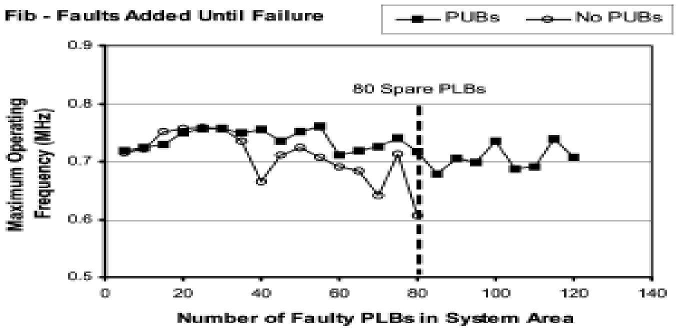
تابع سلول منطق ، LUT بلا استفاده دیگر را ترک می کند و برای تعویض LUT عیب دار متصل می شود. این نیازمند تنظیم محلی تابع سلول منطق است[20].

نتایج استفاده دوباره از PLB نقص دار

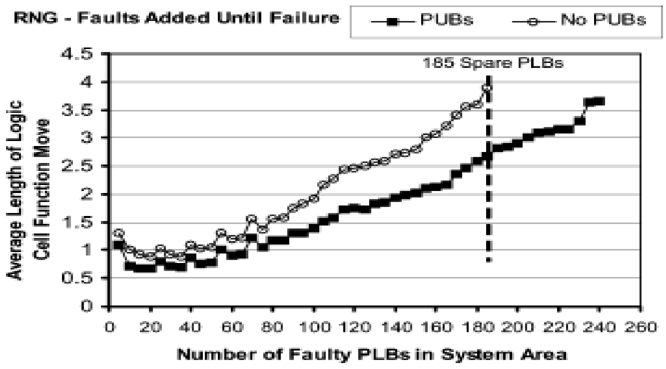
برای اینکه اثر PUBها را نشان می دهیم وقتی که برای تحمل خطای منطق تنظیم دوباره انجام می دهیم ، از داده های تجربی مدارات نمونه استفاده می کنیم. اگر نقصی در یک PLB رخ دهد ، دیگر از آن استفاده نمی شود. مدارهای تست روی ORCA 2C15 FPGA پیاده سازی شدند. ORCA 2C15 شامل آرایه ۲۰ در ۲۰ از PLB ها است. چون STAR ها از دو ستون (V-STAR) و دو سطر (H STAR) استفاده می کنند ما ۳۲۴ PLB برای کار سیستم در دسترس داریم. داده های بعدی نتایج را برای یک مدار مدوله کننده تک باندی دیجیتال DSSM ، یک مولد عدد فیبوناتچی FIB ، و یک مدار مولد عدد تصادفی RNG توصیف می کنند. لز ۳۲۴ PLB در ناحیه کاری FPGA ، DSSM از ۲۲۹ PLB (۷۰ درصد) استفاده کرد ، FIB از ۲۴۴ PLB (۷۵ درصد) ، و RNG از ۱۳۹ PLB (٪۴۳). ما بطور تصادفی مجموعه ای از PLB ها را برداشتیم و یک LUT و FF درون هر PLB بصورت عیب دار انتخاب شدند.



