Journal of Soft Computing and Information Technology (JSCIT)
Babol Noshirvani University of Technology, Babol, Iran
Journal Homepage: jscit.nit.ac.ir
Volume 10, Number 3, Autumn 2021, pp. 47-59
Received: 27/11/2020, Revised: 23/05/2021, Accepted: 02/07/2021

Introducing a Multi-Agent Strategy for Designing Reconfigurable Systems to Increase Their Flexibility and Lifetime

Hossein Hosseinzadeh[1], Hakem Beitollahi[2*] and Nasser Mozayani[3]
1- School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran.
2*- School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran.
3- School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran.
[1]h_hoseinzade@comp.iust.ac.ir, [2*] beitollahi@iust.ac.ir, and [3] mozayani@iust.ac.ir

Corresponding author's address: Hakem Beitollahi, Faculty of Computer Engineering, Iran University of Science and Technology, Tehran, Iran.

Abstract- One of the prominent features of the reconfigurable systems is flexibility. The flexibility of these systems depends largely on how they are reconfigured. There are many methods for reconfiguration that are divided into static reconfiguration, partial dynamic reconfiguration, and full dynamic reconfiguration. Of the proposed methods, the methods that can be reconfigured online are of particular importance. In this paper, using the redistributive properties of the systems, we designed these types of systems based on the organization of multi-agent systems. The proposed method introduces an intelligent reconfigurable system that changes the configuration of the system by the interaction of the agents with each other. One of the important goals of this new method is to extend the life-time of the reconfigurable system. In this paper, the proposed method is simulated using VHDL hardware description language and its synthesis is done using Design Compiler software. The simulation results show that the service life-time of the reconfigurable system is increased depending on the number of resources consumed on the chip design and the system also has good flexibility due to its intelligence and reconfigurability at runtime. Our results show that maximum usage of the logic cells on system has decreased by 1.05 to 4.23 times than **Transmap** and **Resource Partitioning and Application Scheduling (RPAS)** methods. So it causes to increase the life time of the system. Our proposed method has data transfer overhead between agents that our results show 1458, 1218 and 770 bits for 32 bit ALU, 64 to 6 encoder and 64 bit barrel shifter respectively.

Keywords- *Reconfigurable systems, multi-agent systems, online reconfiguration*

مجله علمی پژوهشی رایانش نرم و فناوری اطلاعات
دانشگاه صنعتی نوشیروانی بابل
صفحه مجله: jscit.nit.ac.ir
جلد ۱۰، شماره ۳، پاییز ۱۴۰۰، صفحه ۴۷-۵۹
دریافت: ۱۳۹۹/۰۹/۰۷، بازنگری: ۱۴۰۰/۰۳/۰۲، پذیرش: ۱۴۰۰/۰۴/۱۱

ارائه‌ی یک راهبرد چندعامله برای طراحی سیستم‌های قابل بازپیکربندی جهت افزایش انعطاف‌پذیری و طول عمر

حسین حسین زاده[۱]، حاکم بیت الهی*[۲]، ناصر مزینی[۳]
۱- دانشکده مهندسی کامپیوتر، دانشگاه علم و صنعت ایران، تهران، ایران.
*۲- دانشکده مهندسی کامپیوتر، دانشگاه علم و صنعت ایران، تهران، ایران.
۳- دانشکده مهندسی کامپیوتر، دانشگاه علم و صنعت ایران، تهران، ایران.
[1]h_hoseinzade@comp.iust.ac.ir, [2*] beitollahi@iust.ac.ir, [3] mozayani@iust.ac.ir

* نشانی نویسنده مسئول: تهران، میدان رسالت، خیابان هنگام، دانشگاه علم و صنعت ایران، دانشکده مهندسی کامپیوتر، حاکم بیت الهی.

چکیده- یکی از ویژگی‌های مهم و برجسته‌ی سیستم‌های قابل بازپیکربندی، انعطاف‌پذیری می‌باشد. انعطاف‌پذیری این دسته از سیستم‌ها وابستگی زیادی به نحوه‌ی بازپیکربندیشان دارد. روش‌های زیادی در راستای بازپیکربندی ارائه شده‌اند که به دسته‌های بازپیکربندی ایستا، بازپیکربندی پویای جزئی و بازپیکربندی پویای کامل تقسیم‌بندی می‌شوند. از بین روش‌های ارائه شده روش‌هایی که قابلیت بازپیکربندی به صورت برخط را داشته باشند، از اهمیت ویژه‌ای برخوردار هستند. در این مقاله با استفاده از خاصیت توزیع-پذیری سیستم‌های قابل بازپیکربندی، این دسته از سیستم‌ها را برمبنای سازمان سیستم‌های چندعامله طراحی کرده‌ایم. در روش پیشنهادی یک سیستم قابل بازپیکربندی هوشمند ارائه شده است که در زمان اجرای سیستم، عامل‌های هوشمند با تعامل و مذاکره با یکدیگر، بصورت خودمختار (مستقل از ابزار سنتز) پیکربندی سیستم را تغییر می‌دهند. یکی از هدف‌های مهم این روش جدید، افزایش طول عمر سیستم قابل بازپیکربندی می‌باشد. در این مقاله، شبیه‌سازی روش پیشنهادی با استفاده از زبان توصیف سخت‌افزار و سنتز آن با استفاده از نرم افزار Design Compiler انجام گردیده‌است. نتایج شبیه‌سازی نشان می‌دهد که حداکثر میزان فعالیت سلول‌ها با توجه به الگوها و الگوریتم‌های مورد ارزیابی، به میزان ۱/۰۵ الی ۴/۲۳ برابر نسبت به روش‌های Transmap و Resource Partitioning and Application Scheduling (RPAS) کاهش یافته و در نتیجه طول عمر سیستم قابل بازپیکربندی افزایش یافته است. همچنین روش پیشنهادی با توجه به تعامل بین عامل‌های هوشمند، شامل سربار تبادل پیام می‌باشد که مقادیر آن برای الگوریتم‌های ALU ۳۲ بیتی، رمزکننده ۶۴ به ۶ و Barrel Shifter ۶۴ بیتی به ترتیب مطابق الگوهای تحت ارزیابی به طور میانگین ۱۴۵۸، ۱۲۱۸ و ۷۷۰ بیت بدست آمده‌است.

واژه‌های کلیدی: تراشه‌های قابل بازپیکربندی، سیستم‌های چندعامله، سالخوردگی، بازپیکربندی برخط

۱- مقدمه

به دلیل نیاز به افزایش سرعت و کارایی همراه با کاهش هزینه‌های سخت‌افزاری جهت ساخت دوباره و همچنین افزایش انعطاف پذیری، سیستم‌های قابل بازپیکربندی از اهمیت ویژه‌ای جهت پیاده‌سازی سیستم‌های دیجیتال برخوردار شده‌اند [۱]. سیستم-های قابل بازپیکربندی می‌توانند در زمان‌های مختلف به معماری-های متنوعی تغییر شکل می‌یابند. یک سیستم قابل بازپیکربندی می‌تواند یک معماری ساده مانند تولید پالس ساعت[۲] تا یک سیستم پیچیده مانند شبکه‌ای بزرگ از پردازنده‌ها که به یکدیگر متصل هستند، را پیاده‌سازی کند.

به طور کلی سیستم‌های قابل بازپیکربندی از پیکربندی در سطح بیت مانند Field Programmable Gate Array (FPGA) تحت عنوان ریزدانه[۳] و سطح کلمه تحت عنوان درشت‌دانه[۴] را در بر می-گیرند. این سیستم‌ها از تعداد زیادی سلول‌های منطقی[۵] تشکیل شده‌اند که هر کدام از این سلول‌ها می‌تواند بخشی از یک معماری را پیاده‌سازی نماید. یکی از مسائل بسیار مهم، میزان انعطاف-پذیری این سیستم‌ها می‌باشد. انعطاف‌پذیری نحوه‌ی بازپیکربندی سلول‌های منطقی و قابلیت بازپیکربندی سیستم به صورت برخط را شامل می‌شود [۲]. در بسیاری از سیستم‌ها مانند هواپیماهای بدون‌سرنشین، ماهواره‌ها، پردازشگرهای سیگنال‌های راداری و غیره، نیاز است تا بخشی از پیکربندی‌های این سیستم‌ها در حین اجرا تغییر کند[۳]. در روش‌هایی که تاکنون برای معماری سیستم‌های قابل بازپیکربندی ارائه شده‌اند، از بخشی از سلول‌ها برای پیاده‌سازی طرح‌های مد نظر استفاده می‌شود در حالی که سلول‌های دیگر در حالت بیکار و غیرفعال به سر می‌برند که این امر موجب سالخوردگی سلول‌هایی می‌شود که دائماً در حالت فعال به سر می‌برند که در نتیجه کاهش طول عمر کل سیستم را در پی دارد. زیرا یکی از پارامترهای تاثیرگذار در طول عمر، میزان استفاده[۶] از سلول‌ها در حالت فعال می‌باشد[۴] [۵].

در روش پیشنهادی سعی داریم تا با استفاده از خاصیت توزیع پذیری سیستم‌های قابل بازپیکربندی و با در نظر گرفتن سلول-های منطقی به عنوان عامل هوشمند، یک سیستم قابل بازپیکربندی خودمختار و مستقل از ابزار سنتز بر پایه‌ی سیستم-های چندعامله را پیاده‌سازی کنیم. در واقع، ابزار سنتز فقط بار اول تراشه را پیکربندی می‌نماید و دفعات بعد به نسبت کاهش سالخوردگی، تراشه به صورت خودمختار توسط عامل‌های هوشمند خودش را بازپیکربندی مجدد می‌نماید. لذا قبل از بیان روش پیشنهادی لازم می‌دانیم تا توضیح مختصری از سیستم‌های چندعامله را داشته باشیم.

