طراحی و بهینه سازی مدارات منطقی ترکیبی با استفاده از الگوریتم ژنتیک

سید محسن موسوی دزفولی دانیال خشابی
دانشکده‌ی مهندسی برق، دانشگاه صنعتی امیرکبیر(پلی تکنیک)، تهران، ایران
d.khashabi@gmail.com moosavi.sm@gmail.com

چکیده- در این مقاله، با ذکر مقدماتی درباره الگوریتم های تکاملی[1]، بخصوص الگوریتم ژنتیک[2]، به معرفی روش هـایی فراتـر از قواعـد موجود برای طراحی مدارات منطقی ترکیبی[3] بر اساس الگوریتم های مذکور پرداخته شده است. روش های ارائه شده همگـی بـر اسـاس الگوریتم ژنتیک می باشند. در معرفی روش ها، جنبه های مختلف پیاده سازی و برتری هرکدام از آنها نسبت به هم بیـان شـده اسـت.در ادامه برخی از این روش ها پیاده سازی شده‌اند و نتایج حاصل از آنها با هم و با طراحی انسانی، مقایسه شده اند.

کلیدواژه- طراحی مدارات ترکیبی، بهینه سازی الگوریتم تکاملی، الگوریتم ژنتیک.

۱-مقدمه

طراحی مهارتی از انسان است که نیازمند دو توانـایی اسـت؛ یکی اطلاعات و دانـش، دیگـری خلاقیـت[1]. پیـاده‌سـازی دانش برای طراحی با استفاده از کـامپیوتر[4]، چنـدان دشـوار نیست. اما داشتن خلاقیتـی هـدف‌دار و بهینـه در طراحـی، برای یک برنامه‌ی رایانه‌ای کمی دشوار می باشـد. بخصـوص اینکه این خلاقیت باید برای محدوده‌ی وسـیعی از شـرایط- دراینجا، مدارات مورد نظر برای طراحی- سریع و بهینه عمل کند. می توان گفت آنچه که یک طـراح حرفـه ای را از یـک رایانه، متمـایز مـی کنـد، خلاقیـت اکتسـابی طـراح در اثـر تجربیات فراوان است. در طراحی و ساخت انسانی یک مـدار دیجیتال، چندین مرحله را باید بصورتی تکراری اجرا کرد تـا اینکه به طراحی نهایی از مدار رسید؛ ساخت، آزمایش، رفـع نقص. در حالی که کیفیت مدار حاصل تا حد زیادی وابسـته به مهـارت هـا و تجربیـات فـرد طـراح دارد. امـا روش هـای تکاملی این امکان را می دهند تـا بتـوان در طراحـی حالـت های ممکن بیشتری را بررسی کرد و احیانا حالاتی را بدست آورد که در حالت عادی به ذهن طراح خطور نمی کنـد[2]. زمینه‌ی تحقیقاتی بوجودآمده در سایه‌ی روش‌های طراحـی نوین سخت افزار، از جمله با استفاده از روش هـای تکـاملی، سخت افـزار تکـاملی[5](EHW) نـام دارد[21]. در واقـع در EHW روش‌های خودکاری برای طراحـی مـدارات بـه ازای ورودی و خروجی مشخص ارائه می شود.

در طراحی که مد نظر است، ورودی و خروجـی هـای تـابع مدار(جدول درستی مدار) در اختیار بـوده و هـدف طراحـی مدارات داخلی است، بطوریکه ورودی و خروجی هـای آن بـا بالاترین دقت منطبـق بـر خواسـته‌ی مسـاله باشـد. معمـولا شرایط به گونه‌ای است که لازم است تطابق تـام بـین مـدار طراحی شده و جدول حقیقت وجود داشته باشد؛ هرچند که نتوان ابعاد طراحی مدار را به بهینـه‌تـرین حالـت رسـاند. در واقع آنچه در اینجا مهم است، بدست آوردن ساده‌ترین مـدار ترکیبی، بـا کـم‌تـرین هزینـه‌ی محاسـباتی بـرای آن اسـت. معیارهای متفاوتی را می توان برای بهینه‌سـازی یـک مـدار ترکیبی در نظر گرفت. از جمله تعداد گیت‌های بکار رفتـه و تعداد سطوح طراحی می‌تواند معیار مناسبی بـرای ایـن کـار باشد. بحث مفصل دربـاره ی معیارهـای طراحـی یـک مـدار ترکیبی در بخش ۳ آمده است.

۱-۱- الگوریتم ژنتیک

الگـوریتم‌هـای ژنتیـک، دسـته‌ای از الگـوریتم‌هـای تکـاملی هستند که با الهام از الگوی طبیعی گذار نسل‌هـا و نظریـه‌ی انتخاب طبیعی داروین طراحی شده اند. برای تبدیل مساله- ی طراحی مدار، به مساله‌ای خـوش تعریـف[6] بـرای حـل بـا روش‌های تکاملی، باید ورودی و خروجی و همچنین دسـت- مایه طراحی-همان گیت‌های منطقی- را کدگذاری و آنهـا را وارد چرخه‌ی بهینه سازی با استفاده از الگوریتم ژنتیک کرد. کدگذاری مناسب برای حل مساله با الگوریتم ژنتیـک، مـی-

تواند تاثیر زیادی در سرعت حل مساله داشته باشد. تابع برازندگی[7]، میزان انطباق کروموزم (میزان مناسب‌بودن جواب متناظر) را در مقایسه با مقادیر خواسته شده نمایش می دهد. در واقع طراحی تابع برازندگی با توجه به ماهیت وابستگی این تابع به مساله‌ی مورد نظر و پارامتر های بهینه سازی انجام می گیرد. با تبدیل مساله به یک مساله‌ی الگوریتم ژنتیک و طراحی تابع برازندگی، اجرای بهینه‌سازی با تولید نسل اولیه شروع می شود.

