مقاله پژوهشی

سنتز سطح بالای مدارهای حسابی دهدهی بر روی معماری‌های قابل بازپیکربندی درشت‌دانه

سمانه امامی

چکیده: افزایش قابلیت‌های مدارهای مجتمع و پیچیدگی برنامه‌های کاربردی، روش‌ها و ابزارهای طراحی سخت‌افزار را به سمت سطوح بالاتری از انتزاع سوق داده و سنتز سطح بالا، یکی از کلیدی‌ترین گام‌ها در افزایش سطح انتزاع می‌باشد. در سال‌های اخیر، تحقیقات گسترده‌ای برای طراحی ساختارهای قابل بازپیکربندی با هدف حساب دهدهی صورت گرفته است. از آنجا که از یک سو، استفاده مؤثر 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 سنتز خودکار توصیف سطح بالا از مدارهای حسابی دهدهی بر روی یک معماری قابل بازپیکربندی درشت‌دانه ارائه خواهد شد. بستر سخت‌افزاری انتخاب‌شده، معماری قابل بازپیکربندی درشت‌دانه DARA بوده و روش‌های پیشنهادشده برای اختصاص منابع در جریان سنتز، شامل دو الگوریتم مکاشفه‌ای و ILP می‌باشند. نتایج به دست آمده نشان می‌دهند که مطابق انتظار، برای ابعاد محدود معماری مورد استفاده، الگوریتم ILP به میزان قابل توجهی (حدود ۳۰٪) بهتر از الگوریتم مکاشفه‌ای عمل می‌نماید.

کلیدواژه: **سنتز سطح بالا، حساب دهدهی، معماری‌های قابل بازپیکربندی درشت‌دانه، نگاشت روی سخت‌افزار، اختصاص منابع.**

۱- مقدمه

از آنجا که طراحی سامانه‌های دیجیتال بسیار پیچیده شده‌اند، توصیف سطح انتقال ثبات (RTL[1]) دیگر برای حل چالش‌های پیش روی طراحان کافی 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 (HLS[2]) 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 می‌نماید [۳]. با توجه به این که سنتز سطح بالا، طراحی سیستم‌های پیچیده امروزی را تسهیل کرده و زمان رسیدن محصول به بازار را کاهش داده است، به سرعت مورد توجه و گسترش قرار گرفت [۴].

با توجه به مباحث مطرح‌شده، سنتز سطح بالا می‌تواند در همه کاربردها مفید واقع شود، اما هرچه توصیف اولیه چکیده‌تر باشد، کارایی 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 ویژه‌ای برخوردار است. زیرا از یک سو، 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، گرایش به پیاده‌سازی سخت‌افزاری حساب دهدهی در سال‌های اخیر افزایش یافته است [۶]. در پیاده‌سازی‌های سخت‌افزاری، طراحی به روش ASIC[3] و استفاده از FPGA[4]ها، دو سر طیف انواع پیاده‌سازی و نقطه مقابل یکدیگر از نظر مساحت، توان، تأخیر، انعطاف‌پذیری، هزینه ساخت و زمان آماده‌شدن محصول هستند [۷] تا [۹]. 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 ASIC و کارایی پایین محاسباتی و سربار بالای مساحت در FPGAها، طراحان معماری را وادار به جستجوی جایگزینی برای این معماری‌ها کرده است. معماری‌های قابل بازپیکربندی درشت‌دانه (CGRA[5]) دارای کارایی محاسباتی بالا، هزینه و صرف زمان پایینی برای طراحی هستند و به همین دلیل، جایگزینی جذاب و مناسب برای طراحی در حوزه کاربردهای خاص‌منظوره هستند. در واقع معماری‌های قابل بازپیکربندی درشت‌دانه، مصالحه‌ای[6] بین ASIC و FPGAها به شمار می‌روند، زیرا نسبت به FPGAها، بازده محاسباتی بهتر و در مقایسه با ASIC، بازده مهندسی بهتری دارند [۸].

این مقاله به ارائه یک روش سنتز خودکار توصیف سطح بالا از مدارهای حسابی دهدهی روی یک معماری قابل بازپیکربندی درشت‌دانه

این مقاله در تاریخ ۲۱ آذر ماه ۱۴۰۰ دریافت و در تاریخ ۲۸ اردیبهشت ماه ۱۴۰۱ بازنگری شد.

سمانه امامی، دانشکده مهندسی برق و کامپیوتر، دانشگاه سمنان، سمنان، ایران، (email: s_emami@semnan.ac.ir).

1. Register Transfer Level
2. High Level Synthesis
3. Application Specific Integrated Circuit
4. Field Programmable Gate Array
5. Coarse Grain Reconfigurable Architecture
6. Trade-off

از این کاربرد می‌پردازد. معماری انتخاب‌شده برای این منظور، DARA[1] نام دارد [۱۰] که در بخش دوم معرفی گردیده و روش‌های پیشنهادشده برای اختصاص منابع در جریان سنتز، شامل دو الگوریتم مکاشفه‌ای و برنامه‌نویسی خطی صحیح (ILP[2]) می‌باشند.

ساختار مقاله به این شرح است که در بخش دوم به مرور کارهای انجام‌شده در زمینه سنتز مدارهای حسابی پرداخته می‌شود. در بخش سوم، روش پیشنهادی برای سنتز سطح بالای مدارهای حسابی دهدهی ارائه می‌گردد. مقایسه نتایج حاصل از روش‌های پیشنهادی در بخش چهارم بیان می‌شود و در بخش پنجم، نتیجه‌گیری و پیشنهادهایی برای کارهای آینده ارائه خواهد گردید.

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[3]در نظر گرفته شده است. در [۱۲]، سنتز فرایندی 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. در [۱۴] منظور از سنتز الگوریتمی، ارائه 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[4]از نظر زمان، مساحت و توان پرداخته شده است. در این منبع به بیان کلیات روند سنتز به کار گرفته شده در این ابزارها، شامل استخراج عملیات حسابی از توصیف انجام‌شده و خوشه‌بندی آنها در دسته‌های بزرگ‌تر، بهینه‌سازی مسیر داده با الگوریتم‌های مختلف حسابی، تولید netlist با استفاده از روش‌های مختلف پیاده‌سازی عملگرها و الگوریتم‌هایی برای کاهش توان مصرفی ناشی از بزرگ‌بودن اندازه مدارها و فعالیت بالای سوئیچ‌ها پرداخته شده است.

