طراحی و بهینه سازی مدارات منطقی ترکیبی
با استفاده از الگوریتم ژنتیک
«پروژه ی درس مدار منطقی»

سید محسن موسوی دزفولی و دانیال خشابی

دانشگاه صنعتی امیرکبیر — دانشگاه صنعتی امیرکبیر

دانشکده ی مهندسی برق — دانشکده ی مهندسی برق

moosavi.sm@gmail.com — d.khashabi@gmail.com

به راهنمایی دکتر ابوالقاسم راعی

بهار ۱۳۸۹

چکیده:در این نوشته، با ذکر مقدماتی درباره الگوریتم های حل تکاملی[1]، بخصوص الگوریتم ژنتیک[2]، به معرفی روش هایی فراتر از قواعد موجود برای طراحی مدارات منطقی ترکیبی[3] بر اساس الگوریتم های مذکور پرداخته شده است. روش های ارائه شده همگی بر اساس الگوریتم ژنتیک می باشند. در معرفی روش ها، جنبه های مختلف پیاده سازی و برتری هرکدام از آنها نسبت به هم بیان شده است.در ادامه برخی از این روش ها، در MATLAB پیاده سازی شده اند و نتایج حاصل از آنها با هم و در مقایسه با طراحی انسانی، مقایسه شده اند. برای نمایش خروجی الگوریتم ها، رابط گرافیکی(GUI) طراحی و پیاده سازی شده است که در ضمیمه معرفی شده است.

کلمات کلیدی: طراحی مدارات ترکیبی، بهینه سازی الگوریتم تکاملی، الگوریتم ژنتیک.

۱. مقدمه

طراحی مهارتی از انسان است که نیازمند دو توانایی است؛ یکی اطلاعات و دانش، دیگری خلاقیت[1]. پیاده سازی دانش برای طراحی با استفاده از کامپیوتر[4]، چندان دشوار نیست. اما داشتن خلاقیتی هدف دار و بهینه در طراحی، برای یک برنامه ی رایانه ای کمی دشوار می باشد. بخصوص اینکه این خلاقیت باید برای محدوده ی وسیعی از شرایط-دراینجا، مدارات مورد نظر برای طراحی- سریع و بهینه عمل کند. می توان گفت آنچه که یک طراح حرفه ای را از یک رایانه، متمایز می کند، خلاقیت اکتسابی طراح در اثر تجربیات فراوان است. در طراحی و ساخت انسانی یک مدار دیجیتال، چندین مرحله را باید بصورتی تکراری اجرا کرد تا اینکه به طراحی نهایی از مدار رسید؛ ساخت(Build)، آزمایش (Test)، رفع نقص(Debug). در حالی که کیفیت مدار حاصل تا حد زیادی وابسته به مهارت ها و تجربیات فرد طراح دارد. اما روش های تکاملی این امکان را می دهند تا بتوان در طراحی حالت های ممکن بیشتری را بررسی کرد و احیانا حالاتی را بدست آورد که در حالت عادی به ذهن طراح خطور نمی کند[2]. زمینه ی تحقیقاتی

1 Evolutionary Algorithms
2 Genetic Algorithm
3 Combinational Logic Circuits
4 Computer Aided Design

بوجود آمده در سایه ی متدهای طراحی نوین سخت افزار، از جمله با استفاده از روش های تکاملی، سخت افـزار تکـاملی[5](EHW) نام دارد[21]. در واقع در EHW روش های خودکاری برای طراحی مدارات به ازای ورودی و خروجی مشخص ارائه می شود.

در طراحی که مد نظر است، ورودی و خروجی های تابع مدار(جدول درستی مدار) در اختیار بوده و هدف طراحی مدارات داخلی است، بطوریکه ورودی و خروجی های آن با بالاترین دقت منطبق بر خواسته ی مساله باشد. معمولا شرایط به گونه ای است که لازم است تطابق تام بین مدار طراحی شده و جدول حقیقت وجود داشته باشد. (هرچند که نتوان ابعاد طراحی مدار را به بهینه ترین حالت رساند.) در واقع آنچه در اینجا مهم است، بدست آوردن ساده ترین مدار ترکیبی، با کم ترین هزینه ی محاسباتی برای آن است. معیار های متفاوتی را می توان برای بهینه سازی یک مدار ترکیبی در نظر گرفت. از جمله تعداد گیت های بکار رفته و تعداد سطوح طراحی می تواند معیار مناسبی برای این کار باشد. بحث مفصل درباره ی معیار های طراحی یک مدار ترکیبی در ادامه آمده است.

۱.۱. الگوریتم ژنتیک

الگوریتمهای تکاملی دسته‌ای از الگوریتمهای جستجو هستند که در هر تکرار، جمعیتی از جوابها را مورد بررسی قرار داده و ذخیره می نمایند. به مجموعه ی جواب های ذخیره شده نسل(Generation) و هر کدام از جواب ها، عضو مفرد(Individual) گفته می شود.

الگوریتم های ژنتیک، دسته ای از الگوریتم های تکاملی هستند که با الهام از الگوی طبیعی گذار نسل ها و نظریه ی انتخاب طبیعی داروین طراحی شده اند. بر طبق این الگوریتم، هر کدام از نقاط فضای جستجو را یک کروموزوم می نامند. نمایش های مختلفی را می توان برای کروموزوم ها در نظر گرفت؛ از جمله بصورت آرایه ای از اعداد دودویی، صحیح، اعشاری و ... . برای تبدیل مساله ی طراحی مدار، به مساله ای خوش تعریف(well-defined) برای حل با روش های تکاملی، باید ورودی و خروجی و همچنین ابزار های طراحی-در واقع همان گیت های منطقی- را کد گذاری و آنها را وارد چرخه ی بهینه سازی با استفاده از الگوریتم ژنتیک کرد. کدگذاری مناسب برای حل مساله با الگوریتم ژنتیک، می تواند تاثیر زیادی در سرعت حل مساله داشته باشد.

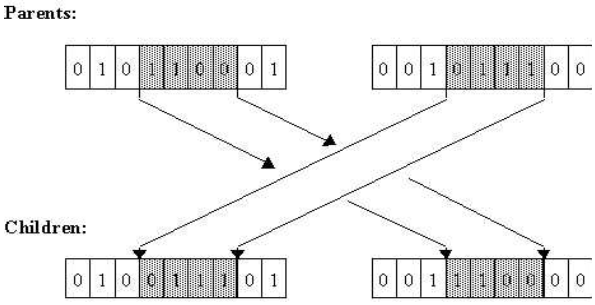
تابع برازندگی (Fitness Function)، میزان انطباق کروموزم (میزان خوب بودن جواب متناظر) را در مقایسه با مقادیر خواسته شده نمایش می دهد. در واقع طراحی تابع برازندگی با توجه به ماهیت وابستگی این تابع به مساله ی مورد نظر و پارامتر های بهینه سازی انجام می گیرد.