می توان مراحل کار الگوریتم ژنتیک را به اینصورت خلاصه کرد:

۱. انتخاب نسل اولیه

۲. ارزیابی هرکدام از افراد نسل اولیه با استفاده از تابع برازندگی

۳. تکرار مراحل زیر تا زمان برقراری شرایط خاتمه

۳.۱. انتخاب از افراد برای تولید مجدد[8]

۳.۲. تولید جمعیت جدید با استفاده از جهش[9] و ترکیب[10] افراد انتخاب‌شده در مرحله‌ی قبل.

۳.۳. ارزیابی افراد با استفاده از تابع برازندگی.

۳.۴. انتخاب زیر مجموعه ای از جامعه به عنوان نسل جدید.

لذا لازم است برای بدست‌آوردن شیوه‌ی حل مساله با الگوریتم ژنتیک، موارد زیر مشخص شوند:

۱. نحوه‌ی کدینگ مدارهای منطقی به عنوان کروموزوم های الگوریتم ژنتیک.

۲. معرفی تابع برازندگی.

۳. انتخاب شیوه‌هایی برای ایجاد تغییر در نسل‌ها.

۴. انتخاب شیوه‌ای برای انتخاب و ایجاد نسل آینده.

۲- مروری بر کارهای انجام شده

احتمالا اولین سعی برای استفاده از الگوریتم‌های تکاملی برای بهینه سازی مدارها توسط Fridman انجام شده‌است که مربوط به دهه ی ۱۹۵۰ است. در رساله Fridman سعی شده است از ابزارهایی مشابه آنچه که امروزه آن را شبکه‌های عصبی می نامیم، برای بهینه‌سازی مدار مورد نظر استفاده شود[3].

اولین استفاده از الگوریتم ژنتیک برای طراحی مدارات منطقی توسط S. J. Louis در [4] است.

اولین بار J.Koza در [5] از "برنامه ریزی ژنتیک[11]" برای بدست آوردن توابع منطقی استفاده کرده است. اما تاکید وی بیشتر بر بدست آوردن جواب و طراحی بوده است، نه بهینه‌سازی آن. در ادامه تلاش‌های صورت گرفته اغلب با استفاده از الگوریتم ژنتیک و با تاکید بر ایجاد روش‌های کد-گذاری جدید و در نظرگرفتن محدودیت های مختلف بوده است. نمونه‌هایی از این دست در [6-7] دیده می‌شود. Coello در [1,8] با در نظرگرفتن محدودیت های تعداد سطوح طراحی، از الگوریتم ژنتیک استفاده کرد. در این مقاله روش معرفی شده (NGA[12]) با مثال‌هایی با نتایج بدست‌آمده از روش‌های متداول Quine-McClusky و جدول کارنو و همچنین سایر روش های قبلی مقایسه شده است و کارایی بهتر آن نشان داده‌شده‌است. در این مقاله از گیت های NOT,XOR,OR,AND به عنوان گیت‌های پایه استفاده شده‌است. در ادامه Coello الگوریتم بهینه-تری تحت عنوان MGA[13] معرفی کرده‌است که در آن تعداد گیت های منتج کمتر شده ولی تعداد ترانزیستورهای استفاده شده در آن بیشتر است[9-10].

در ادامه در [13] در روش Coello تغییراتی تحت عنوان Modified Evolutionary Algorithm ایجاد شده و نشان داده شده‌است که نتایج بهتری نسبت به MGA و NGA بدست می‌دهد.

در [10] ترکیبی از الگوریتم ژنتیک و تبرید تریجی[14] ادعا شده است که استفاده‌ی ترکیبی از این دو الگوریتم، نتیجه ی بهتری را بدست می دهد. Coello در [12] نیز مقایسه‌ای بین روش‌های مختلف پیاده‌سازی ترکیبی SA-GA انجام داده‌است. در [14] نیز روشی مشابه MGA و NGA معرفی شده‌است.

در [15] در طراحی مدار منطقی، تعداد ترانزیستورهای مورد استفاده در گیت‌ها نیز مد نظر گرفته است. در واقع در این مقاله، طراحی از سطح گیت[15]، به سطح ترانزیستور[16] منتقل شده است.

ساختارهای طراحی گرافیکی همچون جدول کارنو[17] برای حالاتی خاص که طراحی دوسطحی مدارات مورد نظر است، بصورت گسترده کاربرد دارد. جدول کارنو تنها توانایی بهینه‌سازی را برای توابعی حداکثر تا ۶ متغیر دارد. (متغیر-های با تعداد بالاتر دارای پیچیدگی بسیاری است.)

الگوریتم‌های محاسباتی همچون روش Quine-McCluskey در برنامه‌های آنالیز مدار همچون

Expresso و MisII با ترکیب با روش‌هایی دیگر، بطور گسترده مورد استفاده قرار می‌گیرند. این الگوریتم، توانایی بهینه‌سازی و بدست‌آوردن مدار دو سطحی با هر تعداد متغیر را داراست. اما با توجه به پیچیدگی محاسباتی نمایی نسبت به تعداد متغیرهای ورودی، هزینه‌ی محاسباتی آن بالاست. علاوه بر آن، پس از یافتن مین‌ترم‌ها[18]، باید در میان مجموعه‌ای از توابع ساده‌شده از مین‌ترم‌ها جستجو کرد تا کل جدول حقیقت را با کمترین هزینه پوشانده‌شود که خود مساله‌ای دارای پیچیدگی محاسباتی غیرچند جمله‌ای[19] است.