عامل هوشمند[۷] موجودیتی است که می تواند به واسطه‌ی حسگرها محیط خود را درک کرده و عملی را در محیط انجام دهد[۶]. عامل‌های هوشمندی که در یک محیط استقرار یافته اند، باید بتوانند با یکدیگر ارتباط داشته باشند و برای انجام وظایف همکاری کنند. یک سیستم چندعامله از چندین عامل هوشمند تشکیل شده است که سعی می‌کنند مسائلی را که حل آنها برای یک سیستم متمرکز و یکپارچه مشکل و گاهی غیرممکن است، به کمک یکدیگر حل کنند[۷]. در سیستم‌های چندعامله هر یک از عامل‌ها سطحی از خودمختاری[۸] را با توجه به تعاملات و بازخوردش از محیط دارا هستند. خودمختاری عبارت است از توانایی عامل در کنترل عمل‌ها و حالات درونی خود، بدون مداخله عوامل خارجی[۷] [۸].

در یک سیستم چندعامله، عامل‌های موجود در محیط ممکن است در راستای هدف مشخص و مشترک و یا اهداف جداگانه با یکدیگر در تعامل[۹] باشند. در حقیقت تعامل به دلیل وجود محدودیت در منابع مشترک، وابستگی داخلی وظایف عامل‌ها و انجام اهداف جمعی امری اجتناب ناپذیر است.کارایی کلی یک سیستم چندعامله، یک کارایی ظهوریافته است که از توانایی تک تک عامل‌ها بیشتر است. در مسائلی که به طور ذاتی توزیع شده هستند، جهت رسیدن به یک هدف یا انجام یک کار بزرگ، می توان آن هدف را به زیرهدف‌ها تقسیم کرد تا هر عامل بتواند انجام یک قسمت را به عهده بگیرد و یا عامل‌های دارای هدف مشترک تشکیل گروه داده و برای انجام آن به تعامل با یکدیگر بپردازند[۹][۱۰].

در ادامه‌ی این مقاله، ابتدا در بخش ۲ به مروری بر روش‌های موجود بازپیکربندی در سیستم‌های قابل بازپیکربندی خواهیم پرداخت. پس از آن در بخش ۳ روش پیشنهادی را ارائه خواهیم کرد. در بخش ۴، به ارزیابی و مقایسه آن با روش‌های موجود خواهیم پرداخت و سرانجام در بخش ۵ مقاله نتیجه‌گیری می‌شود.

۲- مروری بر کارهای گذشته

تاکنون روش‌های متعددی جهت پیکربندی سیستم‌های قابل بازپیکربندی ارائه شده‌اند که هر کدام از روش‌ها سعی در بهبود پارامترهایی از جمله انعطاف‌پذیری، مصرف توان، مصرف انرژی، سربار مساحت[۱۰]، زمان اجرا، تحمل پذیری اشکال را دارند[۱۴][۱۳][۱۲][۱۱]. به طور کلی می‌توان روش‌های بازپیکربندی را به دو دسته‌ی ایستا و پویا تفکیک کرد[۱۴]. در تاریخچه‌ی سیستم‌های قابل بازپیکربندی، ابتدا روش‌های بازپیکربندی به صورت ایستا ارائه شده بودند که بعدها به دلیل اهمیت موضوع قابلیت انعطاف‌پذیری، روش‌های پویای بازپیکربندی جهت افزایش انعطاف‌پذیری سیستم‌های قابل بازپیکربندی ارائه شده‌اند.

۲-۱- روش‌های بازپیکربندی ایستا

روش‌هایی در زمینه‌ی نگاشت[۱۱] و بازپیکربندی ایستا ارائه شده‌اند که یکی از معروف‌ترین آن‌ها الگوریتم نگاشت CHORTLE است که در سال ۱۹۹۰ با هدف مصرف کمینه‌ی منابع موجود در تراشه ارائه شده بود [۱۵]. در سال ۱۹۹۱ حالت پیشرفته‌ی این الگوریتم تحت عنوان CHORTLE-CRF ارائه شد که علاوه بر کمینه ساختن مصرف منابع، سعی در به حداقل رساندن مصرف پورت-

های ورودی و خروجی را نیز داشت[۱۶]. پس از روش CHORTLE ، روشی با هدف کمینه کردن زمان تاخیر اجرای برنامه‌ها بر روی سیستم‌های قابل بازپیکربندی ارائه شد که FLOW MAP نام دارد. این روش با کمینه ساختن عمق شبکه‌ی گراف سلول‌های منطقی، تاخیر را کمینه می‌کند [۱۷]. اساس روش‌های ارائه شده، نمودار تصمیم دودویی[۱۲] است. با بیان توابع منطقی به صورت نمودار تصمیم دودویی، می‌توان به اشغال فضای کمتر تراشه جهت ذخیره‌سازی داده‌ها کمک کرد. در سال‌های اخیر روش‌هایی جهت نگاشت و بازپیکربندی ارائه شده‌اند که در آن‌ها روش‌های CHORTLE و FLOWMAP با یکدیگر تلفیق می‌گردند[۱۸]. استراتژی روش‌های ارائه شده بر مبنای کمینه-سازی پارامترهای مختلفی از جمله مدت تاخیر، مصرف منابع و مصرف توان استوار است. در هر یک از روش‌های ارائه شده، تطبیق پیکربندی بلوک‌های منطقی تجزیه شده با مدارهای تعبیه شده در سطح تراشه حیاتی است. در [۱۹] روشی ارائه شده است که با تجزیه‌ی چندباره‌ی نمودار تصمیم دودویی یک بازپیکربندی انعطاف‌پذیر و کم مصرف در منابع روی تراشه را منجر شده است.

در روش‌های پیکربندی ایستا، دغدغه‌ی محدودیت مهلت زمانی[۱۳] وجود ندارد. این امر به کامپایلرها اجازه می‌دهد تا الگوریتم‌های پیچیده‌ای را جهت استفاده از موازی سازی محاسبه کنند[۱۹]. هر چند که کامپایلرها با توجه به فرصت زمانی که دارند می‌توانند الگوریتم بهینه‌ای را جهت نگاشت و پیکربندی اعمال کنند، اما تصمیمات در خارج از زمان اجرا جهت اعمال پیکربندی برای مخاطرات پیش‌بینی نشده برای معماری‌های دیجیتال در دنیای واقعی مطلوب نمی‌باشد.

امروزه با توجه به ضرورت انعطاف‌پذیری بیشتر برای سیستم‌های قابل بازپیکربندی و انواع کاربردهایی که معماری سیستم در حین اجرا ممکن است تغییر یابد، روش‌های ایستا جهت بازپیکربندی این دسته از سیستم‌ها از کارایی لازم برخوردار نمی‌باشند. بنابراین طراحان به این نتیجه رسیده‌اند که این سیستم‌ها باید قابلیت نگاشت و بازپیکربندی در حین اجرا را داشته باشند.

۲-۲- روش‌های بازپیکربندی پویا

به طور کلی می‌توان روش‌های بازپیکربندی پویا را به دو دسته جزئی و کامل تقسیم کرد. هر دو نوع بازپیکربندی پویای کامل و جزئی می‌توانند منجر به افزایش استفاده از منابع موجود برروی تراشه با استفاده از تسهیم زمانی[۱۴] گردند.

هر کدام از بازپیکربندی‌های کامل و جزئی نیز به دو نوع زمان محور[۱۵] و رویداد محور[۱۶] تقسیم می‌شوند. در حالت زمان محور، بازپیکربندی بر اساس زمانبندی خاص به صورت پویا انجام می-پذیرد در صورتی که در حالت رویداد محور، بازپیکربندی بر اساس ایجاد یک سری تغییرات خاص، مانند تغییرات دمای محیط ، تغییرات فرکانس پالس ساعت صورت می‌پذیرد [۲۰]. در مرجع [۲۱] یک بازپیکربندی پویای کامل رویداد محور برای پیاده‌سازی سیستم‌های کنترلی تحمل پذیر اشکال ارائه شده است. دلیل استفاده بازپیکربندی پویای کامل در این روش، هزینه‌ی سخت-افزاری و فناوری بیشتر در روش بازپیکربندی پویای جزئی است. مرجع [۲۲] نیز بازپیکربندی پویای جزئی را برای سیستم‌های چندهسته‌ای پیاده‌سازی کرده‌است که بخش پویای سیستم قابلیت پیاده‌سازی چندین مدل سخت‌افزاری را دارا می‌باشد.

در بازپیکربندی‌های پویای جزئی، سیستم به دو بخش پویا و ایستا تقسیم می‌شود. بازپیکربندی بخش ایستا در زمان اجرای برنامه تغییری نمی‌کند اما بازپیکربندی بخش پویای سیستم در زمان اجرا تغییر می‌کند.

مزیت بازپیکربندی پویای جزئی نسبت به بازپیکربندی پویای کامل در این است که هزینه‌ی زمانی آن کمتر می‌باشد و همچنین فایل حاوی رشته‌های بیتی[۱۷] آن نیز دارای حجم کمتری می‌باشد ولی انعطاف‌پذیری آن پایین‌تر از بازپیکربندی پویای کامل می-باشد[۲۳].