با تبدیل مساله به یک مساله ی الگوریتم ژنتیک و طراحی تابع برازندگی، اجرای بهینه سازی با تولید نسل اولیه شـروع مـی شـود. معمولا نسل اولیه به صورتی تصادفی تولید می شود که تعداد جمعیت آن به نوع مساله بستگی دارد. معمولا تولید نسل اولیـه بایـد بگونه ای باشد که حالت های مختلفی را پوشش دهد. بطوریکه امکان ایجاد فرد ها را در هر نقطه از فضـای جسـتجو داشـته باشـد. این ویژگی یعنی پراکنده بودن جمعیت اولیه یا جمعیت با پراکندگی(diversity) بالا در فضای جستجو، امکان رسیدن بـه جـواب ها را بالاتر می برد. در ادامه عملیات تکراری روی نسل اولیه با استفاده از عملگر های تغییر انجام مـی پـذیرد. عملگـر هـای ایجـاد تغییر در الگوریتم ژنتیک را می توان اینگونه معرفی کرد:

باز تولید(Reproduction): سوق دادن جواب به بخش هایی از فضا که امکان یافتن جواب هایی با کیفیت بالاتر وجود دارد. در واقع با این کار، از تعداد از اعضای جامعه انتخاب می شوند تا روی آنها عملیاتی برای تولید نصل آینده انجام گیـرد. ایـن عملیـات را در کل می توان به دو دسته تقسیم کرد:

5 Evolvable Hardware

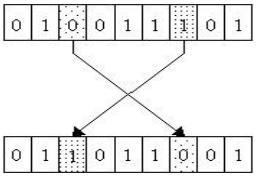
۱. **باز ترکیب(Crossover/Recombination)**: در این تغییر، دو عضو به عنوان والدین انتخاب می شود و با ترکیب آنها، دو عضور جدید(فرزندان) به جامعه اضافه می شود. در شکل زیر ترکیب والدین به روش دو نقطه ای(Two-Point Crossover) دیده می شود.



تصویر۱ ؛ نمایش ترکیب(Crossover) دو نقطه ای روی ژنوم های باینری

گرچه الگوریتم های ترکیب اعضا(Crossover) بر اساس دو والد در طبیعت بیشتر جلوه و مفهوم دارد، برخی از تحقیقات نشان دهنده ی آن است که استفاده از بیش از دو عضو برای تولید نسل بعدی، می تواند کارآمدتر باشد.

۲. **جهش(Mutation)**: ویژگی تصادفی بودن که امکان فرار از افتادن در نقاط بهینه ی محلی[6] را فراهم می آورد. در این عمل تغییراتی در خود کروموزم ها داده می شود. برای مثال اگر کرموزوم ها، رشته ای از اعداد دودویی باشند، می توان با انتخاب نقاطی تصادفی از آن، مقدار آن نقاط را نقیض کرد.



تصویر۲؛ نمایش جهش(Mutation) روی ژنوم های باینری

انتخاب (Selection): در هر نسل، تعدادی از جمعیت حاضر انتخاب میشوند تا نسل بعدی را تولید کنند. معیار انتخاب اعضای نسل با توجه به تابع برازندگی انجام می گیرد. اغلب این توابع اتفاقی-احتمالی- هستند. بطوریکه تنها عده ای از اعضای منفرد را که شایستگی کمتری برای حضور در نسل را دارا هستند، حذف می کنند. این کار علاوه بر اینکه باعث می شود پراکندگی(diversity) جواب ها در فضای جستجو از بین نرود، هزینه ی محاسباتی زیادی به تولید هر نسل از الگوریتم تحمیل نشود. روش انتخاب چرخ رولت(Roulette Wheel Selection) و روش انتخاب رقابتی(Tournament Selection)از مهمترین روش های انتخاب اعضای منفرد هستند. در روش انتخاب Roulette Wheel، به هرکدام از اعضا به نسبت fitness آنها، احتمالی نسبت داده می شود و با توجه به احتمال آنها، اعضای نسل بعدی، از بین آنها انتخاب می شوند. در واقع فردی که دارای fitness بیشتری است، شانس بیشتری را برای حضور در نسل آینده داراست.

6 Local Minima

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

۱. انتخاب نسل اولیه(Initial Population)

۲. ارزیابی هر کدام از افراد نسل اولیه با استفاده از تابع برازندگی(Fitness Function)

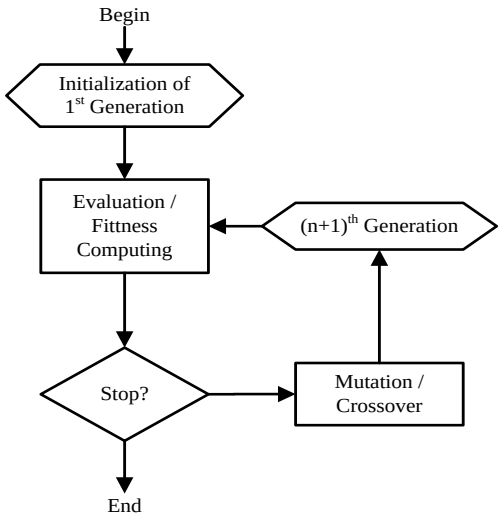
۳. تکرار مراحل زیر تا زمان برقراری شرایط خاتمه(Termination Condition)

۳.۱. انتخاب از افراد برای تولید مجدد(Reproduction)

۳.۲. تولید جمعیت های جدید با استفاده از جهش(Mutation) و ترکیب(Crossover) افراد انتخاب شده در مرحله ی قبل.

۳.۳. ارزیابی افراد با استفاده از تابع برازندگی(Fitness Function).