۳- بحثی بر شاخص های طراحی

هدف نهایی مساله، بدست آوردن طراحی از مدارات منطقی ترکیبی(با مشخص‌بودن محدوده‌ی انواع آن) با داشتن جدول درستی آنها است؛ بطوریکه بتوان بر اساس معیارهای موجود، آن را جوابی نزدیک بهینه نامید. معیارهای متفاوتی می‌توان برای تشخیص بهتربودن جواب به کار برد. برخی از مهم‌ترین موارد بدین صورت هستند:

الف) تعداد گیت های بکار رفته: قطعا هرچه تعداد گیت-های بکار رفته کمتر باشد، بهتر است. چرا که باعث صرفه-جویی در هزینه و فضا و زمان می‌شود.

ب)نوع گیت های بکار رفته: با در نظرگرفتن ساختار داخلی گیت‌ها، می توان به برخی از آنها در مقایسه با سایر گیت ها برتری‌هایی نسبت داد. برای مثال در ساخت گیت-های NOR و NAND از ۴ ترانزیستور و در ساخت گیت-های AND و OR به طور معمول از ۸ ترانزیستور استفاده می‌شود[1]. لذا استفاده از گیت‌های NAND و NOR موجب کاهش ترانزیستورهای مورد استفاده شده و بهینه‌تر است.

ج)تعداد سطوح طراحی مدار: هرچه تعداد سطوح مدار طراحی کمتر باشد، میزان تاخیر ایجادشده در بدست آوردن جواب نیز کمتر خواهدبود. همانطور که گفته شده روش های متداول از جمله روش Quine-McClusky و جدول کارنو، مدارات بهینه دارای دو سطح، با شرط کم بودن تعداد متغیرهای ورودی را بدست می دهند. معمولا می توان از تاخیر ایجادشده در انتشار سیگنال برای وجود بیش از چندین طبقه (برای مثال تا ۴ یا ۵ سطح)، به شرط بهینه-ترشدن تعداد گیت‌های منطقی صرف نظر کرد.

د)پیچیدگی الگوریتم: پیچیدگی الگوریتم ارائه‌شده، مساله‌ی بسیار مهمی است. چرا که برخی از الگوریتم‌ها اگرچه در تئوری امکان پذیر باشند، اما نیازمند پیاده‌سازی عملی هستند تا میزان عملی‌بودن آنها در زمان معقولی بدست‌آید. برای مثال اگرچه روش ارائه‌شده توسط Quine-McClusky در تئوری نشان می‌دهد که به ازای هر مداری می توان مدار بهینه‌ی دوسطحی آن را بدست‌آورد، اما پیچیدگی محاسباتی آن نمایی است. لذا پیاده‌سازی کامپیوتری آن به هیچ وجه مناسب مدارات با تعداد متغیر-های بالا نیست.

می توان با مثال‌هایی نشان داد که با پررنگ کردن وزن هر-کدام از معیارهای بهینه‌سازی مذکور، برای یک جدول حقیقت خاص، مدار می‌تواند جواب‌های متفاوتی بدست دهد. لذا می‌توان گفت با توجه به نیازهای مختلف، می‌توان طراحی الگوریتم را طوری انجام داد که مدارات حاصل، با توجه به نیاز فرد بهینه باشد. معمولا این جهت‌دهی به نتیجه، در مدار حاصل در الگوریتم ژنتیک، در تابع برازندگی و الگوریتم انتخاب انجام می گیرد.

در روش استفاده شده باید توجه داشت که نوع گیت‌های بکار رفته، باید مجموعه‌ی کاملی[20] را تشکیل دهند. کامل-بودن در اینجا به این معنی است که بتوان تمامی مدارات منطقی را با استفاده از آنها، تولید کرد. مجموعه‌های کامل از گیت‌ها عبارتند از:

۱- {AND,OR,NOT}
۲- {AND,NOT}
۳- {OR,NOT}
۴-{NAND}
۵-{NOR}

با توجه به ساختار مورد استفاده برای نمایش یک مدار، لازم است از گیت مجازیNULL برای پرکردن خانه های بدون گیت استفاده کرد. در واقع گیت NULL ورودی خود را به خروجی خود، بطور مستقیم وصل می کند.

زیادبودن تعداد گیت‌ها، مزیت این را دارد که می‌توان با تعداد گیت‌های کمتری به جواب بهینه‌تر رسید. در عین حال به ازای افزایش هر گیت به مجموعه‌ی گیت‌های مورد استفاده، فضای جستجوی شامل مدارهای ممکن، تقریبا دو برابر می شود. در واقع مزیت استفاده از مجموعه های ۵ و ۴ این است که این مجموعه‌ها به همراه گیت NULL

مجموعه‌ی گیت‌های دوعضوی را تشکیل می‌دهند. لذا فضای جستجو در آن نسبت به حالت ۱ بسیار کوچک است. در حالیکه که جواب‌های بدست‌آمده با استفاده از مجموعه-های ۴ و ۵، بطور معمول بزرگتر و دارای تعداد گیت‌های استفاده شده‌ی بیشتری هستند.

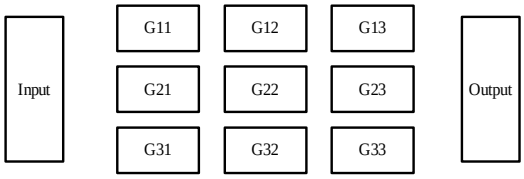
بدیهی است که اجتماع دو یا چند مجموعه‌ی کامل، مجموعه‌ی کاملی را تشکیل خواهدداد. می توان در پیاده-سازی، اجتماعی از مجموعه‌های کامل فوق را در نظر گرفت؛ برای مثال مجموعه‌ی {AND,XOR,OR,NAND} می تواند مجموعه‌ای باشد که بتواند اکثر مدارات را با تعداد گیت‌های کمتری بدست دهد. در [21] از مجموعه‌ی {XOR,NOT,NOR,NAND} برای طراحی مدارات استفاده شده‌است و با مثال‌هایی نشان داده‌شده‌است که نسبت به حالت {AND,OR,NOT} نتیجه‌ی بهتری را ارائه می‌دهد. در [1] با روشی تحت عنوان NGA از مجموعه گیت‌های {XOR,NOT,OR,AND} به عنوان گیت‌های پایه استفاده شده‌است.

