# بهینه‌سازی سنتز مدارهای حسابی بر روی معماری‌های قابل بازپیکربندی درشت‌دانه

سمانه امامی[۱،*]، معصومه نجم[۲] و مهدی صدیقی[۳]

**اطلاعات مقاله**

دریافت مقاله: ۱۳۹۹/۰۹/۱۳
پذیرش مقاله: ۱۴۰۰/۰۳/۲۰

**واژگان کلیدی:**
سنتز سطح بالا،
بهینه‌سازی تاخیر،
بهینه‌سازی مساحت،
معماری قابل بازپیکربندی،
مدارهای حسابی.

**چکیده**

افزایش قابلیت‌های مدارهای مجتمع و پیچیدگی برنامه‌های کاربردی، روش‌ها و ابزارهای طراحی سخت‌افزار را به سمت سطوح بالاتری از انتزاع سوق داده است. سنتز سطح بالا یکی از کلیدی‌ترین گام‌ها در افزایش سطح انتزاع است و هر چه توصیف اولیه در کاربرد مورد نظر چکیده‌تر باشد، سنتز سطح بالا کارایی بیشتری خواهد داشت. کاربردهای حسابی از جمله کاربردهایی هستند که ورودی اولیه در آنها بسیار چکیده است. در سال‌های اخیر، تحقیقات گسترده‌ای در زمینه طراحی ساختارهای قابل بازپیکربندی حسابی صورت گرفته است. از آنجا که از یک سو استفاده موثر از این ساختارها وابسته به وجود الگوریتم‌ها و ابزارهای مناسب جهت پیاده‌سازی طراحی بر روی سخت‌افزار بوده و از سوی دیگر، پژوهش در زمینه توسعه این دسته از الگوریتم‌ها بسیار اندک و محدود بوده است، در این مقاله روش‌هایی برای بهینه‌سازی سنتز خودکار مدارهای حسابی بر روی یک معماری قابل بازپیکربندی درشت‌دانه شامل بهینه‌سازی نگاشت، تاخیر و مساحت ارائه خواهد شد. بستر انتخاب شده برای اجرای الگوریتم پیشنهادی، معماری قابل بازپیکربندی درشت‌دانه DARA می‌باشد که برای حساب دهدهی بهینه‌سازی شده است. نتایج نشان می‌دهد که پیاده‌سازی برنامه محک TELCO بر روی این معماری با استفاده از روش‌های بهینه‌سازی پیشنهادی منجر به بهبود حدود ۳۰ درصدی در مساحت می‌گردد.

## ۱- مقدمه

با افزایش پیچیدگی عملیاتی سامانه‌های دیجیتال، طراحی این سامانه‌ها دشوارتر شده و دیگر توصیف سطح انتقال ثبات[۲] به سادگی پاسخگوی چالش‌های پیش روی طراحان نیست. بنابراین در سالهای اخیر تلاش‌های قابل ملاحظه‌ای برای افزایش سطح انتزاع مدل‌سازی و طراحی سیستم‌های دیجیتال انجام شده است و روش‌های طراحی دیجیتال به سنتز سطح بالا[۳] تکامل یافته‌اند [۱].

سنتز سطح بالا به مجموعه‌ای از عملیات گفته می‌شود که یک توصیف سطح بالا از عملکرد مدار را به توصیف سطح انتقال ثبات آن تبدیل می‌کند [۲]. افزایش سطح انتزاع، تولید سریع یک سخت‌افزار در سطح RTL را با سرعت، مساحت یا توان مصرفی بهینه امکان‌پذیر می‌کند [۳]. از آنجا که سنتز سطح بالا، طراحی سیستم‌های پیچیده امروزی را تسهیل نموده و زمان رسیدن محصول به بازار را کاهش می‌دهد، به سرعت مورد توجه قرار گرفت [۴].

هر چند سنتز سطح بالا می‌تواند در همه کاربردها مفید واقع شود، اما هر چه توصیف اولیه چکیده‌تر باشد، کارآیی جریان سنتز در کاربرد مورد نظر نمود بیشتری خواهد داشت. از جمله کاربردهایی که ورودی اولیه در آنها بسیار چکیده است، کاربردهای حسابی است. زیرا کاربر تنها یک عبارت حسابی را وارد می‌کند و از نحوه پیاده‌سازی آن بر بستر سخت‌افزار بی‌اطلاع است. از میان کاربردهای حسابی که اغلب نیاز به دقت بالایی دارند، حساب دهدهی از اهمیت

* پست الکترونیک نویسنده مسئول: s_emami@semnan.ac.ir
۱. استادیار، دانشکده مهندسی برق و کامپیوتر، دانشگاه سمنان
۲. پژوهشگر، دانشکده مهندسی کامپیوتر، دانشگاه صنعتی امیرکبیر
۳. استاد، دانشکده مهندسی کامپیوتر، دانشگاه صنعتی امیرکبیر

[2] Register Transfer Level (RTL)
[3] High Level Synthesis (HLS)

ویژه‌ای برخوردار است. زیرا از یکسو بیشتر عملیات حسابی در زندگی امروز بشر، مانند کاربردهای مالی و تجاری، با داده‌های دهدهی انجام می‌شوند و از سوی دیگر، بسیاری از اعداد اعشاری دهدهی مانند ۰/۱ نمی‌توانند نمایش دقیقی در مبنای دو داشته باشند [۵]. برای حفظ دقت در این کاربردها یا باید آن‌ها را با استفاده از کتابخانه‌های حساب دهدهی مبتنی بر نرم‌افزار اجرا نمود و یا از مدارهای سخت‌افزاری مناسب برای عملیات دهدهی پایه استفاده کرد. از آنجا که پیاده‌سازی نرم‌افزاری حساب دهدهی بین ۱۰۰ تا ۱۰۰۰ برابر کندتر از پیاده‌سازی دودویی در سخت‌افزار است، در سال‌های اخیر گرایش به پیاده‌سازی سخت‌افزاری حساب دهدهی افزایش یافته است [۶]. در پیاده‌سازی‌های سخت‌افزاری، طراحی به روش ASIC[1] و استفاده از FPGA[2]ها دو سر طیف انواع پیاده‌سازی‌های ممکن و نقطه مقابل یکدیگر از نظر مساحت، توان، تاخیر، انعطاف‌پذیری، هزینه ساخت و زمان آماده شدن محصول هستند [۹-۷]. هدف یک طراحی خوب، رسیدن به سرعتی شبیه به ASIC و قدرت انعطاف، صرف زمان و هزینه‌ای مشابه FPGA است. افزایش هزینه‌های طراحی و صرف زمان زیاد برای طراحی ASIC از یکسو و کارآیی پایین محاسباتی و سربار بالای مساحت در FPGAها از سوی دیگر، طراحان معماری را وادار به جستجوی جایگزینی برای این معماری‌ها کرده است. معماری‌های قابل بازپیکربندی درشت‌دانه[3] دارای کارایی محاسباتی بالا، هزینه‌ها و صرف زمان پایینی برای طراحی هستند و به همین دلیل جایگزین جذاب و مناسبی برای طراحی در حوزه کاربردهای خاص‌منظوره می‌باشند. درواقع معماری‌های قابل بازپیکربندی درشت‌دانه، مصالحه‌ای[4] بین ASIC و FPGAها به شمار می‌روند، زیرا نسبت به FPGAها بازده محاسباتی بهتر و در مقایسه با ASIC بازده مهندسی بهتری دارند [۸].

با توجه به مزایای استفاده از سنتز سطح بالا در پیاده‌سازی‌های سخت‌افزاری، جایگاه CGRAها در طراحی مدارها و اهمیت حساب دهدهی، این مقاله به ارائه چند روش بهینه‌سازی برای سنتز مدارهای حسابی بر روی معماری‌های قابل بازپیکربندی درشت‌دانه پرداخته و از معماری DARA[5] [۱۰] برای نشان دادن کارآیی روش‌های پیشنهادی استفاده می‌کند.

ساختار مقاله به این شرح می‌باشد: در بخش دوم به مرور کارهای انجام شده در زمینه سنتز و بهینه‌سازی مدارهای حسابی پرداخته می‌شود. در بخش سوم، روش‌های پیشنهادی برای بهینه‌سازی سنتز سطح بالای مدارهای حسابی تشریح می‌گردد. در بخش چهارم، نمونه‌ای از پیاده‌سازی این الگوریتم‌ها بر روی DARA نمایش داده شده و در نهایت در بخش پنجم نتایج ارزیابی روش‌های پیشنهادی ارائه می‌شود.