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



تصویر ۳؛ نمایش الگوریتمی ژنتیک

۲. مروری بر کارهای انجام شده

احتمالا اولین سعی برای استفاده از الگوریتم های تکاملی برای بهینه سازی مدارها مربوط است به Fridman که مربوط به دهه ی ۱۹۵۰ است. در رساله Fridman سعی شده است از ابزار هایی مشابه آنچه که امروزه آن را شبکه های عصبی می نامیم، برای بهینه سازی مدار مورد نظر استفاده شده است[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[۷]" برای بدست آوردن توابع منطقی استفاده کرده است. اما تاکید وی بیشتر بر بدست آوردن جواب و طراحی بوده است، نه بهینه سازی آن. در ادامه تلاش های صورت گرفته اغلب با استفاده از الگوریتم ژنتیک و با تاکید بر ایجاد متد های کد گذاری جدید و در نظر گرفتن محدودیت های مختلف بوده است. نمونه هایی از این دست در [6-7] دیده می شود. Coello در [1,8] با در نظر گرفتن محدودیت های تعداد سطوح طراحی، از الگوریتم ژنتیک استفاده کرد.

7 Genetic Programming

در این مقاله روش معرفی شده (NGA[8]) با مثال هایی با نتایج بدست آمده از روش های متداول Quin-McClusky و جدول کارنو و همچنین سایر روش های قبلی مقایسه شده است و کارایی بهتر آن نشان داده شده است.در این مقاله از گیت های NOT,XOR,OR,AND به عنوان گیت های پایه استفاده شده است. در ادامه Coello الگوریتم بهینه تری تحت عنوان MGA[9] معرفی کرده است که در آن تعداد گیت های منتج کمتر شده ولی تعداد ترانزیستور های استفاده شده در آن بیشتر است[9-10].

در ادامه در [13] در روش Coello تغییراتی را با نام Modified Evolutionary Algorithm ایجاد شده و نشان داده شده است که نتایج بهتری نسبت به MGA و NGA گرفته شده است.

در [10] ترکیبی از الگوریتم ژنتیک و SA (Simulated Anealing) و ادعا شده است که نتیجه ی استفاده ی ترکیبی از این دو الگوریتم، نتیجه ی بهتری را بدست می دهد. Coello در [12] نیز مقایسه ای بین روش های مختلف پیاده سازی ترکیبی SA-GA انجام داده است. در [14] نیز روشی مشابه MGA و NGA معرفی شده است.

در [15] در طراحی مدار منطقی، تعداد ترانزیستورهای مورد استفاده در گیت ها بیشتر مورد نظر گرفته است.

ساختار های طراحی گرافیکی همچون جدول کارنو[10] برای حالاتی خاص که طراحی دوسطحی مدارات مورد نظر است، بصورت گسترده کاربرد دارد. جدول کارنو تنها توانایی بهینه سازی را برای توابعی حداکثر تا ۶ متغیر دارد. (متغیر های با تعداد بالاتر دارای پیچیدگی بسیاری است.)

الگوریتم های محاسباتی همچون روش Quine-McCluskey در برنامه های آنالیز مدار همچون Expresso و MisII بطور گسترده مورد استفاده قرار می گیرند. این الگوریتم توانایی بهینه سازی و بدست آوردن مدار دو سطحی با هر تعداد متغیر را داراست. اما با توجه به پیچیدگی محاسباتی نمایی نسبت به تعداد متغیر های ورودی این الگوریتم برای یافتن minterm ها، هزینه ی محاسباتی آن بالاست. علاوه بر آن، پس از یافتن minterm ها، باید درمیان مجموعه ای توابع ساده شده از minterm ها جستجو کرد تا کل جدول حقیقت با کمترین هزینه پوشانده شود که خود مساله ای دارای پیچیدگی محاسباتی غیرچند جمله ای(Non-polynomial) است.

۳. بحثی بر شاخص های طراحی

هدفنهایی مساله بدست آوردن طراحی از مدارات منطقی ترکیبی(با مشخص بودن محدوده ی انواع آن) با داشتن جدول درستی آنها، بطوریکه بر اساس معیاری های موجود، آن را جوابی نزدیک بهینه نامید. معیار های متفاوتی می توان برای تشخیص بهتر بودن جواب به کار رود. برخی از مهم ترین موارد بدین صورت هستند:

- **تعداد گیت های بکار رفته:** قطعا هرچه تعداد گیت های بکار رفته کمتر باشد، بهتر است. چراکه باعث صرفه جویی در هزینه و فضا خواهد شد.
- **نوع گیت های بکار رفته**: با در نظر گرفتن ساختار داخلی گیت ها، می توان به برخی از آنها در مقایسه با سایر گیت ها برتری هایی نسبت داد. برای مثال در ساخت گیت های NOR و NAND از ۴ ترانزیستور و در ساخت گیت های

8N-cardinality Genetic Algorithm
9 Multi-Objective Genetic Algorithm
10Karnaugh Map

AND و OR به طور معمول از ۸ ترانزیستور استفاده شده است[1]. لذا استفاده از گیت های NAND و NOR موجب کاهش ترانزیستور های مورد استفاده شده و بهینه تر است.

- **تعداد سطوح طراحی مدار:** هرچه تعداد سطوح مدار طراحی کمتر باشد، میزان تاخیر ایجاد شده در بدست آوردن جواب نیز کمتر خواهد بود. همانطور که گفته شده روش های متداول از جمله روش Quine-McClusky و جدول کارنو، مدارات بهنیه دارای دو سطح را، با شرط کم بودن تعداد ورودی ها بدست می دهند. معمولا می توان از تاخیر ایجاد شده در انتشار سیگنال برای وجود بیش از چندین طبقه برای مثال تا ۴ یا ۵ سطح دیگر، به شرط بهینه تر شدن تعداد گیت های منطقی صرف نظر کرد.
- **پیچیدگی الگوریتم:** پیچیدگی الگوریتم ارائه شده، مساله ی بسیار مهمی است. چرا که برخی از الگوریتم ها اگرچه در تئوری امکان پذیر باشند، اما نیازمند پیاده سازی عملی هستند تا میزان عملی بودن آنها در زمان معقولی بدست آید. برای مثال اگرچه روش ارائه شده توسط Quine-McClusky در تئوری نشان می دهد که به ازای هر مداری می توان مدار بهینه ی دو سطحی آن را بدست آورد، اما پیچیدگی محاسباتی آن نمایی است. لذا پیاده سازی کامپیوتری آن به هیچ وجه مناسب نیست.

می توان با مثال هایی نشان داد که با پررنگ کردن وزن هر کدام از شرط های بهینه سازی ذکر شده، برای یک جدول حقیقت خاص، مدار می تواند دارای جواب های متفاوتی باشد. لذا هدف آنچه که مدار بهینه را مشخص خواهد کرد، نیاز کاربر و اهمیت نسبی است که وی در طراحی برای خود در نظر گرفته است.