۴- روش پیشنهادی

در ادامه دو روش پیشنهاد شده‌اند. در هر روش مرحله به مرحله، قسمت‌های مختلف الگوریتم و روش‌های استفاده شده، توضیح داده‌می‌شوند.

۴-۱- روش اول

الف) نمایش کد شده ی مدار: برای کدکردن گیت‌ها، ازماتریس دوبعدی بصورت شکل زیر استفاده شده است:



شکل۱: ساختار ماتریس گیت ها

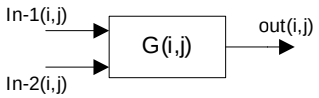
در این ساختار فرض را بر آن گرفته‌ایم که تمامی گیت‌ها بصورت گیت‌های دو ورودی اند. می توان هر مدار منطقی را با مشخصات زیر مشخص کرد:

۱) مکان گیت ها ($G_{i,j}$)

۲) مکان ورودی اول ودوم گیت‌ها ($in\_1_{i,j}$ و $in\_2_{i,j}$)

۳) نوع گیت؛ شامل گیت‌های NULL، AND، OR، NAND، NOR و XOR که با عددی بین ۰ تا ۵ مشخص می شوند.

در این ساختار هر گیت بطور ثابت دارای دو ورودی است که آن ها را از خروجی گیت‌های منطقی سطوح قبلی دریافت خواهدکرد.



شکل۲:ساختار هر گیت در کدینگ نوع اول

در ساختاری که توسط Louis و Miller و Carlos مورد استفاده قرار گرفته است، هر کدام از گیت‌ها، ورودی خود را فقط از طبقه ی ماقبل خود دریافت می کنند.همچنین در روش‌های مذکور برای کدکردن کروموزوم‌ها از نوع رشته‌ی باینری و یا ممیز شناور استفاده شده است[1,4,8,20].

ب)تولید جمعیت اولیه: جمعیت اولیه شامل تعدادی از ساختار های ماتریسی گیت ها می باشد که بطور تصادفی در سراسر فضا تولید می‌شوند. ویژگی تصادفی بودن جمعیت اولیه باعث خواهدشد تا پراکندگی فردها در فضای جستجو بطور یکنواخت صورت گیرد و حرکت به سمت جواب بصورتی یکنواخت انجام شود.

ج)تابع برازندگی: با توجه به معیارهای برتری در طراحی که در بخش ۳ ذکر شد، معیار بهتربودن، یک امید ریاضی[۲۱] از میزان تطابق خروجی مدار با جدول درستی و کم بودن تعداد گیت‌ها در نظر گرفته شده است:

$$f = \frac{w_{match} \times N_{match} + w_{null} \times N_{null}}{w_{match} + w_{null}} \qquad (۱)$$

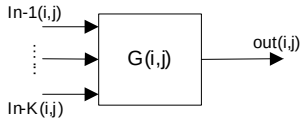
با توجه به اهمیت بیشتر تطابق خروجی مدار با جدول حقیقت، در پیاده‌سازی عملی، معمولا نسبت $\frac{w_{match}}{w_{null}} \cong 10$جواب‌های مناسبی را به دست داده است.

د)عملگر ژنتیکی ترکیب: همانطور که ذکر شد، با استفاده از این عملگر، دو یا چند مدار با هم ترکیب شده و مدار جدیدی را بدست می دهند. در پیاده‌سازی عملی، از روش یک‌نقطه‌ای[۲۲] استفاده شده‌است. لذا دو مدار بطور تصادفی انتخاب شده و با انتخاب تصادفی اندیس j، سطح j-ام آن مدار با یک مدار دیگر تعویض می شود.

ه)عملگر ژنتیکی جهش: در این عملگر، یک گیت خاص از ماتریس گیت‌ها، انتخاب می شود و ویژگی‌های آن بطور تصادفی عوض می شوند(ورودی‌ها و نوع گیت‌ها).

۴-۲- روش دوم

الف) نمایش کد شده‌ی مدار: در این روش نیز، از ساختار ماتریسی مشابه آنچه که در روش اول بکار رفته، استفاده شده‌است. با این تفاوت که در آن به جای گیت‌های دو ورودی از گیت‌های چندورودی استفاده شده‌است و همچنین هر گیت تنها می تواند ورودی خود را از گیت‌های سطر ما قبل خود دریافت کند. ساختار هر گیت در این روش به صورت شکل ۳ است.



شکل۳: ساختار هر گیت در کدینگ نوع دوم

می توان هر مدار منطقی را با ۷ عدد مشخص کرد:

۱) مکان گیت ($G_{i,j}$)

۲) نوع گیت؛ شامل گیت‌های NULL، AND، OR، NAND، NOR و XOR که با عددی بین ۰ تا ۵ مشخص می شوند.

۳) رشته‌ای از اعداد صفر و یک که نشان دهنده‌ی اتصال یا عدم اتصال خروجی‌های طبقه‌ی قبل، به ورودی گیت است. در واقع با توجه به شکل ۳، K تعداد خروجی‌های طبقه ماقبل است. در طبقه‌ی اول، این تعداد تعداد ورودی‌ها است و در سایر طبقات به تعداد گیت‌های طبقه‌ی قبلی است.

ب) تولید جمعیت اولیه: در این روش جمعیت اولیه به صورتی تصادفی تولید می‌شود. (به دلایل ذکر شده در روش قبل)

ج) تابع برازندگی: با توجه به دلایل ذکر شده در روش قبل، تابع برازندگی مورد استفاده در این روش مطابق رابطه (۱) است.