بیشترین کاربرد بازپیکربندی پویای جزئی در FPGA ها می‌باشد. FPGA های جدید، از قابلیت بازپیکربندی جزئی برای اعمال تغییرات پویا بر روی بخشی از سیستم، بدون تغییر بخش‌های دیگر، برخوردار هستند. این ویژگی بهره‌وری متناوب از منابع برنامه‌پذیر یک FPGA در حالت روشن را فراهم می‌آورد[۲۳]. بنابراین، این ویژگی منجر به سودمندی بسیاری برای سیستم از جمله، بهره‌وری مطلوب از منابع موجود و کاهش توان مصرفی می‌شود.

روش‌های بازپیکربندی پویای کامل یک معماری را در سطح کل تراشه جابجا ، موازی یا سری می‌کنند تا برخی از پارامترها بهبود یابند. ضمن اینکه این امر انعطاف‌پذیری سیستم را به صورت قابل توجهی افزایش می‌دهد.

مراجع [۲۴][۲۵] روش‌هایی را جهت بازپیکربندی پویای کامل برای معماری‌های قابل بازپیکربندی دانه‌درشت ارائه داده‌اند. هدف از این روش‌ها ضمن انعطاف پذیری بیشتر، کاهش مصرف انرژی در سطح تراشه است. اساس روش‌های فوق، بر بازپیکربندی پویای سلول‌های منزوی[۱۸]، الگوریتم‌های انتخاب ولتاژ و فرکانس و موازی‌سازی خودکار استوار است. بازپیکربندی پویای سلول‌های منزوی، سربار ناشی از انتخاب پویای فرکانس و ولتاژ را کاهش

می‌دهد. در واقع در اینجا با کمتر کردن فضای غیر قابل استفاده در سطح تراشه، کارایی سیستم افزایش یافته است. ایراد روش‌های ارائه شده‌ی [۲۴][۲۵] این است که بین المان‌ها به شدت وابستگی وجود دارد و این روش‌ها فقط معماری‌های دانه‌درشت را پوشش می‌دهند و جامعیت ندارند.

در مرجع [۲۶] روشی کارا تحت عنوان Transmap ارائه شده است که با ذخیره‌سازی فقط یک رشته‌ی بیتی جابجایی عملیات در سطح سیستم را انجام می‌دهد. جابجایی عملیات در این روش از نوع چرخشی است. این روش کاهش قابل توجهی را در نیاز به حافظه‌ی پیکربندی داشته است. این روش نیز مانند بسیاری از روش‌های ارائه‌شده‌ی پیشین، خاص‌منظوره بوده و فقط معماری-های دانه‌درشت را پوشش می‌دهد. همچنین این روش برای حالتی کارایی دارد که بخش کمی از منابع سیستم اشغال شده باشد. در مرجع [۲۷] روشی برای مقابه با پدیده سالخوردگی مدارات قابل بازپیکربندی ارائه شده که یا محتویات درون Look up Table (LUT) ها را برعکس می‌کنند یا اینکه سعی در تغییر مکان LUT ها دارند. در مرجع [۲۸] با به تعادل‌رساندن استفاده از مسیرهای اتصالی بین سلول‌ها میزان سالخوردگی سیستم را کاهش داده است. در واقع با این کار فشار و استرس را در ترانزیستورهای موجود در شبکه‌ی اتصالات کاهش می‌دهد که این امر موجب کاهش میزان سالخوردگی در مسیرهای ارتباطی به میزان ۱۹ درصد مطابق با سناریوهای به کارگرفته شده در آن روش شده‌است. در مرجع [۲۹] روشی ارائه شده‌است که به منظور استفاده‌ی حداکثر از قابلیت بازپیکربندی پویای جزئی جهت کارایی اجرا، منابع FPGA را به چندین بخش تقسیم می کنند و برنامه را بین این بخش‌ها تقسیم می‌کند که برخی از بخش‌ها دارای قابلیت بازپیکربندی پویا و برخی دیگر ایستا می‌باشند. به منظور بهبود عملکرد الگوریتم ، از تکنیک ادغام ماژول برای بهبود موازی کاری طراحی و اجرا بهره برده‌است[۳۰]. در این کار ، یک روش جدید برای اشکال زدایی در مدار پیاده‌سازی شده در FPGA معرفی شده است که امکان درج زیرساخت اشکال زدایی با سربار کم را با بهره‌گیری از تکنیک بازپیکربندی پویا فراهم می‌کند. هدف آن تسهیل اشکال زدایی ، افزایش مشاهده سیگنال داخلی و کاهش سربار خطایابی (منطقه و پیکربندی مجدد) است .

جدول ۱ مقایسه‌ای کیفی میان روش‌های مختلف بازپیکربندی را نشان می‌دهد.

روش‌هایی که تا به حال ارائه شده‌اند، هیچکدام بازپیکربندی را آگاه از وضعیت سالخوردگی بلوکهای منطقی انجام نمی‌دهند و وضعیت سالخوردگی آن‌ها را نادیده می‌گیرند.

جدول۱: مقایسه‌ی بین انواع مختلف بازپیکربندی

نوع بازپیکربندی	انعطاف-پذیری	مصرف توان	مصرف منابع
ایستا	کم	کم	کم
پویای جزئی	متوسط	متوسط	زیاد
پویای کامل	زیاد	زیاد	زیاد

در روش‌های پیشین، بازپیکربندی توسط ابزار سنتز انجام می‌شود و هیچکدام حتی کوچکترین درجه‌ی خودمختاری را در سطح تراشه ندارند. همچنین عموماً فاقد جامعیت بوده و فقط برای یک نوع از سیستم‌های قابل بازپیکربندی خاص طراحی شده‌اند. در ادامه به بیان روش پیشنهادی در این مقاله می‌پردازیم که با ارائه‌ی یک معماری هوشمند برای سیستم‌های قابل بازپیکربندی، قابلیت بازپیکربندی در حین اجرا را با انعطاف‌پذیری قابل توجه و با در نظر گرفتن میزان سالخوردگی سلول‌های سطح تراشه برای این دسته از سیستم‌ها فراهم می‌آورد تا از حداکثر ظرفیت سلول‌های تراشه استفاده شده و در نتیجه طول عمر سیستم افزایش بیابد.

۳- روش پیشنهادی

در روش‌های بازپیکربندی که تا به حال ارائه شده‌اند، طول عمر سیستم کمتر مورد توجه قرار گرفته است. زیرا در بخشی که بازپیکربندی به صورت ایستا صورت می‌گیرد، بخشی از سلول‌های منطقی سیستم تا زمانی که بازپیکربندی جدیدی صورت نگیرد در حالت فعال و بخشی دیگر از سلول‌ها در حالت غیرفعال به سر می‌برند. در واقع توزیع فعالیت بین سلول‌ها به صورت عادلانه نیست. با توجه به اینکه مدت زمان فعالیت (Usage) عناصر تراشه از عوامل تاثیرگذار در طول عمر تراشه می باشد و با توجه به رابطه‌ی معکوس میان میزان Usage و طول عمر [۴] [۵]، آن دسته از سلول‌هایی که به مدت زیادی در حالت فعال به سر بردند به مرور زمان یا دچار خرابی شده و یا با کاهش کارایی در فرم تاخیر مواجه می‌شوند. این در حالیست که هنوز تعداد قابل توجهی از سلول‌ها که در حین فعالیت سیستم زمان کمی فعال بودند و یا هرگز فعال نبودند، سالم باشند ولی به دلیل خرابی دیگر سلول‌ها بی استفاده می‌مانند. روش‌های بازپیکربندی پویا که در آن‌ها کل سیستم در زمان اجرا بازپیکربندی می‌شود، نیز چنین مشکلی را دارا هستند. زیرا در این نوع از بازپیکربندی تنها یک عامل و یا ابزار سنتز بدون آگاهی از مدت زمان فعالیت سلول‌های منطقی موجود در سیستم، برای نحوه‌ی نگاشت و جایابی عملیات تعیین شده در سلول‌ها تصمیم گیری می‌کنند.

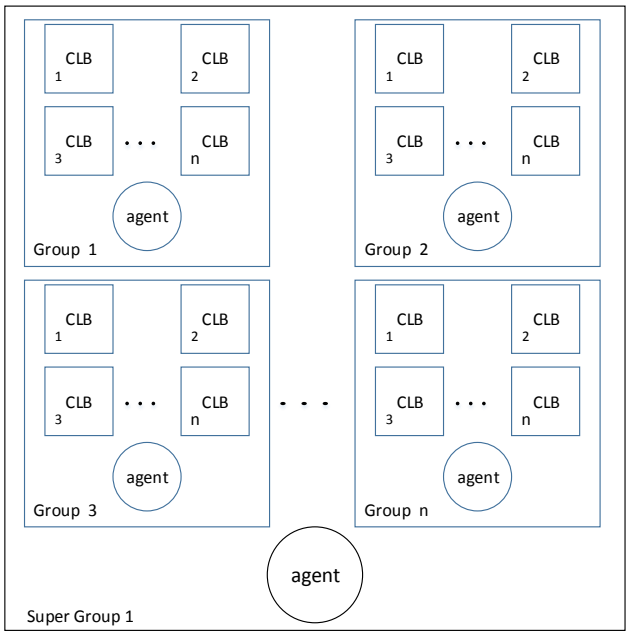
در روش پیشنهادی هدف ما این است که یک سیستم قابل

بازپیکربندی هوشمند با انعطاف‌پذیری بالا و طول عمر بهتر نسبت به روش‌های پیشین ارائه دهیم. منظور از طول عمر بیشتر این است که سیستم بتواند از حداکثر میزان فعالیت تمامی سلول‌های درونی خود بهره ببرد. برای رسیدن به این هدف از خاصیت توزیع-پذیر بودن سیستم‌های قابل بازپیکربندی استفاده کرده و ساختار آن را بر اساس سیستم‌های چند عامله طراحی کرده‌ایم. برای طراحی سیستم‌های قابل بازپیکربندی بر اساس سیستم‌های چندعامله، می‌توانیم هر کدام از سلول‌های منطقی موجود در سطح سیستم را یک عامل هوشمند در نظر بگیریم و این عامل‌ها بتوانند با یکدیگر مذاکره و تعامل کنند. اما این امر موجب می‌شود تا میزان زیاد مذاکرات و تعاملات بین عامل‌ها تاثیر منفی قابل توجهی بر پارامترهایی مانند مصرف توان، زمان اجرا و مصرف منابع بگذارد. به همین خاطر برای کاهش تعداد تعاملات بین عامل‌ها از ساختار سلسه مراتبی سیستم‌های چندعامله برای طراحی سیستم‌های قابل بازپیکربندی استفاده می‌نماییم. در این روش تعدادی سلول گروه را تشکیل خواهند داد و هر گروه دارای یک عامل هوشمند(Agent) به عنوان نماینده جهت تعامل می-باشد. تعدادی گروه نیز ابرگروه(Super Group) را تشکیل داده و هر کدام از این ابرگروه‌ها نیز یک عامل هوشمند را جهت نمایندگی برای مذاکره و تعامل با سایر ابرگروه‌ها را دارا می‌باشند. شکل ۱ نمای کلی از معماری روش پیشنهادی را برای یک ابرگروه نشان می‌دهد.