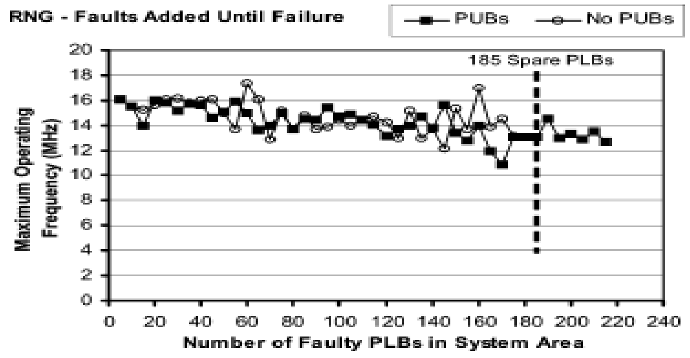
شکل ۵-طول جابجایی های تابع منطق



شکل ۶-ماکزیمم فرکانس عملیاتی سیستم



شکل ۷-طول جابجایی های تابع منطق



شکل ۸-ماکزیمم فرکانس عملیاتی سیستم

ما تعداد نقص را زیاد کردیم تا دیگر نتوانیم مدار را دوباره پیکربندی کنیم. در شکل های بالا ، نتایج تست نمونه برای تنظیم مجدد با و بدون استفاده از PLB نشان داده شده. در شکل های ۳ و ۵ و ۷ ، محور Y میانگین فاصله ای را نشان می دهد که تابع های سلول منطق نگاشته شده به نقص ها جابجا شدند طوری که همه تابع های سلول های منطق ناسازگار به محل های ساگار برای DSSM ، RNG و FIB جابجا شدند. محور X تعداد نقص های تزریق شده درون مدار را نشان می دهد. یک خط عمودی به هر گراف اضافه شد که تعداد کل PLB های کمکی در ناحیه کاری FPGA را نشان دهد. برای تعداد کم نقص های تزریق شده میانگین فاصله جابجا شده کمتر از یک PLB بود. داده ها نشان می دهند با افزایش PLB ها تعداد نقص های بیشتر قابل تحمل است. ما ۱۶۰ خطا برای DSSM ، ۱۲۰ تا برای FIB و ۲۲۰ تا برای RNG می توانیم داشته باشیم. برای همین ما تعداد نقص ها را دو برابر کمکی های موجود در حوزه سیستم در نظر گرفتیم. پس با کاربرد دوباره منطق نقص دار و PLB ها می توانیم حتی اگر تعداد PLB های نقص دار بیشتر از PLB های کمکی بودند ، سیستم را در حالت کار نرمال نگه داریم. شکل های ۴ و ۶ و ۸ اثر پیکربندی دوباره روی عملکرد سیستم برای مدارهای FIB ، DSSM و RNG را نشان می دهد. در محور Y ، ماکزیمم فرکانس عملیاتی بصورت تابعی از تعداد PLB های نقص دار تزریق شده دیده می شوند. از گراف ها ، تنزل تدریجی در عملکرد را بصورت افزایش تعداد نقص ها مشاهده می کنیم.

گذر دادن نقص های ناسازگار

اگر یک نقص با تابع سیستم سازگار نباشد ، کار سیستم باید دوباره تنظیم شود تا از نقص ناسازگار اجتناب گردد. توجه کنید PLB کمی ممکن است بدون نقص نباشد. شرط عوض کردن این است که PLB سازگار با تابع مد نظر پیدا شود. در واقع جایگزینی در گشت زنی بعدی رخ می دهد ، که تنظیم مجدد باعث می شود تابع سلول منطق تعویض روی یک منبع کمکی سازگار قرار بگیرد . با داشتن کمکی سازگار در همسایگی BLP نقص دار ، اندازه ای که این تغییرات در قالب تابع سیستم رخ می دهند حداقل می شود. از جمله تغییرات در تاخیرهای سیگنال هایی که باید دوباره مسیردهی شوند. با حرکت دادن تابع سلول منطق به محل کمکی سازگار ، سیستم کار می کند مادامی که ۱-کمکی سازگار در دسترس باشد ۲-هیچ دو تابع سلول منطقی به کمکی یکسان نیاز نداشته باشند. برای مورد دومی ، ما از تکنیک تطبیق گرید برای تنظیم دوباره استفاده می کنیم. FABRIC ها ، برای PLB های نقص دار از قبل کامپایل می شوند که سریع نگاشت آنها ذخیره شود. در سیستمی که چند نقص گذر داده می شوند، FABRIC های از قبل محاسبه شده ممکن است دیگر معتبر نباشند. پس بعد از کامپایل یک FABRIC، TREC سایر FABRIC های تحت تاثیر پیکربندی افزایشی را به روز می کند. همه این ها با شرط یک عدد نقص هستند اگر چند نقص در ناحیه یکسان روی دهد TREC باید یک FABRIC جدید محاسبه کند که هم زمان همه نقص ها را گذر دهد. این زمان بر است ولی در کار سیستم خلل وارد نمی کند چون یک STAR روی نقص ها پارک کرده است. بعد از رخ دادن چندین قص ، می شود از همه PLBها برای تعویض PLB های نقص دار استفاده کرد ، پس باید کل تابع سیستم دوباره نگاشته شود و بقیه منابع کمکی دوباره تخصیص داده شوند، تا همچنان توزیع زاپاس یکنواخت بماند. این کار توسط TREC آفلاین انجام می شود.

تخصیص از قبل منابع کمکی

روش ما نیازی به تخصیص از قبل PLB های کمکی ندارد. اگر سرعت عملیاتی سیستم کند شو ، می توانیم از محل های PLB کمکی از پیش تعیین شده در ناحیه کاری برای بهبود زمان پیکربندی مجدد، استفاده کنیم. بهرحال در انتخاب محل های کمکی از قبل تعیین شده باید دقت کرد. ما می توانیم اندازه و اثر فابریک های از پیش کامپایل شده را برای تحمل خطای افزایشی کم کنیم. ما دو استراتژی تخصیص PLB کمکی را برای ناحیه کاری FPGA توصیف می کنیم، ولی فقط این ها نیستند. اولین روش تضمین می کند برای کاربرد PLB کمتر از ۸۰ درصد ، هر تابع سلول منطق سیستم مجاور با حداقل یک PLB کمکی باشد. روش