د) عملگر ژنتیکی ترکیب: از روش یک نقطه‌ای استفاده شده‌است. لذا با انتخاب اندیس تصادفی j و دو مدار تصادفی، مدارات دو سمت اندیس را دوبه‌دو به هم متصل کرده و مدار جدیدی را بدست می‌دهد.

ه) عملگر ژنتیکی جهش: این عملگر از جدول گیت‌ها، خانه‌ای به طور تصادفی انتخاب کرده و مقادیر آن را به طور تصادفی عوض می کند.

۵- مقایسه ی نتایج

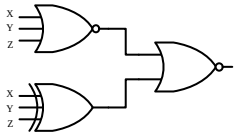
با توجه به اینکه گیت‌های سازنده‌ی روش پیشنهادی دوم می توانند گیت‌های چند ورودی(بزرگتر از ۲) باشند، شاید مقایسه‌ی نتایج این الگوریتم با نتیجه‌ی الگوریتم‌های دیگر که همگی از گیت‌های دو ورودی به عنوان گیت پایه استفاده کرده‌اند، چندان درست نباشد. با این حال در ادامه با ذکر مثال‌هایی مقایسه‌ای بین نتایج الگوریتم‌های مختلف انجام می‌دهیم. باید متذکر شد که تمامی توابعی که در اینجا روی آنها کار می‌کنیم، بدون حالت‌های «بدون اهمیت»[۲۳] هستند. پیاده‌سازی توابع همراه با حالت‌های بدون اهمیت، بسیار ساده‌تر است. چرا که در عملیات محاسبه‌ی برازندگی برای هرکدام از افراد، دیگر احتیاجی به محاسبه‌ی مقدار خروجی به ازای ورودی‌های بدون اهمیت نیست.

۵-۱- آزمایش اول: تابع سه متغیره

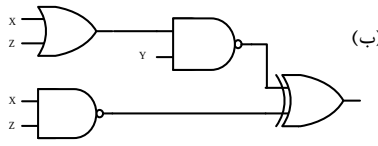
تابع f_1 را با مین‌ترم‌های آن به صورت زیر تعریف می‌کنیم:

$$f_1(X,Y,Z) = \sum(0,0,0,1,0,1,1,0)$$

در ذیل نتایج بدست آمده برای این تابع توسط روش‌های مختلف را آورده‌ایم:



(الف)



(ب)

شکل۴: مدار بدست آمده برای تابع f_1 توسط الگوریتم (الف) روش اول پیشنهاد شده (ب) روش دوم پیشنهاد شده

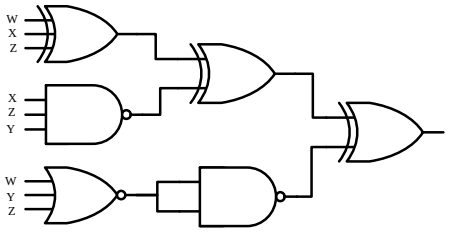
با توجه به جدول ۱ مشاهده می شود که جواب روش اول با وجود تعداد گیت‌های یکسان و تعداد سطوح برابر با روش NGA، دارای گیت‌های NAND به جای AND است. لذا در کل جواب روش اول، نسبت به جواب روش دوم دارای

ترانزیستورهای کمتری است. همچنین با توجه بـه جـدول۱، در جواب روش دوم با افزایش تعداد ورودی گیت‌هـا، تعـداد سطوح و تعداد گیت‌ها، کاهش یافته است.

۵-۲- تابع ۴ متغیره

تابع f_2 را با مینترم‌های آن به صورت زیر تعریف می‌کنیم:

$$f_2(W,X,Y,Z) = \sum(1,1,0,1,0,0,1,1,1,0,1,0,0,1,0,0)$$

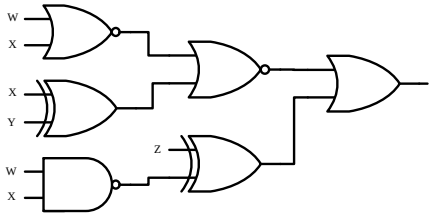


شکل۵: مدار بدست آمده برای تابع f_2 بوسیله روش دوم ارائه شده

مقایسه ی بین نتایج چند الگوریتم در جدول ۲ آمده است.

۵-۳- تابع ۴ متغیره

$$f_3(W,X,Y,Z) = \sum(1,0,1,0,1,0,1,1,1,1,1,0,0,1,1,1)$$

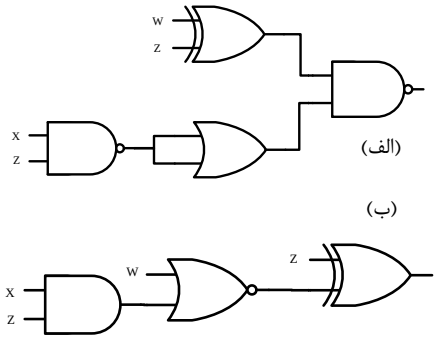


شکل۶: مدار بدست آمده برای تابع f_3 بوسیله الگوریتم روش اول ارائه شده

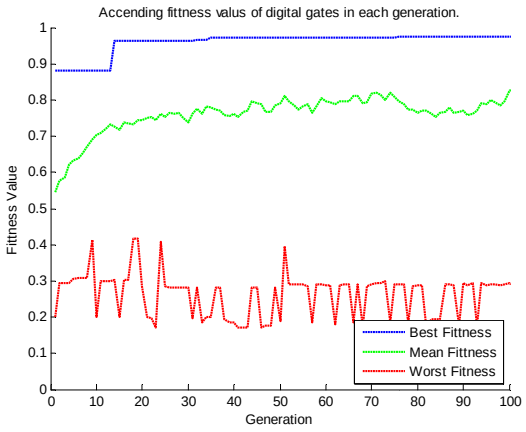
مقایسه ی بین نتایج چند الگوریتم در جدول ۳ آمده است.