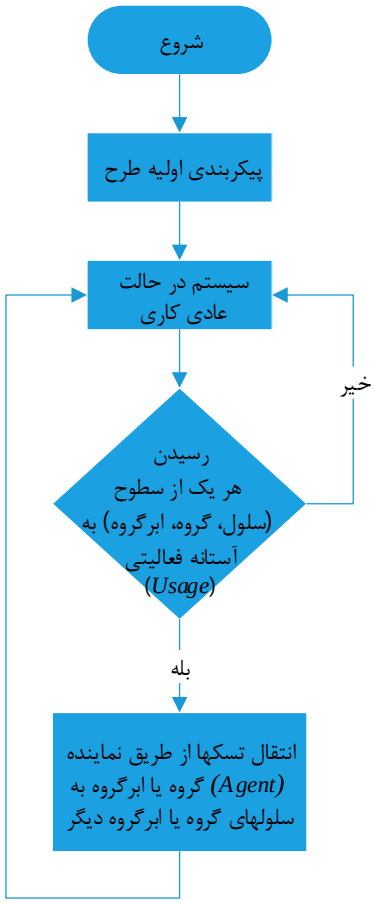
در شکل ۱ ، هر CLB یک سلول منطقی موجود در سطح تراشه می‌باشد. عامل هوشمند موجود در هر یک از ابرگروه‌ها می‌تواند با عامل‌های موجود در ابرگروه‌های دیگر تعامل کند تا درباره‌ی وضعیت گروه‌های موجود تحت نظر خود تصمیم‌گیری کند. ابرگروه‌ها نیز می‌توانند گروه‌های بزرگتری را تشکیل دهند که در این مقاله به دلیل افزایش سطح مذاکرات و زمان اجرا، از این کار صرف نظر کرده‌ایم. بدین ترتیب عامل‌های ابرگروه‌ها می‌توانند مستقیماً با یکدیگر مذاکره و تعامل نمایند.

هدف ما از طرح پیشنهادی علاوه بر افزودن به انعطاف‌پذیری سیستم قابل بازپیکربندی، به تعادل رساندن میزان Usage سلول-های سطح تراشه جهت استفاده‌ی یکنواخت از آن‌ها و در نتیجه افزایش طول عمر این دسته از سیستم‌ها می‌باشد. برای رسیدن به این مقصود، برای هر عامل تابع سودمندی را تعریف نموده‌ایم که به شرح زیر است:

۱. اگر میزان Usage همه‌ی زیرگروه‌ها از حد آستانه عبور کرد، آنگاه باید برای محول کردن وظیفه به گروه‌های دیگر، به عامل سطح بالاتر درخواست بفرستد.



شکل ۱: معماری یک ابرگروه در روش پیشنهادی



شکل ۲: فلوچارت چرخه‌ی کاری روش پیشنهادی

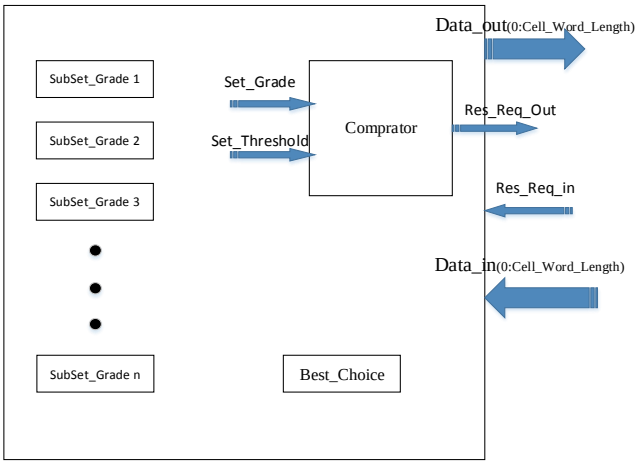
۲. اگر از سمت عامل لایه‌ی بالاتر درخواست منبع آمد، عامل سطح پایین‌تر باید به تشخیص عامل لایه‌ی بالاتر اعتماد کند و در جهت اجرای درخواست لایه‌ی بالایی

برآید.

اگر به تابع سودمندی تعریف شده در روش پیشنهادی برای عامل-های هوشمند دقت کنیم، متوجه این موضوع می‌شویم که تعاملات و تصمیم‌های عامل‌ها بر ا ساس هم کاری با ی کدیگر در را ستای افزایش کارایی سیستم می‌باشد و رقابتی بین آن‌ها مطرح نیست.

شکل ۲ فلوچارت کلی روش پیشنهادی را به تفکیک فازها با توجه به معماری ارائه شده در شکل ۱ نشان می‌دهد که ابتدا بازپیکربندی اولیه یک طرح صورت می‌گیرد و سپس سیستم در حالت عادی کاری قرار می‌گیرد. زمانی که هر کدام از عامل‌های گروه‌ها یا ابرگروه‌ها متوجه رسیدن زیرمجموعه‌های خود به آستانه‌ی سطح فعالیتی شوند، درخواست انتقال فعالیت را به عامل‌های سطح بالاتر ارسال می‌کنند. چگونگی انجام حالت‌های بیان شده در فلوچارت شکل ۲ ، در الگوریتم‌های ۱ و ۲ به تفصیل بیان شده است.

شکل ۳ یک ساختار کلی از عامل‌های سطوح مختلف را نشان می-دهد. این ساختار دارای قابلیت بازپیکربندی نمی‌باشد و به صورت ثابت می‌باشد.



شکل ۳: ساختار کلی عامل‌های هوشمند در روش پیشنهادی

با توجه به شکل ۳، عامل‌ها بافرهایی را تحت کنترل دارند که شامل میزان چرخه‌ی فعالیتی زیرگروه‌های مربوط به آن عامل می‌باشند. اندازه‌ی بافرها بسته به میزان آستانه‌ی فعالیتی آن گروه می‌تواند قابل تغییر باشد. به عنوان مثال اگر آستانه‌ی فعالیتی یک گروه یا سلول برابر ۱۰۰۰ سیکل ساعت باشد، اندازه‌ی بافر مربوط به آن در عامل برابر ۱۰ بیت می‌باشد. بافر *Best_Choice* در عامل هوشمند، شامل اندیس زیرمجموعه‌ای است که بهترین شرایط را برای برعهده گرفتن فعالیت دارد. این زیرمجموعه می‌تواند شامل گروه‌ها یا سلول‌های منطقی باشد. Data_out و Data_in به ترتیب شامل رشته‌های بیتی خروجی و ورودی مورد استفاده برای پیکربندی سلول‌های منطقی می‌باشند. زمانی که گروه‌های بیرونی درخواست منبع داشته باشند این درخواست از طریق پورت Res_Req_in و از طرف عامل بیرونی به عامل گروه مورد درخواست منتقل می‌شود و زمانی که گروه مربوط به یک عامل درخواست منبع جهت محول کردن فعالیت به گروه دیگر داشته باشد، این درخواست از طریق Res_Req_out به عامل لایه‌ی بالاتر ارسال می‌شود.

تعاملات و مذاکرات بین عامل‌ها جهت بازپیکربندی را به دو قسمت قبل از پیکربندی طرح جدید و در حین اجرای طرح فعلی تقسیم بندی می‌نماییم.