در روش استفاده شده باید توجه داشت که نوع گیت های بکار رفته، باید در کل مجموعه ی کاملی را تشکیل دهند. مجموعه های کامل از گیت ها عبارتند از:

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

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

زیاد بودن تعداد گیت ها، مزیت این را دارد که این امکان را که بتوان با تعداد گیت های کمتری به جواب بهینه تر رسید را افزایش می دهد. در عین حال به ازای افزایش هر گیت به مجموعه ی گیت های مورد استفاده، فضای جستجوی شامل مدارها، تقریبا دو برابر می شود. در واقع مزیت استفاده از مجموعه های شماره ی ۵ و ۶ این است که این مجموعه ها به همراه گیت NULL دو حالت را تشکیل می دهند. لذا فضای جستجو در آن نسبت به حالت۱ بسیار کوچک است. در عین حال که جواب های بدست آمده از آنها، دارای اندازه ی بزرگی هستند.

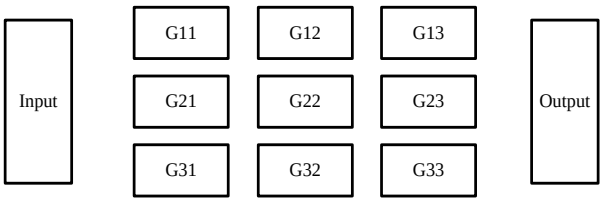
می توان در پیاده سازی، ترکیبی از مجموعه های کامل فوق را در نظر گرفت، برای مثال مجموعه ی {AND,XOR,OR,NAND} می تواند مجموعه ای باشد که بتوان اکثر مدارات را به حالت نزدیک بهینه پیاده سازی کرد. در [21] از مجموعه ی {XOR,NOT,NOR,NAND} برای طراحی مدارات استفاده شده است و با مثال هایی نشان داده شده است که نسبت به حالت {AND,OR,NOT} نتیجه ی بهتری را ارائه می دهد. در [1] با روشی تحت عنوان NGA (N-Cardinal Genetic Algorithm) از مجموعه گیت های {XOR,NOT,OR,AND} به عنوان گیت های پایه استفاده شده است.

۴. روش پیشنهادی

در ادامه دو روش پیشنهاد شده اند. در هر روش مرحله به مرحله، قسمت های مختلف الگوریتم و روش های استفاده شده توضیح داده می شوند.

۴.۱. روش اول

- **نمایش کد شده ی مدار**: ما برای کد کردن گیت ها، ازماتریس دوبعدی بصورت شکل زیر استفاده می کنیم:



تصویر۴؛ساختار ماتریس گیت ها

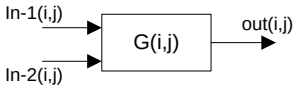
می توان هر مدار منطی را با مشخصات زیر تعیین کرد:

۱) مکان گیت ($G_{i,j}$)

۲) مکان ورودی اول و دوم آن ($In-1_{i,j}$ و $In-2_{i,j}$)

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

در واقع در این ساختار هر گیت بطور ثابت دارای دو ورودی هستند که آنها را از خروجی گیت های منطقی سطوح قبلی دریافت خواهد کرد.



تصویر۵؛ ساختار هر گیت در کدینگ نوع اول

در ساختاری که توسط Louis و Miller و Carlos [1,4,8,20] مورد استفاده قرار گرفته است در ماتریس گیت ها، هر کدام از گیت ها وروردی خود را فقط از طبقه ی ماقبل خود دریافت می کنند.

- **تولید جمعیت اولیه:** جمعیت اولیه شامل تعدادی از ساختار های ماتریسی گیت هاست که بطور تصادفی در سراسر فضا تولید می شوند. ویژگی تصادفی بودن جمعیت اولیه باعث خواهد شد تا پراکندگی فرد ها در فضای جستجو بطور یکنواخت صورت گیرد و حرکت به سمت جواب بصورتی یکنواخت انجام گیرد.
- **تابع برازندگی**: با توجه به معیار های برتری که در قسمت های قبل ذکر شد، ما معیار بهتر بودن را یک امید ریاضی (Expectation) از میزان تطابق خروجی مدار با جدول درستی و کم بودن تعداد گیت ها در نظر می گیریم.

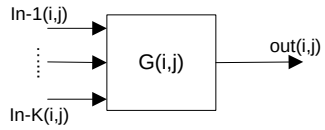
$$f = \frac{w_{match} \times N_{match} + w_{null} \times N_{null}}{w_{match} + w_{null}}$$

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

- **عملگر ژنتیکی Crossover**: 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 پیاده سازی عملی، دو مدار بطور تصادفی انتخاب شده و با انتخاب تصادفی اندیس **j**، سطح **j** ام آن مدار با یک مدار دیگر تعویض می شود.
- **عملگر ژنتیکی Mutation**: در این عملگر، یک گیت خاص از ماتریس گیت ها، انتخاب می شود و ویژگی های آن بطور تصادفی عوض می شوند(ورودی ها و نوع گیت ها).

۴.۲. روش دوم

- **نمایش کد شده ی مدار:** در این روش نیز، از ساختار ماتریسی مشابه آنچه که در روش اول بکار رفته، استفاده شده است. با این تفاوت که در آن به جای گیت های دو ورودی از گیت های چند ورودی استفاده شده است و همچنین هر گیت تنها می تواند ورودی خود را از گیت های سطر ما قبل خود دریافت کند. ساختار هر گیت در این روش به اینصورت است:



تصویر۶؛ 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 گیت های NULL، AND، OR، NAND، NOR و XOR که با عددی بین ۰ تا ۵ مشخص می شوند.

۳) رشته ای از اعداد صفر و یک که نشان دهنده ی اتصال یا عدم اتصال خروجی های طبقه ی قبل، به ورودی گیت است.

- **تولید جمعیت اولیه**: در این روش جمعیت اولیه به صورتی تصادفی تولید می شود. (به دلایل ذکر شده در روش قبل)

- **تابع برازندگی:** با توجه به دلایل ذکر شده در روش قبل، تابع برازندگی مورد استفاده در این روش به این صورت است:

$$f = \frac{w_{match} \times N_{match} + w_{null} \times N_{null}}{w_{match} + w_{null}}$$

- **عملگر ژنتیکی Crossover**: این عملگر، با انتخاب اندیس تصادفی j و دو مدار تصادفی، مدارات دو سمت اندیس را دو بدو به هم متصل کرده و مدار جدیدی را بدست می دهد.
- **عملگر ژنتیکی Mutation**: این عملگر از جدول گیت ها، خانه ای به طور تصادفی انتخاب کرده و مقادیر آن را به طور تصادفی عوض می کند.