۵-۴- تابع ۴ متغیره

$$f_4(W,X,Y,Z) = \sum(1,0,1,0,1,1,1,1,1,0,1,0,1,0,1,01)$$



شکل۷: مدار بدست آمده برای تابع f_4 بوسیله(الف)روش اول ارائه شده (ب)روش دوم ارائه شده



شکل۸: نمایش همگرایی الگوریتم به جواب برای تابع f_4با روش اول

جدول۱:مقایسه‌ی نتایج بدست آمده از پیاده سازی‌های مختلف تابع f_1

طراح	فرمول بسته	تعداد گیت ها	تعداد سطوح
روش اول ارائه شده	$(B.(A+C))'\oplus(AC)'$	۴ گیت (۲ NAND، ۱ OR ، ۱ XOR)	۳
روش اول ارائه شده	$[(X\oplus Y\oplus Z)+(X+Y+Z)']'$	۳ گیت (۱XOR(سه ورودی)، ۲ NOR (سه ورودی))	۲
روش **MGA**	$Z(X+Y)\oplus(XY)$	۴ گیت (۲ AND، ۱ OR، ۱ XOR)	۳
روش **NGA**	$(Z+Y)(Y\oplus(X\oplus Z))'$	۵ گیت (۱ AND، ۱ OR، ۲ XOR، ۱ NOT)	۴
طراحی انسانی	$Z(X\oplus Y)+Y(X\oplus Z)$	۵ گیت (۲ AND، 1OR، ۲ XOR)	۳

جدول۲:مقایسه‌ی نتایج بدست آمده از پیاده سازی‌های مختلف تابع f_2

طراح	فرمول بسته	تعداد گیت ها	تعداد سطوح
روش دوم ارائه شده	$[(W+Y+Z)]\oplus[\,(W\oplus X\oplus Z)\oplus(XYZ)'\,]$	۶ گیت (۳ XOR، ۲ NAND، ۱ NOR)	۳
روش **MGA**[۲۴]	$[\,(\,X\oplus(XY)\,)\oplus(\,(W+Y+Z)\oplus W\,)\,]'$	۷ گیت (۱ AND، ۲OR، ۳XOR، ۱ NOT)	۵
روش **NGA**[۲۵]	$[XY'Z\oplus[\,(X+Z)\oplus W\oplus((Y+Z)+W)\,]]'$	۱۰ گیت (۲ AND، ۳ OR، ۳ XOR، ۲ NOT)	۶
روش **BGA**[۲۵]	$W\oplus\big(\,(\,X\oplus Z)\oplus(YZ)\big)\oplus(W+Y+Z)'$	۸گیت (۱ AND، ۳OR، ۳XOR، ۱ NOT)	۵
روش **Saso**[۲۵]	$Y'\oplus Z'X'\oplus YZ'W'\oplus Y'Z'X$	۱۲ گیت (۳ XOR، ۲ AND، ۴ NOT، ۳ NOR)	۴
روش عسگریان[21]	$((Y+Z)'+(W\oplus Z))'\oplus((YZ)'.X)'$	۶ گیت (۲ NAND،۲NOR ،۲XOR)	۳
طراحی انسانی	$\big((W'Y)\oplus(Y'W')\big)+((X'Y)(Z+W'))$	۱۱ گیت (۴ AND، ۱ OR، ۲ XOR ، ۴ NOT)	۴

جدول۳ ؛ مقایسه‌ی نتایج بدست آمده از پیاده سازی‌های مختلف تابع f_3

طراح	فرمول بسته	تعداد گیت ها	تعداد سطوح
روش اول ارائه شده	$[(X\oplus Y)+(W+X)']'+\;[(WX)'\oplus Z]$	۶ گیت (۲ NOR، ۲ XOR، ۱NAND، ۱ OR)	۳
روش **NGA**	$[\,(XY\oplus(X+Z)).(Y+(W\oplus Z))\,]'$	۷ گیت (۲ XOR، ۲ OR ، ۲ AND ، ۱ NOT)	۴
روش **BGA**	$[\,((W\oplus Z)+WY).((X+Z)\oplus XY)\,]'$	۸ گیت (۲ XOR ، ۳ AND ، ۲ OR ، ۱ NOT)	۴

۶- بررسی علل عدم همگرایی به پاسخ کاملا درست(مدار معتبر)

در برخی موارد، برنامه های پیاده سازی شده‌ی الگوریتم های ارائه شده توسط مقالات، به جواب هایی، با اختلاف ۱یا ۲ خروجی با جدول حقیقت خواسته شده می رسند. در واقع جواب‌های مورد نظر در این حالت به مینیموم محلی رسیده و در آن به دام می افتند. یکی از مهم‌ترین دلایلی که به نظر نویسندگان باعث ایجاد این مشکل می شود، عدم توانایی توابع برازندگی ارائه شده در پیش‌بینی فاصله‌ی نسبی با مدار معتبر است. در واقع راه رسیدن به یک مدار معتبر الزاما از مسیر یک مدار با اختلاف کمتر با جدول حقیقت نمی‌گذرد. برای فایق آمدن بر این مشکل روش هایی استفاده شده که بطور خلاصه ذکر می گردد:

الف)استفاده از روش های انتخاب مختلف: استفاده از روش انتخاب چرخ رولت باعث می‌شود، اعضای نسل بعدی به صورتی پراکنده‌تر شوند. لذا احتمال رسیدن به مدار معتبر را در نقطه‌ای دیگر از فضای جستجو را بیشتر می کند. البته به شرطی مفید واقع می شود، که برازندگی بتواند در مراحل بعدی جواب را به یک مدار معتبر هدایت کند. در غیر اینصورت باید آنقدر منتظر بود تا مدار معتبر بصورت تصادفی ایجاد شود.