تعاملات بین عامل‌ها قبل از پیکربندی یک طرح

در این حالت فرض بر آن است که سیستم بدون آنکه قبلاً طرحی بر روی آن پیاده‌سازی شده باشد، می‌خواهد طرح یا معماری جدیدی را بر روی خود پیاده‌سازی کند یا سیستم در حال اجرای یک طرح است و قرار است طرح جدید بر روی آن پیاده شود. الگوریتم ۱ شبه کدی از تعامل عامل‌ها با یکدیگر برای رسیدن به این هدف را نشان می‌دهد. مطابق الگوریتم ۱، هر عامل از زیرگروهش که کمینه‌ی طول عمر را داراست، با خبر است. در واقع عامل‌های ابرگروه‌ها و گروه‌ها در حیطه‌ی عملیاتی خودشان به ترتیب از ابرگروه و گروه و سلول منطقی با کمترین میزان

فعالیت انجام داده خبر دارند که در الگوریتم ۱ به ترتیب با توابع *First_Super-Group_Candidate*، *First_Group_Candidate* و *First_Cell_Candidate* بیان شده‌اند. این اطلاعات در بافرهایی از نوع غیرفرار[۱۹] نگه‌داری می‌شوند و با خاموش شدن سیستم، این اطلاعات از بین نمی‌روند. در واقع تمامی عامل‌ها وضعیت چرخه‌ی فعالیتی خود را برای عامل سطح بالاترشان شرح می‌دهند. حال با داشتن این اطلاعات نوبت به اختصاص منابع می‌رسد. اولین تصمیم را عامل موجود در بالاترین سطح اخذ می‌کند و درخواست منابع را به ابرگروه با کمترین میزان Usage می‌فرستد. تا زمانی که ابرگروه مورد نظر به طور کامل اشغال نشد، انتقال عملیات به ابرگروه دیگر امکان‌پذیر نمی‌باشد. در ابرگروه انتخاب شده جهت تخصیص منابع، گروه با کمترین مقدار چرخه‌ی فعالیتی انتخاب شده و از آن گروه ابتدا سلولی جهت تخصیص انتخاب می‌شود که کمترین میزان فعالیت را نسبت به سایر سلول‌ها در آن گروه داشته است. در شبه کد موجود در الگوریتم ۱ منظور از *age_grade* همان میزان Usage می‌باشد که برای هر سه سطح ابرگروه، گروه و سلول‌ها دارای مقادیری متفاوت است. به طور

خلاصه، الگوریتم فوق به صورت مجتمع و فشرده عملیات را به جوان‌ترین بخش‌های سطح سیستم واگذار می‌کند.

الگوریتم ۱: تعاملات بین عامل‌ها قبل از پیکربندی اولیه‌ی یک طرح

```
Minimum0 : the best super-group to allocate
Minimum1 : the best group to allocate
Minimum2 : the best cell to allocate
First_Super-Group_Candidate( ){
      if (super-group(i).age_grade < Minimum0)
          Minimum0 = Super-group(i);
}

First_Group_Candidate(super-group){
        if (group(j).age_grade < Minimum1)
            Minimum1 = group(j);
}

First_Cell_Candidate(group){
          elseif (cell(k).age_grade < Minimum2)
               Minimum2 = cell(k);
}
1. Send_ResourceRequest _to_Super-Groups's-agents
2. while ( all of  the tasks assigned) {
3.      Minimum0 = a super-group that is not allocated;
4.       First_Super-Group_Candidate( );
5.       while (First_Super-Group_Candidate( )/=full){
6.           Minimum1 = a group that is not allocated;
7.           First_Group_Candidate(Minimum0)
8.           While(First_Group_Candidate(Minimum0)/=full){
9.                   First_Cell_Candidate(Minimum1) ;
10.                  Allocate( Minimum2);
11.                  if(all of  the tasks assigned){
12.                        RETURN;
13.                  }
14.              }
15.          }
16. }
```

تعاملات بین عامل‌ها در حین اجرای یک طرح

بازپیکربندی سیستم برای این حالت رویداد محور می‌باشد. بدین صورت که عامل‌ها بر اساس یک سری رویدادها تصمیم گیری کرده و واکنش نشان می‌دهند. رویدادها می‌تواند شامل تغییرات محیط اطراف و یا دریافت پیام از جانب عامل‌های دیگر باشد.

در این روش برای هر طبقه از سازمان یک مقدار آستانه‌ی[۲۰] فعالیتی در نظر گرفتیم. به عنوان مثال برای بلوک‌های عملیاتی در سطح سیستم یک مقدار آستانه و برای گروه‌ها نیز یک آستانه‌ی دیگر و برای ابر گروه‌ها نیز مقدار آستانه تعریف نمودیم. میزان Usage هر سلول توسط یک شمارنده‌ی سراسری[۲۱] محاسبه می‌شود.

در واقع زمانی که یک سلول در حالت فعال قرار دارد، میزان فعالیت آن سلول در هر پالس ساعت به اندازه‌ی یک واحد افزایش می‌یابد. الگوریتم ۲ شبه کدی است که ساز و کار عامل‌های هوشمند را جهت بازپیکربندی مجدد یک طرح در زمان اجرای یک سیستم نشان می‌دهد. عامل مربوط به هر گروه از میزان فعالیت هر کدام از سلول‌های داخل گروه که در یک بافر ذخیره شده‌اند، باخبر است. از زمانی که مقدار بافر به اندازه‌ی آستانه‌ی فعالیتی یک بلوک عملیاتی برسد، عامل مربوط به گروه ابتدا سعی دارد تا یکی از سلول‌های منطقی که در همان گروه است و فعالیت نمی‌کند و میزان Usage آن نیز از سلول منطقی فعال فعلی کمتر است، را جایگزین کرده و بار فعالیتی را از سلول فعال که مدت زمان فعالیتش از آستانه گذر کرده است، بردارد. در غیر این صورت اجازه می‌دهد تا سلول به فعالیت خود ادامه بدهد. این امر موجب می‌شود تا میزان Usage همه‌ی سلول‌ها به یک اندازه پیش برود.

مطابق الگوریتم ۲، اگر میزان فعالیت همه‌ی سلول‌های یک گروه از حد آستانه‌ی فعالیتی تعیین شده برای یک گروه عبور کند، درجه-ی فعالیتی(grade) گروه یک واحد افزایش یافته و عامل آن گروه درخواست منبع را به عامل سطح بالا ارسال می‌کند. عامل سطح بالاتر یا همان عامل ابرگروه در ابتدا سعی دارد تا یک گروه موجود در همان ابرگروه را یافته که درجه‌ی فعالیتی آن کمتر از گروهی باشد که درخواست منبع داده است. اگر گروه مد نظر را پیدا کرده است، برایش درخواست منبع می‌فرستد. حال در صورت موجود بودن سلول‌های غیرفعال در گروه درخواست شده، آن سلول‌ها وظایف و عملیات سلول‌های درخواست دهنده را برعهده می‌گیرند.

الگوریتم ۲: تعاملات بین عامل‌ها در حین اجرای یک طرح

```
T0: Threshold for cycles of the super-group lifetime
T1: Threshold for cycles of the group lifetime
T2: Threshold for cycles of the Cell lifetime
Best0:Appropriate Super-group to be requested on chip
Best1: Appropriate Group to be requested in same Super-group
Best2: Appropriate Cell to be requested in same group

1.While(true){
2.       // for each groups:
3.            if  (cell(i).cycle_age >= T2 ){
4.                 if  ( cell(i).age_grade<Best2.age_grade)
5.                      replace_activity( cell(i), Best2);
6.                       cell(i).cycle_age = 0;
7.                 else
8.                       continue;
9.       }
10. // for each super-groups :
11.           if (super- group(j).cycle_age>=T0){
12.            send_request_to other_super-group's_agents;
13.             while(super-group's_agent.find(Best0)){
14.                    super-groups_agent_send_request(Best0);
15.                    super-group-agent-send_ack(group(j));
16.                    replace_activity(group(j) , Best1);
17.                    if( group(j).activity = 0){
18.                         group(j).cycle_age = 0;
19.                          break;
20.                    }
21.           }
```

```
22.
23.       }

24.        // for general system
25.             if ( super-group(h).cycle_age>=T1){
26.               send_request_to_super-group's_agent;
27.                while(super-group's_agent.find(Best0)){
28.                        super-groups_agent_send_request(Best0);
29.                        replace_activity(super-group(h) , Best0);
30.                       if( super-group(h).activity = 0){
31.                            super-group(h).cycle_age = 0;
32.                             break;
33.                       }
34.                }
35.             }
36.   }
```

آگاهی و اطمینان یافتن از این امر که وظایف به درستی منتقل شده‌اند یا خیر توسط ارسال سیگنال تایید از دو طرف مذاکره کننده صورت می‌گیرد. عامل ابرگروه این روند را تا زمانی برای گروه درخواست دهنده ادامه می‌دهد تا تمامی سلول‌های آن گروه آزاد گردند. اطلاعاتی که بین عامل‌ها در حین این ارتباطات منتقل می‌شوند عبارت‌اند از:

۱. سیگنال درخواست منبع که شامل یک بیت می‌باشد.
۲. سیگنال تایید اختصاص یافتن منبع که شامل یک بیت می‌باشد.
۳. اطلاعات درگاه‌های ورودی /خروجی و همچنین بیت‌هایی که عملیات در آن توصیف شده‌اند که با توجه به دانه‌ریز یا دانه‌درشت بودن سیستم قابل بازپیکربندی، می‌تواند متغیر باشد.

در صورتی که عامل موجود در ابرگروه، گروهی برای تخصیص پیدا نکرد، اجازه می‌دهد تا گروه‌ها به همان صورت قبلی به فعالیت خود ادامه بدهند تا اینکه Usage سلول‌های موجود در ابرگروه به حد آستانه برسد. در این صورت عامل ابرگروه درخواست منبع را برای عامل لایه‌ی بالاتر می‌فرستد. در صورتی که عامل لایه‌ی بالاتر یک ابرگروهی را یافت که میزان Usage سلول‌های آن کمتر از درجه‌ی فعالیتی ابرگروه درخواست دهنده باشد، درخواست منبع را برای آن ابرگروه ارسال کرده و مجددا همان ساز و کاری که برای انتقال منابع برای گروه‌ها در نظر گرفته شده بود، حال برای یک سطح بالاتر و ابرگروه‌ها انجام می‌گیرد.