دوم تضمین می کند برای استعمال PLB کمتر از ۹۲ درصد ، هر تابع منطق سلول بیشتر از یک PLB از یک کمکی نباشد. تعیین محل و مسیریابی اولیه منطق سیستم در ناحیه کاری $(N-2)\times(N-2)$ FPGA انجام می شود. برای روش اولی ، ۲۰ درصد PLB های ناحیه کاری را کمکی می گیریم. این زیاد نیست چون کاربرد PLB کمتر از ۸۰ درصد برای بیشتر کاربردها می باشد. استعمالهای بیشتر کار مسیریابی را سخت تر می کنند. اگر این شرط رعایت شود می توانیم هر تابع سلول منطق را مجاور حداقل یک PLB کمکی در ناحیه کاری FPGA قرار دهیم. شکل ۹ قرارگیری برای FPGA ۱۲ در ۱۲ را نشان می دهد. برای این که ابزار استاندارد حتما کمکی های مساوی توزیع شده طراحی کنند محل PLB های کمکی را با از پیش تعیین کردن توابع سلول منطق ساختگی در PLB های کمکی قبل از اجرای الگوریتم PAR استاندارد رزرو می کنیم. ما برای هر تابع، یکی از کمکی های مجاور را بعنوان قرارگیری ارجح آن انتخاب می کنیم. توجه کنید کمکی یکسان بصورت قرارگیری ارجح برای چندین تابع سلول منطق کاری طراحی می شود. در شکل ۹، روابط زیر را استفاده می کنیم که ببینیم موقعیت با مختصات ناحیه کاری (r,c) بعنوان یک کمکی رزرو شود. اگر فرض شود $\delta = r (\mathrm{mod}\ 5)$ و اگر $c(\mathrm{mod}\ 5) = (2\delta+1)(\mathrm{mod}\ 5)$ ، آنگاه محل ناحیه کاری r ، c یک محل کمکی است.

جدول ۲-جریمه عملکرد با plb های کمکی

TABLE II
PERFORMANCE PENALTY WITH SPARE PLBS

Circuit	System	System w/ STARs	System w/STARs and FT spares	% diff
Huffman	87.5 MHz	98.9 MHz	116.6 MHz	15.1
FIB	130.7 MHz	130.8 MHz	136.0 MHz	3.8
WTM	144.3 MHz	149.7 MHz	166.8 MHz	10.3
DSSM	75.1 MHz	76.6 MHz	83.3 MHz	7.2
Hilbert	73.2 MHz	87.5 MHz	90.6 MHz	3.4
RNG	49.4 MHz	56.4 MHz	60.1 MHz	6.2
FFT	117.8 MHz	120.8 MHz	123.9 MHz	2.5

کمکی ها روی لبه ناحیه کاری باید باشند تا تضمین شود plb های حاشیه بقدر کافی به یک plb کمکی تطبیق دهنده نزدیک هستند ، تا معیارهای فاصله تعریف شده رعایت شوند. این شرط پایین تر از ۸۰ درصد دسترس جذیری ناحیه سیستم است. شکل ۹ ب همین مفهوم را برای استراتژی دومی نشان می دهد.ما انتظار نداریم استعمال PLB از طرح اولیه بیشتر باشد. استفاده از کمکی های از پیش تخصیص داده شده ، زمان لازم برای تنظیم مجدد، در حضور نقص ها را ، کاهش می دهد ، یک قیمت باید برای استفاده از رویکرد STAR پرداخته شود و هزینه اضافی برای از پیش تخصیص کردن آنها لازم است. اگر هزینه در محدوده عملیاتی باشد آنگاه از پیش تخصیص دادن قابل قبول است. جدول ۲ جریمه عملکرد برای چندین مدار نمونه را نشان می دهد که در یک ORCA 2C series FPGA پیاده سازی شدند. ستونی که با کلمه سیست مشخص شده ، بدترین عملکرد را برای مدار تست ، بدون STAR ها یا کمکی های از پیش تعیین شده ، نشان می دهد. این داده مبنایی برای تعیین جریمه عملکرد افزودن STAR ها و کمکی های از قبل مشخص شده می باشد. ستون "System w/STARs" بدترین عملکرد برای مدار نمونه با STAR و بدون PLB کمکی از قبل تعیین شده را نشان می دهد. ستون "System w/STARs and FT spares" بدترین فرکانس عملیاتی برای تابع سیستم با کمکی های از پیش تعیین شده برای هر ده موقعیت STAR را نشان می دهد. ستون آخری اختلاف درصد در فرکانس عملیاتی بین مدارات بدون و با کمکی های از پیش تعیین شده را نشان می دهد. برای مدارهای تست شده ، محدوده اختلاف بین ۲,۵ و ۱۵,۱ درصد است. کمکی های از پیش تخصیص داده شده فقط زمان برای پیکربندی دوباره افزایشی برای تعداد نقص های کم را پایین می آورند و الزام نیستند. برای ۱۰۰ درصد کاربرد و استفاده از PLB ، ۲۵ درصد منطق باید بدون استفاده کنار گذاشته شود.

