



دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی برق

پایان نامه کارشناسی

گرایش الکترونیک

عنوان

طراحی یک سیستم توصیه‌گر منطق برای بهینه‌سازی مدارهای مجتمع

دیجیتال بر مبنای الگوریتم‌های هوش محاسباتی

**Design a Logic recommender system for optimization of digital
CMOS Integrated Circuits by means of computational
intelligence**

نگارش

سپند حقیقی

استاد راهنما

دکتر مجید شالچیان

۱۳۹۵ مهر

به نام خدا

تاریخ:

تعهدنامه اصالت اثر



اینجانب سپند حقیقی متعهد می‌شوم که مطالب مندرج در این پایان‌نامه حاصل کار پژوهشی اینجانب تحت نظرارت و راهنمایی استادید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع داده شده و در فهرست منابع و مأخذ ذکر گردیده است. این پایان‌نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان‌نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان‌نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مأخذ بلامانع است.

سپند حقیقی

نرم افزار عاطفه کدهای لذت و درد . . .

پ-۵

به یاد دکتر شهلا فرجاد آزاد

چکیده

در این پژوهه، یک برنامه کامپیوتری به منظور توصیه منطق^۱ مناسب برای پیاده‌سازی بهینه مدارهای مجتمع دیجیتال از نظر توان، تاخیر و مساحت طراحی شده است. این برنامه توصیف مدار را به صورت‌های مختلف (تابع بولی، زبان توصیف سخت‌افزار^۲ و ...) از کاربر گرفته و با پردازش و ارزیابی ویژگی‌های قسمت‌های مختلف مدار برای هر بخش از طرح منطق بهینه برای پیاده‌سازی آن بخش را پیشنهاد می‌دهد. برای هوشمندسازی این برنامه از روش‌های یادگیری ماشین^۳ استفاده شده است، به این صورت که این برنامه در ابتدا به یک سنترکننده مدارهای مجتمع متصل بوده و با استفاده از روش‌های یادگیری ناظرت‌شده^۴ و داده‌های ورودی آموزش می‌بیند. پس از اتمام آموزش این برنامه توانایی پیشنهاد منطق مناسب به ازای هریک از پارامترهای بهینه سازی(مساحت، توان و تاخیر) را به صورت جداگانه برای مدارهای جدید دارا می‌باشد. در این مرحله برنامه از سنترکننده جدا شده و به صورت مستقل عمل می‌کند و برای ارزیابی عملکرد آن از داده‌های آزمون(جدید) و ماتریس درهم‌ریختگی استفاده می‌شود.

واژه‌های کلیدی:

زبان توصیف سخت‌افزار (HDL)، یادگیری ماشین (Machine Learning)، سنتر
منطقی (Supervised Logic)، شبکه‌ی عصبی (NN)، یادگیری ناظرت شده (Learning

¹ Logic Recommender System

² HDL

³ Machine Learning

⁴ Supervised Learning

۱	چکیده.....
۲	۱ فصل اول مقدمه
۳	۱.۱ مراحل طراحی و ساخت مدارهای مجتمع دیجیتال [۱]
۴	۱.۱.۱ فاز توصیف سیستمی
۵	۱.۱.۲ فاز طراحی عملکرد
۶	۱.۲.۱.۱ طراحی و پیاده‌سازی رفتاری و الگوریتمی
۷	۱.۲.۱.۲ طراحی RTL
۸	۱.۲.۱.۳ طراحی ساختاری
۹	۱.۲.۱.۴ فاز طراحی منطقی
۱۰	۱.۲.۱.۵ فاز طراحی مداری
۱۱	۱.۲.۱.۶ فاز طراحی چینش
۱۲	۱.۲.۱.۷ فاز ساخت تراشه
۱۳	۲.۱ خانواده‌های منطقی مورد استفاده در طراحی دیجیتال
۱۴	۲.۱.۱ انواع منطق ایستا
۱۵	۲.۱.۲ منطق CMOS ایستا
۱۶	۲.۱.۳ منطق NMOS مقاومتی
۱۷	۲.۱.۴ منطق NMOS افزایشی
۱۸	۲.۱.۵ منطق شبه NMOS
۱۹	۲.۱.۶ منطق NMOS تخلیه‌ای
۲۰	۲.۱.۷ منطق ترانزیستورهای عبوری
۲۱	۲.۱.۸ منطق دروازه انتقال
۲۲	۲.۲ منطق های پویا
۲۳	۲.۲.۱ منطق CMOS پویا
۲۴	۲.۲.۲ منطق CMOS دامینو
۲۵	۲.۲.۳ منطق CMOS کلاک دار
۲۶	۲.۳ پیشینه روش‌های بهینه سازی
۲۷	۲ فصل دوم طراحی کتابخانه سلول‌های استاندارد
۲۸	۲.۱ رسم شماتیک و لی‌اوٹ سلول‌ها
۲۹	۲.۲ مشخصه یابی سلول‌های استاندارد و ایجاد کتابخانه‌های (.lib) و (.db)
۳۰	۲.۳ خلاصه
۳۱	۳ فصل سوم یادگیری ماشین و طراحی سیستم توصیه‌گر منطق
۳۲	۳.۱ یادگیری ماشین

۲۷	۱.۱.۳ یادگیری ناظارت شده
۲۷	۲.۱.۳ یادگیری بی ناظارت
۲۷	۳.۱.۳ یادگیری تقویتی
۲۸	۲.۳ شبکه های عصبی
۲۹	۱.۲.۳ شبکه عصبی پس انتشار
۲۹	۳.۲.۲ پرسپترون
۳۰	۳.۲.۲.۱ پرسپترون تک لایه
۳۱	۲.۲.۲.۳ پرسپترون چندلایه دودویی
۳۲	۳.۲.۲.۳ پرسپترون چندلایه - چندطبقه
۳۳	۴.۲.۲.۳ آموزش پرسپترون
۳۴	۵.۲.۲.۳ شرط توقف آموزش
۳۶	۳.۳ سیستم توصیه گر منطق
۳۶	۱.۳.۳ توصیف سیستم
۴۰	۴ فصل چهارم پیاده سازی
۴۱	۱.۴ مازول ها و کتابخانه ها
۴۲	۲.۴ شی اصلی
۴۴	۳.۴ تجزیه کننده
۴۵	۴.۴ تولید کننده داده آموزش
۴۶	۵.۴ متده تولید فایل Verilog
۴۷	۶.۴ تولید اسکریپت برای سنتز کننده
۴۹	۷.۴ سایر توابع
۵۰	۸.۴ ارزیابی زمانی
۵۲	۹.۴ برنامه آموزش شبکه
۵۲	۱.۹.۴ نصب کتابخانه
۵۲	۲.۹.۴ ساخت شبکه عصبی
۵۳	۳.۹.۴ تعریف داده های ورودی
۵۳	۴.۹.۴ اعمال الگوریتم های یادگیری
۵۴	۵.۹.۴ توقف آموزش
۵۵	۵ فصل پنجم نتایج و جمع بندی
۵۶	۱.۵ ارزیابی سیستم
۶۱	۲.۵ جمع بندی و پیشنهادها
۶۲	۶ منابع و مأخذ
۶۴	[۸] پیوست ۱

۷۲.....	پیوست ۲
۷۴	پیوست ۳

صفحه	فهرست اشکال
۳	شکل ۱-۱ مراحل ساخت مدارهای مجتمع دیجیتال [۱]
۷	شکل ۲-۱ وارونگر طراحی شده در منطق CMOS ایستا
۸	شکل ۳-۱ وارونگر طراحی شده در منطق NMOS مقاومتی
۹	شکل ۴-۱ وارونگر طراحی شده در منطق NMOS افزایشی
۹	شکل ۵-۱ وارونگر طراحی شده در منطق شبه NMOS
۱۰	شکل ۶-۱ وارونگر طراحی شده در منطق NMOS تخلیه ای
۱۱	شکل ۷-۱ گیت AND طراحی شده در منطق ترانزیستور عبوری
۱۲	شکل ۸-۱ گیت MUX طراحی شده در منطق دروازه انتقال
۱۳	شکل ۹-۱ قالب کلی مدارات CMOS پویا
۱۳	شکل ۱۱-۱ قالب کلی مدارات CMOS دامینو
۱۴	شکل ۱۱-۱ قالب کلی مدارات CMOS کلاک دار
۱۸	شکل ۱-۲ مراحل طراحی کتابخانه [۸]
۲۰	شکل ۲-۲ شماتیک سلول های واحد [۸]
۲۱	شکل ۳-۲ لی اوت سلول های واحد [۸]
۲۲	شکل ۴-۲ شماتیک و لی اوت گیت AND/NAND [۸]
۲۳	شکل ۵-۲ نحوه تعیین مولفه های ماتریس دو بعدی حاصل از مشخصه یابی
۲۴	شکل ۶-۲ کتابخانه های ایجاد شده [۸]
۲۹	شکل ۱-۳ ساختار کلی یک شبکه عصبی پس انتشاری تمام متصل [۱۲]
۳۰	شکل ۲-۳ ساختار کلی یک پرسپترون تک لایه [۱۳]
۳۲	شکل ۳-۳ ساختار کلی یک پرسپترون چندلایه باینری [۱۵]
۳۲	شکل ۴-۳ ساختار کلی یک پرسپترون چندلایه - چندطبقه [۱۵]
۳۵	شکل ۵-۳ مقایسه عملکرد شبکه بر روی داده های آموزش و داده های تست [۱۶]
۳۷	شکل ۶-۳ دیاگرام کلی سیستم
۳۸	شکل ۷-۳ دیاگرام بلوکی بخش سنتز کننده و شبیه ساز
۳۹	شکل ۸-۳ دیاگرام بلوکی بخش تصمیم گیرنده
۴۳	شکل ۱-۴ فراخوانی شی اصلی
۴۴	شکل ۲-۴ قسمتی از یک جدول درستی ساخته شده
۴۵	شکل ۳-۴ نحوه انتخاب فایل و ذخیره سازی فایل های نمونه
۴۶	شکل ۴-۴ یک نمونه از فایل های تولید شده توسط ماژول generator.py
۴۷	شکل ۵-۴ نمونه ای از فایل Verilog تولید شده توسط make_verilog
۴۸	شکل ۶-۴ مراحل مختلف استفاده از Design Compiler [۱۷]

۴۹ شکل ۷-۴ نمونه ای از اسکریپت تولید شده توسط make_script_file
۵۱ شکل ۸-۴ نمونه ای از فایل های تولید شده توسط برنامه time_test.py
۵۲ شکل ۹-۴ روش clone و نصب کتابخانه [۱۸] pybrain
۵۲ شکل ۱۰-۴ روش ساخت شبکه در pybrain
۵۳ شکل ۱۱-۴ روش تعریف داده های ورودی در pybrain
۵۴ شکل ۱۲-۴ روش آموزش شبکه های عصبی در pybrain
۶۴ شکل پ-۱ مراحل تولید کتابخانه سلول های استاندارد. [۹]
۶۷ شکل پ-۲ ورودی ها و خروجی های NCX
۶۷ شکل پ-۳ محتوای فایل .opt در NCX
۶۸ شکل پ-۴ مسیر دسترسی به توان نشتی محاسبه شده.
۶۹ شکل پ-۵ محتویات فایل .opt .
۶۹ شکل پ-۶ محتویات فایل .indices
۷۰ شکل پ-۷ محیط برنامه NCX

صفحه	فهرست جداول
	جدول ۱-۲ نحوه ایجاد توابع مختلف کتابخانه از طریق سلول پایه [۸] ۱۹
	جدول ۲-۲ شرایط بکار رفته در رسم لی اوت سلول ها [۸] ۲۱
	جدول ۱-۵ مقایسه برخی از توابع پایه در منطق های GDI و CMOS ایستا ۵۶
	جدول ۲-۵ ماتریس درهم ریختگی سیستم برای پارامتر توان ۵۷
	جدول ۳-۵ ماتریس درهم ریختگی سیستم برای پارامتر مساحت ۵۷
	جدول ۴-۵ ماتریس درهم ریختگی سیستم برای پارامتر تاخیر ۵۸
	جدول پ-۱ اطلاعات موجود در فایل .lib ۷۲
	جدول پ-۲ قطعه کد globals.py ۷۴
	جدول پ-۳ تابع functions.py در make_script_files ۷۴
	جدول پ-۴ سایر توابع مازول functions.py ۷۵
	جدول پ-۵ تابع مازول generator.py ۷۶
	جدول پ-۶ تابع مازول logic.py ۷۸
	جدول پ-۷ تابع مازول parser.py ۷۹
	جدول پ-۸ تابع مازول test.py ۸۱
	جدول پ-۹ متدهای عمومی کلاس VLSI ۸۳
	جدول پ-۱۱ متدهای ساخت فایل Verilog کلاس VLSI ۸۸

فهرست اختصارات

LVS	Logic Vs. Schematic
DRC	Design Rule Checking
SA	Simulated Annealing
TG	Transmission Gate
PTL	Pass Transistor Logic
RTL	Register Transfer Level
MSE	Mean Squared Error
NN	Neural Network
CMOS	Complementary Metal Oxide Semiconductor
LRS	Logical Recommender System
HDL	Hardware Description Language
SLP	Single Layer Perceptron
MLP	Multilayer Perceptron
SVM	Support Vector Machine

۱

فصل اول

مقدمه

مقدمه

امروزه با پیشرفت تکنولوژی ساخت نیمه هادی‌ها و گسترش مدارات الکترونیکی کم مصرف، قابل حمل و پرسرعت طراحی مدارات مجتمع دیجیتال و مبدل داده از پیچیدگی و دشواری خاصی برخوردار شده است. از این روزت که در سال‌های اخیر تحقیقات زیادی در زمینه اتوماسیون طراحی این گونه مدارات صورت گرفته است تا با استفاده از ابزارهای هوشمند و توانمند مدت زمان اختصاص داده شده به طراحی تا زمان ساخت تراشه نهایی^۵، کاهش داده و به مداراتی با قابلیت اطمینان بالا، دقت و پایداری مطلوب دست یابیم.

تاکنون روش‌های بهینه سازی متعددی توسط طراحان به کار گرفته شده‌اند. بسیاری از ابزارهای طراحی که تا به امروز مورد استفاده طراحان قرار گرفته‌اند بر مبنای روش‌های گرادیان یا الگوریتم‌های ساده عمل می‌نمایند که برای طراحی پیشرفت‌های امروزی با مشخصات متنوع و بعضی در تعارض با یکدیگر مناسب نمی‌باشند. امروزه، توجه بسیاری از طراحان در زمینه‌های گوناگون مهندسی به استفاده از الگوریتم‌های غیر خطی محاسباتی و الگوریتم‌های ژنتیک چند جهته معطوف شده است.

به طور کلی طراحی مدارات مجتمع دیجیتال شامل دو مرحله است. مرحله اول مربوط به انتخاب توبولوژی مدار به گونه‌ای است که پتانسیل لازم جهت براورده ساختن مقتضیات طراحی را دارا باشد. مرحله‌ی دوم، که غالباً بخش عمده‌ای از زمان طراحی را به خود اختصاص می‌دهد، مربوط به تعیین پارامترهای طراحی مدار به عنوان مثال سایز ترانزیستورها و مقادیر عناصر می‌باشد به گونه‌ای که مشخصات مورد نظر طراح حاصل شود. در روش‌های طراحی کلاسیک که بر مبنای اصول تقریبی می‌باشند، یافتن مقادیر این پارامترها مستلزم انجام چندین مرحله سعی و خطأ^۶ و در نهایت صرف وقت قابل توجهی برای یافتن پاسخ مناسب مدار می‌باشد.

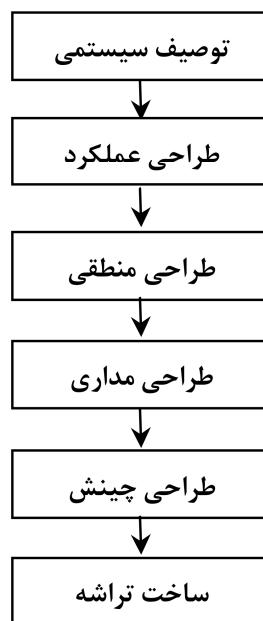
⁵ Time To Market

⁶ Trial And Error

در ادامه به شرح کوتاهی از مراحل مختلف ساخت مدارهای مجتمع دیجیتال و روش‌های بهینه سازی آن‌ها می‌پردازیم.

1.1 مراحل طراحی و ساخت مدارهای مجتمع دیجیتال [۱]

شکل زیر مراحل مختلف طراحی یک مدار مجتمع یا تراشه را نشان می‌دهد. در ادامه هریک از این مراحل شش‌گانه به اختصار تشریح می‌شوند.



شکل ۱-۱ مراحل ساخت مدارهای مجتمع دیجیتال [۱]

1.1.1 فاز تصویف سیستمی

در این فاز ابتدا نیازها، خصوصیات و محدودیت‌های طرح مورد نظر استخراج می‌گردد. سپس سعی می‌شود با در نظر گرفتن فاکتورهای مهم در بحث طراحی مانند سرعت پردازش، تاخیر

المانها، توان مصرفی تراشه، سطح مقطع گیت و ... یک ساختار کلی از سیستم در قالب زیرسیستم‌های مرتبط با هم ارائه شود و محدودیت‌های مذکور در آن لحاظ گردد.

۲.۱.۱ فاز طراحی عملکرد

زیرسیستم‌های شناسایی شده در مرحله توصیف سیستمی به صورت جزئی‌تر تشریح می‌گردد و گاه خود نیز به چندین مازول دیگر تقسیم می‌شوند. سپس هریک از این مازول‌ها به صورت جداگانه طراحی می‌شوند. در گام بعد ارتباط بین تمامی این مازول‌های مستقل به صورت مناسب و مطلوب برقرار می‌گردد.