۵. مقایسه ی نتایج

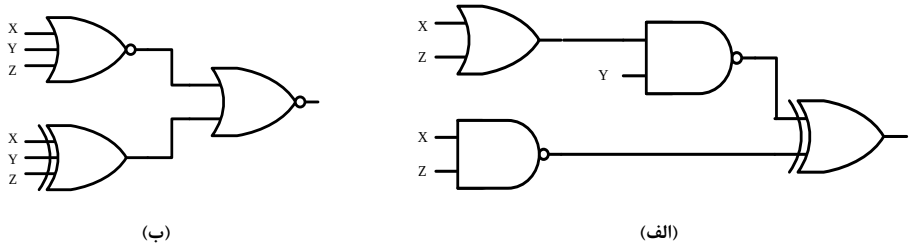
با توجه به اینکه گیت های سازنده ی روش پیشنهادی دوم می توانند گیت های چند ورودی(بزرگتر از ۲) باشند، شاید مقایسه ی نتایج این الگوریتم با نتیجه ی الگوریتم ها چندان درست نباشد. با این حال ما در ادامه با ذکر مثال هایی که همگی آنها، توسط گذشتگان آزموده شده اند، نتایج همه ی آنها را در جدول می آوریم تا مقایسه ای بین آنها داشته باشیم. باید متذکر شد که تمامی توابعی که در اینجا روی آنها کار می کنیم، توابع بدون Don’t Care هستند. پیاده سازی توابع همراه با Don’t Care بسیار ساده تر است. چرا که در عملیات محاسبه ی Fitness برای هرکدام از افراد، دیگر احتیاجی به محاسبه ی مقدار خروجی به ازای ورودی های Don’t Care نیست.

۵.۱. آزمایش اول: تابع سه ورودی بدون Don’t Care

تابع f_1 را با minterm های آن به صورت زیر تعریف می کنیم:

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

در ذیل نتایج بدست آمده برای این تابع توسط روش های مختلف را آورده ایم:

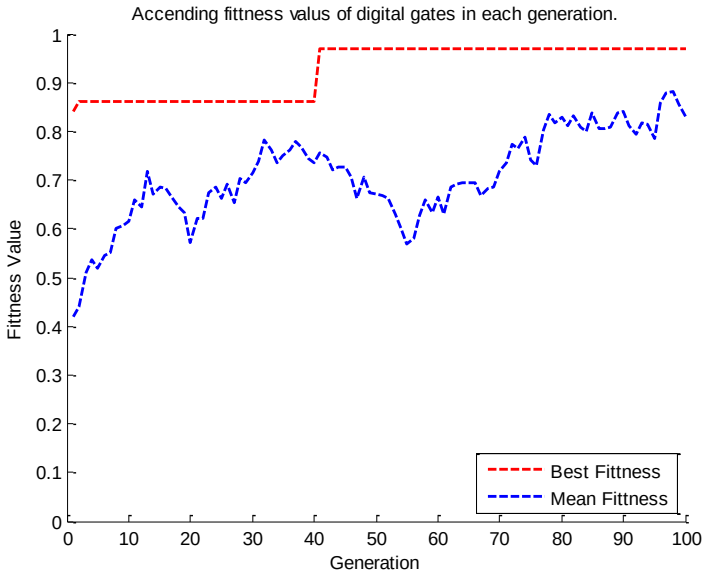


تصویر۷ ؛ مدار بدست آمده برای تابع f_1 توسط الگوریتم (الف) روش اول پیشنهاد شده (ب) روش دوم پیشنهاد شده

جدول ؛ مقایسه ی نتایج بدست آمده از پیاده سازی های 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 (۱ XOR(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، 1 OR، ۲ XOR)	۳

با توجه به جدول فوق دیده می شود که جواب روش اول با وجود تعداد گیت های یکسان و تعداد سطوح برابر با روش NGA، دارای گیت های NAND به جای AND است. لذا در کل جواب روش اول، نسبت به جواب روش دوم دارای ترانزیستور های کمتری است. همچنین با جدول، در جواب روش دوم با افزایش تعداد ورودی گیت ها، سطح و تعداد گیت ها، کاهش یافته است.

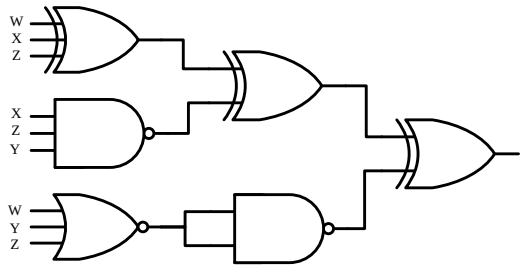


تصویر۸ ؛ نمودار همگرایی در روش شماره ۲ برای تابع f_1

۵.۲. تابع ۴ متغیر بدون Don’t Care

تابع f2 را با minterm های آن به صورت زیر تعریف می کنیم:

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



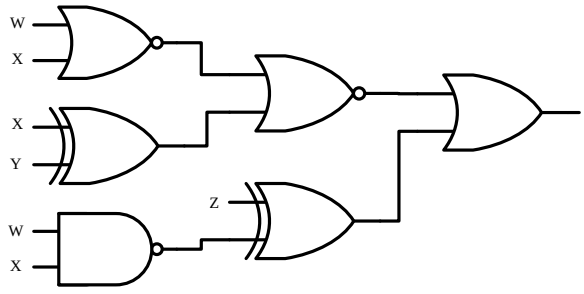
تصویر۹ ؛ مدار بدست آمده برای تابع f_2 توسط الگوریتم روش دوم ارائه شده

جدول ؛ مقایسه ی نتایج بدست آمده از پیاده سازی های مختلف

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**[۱۱]	$[\,(X\oplus(XY)\,)\oplus(\,(W+Y+Z)\oplus W\,)\,]'$	۷ گیت (۱ AND، ۲OR، ۳XOR، ۱ NOT)	۵
روش **NGA** [۱۱]	$[XY'Z\oplus[\,(X+Z)\oplus W\oplus((Y+Z)+W)\,]]'$	۱۰ گیت (۲ AND، ۳ OR، ۳ XOR، ۲ NOT)	۶
روش **BGA** [۱۲]	$W\oplus\big((\,X\oplus Z)\oplus(YZ)\big)\oplus(W+Y+Z)'$	۸گیت (۱ AND، ۳OR، ۳XOR، ۱ NOT)	۵
روش **Saso** [۱۱]	$Y'\oplus Z'X'\oplus YZ'W'\oplus Y'Z'X$	۱۲ گیت (۳ XOR، ۲ AND، ۴ NOT، ۳ NOR)	۴
روش عسگریان	$((Y+Z)'+(W\oplus Z))'\oplus((YZ)'.X)'$	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

۵.۳. تابع ۴ متغیر بدون Don’t Care

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



تصویر ۱۰؛ مدار بدست آمده برای تابع f_3 توسط الگوریتم روش اول ارائه شده

[۱۱] 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.