نقص های منطق چندگانه

وقتی دو تابع سلول منطق ، محل PLB ترجیح داده شده همسان دارند ، برخورد وجود دارد اگر هر دو نیازمند نگاشت دوباره به یک محل کمکی باشند. وقتی این رخ دهد ، ما از استراتژی تطبیق دادن استفاده می کنیم که توابع سلول منطق را با محل های کمکی سازگار تطبیق دهیم. تطبیق MINMAX برای انطباق دادن چند تابع سلول منطق به محل های کمکی سازگار مورد استفاده قرار می گیرد طوری که ماکزیمم فاصله بین هر تابع سلول منطق و محل کمکی متناظرش کمینه می شود. از این رویکرد برای کاهش فاصله جابجایی های تابع در حین تنظیمات دوباره متحمل خطا و برای کاهش اثرات منفی روی سرعت سیستم استفاده می کنیم.

در شکل ۱۰ از تطبیق MINMAC برای انتساب توابع سلول منطق ناسازگار به محل های PLB کمکی در ناحیه کاری FPGA استفاده شده

است. سه تابع وجود دارند که با نقص ها در محل های فعلی خود ناسازگار هستند. اگر طول MINMAX را L=1 تنظیم کنیم ، هر تابع سلول منطق ناسازگار را به کمکی مجاورش تطبیق می دهیم ، ولی دو سلول بدون تطبیق می مانند. با افزایش دادن طول MINMAX به L=2 ، هر تابع سلول منطق ناسازگار را به یک کمکی سازگار تطبیق می دهیم. در شکل ۱۰ ب ، محل های انطباق داده شده برای هر تابع ناسازگار برای L=2 نشان می دهد. شکل ۱۰ ج ، مدار را بعد از جابجا شدن هر تابع به محل کمکی اش نشان می دهد. ما از استراتژی تطبیق MINMAX استفاده کردیم ، وقتی که داده های PUBs/No PUBs در شکل های ۳-۸ مد نظر ما بودند. وقتی یک برخورد روی انتساب کمکی روی داد ، از استراتژی MINMAX برای کم کردن بدترین حالت فاصله بین تابع سلول منطق محل آن بعد از تحمل خطا، استفاده کردیم. داده ها در شکل های ۳ و ۵ و ۷ دو رویه را نشان می دهد ابتدا ، طول متوسط جابجایی های تابع سلول منطق (محور Y شکل های ۳ و ۵ و ۷) کمتر از وقتی که ما از PUB ها استفاده می کنیم. این تفاوت ایجاد می کند چون PUB ها منابع کمکی بیشتری نسبت به استفاده از PLB های بدون نقص فراهم می کنند. بطور متوسط ، یک نقص به یک کمکی سازگار با PUBها نزدیک تر می شود. دوم ، برای مدارات فیبوناتچی و DSSM متراکم تر ، ماکزیمم فرکانس عملیاتی (محور Y شکل های ۴ و ۶) برای PUB ها بیشتر از مورد NO PUB است. سرعت عملیاتی مدار هر چه بیشتر باشد بهتر است ، پس این از کاربرد PUB ها طرفداری می کند. در ضمن کاربرد الگوریتم تطبیق برای کاهش تنزل فرکانس عملیاتی سیستم در حضور خطای بسیار توجیه می شود [19].