در سیستم‌های دیجیتال، این فاز را می‌توان به چندین شکل و در سطوح مختلف انجام داد:

۱.۲.۱.۱ طراحی و پیاده‌سازی رفتاری و الگوریتمی

در این سطح با یک نگاه بسیار کلی، صرفاً به پیاده‌سازی رفتار سیستم توجه می‌شود. معمولاً این پیاده‌سازی جهت اطمینان از عملکرد سیستم طراحی شده به کار می‌رود و با استفاده از زبان‌های برنامه نویسی سطح بالا مثل C یا نرم‌افزاری همچون MATLAB انجام می‌گیرد.

۲.۲.۱.۱ ^۷ RTL طراحی

در این سطح سیستم به صورت مجموعه‌ای از ثبات‌ها، حافظه و گذرگاه‌های ارتباطی ترسیم و طراحی می‌گردد. به بیان دیگر در این سطح، سیستم به صورت یک جریان داده‌ای پیاده‌سازی می‌شود.

⁷ Register Transfer Level

۳.۲.۱.۱ طراحی ساختاری

در این سطح، سیستم به شکل مجموعه‌ای از بلاک‌ها ترسیم می‌گردد که ارتباطات میان آنها، چگونگی جریان داده را در کل سیستم نشان می‌دهد. این ساختار با همکاری طراحان سیستمی دیجیتالی پیاده‌سازی می‌گردد و درک این دو گروه را از سیستم یکسان می‌سازد.

معمولًا با ترکیب دو ساختار قبلی تلاش می‌شود تا به یک سیستم قابل درک پیاده‌سازی توسط یکی از زبان‌های توصیف سخت‌افزار (HDL) برسیم. در طراحی رفتاری نیز سیستم به صورت بلاک‌هایی طراحی می‌گردد که ساختار درونی هریک از آنها یک توصیف RTL باشد.

۳.۱.۱ فاز طراحی منطقی

این مرحله در مدارهای مجتمع آنالوگ وجود ندارد زیرا در این گروه پس از شکستن سیستم به بخش‌های کوچکتر، مستقیماً طراحی در سطح مدار انجام می‌شود. در این مرحله از طراحی، عملکرد به دست آمده از فاز قبل به کمک قوانین منطقی و جبر بولی تبدیل به یک سری تابع منطقی با ضابطه‌های مشخص شده و به شکل گیتهای منطقی ترکیبی، مدارات ترتیبی همزمان با غیر همزمان پیاده‌سازی می‌شود. خروجی این فاز یک طرح در سطح گیتهای منطقی است.

۴.۱.۱ فاز طراحی مداری

در این فاز مدار منطقی به دست آمده از گام قبلی با المان‌های الکترونیکی معادل جایگزین و شماتیک ترانزیستوری مدار، مشخصات الکترونیکی هریک از ترانزیستورها از نظر اندازه، تأخیر، سرعت سوئیچینگ، میزان مصرف توان، خازن‌های پارازیتی و ... نیز به دست می‌آید تا طرح در مراحل نهایی با استفاده از این پارامترها از صحت عملکرد نهایی اطمینان حاصل کند.

۵.۱.۱ فاز طراحی چینش

منظور از این فاز، طراحی مکان صحیح ترانزیستورهای مدار از نظر هندسی و اتصالات میان آن‌ها است. در این فاز، مکان فیزیکی هر ترانزیستور در فضای سیلیکونی تراشه و نقشه اتصالات درون آن معین می‌شود. این نقشه می‌تواند به صورت میله‌ای^۸ یا نمودار جانمایی^۹ باشد. نقشه ارتباطات ترانزیستوری یک تراشه علاوه بر فرم چینش و تعیین اتصالات ترانزیستورها، تعداد لایه‌های سیم بندی آن‌ها را نیز نشان می‌دهد. نقشه به دست آمده از این قسمت دارای قوانین ترسیمی مختلفی است که می‌بایست در طراحی مدار حتماً مورد توجه قرار گیرد. به این قوانین، قوانین طراحی^{۱۰} گفته می‌شود و به عملیات بررسی آنها در یک نقشه، DRC^{۱۱} اطلاق می‌گردد. تمامی فازهای طراحی مدار‌های مجتمع دیجیتال دارای یک مرحله وارسی به منظور بررسی صحت عملکرد فاز مربوطه می‌باشند. به عنوان نمونه، وارسی فاز طراحی چینش با عملیات DRC و LVS^{۱۲} پیگیری می‌شود.

۶.۱.۱ فاز ساخت تراشه

در این فاز از نقشه به دست آمده از مرحله قبلی جهت انتقال به ویفر اصلی استفاده می‌شود و پس از تقسیم بندی ویفر به die^{۱۳}‌های مشابه جهت طراحی مدار، نقشه را روی die^{۱۴}‌ها ایجاد می‌نمایند. ویفر بستری سیلیکونی است که در ساخت تراشه‌ها به عنوان لایه زیرین مدار قرار گرفته و مدار روی آن ساخته می‌شود. هر ویفر به تعداد زیادی die^{۱۵} تقسیم می‌شود. هر die^{۱۶} به یک تراشه اختصاص می‌یابد. به بیان دیگر کل مدارات درونی یک تراشه، برروی یک die^{۱۷} قرار می‌گیرد. در

⁸ Stick Diagram

⁹ Layout

¹⁰ Design Rules

¹¹ Design Rules Checking

¹² Logic VS. Schematic

تولید انبوه، اندازه ویفر معمولاً ثابت است و بر حسب بزرگی طرح مورد نظر، اندازه **die** در نتیجه تعداد **die** روی یک ویفر تغییر می‌کند.

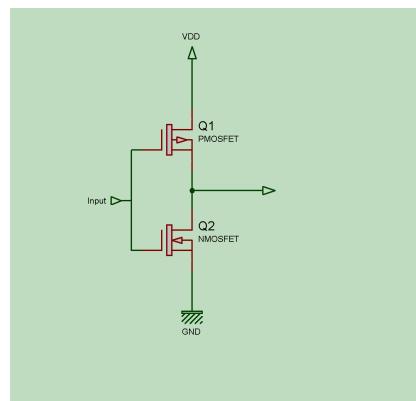
۲.۱ خانواده‌های مورد استفاده در طراحی دیجیتال

به طور کلی انواع منطق مورد استفاده در طراحی مدارهای مجتمع دیجیتال **CMOS** به دو دسته ایستا و پویا تقسیم‌بندی می‌شوند. هریک از این دسته‌ها خود دارای خانواده‌های بسیار گوناگونی هستند. که در ادامه به صورت خلاصه در مورد آنها بحث می‌شوند.

۱.۲.۱ انواع منطق ایستا

۱.۱.۲.۱ منطق CMOS ایستا

در این تکنولوژی که معروف‌ترین تکنولوژی ایستا می‌باشد، هر دو ترانزیستور **PMOS** و **NMOS** به ترتیب به عنوان مدارهای پایین‌کش و بالاکش استفاده می‌شوند.

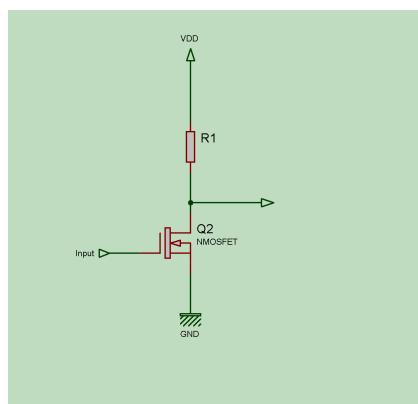


شکل ۲-۱۱ وارونگر طراحی شده در منطق CMOS ایستا

با توجه به اینکه در حالت پایدار، یکی از دو طبقه بالاکش و پایین‌کش فعال است، پس مسیری بین زمین و منبع تغذیه وجود ندارد. بنابراین مقدار **VOL** و **VOH** در این وارونگر با مشخصات دلخواه، همیشه به ترتیب برابر **VDD** و **GND** است.^[۲]

۲.۱.۲.۱ منطق NMOS مقاومتی

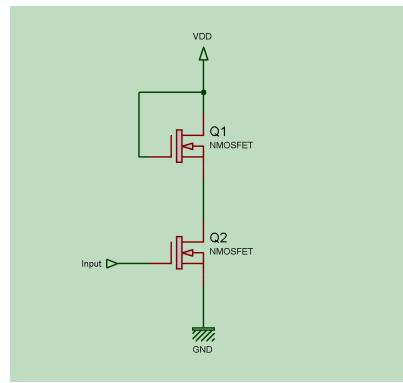
در این منطق طبقه پایین‌کش، از نوع ترانزیستورهای NMOS و طبقه بالاکش یک مقاومت خطی است. از آنجا که المان مقاومت یک المان الکتریکی حجیم (در مقایسه با اندازه ترانزیستورها در مدارات فشرده) است که همیشه رفتار یکسانی از خود نشان می‌دهد. با عبور جریان از این مقاومت می‌توان شاهد ایجاد اختلاف پتانسیلی در دو سر آن بود. و بدین ترتیب مسیری بین زمین و منبع تغذیه به وجود می‌آید که موجب تلفات می‌شود.^[۲]



شکل ۳-۱ وارونگر طراحی شده در منطق NMOS مقاومتی

۳.۱.۲.۱ منطق NMOS افزایشی

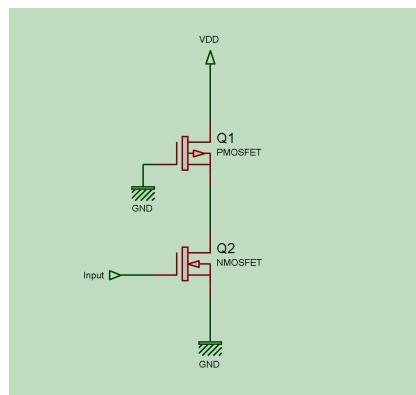
این منطق نیز از نظر ساختار کلی شبیه ساختارهای قبلی است با این تفاوت که بخش بالاکش مدار نیز مانند پایین‌کش آن، به صورت ترانزیستوری طراحی می‌شود. استفاده از مقاومت بالاکش در منطق NMOS مقاومتی باعث افزایش حجم مدار می‌شود، انعطاف پذیری طرح را در ساخت مدارات متنوع با مشخصات الکترونیکی قابل تنظیم کاهش می‌دهد.^[۳]



شکل ۴-۱۱ وارونگر طراحی شده در منطق NMOS افزایشی

منطق شبه NMOS^{۱۳} ۴.۱.۲.۱

منطق شبه NMOS شبیه NMOS افزایشی است با این تفاوت که طبقه بالاکش آن شامل یک ترانزیستور PMOS همیشه فعال است که گیت آن با ولتاژ زمین ثبیت شده است. این شیوه استفاده از ترانزیستور در طبقه بالاکش، منجر به ایجاد یک مسیر دائمی بین خروجی و منبع تغذیه می‌شود که البته با اعمال ورودی مناسب به مدار می‌توان تاثیر آن را خنثی کرد.^[۳]

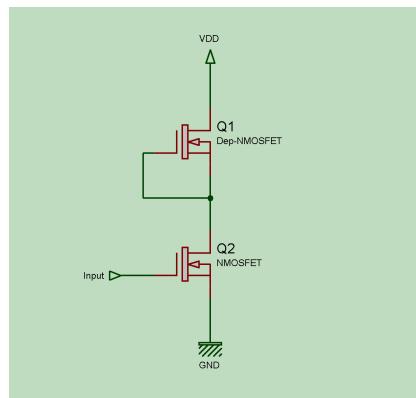


شکل ۵-۱۱ وارونگر طراحی شده در منطق شبه NMOS

¹³Pseudo-NMOS

٥.١.٢.١ منطق NMOS تخلیه ای^{١٤}

همان طور که از اسم این منطق نیز پیدا است، به جای استفاده از طبقه بالاکش افزایشی در مدار، از یک طبقه بالاکش تخلیه‌ای استفاده می‌کند.^[۳]



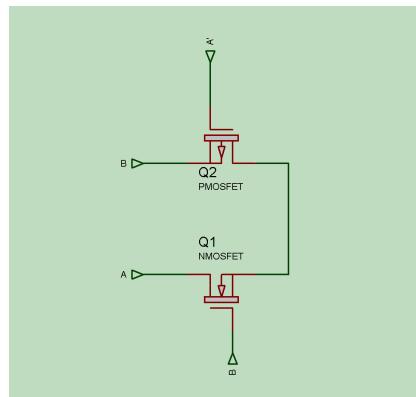
شکل ۱-۶ وارونگر طراحی شده در منطق NMOS تخلیه‌ای

٦.١.٢.١ منطق ترانزیستورهای عبوری^{١٥}

منطق‌های ترانزیستورهای عبوری مبتنی بر استفاده از ترانزیستورهای MOSFET با کنترل گیت، ورودی سورس و خروجی درین است. ولتاژ پایه‌های مختلف ترانزیستور به گونه‌ای است که عمدتاً افتد و لتاژ یکی از نواحی قطع یا خطی قرار می‌گیرند. افت ولتاژ یکی از بزرگترین مشکلات این MOSFET ها در یکی از نواحی قطع یا خطی قرار می‌گیرند. افت ولتاژ یکی از بزرگترین مشکلات این منطق می‌باشد.^[۴]

^{١٤} Depletion-NMOS

^{١٥} PTL(Pass Transistor Logic)



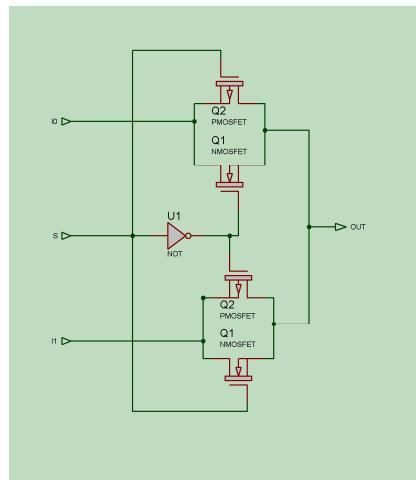
شکل ۷-۱۱ گیت AND طراحی شده در منطق ترانزیستور عبوری

۷.۱.۲.۱ منطق دروازه انتقال^{۱۶}

منطق دروازه انتقال با ظاهری کامل‌تر از ترانزیستورهای عبوری ارائه گردید تا مهم‌ترین عیب آن را رفع نماید. در بخش قبل توضیح دادیم که شاخه‌های تشکیل دهنده مدارات PTL در هر گام از یک ترانزیستور n یا p استفاده می‌کنند و مسیر ورودی تا خروجی را ایجاد می‌نمایند. ترانزیستور نوع p، انتقال دهنده ضعیف صفر و ترانزیستور نوع n، انتقال دهنده ضعیف یک منطقی است.

این منطق از نظر ساختار، شباهت زیادی با PTL دارد و همچنان از گیت ترانزیستورها جهت کنترل مسیر و از کanal سورس-درین آن‌ها به منظور ایجاد مسیر بین ورودی و خروجی استفاده می‌شود. تفاوت اصلی این دو در ترانزیستورهای استفاده شده در مسیرهاست که به جای هر سوئیچ از یک زوج ترانزیستور n و p استفاده می‌شود.^[۴]

^{۱۶} TG(Transmission Gate)



شکل ۸-۱۱ گیت MUX طراحی شده در منطق دروازه انتقال

۲.۲.۱ منطق های پویا^{۱۷}

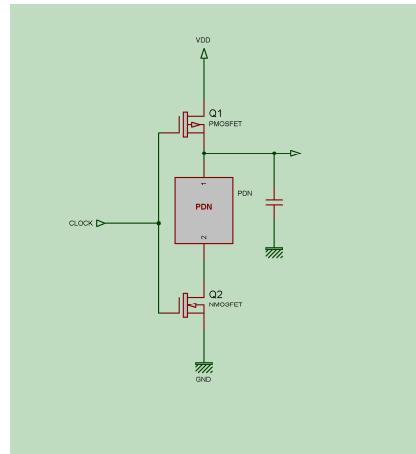
تکنیک‌های پویای CMOS عمدتاً مبتنی بر پالس‌های کلاک می‌باشند و برای طراحی مدارات ترتیبی دیجیتال به کار می‌روند. کلاک، موجی مربعی با فرکانس و دامنه ثابت است که به صورت متناوب مدار ترتیبی را تحریک می‌کند و باعث تغییر وضعیت یا تغییر مقدار خروجی آن با توجه به وضعیت جاری مدار و ورودی‌های فعلی آن می‌شود. مدارهای پویای CMOS علاوه بر طراحی ضابطه‌ی خروجی، ترانزیستورهایی را نیز جهت اعمال پالس ساعت در نظر می‌گیرند و با سپری کردن ۲ فاز عملکردی متوالی، تغییر وضعیت یا خروجی مدار را کنترل می‌کنند.^[۵]

۱.۲.۲.۱ منطق CMOS پویا

این منطق ساده‌ترین شکل پیاده‌سازی کلاک را در مدارات ترتیبی دیجیتال ارائه می‌کند. برای ساخت یک مدار منطقی کلاک‌دار در این قالب باید مشابه تکنیک‌های طراحی مدارات ترکیبی عمل کرد. پس از

¹⁷ Dynamic CMOS

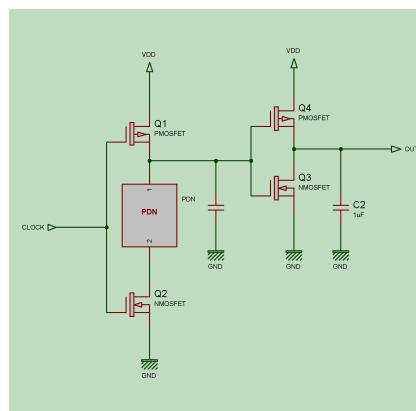
ساخت ضابطه خروجی با افزودن یک ترانزیستور نوع **p** در بالای آن و یک ترانزیستور نوع **n** در پایین آن و اعمال ورودی کلاک به هردوی آن‌ها طراحی را به اتمام می‌رسد.[۵]



شکل ۹-۱۱ قالب کلی مدارات CMOS پویا

منطق CMOS دامینو ۲.۲.۲.۱

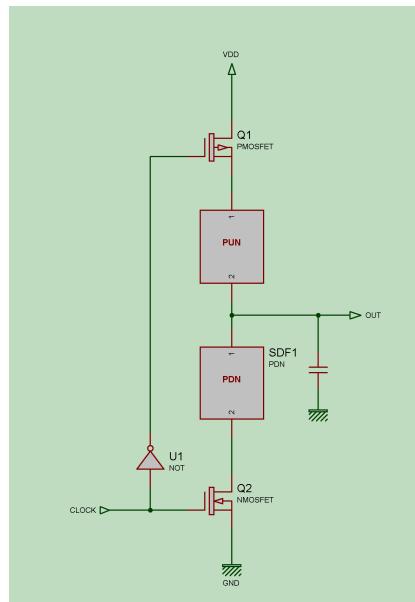
این منطق شباهت‌های فراوانی با منطق CMOS پویا دارد، با این تفاوت که خروجی CMOS دامینو یک NOT با خازن بار مربوط به آن اضافه شده است تا برخی مشکلات CMOS پویا بدین وسیله مرتفع گردد. گیت NOT بعد از خازن خروجی قرار می‌گیرد و باعث جدایی ولتاژ خروجی از عملیات زمان بر شارژ و دشارژ خازن می‌شود.[۵]



شکل ۱۰-۱۱ قالب کلی مدارات CMOS دامینو

۳.۲.۲.۱ منطق CMOS کلک دار^{۱۸}

این منطق از نظر عملکرد منطقی و ساختار ترانزیستوری شباهت زیادی به تکنولوژی CMOS ایستا دارد و روند کار آن مبتنی بر خازن خروجی نمی‌باشد. مداراتی که با این تکنیک طراحی می‌شوند همانند مدارات CMOS ایستا درای یک طبقه PDN^{۱۹} و یک طبقه PUN^{۲۰} می‌باشد.^[۵]



شکل ۱۱-۱۱ قالب کلی مدارات CMOS کلک دار

منطق‌های دیگری مانند PIPELINE Zipper NORA-CMOS هم وجود دارند که شباهت زیادی به منطق‌های معرفی شده دارند.