بعضی منابع، در تعریف سنتز فقط بر روی بهینه‌سازی مدار متمرکز شده‌اند. به عنوان نمونه در [۱۶] با استفاده از ILP، روشی بهینه برای پیاده‌سازی جمع‌کننده Ling ارائه گردیده است. نویسنده این مقاله ادعا کرده که یک فرمول‌بندی ILP بر اساس مدل‌های تلاش منطقی[5]ارائه نموده که با کوچک‌سازی فضای جستجو، زمان اجرای الگوریتم را کاهش می‌دهد. منبع دیگری که به جنبه بهینه‌سازی در سنتز پرداخته است، [۱۷] بوده که ۳ روش مختلف برای بهینه‌سازی مدارهای حسابی ارائه می‌نماید.

مقاله [۱]، سنتز سطح بالا را بر ۳ بخش اصلی استوار می‌داند: ۱) کتابخانه گسترش‌یافته از مؤلفه‌های سطح بالا که ویژگی‌های دقیق آنها مانند تأخیر، مساحت و توان، برای الگوریتم سنتزی که می‌خواهد از آنها استفاده کند، مشخص شده باشد. ۲) یک زبان سطح بالا که از توصیف و مدل‌سازی مداری که باید سنتز شود، پشتیبانی نماید. ۳) یک یا چند الگوریتم که فضای طراحی را به منظور یافتن راه حل بهینه یا نزدیک به بهینه جستجو کند.

معماری DARA که در این مقاله از آن به عنوان بستر سخت‌افزاری الگوریتم‌های پیشنهادی استفاده می‌شود، در شکل ۱ آمده است. این معماری از تکرار منظم ۵ نوع بلوک مختلف، تشکیل و برای پیاده‌سازی عملیات پایه حساب دهدهی بهینه شده است. با توجه به درشت‌دانه‌بودن معماری DARA و دنبال‌کردن رویکرد [۱]، از خود بلوک‌های سازنده آن به عنوان مؤلفه‌های موجود در کتابخانه استفاده گردیده و زبان توصیف مدار دهدهی ورودی نیز SystemC در نظر گرفته شده است. در این مقاله بر روی عنصر سوم جریان سنتز، یعنی الگوریتم نگاشت بر روی سخت‌افزار تمرکز شده و یک روش مکاشفه‌ای[6]یک الگوریتم ILP برای سنتز یک مدار دهدهی ورودی بر روی یک سخت‌افزار قابل بازپیکربندی درشت‌دانه (DARA) پیشنهاد می‌گردد.

۳- روش‌های سنتز پیشنهادی

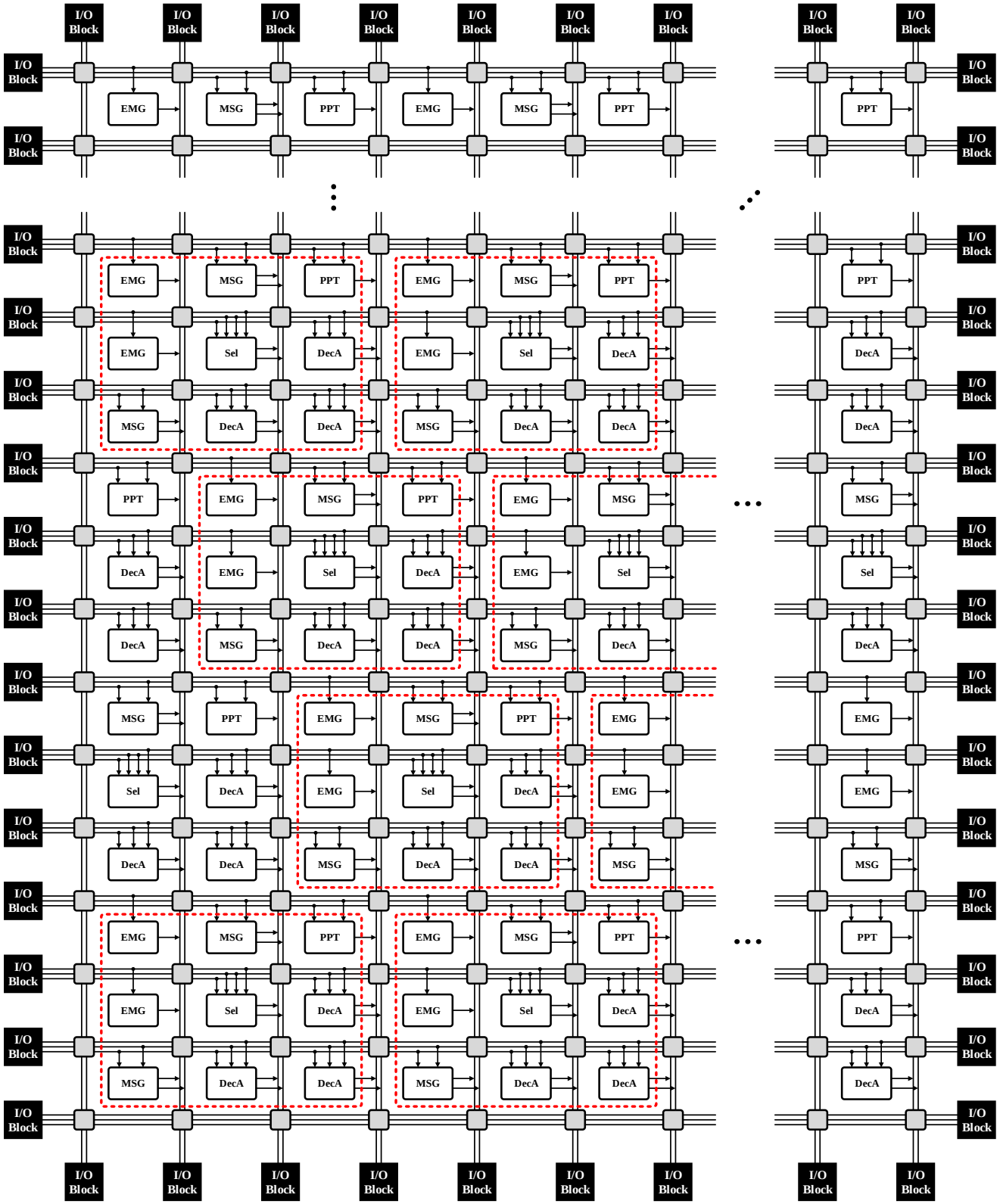
از آنجا که الگوریتم‌های پیشنهادی بر روی یک DFG[7] ورودی عمل می‌کنند، گام اول در جریان سنتز، تبدیل توصیف اولیه مدار حسابی به DFG می‌باشد. برای تولید این DFG که در ادامه آن را DFG سطح بالا (HLDFG[8]) می‌نامیم، از همان الگوریتم ارائه‌شده در [۱] استفاده می‌شود. بعد از تولید DFG سطح بالا، جستجو برای یافتن بهترین نگاشت گره‌های DFG بر روی معماری آغاز خواهد گردید.