نتایج عملی مورد بحث

سرعت عملکرد یکی از نقاط تحمل پذیری خطای آنلاین بلوکهای منطق FPGA است سرعت عملکرد آن بصورت سخت افزاری طراحی شده اند و این امکان را دارند که عملیات ها بصورت موازی انجام دهند این ویژگی ها باعث می شوند تا FPGA ها با سرعت بالا و زمان پاسخ کمتری عمل کنند انعطاف پذیری ها بطور عمده از منابع قابل برنامه ریز مانند فیلیپ فلاپ ها استفاده می کنند تا بصورت LUT بتوانند برای برنامه های مختلف منطق های متنوع و انعطاف پذیر شوند توان مصرفی کم در مقایسه با روش های دیگر از تحمل خطا مانند تکنیک های تشخیص خطا جبران خطا توان مصرفی کمتری دارند استفاده می شود ECC(Error correction Code) تکنیک های مبتنی بر خطا بر رمزگذاری استفاده می شود طراحی و پیاده سازی FPGA پیچیدگی خاص خود را دارد تحمل پذیری خطای آنلاین برای FPGA الگوریتم های تشخیص و جبران خطا باعث افزایش پیچیدگی است استفاده از تکنیک تحمل خطا به افزایش منابع مانند حافظه و منابع محاسباتی منجر می شود و از انعطاف پذیری و کارای کاسته و محدودیت هایی در طراحی و استفاده از آنها ایجاد می کنددو با افزایش لوجیک ها logic ها بلوک منطق تست و تحلیل خطا نیز افزایش می یابد و نیازمند تست پیچیده تر و هزینه بیشتر برای تست و تحلیل خطا است و استفاده از الگوریتم های مرتبط روشی مناسب برای پوشش است[20].

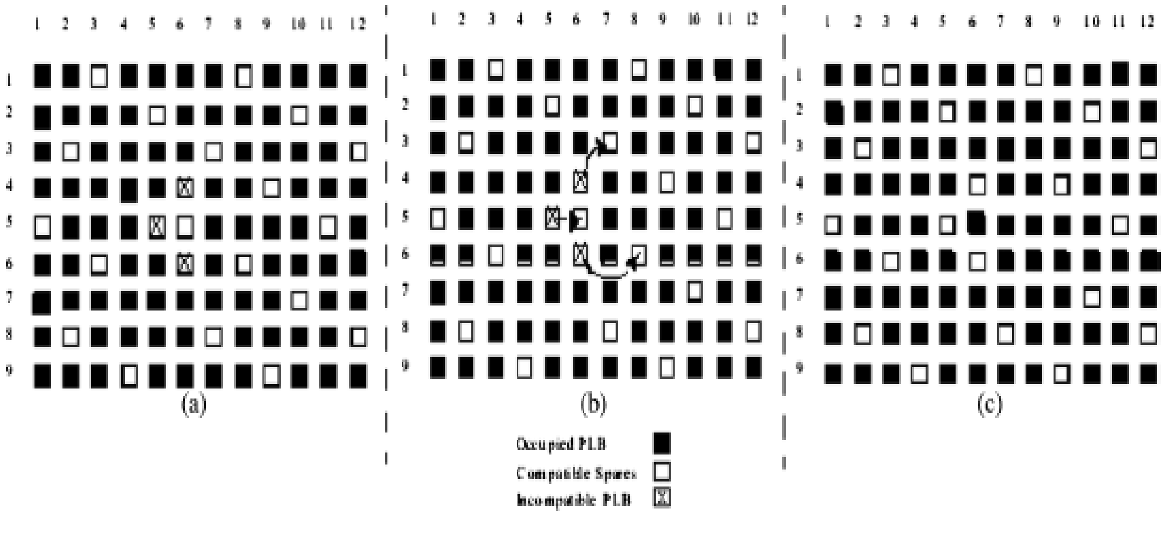
جدول ۳ مقایسه انواع FPGAها

Spartan-3 FPGA Family								
Spartan-3	XC3S50	XC3S200	XC3S400	XC3S1000	XC3S1500	XC3S2000	XC3S4000	XC3S5000
Spartan-3L	—	—	—	XC3S1000L	XC3S1500L	—	XC3S4000L	—
Spartan-3 EasyPath	—	—	—	—	XCE3S1500	XCE3S2000	XCE3S4000	XCE3S5000
System Gates	50K	200K	400K	1000K	1500K	2000K	4000K	5000K
Logic Cells	1,728	4,320	8,064	17,280	29,952	46,080	62,208	74,880
Block RAM Bits	72K	216K	288K	432K	576K	720K	1,728K	1,872K
Distributed RAM Bits	12K	30K	56K	120K	208K	320K	432K	520K
DCMs	2	4	4	4	4	4	4	4
Multipliers	4	12	16	24	32	40	96	104
I/O Standards	24	24	24	24	24	24	24	24
Max Single Ended I/O**	124	173	264	391	487	565	712	784
Package and I/O Offerings								
	XC3S50	XC3S200	XC3S400	XC3S1000	XC3S1500	XC3S2000	XC3S4000	XC3S5000
VQ100 14 x 14 mm	63	63						
TQ144 20 x 20 mm	97	97	97					
PQ208 28 x 28 mm	124	141	141					
FT256 17 x 17 mm		173	173	173*				
FG320 23 x 23 mm			221	221*	221*			
FG456 23 x 23 mm			264	333*	333*			
FG676 27 x 27 mm				391	487*	489		
FG900 31 x 31 mm						565	633*	633
FG1156 35 x 35 mm							712	784