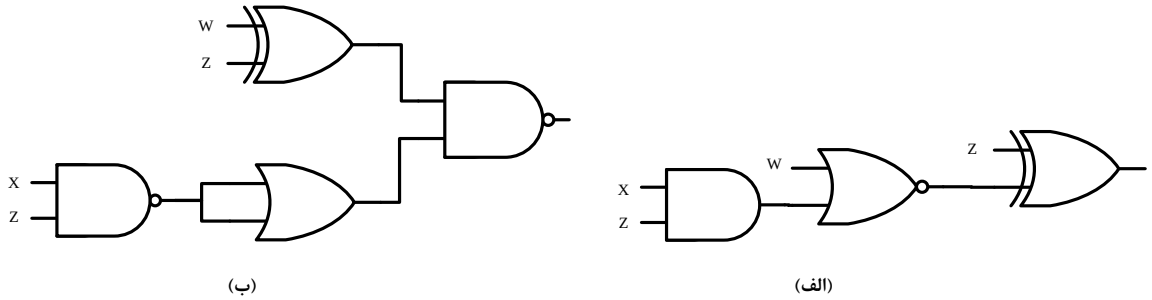
[۱۲] نسبت به مقاله ی اصلی، ترتیب متغیر ها به صورت W,X,Y,Z اصلاح شده است.

جدول ؛ مقایسه ی نتایج بدست آمده از پیاده سازی های 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، ۱ NAND، 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

۵.۴. تابع ۴ متغیر بدون Don't Care

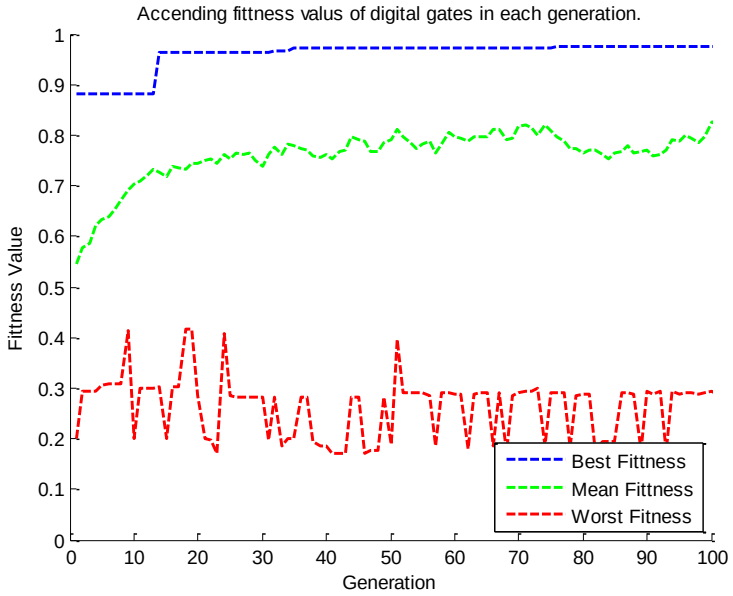
$$f_4(W,X,Y,Z)=\sum(1,0,1,0,1,1,1,1,1,0,1,0,1,0,1,01)$$



(ب) (الف)

تصویر ۱۱؛ مدار بدست آمده برای تابع f_4 توسط الگوریتم (الف)روش اول ارائه شده (ب)روش دوم ارائه شده

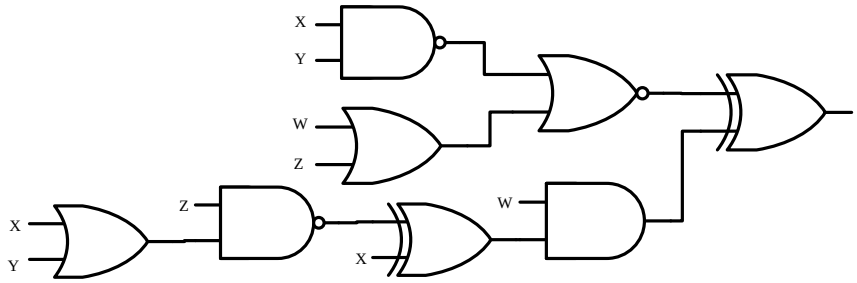
مشاهده می شود که هیچ کدام از مدارات بدست آمده تابع ورودی Y نیستند.



تصویر۱۲ ؛نمایش همگرایی الگوریتم به جواب برای تابع f_4 با روش اول

۵.۵. تابع ۴ متغیر بدون Don't Care

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



تصویر ۱۳؛ مدار بدست آمده برای تابع f_5 توسط الگوریتم روش اول ارائه شده

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

در اکثر موارد، برنامه های پیاده سازی شده، به جواب هایی، با اختلاف ۱یا ۲ خروجی با جدول حقیقت خواسته شده می رسد. در واقع جواب های مورد نظر در این حالت به Local Minimum رسیده و در آن به دام می افتد.

یکی از مهم ترین دلایلی که به نظر نویسندگان باعث ایجاد این مشکل می شود، عدم توانایی توابع Fitness در پیش بینی فاصله ی نسبی با مدار معتبر است. در واقع راه رسیدن به یک مدار معتبر الزاما از مسیر یک مدار با اختلاف کمتر با جدول حقیقت نمی گذرد.