¹⁸ Clocked-CMOS

¹⁹ Pull-down Network

²⁰ Pull-up Network

۳.۱ پیشینه روش‌های بهینه‌سازی

همان‌طور که در بخش‌های قبل مطرح شد، مراحل ساخت مدارهای مجتمع دیجیتال شامل ۶ مرحله می‌باشد و امکان بهینه‌سازی در هر یک از این مراحل وجود دارد اما بیشتر روش‌های بهینه‌سازی مورد نظر مربوط به مراحل طراحی منطقی، طراحی مداری و طراحی چینش است.

در سال‌های اخیر در راستای گسترش طراحی اتوماتیک مدارهای دیجیتال، روش‌های طراحی بسیاری مورد استفاده قرار گرفته‌اند. در کلیه این روش‌ها، مساله طراحی به عنوان یک مساله بهینه‌سازی کلی مطرح می‌شود که به جستجوی پاسخ‌هایی از مدار می‌پردازند که اهداف طراحی در مراحل مطرح شده را ماکریم مینیمیم یا مکریم مینیمیم. روش‌های پیش طراحی مدارهای دیجیتال، غالباً بر مبنای جستجوی گرادیان عمل می‌کنند. مشکلاتی که این روش‌های همراه دارند عبارتند از: نیاز به حدس اولیه، گیر کردن در نقاط مینیمیم محلی واعلام آن به عنوان نتیجه نهایی و مشکلات کار با متغیرهای گستته و قیود غیر خطی.^[۶]

روش‌های کاربردی عمدتایی که در طراحی مدارات آنالوگ و دیجیتال به کار می‌روند و جز روش‌های تکاملی محسوب می‌شوند عبارتند از: الگوریتم‌های ژنتیک، برنامه‌نویسی ژنتیک^{۲۱}، تبرید شبیه‌سازی شده^{۲۲}، که تا کنون ابزارهای طراحی متنوعی بر اساس آن‌ها ساخته شده است که می‌توان به عنوان نمونه به موارد زیر اشاره کرد:

- ۱ - در ابزار طراحی محاوره‌ای **IDAC** کاربر این امکان را دارد تا از میان توبولوژی‌های تعریف شده در برنامه، توبولوژی‌های مورد نظر خود را انتخاب نماید، سپس این ابزار مقادیر پارامترهای طراحی توبولوژی فوق را ارائه می‌دهد و کاربر می‌تواند بهترین مدار سایزبندی شده را انتخاب کند. اما این نرمافزار در هر مرحله این عمل را فقط برای یک منطق انجام می‌دهد.^[۷]

²¹ Genetic Programming

²² Simulated Annealing

۲- در **OASYS** در ابتدا یک توپولوژی مدار بر مبنای قوانین اکتشافی^{۲۳} تعیین می‌شود. سپس توسط ابزار طراحی، سایز بندی می‌گردد.^[۷]

۳- در **SEAS** از اصل تکامل جهت یافتن توپولوژی مورد نظر و از تبرید شبیه‌سازی شده جهت سایزبندی مدار استفاده می‌شود.^[۷]

²³ **Heuristic**

۲

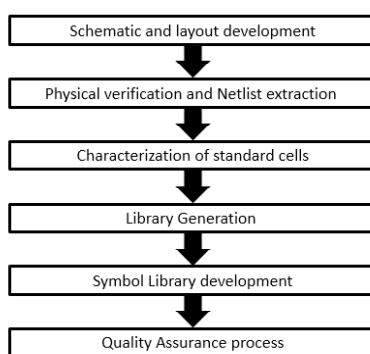
فصل دوم

طراحی کتابخانه سلول‌های استاندارد

طراحی کتابخانه سلول‌های استاندارد

در این فصل مراحل مختلف طراحی یک کتابخانه سلول استاندارد (برای نمونه منطقی DMTGDI) شرح داده می‌شود.

شکل ۱.۲ روند ایجاد کتابخانه سلول‌های استاندارد را نشان می‌دهد.



شکل ۱-۲۲ مراحل طراحی کتابخانه [۸]

در مرحله‌ی اول شماتیک و لی‌اوٹ هر سلول رسم می‌شود. به دنبال آن تست‌های LVS و DRC روی آن اعمال شده و نت‌لیست طراحی استخراج می‌گردد. سپس، سلول‌ها مشخصه‌یابی شده و فایل‌های کتابخانه ایجاد خواهند شد. در نهایت کیفیت کتابخانه ایجاد شده با سنتز نمونه کد Verilog در این کتابخانه مورد ارزیابی قرار می‌گیرد. در این فصل، پنج مرحله اول بطور اجمالی مرور خواهند شد. جزئیات بیشتر نحوه اجرای مراحل مختلف ایجاد کتابخانه سلول‌های استاندارد، در پیوست ۱ بررسی شده است.

۱.۲ رسم شماتیک و لی‌اوت سلول‌ها

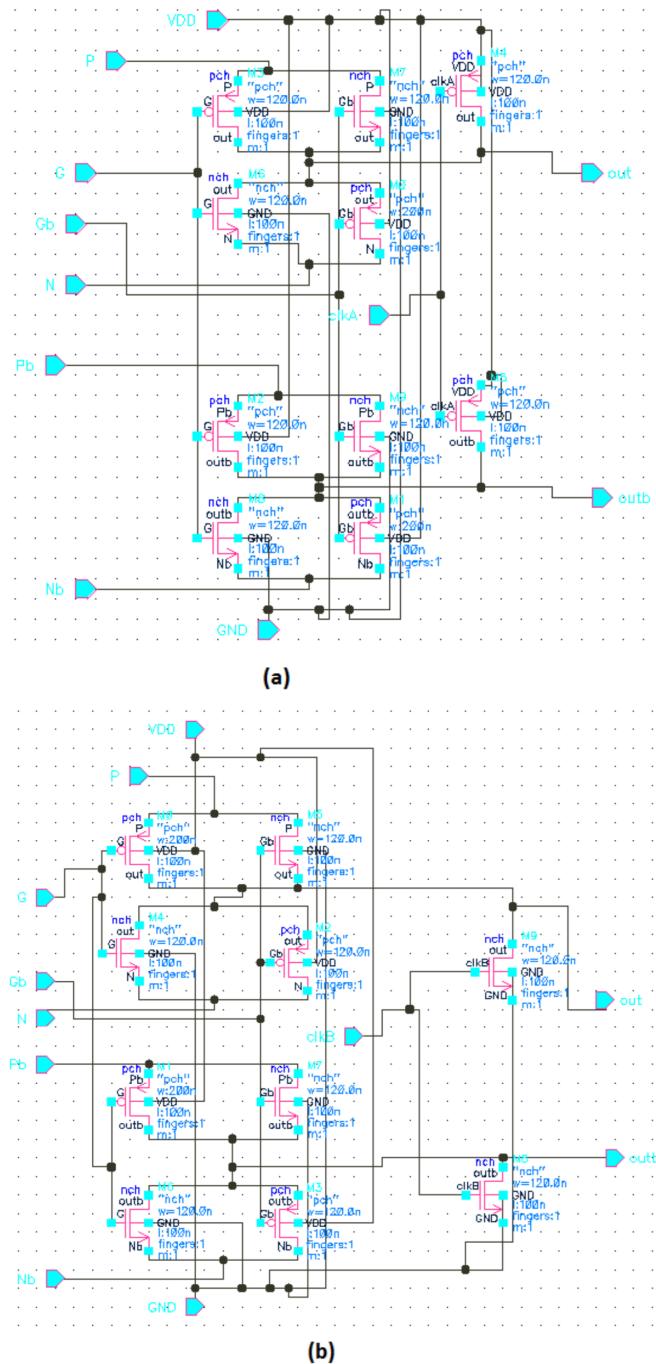
سلول‌های استاندارد منطق DMTGDI از ۱۰ ترانزیستور تشکیل شده که شامل ۸ ترانزیستور اصلی و دو ترانزیستور انتخاب کننده حالت است. از آنجا که هدف این فصل از پایان‌نامه تنها نشان دادن مراحل ساخت کتابخانه است، به جزئیات این منطق کم‌توان پرداخته نمی‌شود.

ایجاد توابع مختلف در این منطق تنها به ورودی‌های اعمال شده به هسته آن بستگی دارد. بنابراین تنها کافیست دو نمونه سلول واحد برای نوع **A** و **B** ساخته شده و ایجاد توابع مختلف با اعمال ورودی‌های متفاوت به این سلول‌های واحد امکان پذیر خواهد بود. جدول ۱، ۲ اورودی‌های اعمال شده به سلول واحد را برای ساخت هر تابع نشان می‌دهد. این ویژگی باعث می‌شود که یکنواختی و هماهنگی سلول‌ها در این خانواده منطقی بیشتر باشد.

جدول ۱-۲ نحوه ایجاد توابع مختلف کتابخانه از طریق سلول پایه [۸]

Inputs	Out/ \bar{Out}	AND	OR	Inverter	XOR	MUX	F1	F2
P	GND	A	VDD		B	A	B	VDD
N	B	A	GND		\bar{B}	B	GND	B
G	A	A	A		A	S	A	A
Out function	AB	A+B	\bar{A}		$\bar{A}B + A\bar{B}$	$\bar{S}A + SB$	$\bar{A}B$	$\bar{A} + B$

طرح شماتیک سلول‌های واحد نوع A و B در شکل ۲-۲ آمده است:

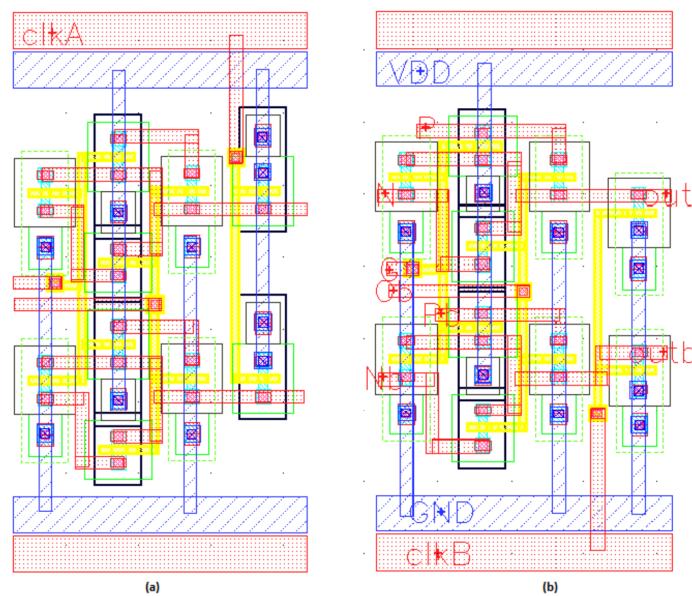


شکل ۲-۲ شماتیک سلول‌های واحد [۸]

پس از شماتیک نوبت به رسم لی‌اوتنمی‌های واحد میرسد. مشخصات لی‌اوتنمی‌های رسم شده در جدول ۲،۲ خلاصه شده است. مطابق این جدول، در ایجاد لی‌اوتنمی‌های استاندارد فقط از دولایه فلزی Metal1 و Metal2 استفاده شده و لایه‌های بالاتر فلز برای اتصال سلول‌ها در مرحله P&R بکار برده خواهند شد. لی‌اوتنمی‌های واحد نوع A و B در شکل ۳،۲ نشان داده شده است.

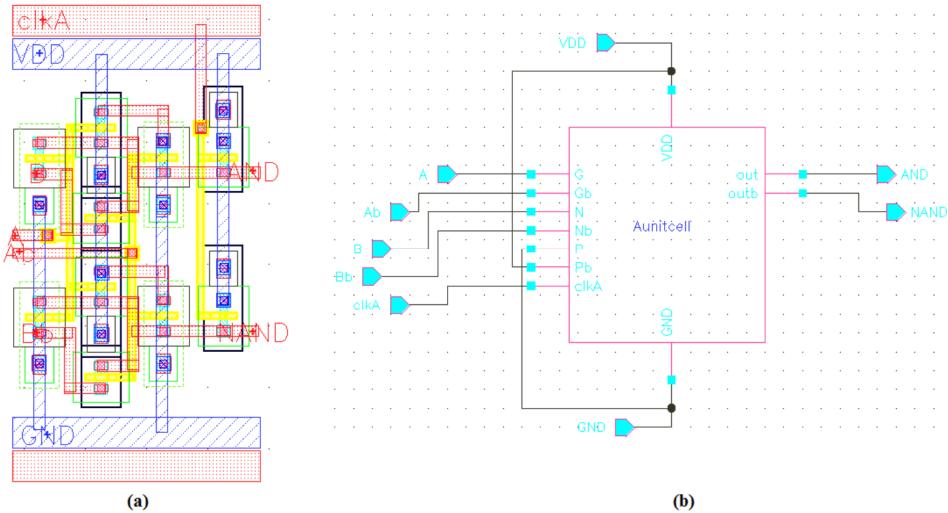
جدول ۲-۲ شرایط بکار رفته در رسم لی‌اوتنمی‌های سلول‌ها [۸]

5.670um	ارتفاع سلول
3.520um	عرض سلول
Metal1 & metal2	لایه‌های فلزی
180nm	عرض متال
495nm	عرض ریل‌های VSS و VDD



شکل ۳-۲۲ لی‌اوتنمی‌های واحد [۸]

بدین ترتیب می‌توان تنها با اعمال تغییرات جزئی در سلول‌های استاندارد پایه، توابع مختلف را با مساحت یکسان پیاده‌سازی کرده و ریلهای **VSS** و **VDD** تمام سلول‌ها برقیکار منطبق خواهد بود. به عنوان مثال شکل ۴.۲ شماتیک و لی‌اوت گیت **AND/NAND** دو ورودی را در منطق **A** نوع **DMTGDI** نشان می‌دهد.



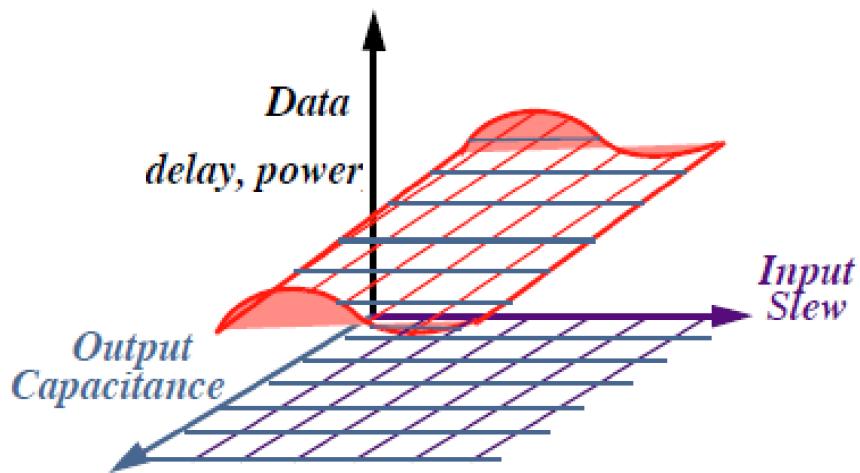
[۸] شکل ۴-۲۲ شماتیک و لی‌اوت گیت **AND/NAND**

پس از رسم شماتیک و لی‌اوت سلول‌ها صحت فیزیکی آن‌ها بررسی شده و در نهایت نت‌لیست هر سلول به همراه المان‌های پارازیتی مربوط به آن استخراج خواهد شد. از نت‌لیست‌های بدست آمده در این بخش برای ایجاد کتابخانه **.lib** استفاده می‌شود.