جدول۴. مقایسه روش با روش های قبلی

استفاده از روش کد گذاری و تشخیص خطا	این روش از روش های کدگذاری و تشخیص خطا (parity coding) . هم کدگذاری یا کدگذاری هم وزن (weighted coding) استفاده می شود این روش ا با افزودن بیت های کدگذاری به اطلاعات ورودی قادر به تشخیص و جبران خطاهای مشخصی هستند.
۲-استفاده از تکنیک های مبتنی بر دیپلکس	در این روش بلوک طراحی می شود بطوریکه بخشی از بلوک به صورت اصلی و بخشی دیگر بصورت رزو شده استو خطا در بخش اصلی رخ میدهد بخش رزرو خطا را تشخیص میدهد و بعنوان جایگزین فعال می شود.
۳-استفاده از روش های تفکیک خطا	در این روش بطور فیزیکی به FPGAبلوک به چندین بخش تقسیم می شود و هر بخش بصورت مستقل عمل می کند با استفاده از روش های تفکیک خطا اگر یکی از بخش ها خطا داشته باشد بخش دیگر قادر به تولید خروجی صحیح است.
۴-استفاده از روش تحلیل خطا	در این روش برای تشخیص و تحلیل خطا از روش های تحلیل FPGAآنلاین در بلوک error و تحلیلگرهای خطا detector analyzers استفاده می شود.
۵-در این روش از روش های تشخیص خطا مبتنی بر پوشش	DFT(Design for Testability) که قادر هستند به تشخیص خطا با استفاده از الگوریتم تست تولید FPGA پوششی در طراحی بلوک شوند.
۶-در FPGA روش کاما	در FPGA این روش اطلاعات ورودی به بلوک بصورت مکرر تکرار می شود و خروجی های متفاوت مقایسه می شوند. با استفاده از این تکنیک خروجی نادرست مشخص می شود و از آن صرفه نظر می شود.

ما تحمل عیب منطق را در PRT FPGA ها بررسی کردیم. ما چند مفهوم جدید برای FT منطق ارائه کردیم. چون تولید دوباره همه پیکربندی های FT با یک یا هر دو STAR پارک شده روی نقص ها همراه می شود تابع سیستم به اجرای خود ادامه می دهد. وقتی از STAR های گشت زن برای تست و تشخیص نقص ها در FPGA ها استفاده کردیم هیچ زمان خرابی اضافی برای تحمل خطا ایجاد نشد. همه تنظیمات FT و تغییرات لازم در زمان کار سیستم بصورت آنلاین رفع و رجوع می شوند. هر جا لازم باشد از منبع قابل برنامه نویسی نقص دار استفاده می کنیم. برای منطق ، ما از PLB نقص دار استفاده می کنیم اگر این با کار تابع سازگار باشد. اگر این طور نبود منطقی پیدا می کنیم که با نقص سازگاری داشته باشد. اگر آن شکست بخورد از PLB ناقص بعنوان یک PUB استفاده می کنیم ، و از بخش های غیر ناقص PLB استفاده می نماییم. این روش بهبود بزرگی نسبت به روش های قبلی ایجاد می کند.



شکل ۱۰-مثال پیکربندی دوباره MINMAX ناحیه کاری برای ناسازگاری نقص-چندگانه برای سه خطای ناسازگار و نتایج تطبیق MINMAX و تابع سیستم دوباره تنظیم شده