گام دوم و مشترک در هر دو الگوریتم پیشنهادی، بررسی وجود راه حل برای نگاشت یا همان امکان‌سنجی مسئله می‌باشد. اگر هدف، رسیدن به یک تأخیر یا فرکانس مشخص در مدار باشد، باید ابتدا تمام گره‌ها به سریع‌ترین پیاده‌سازی‌های جمع، تفریق و ضرب نگاشت شوند. در این حالت اگر مسیر بحرانی یافت‌شده، بزرگ‌تر از محدودیت تأخیر داده‌شده باشد، راه حلی برای مسئله وجود ندارد. در غیر این صورت، تمام گره‌ها به کندترین یا کوچک‌ترین پیاده‌سازی‌ها (یعنی با کمترین تعداد بلاک) نگاشت می‌شوند. در این صورت نیز اگر منابع موجود در معماری پیشنهادی (شامل بلاک‌ها، سوئیچ‌ها، مسیرهای ارتباطی و I/O) برای پیاده‌سازی مدار کافی نباشد، راه حلی برای مسئله وجود ندارد. اگر هیچ یک از شرایط بالا برقرار نباشد، یعنی باید الگوریتمی برای نگاشت بهینه یا نزدیک به بهینه مدار بر روی معماری اجرا شود.

گام سوم، تبدیل DFG سطح بالای به دست آمده از توصیف ورودی به DFG سطح پایین (LLDFG[9]) می‌باشد. به این معنا که برای هر گره که نشان‌دهنده یک عملیات حسابی دهدهی است، یکی از پیاده‌سازی‌های ممکن آن با استفاده از مجموعه بلاک‌های موجود در معماری، جایگزین گره مربوط در DFG سطح بالا گردد. با توجه به این که برای هر عملیات حسابی دهدهی، پیاده‌سازی‌های مختلفی از نظر تأخیر مدار و بلاک‌های استفاده‌شده وجود دارد، بنابراین برای هر DFG سطح بالا، چندین DFG سطح پایین وجود خواهد داشت و ایجاد DFG سطح پایین مناسب، حائز اهمیت است.

برای ایجاد DFG سطح پایین مناسب، ابتدا باید با توجه به حداقل و حداکثر کلاک‌های مورد نیاز برای پیاده‌سازی‌های مختلف هر عملیات حسابی دهدهی، مسیر بحرانی را در DFG سطح بالا یافته و سپس

1. Decimal Arithmetic Reconfigurable Architecture
2. Integer Linear Programming
3. Truth Table
4. Cell-Based Design
5. Logical Effort
6. Heuristic
7. Data Flow Graph
8. High Level DFG
9. Low Level DFG



شکل ۱: معماری DARA [۱۰].

عملیات روی مسیر بحرانی را با سریع‌ترین پیاده‌سازی ممکن و سایر عملیات‌ها را با پیاده‌سازی‌های کندتر و کوچک‌تر (تعداد بلاک کمتر) جایگزین نمود. شبه‌کد مربوط به الگوریتم تبدیل DFG سطح بالا به DFG سطح پایین در شکل ۲ نشان داده شده است.

بدیهی است که اگر با DFG سطح پایین به دست آمده، پس از اجرای الگوریتم نگاشت، نتیجه مطلوب حاصل نشد، می‌توان DFG سطح پایین را با تغییر پیاده‌سازی‌های موجود، اصلاح نمود.

در گام آخر برای نگاشت DFG سطح پایین بر روی معماری DARA، دو روش پیشنهاد می‌گردد.

۱-۳ الگوریتم نگاشت مکاشفه‌ای

اساس کار این الگوریتم بر مبنای یکی از انواع الگوریتم‌های یافتن کوتاه‌ترین مسیر در گراف‌ها مانند الگوریتم Dijkstra می‌باشد. در این روش، بلاک‌های موجود در معماری (شامل بلاک‌های I/O و عملیات) به

```
Algorithm (HLDFG, delay_const, available_resources)
{
Set all nodes to the fastest/biggest implementations of adder,
subtractor and multiplier;
if (delay_const < delay)
   There is no solution for this problem!
else
   Set all nodes to the smallest/slowest implementations of adder,
   subtractor and multiplier;
   if (available_resources < needed resources)
      There is no solution for this problem!
   else
      findCritical (path_1, path_2,..., path_n);
      replaceCritical (fastest implementations);
      do
         replaceNonCritical (smaller implementations);
      until their delay does not exceed the critical path delay;
}
```

شکل ۲: شبه‌کد الگوریتم تبدیل HLDFG به LLDFG.

عنوان گره‌های گراف و کانال‌های ارتباطی بین بلاک‌ها به عنوان یال‌های گراف در نظر گرفته شده و وزن هر یال، معادل تعداد سوئیچ‌های بین ۲ بلاک می‌باشد. با توجه به این که در معماری DARA، جریان داده‌ها در مسیرهای افقی فقط از سمت چپ به راست و در مسیرهای عمودی فقط از پایین به بالا بوده و ورودی بلاک‌های سازنده از مسیرهای افقی بالای آنها تأمین شده و خروجی به مسیرهای عمودی سمت راست آنها داده می‌شود، وزن یال‌ها (یا همان فاصله هر بلاک از بلاک‌های اطراف خود بر حسب تعداد سوئیچ‌های بین آنها) مطابق شکل ۳ به دست می‌آید.

از آنجا که در تقاطع هر سطر و ستون در معماری DARA یک سوئیچ وجود دارد و فاصله بین بلاک‌ها با تعداد سوئیچ‌ها محاسبه می‌شود و ورودی بلاک‌ها از سمت بالای بلاک تأمین گردیده و خروجی آنها به سمت راست داده می‌شود (با توجه به شکل ۳)، فاصله بلاک‌های a و b از (۱) به دست می‌آید که در آن x_i و y_i به ترتیب شماره ستون و سطر مربوط به بلاک i هستند

$$dist(a,b)=\begin{cases}|x_b-x_a|+|y_b-y_a|+1 \\ \quad \text{if } y_b \geq y_a \text{ and } x_b \leq x_a \\ |x_b-x_a|+|y_b-y_a|-1 \\ \quad \text{if } y_b < y_a \text{ and } x_b > x_a \\ |x_b-x_a|+|y_b-y_a| \quad \text{else}\end{cases} \qquad (1)$$

پس به ترتیب با شروع از بلاک‌های موجود در مسیر بحرانی LLDFG، الگوریتم یافتن کوتاه‌ترین مسیر بین هر دو بلاک اجرا شده و بلاک‌های انتخاب‌شده از لیست منابع موجود در معماری حذف می‌گردند. با توجه به این که بعضی از بلاک‌ها در LLDFG به بیش از یک بلاک دیگر وابستگی داده‌ای دارند، در این شرایط، الگوریتم تلاش می‌کند که بلاکی را بیابد که مجموع فواصل آن از بلاک‌های وابسته حداقل باشد. این کار تا نگاشت کامل LLDFG بر روی معماری DARA ادامه دارد.