۲.۲ مشخصه‌یابی سلول‌های استاندارد و ایجاد کتابخانه‌های (**.db**) و (**.lib**)

مشخصه‌یابی سلول‌ها به معنای استخراج پارامترهای کلیدی سلول نظیر مساحت، تاخیرهای بالارونده، پایین‌رونده و توان مصرفی سلول است. این کار با شبیه‌سازی نت‌لیست‌های بدست آمده از روی لی‌اوت و توسط نرم افزار **HSPICE**، تحت مقادیر مختلف مخازن بار و شیب گذار ورودی، انجام خواهد شد.^[۹]

مشخصه‌های ورودی به صورت یک ماتریس دو بعدی، مطابق شکل ۵,۲ است که در آن به ازای هر شبیه‌ورودی و خازن خروجی یک مقدار توان و تاخیر بدست آمده و در ماتریس مشخصه‌یابی وارد خواهد شد.



شکل ۵-۲۲ نحوه تعیین مولفه‌های ماتریس دو بعدی حاصل از مشخصه‌یابی

اجرای دستی عملیات مشخصه‌یابی برای تک تک سلول‌ها فرایند بسیار زمانبری خواهد بود. لذا برای اجرای اتوماتیک این فرآیند، نرم افزارهایی مختلفی ارائه شده‌اند. در این پژوهه از نرم افزار **NCX** استفاده شده است.

ضمن اجرای برنامه **NCX** لازم است که توان نشتی سلول‌ها، به عنوان یکی از ورودی‌ها به برنامه داده شود. نحوه اندازه‌گیری توان نشتی سلول‌ها در پیوست ۱ آورده شده است. توان نشتی سلول‌ها در حالت واقعی به ورودی‌های سلول وابسته است، با این حال برای کار با نرم افزار **NCX** نیاز به معرفی یک مقدار ثابت برای هر سلول می‌باشد، لذا برای هر سلول، توان نشتی مربوط به بدترین حالت را وارد کرده و بدین ترتیب این اطمینان حاصل می‌شود که توان نشتی تابع سنتز شده در این کتابخانه قطعاً کوچکتر از مقدادر بدست آمده خواهد بود.

با اجرای برنامه **NCX**، در نهایت فایل **.lib** از این بخش بدست می‌آید. توصیف اطلاعات موجود در این کتابخانه در پیوست ۲ آورده شده است. پس از این مرحله مطابق روند توصیف شده در پیوست ۱، فایل **.db** نیز از روی این کتابخانه‌ها ساخته شده که در مرحله‌ی سنتز به کار می‌آید.

در نهایت کتابخانه سمبل‌ها نیز، مطابق روند مطرح شده در پیوست ۱، برای مجموعه سلول‌های این کتابخانه ایجاد شده و بدین ترتیب فرایند ساخت کتابخانه‌های مورد نیاز برای تست و سنتز منطق مورد نظر تکمیل می‌شود.

۳.۲ خلاصه

در این فصل، مراحل ایجاد کتابخانه‌های مورد نظر در تکنولوژی ۹۰ نانومتر بطور اجمالی مورد بررسی قرار گرفته شد. فایل‌های بدست آمده از این فصل در شکل ۶،۲ آمده است.



شکل ۶-۲۲ کتابخانه‌های ایجاد شده.^[۸]

۳

فصل سوم

یادگیری ماشین و طراحی سیستم توصیه‌گر منطق

یادگیری ماشین و طراحی سیستم توصیه‌گر منطق

پیش از ورود به جزئیات طراحی این سیستم توصیه‌گر، لازم است برخی مفاهیم و تعاریف مورد استفاده علم یادگیری ماشین در این زمینه را مطرح نمود.

۱.۳ یادگیری ماشین^{۲۴}

به عنوان یکی از شاخه‌های وسیع و پرکاربرد هوش مصنوعی، یادگیری ماشین به تنظیم و اکتشاف شیوه‌ها و الگوریتم‌هایی می‌پردازد که براساس آن رایانه‌ها و سامانه‌ها توانایی تعلم و یادگیری پیدا می‌کنند.

هدف یادگیری ماشین این است که کامپیوتر (در کلی ترین مفهوم آن) بتواند به تدریج و با افزایش داده‌ها کارایی بهتری در انجام وظیفه مورد نظر پیدا کند. گستره این وظیفه می‌تواند از تشخیص خودکار چهره با دیدن چند نمونه از چهره مورد نظر تا فرآگیری شیوه گامبرداری ربات‌های دوپا با دریافت سیگнал پاداش و تنبیه باشد.^[۱۰]

طیف پژوهش‌هایی که در یادگیری ماشینی می‌شود گسترده‌است. در سوی نظری آن پژوهش‌گران برآن‌اند که روش‌های یادگیری تازه‌ای به وجود بیاورند و امکان‌پذیری و کیفیت یادگیری را برای روش‌های شان مطالعه کنند و در سوی دیگر عده‌ای از پژوهش‌گران سعی می‌کنند روش‌های یادگیری ماشینی را بر مسائل تازه‌ای اعمال کنند. البته این طیف گستته نیست و پژوهش‌های انجام‌شده دارای مولفه‌هایی از هر دو رویکرد هستند.^[۱۰]

روش‌های یادگیری ماشین به صورت کلی به ۳ دسته یادگیری نظارت شده، یادگیری بی نظارت و یادگیری تقویتی تقسیم بندی می‌شوند.

²⁴Machine Learning

۱.۱.۳ یادگیری نظارت شده^{۲۵}

این روش، یک روش عمومی در یادگیری ماشین است که در آن به یک سیستم، مجموعه جفت‌های ورودی-خروجی ارائه شده و سیستم تلاش می‌کندتا تابعی از ورودی به خروجی را فرآورد. یادگیری نظارت شده نیازمند تعدادی داده ورودی به منظور آموزش سیستم است. با این حال ردهای از مسائل وجود دارند که خروجی مناسب که یک سیستم یادگیری تحت نظارت نیازمند آن است، برای آن‌ها موجود نیست. این مسائل چندان قابل حل با این روش نیستند و یادگیری تقویتی روشی برای حل این سری از مسائل را ارائه می‌دهد. در یادگیری تقویتی، سیستم تلاش می‌کند تا تقابلات خود با یک محیط پویا را از طریق آزمون و خطابه نماید.^[۱۱]

۲.۱.۳ یادگیری بی‌نظارت^{۲۶}

یادگیری بی‌نظارت یکی از انواع یادگیری ماشین است که در آن زوج‌های ورودی-خروجی نداریم و سیستم سعی می‌کند با استفاده از داده‌های تکی عملیات برچسب‌گذاری و خوشه‌بندی را انجام دهد. از جمله روش‌های معروف یادگیری ماشین بی‌نظارت می‌توان به خوشه‌بندی، مدل پنهان مارکف و برخی از شبکه‌های عصبی مصنوعی نام برد.^[۱۱]

۳.۱.۳ یادگیری تقویتی^{۲۷}

یادگیری تقویتی، روشی است که بیشتر از روش‌های برنامه نویسی پویا، استفاده می‌کند و معمولاً به صورت یک فرآیند تصمیم‌گیری مارکف^{۲۸} مدل می‌شود. تفاوت اصلی بین روش‌های سنتی والگوریتم‌های یادگیری ماشین تقویتی این است که در یادگیری تقویتی نیازی به داشتن اطلاعات راجع به فرآیند

²⁵Supervised Learning

²⁶Unsupervised Learning

²⁷Reinforcement Learning

²⁸Markov Decision Process

تصمیم‌گیری نیست و این که این روش روی فرآیندهای مارکف بسیار بزرگی کار می‌کند که روش‌های سنتی در آنجا ناکارامند.^[۱۱]

یادگیری تقویتی با یادگیری با نظارت معمول دو تفاوت دارد، نخست اینکه در آن زوج‌های ورودی-خروجی وجود ندارند و رفتارهای ناکارامد سیستم از بیرون اصلاح نمی‌شود، و دیگری اینکه تمرکز زیادی روی کارایی زنده وجود دارد که نیازمند پیداکردن یک تعادل نسبی بین اکتشافات چیزهای جدید و بهره برداری از دانش اندوخته شده است.

۲.۳ شبکه‌های عصبی^{۲۹}

شبکه‌های عصبی مصنوعی، یا به زبان ساده‌تر شبکه‌های عصبی سیستم‌ها و روش‌های محاسباتی نوینی برای یادگیری ماشین، نمایش دانش و در انتهای اعمال دانش به دست آمده در جهت پیش‌بینی پاسخ‌های خروجی از سامانه‌های پیچیده هستند. ایده اصلی اینگونه شبکه‌ها الهام گرفته از شیوه کارکرد سیستم عصبی زیستی، برای پردازش داده‌ها و اطلاعات برای یادگیری و ایجاد دانش است. عنصر کلیدی این ایده، ایجاد ساختارهای جدید برای سامانه پردازش اطلاعات است.^[۱۲]

با استفاده از دانش برنامه‌نویسی رایانه می‌توان ساختار داده‌ای طراحی کرد که همانند یک نرون عمل می‌کند. سپس با ایجاد شبکه‌ای از این نرون‌های مصنوعی به هم پیوسته، ایجاد یک الگوریتم آموزشی برای شبکه و اعمال این الگوریتم به شبکه آن را آموزش داد.

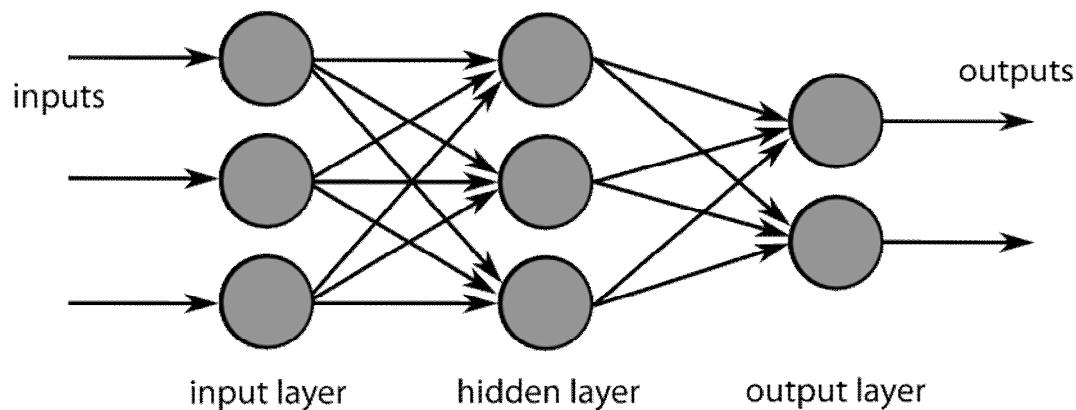
یک شبکه عصبی از سه لایه ورودی، خروجی و پردازش تشکیل شده است. هر لایه شامل گروهی از سلول‌های عصبی (نورون) است که عموماً با کلیه نورون‌های لایه‌های دیگر در ارتباط هستند. مگر اینکه کاربر این ارتباطات را محدود کند، ولی نورون‌های هر لایه با سایر نورون‌های آن لایه ارتباطی ندارند.
[۱۲]

شبکه‌های عصبی انواع گوناگونی دارد که در زیر به اختصار یک خانواده از آن‌ها که در طراحی این پروژه به کار رفته‌اند را توضیح می‌دهیم.

²⁹Artificial Neural Network

۱.۲.۳ شبکه عصبی پس انتشار^{۳۰}

این شبکه مجموعه‌ای از نورون‌ها است که در لایه‌های مختلفی پشت سر هم قرار گرفته‌اند. مقادیر ورودی پس از ضرب در وزن‌های موجود در گذرگاه‌های بین لایه‌ها به نورون بعدی رسیده و در آن جا با هم جمع می‌شوند و پس از عبور از تابع شبکه مربوطه خروجی نورون‌ها را تشکیل می‌دهند. در پایان خروجی به دست آمده با خروجی مورد نظر مقایسه شده و خطای به دست آمده جهت اصلاح وزن‌های شبکه به کار می‌رود، این امر اصطلاحاً آموزش شبکه عصبی نامیده می‌شود. [۱۲]



شکل ۱-۳۳ ساختار کلی یک شبکه عصبی پس انتشاری تمام متصل^{۳۱} [۱۲]

۲.۲.۳ پرسپترون

پرسپترون‌ها متدالو ترین و شناخته شده ترین نوع از میان انواع شبکه‌های عصبی پس انتشاری هستند که برای طبقه‌بندی استفاده می‌شوند.

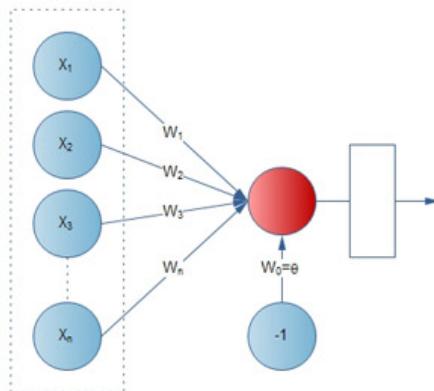
³⁰Feed-Forward

³¹Fully Connected

۱.۲.۲.۳ پرسپترون تک لایه^{۳۲}

پرسپترون در حالت پایه یک نوع تفکیک کننده دودویی است که ورودی خود (یک بردار از نوع اعداد حقیقی) را به یک مقدار خروجی (یک اسکالر از نوع اعداد حقیقی) که به صورت زیر محاسبه می‌شود متناظر می‌کند:

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$



شکل ۲-۳۳ ساختار کلی یک پرسپترون تک لایه [۱۳]

قضیه همگرایی بیان می‌کند که اگر بتوان پاسخی را وسیله‌ی یک پرسپترون پیاده‌سازی کرد، قاعده آموزش در تعداد متناهی گام به یک جواب خواهید رسید. مشکل بزرگ پرسپترون تک لایه عدم توانایی آن در جداسازی الگوهای در توابعی است که تفکیک‌پذیر خطی نیستند. [۱۳]

³²SLP(Single Layer Perceptron)

۲.۲.۲.۳ پرسپترون چندلایه دودویی^{۳۳}

پرسپترون چندلایه قادر به طبقه‌بندی‌های پیچیده است، مشروط به اینکه تعداد کافی لایه پرسپترون در شبکه و تعداد کافی نورون در هر لایه وجود داشته باشد.

در مورد تعداد لایه‌های مورد نیاز برای پرسپترون چند لایه، قضیه لیپمن^{۳۴} بیان می‌کند که هر تابع n متغیره پیوسته‌ای را می‌توان تنها با استفاده از توابع خطی و توابع غیرخطی اما پیوسته صعودی یک متغیره محاسبه کرد. در واقع این قضیه بیان می‌کند که در پرسپترون سه لایه‌ای با $n(2n+1)$ گره که از غیرخطی‌های پیوسته صعودی استفاده می‌کند می‌تواند هر تابع n متغیره پیوسته‌ای را محاسبه کند بر این اساس با استفاده از یک پرسپترون سه لایه هر تابع درستنمایی^{۳۵} پیوسته‌ای را که در یک طبقه‌بندی کننده لازم است، ایجاد کرد.^[۱۴]

تایید بیشتر این نتیجه‌گیری را می‌توان در (لپیدس و فاربر، ۱۹۸۸)^{۳۶} یافت که در آنجا چنین بحث شده است:

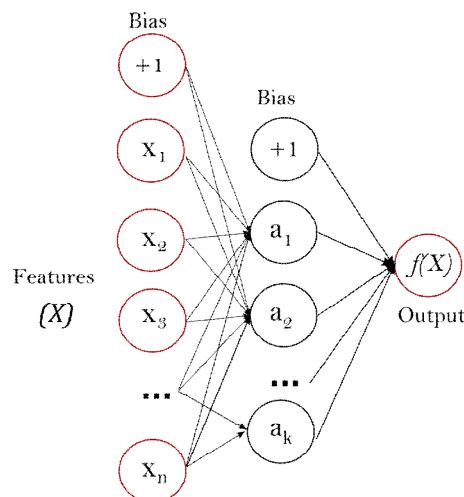
برای پردازش داده ورودی با مقدار حقیقی به بیش از دو لایه پنهان نیاز نیست و دقت تقریب توسط تعداد سلول‌های عصبی در هر لایه، و نه با تعداد لایه‌ها، کنترل می‌شود.^[۱۴]

^{۳۳} BMLP(Binary Multilayer Perceptron)

^{۳۴} (Lippmann, 1987)

^{۳۵} Likelihood function

^{۳۶} (Lapedes and Farber, 1988)

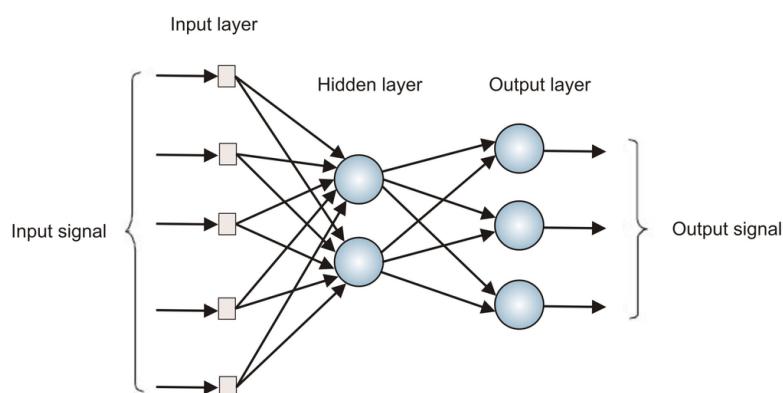


شکل ۳-۳۳ ساختار کلی یک پرسپترون چندلایه با اینتری [۱۵]

۳.۲.۲.۳ پرسپترون چندلایه - چندطبقه

پرسپترون‌ها همان‌طور که در ابتدا نیز اشاره شد در حالت پایه به صورت دودویی بوده و برای طبقه‌بندی‌های دوتایی استفاده می‌شوند. که البته امکان تعمیم آن‌ها به حالت طبقه‌بندی چندتایی نیز وجود دارد.

برای ایجاد طبقه‌بندی‌های مختلف در لایه خروجی به جای استفاده از یک نورون، به تعداد طبقه‌ها نورون قرار می‌دهیم و شبکه عصبی به یک شبکه رقابتی تبدیل می‌شود. به ازای هر بردار ورودی نورون خروجی که مقدار آن بیشتر باشد برنده می‌شود.