ما نشان دادیم حتی تعداد نقص ها بیشتر از منابع PLB کمکی باشد باز هم سیستم به کار خود ادامه می دهد. ما از منابع کمکی از پیش تخصیص داده شده هم می توانیم استفاده کنیم. روش STAR نیاز به جریمه عملکرد نسبت به سرعت عمل کلاک سیستم دارد. استراتژی های تخصیص منبع ما تضمین می کند کمکی محلی در مجاورت وجود دارد اگر که کاربرد منبع زیر ۸۰ درصد باشد. اگر کمکی های محلی در دسترس نبودند یا گروهی از PLB ها ناقص شدند ، از استراتژی تطبیق گرید MIMMAX استفاده می کنیم که همه محل های سلول منطق را به منابع سازگار منتسب کند طوری که مقدار پیکربندی دوباره به حداقل برسد. کلاک قابل برنامه نویسی است تا از اول در سریع ترین مقدار خود باشد و در مسیرهای بحرانی به حد لازم کم شود. در بیشتر جاها سرعت کلاک کمتر نتیجه بهتری می دهد. هر دو روش محلی و سراسری برای تحمل خطا و استفاده از آنها همراه STAR BIST ، بر محدودیت های افزونگی سطر و ستون در کارهای قبل ، و کاشی کاری یا رویکردهایی مثل شیفت دادن دانه دانه ای چیره می شود. ما محدود به تعداد ثابت PLB ها در ناحیه محلی نیستیم مثل یک کاشی ، سلول یا سطر یا ستون. ولی با استفاده از فابریک های از قبل کامپایل شده در یک محل کوچک می توانیم کنترل بیشتری روی افزونگی داشته باشیم. ما با کمک روش تطبیق گرید حتی اگر کمکی ها در ناحیه محلی نبودند می توانستیم از کمکی ها استفاده کنیم. ما روش خود را روی ORAC 2C15 GPGA نشان دادیم. اما می توانیم روی خانواده های FPGA دیگر هم امتحان کنیم. برای توضیح روش مان ، ما ابزارهای CAD تجاری را برای نقشه کشی و نگاشت مجدد مدار به کار بردیم. یک ماسک با منطق ساختگی و مسیریابی ایجاد شده که از نواحی STAR جلوگیری شود. وقتی یک ماسک برای یک محل STAR ایجاد می شود ایجاد ماسک ها برای نواحی STAR دیگر بطور خودکار رخ می دهند. بیشتر تکنیک ها با کاربردهای آفلاین سر و کار داشتند. ولی ویژگی RTR در آنها پشتیبانی نمی شد. ولی روش ما برای هر سیستم قابل کاربرد است از جمله بهبود بهره ساخت سیستم بر تراشه آفلاین.

نتیجه گیری نهایی و کارهای آتی

تحمل خطای آنلاین برای FPGA می تواند بلوک منطق را بهبود و سرعت عملکرد و انعطاف پذیری را افزایش دهد با امکان انجام عملیات بصورت موازی و استفاده از منابع قابل برنامه ریزی سرعت بالاتر و زمان پاسخ کمتر خواهد بود استفاده از تکنیک تشخیص و جبران خطا مبتنی بر خطای رمزگذاری باعث کاهش توان مصرفی می شود با این حال تحمل خطای آنلاین باعث FPGAآنلاین در افزایش پیچیدگی و محدودیت های طراحی شده و نیازمند تست پیچیده تر و هزینه بیشتر است بنابراین استفاده از الگوریتم مناسب و تست دقیق می تواند بهبود قابل توجهی در تحمل خطای آنلاین برای بلوک FPGAداشته باشدبطور کلی مقاله بررسی کرد تحمل خطای آنلاین FPGAتأثیر مثبتی بر عملکرد و انعطاف پذیری آنها داشته و استفاده از تکنیک های تشخیص خطا مبتنی بر رمزگذاری و ECCالگوریتم های مناسب می تواند به کاهش مصرف توان و بهبود عملکرد خطا کمک کند با این حال تحمل خطای آنلاین در بلوک های منطق ممکن است باعث افزایش FPGA پیچیدگی و محدودیت های طراحی شود و نیازمند تست دقیق تر و هزینه بیشتر است در کارهای آتی به بررسی الگوریتم های پوشش تست مناسب برای تحمل خطای آنلاین در بلوک های منطق FPGA بهبود و بهینه سازی تشخیص و جبران خطا و همچنین بررسی روش های جدید برای کاهش پیچیدگی و محدودیت های طراحی در تحمل خطای آنلاین در بلوک ها وارزیابی و عملکرد و مقایسه تحمل خطا آنلاین در بلوک با روش FPGA های منطق تحمل خطا منجر به بهبود قابل توجهی می شود

تعارض منافع

«هیچ‌گونه تعارض منافع توسط نویسندگان بیان نشده است»

منابع و مآخذ

[1] Marcos Santana Farias, Nadia Nedjah, Paulo Victor R. de Carvalho, "Resilient Hardware Design for Critical Systems", 2019 IEEE 10th Latin American

[2] Kibum Lee, S. Simon Wong, "Fault-Tolerant FPGA with Column-Based Redundancy and Power Gating Using RRAM", IEEE Transactions on Computers, vol.66, no.6, pp.946-956, 2017.

[3] Martin A. Trefzer, David M. R. Lawson, Simon J. Bale, James A. Walker, Andy M. Tyrrell, "Hierarchical Strategies for Efficient Fault Recovery on the Reconfigurable PAnDA Device", IEEE Transactions on Computers, vol.66, no.6, pp.930-945, 2017

[4] Gert Schley, Atefe Dalirsani, Marcus Eggenberger, Nadereh Hatami, Hans-Joachim Wunderlich, Martin Radetzki, "Multi-Layer Diagnosis for Fault-Tolerant Networks-on-Chip", IEEE Transactions on Computers, vol.66, no.5, pp.848-861, 2017.