ب)تقسیم جدول حقیقت: در صورتی که تشخیص داده شود که جواب در مقدار حداکثر مشخصی (معمولا عددی حدود ۲۰۰) به جواب نمی رسد، جدول حقیقت به دو قسمت تقسیم می شود و هرکدام جداگانه بهینه شده و با هم ترکیب می شوند. سپس مدار بدست‌آمده ترکیبی را وارد جمعیت جدیدی کرده و سعی در بهینه‌سازی آن می شود. این روش تضمین آن را به ما می دهد که همواره و به ازای هر جدول حقیقتی بتوان به جوابی درست دست یافت.

طی مکاتباتی که با Carlos Coello و Arturo Hernandez نویسندگان [1,8,9,10,12] انجام شد، ایشان ادعا بر استفاده از تابع Mutation با نسبت بسیار بالا داشته‌اند. اگرچه با آزمایش پیشنهادات این افراد، متاسفانه مشکل به دام افتادن در بهینه ی محلی در الگوریتم های معرفی شده، برطرف نشد. به نظر نویسندگان، بهینه‌سازی‌های انجام شده، توسط سایر مقالات، به هیچ وجه کارآمد نیست. در واقع در تمامی مقالات با عناوینی همچون بهینه‌سازی چندهدفه توابع برازندگی مطرح شده، اگرچه میزان درست بودن جواب را نشان می دهد، اما به

هیچ وجه فاصله‌ای نسبی را بـه سـمت جـوابی معتبـر بیـان نمی‌کند. لذا ظن این می رود، که جواب هـای بدسـت آمـده درمقالات، اکثرا تصادفی و با ایجاد نسـل‌هـا و جمعیـت بـالا بدست آمده باشد.(چنان که در مقالات به این مسـاله اشـاره شده است.) باید تاکید کرد که بدسـت آوردن جـواب هـایی درست با نسل‌ها و جمعیت بالا، از نظر نویسندگان بـه هـیچ وجه قابل قبول نیست. چرا که با داشتن هزینه‌ی محاسـباتی بالا و نیاز به زمان بالا برای محاسـبات، کارآمـدی در مقابـل روش‌های موجود ندارد. چرا که در واقع در این حالت برنامـه از ویژگی‌های الگوریتم ژنتیک، یعنی سـوق دادن جـواب بـه سمت بهترین جواب، با استفاده از تـابع برازنـدگی، اسـتفاده نمی‌شود. و این یعنی اینکه در این حالت، آنچه انجـام مـی-شـود، تفـاوت چنـدانی بـا جسـتجوی کورکورانـه در فضـای جستجو نخواهد داشت.

۷- جمع بندی، نتیجه گیری و کار آینده

در این نوشته، علاوه بر معرفی کلی الگـوریتم ژنتیـک، از آن در بهینه‌سازی یک مدار ترکیبی استفاده شـد. بـا توجـه بـه مطالب گفته شده، دیده شـد کـه شـیوه‌ی کـدینگ مسـئله، می‌تواند تاثیر زیادی در بدست آوردن جواب داشته باشد. در واقع پایداری راه حـل‌هـا، در بدسـت آوردن جـواب، لغزنـده است. بدین معنی که تغییر کوچکی نحوه‌ی اجرای الگـوریتم و در نسـبت اجـرای عملگرهـای ژنتیکـی، مـی توانـد باعـث بدست‌آمدن نتایج جدیدتری شود. با توجه به دو کدینگ کـه در این مقاله ارائه شد، نتایج جدیدی بدست آمد که با برخی از مدارات سابقا بدست‌آمده، مقایسه شدند.

همانطور که گفته‌شد در طراحی مدارات منطقی بـا اسـتفاده از الگوریتم تکاملی بطور کلـی دو نقـص روش‌هـای متـداول برطرف می شود:

۱- طراحی مـدارات بـا سـطوح بیشـتر بـرای بدسـت‌آوردن طراحی بهینه‌تر.

۲- بهینه‌سازی چندین تـابع بطـور همزمـان (بـر روی یـک ماتریس یکسان).

باید توجه کرد که با توجه به تمـامی روش‌هـای نـوین ارائـه شده برای طراحی مدارات ترکیبی، هنـوز مهمتـرین اشـکال طراحی‌های سنتی(نظیر جدول کـارنو و الگـوریتم Quine-McClusky) برطرف نشـده‌اسـت و آن عـدم توانـایی ایـن روش‌هـا در طراحـی مـدارات بـا تعـداد ورودی بالاسـت. محدودیت‌های پردازشی و حافظه‌ای، لازم می دارند تا بـرای توابع با تعداد متغیرهـای بـالا، روش هـایی خـارج از حـدود عـادت(Heuristic) ابـداع و بـا روش هـای موجـود همـراه شوند. بهینه سازی روی طراحی مدارات منطقـی ترتیبـی بـا استفاده از الگوریتم های تکاملی کـه دارای عناصـری مثـل فیدبک یا عناصر حافظه‌اند، می تواند جالب باشد. اسـتفاده از روش‌های دیگر بهینه‌سازی از جمله بهینه‌سـازی مورچگـان، ازدحام ذرات و ... برای این‌کار نیز مـورد توجـه مـی باشـد. نمونه هایی از این پیاده‌سازی‌ها در [17-19] انجـام گرفتـه شده است.

سپاسگزاری

نویسندگان از جناب آقای دکتر ابوالقاسـم راعـی و مهنـدس احسان امیدی بـه خـاطر راهنمـای ایشـان در پیـاده‌سـازی الگوریتم‌ها، تشکر و قدردانی می‌نمایند.

مراجع

[1] C.A.C. Coello, A.D.Christiansen, A.H.Aguirro, "Toward Automated Evolutionary Design of Combinational Circuits", Department of Computer Science, Tulane University, New Orleans, USA, 1999.