شکل ۴-۳۳ ساختار کلی یک پرسپترون چندلایه - چندطبقه [۱۵]

آموزش پرسپترون

۴.۲.۲.۳

روش‌های آموزش مختلفی برای پرسپترون‌ها ارائه شده است که معروفترین آنها روش پس انتشار خطای است. در این روش برای آموزش در خلاف جهت حرکت شبکه خطاهای را محاسبه و به لایه قبلی انتشار می‌دهیم تا به لایه اول برسیم. در این بخش ابتدا روش آموزش پرسپترون چند لایه دودویی را شرح می‌دهیم سپس آن را به پرسپترون چندلایه-چندطبقه تعمیم می‌دهیم. [۱۴]

در ابتدا لازم است برخی از نمادها را معرفی کنیم. از آنجا که خروجی‌های یک لایه ورودی‌های لایه بعدی می‌شود، استفاده از نشانه‌های مختلف برای ورودی و خروجی‌ها کار موجبه نیست، بنابراین از \mathbf{x} برای نمایش مقدار وارد شده از یک پرسپترون استفاده خواهد شد. هر لایه با شروع از ۱ در لایه ورودی شماره‌ای خواهد داشت. ورودی‌های شبکه $(\mathbf{x}_0(1), \mathbf{x}_0(2), \dots, \mathbf{x}_0(n))$ هستند. خروجی‌های لایه اول، که ورودی‌های لایه دوم هستند $(\mathbf{x}_1(1), \mathbf{x}_1(2), \dots, \mathbf{x}_1(m))$ وغیره هستند. بنابراین (\mathbf{x}_i) خروجی سلول عصبی i -ام در لایه $-l$ است.

در مورد وزن‌ها زیرنویس معرف لایه خواهد بود. بنابراین به عنوان مثال $w_{2,1}$ نشان دهنده وزنی در لایه دوم است که به $x_1(3)$ متصل و در خروجی $x_2(1)$ از طریق تابع غیر خطی f به صورت زیر سهیم است.

$$x_2(1) = f(\text{net}_2(1))$$

برای شبکه کاملاً همبند مقدار net به صورت زیر محاسبه می‌شود:

$$\text{net}_2(1) = \sum_{i=0}^3 w_2(i, 1)x(i)$$

در طبقه آخر معمولاً از تابع سیگموید^{۳۷} یا تانژانت هیپربولیک^{۳۸} استفاده می‌شود که مشتق گیری آن‌ها بسیار ساده است و همچنین معیاری احتمالی را نشان می‌دهند. [۱۴]

³⁷Sigmoid Function

³⁸Hyperbolic Tangent

$$\begin{aligned}y &= \frac{1}{1+e^{-x}} \\ \frac{\partial y}{\partial x} &= y(1-y)\end{aligned}$$

میزان تغییر وزن‌ها در یک پرسپترون با تابع زیگموید به صورت زیر است :

$$\Delta w_i = \frac{\eta}{P} \sum_{p=1}^P x_{ip} \delta p$$

که در آموزش برخط می‌توان عبارت بالا را به صورت زیر تقریب زد :

$$\Delta w_i = \eta x_i \delta$$

که در این معادله δp برای لایه آخر به صورت زیر تعریف می‌شود :

$$\delta p = y_p(1-y_p)(d_p - y_p)$$

برای محاسبه خطای لایه‌های قبلی از خطای لایه آخر استفاده می‌کنیم :

$$\begin{aligned}\Delta w_2 &= \eta x_1(1) \delta_2(2) \\ \delta_2(2) &= x_2(2)[1-x_2(2)]w_3(2,1) \delta_3(1) \\ \delta_3(1) &= x_3(1)[1-x_3(1)][d - x_3(1)]\end{aligned}$$

برای تعمیم این روش آموزش برای پرسپترون‌های چند طبقه در هر لحظه خروجی را به صورت برنده خروجی‌ها معرفی می‌کنیم :

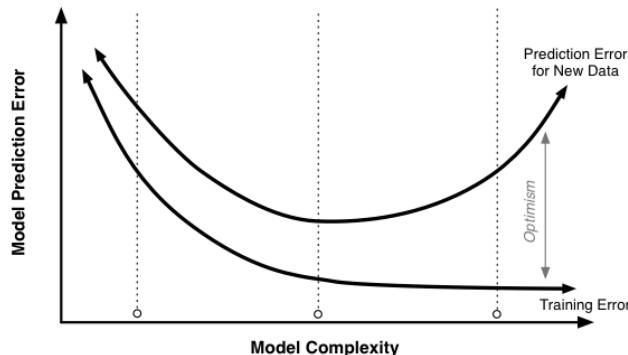
$$\begin{aligned}\hat{y} &= \arg \max_y f(x, y).w \\ w_{t+1} &= w_t + f(x, y) - \hat{f}(x, \hat{y})\end{aligned}$$

شرط توقف آموزش ۵.۲.۲.۳

هنگام آموزش یک پرسپترون با مجموعه‌ای از داده‌ها شروع می‌کنیم. در گام نخست داده‌ها را به دو قسمت آموزش و آزمون تقسیم می‌کنیم. با استفاده از داده‌های آموزش شبکه را تعلیم می‌دهیم سپس با استفاده از مجموعه آزمون، آن را آزمایش می‌کنیم تا عملکرد شبکه بر روی داده‌های جدید را مشاهده کنیم.

تمایل طبیعی این است که بخواهیم خطا را حد ممکن کوچک شود، به طوری که شبکه عصبی بتواند داده آموزشی را با بیشترین دقت ممکن بازسازی کند. اما تجربه نشان می‌دهد که شبکه هرگز به آن خوبی که برروی مجموعه آموزش عمل می‌کند ببروی مجموعه آزمون عمل نمی‌کند. این پدیده به عنوان آموزش زیادی شناخته می‌شود و گفته می‌شود که شبکه برای داده آموزش برازش زیادی^{۳۹} دارد.

در حین فرآیند آموزش، خطای بین خروجی مطلوب و خروجی واقعی باید کاهش یابد، اما همان طور که در شکل ۵.۳ مشاهده می‌شود، خطای داده‌های آزمون^{۴۰} هم باید کاهش یابد ولی بیشتر از خطای داده آموزش^{۴۱} باشد. در مرحله‌ای از آموزش، کاهش خطای داده آزمون متوقف می‌شود و حتی ممکن است افزایش هم داشته باشد. این وقتی است که آموزش زیادی شروع می‌شود و شبکه شروع به برازش زیادی داده آموزشی می‌کند. اگر آموزش در نقطه‌ای که خطای داده‌های آموزش شروع به افزایش می‌کند متوقف شود، آن گاه از آموزش زیادی می‌توان اجتناب کرد. به این فرآیند قانون آرنج^{۴۲} هم گفته می‌شود. [۱۴]



شکل ۵-۳ مقایسه عملکرد شبکه برروی داده‌های آموزش و داده‌های تست [۱۶]

³⁹Overfitting

⁴⁰Out-Sample Error

⁴¹In-Sample Error

⁴²Elbow Rule

۳.۳ سیستم توصیه‌گر منطق

همان طور که در بخش‌های قبل نیز اشاره شد این سیستم توانایی توصیه منطق مورد نظر برای هر قسمت از مدار را برای بهبود هریک از پارامترهای مورد نظر یعنی توان مصرفی، تاخیر و مساحت را دارا می‌باشد.

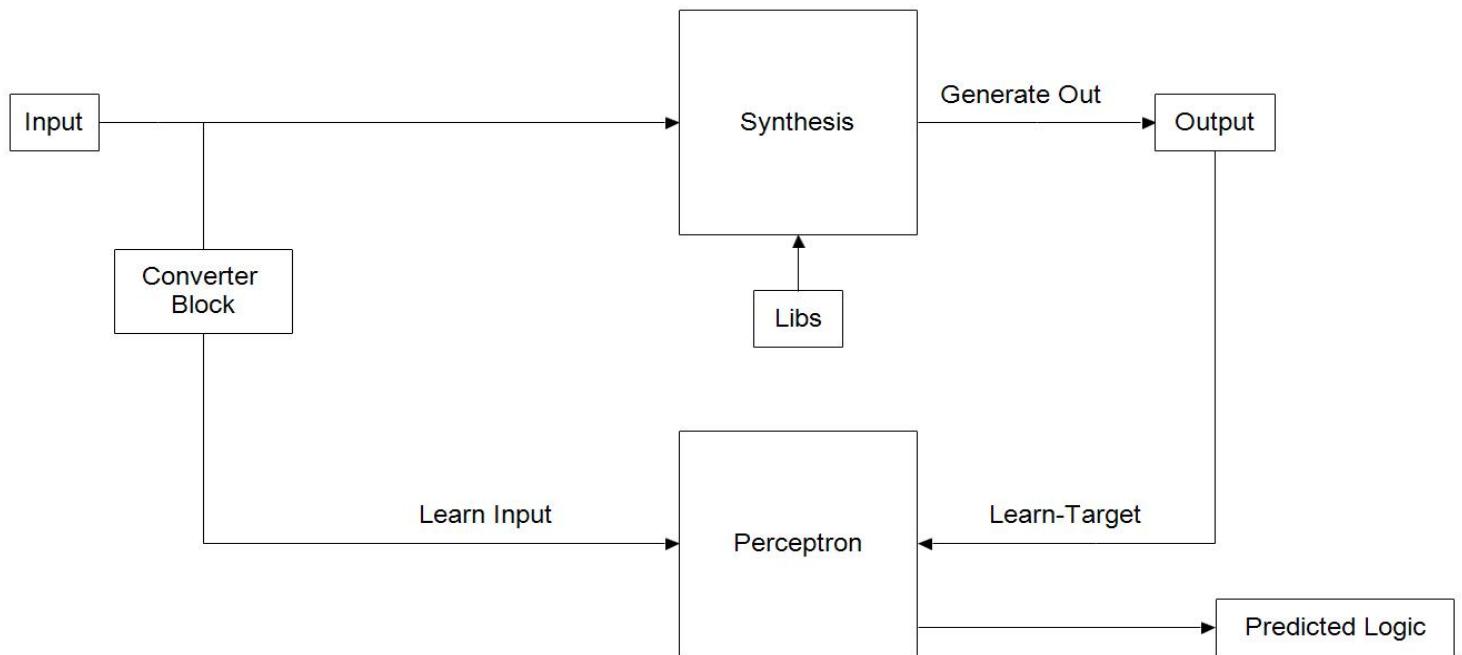
در این بخش روند طراحی و عملکرد این سیستم را به صورت کلی تشریح می‌کنیم و در فصل بعد نحوه پیاده‌سازی و ارزیابی عملکرد سیستم بیان خواهد شد.

۱.۳.۳ توصیف سیستم

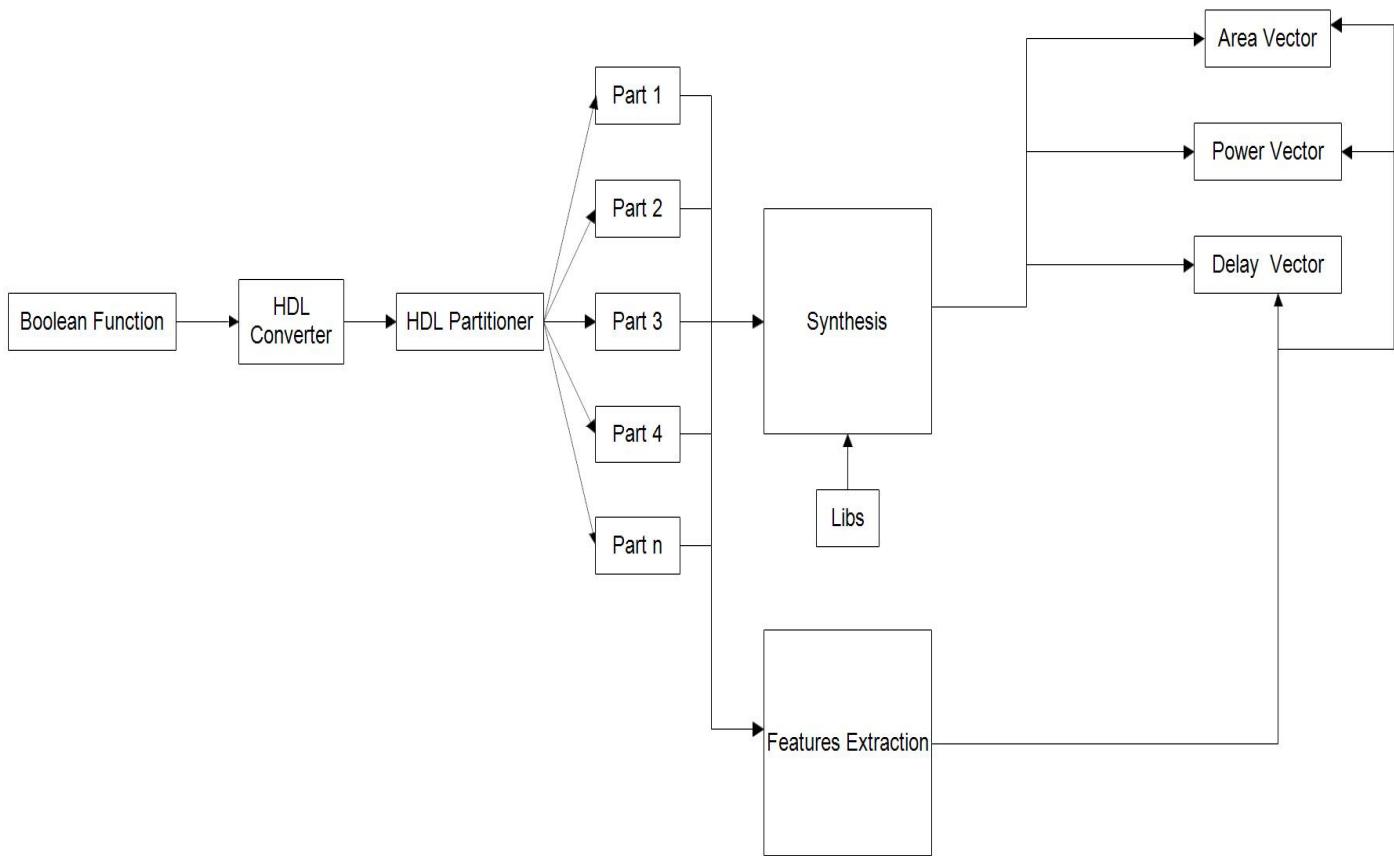
این سیستم را به طور کلی می‌توان به دو بخش سنتزکننده-شبیه‌ساز و تصمیم‌گیرنده تقسیم کرد. نمای کلی سیستم در شکل ۶,۳ آمده است. همان‌طور که در شکل ۷,۳ نشان داده شده در قسمت ورودی بخش سنتزکننده-شبیه‌ساز در صورتی که تابع به صورت تابع منطقی باشد با استفاده از بلوک تبدیل، به یک فایل زبان توصیف سختافزاری تبدیل می‌شود. در ادامه این فایل به بلوک جداکننده وارد می‌شود (معیار جداسازی بلوک‌ها به دو صورت دستی و اتوماتیک قابل برنامه‌ریزی است که در حالت دستی توسط کاربر و بر مبنای تعداد سطوح منطقی مشخص می‌شود و در حالت اتوماتیک با استفاده از نشانه‌گذاری بافرهای ابتدا و انتهای هر بلوک) و خروجی این قسمت فایل‌های توصیف سختافزاری جزئی‌تر است که هر قسمت از مدار کلی را نشان می‌دهند. در ادامه این فایل‌های توصیف سختافزاری به همراه فایل کتابخانه هر تکنولوژی وارد سنتزکننده شده و خروجی سنتزکننده به صورت جداگانه در بردارهای مختلف ذخیره می‌شود. از طرف دیگر هریک از این فایل‌های توصیف سخت افزار وارد بلوک استخراج مشخصات شده و مشخصات آن‌ها نیز در جدول‌های ذکر شده ذخیره می‌شود.