نکات قابل درک از الگوریتم های ۱ و ۲ این است که در روش پیشنهادی بازپیکربندی به صورت فشرده انجام می پذیرد. بدین صورت که منابع به صورت پراکنده اختصاص نمی‌یابند. فارغ از زمانی که می خواهیم طرح جدید را پیکربندی کنیم، مقداری تاخیر به سیستم تحمیل می‌شود، در هنگام اجرای طرح پیاده‌سازی شده برروی سیستم میزان تاخیر کم است. زیرا همانطور که می‌دانیم، یکی از عوامل موثر در امر تاخیر سیستم های قابل بازپیکربندی طول سیم‌های ارتباطی می‌باشد و با مجتمع و فشرده بودن نگاشت، طول سیم‌های ارتباطی کوتاه بوده و میزان تاخیر کاهش می‌یابد. همچنین پراکنده نبودن نگاشت امکان استفاده‌ی حداکثر منابع سیستم را به ما می‌دهد. بدین صورت که این امکان برای ما به وجود می‌آید تا طرح‌های بزرگ‌تری را بتوانیم بر روی سیستم‌های قابل بازپیکربندی پیاده‌سازی نماییم.

۴- شبیه‌سازی و ارزیابی

در این مقاله، روش پیشنهادی با استفاده از زبان توصیف سخت افزار VHDL توصیف و با استفاده از نرم‌افزار Modelsim شبیه‌سازی و به وسیله‌ی نرم‌افزار Design Compiler سنتز شده‌است. زبان VHDL هیچ چهارچوبی[۲۲] جهت توصیف و پیاده سازی سیستم‌های چندعامله ندارد. به همین خاطر، از پایه به تعریف عامل هوشمند و ساختار سیستم های چندعامله در این زبان پرداخته‌ایم.

جهت ارزیابی روش پیشنهادی، از سه الگوریتم محک ALU ۳۲ بیتی ، رمزکننده‌ی (Encoder) ۶۴ به ۶ و Barrel Shifter ۶۴ بیتی استفاده کرده‌ایم.

برای ارزیابی الگوریتم‌های محک مطرح شده، این الگوریتم ها را در حالت‌های مختلفی که درصدی از حجم منابع سطح سیستم را اشغال می‌کنند، ارزیابی می‌کنیم.

جدول ۲ الگوهای مختلف جهت شبیه‌سازی الگوریتم‌های محک را نشان می‌دهد. با توجه به جدول ۲، تعداد کل سلول‌های منطقی سطح تراشه در الگوهای ۱، ۲ و ۳ به ترتیب ۱۲۸، ۲۵۶ و ۵۱۲ می‌باشد. در جدول ۲ منظور از هر سلول یک LUT[۲۳] با چهار ورودی می‌باشد.

جدول ۳ نیز میزان آستانه‌های فعالیتی تعیین شده برای سلول‌ها، گروه‌ها، ابرگروه‌ها را نشان می‌دهد. میزان چرخه‌ی فعالیتی هر سطح وابسته به میزان چرخه‌ی فعالیتی زیر مجموعه‌ی آن می‌باشد. با توجه به جدول ۳ میزان آستانه‌ی فعالیتی گروه‌ها و ابرگروه‌ها بدین صورت است که اگر میزان فعالیت سلول‌های منطقی موجود در آن‌ها به حد تنظیم شده در جدول برسد، عامل مربوط به آن دسته باید برای انتقال فعالیت به گروه یا ابرگروه دیگر اقدام کند.

هدف ما از شبیه‌سازی روش پیشنهادی این است که نشان دهیم ضمن برخورداری از انعطاف‌پذیری و هوشمندی ، میزان Usage سلول‌ها در مدت زمان معین از حیات سیستم نسبت به روش‌های پیشین کاهش یافته که در نتیجه می‌توان استنباط کرد که طول

عمر سیستم افزایش می‌یابد. مدت زمان شبیه‌سازی را برابر $۱۰^{۱۲}$ (۱۰۰۰ گیگا) سیکل ساعت در نظر گرفته‌ایم.

جدول ۲: الگوهای تعیین شده جهت ارزیابی روش پیشنهادی به وسیله‌ی الگوریتم‌های محک

شماره‌ی الگو	الگوریتم محک	درصد اشغال منابع	تعداد سلول در گروه	تعداد گروه در ابرگروه	تعداد ابرگروه
۱	۳۲ ALU بیتی	۹۵	۴	۸	۴
	رمزکننده ۶۴ به ۶	۸۰			
	۶۴ Barrel Shifter بیتی	۵۰			
۲	۳۲ ALU بیتی	۵۱	۸	۸	۴
	رمزکننده ۶۴ به ۶	۴۰			
	۶۴ Barrel Shifter بیتی	۲۶			
۳	۳۲ ALU بیتی	۲۸	۸	۸	۸
	رمزکننده ۶۴ به ۶	۲۲			
	۶۴ Barrel Shifter بیتی	۱۴			

جدول ۳: تنظیمات میزان آستانه‌ی فعالیتی اجزای سیستم

سطح سیستم	آستانه‌ی چرخه‌ی فعالیت(سیکل ساعت)
سلول	$۱۰^{۶}$
گروه	$۱۰^{۹}$
ابرگروه	$۱۰^{۱۰}$

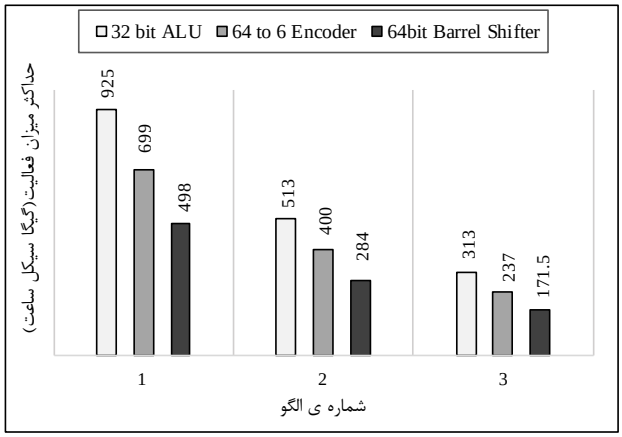
شکل ۴ میزان حداکثر Usage سلول‌های سطح تراشه‌ی شبیه-سازی شده تحت روش پیشنهادی را با توجه به الگوهای جدول ۲ نمایش می‌دهد. با توجه به شکل۴ مشاهده می‌کنیم که روش پیشنهادی تاثیر قابل توجهی درکاهش میزان فعالیت(Usage) سلول‌ها در یک زمان مشخص با توجه به میزان منابع اشغال شده، داشته است. با توجه به الگوهای جدول ۲ و شکل۴ در می‌یابیم که هرچه میزان منابع مصرفی توسط الگوریتم‌های قابل پیاده‌سازی کمتر باشد، میزان Usage کمتر خواهد بود. به عنوان مثال در جدول ۲ الگوریتم محک Barrel Shifter مطابق با الگوی اول ، ۵۰ درصد و مطابق با الگوی دوم، ۲۶ درصد از فضای تراشه را اشغال کرده است و در شکل ۴ می‌بینیم که حداکثر Usage در میان سلول‌های موجود در سطح سیستم تحت الگوریتم محک Barrel Shifter طبق الگوی دوم حدود ۱.۵ برابر کمتر از حالتی است که این الگوریتم محک طبق الگوی ۱ باشد.

در این مقاله روش Transmap را نیز مطابق الگوهای جدول ۲ و تنظیمات میزان حد آستانه‌های جدول ۳ شبیه سازی و ارزیابی نموده‌ایم. هدف اصلی این روش افزایش بهره‌وری سطح تراشه بود که این کار را با استفاده تغییر پیکربندی سیستم در زمان اجرا و به تعادل رساندن میزان فعالیت سلول‌ها انجام داده‌است که منجر به کاهش Usage آن‌ها در مدت زمان مشخص می‌گردد.

شکل ۵ نسبت Usage روش Transmap به روش پیشنهادی در این مقاله را نشان می‌دهد. در واقع این نسبت بیانگر میزان بهبود میزان Usage روش پیشنهادی نسبت به روش Transmap در مدت زمان $۱۰^{۱۲}$ سیکل ساعت می‌باشد.

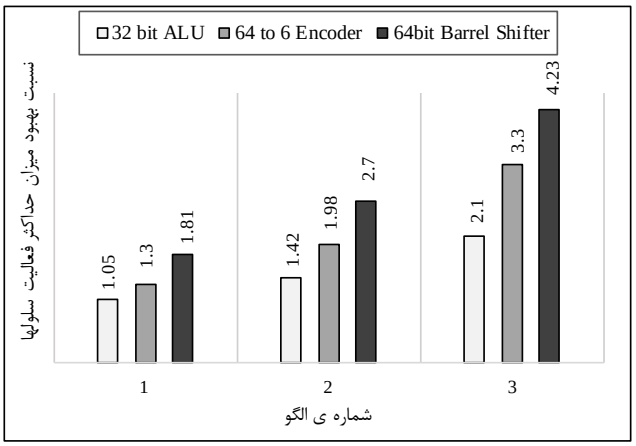
با توجه به شکل ۵ و جدول ۲ به این نتیجه می‌رسیم که هرچه معماری پیاده‌سازی شده بر روی سیستم فضای کمتری از سیستم قابل بازپیکربندی را اشغال کند، در آن صورت کارایی روش پیشنهادی در میزان بهره‌وری و Usage سلول‌ها نسبت به یکی از بهترین روش‌های پیشین (Transmap) افزایش می‌یابد.