## ۲- مروری بر کارهای گذشته

در پژوهش‌های مختلف، نگاه متفاوتی به تعریف سنتز مدارهای حسابی وجود دارد. در برخی منابع مانند [۱۱]، مفهوم سنتز در معنای عام تبدیل الگوریتم به مدار و معادل با طراحی مدار با استفاده از عبارات منطقی، جدول کارنو و جدول درستی[6] در نظر گرفته شده است. در [۱۲]، سنتز فرآیندی است که یک توصیف سطح بالای سخت‌افزار را با در نظر گرفتن زمان‌بندی سلول‌ها و محدودیت‌های دیگر، به netlist تبدیل می‌نماید. در [۱۳]، اصطلاح سنتز به معنای عام طراحی با در نظر گرفتن انتخاب‌های مختلف در فضای جواب به کار می‌رود و در [۱۴]، منظور از سنتز الگوریتمی، ارائه‌ی یک الگوریتم برای تبدیل داده‌های ورودی به خروجی است که به سادگی، با استفاده از یک مدار ترکیبی پیاده‌سازی شده است. در [۱۵]، تنها به بیان معماری‌ها و روش‌های به کار رفته در ابزارهای سنتز تجاری برای بهبود طراحی مبتنی بر سلول[7] از نظر زمان، مساحت و توان پرداخته شده است. در این منبع به روند سنتز به کار گرفته شده در این ابزارها، شامل استخراج عملیات حسابی از توصیف انجام شده و خوشه‌بندی آن‌ها در دسته‌های بزرگتر، بهینه‌سازی مسیر داده با استفاده از الگوریتم‌های مختلف حسابی، تولید netlist با استفاده از روش‌های مختلف پیاده‌سازی عملگرها و الگوریتم‌هایی برای کاهش توان مصرفی ناشی از بزرگ بودن اندازه‌ی مدارها و فعالیت بالای سویچ‌ها پرداخته شده است. بعضی منابع، در تعریف سنتز فقط بر روی بهینه‌سازی مدار متمرکز شده‌اند. به عنوان نمونه در [۱۶] با استفاده از ILP[8]، روشی بهینه برای

[1] Application Specific Integrated Circuit
[2] Field Programmable Gate Array
[3] Coarse-Grained Reconfigurable Architecture (CGRA)
[4] Trade-off

[5] Decimal Arithmetic Reconfigurable Architecture
[6] Truth table
[7] Cell-based design
[8] Integer Linear Programming

پیاده‌سازی Ling adder ارائه گردیده است. نویسنده‌ی این مقاله ادعا کرده است که یک فرمول‌بندی ILP بر اساس مدل‌های logical effort ارائه نموده که با کوچک‌سازی فضای جستجو، زمان اجرای الگوریتم را کاهش می‌دهد. منبع دیگری که به جنبه‌ی بهینه‌سازی در سنتز پرداخته است، [۱۷] می‌باشد که سه روش مختلف برای بهینه‌سازی مدارهای حسابی ارائه می‌نماید. در روش اول تلاش شده است تا با استفاده از معادلات و روابط گوناگون، بتوان عملگرهای مختلف را در یک گراف روند داده (DFG[1]) با یکدیگر جابه‌جا نمود. به این ترتیب عملگرهای حسابی در کنار هم و عملگرهای غیر حسابی نیز در مجاورت هم قرار گرفته و لایه‌های مجزای عملیات حسابی و غیر حسابی در گراف روند داده تشکیل می‌شود. در این حالت، بهینه‌سازی جداگانه‌ی هر لایه و در نتیجه کل گراف روند داده تسهیل می‌گردد. در روش دوم، با اجرای یک الگوریتم تکرار شونده بر روی بیت‌های ورودی، ماژول‌های بزرگ شکسته شده و ماژول‌هایی ساخت‌یافته با تعداد انشعاب ورودی کمتر تشکیل گردیده است. در نهایت سعی شده است در صورت وجود روابط خاص بین بیت‌های ورودی، از آن‌ها برای ساده‌سازی بیشتر مدار و رسیدن به جواب نزدیک به بهینه استفاده گردد. با توجه به کاربرد گسترده‌ی مدارهای حسابی و اهمیت بهینه‌سازی این مدارها، برخی از پژوهش‌های جدید به ابعاد تازه‌تری از این مسئله تمرکز کرده‌اند. به عنوان مثال، [۱۸] به بهبود روش‌های بهینه‌سازی کامپایلر برای عبارات چند جمله‌ای حسابی پرداخته و [۱۹] معماری بلوک‌های سازنده FPGAها را برای عملیات حسابی بهبود بخشیده است. برای پیاده‌سازی یک تابع ریاضی، توسعه-دهنده باید از بین گزینه‌های مختلف پیاده‌سازی (مانند پهنای کلمه، سیستم عددی، فرم‌های محاسباتی دیگر، سریال یا موازی بودن محاسبات، خط لوله، ریز/ درشت دانه‌گی[2] معماری قابل بازپیکربندی و غیره)، بهترین انتخاب را انجام دهد. این انتخاب‌ها نتایج مختلفی را از نظر کارآیی، بهره‌وری استفاده از منابع، دقت و توان تولید می‌کنند. بهترین روش پیاده‌سازی، به معماری هدف و کاربرد خاص مورد نظر وابسته است. برای به دست آوردن بهترین مصالحه با توجه به این دو پارامتر، روش‌های مختلف محاسباتی باید ارزیابی گردند. با توجه به جایگاه و اهمیت بهینه‌سازی در زمینه‌های علمی گوناگون [۲۲-۲۰]، در ادامه این مقاله، به ارائه چند روش بهینه‌سازی برای سنتز خودکار مدارهای حسابی پرداخته می‌شود و سپس به پیاده‌سازی روش‌های ارائه شده بر روی DARA به عنوان معماری هدف خواهیم پرداخت.

## ۳- روش‌های بهینه‌سازی پیشنهادی

از آنجا که معمولا الگوریتم‌های سنتز بر روی یک گراف روند داده ورودی تعریف می‌شوند، در این مقاله نیز فرض می‌شود که مدار حسابی ورودی به شکل یک گراف روند داده از عملیات حسابی شامل جمع و ضرب توصیف گردد. نکته‌ای که در اینجا باید به آن اشاره کرد این است که افزودن عملیات تفریق و تقسیم به عبارت حسابی ورودی با جزئیات اندکی قابل اعمال بوده و به دلیل ساده‌تر شدن الگوریتم‌ها و مثال‌های ارائه شده در این مقاله از آنها صرف‌نظر شده است. بنابراین الگوریتم پیشنهادی، یک گراف روند داده از عملیات حسابی را به عنوان ورودی دریافت کرده و آن را برای پیاده‌سازی بر روی معماری قابل بازپیکربندی درشت‌دانه هدف بهینه می‌کند. هدف اصلی در این پژوهش، کمینه کردن تأخیر تحت محدودیت منابع است.

الگوریتم سنتز، برای پیاده‌سازی مدار ورودی توصیف شده بر روی معماری مورد نظر از کتابخانه‌ای از مدارهای حسابی استفاده می‌کند. در واقع عناصر موجود در کتابخانه جریان سنتز، جمع‌کننده‌ها و ضرب‌کننده‌های مختلفی هستند که هر یک از آنها از چیدمان متفاوتی از بلوک‌های معماری درشت‌دانه هدف تشکیل شده و در نتیجه دارای تاخیر و مساحت متفاوتی می‌باشند. در مرحله اول به منظور بررسی وجود جواب برای مسئله، نوع همه گره‌های گراف روند داده به کندترین عناصر درون کتابخانه جریان سنتز تنظیم شده و تعداد منابع مصرفی محاسبه می‌شود. اگر منابع مورد نیاز با کندترین مدارها که کمترین منابع را مصرف می‌کنند بیشتر از منابع موجود باشد، این طراحی قابل جاینشانی و نگاشت بر روی معماری مورد نظر نبوده و الگوریتم در همان ابتدا خاتمه می‌یابد. در صورتی که امکان جاینشانی وجود داشته باشد، ابتدا با استفاده از الگوریتم ASAP[3] تأخیر مسیر بحرانی با کندترین مدارها اندازه‌گیری شده و به عنوان حد بالایی برای تأخیر در نظر گرفته می‌شود. سپس تمام گره‌های گراف روند داده به سریع‌ترین مدارهای حسابی

[1] Data Flow Graph
[2] Fine vs. coarse-grained
[3] As Soon As Possible

تنظیم شده و تأخیر مسیر بحرانی گراف روند داده و منابع مورد نیاز با سریع‌ترین ماژول‌ها به دست می‌آید. اگر تعداد منابع با سریع‌ترین ماژول‌ها از منابع کل موجود کمتر باشد، گراف روند داده با هدف تأخیر و سپس مساحت بهینه‌سازی می‌شود. اما اگر منابع به اندازه کافی نباشد، ابتدا باید گراف روند داده با هدف نگاشت بهینه‌سازی شود، به طوریکه با حداقل تغییرات در نوع ماژول‌ها و کمترین افزایش تأخیر، گراف روند داده قابل جاینشانی روی معماری باشد.

شبه‌کد مربوط به الگوریتم پیشنهادی در شکل (۱) نشان داده شده است. با توجه به توضیحات ارائه شده، الگوریتم بهینه‌سازی پیشنهادی شامل سه گام اصلی بهینه‌سازی نگاشت، بهینه‌سازی تاخیر و بهینه‌سازی مساحت می‌باشد و از آنجا که در دو گام بهینه‌سازی نگاشت و بهینه‌سازی مساحت، از گراف پیمایش فضای طراحی[1] استفاده می‌شود، ابتدا به توصیف این گراف و چگونگی تشکیل آن خواهیم پرداخت.

```
Obtimization_Algorithm(DFG, Existing_BB)
{
      Integer BB_with_Slow_Res;
      Initialize_Nodes_To_Slowest_Res(DFG);
      BB_with_Slow_Res = CalCulating_BB(DFG);
      If BB_with_Slow_Res > Existing_BB Then
            //Mapping Is Impossible;
      Else
            Integer Delay_with_Fast_Res;
            Integer Delay_with_Slow_Res;
            Integer BB_with_Fast_Res;
            Delay_with_Slow_Res = ASAP(DFG);
            Initialize_Nodes_To_Fastest_Res(DFG);
            Delay_with_Fast_Res = ASAP(DFG);
            BB_with_Fast_Res = CalCulating_BB(DFG);
            If BB_with_Fast_Res <= Existing_BB Then
                  ALAP(DFG, CP);
                  Optimize_For_Delay(DFG, Delay_with_Fast_Res);
                  Optimize_For_Area(DFG, Delay_with_Fast_Res);
                  //Optimizing Is Finished Successfully;
            Else
                  For CP From Delay_with_Fast_Res To Delay_with_Slow_Res
                        Integer BB;
                        Boolean Flag;
                        ALAP(DFG, CP);
                        Flag = Optimize_For_Mapping(DFG, CP);
                        If Flag = true
                              Optimize_For_Delay(DFG, CP);
                              Optimize_For_Area(DFG, CP);
                              //Optimizing Is Finished Successfully;
                        Else
                              Initialize_Nodes_To_Fastest_Res(DFG);
}
```

شکل ۱- شبه‌کد الگوریتم بهینه‌سازی پیشنهادی

### ۲-۱-گراف پیمایش فضای طراحی

این گراف به ازای هر مسیر از گراف روند داده تشکیل شده و فضای طراحی را برای تغییر انواع عملیات موجود در آن مسیر به منابع کندتر، بدون نقض تأخیر مورد نظر، جستجو می‌کند.