۲-۳ الگوریتم نگاشت ILP

در این روش، جستجویی خطی و جامع[1] برای یافتن جواب بهینه انجام می‌شود و به همین دلیل در صورت بزرگ‌بودن فضای مسئله، زمان‌بر و پیچیده خواهد بود. برای ارائه الگوریتم پیشنهادی، نگاشت‌های مختلف هر عملیات، ابتدا بر روی معماری DARA (در قالب درخت‌های تشکیل‌شده

1. Exhaustive

...

	۵	۴	۳	۳	۴
	۴	۳	۲	۲	۳
...	۳	۲	–	۱	۲	...
	۳	۲	۱	۱	۲
	۴	۳	۲	۲	۳

...

شکل ۳: فاصله بلاک‌ها در معماری DARA بر حسب تعداد سوئیچ بین آنها.

از بلاک‌های مورد نیاز) در نظر گرفته شده و به هر نگاشت، وزنی معادل تعداد سوئیچ‌های مسیر بحرانی آن نسبت داده می‌شود. بدیهی است که هر یک از انواع درخت‌ها می‌توانند در چندین نقطه مختلف از معماری نگاشت شوند. جدول ۱، نمونه‌هایی از پیاده‌سازی‌های مختلف عملیات حسابی دهدهی را روی معماری DARA نشان می‌دهد. بلاک‌های استفاده‌شده در پیاده‌سازی هر درخت، همان بلاک‌های پیشنهادشده در [۱۰] است که با چیدمان‌های مختلف کنار هم قرار گرفته‌اند.

تابع هدف در این الگوریتم، کم‌کردن پریود کلاک و در نتیجه تأخیر مدار می‌باشد. اگر فرض کنیم پریود کلاک برابر d_b+nd_s باشد که در آن، d_b و d_s به ترتیب تأخیر مربوط به بلاک و سوئیچ بوده و n برابر تعداد سوئیچ‌های مسیر بحرانی است، دو پارامتر اول مقداری ثابت هستند. در نتیجه کاهش n تأثیر مستقیمی در کم‌کردن پریود کلاک دارد و این در حالی است که مقدار n هم به وزن درخت‌های انتخاب‌شده و هم به فاصله آنها از یکدیگر وابسته است.

عناصر و مجموعه‌های مورد نیاز برای پیاده‌سازی الگوریتم پیشنهادی عبارتند از:

$nodes$: مجموعه گره‌های گراف ورودی با در نظر گرفتن بلاک‌های I/O

DFG: مجموعه وابستگی داده‌ای بین عملیات (یال‌های گراف)

ops: مجموعه عملیات موجود در گراف ورودی مانند جمع، تفریق و ضرب

$treeN$: مجموعه تمام درخت‌های ممکن برای نگاشت عملیات در معماری DARA

$opTrees[ops]$: مجموعه درخت‌های ممکن برای پیاده‌سازی عملیات مورد نظر ops

$tree۲$: مجموعه تمام زوج‌مرتب‌های ممکن از درخت‌ها

$type[nodes]$: پارامتر مشخص‌کننده نوع عملیات گره $nodes$ از DFG ورودی

$x[treeN]$: متغیر دودویی نشان‌دهنده انتخاب‌شدن یا نشدن درخت $treeN$

$y[tree۲]$: متغیر دودویی برای دخالت‌دادن یا ندادن نزدیکی درخت‌های وابسته مشخص‌شده در $tree۲$ در میزان هزینه