در روش پیشنهادی جهت انجام بازپیکربندی در زمان اجرای سیستم، عامل‌های هوشمند با یکدیگر مذاکره و تعامل می‌کنند. این امر موجب ایجاد سربار توان مصرفی و مساحت و همچنین موجب سربار تبادل پیغام می‌گردد. جدول ۴ الگوهایی را برای ارزیابی میزان سربار مساحتی و توان مصرفی را بیان می‌دارد. همانطوری که در جدول۴ مشاهده می‌کنیم، در تمامی الگوها تعداد کل سلول‌های سطح سیستم برابر می‌باشند و تنها در نحوه‌ی گروه‌بندی سلول‌ها و تعداد عامل‌های هوشمند تمایز قائل شدیم.



شکل ۴: حداکثر میزان فعالیت (Usage) سلول‌ها مطابق الگوهای جدول ۲ در سیستم به مدت زمان $۱۰^{۱۲}$ سیکل ساعت کارکرد

لازم به ذکر است که با توجه به اینکه عامل‌ها به صورت رفتاری پیاده‌سازی شده‌اند و سخت‌افزار مساحت محاسبه شده بر اساس بینش ابزار Design Compiler بدست آمده است.



شکل۵: نسبت حداکثر Usage سلول‌ها در روش پیشنهادی به روش Transmap [۲۶]

جدول ۵ میزان مصرف توان و مصرف منابع را روش پینهادی و حالت عادی را برای برنامه محک ALU ۳۲ بیتی نشان می‌دهد. شکل۶ میزان سربار مساحتی سیستم را هنگامی که سیستم مطابق با روش پیشنهادی طراحی شده باشد را نسبت به یک سیستم بدون اعمال روش پیشنهادی نشان می‌دهد. در واقع این مقادیر با تقسیم میزان مصرف منابع روش پیشنهادی و بدون روش پیشنهادی به دست آمده‌است. با توجه به شکل۵ می‌بینیم که هرچه تعداد سلول‌ها در یک گروه بیشتر باشد، سربار مساحتی سیستم کمتر خواهد بود. زیرا که تعداد عامل‌های کمتری را به ازای گروه‌های بزرگتر در سیستم خواهیم داشت. علت این سربارها ساختار عامل‌ها، بافرهای ذخیره‌کننده‌ی درجه‌ی فعالیت اجزای مختلف سازمان و داده‌های مبادله شده در حین مذاکره و تعامل بین عامل‌ها می‌باشند.

شکل ۷ میزان سربار توان مصرفی سیستم را مطابق با الگوهای جدول ۴ و برای الگوریتم محک ALU ۳۲ بیتی نشان می‌دهد. با مقایسه‌ی شکل‌های ۶ و ۷ می‌بینیم که سربار توان مصرفی در

جدول ۴: الگوهایی جهت ارزیابی سربار مساحتی و توان مصرفی روش پیشنهادی

شماره‌ی الگو	تعداد سلول در گروه	تعداد گروه در ابرگروه	تعداد ابرگروه
۱	۲	۳۲	۲
۲	۴	۱۶	۲
۳	۸	۸	۲
۴	۱۶	۴	۲
۵	۳۲	۲	۲

شکل۶: درصد سربار مساحتی روش پیشنهادی مطابق با الگوهای جدول ۴

روش پیشنهادی کمتر از سربار مساحتی می‌باشد. این امر بدین دلیل است که عکس‌العمل عامل‌های هوشمند در روش پیشنهادی رویداد محور است و تا زمانی که اتفاقی در محیط اطراف آن‌ها را تحریک نکند، فعالیت نمی‌کنند.

جدول ۶ میزان میانگین سربار تبادل پیام بین عامل‌ها را با تو جه به الگوریتم‌های محک والگوهای جدول ۲ برحسب بیت نشان می‌دهد. مقادیر این جدول وابسته به میزان پورت داده‌ی ورودی و خروجی (رشته‌های بیتی) و همچنین سیگنال های درخواست یا تایید منابع ، مطابق شکل ۳ می‌باشد. که در واقع اندازه‌ی پورت ورودی و خروجی به تعداد ورودی LUT های مورد استفاده در تراشه‌ی قابل بازپیکربندی دارد. بدین صورت که اگر تعداد ورودی LUTهای مورد استفاده در تراشه قابل بازپیکربندی برابر n باشد، آنگاه عرض پورت ورودی و خروجی داده در عامل هوشمند برابر 2^n می‌باشد.

جدول ۵: میزان مصرف توان و منابع با استفاده از روش پیشنهادی و بدون استفاده از روش یشنهادی در حالت اجرای الگوریتم ALU ۳۲ بیتی

شماره الگو	میزان مصرف توان (mw)		میزان مصرف منابع (logic cells)	
	بدون استفاده از روش پیشنهادی	با استفاده از روش پیشنهادی	بدون استفاده از روش پیشنهادی	با اسفتاده از روش پیشنهادی
۱	۰.۳۰۲	۰.۵۰۴	۳۷۷	۹۲۳
۲	۰.۳۰۴	۰.۴۵۹	۳۷۴	۶۸۴
۳	۰.۳۰۴	۰.۴۰۴	۳۷۳	۵۸۵
۴	۰.۳۱	۰.۳۷۵	۳۷۱	۴۹۳
۵	۰.۳۱۲	۰.۳۴۹	۳۶۸	۴۳۴

شکل۷: درصد سربار توان مصرفی روش پیشنهادی مطابق با الگوهای جدول ۴

جدول۶: میانگین میزان سربار تبادل پیام برای الگوریتم‌های محک و الگوهای جدول ۲

میزان سربار تبادل پیام بر حسب bit	الگوریتم محک
۱۴۵۸	ALU ۳۲ بیتی
۱۲۱۸	رمزکننده ۶۴ به ۶
۷۷۰	Barrel Shifter ۶۴ بیتی

جدول۷: مقایسه‌ی کمی و کیفی روش پیشنهادی با روش‌های Resource Partitioning and Application Scheduling ،Transmap, Chortle ،Partial Reconfiguration

نام روش	میانگین بیشترین Usage سلول‌ها مطابق الگوی ۱	دسته‌های قابل پوشش	انعطاف‌پذیری	مصرف توان و منابع
روش پیشنهادی	۷۰۷	ریزدانه-درشت دانه	بالا+	بالا
Resource Partitioning and Application Scheduling	۸۸۴	ریزدانه-درشت دانه	بالا	بالا
Transmap	۹۶۱	درشت دانه	بالا	بالا
Partial Reconfiguration	۱۰۰۰	ریز دانه	متوسط	متوسط
Chortle	۱۰۰۰	ریز دانه	پایین	پایین

برای نشان‌دادن بهتر مزایا و معایب روش پیشنهادی، مقایسه‌ی کمی و کیفی آن را با ۴ تا از روش‌های پیشین به نام‌های Chortle [15]، Partial Reconfiguration [23] ، Transmap[26] , Resource Partitioning and Application Scheduling [29] در جدول ۷ نشان داده‌ایم که به ترتیب در دسته‌های سیستم‌های قابل بازپیکربندی ایستا، پویای جزئی و پویای کامل قرار می‌گیرند.

با توجه به جدول ۷ درمی‌یابیم که میزان Usage سلول‌های تراشه در مدت زمان $۱۰^{۱۲}$ سیکل ساعت اجرای سیستم، کمترین مقدار به روش پیشنهادی اختصاص یافته است و روش Chortle که یک روش ایستا می‌باشد به دلیل عدم تغییر حالت فعالیت (فعال بودن یا نبودن) سلول‌ها در حین اجرا، بیشترین میزان Usage سلول‌ها در آن برابر با کل زمان اجرای سیستم ($۱۰^{۱۲}$ سیکل ساعت) می‌باشد. این امر برای روش Partial Reconfiguration نیز صدق می‌کند زیرا بخشی از سیستم به صورت ایستا می‌باشد. در جدول ۷ برای مقایسه‌ی کمی حداکثر میزان Usage سلول‌ها به الگوی ۱ جدول ۲ بسنده کرده‌ایم زیرا مقایسه با روش Transmap بر اساس سایر الگوهای جدول ۲ به تفصیل در شکل ۴ به نوعی دیگر بیان شده است.روش پیشنهادی برای هر دو نوع سیستم‌های قابل بازپیکربندی درشت‌دانه و ریزدانه قابل پیاده‌سازی می‌باشد در صورتی که سایر روش‌ها تنها یک نوع را پوشش می‌دهند. از نظر میزان انعطاف‌پذیری نیز به دلیل اینکه روش پیشنهادی در زمان اجرا ، به صورت کامل و توزیع شده قابلیت بازپیکربندی را دارا می‌باشد، بیشترین مقدار(بالا+) را به خود اختصاص می‌دهد و میزان انعطاف‌پذیری روش Transmap به این دلیل پایین‌تر از روش پیشنهادی قرار می‌گیرد که سیستم دارای هوش توزیع شده نمی‌باشد و بازپیکربندی توسط تنها یک عامل انجام می‌پذیرد.

با وجود آنکه روش پیشنهادی دارای سربارهایی از لحاظ مصرف توان و مساحت می‌باشد، ولی با توجه به انعطاف پذیری و افزایش طول عمری که برای سیستم به ارمغان می‌آورد، به صرفه و کارا می‌باشد.

به طور کلی در این مقاله روشی ارائه داده‌ایم که تا سیستم های قبل بازپیکربندی دارای هوش توزیع شده در پیکره و معماری خود باشند و به صورت خودمختار و با انعطاف بالا و با هدف به تعادل رساندن فعالیت سلول‌های مجود در سطح سیستم(افزایش طول عمر سیستم)، در زمان اجرا خود را بازپیکربندی کنند.