الگوریتم تشکیل گراف، تمام حالت‌های ممکن برای تغییر نوع گره‌ها به ماژول‌های کندتر را تا زمانی که slack (تفاضل زودترین و دیرترین زمان ممکن اجرای آن ماژول نسبت به مسیر بحرانی) برای مسیر باقی بماند، بررسی می‌کند. برای آنکه تمام انتخاب‌های ممکن بررسی شوند، به ازای هر انتخاب برای هر گره از یک نوع عملیات مانند ضرب‌کننده، باید تمام انتخاب‌های گره‌های دیگر از انواع دیگر مثل جمع‌کننده و غیره بررسی شوند. اما چنانچه گره‌ها از یک نوع باشند، اگر گره اول به هر نوعی از مدارهای کندتر تغییر یابد، گره دوم باید از همان جنس یا کندتر باشد.برای نشان دادن نوع عملیات درون مسیر، به ازای هر نوع عملیات، یک آرایه تعریف می‌شود. تعداد عناصر این آرایه‌ها معرف تعداد عملیات موجود در مسیر از نوع مشخص می‌باشد.

[1] Design Space Exploration Graph

اگر مدارهای موجود در کتابخانه سنتز (برای هر عملیات) به ترتیب تأخیر مرتب شوند و به هر کدام از آنها عددی بین صفر تا m-1 اختصاص داده شود، که m تعداد ماژول‌های کتابخانه از نوع عملیات مورد نظر باشد، هر عنصر آرایه مربوط به عملیات مورد نظر نیز می‌تواند یک عدد بین صفر (سریعترین ماژول) تا m-1 (کندترین ماژول) باشد که بیانگر نوع ماژول از عملیات مربوطه است. برای هر گره از گراف، دو متغیر تعریف می‌شود که محتوای یکی از آنها slack باقی‌مانده برای مسیر و دیگری تعداد بلوکهای مصرفی مسیر از معماری مورد نظر است. چنانچه slack باقی‌مانده گره، صفر باشد، یکی از پاسخ‌های ممکن برای مسئله بهینه‌سازی به دست آمده است که به لیست پاسخ‌ها اضافه می‌شود. اگر slack منفی به دست آید، این مسیر از گراف پیمایش فضای طراحی، دیگر گسترش نخواهد یافت و اگر slack بزرگتر از صفر باشد، بررسی این مسیر از گراف ادامه می‌یابد (زیرا هنوز امکان بهینه‌سازی وجود دارد و با توجه به هدف بهینه‌سازی ممکن است این حالت به لیست پاسخ‌ها اضافه شود).

### ۲-۲- بهینه‌سازی نگاشت

در بهینه‌سازی نگاشت تلاش می‌شود تا کمترین تغییرات در نوع گره‌های گراف روند داده ایجاد شده و کمترین تاثیر منفی بر روی تاخیر مدار حاصل گردد. بنابراین، تغییر از مسیرهایی آغاز می‌گردد که دارای بیشترین slack باشند تا با بهینه‌سازی این مسیرها، از تغییر نوع گره‌های مسیرهای دیگر جلوگیری شود. در این الگوریتم، اولین پاسخی که محدودیت منابع را تأمین کند، پذیرفته خواهد شد.

برای شروع عملیات بهینه‌سازی، ابتدا تمام مسیرهای ممکن از ورودی‌های اولیه تا خروجی بر اساس مقدار slack گره اول مسیر به صورت نزولی مرتب می‌شوند. بهینه‌سازی از مسیر با بیشترین slack آغاز می‌گردد و تا زمانی که تعداد منابع مصرفی از منابع موجود کمتر شود ادامه خواهد داشت. بنابراین پیش از بهینه‌سازی هر مسیر، ابتدا منابع مصرفی برای گراف روند داده محاسبه شده و اختلاف آن با منابع موجود در متغیری به نام Deficiency برای آن مسیر ذخیره می‌شود (این متغیر برای هر بلوک موجود در معماری به صورت مجزا تعریف می‌شود). در واقع هدف اصلی این الگوریتم کاهش مقدار متغیر Deficiency می‌باشد که متأثر از نحوه جستجوی گراف پیمایش فضای طراحی و نحوه پذیرش پاسخ‌های به دست آمده از آن خواهد بود.

در هر سطح از گراف پیمایش فضای طراحی، وضعیت یک گره از مسیر مشخص شده و منابع مصرفی به ازای هر تغییر نوع محاسبه و از مقدار منابع مصرفی قبل از بهینه‌سازی برای آن مسیر کسر می‌گردد. باقی‌مانده این تفریق، میزان منابع صرفه‌جویی شده ناشی از این تغییر می‌باشد. اگر این اختلاف را SavedBB بنامیم، حاصل «SavedBB – Deficiency» مقدار Deficiency جدید پس از اعمال تغییر ایجاد شده در نوع یک گره از مسیر خواهد بود (بلوک‌هایی که از ابتدا به تعداد مورد نیاز وجود داشته‌اند در این حاصل دخالتی ندارند).

مقدار جدید Deficiency به دست آمده برای هر نوع بلوک می‌تواند مثبت یا منفی باشد. مقدار منفی برای یکی از انواع بلوک‌های معماری به این معنی است که بیش از مقدار مورد نیاز از منابع مصرفی کاسته شده است و پاسخ به دست آمده از حداقل تغییرات فاصله گرفته است. چنانچه slack گره‌ای از گراف صفر باشد یا با slack غیر صفر به کندترین مدار تغییر یافته باشد، به لیست پاسخ‌ها اضافه خواهد شد. اما مقدار منفی Deficiency آن محاسبه شده و در متغیری ذخیره می‌گردد. با اضافه شدن هر گره به لیست پاسخ‌ها این متغیر نیز به‌روزرسانی می‌شود. چنانچه گره‌ای دارای slack غیر صفر باشد، اما قدر مطلق مقدار منفی Deficiency آن از قدر مطلق مقدار متغیر یاد شده کمتر باشد، آن گره نیز به عنوان یک پاسخ احتمالی به لیست پاسخ‌ها اضافه خواهد شد. در نهایت در هر مرحله، با بررسی هر گره از گراف پیمایش فضای طراحی، اگر slack گره‌ای بزرگتر از صفر و Deficiency آن مساوی صفر باشد، به عنوان پاسخ نهایی پذیرفته خواهد شد.

اگر در میان گره‌های گراف پیمایش فضای طراحی هیچ گره‌ای دارای Deficiency صفر نباشد، آنگاه گراف تا سطح آخر پیمایش خواهد شد. حال از میان پاسخ‌های به دست آمده باید مناسب‌ترین پاسخ انتخاب گردد. منظور از پاسخ مناسب پاسخی است که کمترین مقدار مثبت و بزرگترین مقدار منفی (نزدیک به صفر) را در متغیر Deficiency داشته باشد. بنابراین ابتدا از میان پاسخ‌های به دست آمده لیستی از پاسخ‌ها با کمترین مقدار مثبت Deficiency (صرف نظر از مقدار منفی Deficiency برای هر پاسخ)

تهیه می‌شود (زیرا هدف این الگوریتم کاهش و به صفر رساندن مقدار این متغیر می‌باشد). سپس در لیست به دست آمده تمام پاسخ‌هایی که Deficiency منفی ندارند، جدا می‌شوند (زیرا Deficiency منفی باعث فاصله گرفتن از پاسخی با حداقل تغییرات خواهد شد) و در صورت وجود، پاسخی که بیشترین slack باقی‌مانده را دارد، به عنوان پاسخ نهایی پذیرفته می‌شود. اما اگر چنین پاسخی وجود نداشته باشد، از لیست به دست آمده، تمام پاسخ‌هایی که دارای بزرگترین مقدار منفی (نزدیک به صفر) هستند جدا شده و از بین آنها موردی که دارای بیشترین slack است، به عنوان پاسخ نهایی انتخاب خواهد شد.

در این مرحله، بهینه‌سازی یک مسیر از گراف روند داده به اتمام می‌رسد. اما چالش پیش روی دیگر، نحوه اختصاص انواع به دست آمده از بهینه‌سازی یک مسیر به گراف روند داده می‌باشد. برای مثال، معمولا گره سطح آخر گراف روند داده که به خروجی متصل است، بین چند مسیر مشترک می‌باشد. در این صورت، از آنجا که هر مسیر به صورت جداگانه و بر اساس slack گره اول از مسیر بهینه‌سازی شده است، ممکن است برای پیاده‌سازی آن، چندین انتخاب در مسیرهای مختلف به دست آمده باشد. در چنین شرایطی اگر یک گره برای اولین بار مورد پردازش قرار گرفته باشد، بررسی می‌شود که با توجه به slack گره، امکان تغییر نوع آن به ماژول کندتر وجود دارد یا خیر (اگر slack گره مورد نظر به اندازه کافی باشد تغییر انجام خواهد شد). اما اگر گره‌ای در دو یا چند مسیر به صورت مشترک قرار داشته باشد و قبلا پردازش شده باشد، از بین گزینه‌های به دست آمده، ماژول سریع‌تر انتخاب می‌شود. زیرا قطعا تحت محدودیت تأخیر در یکی از مسیرها نمی‌تواند به ماژول‌های کندتر تغییر یابد. شبه‌کد مربوط به الگوریتم بهینه‌سازی نگاشت در شکل (۲) نشان داده شده است.