[5] Hongyan Zhang, Lars Bauer, Michael Andreas Kochte, Eric Schneider, Hans-Joachim Wunderlich, Jörg Henkel, "Aging Resilience and Fault Tolerance in Runtime Reconfigurable Architectures", IEEE Transactions on Computers, vol.66, no.6, pp.957-970, 2017.

[6] S. Aishwarya, G. Mahendran, "Multiple bit upset correction in SRAM based FPGA using Mutation and Erasure codes", 2016 International Conference on

[15] Gabriel L. Nazar, Luigi Carro, "Fast error detection through efficient use of hardwired resources in FPGAs", 2012 17th IEEE European Test Symposium (ETS), pp.1-6, 2012

[16] Emmert JM, Stroud CE, Abramovici M. Online fault tolerance for FPGA logic blocks. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2007 Feb;15(2):216-26.

[17] Abramovici M, Stroud CE, Emmert JM. Online BIST and BIST-based diagnosis of FPGA logic blocks. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2004 Dec;12(12):1284-94.

[18] Ruiz-Rosero J, Ramirez-Gonzalez G, Khanna R. Field programmable gate array applications—A scientometric review. Computation. 2019 Nov 11;7(4):63.

[19] Zhang H, Bauer L, Kochte MA, Schneider E, Wunderlich HJ, Henkel J. Aging resilience and fault tolerance in runtime reconfigurable architectures. IEEE Transactions on Computers. 2016 Oct 11;66(6):957-70.

[20] McWilliam R, Khan S, Farnsworth M, Bell C. Zero-maintenance of electronic systems: Perspectives, challenges, and opportunities. Microelectronics Reliability. 2018 Jun 1;85:122-39.

سپیده گوهری

سرکار خانم سپیده گوهری در سال ۱۳۹۴ مدرک کارشناسی ناپیوسته نرم افزار را از دانشکده مهندسی کامپیوتر دانشگاه آزاد اسلامی واحد تهران شمال را دریافت نموده اند. ایشان در سال ۱۴۰۰ مدرک کارشناسی ارشد معماری سیستم های کامپیوتری را از دانشکده برق و کامپیوتر دانشگاه خواجه نصیرالدین طوسی اخذ نموده و دوره های فنی مرتبط را در مقطع کارشناسی ارشد در دانشگاه علم صنعت نیز گذرانده اند. ایشان اکنون در دوره دکتری تخصصی در دانشکده مهندسی کامپیوتر دانشگاه آزاد اسلامی واحد تهران شمال مشغول به تحصیل و تدریس در مقطع دورس کارشناسی می باشند.

Advanced Communication Control and Computing Technologies (ICACCCT), pp.202-206, 2016.

[7] Ahmad Alzahrani, Ronald F. DeMara, "Hypergraph-Cover Diversity for Maximally-Resilient Reconfigurable Systems", 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, pp.1086-1092, 2015

[8] Chi-Chou Kao, "Performance-Oriented Partitioning for Task Scheduling of Parallel Reconfigurable Architectures", IEEE Transactions on Parallel and Distributed Systems, vol.26, no.3, pp.858-867, 2015.]

[9] David M. R. Lawson, James Alfred Walker, Martin A. Trefzer, Simon J. Bale, Andy M. Tyrrell, "Evolving hierarchical low disruption fault tolerance strategies for a novel programmable device", 2014 IEEE International Conference on Evolvable Systems, pp.77-84, 2014

[10] Reza Ramezani, Yasser Sedaghat, "Scheduling periodic real-time hardware tasks on dynamic partial reconfigurable devices subject to fault tolerance", 2014 4th International Conference on Computer and Knowledge Engineering (ICCKE), pp.479-484, 2014..

[11] Martin A. Trefzer, Andy M. Tyrrell, "Improved fault-tolerance through dynamic modular redundancy (DMR) on the RISA FPGA platform", 2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pp.39-46, 2014

[12] Hasan Baig, Jeong-A Lee, Zahid Ali Siddiqui, "A Low-Overhead Multiple-SEU Mitigation Approach for SRAM-based FPGAs with Increased Reliability", IEEE Transactions on Nuclear Science, vol.61, no.3, pp.1389-1399, 2014.

[13] Han Zhang, Yansong Wu, Sen Zhao, Lei Zhao, "Fine granularity optimal spare allocation and replacement in reconfiguration system", 2013 Sixth International Conference on Advanced Computational Intelligence (ICACI), pp.177-182, 2013..

[14] B. Harikrishna, S. Ravi, "A survey on fault tolerance in FPGAs", 2013 7th International Conference on Intelligent Systems and Control (ISCO), pp.265-270, 2013

COPYRIGHTS

©2021 The author(s). This is an open access article distributed under the terms of the Creative Commons Attribution (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, as long as the original authors and source are cited. No permission is required from the authors or the publishers.