بخش تصمیم‌گیرنده سیستم که در واقع متشکل از ابزارهای یادگیری ماشین است به صورت دیاگرامی در شکل ۸,۳ آمده است. در این بخش از بردارهای ایجاد شده در بخش قبل استفاده شده و سیستم آموزش می‌بیند. نوع این آموزش از نوع نظارت‌شده بوده و از ساختارهای منطقی که منطق بهینه آن‌ها با تقریب قابل قبولی مشخص است برای ایجاد الگو استفاده می‌شود. خروجی سیستم تصمیم‌گیرنده منطق

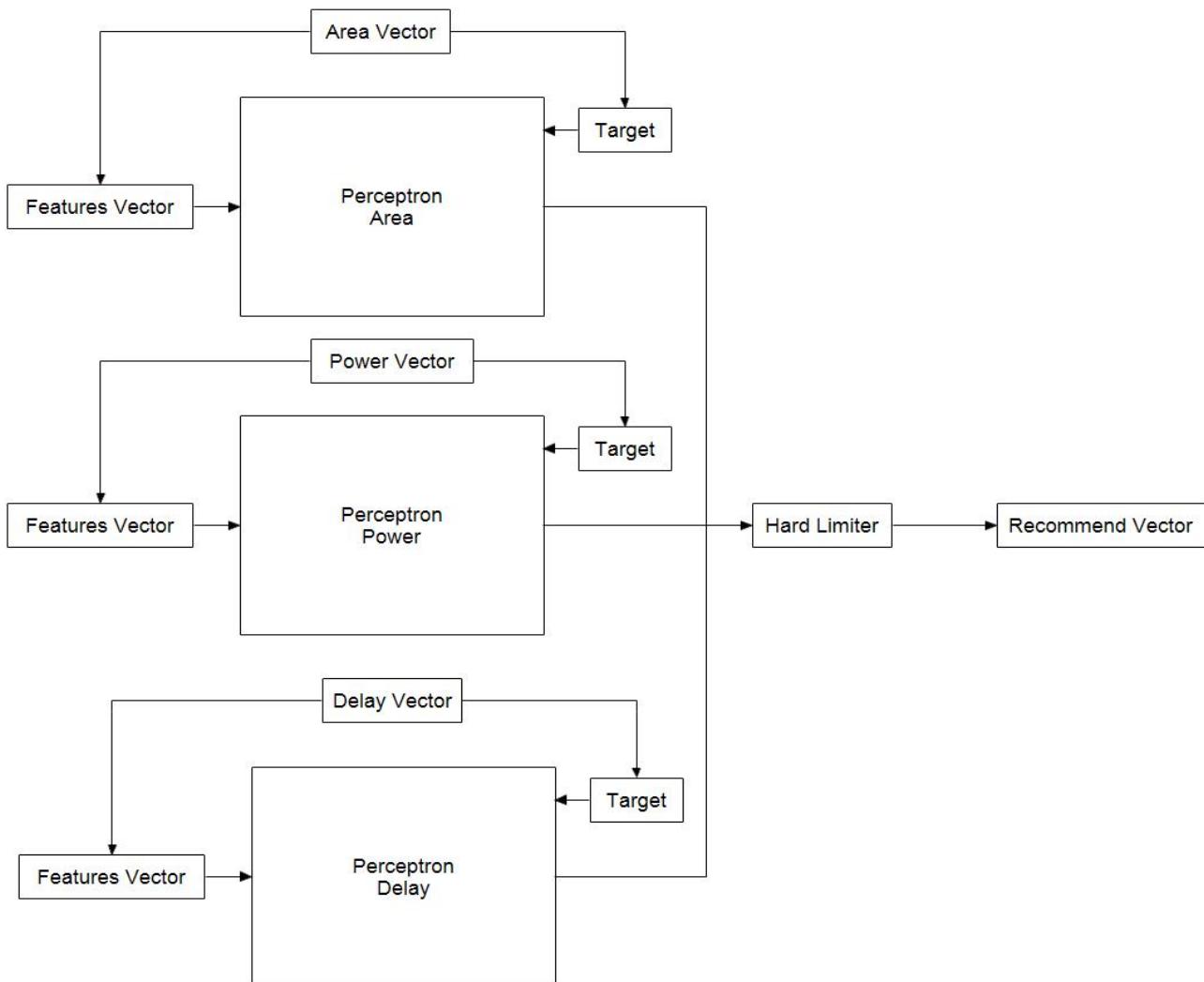
برنده را در هر مرحله برای هریک از ۳ پارامتر مساحت، توان و تاخیر به صورت جداگانه مشخص می‌کند.
 (مراحل سیستم آموزش و نحوه ارزیابی آن در فصل‌های بعدی مورد بحث قرار گرفته‌اند).



شکل ۶-۳۳ دیاگرام کلی سیستم



شکل ۷-۳ دیاگرام بلوکی بخش سنتزکننده و شبیه‌ساز



شكل ۸-۳ دیاگرام بلوکی بخش تصمیم‌گیرنده

۴

فصل چهارم

پیاده‌سازی

پیاده‌سازی

در این فصل پیاده‌سازی سیستم توصیف شده در پایان فصل قبل را شرح می‌دهیم. برای پیاده‌سازی سیستم مذکور در این پروژه از زبان برنامه‌نویسی پایتون ورژن ۳،۴ استفاده شده است. دلیل اصلی استفاده از این زبان برنامه‌نویسی توانایی بالای آن در کار با رشته‌های متند و سادگی اسکریپت‌نویسی می‌باشد.

۱.۴ مازول‌ها و کتابخانه‌ها

در این برنامه سعی شده است تا حد امکان از مازول‌های زبان برنامه‌نویسی پایتون که به صورت پیش فرض همراه آن نصب می‌شود استفاده شود. لیست مازول‌های استفاده شده در این برنامه در جدول ۱،۴ آمده است.

جدول ۱-۴ مازول‌های بکار رفته در برنامه

توضیحات	اسم مازول
به منظور کار با زمان و اندازه‌گیری عملکرد زمانی	time
به منظور کار کردن با رشته‌ها	string
به منظور استفاده از امکانات تقویم و ساعت سیستم	datetime
استفاده از regular expression برای جستجو	re
استفاده از توزیع‌های تصادفی مختلف	random
استفاده از ابزارهای کنترلی سیستم عامل	os

برای پیاده‌سازی الگوریتم‌های هوش محاسباتی نیز از کتابخانه **pyBrain** که توانایی اجرای الگوریتم‌های مختلف یادگیری نظارت شده و بدون نظارت را دارا می‌باشد، استفاده شده است.

برای کنترل منابع برنامه نیز از کنترل نسخه گیت^{۴۳} استفاده شده و تمامی سورس کدها در پیوست ۳ آمده است.

۲.۴ شی اصلی

هسته اصلی برنامه شی^{۴۴} مرکزی **VLSI** است که سایر توابع و متدهای برنامه بر روی آن تعریف می‌شود. این شی که در فایل **VLSI.py** تعریف شده است یک عبارت منطقی را به صورت یک رشته در ورودی دریافت کرده و پس از بررسی اعتبار رشته ورودی، مقادیر خالی **func**, **input**, **wire** و **variables** را برای این کلاس ایجاد می‌کند.

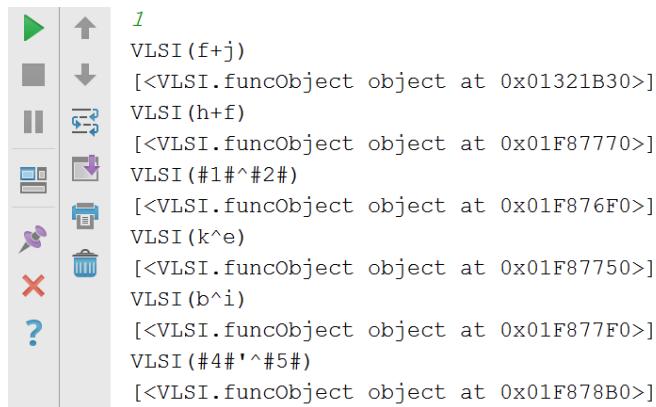
در ادامه با یک فرآیند پویش متن در داخل این کلاس، عبارت منطقی ورودی که به صورت یک رشته بود به یک تابع منطقی قابل فراخوانی تبدیل می‌شود و در مقادیر خالی تولید شده در مرحله قبل ذخیره می‌شوند. این فرآیند پویش بر اساس تعداد پرانتزها در رشته ورودی بوده و تعداد گیت‌های پایه موجود و ترتیب قرارگیری آنها نیز به عنوان ویژگی ذخیر می‌شود. لازم به ذکر است که تمام توابع منطقی پایه جداسازی شده و در فایل **logic.py** در دسترس می‌باشند.

سایر متغیرهای عمومی که در برنامه چندین بار فراخوانی شده‌اند نیز در فایل **globals.py** قرار دارند.

در ادامه این کلاس متدهای **str** و **repr** برای نمایش این شی وجود دارد که در صورت فراخوانی شی یا استفاده از تابع **print** در برنامه از آن‌ها استفاده می‌شود. نمونه‌ای از فراخوانی این شی در برنامه در شکل ۱،۴ آمده است.

⁴³Git Version Control

⁴⁴Object



شکل ۱-۴ فراخوانی شی اصلی

در فایل **VLSI.py** متدهای دیگری برای ساخت فایلهای **verilog** هم وجود دارد که در ادامه این فصل توضیح داده خواهد شد.

۳.۴ تجزیه‌کننده

تجزیه‌کننده این پروژه که در فایل **parser.py** قرار داد یک مازول بوده که شامل ۴ تابع اصلی می‌باشد. وظیفه اصلی این مازول تجزیه عبارت منطقی به عبارت‌های مجزا و ذخیره آن‌ها به عنوان یک شی منطقی کوچکتر در یک بردار، تشکیل جدول درستی و چاپ آن می‌باشد.

تابع **parse_string** با استفاده از روش‌های پایش متن رشته ورودی را با توجه به پرانتزها به قسمت‌های کوچکتر تقسیم کرده هر کدام از آن‌ها را به عنوان یک شی در یک لیست ذخیر می‌کند.

تابع **make_table** ابتدا تعداد ورودی‌های عبارت را خوانده سپس یک جدول درستی خالی برای آن تابع درست می‌کند. در مرحله بعدی با فراخوانی تابع **calculate_result** جدول را به صورت مناسب تکمیل می‌کند.

تابع **calculate_result** مقدار خروجی هر یک از عبارت‌های منطقی را محاسبه کرده و بر می‌گرداند که در تابع **make_table** از این تابع برای پر کردن جدول استفاده می‌شود.

تابع **print_result** جدول را چاپ می‌کند. قسمتی از خروجی یک جدول فرضی در شکل ۲.۴ آمده است.

fjhkebi	out
0000000	0
1000000	0
0100000	1
1100000	0
0010000	1
1010000	0
0110000	0
1110000	0
0001000	0
1001000	0
0101000	1
1101000	0
0011000	1
1011000	0
0111000	0
1111000	0
0000100	0
1000100	0
0100100	1
1100100	0
0010100	1
1010100	0
0110100	0

شکل ۲-۴ قسمتی از یک جدول درستی ساخته شده

۴.۴ تولید کننده داده آموزش

از آنجا که برای آموزش و ارزیابی سیستم به مدارهای متنوع با انواع تعداد گیت و ترتیب های مختلف نیاز است. مأژولی برای تولید تصادفی مدارهای ترکیبی نوشته شده است. این برنامه که در فایل قرار دارد دارای دو تابع اصلی است.

-۱ **gate_gen** : این تابع تعداد گیت‌های مورد نظر و تعداد متغیرهای تابع را به عنوان ورودی دریافت کرده و به صورت تصادفی با توزیع نرمال گیت‌های مختلف را تولید می‌کند و با احتمال برابر پشت هر عبارت علامت **NOT** قرار می‌دهد (در صورتی که از این گزینه استفاده شود در حالت پیش فرض این گزینه غیر فعال است) و عبارت‌های خروجی را در یک لیست ذخیره می‌کند.

-۲ **generator** : این تابع یک لیست از گیت‌ها را به عنوان ورودی دریافت می‌کند و به صورت تصادفی با توزیع نرمال این عبارت‌ها را با استفاده از عملگرهای استاندارد به هم پیوند می‌دهد. در این مرحله هم مانند تابع قبل گزینه‌ای برای استفاده از گیت **NOT** به صورت تصادفی قرار دارد.

در آخر این مأژول هم عبارت منطقی تولید شده در یک فایل متنی ذخیر می‌شود. در شکل ۳،۴ نحوه ذخیره‌سازی و در شکل ۴،۴ یک نمونه از فایل‌های تولید شده توسط این مأژول آمده است.

19- VLSI.py

20- __pycache__

```
Please Enter One Of This File For Generate :
15
Please Enter Number Of 2-Input Gates in File : 10
Please Enter Total Number Of Var in File 4
[ '(d.b)', '(a.b)', '(c+d)', '(b^c)', '(a.c)', '(c+d)', '(d+a)', '(b.a)', '(c+a)', '(b.a)' ]
Function : ((c+a)^((b.a)))^(((d.b).(a.b)).((c+d).((b^c)))) (((a.c).((c+d))^((d+a).((b.a)))))

Done!!
```

شکل ۳-۴ نحوه انتخاب فایل و ذخیره‌سازی فایل‌های نمونه

```
((c+a)^(b.a))^(((d.b)'.(a.b)).((c+d).(b^c))).(((a.c).(c+d))^(d+a).(b.a)))
```

شکل ۴-۴ یک نمونه از فایل‌های تولید شده توسط مژول **generator.py**

۵.۴ متدهای تولید فایل **Verilog**

این متدهای تولید فایل **Verilog** در کلاس **VLSI** قرار دارد و یکی از متدهای شی اصلی به حساب می‌آید. به این صورت که با فراخوانی این متدها شی مورد نظر با ویژگی‌های خود تبدیل به یک فایل **Verilog** با فرمت **.v** شده و در مسیر مورد نظر ذخیره می‌شود. نحوه تبدیل عبارت‌های منطقی درون شی **VLSI** به زبان توصیف سخت افزار **Verilog** از نوع جریان داده‌ای می‌باشد.

نحوه نام‌گذاری متغیرها، سیم‌ها، خروجی‌ها و ورودی‌ها همگی به صورت تصادفی بوده و استفاده اصلی این متدها در تبدیل عبارت‌های بولی به فایل‌های **Verilog** و همچنین شکستن یک فایل **Verilog** کلی به قسمت‌های مختلف است.

نمونه‌ای از فایل تولید شده توسط این برنامه در شکل ۵.۴ آمده است.

```

1 module F1 (a , b , VV1V);
2   input a , b;
3   output VV1V;
4   or f0 (VV1V , a , b);
5 endmodule
6
7

```

شکل ۵-۴ نمونه ای از فایل Verilog تولید شده توسط make_verilog

۶.۴ تولید اسکریپت برای سنتز کننده

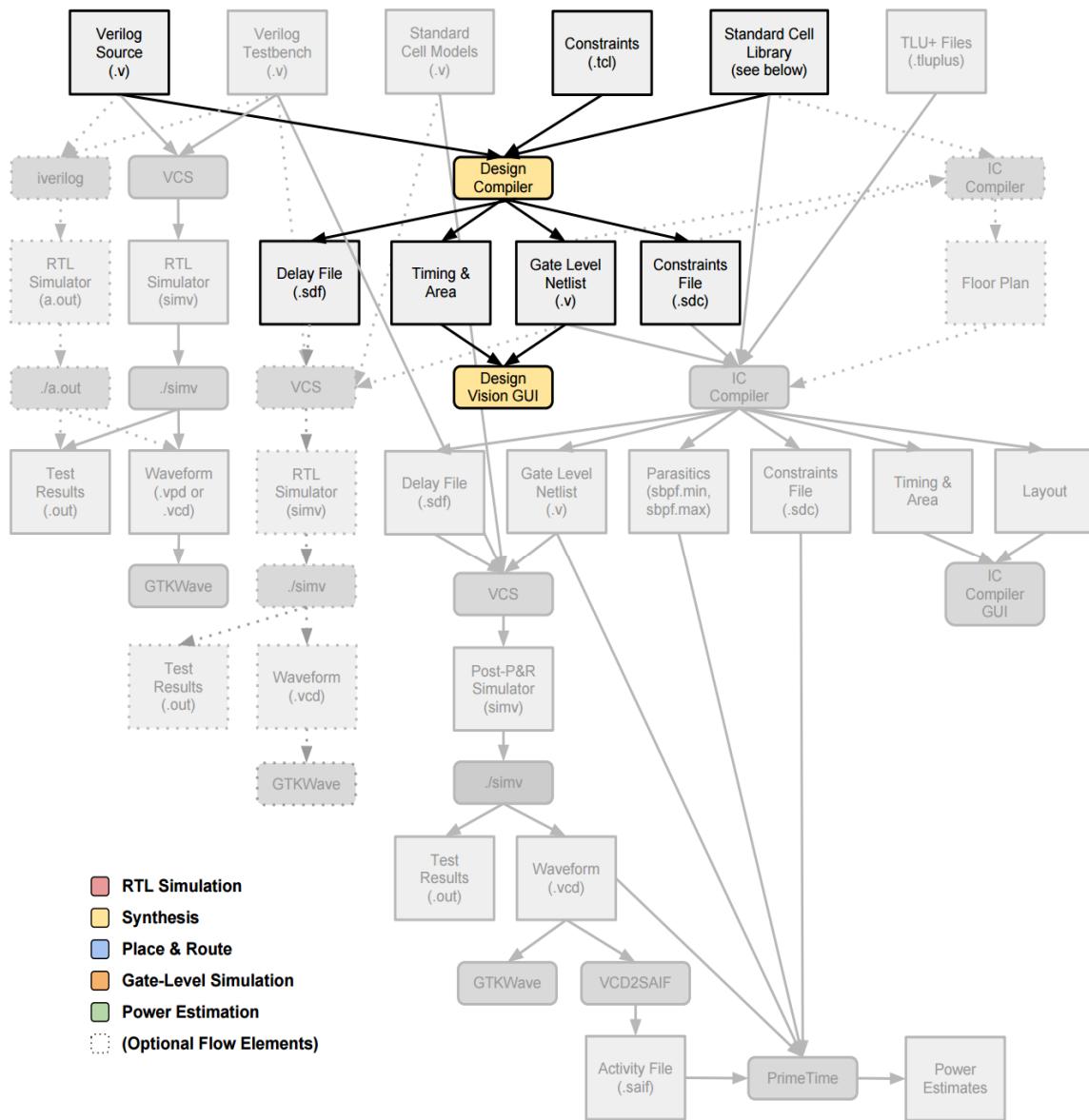
در این پروژه برای سنتز از ابزار SDC^{۴۵} استفاده شده است. مراحل مختلف برای دستیابی به خروجی سنتز کننده در شکل ۶,۴ آمده است.

برای ارتباط برقرار کردن با این ابزار جهت سنتز کدهای نوشته شده به زبان Verilog روش‌های مختلفی وجود دارد که یکی از آن‌ها استفاده از اسکریپت برای پیاده سازی دستورات از طریق خط فرمان UNIX^{۴۶} می‌باشد. برای این نیاز به اسکریپت‌هایی با فرمت src. است که در آن محل ذخیره خروجی‌های (توان، تاخیر و مساحت)، محل فایل Verilog و چند پارامتر دیگر باید تنظیم شود.

برای این موضوع تابعی به اسم make_script_file که در فایل functions.py قرار دارد نوشته شده است که این تابع با یک حلقه بررسی تمام اعضای لیست اشیای VLSI ذخیره شده اعمال می‌شود و فایل‌هایی با فرمت src. به صورت مجزا برای هر کدام از آن‌ها تولید می‌کند.