$N[ops]$: پارامتر نشان‌دهنده تعداد درخت مورد نیاز از عملیات نوع ops

$C[treeN]$: پارامتر نشان‌دهنده وزن درخت $treeN$

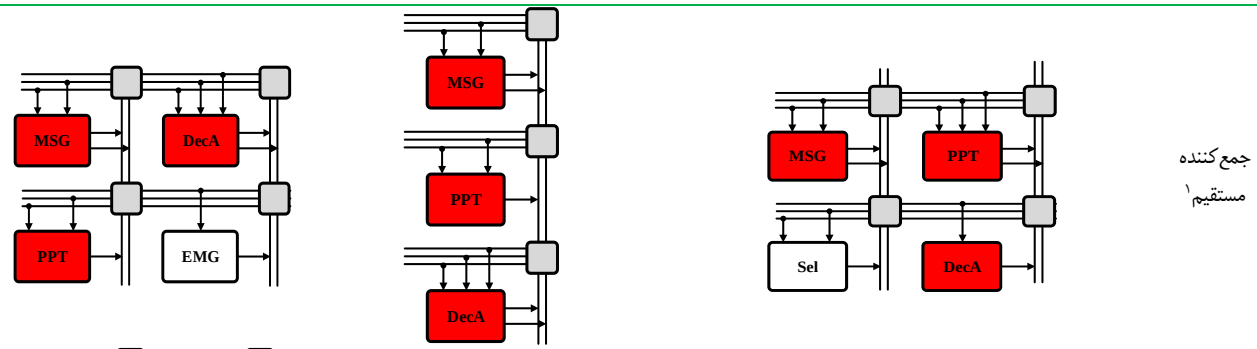
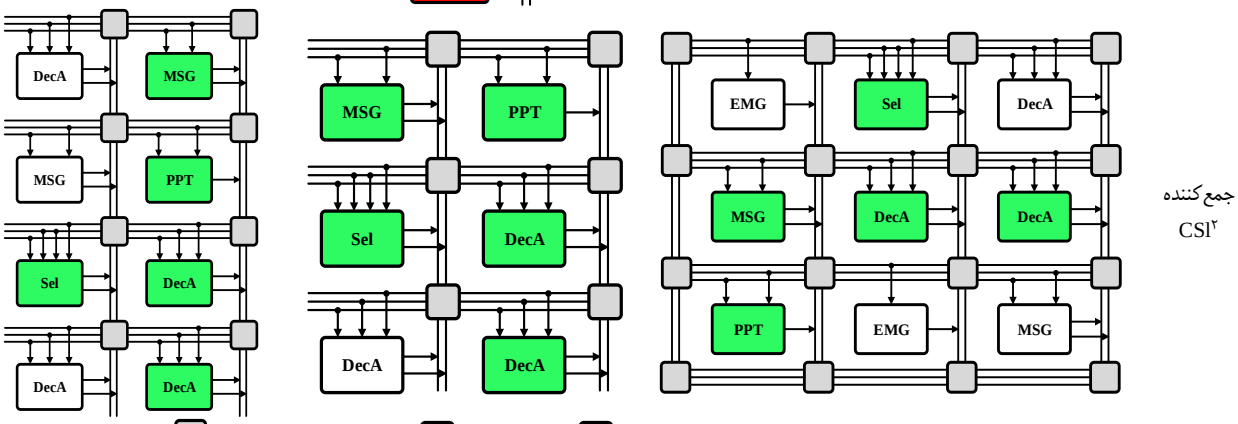
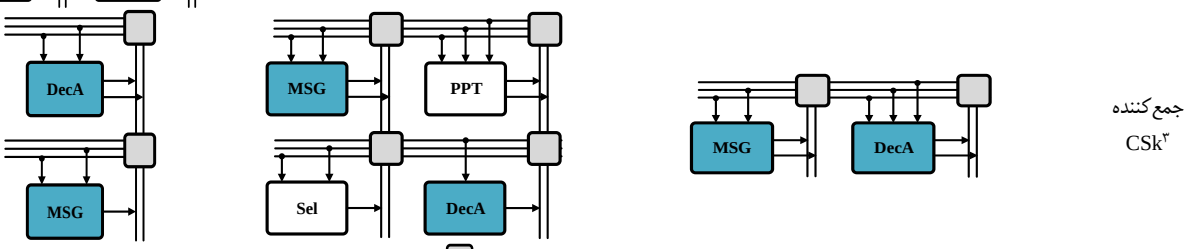
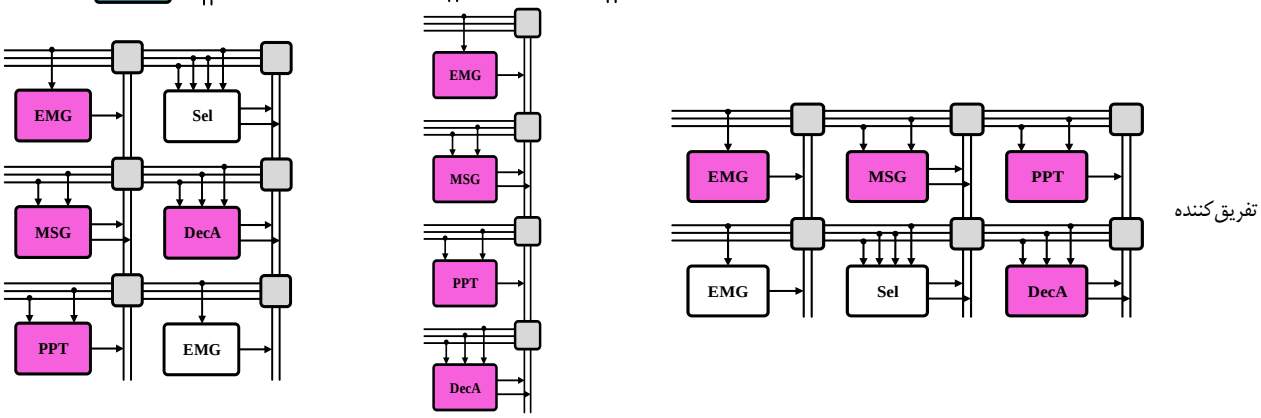
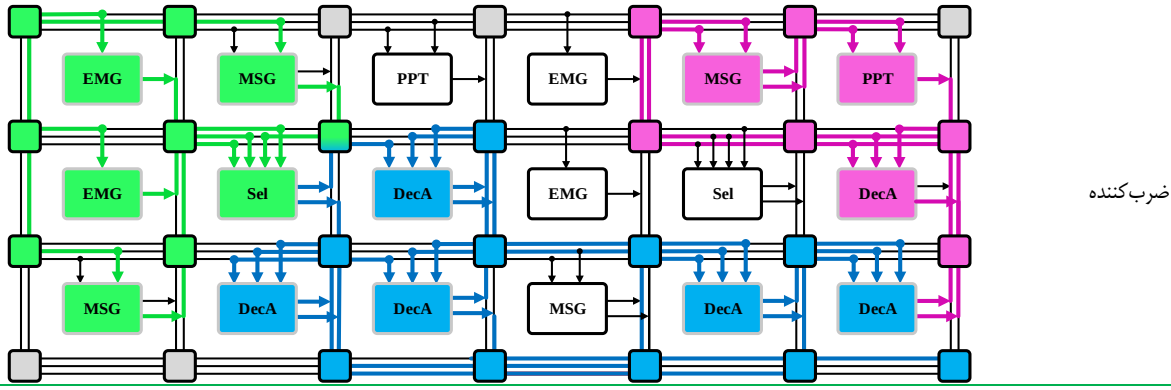
$BN[treeN]$: پارامتر نشان‌دهنده بلاک آغازین درخت $treeN$

$EN[treeN]$: پارامتر نشان‌دهنده بلاک پایانی درخت $treeN$

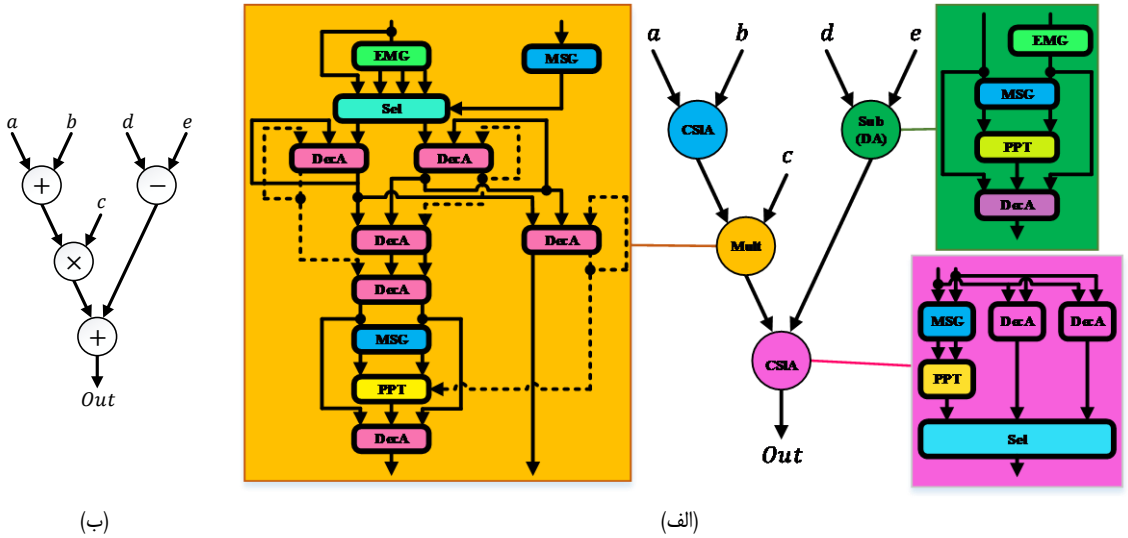
$treesDep[treeN]$: مجموعه نشان‌دهنده درخت‌های دارای بلاک مشترک از مجموعه $treeN$ برای بررسی عدم استفاده هم‌زمان از آنها

$dist(a,b)$: تابع محاسبه‌کننده فاصله درخت‌های a و b

جدول ۱: مثال‌هایی از پیاده‌سازی‌های مختلف عملیات حسابی دهدهی بر روی معماری DARA.