[2] Greene J., "Simulated Evolution and Adaptive Search in Engineering Design", Experiences at the University of Cape Town, in 2nd Online Workshop on Soft Computing, July 1997.

[3] Arturo Hernández Aguirre and Carlos A. Coello Coello, "Using Genetic Programming and Multiplexers for the Synthesis of Logic Circuits",*Engineering Optimization*, Vol. 36, No. 4, pp. 491--511, August 2004.

[4] Sushil J. Louis, Gregory J.E. Rawlins: "Designer Genetic Algorithms: Genetic algorithms in StructureDesign", *Procs of the Fourth InternationalConference on Genetic Algorithm*, pages 53-60, 1991

[5] J. R. Koza, "Genetic Programming; On the Programming of Computers by Means of Natural Selection", Cambridge, MA; MIT Press, 1992.

[6] Cecília Reis , J. A. Tenreiro Machado , J. Boaventura Cunha, "Synthesis of Logic Circuits Using Fractional-Order Dynamic Fitness Functions", *Procs. of the ICCI'2004–*

International Conf. on Computational Intelligence.

[7] Slowik, A. Bialko, M., "Evolutionary design of combinational digital circuits: State of the art, main problems, and future trends",*1st International Conf. on IT*, 2008.

[8] C.A.C. Coello,A.D.Christiansen, A.H.Aguirro, "Using Genetic Algorithms to Design Combinational Logic Circuits", Department of Computer Science, Tulane University, New Orleans, USA, 1996.

[9] Coello Coello, Carlos A. and Hernández Aguirre, Arturo, "Use of a Population-based Evolutionary Multiobjective Optimization Technique to Design Combinational Logic Circuits", *Tercer Encuentro Internacional de Ciencias de la Computación (ENC'01)*, Tomo I, pp. 95--104, Aguascalientes, Aguascalientes, Septiembre 2001.

[10] Carlos A. Coello Coello and Arturo Hernández Aguirre , "Design of Combinational Logic Circuits through an Evolutionary Multi-objective Optimization Approach", *Journal of Artificial Intelligence for Engineering, Design, Analysis and Manufacture*, 16(1), pp. 39-53, January 2002.

[11] I. R. Obregonand and A. P. Pawlovsky , "A Hybrid SA-GA Method for Finding the Maximum Number of Switching Gates in a Combinational Circuit", *The 23rd International Technical Conf. on Circuits/Systems, Computers and Communications*, July 6-9, 2008.

[12] C. A. C. Coello, E. Alba, G. Lugue, A. H. Aguirre, "Comparing Different Serial and Parallel Heuristics to Design Combinational Logic Circuits", *Procs. of the 2003 NASA/DoD Conference on Evolvable Hardware*.

[13] A. Słowik, M Białko, "Design and Optimization of Combinational Digital Circuits Using Modified Evolutionary Algorithm",*Procs. of 7th International Conf. on Artificial Intelligence and Soft Computing*, Lecture Notes in Artificial Intelligence, 2004.

[14] C. Reis, J. A. T. Machado, J. B. Cunha , "Evolutionary Design of Combinational Logic Circuits", *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Fuji Technology Press , 2004.

[15] Z. Gajda, L. Sekanina, "Reducing the Number of Transistors in Digital Circuits Using Gate-Level Evolutionary Design", *Procs. of the 9th Annual Conf. on Genetic and Evolutionary Computation*, 2007.

[16] M. Bialko, A. Slowik, "Evolutionary Design and Optimization of Combinational Digital Circuits with Respect to Transistor Count", *Bulletin of the Polish Academy of Sciences*, Technical Sciences, Volume 54, Number 4.

[17] V.G. Gudise, G. K. Venayagamoorthy, "Evolving Digital Circuits Using Particle Swarm", Dept. of Electrical and Computer Engineering, University of Missouri - Rolla, USA.

[18] C.A.C. Ceollo et al, "Ant Colony System for the Design of Combinational Logic Circuits", EECS Department, Tulane University, New Orleans, LA, USA, 2000.

[19] E. H. Luna, C. A. C. Coello, A.H.Aguirre, "On the Use of a Population-Based Particle Swarm Optimizer to Design Combinational Logic Circuits", Evolutionary Computation Group, Dpto. deIng. Elect./Secc. Computaci´on, MEXICO.

[20] J. F. Miller, P. Thomson, T. Fogarty, "Designing Electronic Circuits Using Evolutionary Algorithm", Dept. of Computer Studies, Napier University, 1997.

[۲۱] احسان عسگریان، جعفر حبیبی، "بهینه سازی مدارات ترکیبی در سطح گیت با استفاده از الگوریتم ژنتیک"، *اولین کنفرانس مشترک فازی و سیستم های هوشمند، دانشگاه فردوسی مشهد*، شهریور ۸۶، مشهد.

پی نوشت ها:

1 Evolutionary Algorithms
2 Genetic Algorithm
3 Combinational Logic Circuits
4 Computer Aided Design
5 Evolvable Hardware
6 Well-Defined
7 Fittness Function
8 Reproduction
9 Mutation
10 Crossover
11 Genetic Programming
12 N-cardinality Genetic Algorithm
13 Multi-Objective Genetic Algorithm
14 Simulated Anealing
15 Gate Level Design
16 Transistor Level Design
17 Karnaugh Map
18 Minterm
19 Non-Polynomial
20 Complete Set
21 Expectation
22 One-Point Crossover
23 Don't Care

۲۴ در مقاله ی اصلی ۸ گیت نوشته شده است، که با توجه فرمول تابع ۷ گیت است. در ضمن ترتیب متغیر ها به صورت W,X,Y,Z اصلاح شده است.

۲۵ نسبت به مقاله ی اصلی، ترتیب متغیر ها به صورت W,X,Y,Z اصلاح شده است.