برای فایق آمدن بر این مشکل روش های متعددی استفاده شده که بطور خلاصه ذکر می گردد:

- **استفاده از روش های انتخاب (Selection) مختلف:** استفاده از روش انتخاب Roulette باعث می شود، اعضای نسل بعدی به صورتی پراکنده در نسل های آینده پخش باشند. لذا امکان رسیدن به مدار معتبر را فراهم می کند. البته به شرطی مفید واقع می شود، که fitness در مراحل بعدی جواب را به یک مدار معتبر راهنمایی کند. در غیر اینصورت باید آنقدر منتظر بود تا مدار معتبر بصورت تصادفی ایجاد شود.
- **تقسیم جدول حقیقت:** در صورتی که تشخیص داده شود که جواب در تعداد generation_thereshold (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. اینگونه مدار بدست آمده ترکیبی را وارد جمعیت جدیدی کرده و سعی در بهینه سازی 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 داشته اند. اگرچه با آزمایش پیشنهادات این افراد، متاسفانه مشکل گیر افتادن در بهینه ی محلی برطرف نشد. به نظر نویسندگان، بهینه سازی های انجام شده، توسط سایر مقالات، به هیچ وجه کارآمد نیست. در واقع در

تمامی مقالات با عناوینی همچون Multi-Objective Optimization توابع fitness مطرح شده، اگرچه میزان درست بودن جواب را نشان می دهد، اما به هیچ وجه فاصله ای نسبی را به سمت جوابی معتبر بیان نمی کند. لذا ظن این می رود، که جواب های بدست آمده درمقالات، اکثرا تصادفی و با ایجاد Generation و Population های بالا بدست آمده باشد.(چنان که در مقالات به این مساله اشاره شده است.) باید تاکید کرد که بدست آوردن جواب هایی درست با Generation و Population بالا، از نظر نویسندگان به هیچ وجه قابل قبول نیست. چرا که با داشتن هزینه ی محاسباتی بالا و نیاز به زمان بالا برای محاسبات، کارآمدی در مقابل روش های موجود ندارد. چرا که در واقع در این حالت برنامه از ویژگی های الگوریتم ژنتیک، یعنی سوق دادن جواب به سمت بهترین جواب، با استفاده از تابع fitness، استفاده نمی کند. و این یعنی اینکه در این حالت، آنچه انجام می شود، تفاوت چندانی با جستجوی کورکورانه در فضای جستجو نخواهد داشت.

۷. نتیجه گیری و کار آینده

در این نوشته، علاوه بر معرفی کلی الگوریتم زنتیک، از آن در بهینه سازی یک مدار ترکیبی استفاده شد. با توجه به مطالب گفته شده، دیده شد که نحوه ی کدینگ مسئله، می تواند تاثیر زیادی در بدست آوردن جواب داشته باشد. در واقع پایداری راه حل ها، در بدست آوردن جواب، لغزنده است. بدین معنی که تغییر کوچکی نحوه ی اجرای الگوریتم ها و در نسبت اجرای آنها، می تواند باعث بدست آمدن نتایج جدیدتری شود. با توجه به دو کدینگ که در این گزارش ارائه کردیم، نتایج آنها با برخی از مدارات سابقا بدست آمده مقایسه شد.

لذا در طراحی مدارات منطقی با استفاده از الگوریتم تکاملی بطور کلی دو نقص روش های متداول برطرف می شود:

۱- طراحی مدارات با چندین سطح بیشتر برای بدست آوردن طراحی بهینه تر.

۲- بهینه سازی چندین تابع بطور همزمان (بر روی یک ماتریس یکسان).

باید توجه کرد که با توجه به تمامی روش های نوین ارائه شده برای طراحی مدارات ترکیبی، هنوز مهمترین اشکال طراحی های سنتی(نظیر جدول کارنو و الگوریتم Quine-McClusky) برطرف نشده است و آن طراحی مدارات با تعداد ورودی بالاست. محدودیت های پردازش و حافظه لازم دارند تا برای متغیر های با تعداد بالا، روش هایی خارج از حدود عادت(Heuristic) ابداع و با روش های مذکور همراه شوند. چنین روش هایی بسیار احتیاج به کار و خلاقیت دارند. بهینه سازی روی طراحی مدارات منطقی ترتیبی که دارای عناصری مثل فیدبک یا عناصر حافظه اند، می تواند جالب باشد. متد مورد توجه دیگر که از لحاظ نحوه ی کدینگ و روش کار بسیار با روش های معرفی شده تفاوت دارد، استفاده از Genetic Programming برای بهینه سازی مدارات منطقی است. بدین صورت که می توان به جای بهینه سازی گیت ها و ورودی های آنها، یک عبارت ریاضی(منطقی) شامل عملگرهای منطقی $\{\downarrow,\uparrow,\odot,\oplus,+,.\}$ و متغیر های آنها را بهینه سازی کرد. همچنین استفاده از ساختار درختی به جای ساختار ماتریسی و کدینگ باینری گیت ها و ورودی های آنها، می توان موجب بهینه سازی هزینه در زمان و پردازش شود. استفاده از روش های دیگر بهینه سازی از جمله Ant-Colony، Particle Swarm Optimization و ... برای این کار نیز مورد توجه می باشد. نمونه هایی از این پیاده سازی ها در [17-19] انجام گرفته شده است.

تقدیر و تشکر

نویسندگان از آقای احسان امیدی (دانشجوی ورودی ۸۶ مهندسی برق) به خاطر کمک شایان ایشان در پیاده سازی نمایشگر مدارات منطقی، تشکر و قدردانی می نمایند.

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-A. H. Aguirre, C. A. C. Coello, "Using genetic programming and multiplexers for the synthesis of logic circuits", Engineering Optimization, 2004.

4- Sushil J. Louis, Gregory J.E. Rawlins: "Designer Genetic Algorithms: Genetic algorithms in StructureDesign" Proceedings 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", Proceedings of the ICCI'2004 – International Conference 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 Conference 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- Carlos A. Coello Coello and Arturo Hernández Aguirre , "Design of Combinational Logic Circuits through an Evolutionary Multi-objective Optimization Approach", Artificial Intelligence for Engineering, Design, Analysis and Manufacture, 16(1), pp. 39-53, January 2002.

10- Carlos A. Coello Coello and Arturo Hernández Aguirre , "Design of Combinational Logic Circuits through an Evolutionary Multi-objective Optimization Approach", 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 Conference 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" Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware.