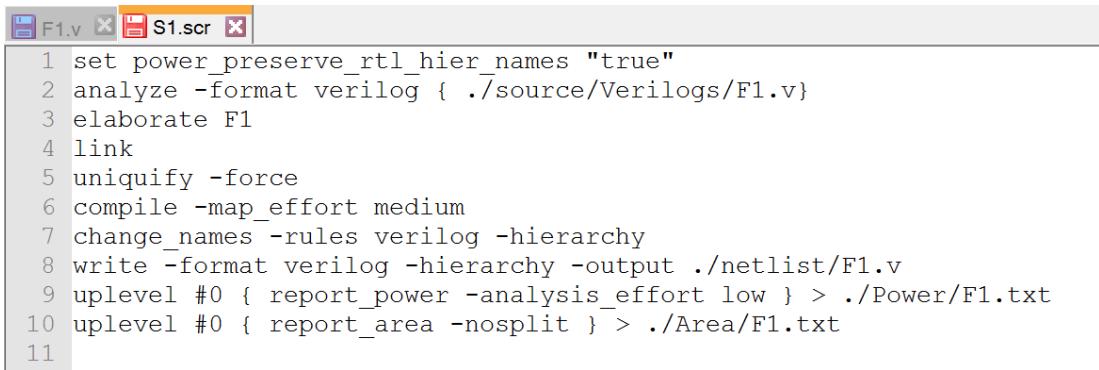
⁴⁵Synopsys Design Compiler

⁴⁶UNIX Command Line



[۱۷] شکل ۶-۴ مراحل مختلف استفاده از Design Compiler

نمونه ای از اسکریپت‌های تولید شده توسط این تابع در شکل ۷.۴ آمده است :



```

1 set power_preserve_rtl_hier_names "true"
2 analyze -format verilog { ./source/Verilogs/F1.v}
3 elaborate F1
4 link
5 uniquify -force
6 compile -map_effort medium
7 change_names -rules verilog -hierarchy
8 write -format verilog -hierarchy -output ./netlist/F1.v
9 uplevel #0 { report_power -analysis_effort low } > ./Power/F1.txt
10 uplevel #0 { report_area -nosplit } > ./Area/F1.txt
11

```

شکل ۷-۴ نمونه ای از اسکریپت تولید شده توسط **make_script_file**

۷.۴ سایر توابع

چند تابع دیگر نیز در این برنامه به صورت جانبی استفاده شده است که همگی آنها در فایل **functions.py** قرار دارند که در زیر عملکرد هر کدام را به صورت خلاصه شرح می‌دهیم :

input_num: این تابع یک رشته را به عنوان ورودی دریافت می‌کند که در واقع یک عبارت منطقی است که به صورت یک رشته بیان شده است، با استفاده از یک حلقه و چند شرط تعداد متغیرهای بکار رفته در این عبارت منطقی را محاسبه و ذخیره می‌کند.

input_op: این تابع نیز یک رشته را به عنوان ورودی دریافت کرده که در واقع یک عبارت منطقی است که به صورت یک رشته بیان شده است، با استفاده از یک حلقه و چند شرط تعداد هر یک از عملگرهای منطقی پایه به کار رفته در رشته را محاسبه و ذخیره می‌کند.

check_valid: این تابع یک رشته را به عنوان ورودی گرفته و تشخیص می‌دهد که رشته وارد شده یک عبارت منطقی معتبر هست یا خیر و خروجی آن نیز به صورت عبارت منطقی می‌باشد.

۸.۴ ارزیابی زمانی

برای ارزیابی و بهینه‌سازی زمانی قسمت‌های مختلف سیستم، برنامه‌ای با استفاده از تایمر عملکردی^{۴۷} زبان برنامه نویسی پایتون نوشته شده است که در آن قسمت‌های مختلف برنامه به صورت ترتیبی^{۴۸} فراخوانی می‌شوند و زمانی که **CPU** برای اجرای هر بخش صرف کرده است محاسبه و به همراه مشخصات سیستم مثل سیستم عامل، پردازنده و ... در یک فایل متند ذخیر می‌شود.

این برنامه در فایل **time_test.py** قرار دارد و نمونه‌ای از فایل تولید شده توسط این ماژول در شکل ۸.۴ آمده است.

^{۴۷}Performance Timer

^{۴۸}Sequential

Table 1 - Result	
1	Test_Case1.txt : ,Input Var Number: 7 Input Operation Number: 8 Elapsed Time: 0.2106578520709678 Sec 2015-09-13 22:44:00.539153 CPU: Intel® Family 6 Model 55 Stepping 0, GenuineIntel
2	Test_Case3.txt : ,Input Var Number: 5 Input Operation Number: 401 Elapsed Time: 11.187543079000178 Sec 2015-09-23 10:54:08.885156 CPU: i3_86
3	Manual Function : ,Input Var Number: 3 Input Operation Number: 3 Elapsed Time: 0.00730027799947366 Sec 2015-09-23 11:26:21.435842 CPU: i3_86
4	Manual Function : ,Input Var Number: 3 Input Operation Number: 3 Elapsed Time: 0.00730027799947366 Sec 2015-09-23 11:29:39.752115 CPU: i3_86
5	Manual Function : ,Input Var Number: 3 Input Operation Number: 3 Elapsed Time: 0.00812018899970284 Sec 2015-09-23 11:30:47.560209 CPU: i3_86
6	Manual Function : ,Input Var Number: 3 Input Operation Number: 3 Elapsed Time: 0.007468150011444155 Sec 2015-09-23 11:33:52.875276 CPU: i3_86
7	Manual Function : ,Input Var Number: 3 Input Operation Number: 3 Elapsed Time: 0.0037121890010591596 Sec 2015-09-23 12:54:45.057027 CPU: i3_86
8	Manual Function : ,Input Var Number: 3 Input Operation Number: 2 Elapsed Time: 0.0058966230000573385 Sec 2015-09-27 16:49:24.345252 CPU: i3_86
9	Manual Function : ,Input Var Number: 3 Input Operation Number: 2 Elapsed Time: 0.264411511999559 Sec 2015-09-27 16:55:16.375510 CPU: i3_86
10	Manual Function : ,Input Var Number: 3 Input Operation Number: 2 Elapsed Time: 0.004894009000054211 Sec 2015-09-27 16:57:42.739605 CPU: i3_86
11	Manual Function : ,Input Var Number: 3 Input Operation Number: 2 Elapsed Time: 0.005669921999874059 Sec 2015-09-27 16:57:58.98379 CPU: i3_86
12	Manual Function : ,Input Var Number: 3 Input Operation Number: 2 Elapsed Time: 0.006018648010033645 Sec 2015-09-27 16:59:31.570966 CPU: i3_86
13	Manual Function : ,Input Var Number: 3 Input Operation Number: 2 Elapsed Time: 0.004960340000250986 Sec 2015-09-27 17:00:48.024822 CPU: i3_86
14	Manual Function : ,Input Var Number: 3 Input Operation Number: 2 Elapsed Time: 0.0053166300119187 Sec 2015-09-27 17:03:15.896470 CPU: i3_86
15	Manual Function : ,Input Var Number: 3 Input Operation Number: 2 Elapsed Time: 0.0049628500146949 Sec 2015-09-27 17:04:13.297183 CPU: i3_86
16	Manual Function : ,Input Var Number: 3 Input Operation Number: 2 Elapsed Time: 0.00471365099994561 Sec 2015-09-27 17:05:22.723448 CPU: i3_86
17	Manual Function : ,Input Var Number: 3 Input Operation Number: 2 Elapsed Time: 0.0050125300004927 Sec 2015-09-27 17:06:45.892209 CPU: i3_86
18	Manual Function : ,Input Var Number: 3 Input Operation Number: 2 Elapsed Time: 0.00564525200022083 Sec 2015-09-27 17:08:28.123799 CPU: i3_86
19	Manual Function : ,Input Var Number: 3 Input Operation Number: 2 Elapsed Time: 0.00560990299982225 Sec 2015-09-27 17:10:58.859123 CPU: i3_86
20	Manual Function : ,Input Var Number: 3 Input Operation Number: 2 Elapsed Time: 0.01512707998322 Sec 2015-09-27 17:11:52.943967 CPU: i3_86

شکل ۴-۸ نمونه‌ای از فایل‌های تولید شده توسط برنامه time_test.py

۹.۴ برنامه آموزش شبکه

همان‌طور که در بخش اول این فصل اشاره شد، برای بخش پردازشی و شبکه عصبی این پروژه از کتابخانه **pybrain** که یک کتابخانه‌ی متن باز می‌باشد استفاده شده است. این کتابخانه توان پیاده‌سازی روش‌های مختلف یادگیری ماشین را دارد، که در این پروژه از پرسپترون‌های چند طبقه-چند لایه استفاده شده است.^[۱۸]

در ادامه نحوه تولید شبکه و بردار داده‌های آموزشی و ارزیابی با استفاده از این کتابخانه آمده است.

۱.۹.۴ نصب کتابخانه

همان‌طور که گفته شد این کتابخانه متن باز بوده و سورس کدهای آن بر روی **github** قرار دارد برای نصب این کتابخانه کافی است مخزن کد^{۴۹} آن بر روی سرور را با استفاده از دستور **clone** استخراج کنیم.

```
$ git clone git://github.com/pybrain/pybrain.git
$ python setup.py install
```

شکل ۹-۴ روش **clone** و نصب کتابخانه **pybrain**^[۱۸]

۲.۹.۴ ساخت شبکه عصبی

همان‌طور که در شکل ۱۰،۴ نشان داده شده برای ساخت شبکه عصبی کافی است از میانبرهای تعریف شده در این کتابخانه برای این کار استفاده کنیم.

```
from pybrain.tools.shortcuts import buildNetwork
from pybrain.structure import TanhLayer
from pybrain.structure import SoftmaxLayer
net=buildNetwork(3,3,3,hiddenclass=TanhLayer, outclass=SoftmaxLayer)
```

شکل ۱۰-۴ روش ساخت شبکه در **pybrain**

⁴⁹Code Repository

اعداد وارد شده در تابع ساخت شبکه به ترتیب به معنای تعداد نرون‌های لایه ورودی، لایه پنهان و لایه خروجی است. در ادامه نیز نوع تابع‌های فعال سازی هر لایه مشخص شده است. در این پژوهه تابع فعال سازی لایه پنهان از نوع تانژانت هیپربولیک و لایه خروجی از نوع محدود کننده نرم^{۵۰} می‌باشد که خروجی را به صورت معیار احتمالی نشان می‌دهد.

۳.۹.۴ تعریف داده‌های ورودی

برای تعریف داده‌های ورودی نیز به راحتی می‌توان از میانبرهای کتابخانه استفاده نمود. همان طور که در شکل ۱۱,۴ نشان داده شده است باید تعداد ورودی‌ها و خروجی‌ها را در تابع میانبر مشخص کنیم. در این مثال ورودی یک بردار دوتایی و خروجی یک بردار ۳ تایی است.

```
from pybrain.datasets import SupervisedDataSet
ds = SupervisedDataSet(2, 3)
ds.addSample((0, 1), (1,0,0))
ds.addSample((1, 0), (1,1,0))
ds.addSample((1, 1), (0,0,1))
```

شکل ۱۱-۴ روش تعریف داده‌ای ورودی در **pybrain**

۴.۹.۴ اعمال الگوریتم‌های یادگیری

پس از ساخت شبکه و تعریف داده‌های ورودی آخرین مرحله آموزش شبکه برروی این داده‌ها می‌باشد که برای این مرحله نیز مانند دو مرحله‌ی قبل از میانبرهای کتابخانه استفاده می‌کنیم. روش‌های مختلفی در این کتابخانه برای یادگیری ماشین وجود دارد که در این پژوهه همان‌طور که قبلاً هم اشاره شد از روش پس انتشار خطأ استفاده می‌شود.

چگونگی فرآیند آموزش در شکل ۱۲,۴ آمده است.

⁵⁰Soft Limiter Function

```
from pybrain.supervised.trainers import BackpropTrainer
net = buildNetwork(2, 3, 3, bias=True, hiddenclass=TanhLayer)
trainer.train()
trainer.trainUntilConvergence()
```

شکل ۱۲-۴ روش آموزش شبکه های عصبی در **pybrain**

۵.۹.۴ توقف آموزش

برای توقف آموزش راههای بسیار زیادی وجود دارد که در فصل قبلی به آنها اشاره شد. در کتابخانه **pybrain** میانبرهایی برای این کار وجود دارد که به صورت استاندارد داده‌ها را به توجه به حجم آنها دسته بندی می‌کنند. در صورت زیاد بودن داده‌های ورودی ۶۰٪ از آنها برای آموزش، ۲۰٪ برای آزمون و ۲۰٪ هم برای اعتبارسنجی استفاده می‌شود. در صورتی که داده‌ها کم باشند قسمت اعتبارسنجی حذف شده و ۷۰٪ داده‌ها برای آموزش و ۳۰٪ باقی مانده برای آزمون استفاده می‌شود. [۱۸]

همان‌طور که در شکل ۱۲-۴ هم آمده است متند میانبری به نام **trainUntilConverge** وجود دارد که تا برآورده شدن شرط توقف، آموزش را ادامه می‌دهد. [۱۸]

۵

فصل پنجم

نتایج و جمعبندی

در فصل پیش چگونگی پیاده‌سازی سیستم توصیه گر منطق در زبان برنامه‌نویسی پایتون و قسمت‌های مختلف آن مورد بحث قرار گرفت.

در این فصل در ابتدا در مورد نحوه ارزیابی سیستم و سپس در مورد مشکلات و روش‌های پیشنهادی برای برطرف کردن آن‌ها توضیحاتی ارائه خواهد شد.

۱.۵ ارزیابی سیستم

برای ساخت بردارهای ورودی سیستم از ساختارهای منطقی که منطق بهینه آن‌ها با تقریب قابل قبولی مشخص است استفاده شده است. برای طبقه‌بندی منطق‌ها هم با توجه به مشکلات بسیار زیاد تولید کتابخانه‌های مختلف که یک کار کاملا صنعتی می‌باشد از دو کتابخانه موجود در ابعاد ۹۰ نانومتر استفاده شده است. که یکی از آن‌ها تکنولوژی CMOS ایستا و دیگری یک تکنولوژی مبتنی بر مدارهای کم‌توان است. در جدول ۱.۵ مقایسه برخی از توابع پایه برای این دو منطق آمده است.

جدول ۱-۵ مقایسه برخی از توابع پایه در منطق‌های GDI و CMOS ایستا

		GDI		CMOS	
		Power(uW)	Delay(nsec)	Power(uW)	Delay(nsec)
MUX	$\bar{A}B+AC$	35.7	1.1	49.7	2.1
OR	$A+B$	26.3	1.2	32.9	1.7
AND	AB	25.7	0.9	34.1	1.4
F1	$\bar{A}B$	31.2	0.8	45.2	1.5
F2	$\bar{A}+B$	32.0	1.3	43.1	1.9

طول بردار آموزش ۱۰۰۰ نمونه و طول بردار آزمون ۲۰۰ نمونه (۱۰۰ نمونه به ازای هر کلاس) می‌باشد.

برای ارزیابی قدرت تفکیک و کلاس بندی سیستم از روش ماتریس درهم‌ریختگی^{۵۱} استفاده شده است. ماتریس درهم‌ریختگی به ماتریسی گفته می‌شود که در آن عملکرد الگوریتم‌های مربوطه را نشان می‌دهند. هر ستون ماتریس، نمونه‌ای از مقدار پیش‌بینی^{۵۲} را نشان می‌دهد. در صورتی که هر سطر نمونه‌ای واقعی^{۵۳} را در بردارد. در این ماتریس شاخص‌های دقت^{۵۴}، صحت^{۵۵} و چند معیار دیگر برای هریک از کلاس‌ها محاسبه و بررسی می‌شوند.

ماتریس درهم‌ریختگی این سیستم برای ارزیابی بروی ۲۰۰ نمونه برای هریک از پارامترهای بهینه‌سازی (توان، مساحت و تاخیر) به ترتیب در جدول‌های ۱,۵ و ۲,۵ و ۳,۵ آمده است.

جدول ۲-۵ ماتریس درهم‌ریختگی سیستم برای پارامتر توان

		Predicted	
		AUTLib	SCMOS
Actual Class	AUTLib	72	28
	SCMOS	37	63

جدول ۳-۵۵ ماتریس درهم‌ریختگی سیستم برای پارامتر مساحت

		Predicted	
		AUTLib	SCMOS
Actual Class	AUTLib	۹۴	۶
	SCMOS	۱۳	۸۷

^{۵۱}Confusion Matrix

^{۵۲}Predicted Value

^{۵۳}Actual Value

^{۵۴}Precision

^{۵۵}Recall

جدول ۴-۵ ماتریس درهم‌ریختگی سیستم برای پارامتر تاخیر

		Predicted	
		AUTLib	SCMOS
Actual Class	AUTLib	۸۰	۲۰
	SCMOS	۳۹	۶۱

در ادامه محاسبات مربوط به صحت و دقت هر کلاس طبقه‌بندی آمده است :

پارامترهای ارزیابی توان :

$$R_{AUTLib} = SPC_{SCMOS} = \frac{72}{72 + 28} = 0.72$$

$$P_{AUTLib} = \frac{72}{72 + 37} = 0.66$$

$$R_{SCMOS} = SPC_{AUTLib} = \frac{63}{63 + 37} = 0.63$$

$$P_{SCMOS} = \frac{63}{63 + 28} = 0.69$$

پارامترهای ارزیابی مساحت :

$$R_{AUTLib} = SPC_{SCMOS} = \frac{94}{94+6} = 0.94$$

$$P_{AUTLib} = \frac{94}{94+13} = 0.87$$

$$R_{SCMOS} = SPC_{AUTLib} = \frac{87}{87+13} = 0.87$$

$$P_{SCMOS} = \frac{87}{87+6} = 0.93$$

پارامترهای ارزیابی تاخیر :

$$R_{AUTLib} = SPC_{SCMOS} = \frac{80}{80+20} = 0.80$$

$$P_{AUTLib} = \frac{80}{80+39} = 0.67$$

$$R_{SCMOS} = SPC_{AUTLib} = \frac{61}{61+39} = 0.61$$

$$P_{SCMOS} = \frac{61}{61+20} = 0.75$$

سایر پارامترهای ارزیابی نیز در ادامه محاسبه شده است :

$$FNR_{AUTLib}(Power) = \frac{28}{28+72} = 0.28, FNR_{AUTLib}(Area) = \frac{6}{6+94} = 0.06, FNR_{AUTLib}(Delay) = \frac{20}{20+80} = 0.20$$

$$FNR_{SCMOS}(Power) = \frac{37}{37+63} = 0.37, FNR_{SCMOS}(Area) = \frac{13}{13+87} = 0.13, FNR_{SCMOS}(Delay) = \frac{39}{39+61} = 0.39$$

$$F1_{AUTLib}(Power) = \frac{2 \times 0.72 \times 0.66}{0.66 + 0.72} = 0.69, F1_{AUTLib}(Area) = \frac{2 \times 0.94 \times 0.87}{0.94 + 0.87} = 0.90, F1_{AUTLib}(Delay) = \frac{2 \times 0.80 \times 0.67}{0.67 + 0.80} = 0.73$$

$$F1_{SCMOS}(Power) = \frac{2 \times 0.63 \times 0.69}{0.63 + 0.69} = 0.66, F1_{SCMOS}(Area) = \frac{2 \times 0.87 \times 0.93}{0.87 + 0.93} = 0.94, F1_{SCMOS}(Delay) = \frac{2 \times 0.61 \times 0.75}{0.61 + 0.75} = 0.67$$

در ادامه ۳ تا از مهمترین پارامترهای ذکر شده را به صورت مختصر تفسیر می‌کنیم.