۵- نتیجه‌گیری و کار آتی

در این مقاله یک معماری برای سیستم‌های قابل بازپیکربندی بر اساس سیستم‌های چندعامله ارائه داده‌ایم که در آن تراشه فقط بار اول توسط ابزار سنتز پیکربندی می‌شود و در دفعات بعد با استفاده از مذاکره و تعامل بین عامل‌ها و با توجه به میزان سالخوردگی هر بخش خودش را مجدداً بازپیکربندی می‌کند که بیانگر انعطاف‌پذیری بالای سیستم می‌باشد. روش پیشنهادی را با استفاده از زبان VHDL شبیه‌سازی نموده و آن را با استفاده از سه الگوریتم محک مورد ارزیابی قرار داده‌ایم که نتایج حاکی از آن است که علاوه بر صحت انجام عمل بازپیکربندی در زمان اجرا، حداکثر میزان فعالیت(Usage) سلول‌ها در مدت زمان مشخص کاهش قابل توجهی یافته است که این امر بیانگر افزایش طول عمر سیستم می‌باشد. روش پیشنهادی برای هر دو نوع از سیستم های قابل بازپیکربندی ریزدانه و درشت‌دانه مناسب می‌باشد. وجود عامل‌های هوشمند و تعامل بین آنها، سربارهایی را در توان مصرفی و منابع ایجاد کرده است که به دلیل افزایش قابل توجه میزان انعطاف پذیری و طول عمر سیستم، قابل توجیه می‌باشد. در این مقاله عامل ها برای قسمت جایابی (routing) سیستم های قابل بازپیکربندی تصمیمی را اتخاذ نمی‌کنند و ما این کار را در این روش به ابزار سنتز واگذار کرده‌ایم که در آینده باید راه کاری هوشمند برای این چالش برگزینیم.

مراجع

[1] M. J.Flynn, "*Area – Time – Power and Design Effort; The Basic Trade-offs in Application Specific Systems*", in Proceeding of IEEE 16th International Conference on Application-specific Systems, Architectures and Processors, Samos, Greece, pp. 3-6, July 23-25 2005.

[2] R. Hartenstein, "*A Decade of Reconfigurable Computing: A Visionary Retrospective*" in Proceedings of IEEE Conference on Design, Automation and Test in Europe, Munich, Germany, pp. 642-649, Mar. 10-14, 2001.

[3] Oquzhan Atak, Abdullah Atalar "*An efficient computing model for coarse graned reconfigurable architectures and its application to a reconfigurable compuer*" 21st IEEE international Conference on Application-specific Systems, Architectures and Processors, Rennes, France, july 7-9, 2010.

[4] Abdulazim Amouri, Florent Bruguier, Saman Kiamehr, Pascal Benoit, Lionel Torres and Mehdi Tahoori, "*Aging effects in FPGAs: An experimental analysis*". 24th international Conference on Field programmable logic and applications, Munich, Germany, 2-4 Sept, 2014.

[5] Zana Ghaderi, "*Aging-induced Performance Degradation: Monitoring and Mitigation*". submitted in partial satisfaction of the requirements for the degree of doctor of philosophy in Computer Science, university of california, Irvine, 2017.

[6] Mohammed Ahmed Jubair ; Salama A. Mostafa ; Aida Mustapha ; Hanayanti Hafit ., "*A Survey of Multi-agent Systems and Case-Based Reasoning Integration*", International Symposium on Agent, Multi-Agent Systems and Robotics (ISAMSR) , Putrajaya, Malaysia, 27-28 Aug. 2018.

[7] A. Halinka, P.Rzepka, M.Szablichi, "*Agent model of multi-agent system for area power system protection* " Modern Electric Power Systems (MEPS), Wroclaw, Poland, pp. 1-4, July 6-9, 2015.

[8] Ababii Victor, Sudacevschi Viorica, Munteanu Silvia, Bordian Dimitrie, Calugari Dmitri, Nistiriuc Ana,"*Multi-agent cognitive system for optimal solution search*", International Conference on Development and Application Systems(DAS), Suceava, Romania, pp.53-56, May 24-26, 2018.

[9] Munan Li, Fen Huang, "*Formal describing the organization in the pervasive healthcare information system: Multi-agent system perspective*" International Conference on Advanced Robotics and Mechatronics (ICARM), Macau, China, pp. 524-529 , Aug 18-20, 2016.

[10] Wooldridge M., "*An Introduction to MultiAgent Systems*", Second edition, WILEY Publication 2009.

[11] Prabhat Ranjan Singh, Bishwajeet Pandey, Tanesh Kumar, Teerath Das " *LVCMOS I/O standard based million MHz high performance efficient design on FPGA* ", International Conference on Communication and Computer Vision (ICCCV), Coimbatore, India, pp. 1-4, Dec 20-21 , 2013.

[12] Bo Wang, Leibo Liu "*A flexible and energy-efficient reconfigurable architecture for symmetric cipher processing*" IEEE international Symposium on Circuits and Systems, Lisbon, Portugal, pp. 1182-1185, May 24-27, 2015.

[13] Igor A. Kalyaev, Ilya Levin, Alexey Dordopulo, Liubov M.Slasten "*FPGA-based reconfigurable computer systems*" , Science and Information Conference, London, Uk, pp. 148-155, Oct 7-9, 2013.

[14] Tomáš Drahoňovský ,Martin Rozkovec , Ondřej Novák "*A highly flexible reconfigurable system on a Xilinx FPGA*" International Conference on Reconfigurable Computing and FPGAs, Cancun, Mexico, pp. 1-6, Dec 8-10, 2014.

[15] R. J. Francis, J. Rose, and K. Chung, "Chortle: A technology mapping program for lookup table-based field programmable gate arrays," in *ACM/IEEE Conference on Design Automation*, pp. 613-619, 1991.

[16] R. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast technology mapping for lookup table-based FPGAs," in *ACM/IEEE 28th Conference on Design Automation*, pp. 227-233, 1991.

[17] J. Cong, Y. Ding, and Systems, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*,vol. 13, no. 1, pp. 1-12, 1994.

[18] Y.-Y. Liang, T.-Y. Kuo, S.-H. Wang, W.-K. Mak, and Systems, "ALMmap: Technology mapping for FPGAs with adaptive logic modules," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 31, no. 7, pp. 1134-1139, 2012.

[19] M. Kubica and D. Kania, "Area–Oriented Technology Mapping for LUT–Based Logic Blocks," *International Journal of Applied Mathematics Computer Science*, vol. 27, no. 1, pp207-222, 2017.

[20] C. V. Paiz Gatica, "Dynamically reconfigurable hardware for embedded control systems," 2012.

[21] G. Venkataramani, M. Budiu, T. Chelcea, and S. C. Goldstein, "C to asynchronous dataflow circuits: An end-to-end toolflow," *International Workshop on Logic Synthesis*, 2004.

[22] K. Danne, C. Bobda, and H. Kalte, "Run-time exchange of mechatronic controllers using partial hardware reconfiguration," in *International Conference on Field Programmable Logic and Applications* , pp. 272-281, 2003.

[23] M. Rabozzi, G. C. Durelli, A. Miele, J. Lillis, and M. D. Santambrogio, "Floorplanning automation for partial-reconfigurable fpgas via feasible placements generation," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 1, pp. 151-164 , 2017.

[24] S. M. Jafri *et al.*, "Energy-aware coarse-grained reconfigurable architectures using dynamically reconfigurable isolation cells," in *IEEE 14th International Symposium on Quality Electronic Design (ISQED)*, pp. 104-111, 2013.

[25] S. M. Jafri, M. A. Tajammul, A. Hemani, K. Paul, J. Plosila, and H. Tenhunen, "Energy-aware-task-parallelism for efficient dynamic voltage, and frequency scaling, in cgras," in *IEEE International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII)*, pp. 104-112, , 2013.

[26] S. M. Jafri, M. Daneshtalab, N. Abbas, G. S. Leon, and A. Hemani, "Transmap: Transformation based remapping and parallelism for high utilization and energy efficiency in cgras," *IEEE Transactions on Computers*, vol. 65, no. 11, pp. 3456-3469, 2016.

[27] Z. Ghaderi, N. Bagherzadeh, A. Albaqsami, "STABLE: Stress-aware Boolean Matching to Mitigate BTI-induced SNM Reduction in SRAM-based FPGAs", IEEE Transaction on Computers, Vol. 67, Issue. 1, PP. 102-114, 2017.

[28] B. Khalegi, B. Omidi, H. Amrouch, J. Henkel and H. Asadi, "Estimating and Mitigating Aging Effects in Routing Network of FPGAs", IEEE Transaction on Very Large Integrated Systems (VLSI), Vol. 27, Issue. 3, PP. 651-664, 2019.

[29] Zhe Wang, Qi Tang , Biao Guo, Ji-Bo Wei and Ling Wang"*Resource Partitioning and Application Scheduling with Module Merging on Dynamically and Partially Reconfigurable FPGAs"* ACM Transactions on Reconfigurable Technology and Systems, *9*(9), 1461; September 2020.

[30] Alexandra Kourfali, Dirk Stroobandt " In-Circuit Debugging with Dynamic Reconfiguration of FPGA Interconnects" ACM Transactions on Reconfigurable Technology and Systems ,January 2020.

پاورقی‌ها:

[1] Clock Pulse
[2] Fine Grained
[3] Coarse Grained
[4] Logic Cells
[5] Usage
[6] Intelligent Agent
[7] Autonomy
[8] Interaction
[9] Area overhead
[10] Mapping
[11] Binary decision diagram
[12] Deadline
[13] Time Sharing
[14] Time Driven
[15] Event Driven
[16] Bit Streams
[17] Isolated Cells
[18] Non-volatile
[19] Threshold
[20] Global
[21] Framework
[22] Look Up Table