عملیات	درخت‌های مختلف پیاده‌سازی
جمع‌کننده مستقیم[1]	
جمع‌کننده CSl[2]	
جمع‌کننده CSk[3]	
تفریق‌کننده	
ضرب‌کننده	

1. Direct Adder
2. Carry Select
3. Carry Skip



(ب) (الف)

شکل ۴: (الف) LLDFG و (ب) HLDFG مربوط به (۲).

با توجه به این که هدف الگوریتم، کاهش تأخیر مدار با استفاده از به حداقل رساندن پریود کلاک بوده و همان طور که بیان شد، پریود کلاک وابستگی مستقیمی با تعداد سوئیچ‌های مسیر بحرانی دارد، بنابراین تابع هدف به صورت زیر تعریف می‌شود

$$\text{Minimize } obj : \max\{C[tr], dist(EN[tr_i], BN[tr_j])\}$$
$$, \forall tr \in treeN$$

یعنی از میان تمام پیاده‌سازی‌های ممکن، درخت‌هایی انتخاب شوند که حداکثر مقدار میان وزن درخت (تعداد سوئیچ‌های مسیر بحرانی در پیاده‌سازی یک عملیات) و فاصله بین خروجی یک درخت تا ورودی درخت بعدی (تعداد سوئیچ‌ها در مسیر وابستگی داده‌ای عملیات مختلف) به حداقل برسد.

همچنین برای انتخاب درخت‌های ممکن برای پیاده‌سازی عملیات، محدودیت‌هایی وجود دارد که عبارت هستند از:

- هیچ درختی بیش از یک بار انتخاب نشود، یعنی

$$X[tr] = ۰ \text{ or } ۱ \ , \ \forall tr \in treeN$$

به عنوان مثال، برای پیاده‌سازی دو عمل جمع موازی، از یک شماره درخت ثابت استفاده نشود.

- تعداد درخت‌های انتخاب‌شده از یک نوع عملیات، برابر با تعداد گره‌های آن عملیات در DFG ورودی باشد، یعنی

$$\sum_{tr \in opTrees[op]} x[tr] = N[op] \ , \ \forall op \in ops$$

مثلاً مجموع تعداد درخت‌های جمع‌کننده انتخاب‌شده با پیاده‌سازی‌های مختلف برابر با تعداد عملیات جمع مدار توصیف‌شده ورودی باشد.

- درخت‌هایی که از بلاک‌های مشترک استفاده می‌کنند، هم‌زمان انتخاب نشوند، یعنی

$$\sum_{tr \in treesDep[t]} x[tr] \le ۱ \ , \ \forall t \in treeN$$

به عنوان مثال، یک بلاک MSG مشترک میان درخت‌های انتخاب‌شده برای پیاده‌سازی جمع و ضرب قرار نگیرد.

٤- پیاده‌سازی و مقایسه نتایج

برای مقایسه الگوریتم‌های پیشنهادی و نتایج حاصل از آنها، ابتدا از مدار حسابی توصیف‌شده در (۲) به عنوان مثال کمک گرفته می‌شود و در مرحله نخست، مدار ورودی به HLDFG و سپس به LLDFG تبدیل می‌گردد. خروجی این مرحله در شکل ۴ قابل مشاهده است

$$out = (a+b)\times c + d - e \qquad (۲)$$

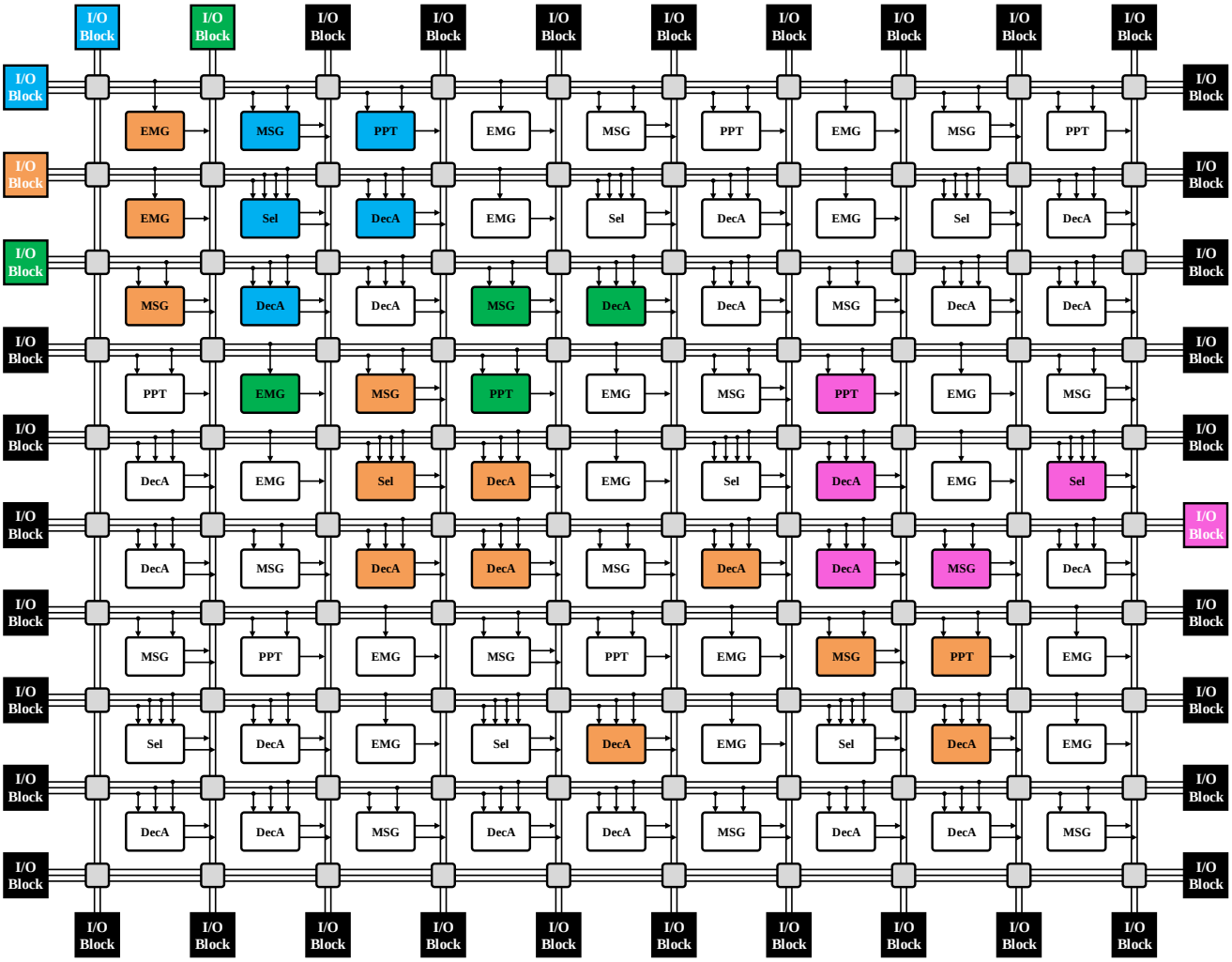
برای پیاده‌سازی الگوریتم مکاشفه‌ای از زبان C استفاده گردیده و برای پیاده‌سازی الگوریتم ILP، روابط بیان‌شده در بخش قبل به زبان zimpl [۱۸] توصیف و برای حل به ابزار SCIP [۱۹] داده شده است. خروجی این ابزار برای مسئله توصیف‌شده، شماره درخت‌های انتخابی از معماری DARA می‌باشد. نگاشت نهایی حاصل از اجرای روش مکاشفه‌ای و ILP به ترتیب در شکل‌های ۵ و ۶ نمایش داده شده است. رنگ بلوک‌ها در این دو شکل، مطابق با رنگ گره متناظر در LLDFG در شکل ۴ و نشان‌دهنده بلوک‌های شرکت‌کننده در اجرای آن عملیات می‌باشد.