- مرتب‌سازی مسیرها بر اساس سیر نزولی slack و شروع بهینه‌سازی از مسیرهای دارای بیشترین slack
- محاسبه منابع مصرفی هر مسیر از DFG و اختلاف آن با منابع موجود (Deficiency)
- به ازای هر گره از گراف جستجو در هر سطح:
  - محاسبه‌ی منابع مصرفی به ازای هر تغییر نوع گره
  - کسر عدد به دست آمده از مقدار منابع مصرفی قبل از بهینه‌سازی مسیر ← میزان صرفه‌جویی شده (SavedBB)
  - محاسبه مقدار Deficiency جدید پس از تغییر نوع گره (Deficiency – SavedBB)
  - محاسبه Slack جدید برای گره تغییر یافته
  - تصمیم‌گیری در مورد «ادامه/عدم ادامه مسیر گراف/ اضافه‌شدن به لیست پاسخ‌ها» با توجه به Deficiency و Slack جدید

شکل ۲- شبه‌کد الگوریتم بهینه‌سازی نگاشت

### ۲-۳- بهینه‌سازی تاخیر

در الگوریتم پیشنهادی، تمام زیرعبارت‌ها از یک نوع عملیات به صورت درخت کامل ساخته می‌شوند. زیرا در صورتیکه تأخیر تمام گره‌ها یکسان باشد، این ساختار در بین تمام ساختارهای معادل دارای کمترین تأخیر ممکن خواهد بود. اما اگر برخی گره‌های این ساختار پس از بهینه‌سازی به انواع دیگر تغییر یابند، لزوما کمترین تأخیر به دست نخواهد آمد.

در عمل تشخیص این که از ابتدا کدام ساختار برای بهینه‌سازی مناسب‌تر است، غیر ممکن می‌باشد. زیرا زمانی که تعداد گره‌های گراف روند داده افزایش می‌یابد، تعداد ساختارهای معادل برای آن به صورت نمایی افزایش یافته و ترکیب آنها از عملیات ضرب و جمع در سطوح مختلف نیز فضای بزرگتری ایجاد می‌کند. به همین دلیل، در این پژوهش، الگوریتم بهینه‌سازی با استفاده از ساختار درخت کامل آغاز می‌گردد، اما پس از بهینه‌سازی نگاشت که حداقل تغییرات جهت نگاشت روی گراف روند داده اعمال می‌شود، گراف روند داده با هدف تأخیر بهینه‌سازی شده و در صورت نیاز ساختار اولیه آن تغییر می‌یابد.

هدف از بهینه‌سازی تأخیر، کاهش تأخیر مسیرهای بحرانی یا کاهش تعداد آنهاست. کاهش تأخیر مسیرهای بحرانی در راستای هدف مشخص شده الگوریتم پیشنهادی بوده، در حالیکه کاهش تعداد مسیرهای بحرانی باعث افزایش slack مسیرها شده و امکان بهینه‌سازی بیشتری را برای کاهش منابع مصرفی فراهم می‌آورد.

به طور کلی می‌توان دو نوع جابجایی برای تغییر ساختار یک گراف روند داده تعریف کرد: جابجایی گره‌ها و جابجایی زیرگراف‌ها. جابجایی گره‌ها تنها بر روی گراف‌های روند

داده‌ای قابل اعمال است که شامل یک نوع عملیات حسابی باشند. بنابراین اگر یک گراف روند داده شامل عملیات مختلف باشد، این نوع جابجایی تنها بر روی زیرگراف‌های سطح آخر که شامل ورودی‌های اولیه مدار هستند، قابل اعمال خواهد بود.

برای جابجایی گره‌ها، ابتدا گراف روند داده سطح‌بندی شده و فهرستی از تمام مسیرهای بحرانی تشکیل می‌گردد. سپس در مسیرهای به دست آمده، تمام گره‌هایی که slack فرزند چپ و راست آنها باهم متفاوت است یا یکی از فرزندان آنها از ورودی‌های اولیه مدار می‌باشد، در فهرست جداگانه‌ای ذخیره می‌شوند. از بین گره‌های تعیین شده گره‌ای که در سطحی با شماره کمتر قرار گرفته است، برای عمل بهینه‌سازی انتخاب می‌گردد. چنانچه چند گزینه در یک سطح وجود داشته باشد، گره دارای تعداد گره‌های بیشتر در سطوح ماقبل، انتخاب می‌شود. اگر امکان بهینه‌سازی برای گره منتخب وجود نداشته باشد، گره‌ای در همان سطح (در صورت وجود) و یا در سطح بعدی انتخاب خواهد شد.

شرط لازم برای جابجایی این است که مجموع تأخیر گره انتخاب شده و تأخیر شاخه دارای slack غیر صفر، از تأخیر شاخه با slack صفر کمتر باشد. برای این کار لیستی از گره‌های سطح اول (به جز گره‌هایی که قرار است جابجا شوند) تهیه شده و تعداد آنها محاسبه می‌گردد. در این حالت، در صورت برقراری نامعادله (۱)، امکان جابجایی وجود دارد و بهینه‌سازی انجام خواهد شد و گره‌ها به ترتیب به گره‌های سطح اول اضافه می‌شوند (گره‌ها ابتدا به گره‌هایی که دارای slack بیشتر هستند اضافه می‌شوند).

$$\max(D_n) \times \left\lceil \frac{n}{m \times 2} \right\rceil < \min(S_m) \qquad (۱)$$

در نامعادله (۱)، $n$ نشان‌دهنده‌ی تعداد گره‌های در حال جابجایی، $m$ تعداد گره‌های سطح اول، $D_n$ مجموعه تأخیر گره‌هایی که قرار است جابجا شوند و $S_m$ مجموعه slack گره‌های سطح اول از گراف روند داده می‌باشند.

پس از اتمام هر مرحله، الگوریتم ALAP[1] با تأخیر جدید اعمال می‌شود تا slack گره‌ها به‌روزرسانی گردد. این مراحل تا زمانی که گره انتخاب شده گره ریشه باشد یا رابطه (۱) نقض شود، ادامه می‌یابد. پس از آنکه دیگر هیچ گره‌ای قابل جابجایی نباشد، مرحله دوم یا همان جابجایی زیرگراف‌ها آغاز می‌شود. همانند مرحله قبل، شرط لازم برای جابجایی دو زیرگراف این است که در اثر جابجایی، تأخیر گراف روند داده افزایش نیابد و در صورت عدم تغییر در تاخیر، از تعداد مسیرهای بحرانی کاسته شود. در غیر این صورت جابجایی انجام نخواهد شد.

برای انجام این کار، ابتدا فهرستی از همه مسیرها با slack صفر تهیه شده و از بین مسیرهای انتخاب شده، گره‌های دارای slackهای متفاوت مشخص می‌شوند. سپس فهرستی از تمام زیرگراف‌های متصل به این گره‌ها، به جز زیرگراف‌های دارای slack صفر، ایجاد شده و از میان آنها زیرگرافی که دارای کمترین زمان اجرا است، انتخاب می‌شود. اگر زمان اجرای زیرگراف انتخاب شده از زمان اجرای زیرگراف با slack صفر کمتر یا مساوی باشد، دو زیرگراف به صورت موقت جابجا می‌شوند. در غیر این صورت، امکان جابجایی برای زیرگراف مورد نظر (زیرگراف با slack صفر) وجود ندارد و این فرآیند برای انتخاب‌های دیگر تکرار می‌شود. در صورت جابجایی باید شرط لازم برای جابجایی نیز بررسی گردد. یعنی اگر با جابجایی انجام شده تأخیر گراف روند داده کاهش نیابد یا تعداد مسیرهای بحرانی نسبت به قبل تغییری نکند، جابجایی انجام شده لغو شده و زیرگراف‌ها به موقعیت قبلی باز می‌گردند. پس از هر جابجایی، بار دیگر مسیرهای بحرانی و گره‌های با slack متفاوت درون این مسیرها مشخص شده و این فرآیند تا جایی تکرار می‌گردد که به ازای تمام گزینه‌های ممکن، دیگر هیچ زیرگرافی جابجا نشود.

روش‌های بیان شده (جابجایی گره‌ها و زیرگراف‌ها) تنها برای بهینه‌سازی تاخیر در گراف‌های روند داده‌ای که شامل یک نوع عملیات هستند، کاربرد دارند. حال اگر گراف روند داده شامل عملیات مختلف باشد، بهینه‌سازی تاخیر و جابجایی عملیات نباید منجر به تغییر در حاصل عبارت شود. بنابراین، بهینه‌سازی به صورت سطح به سطح و از سطح آخر به اول انجام خواهد شد. همچنین در هر سطح، هر زیرگراف با توجه به تأخیر اجرای خودش و نه موقعیتش در کل گراف روند داده، بهینه‌سازی می‌گردد. به عبارت دیگر slackها به صورت محلی محاسبه می‌شوند.

همانطور که پیش‌تر گفته شد، سطح آخر از گراف روند داده

[1] As Late As Possible

شامل زیرگراف‌هایی است که دارای ورودی‌های اولیه مدار هستند، پس برای تمام زیرگراف‌های سطح آخر، بهینه‌سازی به صورت مستقل انجام شده و امکان جابجایی از هر دو نوع بررسی خواهد شد. پس از بهینه‌سازی سطح آخر، برای سطوح بعدی، زیرگراف‌های سطح قبل به یک گره بزرگ (دسته‌ای از گره‌ها) تبدیل می‌شود که تأخیر آن برابر زمان اجرای طولانی‌ترین مسیر داخلی آن خواهد بود. این کار باعث می‌شود تا در حین اجرای الگوریتم جابجایی زیرگراف‌ها، سطح‌بندی عملیات حفظ شده و هیچ زیرگرافی از سطوح بالاتر به سطوح بعدی جابجا نشود. پس از دسته‌بندی گره‌های سطح قبل در هر مرحله، الگوریتم جابجایی زیرگراف‌ها بر روی تمام زیرگراف‌های سطح جاری اعمال می‌گردد. شبه‌کد مربوط به الگوریتم بهینه‌سازی تاخیر در شکل (۳) نشان داده شده است.