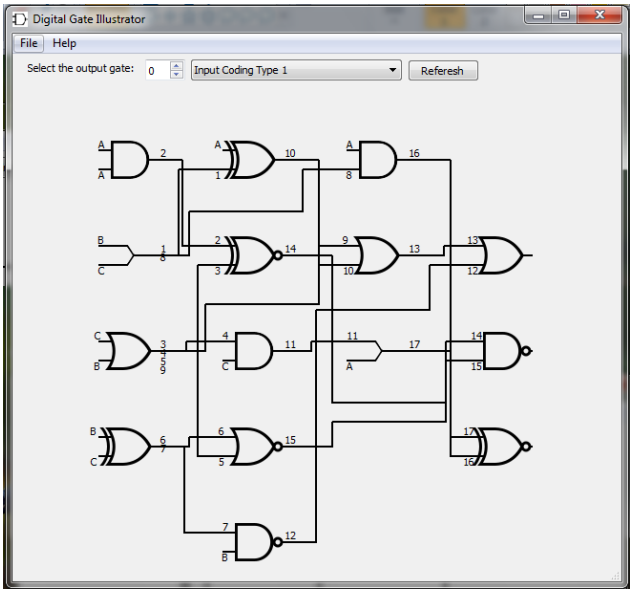
13- A. Słowik, M Białko, "Design and Optimization of Combinational Digital Circuits Using Modified Evolutionary Algorithm", Proceedings of 7 th International Conference on Artificial Intelligence and Soft Computing, Lecture Notes in Artificial Intelligence, 2004.

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”, Proceedings of the 9th annual conference 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.

ضمیمه۱

نرم افزار ارائه کننده ی مدارات منطقی ترکیبی(Digital Gate Illustrator)

همانطور که در متن گزارش اشاره شد، استفاده از امکانات بهینه سازی الگوریتم ژنتیک تنها در صورتی امکان پذیر است که مساله ی مورد نظر به صورتی مناسب تبدیل به داده های ورودی الگوریتم ژنتیک شوند. در واقع در الگوریتم ژنتیک به جای اینکه با مقاهیمی به نام گیت و اتصال آنها سر و کار داشته باشیم، با معادل کد شده ی آنها کار می کنیم. طبیعتا خروجی چنین الگوریتمی نیز مجموعه ای از اعداد خواهد بود که نشان دهنده ی کیفیت قرار گیری گیتها و اتصال ها خواهد بود. کار کردن با داده های کد شده و اعداد برای درک درستی راه حل و بدست آوردن شهود سریع و مناسب از جواب مساله، به هیچ وجه مناسب نیست. لذا در ادامه ی طراحی الگوریتم بهینه سازی مدارات منطقی، لازم دیدیم تا نرم افزاری جهت نشان دادن خروجی بهینه شده طراحی و از آن استفاده کنیم.



تصویر ۱۴؛ نمونه از نمایش مدار خروجی.

در این تصویر A و B و C ورودی های مدار هستند. خروجی را می توان هرکدام از گیت های طبقه ی آخر در نظر گرفت.

در واقع بهینه سازی مدارات در نرم افزار Matlab انجام می گیرند و خروجی به صورت فایل در شاخه ی نرم افزار ارائه کننده ذخیره می شوند. با اجرای نرم افزار ارائه کنند و یا با فشار دادن دکمه ی Refresh، داده های فایل داده ها، پردازش و در خروجی نمایش داده می شوند.

ساختار فایل ورودی نرم افزار به صورت زیر است. تمامی سطر هایی که با # شروع می شوند، خطوط اضافی(Comment) هستند و نشان دهنده ی توضیحات اضافی می باشند. هر سطر نشان دهنده ی اطلاعات مربوط به یک گیت خاص است. ترتیب اعداد در

هر سطر، در سطر دوم توضیحات اضافی آمده است. آخرین عدد در هر سطر نشان دهنده ی نوع گیت است که روش کدینگ گیت ها، در سطر اول توضیحات اضافی آمده است. برای مثال اولین سطر از اعداد نشان دهنده ی گیت AND در مکان (۱و۱) است که هر دو ورودی اش را از A می گیرد. سطر دوم نشان دهنده ی گیت XOR است که ورودی اش را از A و خروجی گیت در ستون اول و سطر دوم می گیرد. این دو گیت در شکل بالا نیز نمایش داده شده اند.

```
# Gate Types: 0:NULL 1:NOT 2:AND 3:OR 4:NAND 5:NOR 6:XOR 7:XNOR
# gate_x, gate_y, input1_x, input1_y, input2_x, input2_y, gate_type
1 1 0 1 0 1 2
2 1 0 1 1 2 6
3 3 2 3 0 1 0
4 2 3 2 2 5 3
```

همانطور که در متن اصلی اشاره شده است در پیاده سازی از گیت هایی مجازی (NULL-gate) استفاده شده است که هدف از آنها، ایجاد اتصال مستقیم بین ورودی و خروجی است. با توجه به ساختار ماتریسی لازم بود که ساختار دو ورودی برای آن در نظر گرفته شود. بطور منطقی، می توان خروجی آن را بر اساس طراحی الگوریتم ، بر اساس یکی از ورودی های اول یا دوم در نظر گرفت.

باید توجه کرد که در این ساختار می توان هرکدام از گیت های لایه ی آخر را به عنوان خروجی مدار در نظر گرفت. در عین حال که می توان با در نظر گرفتن چندین خروجی، عملیات بهینه سازی را همزمان روی چندین تابع انجام داد. لذا بر کاربر نرم افزار است تا آنگونه که به آن نیازمند است، از آن بهره گیرد.

طراحی نرم افزار مذکور با قدرت Qt4 صورت گرفته است که بر اساس زبان برنامه نویسی C++ شکل گرفته است. با توجه توانمندی ابزار توسعه ی Qt، نرم افزار ارائه کننده ی مدارات منطقی، دارای هر سه نسخه ی ویندوز، لینوکس و مکینتاش است.

با توجه به لیسانس استفاده از ابزار توسعه ی Qt، این نرم افزار بعد از اتمام پروژه، به صورت آزاد و تحت لیسانس GPL[۱۳]منتشر خواهد شد. در واقع این یکی دیگر از اهداف دیگر جانبی این پروژه است که با توسعه ی این نرم افزار، زمینه ی پیشرفت های آتی، با استفاده از مستندات و ابزار های ایجاد شده توسط محققان قبلی، در اختیار افراد علاقمند بعدی قرار گیرد [۱۴] تا بتوانند با تمرکز بیشتر روی اساس الگوریتم ها، به توسعه و ارتقای روش ها بپردازند. با توجه به انتشار این نرم افزار تحت لیسانس GPL از تمامی محققان دعوت می گردد که با ارتقای این نرم افزار و استفاده از آن، به شرط انتشار آزاد، به پیشرفت و ارتقای آن کمک کنند.

13 General Public License

[۱۴] برای دریافت آخرین نسخه از نرم افزار به سایت زیر مراجعه کنید:

http://ele.aut.ac.ir/~khashabi/wp/?p=82