مقایسه پیاده‌سازی‌های انجام‌شده نشان می‌دهد که مطابق آنچه انتظار می‌رفت، الگوریتم ILP نگاشت بهتری نسبت به روش مکاشفه‌ای انجام داده است. به طوری که تعداد سوئیچ‌های مسیر بحرانی در نگاشت انجام‌شده توسط الگوریتم مکاشفه‌ای برابر ۷ می‌باشد، در حالی که تعداد این سوئیچ‌ها در نگاشت صورت‌گرفته توسط ILP، ۴ است. بنابراین پریود کلاک در پیاده‌سازی انجام‌شده توسط ILP کمتر، فرکانس کلاک بیشتر و در نتیجه، تأخیر مدار ورودی کمتر خواهد بود.

برای ارزیابی دقیق‌تر الگوریتم‌های ارائه‌شده در هر حوزه، بهتر است که از برنامه‌های محک[1] موجود برای آنها استفاده گردد. اما متأسفانه برای ارزیابی عملکرد مدارهای حسابی دهدهی با ورودی اعداد صحیح، هیچ برنامه محکی در دسترس نیست. تنها محک موجود، برنامه TELCO [۲۰] است که برای مدارهای حسابی دهدهی ممیز شناور مورد استفاده قرار می‌گیرد. شبه‌کد و DFG ساده‌شده این برنامه در شکل ۷ نشان داده شده است. همان طور که از این شکل مشخص می‌شود، این برنامه محک از دو عملیات جمع و دو عملیات ضرب تشکیل شده است. با فرض این که ورودی‌های برنامه، اعداد صحیح دهدهی باشند، نگاشت این مدار بر روی معماری با استفاده از الگوریتم مکاشفه‌ای، شامل ۶ سوئیچ و با الگوریتم ILP شامل ۴ سوئیچ در مسیر بحرانی خواهد بود.

با توجه به این که الگوریتم مکاشفه‌ای بر اساس هوشمندی محدود تعریف‌شده برای آن عمل می‌کند و الگوریتم ILP، بررسی جامعی از تمام

1. Benchmark



شکل ۵: نگاشت LLDFG (۲) بر روی DARA با استفاده از روش مکاشفه‌ای.

حالات ممکن نگاشت مدار بر روی معماری انجام می‌دهد، پیش‌بینی اولیه در مورد عملکرد بهتر الگوریتم ILP در مقابل الگوریتم مکاشفه‌ای بود که نتایج به دست آمده نیز آن را تأیید نموده است.

۵- نتیجه‌گیری و کارهای آینده

در این مقاله، روش‌هایی برای سنتز خودکار توصیف سطح بالای مدارهای حسابی دهدهی روی یک معماری قابل بازپیکربندی درشت‌دانه ارائه گردید. معماری قابل بازپیکربندی درشت‌دانه DARA به عنوان بستر سخت‌افزاری آزمایش انتخاب شده و دو الگوریتم مکاشفه‌ای و ILP برای اختصاص منابع در جریان سنتز ارائه گردید. بررسی‌های انجام‌شده مشخص نمود که برای ابعاد محدود معماری مورد استفاده، نگاشت انجام‌شده توسط الگوریتم ILP نسبت به الگوریتم مکاشفه‌ای حدود ۳۰٪ در پریود کلاک، کاهش نشان می‌دهد. در ادامه این پژوهش برای بهبود نتایج الگوریتم‌ها، می‌توان پارامترهای دخیل در محاسبه فاصله بلاک‌ها را دقیق‌تر کرد. به علاوه می‌توان در الگوریتم ILP درخت‌های متنوع‌تری را برای پیاده‌سازی عملیات مختلف در نظر گرفت.

مراجع

[1] M. Sedighi, F. Haddadi, S. Emami, and M. Saffarpour, "A heuristic algorithm for high level synthesis of decimal arithmetic circuits using systemC," in *Proc. 10th Int. Conf. on Design & Technology of Integrated Systems in Nanoscale Era, DTIS'15*, 6 pp., Napoli, Italy, 21-23 Apr. 2015.

[2] D. D. Gajski and L. Ramachandran, "Introduction to high-level synthesis," *IEEE Design & Test of Computers*, vol. 11, no. 4, pp. 44-54, Winter 1994.

[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, pp. 8-17, Aug. 2009.

[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 Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591-1604, Oct. 2016.

[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 J. of Research and Development*, vol. 54, no. 2, pp. 8:1-8:15, Mar./Apr. 2010.

[6] A. Nannarelli, "FPGA based acceleration of decimal operations," in *Proc. Int. Conf. on Reconfigurable Computing and FPGAs, ReConFig'11, Cancun*, pp. 146-151, Cancun, Mexico, 30-30 Nov. 2011.