۱. جابجایی در گره‌ها
- سطح‌بندی DFG و تشکیل لیستی از تمام مسیرهای بحرانی
- در مسیرهای بدست آمده، جداسازی تمام گره‌هایی که slack فرزند چپ و راست آنها باهم متفاوت یا یکی از فرزندان آنها از ورودی‌های اولیه مدار باشند.
- انتخاب گره قرار گرفته در سطحی با شماره کمتر برای عمل بهینه‌سازی
- بررسی شرط لازم برای جابجایی دو گره (مجموع تأخیر گره انتخاب شده و تأخیر شاخه دارای slack غیر صفر، از تأخیر شاخه با slack صفر کمتر باشد.)
- تهیه لیستی از گره‌های سطح اول (بجز گره‌هایی که قرار است جابجا شوند) و بررسی امکان جابجابجایی آنها با استفاده از رابطه $\max(D_n) \times \left\lceil \frac{n}{m \times 2} \right\rceil < \min(S_m)$
- در پایان هر مرحله، اعمال الگوریتم ALAP با تأخیر جدید به منظور به‌روز رسانی slack گره‌ها (ادامه کار تا رسیدن به گره ریشه یا نقض رابطه فوق)

۲. جابجایی زیرگراف‌ها
- تهیه لیستی از همه مسیرها با slack صفر و از بین مسیرهای انتخاب شده، جداسازی گره‌های دارای فرزند چپ و راست با slackهای متفاوت
- ایجاد لیستی از تمام زیرگراف‌هایی که به این گره‌ها متصل‌اند (به جز زیرگراف‌های دارای slack صفر)
- آغاز بهینه‌سازی از گره‌های بالاترین سطح (نزدیک به ورودی‌های اولیه)
- جابجایی موقت زیرگراف‌ها در صورت کمتر یا مساوی بودن زمان اجرای زیرگراف انتخاب شده از زمان اجرای زیرگراف با slack صفر
  - در غیر این صورت، عدم امکان جابجایی و تکرار فرآیند برای انتخاب‌های دیگر
  - در صورت جابجایی، بررسی شرط لازم برای جابجایی قطعی (کاهش تأخیر DFG یا کاهش تعداد مسیرهای بحرانی)
- تعیین مسیرهای بحرانی و گره‌های با slack متفاوت درون این مسیرها بعد از هر جابجایی و تکرار این فرآیند تا بررسی تمام گزینه‌های ممکن

شکل ۳- شبه‌کد الگوریتم بهینه‌سازی تاخیر

### ۲-۴- بهینه‌سازی مساحت

گراف روند داده حاصل از بهینه‌سازی تأخیر، قابل جاینشانی بر روی معماری مورد نظر می‌باشد، اما این گراف روند داده با حداقل تغییرات در نوع گره‌ها به دست آمده است و امکان تغییر نوع برخی از گره‌های آن به ماژول‌های کندتر وجود دارد. هدف از الگوریتم بهینه‌سازی مساحت این است که با حفظ تأخیر به دست آمده برای گراف روند داده، منابع مصرفی به حداقل برسد.

اساس کار این الگوریتم مانند الگوریتم بهینه‌سازی نگاشت، تشکیل یک گراف پیمایش فضای طراحی و پیمایش آن است. نحوه اختصاص انواع عملیات به گره‌ها در هر دو حالت یکسان بوده و تفاوت اصلی آنها در نحوه پیمایش و پذیرش پاسخ‌های به دست آمده می‌باشد. در الگوریتم بهینه‌سازی نگاشت، هدف رسیدن به حداقل تغییرات است، در حالی که در الگوریتم بهینه‌سازی مساحت حداکثر تغییرات ممکن مد نظر می‌باشد. بنابراین باید به ازای تمام مسیرهای موجود در گراف روند داده، گراف پیمایش فضای طراحی تشکیل شده و از بین پاسخ‌های به دست آمده، پاسخی که دارای کمترین مجموع بلوک‌های پایه است، انتخاب گردد. به عبارت دیگر، پاسخی پذیرفته می‌شود که دارای بیشترین مقدار برای متغیر SavedBB باشد.

### ۳-۵- ارزیابی و تحلیل زمانی الگوریتم بهینه‌سازی مسیر

در بخش‌های قبل بیان گردید که یکی از عوامل کاهش

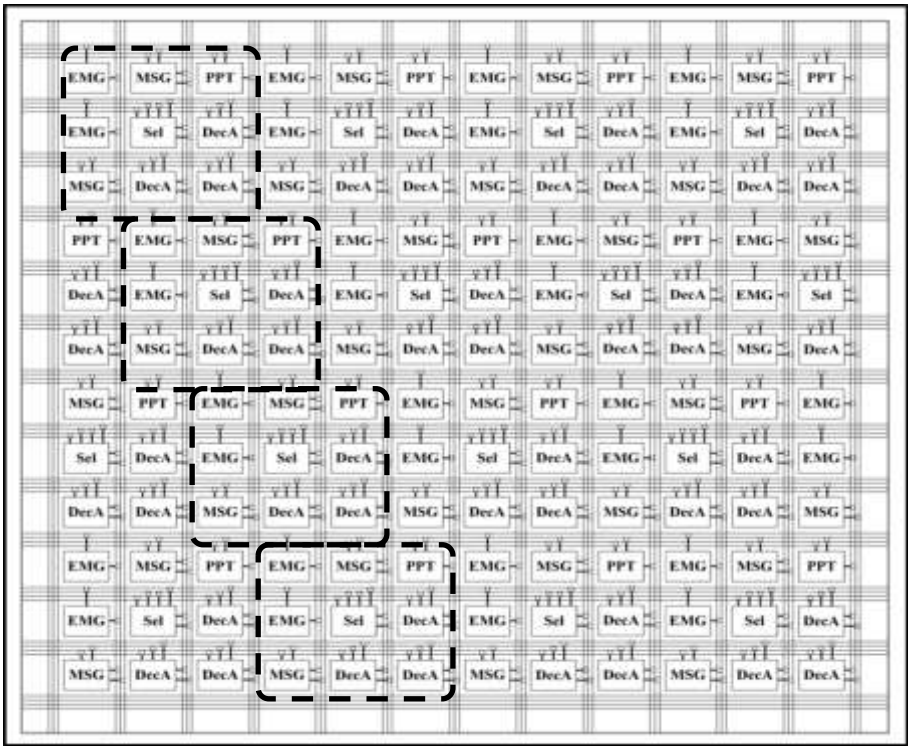
زمان جستجو در گراف پیمایش فضای طراحی، هرس شدن گراف با متغیر slack می‌باشد. زیرا در روش پیشنهادی، تنها شاخه‌های دارای slack غیر صفر گسترش می‌یابند. اما مهمترین عاملی که از رشد نمایی این گراف جلوگیری می‌کند، حذف ترتیب از نوع منابع است. به عنوان مثال، در آرایه‌ای با سه عنصر مانند [۰۰۰]، عنصر اول لزوما بیانگر نوع اولین گره از مسیر نبوده و تنها نوع سه عملیات (جمع/ضرب) را نشان می‌دهد و ترتیب آنها را مشخص نمی‌کند. در واقع پس از به دست آمدن پاسخ مناسب، کندترین مدارهای حسابی به گره‌های ابتدای مسیر و سریع‌ترین‌ها به گره‌های انتهایی اختصاص می‌یابند. بنابراین آرایه‌های [۰۱۲]، [۰۲۱]، [۱۰۲]، [۲۰۱]، [۱۲۰] و [۲۱۰] همگی دارای یک مفهوم‌اند. بنابراین، زمان تشکیل گراف پیمایش فضای طراحی برای بهینه‌سازی مسیرها از مرتبه چندجمله‌ای می‌باشد.

به طور کلی، اگر تعداد عناصر آرایه یا همان تعداد عملیات مسیر برابر $n$ و تعداد پیاده‌سازی‌های متفاوت یک عملیات برابر $t$ باشد، مرتبه زمانی جستجوی گراف از مرتبه $O(n^{t-1})$ خواهد بود. به علاوه، اگر در یک مسیر چندین نوع عملیات (ضرب، جمع و غیره) وجود داشته باشد، مرتبه زمانی کل برابر حاصلضرب مرتبه‌های زمانی آنها می‌باشد.