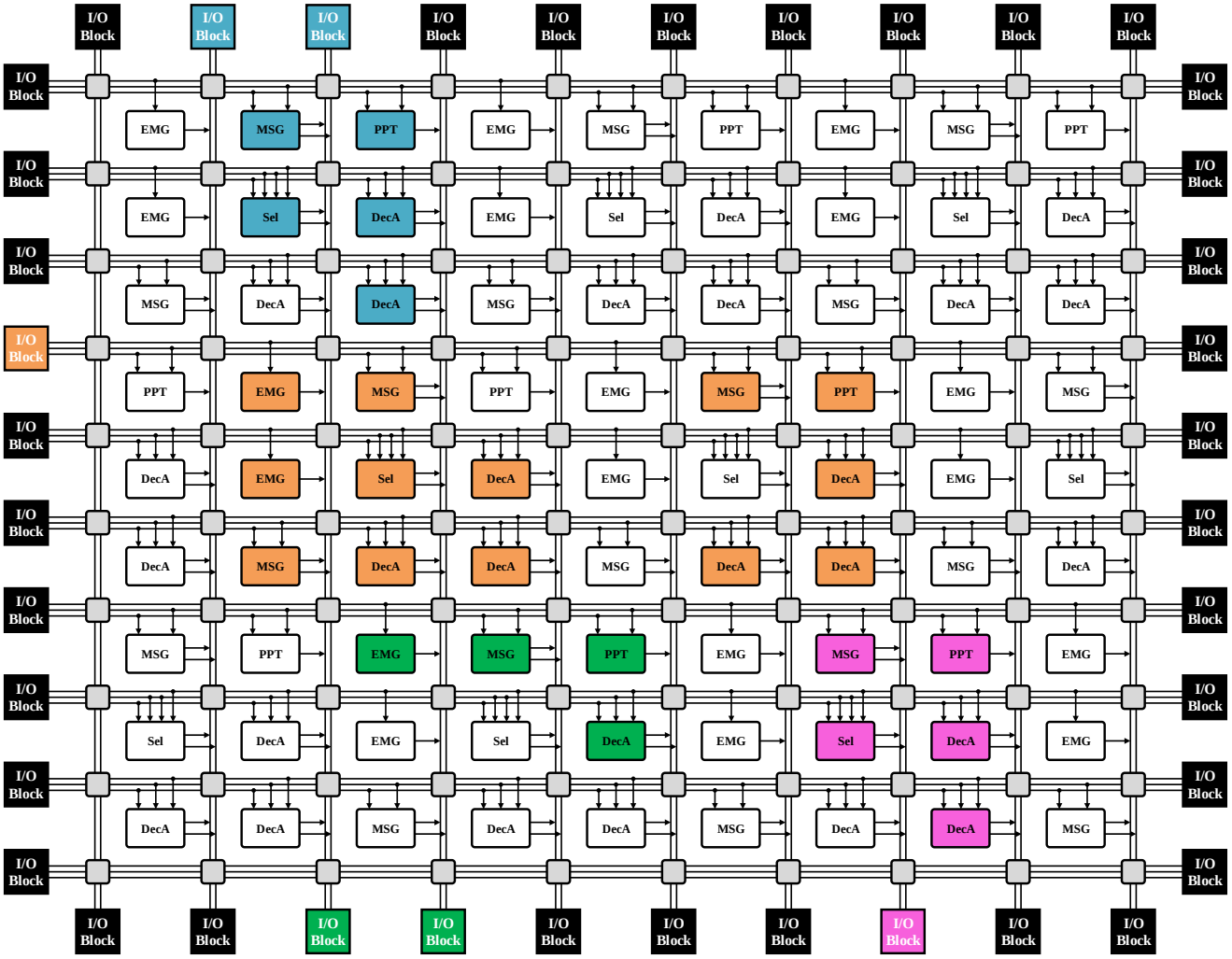
[7] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: from prototyping to deployment," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473-491, Apr. 2011.

[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 Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 10, pp. 1471-1482, Oct. 2010.

[10] S. Emami and M. Sedighi, "An optimized reconfigurable architecture for hardware implementation of decimal arithmetic," *Computers & Electrical Engineering*, vol. 63, pp. 18-29, Oct. 2017.

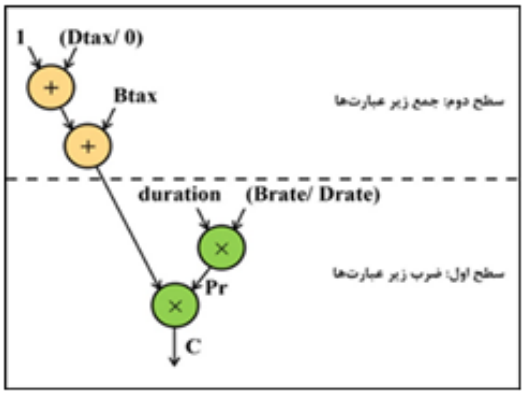
[11] M. Vladutiu, "Functional analysis and synthesis of binary and decimal adding and subtracting devices," in *Computer Arithmetic Algorithms and Hardware Implementations, Springer Berlin Heidelberg*, 2012.



شکل ۶: نگاشت LLDFG (۲) بر روی DARA با استفاده از روش ILP.

```
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));
```

(الف)



(ب)

شکل ۷: (الف) شبه‌کد و (ب) DFG ساده‌شده برنامه TELCO.

[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, pp. 233-240, 2009.

[15] R. Zimmermann, "Datapath synthesis for standard-cell design," in *Proc. of the 19th IEEE Symposium on Computer Arithmetic*, pp. 207-211, Portland, OR, USA, 08-10 Jun. 2009.

[16] C. K. Cheng, "Design space exploration for power-efficient mixed-radix ling adders," in *Proc. of the 19th IEEE Symp. on Computer Arithmetic*, pp. 212-212, Portland, OR, USA, 8-10 Jun. 2009.

[17] A. K. Verma, P. Brisk, and P. Ienne, "Challenges in automatic optimization of arithmetic circuits," in *Proc. of the 16th IEEE Symposium on Computer Arithmetic*, pp. 213-218, Portland, OR, USA, 8-10 Jun. 2009.

[18] Zimpl, *Zuse Institute Mathematical Programming Language*, Available at: http://zimpl.zib.de.

[19] SCIP, *Solving Constraint Integer Programs*, Available at: http://scip.zib.de.

[20] IBM Corporation, *The Telco Benchmark*, Retrieved May 10, 2022, from: http://speleotrove.com/decimal/telcoSpec.html.

سمانه امامی تحصیلات خود در مقاطع کارشناسی و کارشناسی ارشد را در گرایش نرم‌افزار دانشگاه شهید بهشتی به ترتیب در سال‌های ۱۳۸۷ و ۱۳۸۹ و در مقطع دکتری در گرایش معماری کامپیوتر دانشگاه صنعتی امیرکبیر در سال ۹۶ با درجه عالی به پایان رسانده است. وی هم‌اکنون استادیار دانشکده مهندسی برق و کامپیوتر دانشگاه سمنان می‌باشد. زمینه‌های پژوهشی مورد علاقه ایشان عبارتند از: حساب کامپیوتری دهدهی، سنتز سطح بالا و معماری‌های قابل بازپیکربندی.