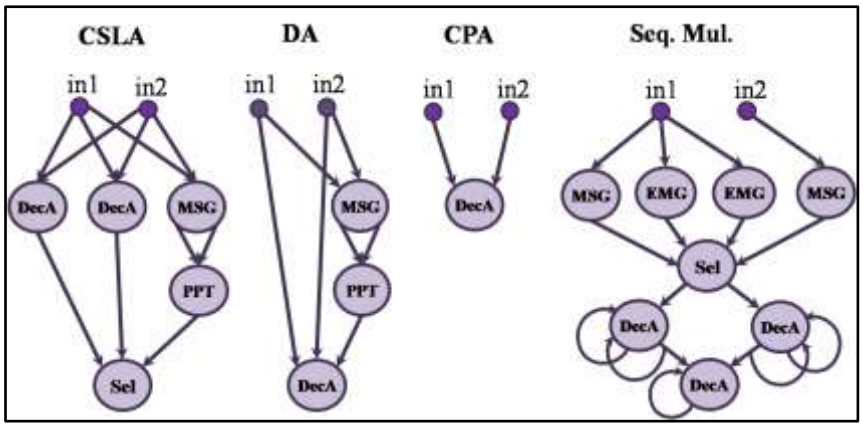
با توجه به اینکه روش‌های بهینه‌سازی پیشنهادی به صورت متوالی بر روی گراف روند داده اولیه اجرا می‌شوند، مرتبه زمان الگوریتم کلی متناسب با بزرگترین مرتبه زمانی از میان سه نوع بهینه‌سازی نگاشت، تاخیر و مساحت خواهد بود. از سوی دیگر همانگونه که بحث شد، بخش اصلی بهینه‌سازی نگاشت و بهینه‌سازی مساحت بر مبنای تشکیل گراف پیمایش فضای حالت است که از مرتبه چند جمله‌ای می‌باشد. به علاوه، با دقت در گام‌های الگوریتم بهینه‌سازی تاخیر مشاهده می‌شود که این الگوریتم نیز از مرتبه چندجمله‌ای می‌باشد. بنابراین روش‌های بهینه‌سازی پیشنهادی در مجموع دارای مرتبه زمانی چندجمله‌ای خواهند بود.

## ۳- مثالی از پیاده‌سازی الگوریتم بهینه‌سازی بر روی معماری DARA

معماری DARA که در این مقاله از آن به عنوان بستر سخت‌افزاری پیاده‌سازی الگوریتم‌های پیشنهادی استفاده می‌شود، در شکل (۴) نشان داده شده است. این معماری از تکرار منظم پنج نوع بلوک مختلف تشکیل شده و برای پیاده‌سازی بهینه عملیات پایه حساب دهدهی طراحی شده است. از میان انواع پیاده‌سازی‌های ممکن برای طراحی جمع‌کننده‌ها و ضرب‌کننده‌ها بر روی معماری DARA مواردی حائز اهمیت هستند که با تأخیر یکسان دارای منابع مصرفی کمتری باشند.




شکل ۴- چینش ۴×۴ از بلوک‌های ۳×۳ از معماری DARA




شکل ۵- ماژول‌های کتابخانه سنتز مبتنی بر معماری DARA

جدول ۱- ویژگی‌های ماژول‌های موجود در کتابخانه سنتز

| کتابخانه نرم | | | | | | | |
|---|---|---|---|---|---|---|---|
| نام ماکرو سلول | تعداد و نوع بلوک‌های مصرفی | | | | | تعداد کل بلوکهای مصرفی | تعداد سیکل‌ها |
| | DecA | PPT | MSG | EMG | Sel | | |
| Carry Select Adder (CSLA) | ۳* | ۱ | ۱ | - | ۱ | ۶* | ۳ |
| Direct Adder (DA) | ۱ | ۱ | ۱ | - | - | ۳ | ۴ |
| Carry Propagate Adder (CPA) | ۱ | - | - | - | - | ۱ | ۳۳ |
| Sequential Multiplier (Seq. Mul.) | ۳ | - | ۲ | ۲ | ۱ | ۸ | ۲۳ |

در شکل (۵) ساختار برخی از این مدارها، بلوک‌های سازنده و نحوه اتصالات داخلی آن‌ها و به عبارت دیگر، گراف روند داده مربوط به عملیات مختلف نشان داده شده است. بلوک‌های سازنده این ماژول‌ها همان بلوک‌های معماری DARA هستند. این مدارهای حسابی می‌توانند به عنوان بخشی از عناصر موجود در کتابخانه جریان سنتز مبتنی بر معماری DARA در نظر گرفته شوند. همچنین، جدول ۱ لیست منابع مورد استفاده در این ماژول‌ها را به همراه تعداد سیکل‌های زمانی آنها نمایش می‌دهد. علامت (*) در جدول به این موضوع اشاره دارد که تعداد بلوک‌های DecA از جمع‌کننده CSLA در واقع دو عدد است، اما به هنگام نگاشت، یک بلوک DecA دیگر نیز به خاطر منابع مسیریابی مصرف خواهد شد و تعداد بلوک‌های CSLA در مجموع به شش عدد می‌رسد.

برای پیاده‌سازی مثالی از مدارهای حسابی بر روی معماری DARA و نحوه عملکرد الگوریتم‌های بهینه‌سازی پیشنهاد شده بر روی آن، از عبارت نشان داده شده در رابطه (۲) استفاده می‌شود.

$$X_{00} = \big((A \times B) + C\big) + \big((D \times E \times F) + ((G + H) \times \big((I + J) + (K + L)\big)\big) \quad (۲)$$

شکل (۶) گراف روند داده معادل این عبارت حسابی را به صورت سطح‌بندی شده نشان می‌دهد. همان طور که مشاهده می‌شود، این عبارت دارای ۷ جمع‌کننده و ۴ ضرب‌کننده است. طبق جدول ۱ در کتابخانه سنتز، کندترین جمع‌کننده CPA بوده و تنها یک نوع ضرب‌کننده موجود است. بنابراین چنانچه گراف روند داده مورد نظر به کندترین ماژول‌ها تنظیم شود، کل طراحی به ۱۹ بلوک DecA، ۸ بلوک EMG، ۸ بلوک MSG و ۶ بلوک Sel برای پیاده‌سازی نیاز دارد که با توجه به شکل (۴)، این تعداد از بلوک‌ها در معماری موجود بوده و این طراحی قابل جاینشانی روی معماری مورد نظر خواهد بود.

همانطور که گفته شد، گراف روند داده شکل (۶) به دلیل کافی بودن منابع موجود، بی‌نیاز از بهینه‌سازی نگاشت می‌باشد. اما برای شرح بهتر الگوریتم بهینه‌سازی، مسیری از این گراف روند داده انتخاب می‌شود تا روند انجام

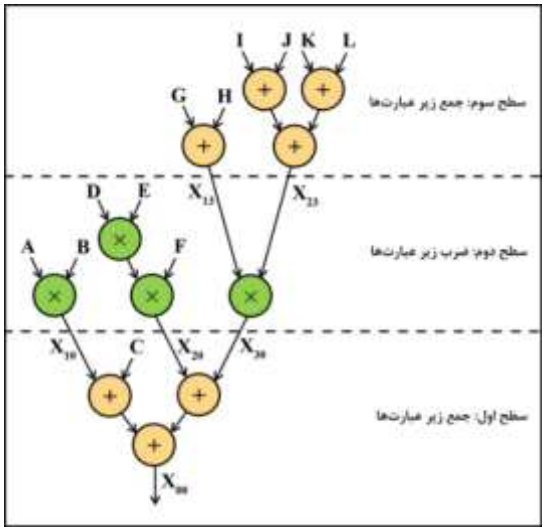
بهینه‌سازی مسیری متشکل از انواع عملیات، نشان داده شود. مسیر انتخاب شده، با دایره‌های رنگی در شکل (۷) مشخص شده است. در این شکل، عدد داخل گره‌ها نشان‌دهنده‌ی تاخیر گره مورد نظر و عدد کنار آنها بیانگر slack گره می‌باشد. با توجه به تعریف slack، مقادیر این پارامتر از تفاوت زمانی میان اجرای الگوریتم‌های ASAP و ALAP بر روی گراف روند داده به دست آمده است.

برای نشان دادن نوع عملیات درون این مسیر، دو آرایه تعریف شده که یکی از آنها مربوط به عملیات جمع و دیگری مربوط به عملیات ضرب است. این آرایه‌ها به ترتیب به صورت A[...] و M[...] نشان داده شده و تعداد عناصر آنها معرف تعداد عملیات جمع و ضرب موجود در مسیر می‌باشد. در مسیر مشخص شده در شکل (۷) سه گره از نوع عملیات جمع و یک عملیات ضرب وجود دارد که در ابتدا همه آنها از سریع‌ترین انواع ماژول‌های کتابخانه انتخاب می‌شود. در کتابخانه سنتز، سه نوع جمع‌کننده برای عملیات جمع موجود است که سریع‌ترین آنها CSLA و کندترین CPA است. بنابراین با توجه به قرارداد بیان شده برای عددگذاری ماژول‌ها، به CSLA عدد صفر، به DA عدد یک و به CPA عدد دو اختصاص داده می‌شود. در نتیجه آرایه‌های تعریف شده در آغاز به صورت A[۰۰۰] و M[۰] خواهند بود. برای بهینه‌سازی، دو متغیر نیز مورد نیاز است که محتوای یکی از آنها slack باقی‌مانده برای مسیر و دیگری تعداد بلوک‌های مصرفی مسیر از هر پنج نوع بلوک موجود در معماری را نشان می‌دهد. گراف پیمایش فضای طراحی تشکیل شده برای این مسیر، در شکل (۸) قابل مشاهده است. همانطور که در این شکل مشخص شده است، slack مسیر در ابتدا برابر ۲۰ بوده و با سریع‌ترین ماژول‌ها، در مجموع شامل ۲۳ بلوک (۹ بلوک DecA، ۵ بلوک MSG، ۲ بلوک EMG، ۴ بلوک Sel و ۳ بلوک PPT) می‌باشد.

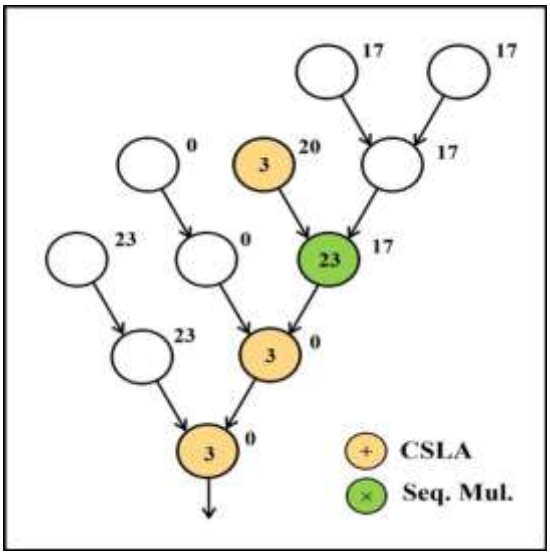
همان طور که شرح داده شد، در شکل (۶) با تغییر گره‌ها به سریع‌ترین مدارها و اعمال الگوریتم ASAP، چون تعداد منابع مورد نیاز سریع‌ترین ماژول‌ها از تعداد منابع موجود در معماری کمتر است، نیازی به بهینه‌سازی با هدف نگاشت نبوده و تأخیر مسیر بحرانی ۵۲ کلاک می‌باشد. در این مرحله باید گراف روند داده با استفاده از الگوریتم ALAP زمان‌بندی و slack مسیرها مشخص شود و پس از آن، بهینه‌سازی تأخیر و مساحت بر روی آن اعمال گردد.

شکل (۹) گراف روند داده را پس از اعمال الگوریتم ALAP نشان می‌دهد.

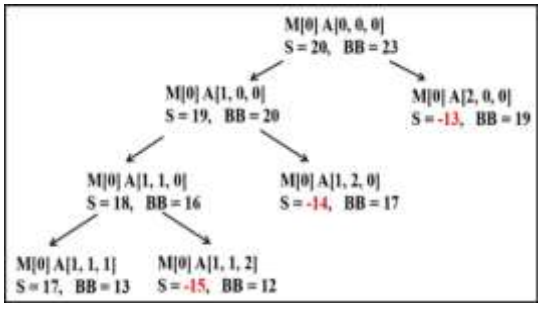
در این شکل نیز، اعداد کنار گره‌ها slack مسیرها و اعداد داخل گره‌ها تأخیر آن گره را نشان می‌دهند.



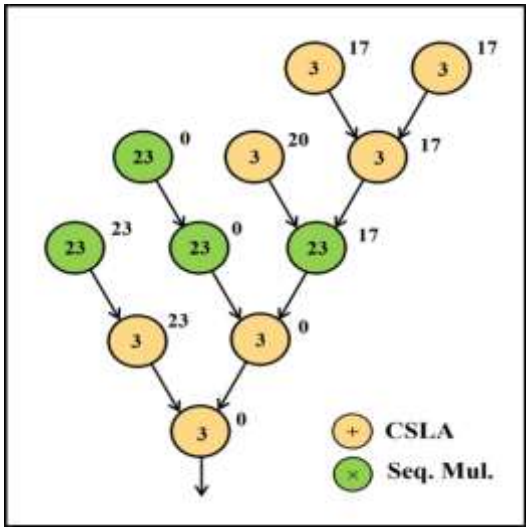

شکل ۶- گراف روند داده سطح‌بندی شده معادل رابطه (۲)




شکل ۷- مسیر انتخاب شده از گراف روند داده برای اعمال بهینه‌سازی



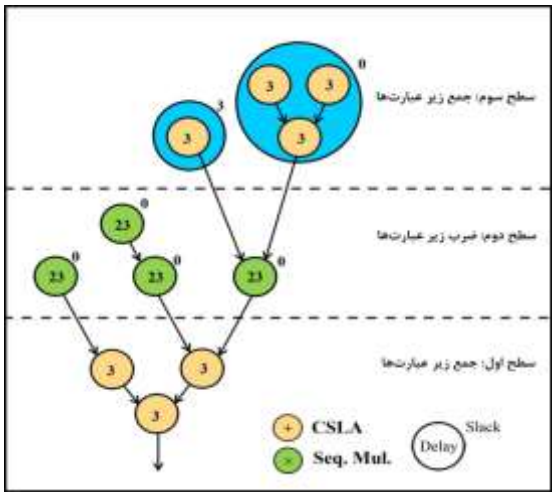

شکل ۸- گراف پیمایش فضای طراحی برای مسیر مشخص شده در شکل (۷)



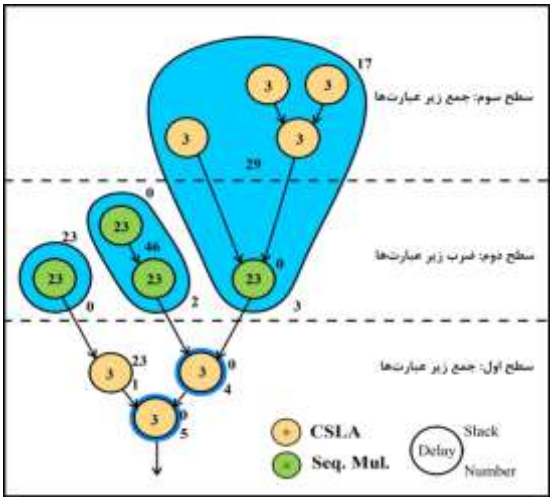

شکل ۹- اعمال الگوریتم ALAP بر روی گراف شکل (۸)

مراحل انجام بهینه‌سازی تاخیر با استفاده از جابجایی گره‌ها و زیرگراف‌ها در شکل (۱۰) نشان داده شده است.

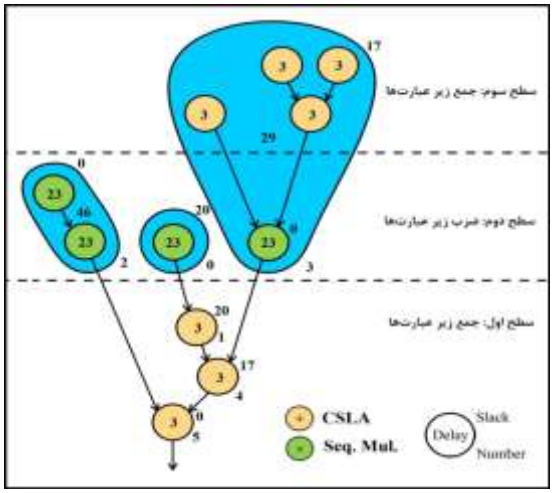
همان طور که مشاهده می‌شود پس از این مرحله، ساختار جمع‌کننده‌های سطح اول از گراف روند داده تغییر کرده و تأخیر مسیر بحرانی (که دارای slack صفر است) از ۵۲ کلاک به ۴۹ کلاک کاهش یافته است. پس از بهینه‌سازی تاخیر، نوبت به بهینه‌سازی مساحت می‌رسد. در این مرحله نیز همانند گام بهینه‌سازی مسیر، با تشکیل گراف پیمایش فضای طراحی و محاسبه منابع مصرفی مسیرهای مختلف، می‌توان برخی از گره‌های گراف روند داده را با ماژول‌های کندتر جایگزین نمود. شکل (۱۱) خروجی حاصل از اعمال بهینه‌سازی مساحت بر روی گراف مورد نظر را نشان می‌دهد. همان طور که مشاهده می‌شود، در این شکل تأخیر مسیر بحرانی گراف روند داده ۴۹ کلاک باقی مانده، اما با تغییر نوع گره‌های جمع‌کننده از CSLA به DA منابع مصرفی کاهش یافته است.



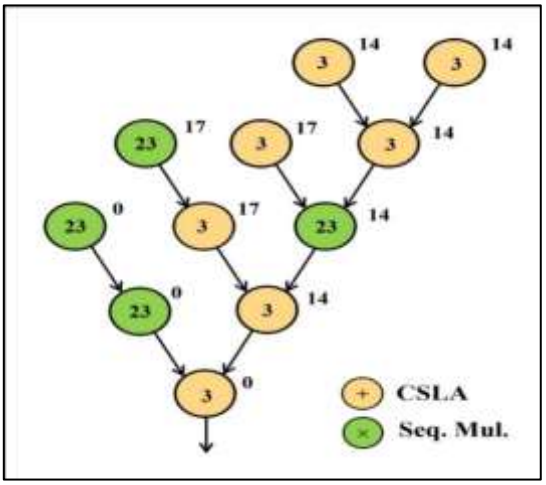

الف) مرحله اول بهینه‌سازی تاخیر




ب) دسته‌بندی گره‌های ماقبل سطح اول در گراف روند داده

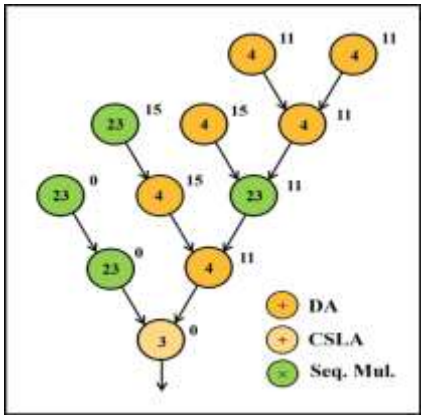



ج) گراف روند داده بهینه‌شده با هدف تأخیر در مرحله آخر




د) خروجی نهایی بهینه‌سازی تاخیر

شکل ۱۰- مراحل انجام بهینه‌سازی تاخیر




شکل ۱۱- گراف نهایی پس از بهینه‌سازی مساحت

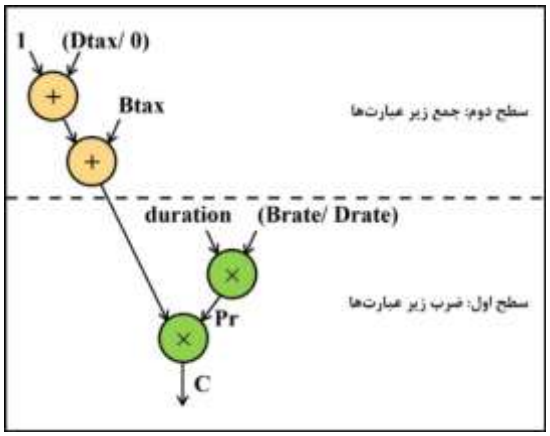
## ۵- ارزیابی الگوریتم پیشنهادی

برای ارزیابی روش‌های ارائه شده، از برنامه محک TELCO که معیاری در عملکرد مدارهای حسابی دهدهی می‌باشد، استفاده می‌شود. شبه‌کد و گراف روند داده ساده شده این برنامه به ترتیب در شکل‌های (۱۲) و (۱۳) نشان داده شده است. طبق الگوریتم بهینه‌سازی، ابتدا تمام گره‌ها به کندترین ماژول‌ها تنظیم شده و تعداد منابع مصرفی محاسبه می‌شوند. در گراف روند داده برنامه محک TELCO، دو عملیات جمع و دو ضرب موجود است که اگر از کندترین ماژولها استفاده شود، با توجه به جدول ۱، دو بلوک DecA برای دو جمع‌کننده از نوع CPA و شش بلوک DecA، چهار بلوک EMG، چهار بلوک MSG و دو بلوک Sel برای دو ضرب‌کننده موجود مورد نیاز می‌باشد. این تعداد بلوک بسیار کمتر از منابع موجود بر روی معماری بوده و بنابراین شرط اول برای بهینه‌سازی برقرار است.

```
if (calltype = L)
        P = duration * Brate;
else
        P = duration * Drate;
Pr = RoundtoNearestEven(P);
C = Trunc(Pr * (1 + Btax));
if (calltype = D)
        C = Trunc(Pr * (1 + Dtax + Btax));
```

شکل ۱۲- کد ساده‌شده برنامه محک TELCO

در مرحله بعد، تمام گره‌های گراف روند داده به سریع‌ترین مدارهای کتابخانه سنتز تغییر یافته و تأخیر مسیر بحرانی با استفاده از الگوریتم ASAP محاسبه می‌گردد که این مقدار برابر ۴۶ سیکل به دست آمده است. از آنجا که تعداد بلوک‌های مصرفی برای سریع‌ترین ماژول‌ها همچنان بسیار کمتر از منابع موجود در معماری است، نیازی به اجرای الگوریتم بهینه‌سازی نگاشت برای این گراف روند داده نخواهد بود. همچنین، الگوریتم بهینه‌سازی تأخیر نیز ساختار گراف روند داده را تغییر نداده و تنها بهینه‌سازی مساحت موجب تغییر نوع جمع‌کننده‌ها از CSLA به DA خواهد شد. با این تغییر پس از بهینه‌سازی، چهار بلوک DecA و دو بلوک Sel از منابع مصرفی کاسته شده است.




شکل ۱۳- گراف روند داده معادل برنامه محک TELCO

جدول ۲، تاخیر و مساحت هر پنج بلوک اصلی معماری DARA را که با استفاده از کتابخانه NanGateFreePDK45 سنتز شده‌اند، نشان می‌دهد. بر اساس مقادیر جدول ۲ و در نظر گرفتن بلوک‌های دخیل در گراف روند داده برنامه محک، مساحت کل طراحی این مدار بر روی معماری DARA قبل و بعد از بهینه‌سازی گراف روند داده، به ترتیب ۷۳۱۴۴ و ۵۱۳۶۶ میکرومتر مربع می‌باشد و در نتیجه، بهینه‌سازی مساحت در این مدار منجر به بهبود حدود ۳۰ درصدی در مساحت کل گردیده است.

جدول ۲- ویژگی‌های مولفه‌های معماری DARA

| | DecA | PPT | EMG | MSG | Sel |
|---|---|---|---|---|---|
| تأخیر (ns) | ۰/۴۶ | ۰/۳۷ | ۰/۲۷ | ۰/۳۶ | ۰/۳۹ |
| مساحت ($\mu m^2$) | ۳۷۲۳ | ۵۳۲ | ۱۲۸۴ | ۱۴۱۶ | ۳۴۴۳ |

## ۶- نتیجه گیری

در این مقاله روش‌هایی برای بهینه‌سازی سنتز خودکار مدارهای حسابی بر روی یک معماری قابل بازپیکربندی درشت‌دانه پیشنهاد گردید. الگوریتم ارائه شده شامل سه گام اصلی بهینه‌سازی نگاشت، بهینه‌سازی تاخیر با استفاده از جابجایی گره‌ها و زیرگراف‌ها و بهینه‌سازی مساحت بود.

مثال‌هایی از پیاده‌سازی الگوریتم پیشنهادی برای عبارات حسابی در بستر معماری قابل بازپیکربندی درشت‌دانه DARA بیان شد. نتایج به دست آمده نشان می‌دهد که گام بهینه‌سازی مساحت به تنهایی توانسته است بهبود حدود ۳۰ درصدی در پیاده‌سازی مدار محک بررسی شده ایجاد نماید.

## مراجع

[1] M. Sedighi, F. Haddadi, S. Emami, and M. Saffarpour, “A Heuristic Algorithm for High Level Synthesis of Decial Arithmetic Circuits Using SystemC,” 10th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS), 2015, pp. 1-6.

[2] D. D. Gajski, and L. Ramachandran, “Introduction to High-Level Synthesis,” IEEE Design & Test of Computers, vol. 11, no. 4, 1994, pp. 44-54.

[3] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, “An Introduction to High-Level Synthesis,” IEEE Design & Test of Computers, vol. 26, no. 4, 2009, pp. 8-17.

[4] R. Nane, V. M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, “A Survey and Evaluation of FPGA High-Level Synthesis Tools,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35 , no. 10, 2016, pp. 1591-1604.

[5] L. K. Wang, M. A. Erle, C. Tsen, E. M. Schwarz, and M. J. Schulte, “A Survey of Hardware Designs for Decimal Arithmetic,” IBM Journal of Research and Development, vol. 54, no. 2, 2010, pp. 8:1-8:15.

[6] A. Nannarelli, “FPGA Based Acceleration of Decimal Operations,” International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2011, pp. 146-151.

[7] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, “High-Level Synthesis for FPGAs: From Prototyping to Deployment,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 4, 2011, pp. 473-491.

[8] M. A. Shami, “Dynamically Reconfigurable Resource Array,” Ph.D. Dissertation, KTH Sch. Inf. Tech. Sweden, Kista, 2012.

[9] Y. Kim, R. N. Mahapatra, and K. Choi, “Design Space Exploration for Efficient Resource Utilization in Coarse-Grained Reconfigurable Architecture,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 18, no. 10, 2010, pp. 1471-1482.

[10] S. Emami, and M. Sedighi, “An Optimized Reconfigurable Architecture for Hardware Implementation of Decimal Arithmetic,” Computers & Electrical Engineering, vol. 63, 2017, pp. 18-29.

[11] M. Vladutiu, “Functional Analysis and Synthesis of Binary and Decimal Adding and Subtracting Devices”, Computer Arithmetic Algorithms and Hardware Implementations, Springer Berlin Heidelberg, 2012.

[12] I. D. Castellanos, “Analysis and Implementation of Decimal Arithmetic Hardware in Nanometer CMOS technology”, Ph.D. Dissertation, Oklahoma State University, USA, 2008.

[13] J. P. Deschamps, G. J. A. Bioul, and G. D. Sutter, “Synthesis of Arithmetic Circuits- FPGA, ASIC and Embedded Systems”, Wiley-Interscience, 2006.

[14] M. A. Gladshtein, “Algorithmic synthesis of a combinational adder of decimal digits encoded by the Johnson-Mobius code”, Automatic Control and Computer Sciences, vol. 43, no. 5, 2009, pp. 233–240.

[15] R. Zimmermann, “Datapath Synthesis for Standard-Cell Design”, Proceedings of the 19th IEEE Symposium on Computer Arithmetic, 2009, pp. 207-211.

[16] C. K. Cheng, “Design Space Exploration for Power-Efficient Mixed-Radix Ling Adders”, Proceedings of the 19th IEEE Symposium on Computer Arithmetic, 2009, pp. 212-212.

[17] A. K. Verma, P. Brisk, and P. Ienne, “Challenges in automatic optimization of arithmetic circuits”, Proceedings of the 16th IEEE Symposium on Computer Arithmetic, 2009, pp. 213-218.

[18] X. Liu, C. Yang, and Z. Guan, “Efficient arithmetic expression optimization with weighted adjoint matrix,” 2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC), 2020, pp. 1-8.

[19] K. E. Murray, J. Luu, M. J. P. Walker, C. McCullough, S. Wang, S. Huda, B. Yan, C. Chiasson, K. B. Kent, J. Anderson, J. Rose, and V. Betz, “Optimizing FPGA Logic Block Architectures for Arithmetic,” in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 6, 2020, pp. 1378-1391.

[۲۰] بهرام عزیزالله گنجی و صدیقه بابایی صداقت، "بهینه‌سازی ساختار میکروفن خازنی جدید با دیافراگم قورباغه‌ای جهت افزایش حساسیت و کاهش ولتاژ تغذیه"، نشریه مدل‌سازی در مهندسی، دوره ۱۷، شماره ۵۹، زمستان ۱۳۹۸، صفحه ۱۵۱- ۱۴۱.

[۲۱] امین رضایی‌پناه، علی مبارکی و سعید بحرانی خادمی، "بهینه‌سازی شبکه عصبی MLP با استفاده از الگوریتم ژنتیک موازی FinGrain برای تشخیص سرطان سینه"، نشریه مدل‌سازی در مهندسی، دوره ۱۷، شماره ۵۷، تابستان ۱۳۹۸، صفحه ۱۸۶- ۱۷۳.

[۲۲] مهدی یعقوبی و مرتضی زاهدی، "بهینه‌سازی در همروندی فرآیندهای کسب‌وکار با هدف تعادل بار کاری"، نشریه مدل‌سازی در مهندسی، دوره ۱۷، شماره ۵۷، تابستان ۱۳۹۸، صفحه ۱۷۲- ۱۵۹.