صحت : که با نماد **R** نشان داده می‌شود بیانگر میزان تشخیص درست در طبقه‌بندی آن کلاس نسبت به واقعیت می‌باشد که به آن حساست هم گفته می‌شود. [۱۹]

دقت : که با نماد **P** نشان داده می‌شود بیانگر میزان تشخیص درست در طبقه‌بندی آن کلاس نسبت به طبقه‌های تخمینی می‌باشد. [۱۹]

فاکتور **F1** : میانگین همساز بین دقت و صحت می‌باشد که در واقع یک فاکتور برای ترکیب این دو پارامتر به شمار می‌رود. [۱۹]

۲.۵ جمع‌بندی و پیشنهادها

همان‌طور که در بخش قبل مشاهده شد این سیستم از نظر آماری در تشخیص الگوهای تاخیر چندان قوی نیست(با نرخ خطای ۲۰ درصد برای کتابخانه کم توان و ۳۹ درصد برای کتابخانه CMOS ایست) ولی در پارامترهای مساحت(دقت و صحت بالای ۸۵ درصد در هردو الگو) و تا حدودی توان(صحت بالای ۷۰ درصد برای کتابخانه کم توان) نتایج قابل قبول است. مشکلات و نقایص اصلی این پروژه را می‌توان در چند مورد برشمرد که هریک می‌توانند زمینه‌ی تعریف پروژه‌هایی دیگر را فراهم آورند.

۱ - بزرگترین مشکل سیستم حال حاضر کم بودن کتابخانه‌های دردسترس برای آموزش شبکه است. از آنجایی که هریک از منطق‌های نام برده شده در فصل اول دارای مشخصات متفاوتی هستند. نبود هریک از آن‌ها باعث کاهش توانایی سیستم در طبقه‌بندی الگوها می‌شود.

۲ - با توجه به غیرخطی بودن فضای الگوها و همچنین همبستگی^{۵۶} بسیار بالا بین آنها انتظار می‌رود استفاده از روش‌های مقاوم‌تر مانند ماشین بردار پشتیبانی^{۵۷} و یا استفاده از مفاهیم یادگیری عمیق^{۵۸} و شبکه‌های عصبی عمیق^{۵۹} بتوان عملکرد سیستم را بهبود ببخشد.

۳ - در این پروژه تنها از دو مشخصه مدار به عنوان ویژگی استفاده شده است در حالی که می‌توان ویژگی‌های مستقل دیگری نیز از مدار استخراج کرد که فضای الگوها را بهتر پوشش دهند.

^{۵۶}Correlation

^{۵۷}Support Vector Machine

^{۵۸}Deep Learning

^{۵۹}Deep Neural Network

۶ منابع و مأخذ

[۱] پورمظفری، سعادت، هرندي زاده، حامد؛ طراحی مدارات VLSI، انتشارات مدرسان شریف

تهران، ویرایش اول، ۱۳۹۲

- [۲] N.Weste, D.Harris; *CMOS VLSI Design: A Circuits and Systems Perspective*, 3rd Edition, Pearson,2004.

- [۳] R.Singh; *Trends in VLSI Design: Methodologies and CAD Tools*, IC Design Group CEERI, Pilani.

- [۴] T.A.Demassa, Z.Ciccone; *Digital Integrated Circuits*, 1st Edition, John Wiley and Sons, 1996.

- [۵] J. M. Rabaey, A.Chandrakassan, B.Nikolie; *Digital Integrated Circuits: A Design Perspective*, 2nd Edition, Prentice-Hall, 2003.

- [۶] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya; “A Convex Optimization Approach to Transistor Sizing for CMOS Circuits”, Proc. ICCAD 1992, pp482-485.

[۷] بابایان مشهدی، سمانه؛ بهینه سازی طراحی مدارات مجتمع آنالوگ ولتاژ پایین با توان

صرفی کم بوسیله الگوریتم ژنتیک چند هدفه، پایان نامه کارشناسی ارشد، دانشگاه فردوسی مشهد، ۱۳۸۷

[۸] رستگار پاشکی، الهه؛ طراحی و پیاده‌سازی کتابخانه سلول های استاندارد کم مصرف و سرعت بالا در فناوری ۹۰ نانومتر، پایان نامه کارشناسی ارشد، دانشگاه صنعتی امیرکبیر، ۱۳۹۴

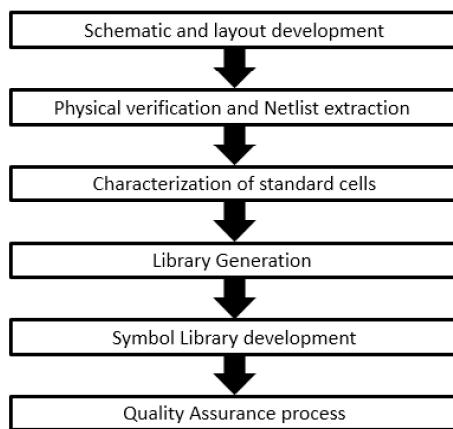
- [۹] Liberty™ NCX User Guide Version F-2011.06, June 2011
<https://docs.google.com/open?id=0B5uiplhpN9bddzRZUGhFQkZDLWc>

- [10] Mitchell, T.; *Machine Learning*, 2nd Edition, McGraw Hill, 1997.
- [11] Bishop, C.M.; *Pattern Recognition and Machine Learning*, 2nd Edition, Springer, 2006.
- [12] Wasserman, Philip; *Advanced Methods in Neural Computing*, 1st Edition, Van Nostrand Reinhold, 1993.
- [13] Yin, Hongfeng; *Perceptron-Based Algorithms and Analysis*, Spectrum Library, Concordia University, Canada, 1996.
- [14] Picton, Phil.; *Neural Network*, 2nd Edition, Palgrave Macmillan, 2001.
- [15] Widrow, B., Lehr,M.A., “30 years of Adaptive Neural Network: Perceptron, Madaline, and Backpropagation,” Proc. IEEE,vol 78,no 9,pp. 1415-1442, (1990).
- [16] Draper, Norman R., Smith, Harry; *Applied regression analysis*, 3rd Edition, New York:Wiley, 1998.
- [17] Lockhart,Derek; RTL-to-Gates Synthesis using Synopsys Design Compiler; ECE5745 Tutorial 2(Version 606ee8a);
www.csl.cornell.edu/courses/ece5745/handouts/ece5745-tut2-dc.pdf
- [18] Tom Schaul, Justin Bayer, Daan Wierstra, Sun Yi, Martin Felder, Frank Sehnke, Thomas Rückstieß, Jürgen Schmidhuber. PyBrain. To appear in:*Journal of Machine Learning Research*, 2010
- Powers, David M W; Evaluation:From Precision, Recall and F-Measure to ROC, Informendess, Markedness & Correlation;
- [19] http://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf

[۸] پیوست ۱

راهنمای ایجاد کتابخانه سلول‌های استاندارد :

روند ایجاد کتابخانه سلول‌های استاندارد در شکل پ-۱ نشان داده شده است که در ادامه به توصیف هریک از این مراحل خواهیم پرداخت.



شکل پ-۱ مراحل تولید کتابخانه سلول‌های استاندارد.^[۹]

(۱) رسم شماتیک و لی‌اوٹ سلول‌ها :

در طراحی کتابخانه سلول‌های استاندارد، اولین مرحله رسم لی‌اوٹ و شماتیک سلول‌های مورد نظر است. رسم لی‌اوٹ سلول‌ها و اجرای تست‌های مختلف روی آن در نرم افزار **virtusu-cadence** صورت می‌گیرد.

لی‌اوٹ هر سلول بر مبنای طرح شماتیک مربوط به آن ساخته می‌شود. در شماتیک هر سلول ویژگی‌های تمامی قطعات و نحوه اتصال آن‌ها نمایش داده شده است.

برای رسم لی اوت، سلول جدیدی با نامی مشابه با شماتیک ایجاد کرده نوع آن را به صورت **virtuoso** تنظیم می کنیم. با اینکار پنجره طراحی لی اوت به همراه پنجره انتخاب لایه^{۶۰} باز می شود.

در **LSW** تمامی لایه های مورد نیاز در پروسه موجود می باشد. سپس قطعات **NMOS** و **PMOS** را مشابه آنچه در شماتیک بوده در پنجره لی اوت قرار می دهیم. حال نوبت به برقراری اتصالات ترانزیستورها است. برای اتصال گیت ترانزیستورها از لایه **poly** استفاده شده و اتصال سایر ترمینال ها از طریق لایه های **metal** صورت میگیرد. در آخرین مرحله از طراحی لی اوت باید **port** ها را وارد نمود. برای این منظور اسم **port** را دقیقا مشابه آنچه در شماتیک تعریف شده، بصورت یک **Table**، وارد کرده و جنس آن را مطابق با لایه فلزی که روی آن واقع شده تنظیم خواهیم کرد.

۲) صحت فیزیکی سلول ها و استخراج نت لیست:

پس از رسم شماتیک و لی اوت سلول ها، ابتدا صحت قواعد طراحی لی اوت با اجرای تست **DRC** مورد ارزیابی قرار گرفته و به دنبال آن با اجرای تست **LVS** تطابق شماتیک و لی اوت رسم شده برای هر سلول بررسی خواهد شد.

در تکنولوژی ۹۰ نانومتر اجرای تست های **LVS** و **DRC** در محیط **caliber** صورت میگیرد. برای اجرای **DRC** در **caliber** به فایل **gds** ساخته شده از روی لی اوت نیاز داریم. این فایل حاوی اطلاعاتی در خصوص لایه های بکار رفته در لی اوت سلول و نحوه قرار گیری این لایه ها نسبت به یکدیگر است. پس از **export** فایل **gds** هر سلول، آن را در بخش ورودی تست **DRC** فراخوانی کرده و سپس آن را اجرا می کنیم. پس از آنکه **DRC** اجرا شد در **calibre** پنجره های موسوم به پنجره خطاهای باز می شود که در آن خطاهای **DRC** نوشته شده است. با کلیک روی این خطاهای محل دقیق خطای روی لی اوت نشان داده می شود. خطاهای مربوط به چگالی و مساحت فلزهای بکار رفته در سلول، در این سطح از طراحی اهمیت نداشته و در طراحی کل تراشه مورد توجه خواهد بود. پس از رفع خطاهای مربوط به **DRC** نوبت به اجرای تست **LVS** میرسد. برای اجرای **LVS** در علاوه بر فایل **gds** به فایل دیگری با پسوند **cdl** نیز نیاز داریم که شماتیک سلول را توصیف

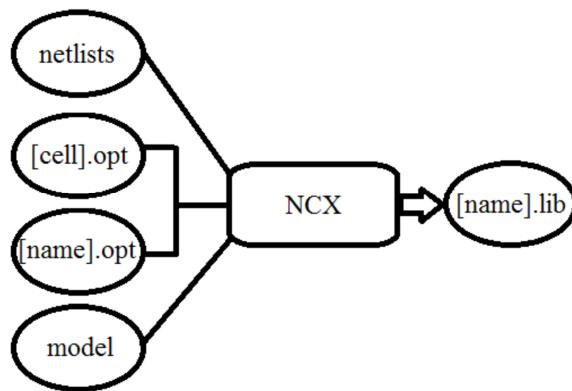
⁶⁰(LSW) Layer Selection Window

میکند. پس از **export** فایل **cdl** از روی شماتیک سلول، این دو فایل را در پنجره **LVS** در بخش ورودی‌ها فراخوانی کرده و سپس آن را اجرا می‌کنیم. در صورت تطابق کامل شماتیک و لی‌او، در پنجره خطای **correct** عبارت **LVS** نمایش داده خواهد شد.

پس از تایید تست‌های **DRC** و **LVS** می‌توان از شبیه‌سازی بعد از لی‌او برروی نت‌لیست‌های استخراج شده از سلول، جهت بررسی دقیق‌تر عملکرد الکتریکی سلول‌های طراحی شده، استفاده کرد. با انجام شبیه‌سازی پس از لی‌او، می‌توان سرعت مدار را تخمین زده و تاثیر مقاومت‌ها و خازن‌های پارازیتی را برروی سلول مشاهده کرد. شبیه‌سازی پس از لی‌او با اجرای **PEX** در **Calibre** صورت می‌گیرد. برای این کار، مشابه **LVS**، به فایل‌های **cdl**، **gds**، **ncd** نیاز داریم. پس از ایجاد فایل‌های مورد نیاز آن‌ها را در بخش ورودی پنجره **PEX** فراخوانی خواهیم کرد. در بخش خروجی **PEX** باید تنظیمات فایل استخراج شده را تعیین کنیم. برای این منظور نوع خروجی را در حالت **HSPICE** تنظیم کرده و در بخش **extraction type** حالت المان‌های پارازیتی را روی **R+C** تنظیم خواهیم نمود. بدین ترتیب با اجرای **PEX**، خروجی بدست آمده شامل نت‌لیست **HSPICE** (sp) سلول همراه با مقاومت‌ها و خازن‌های پارازیتی استخراج شده از لی‌او، طرح خواهد بود.

(۳) مشخصه‌یابی سلول‌های استاندارد و ایجاد کتابخانه با پسوند **.lib** :

در این پژوهه، عملیات مشخصه‌یابی با استفاده از نرم‌افزار **NCX** انجام شده است. مطابق شکل پ-۲ ورودی‌های این برنامه نت‌لیست‌های **HSPICE** استخراج شده از بخش قبل، مدل **TMSC** ترانزیستور های **NMOS** و **PMOS** در تکنولوژی ۹۰ نانومتر و فایل‌هایی با پسوند **.opt** است. خروجی این برنامه نیز یک کتابخانه با پسوند **.lib** است.



شکل پ-۲ ورودی‌ها و خروجی‌های .NCX.

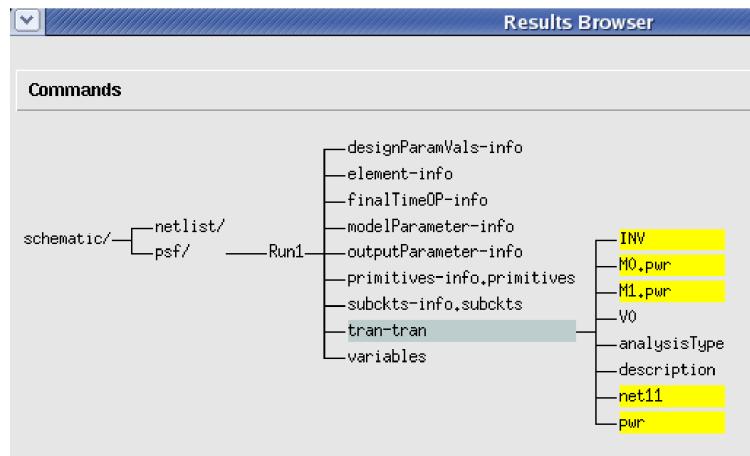
شکل پ-۳ اطلاعات فایل **.opt**. سلول وارونگر را نشان می‌دهد که باید برای هر سلول با نامی مشابه با نت لیست **Hspice** آن ایجاد شود. در این فایل ابتدا باید نوع هر پین (ورودی یا خروجی) تعیین شده و توصیف **Verilog** پین خروجی مشخص شود. همچنین باید میزان توان نشتی هر سلول نیز در این بخش وارد شود.

```

* Generated by Liberty NCX vEngineering.red
scaling_factors : Training_average_factors ;
cell_leakage_power : 127.3000000 ;
pin A {
    input voltage : DC_46 ;
    direction : input ;
}
pin INV {
    output_voltage : DC_47 ;
    direction : output ;
    max_transition : 9.0000000 ;
    function : (A)' ;
    max_capacitance : 10.0000000 ;
    ncx_internal_power_rise_input_transition_time_index : 0 ;
    ncx_internal_power_fall_input_transition_time_index : 0 ;
    ncx_internal_power_rise_total_output_net_capacitance_index : 0 ;
    ncx_internal_power_fall_total_output_net_capacitance_index : 0 ;
}
  
```

شکل پ-۳ محتوای فایل **.opt** در .NCX

محاسبه توان نشتی هر سلول در برنامه **Cadence** انجام شده است. برای محاسبه توان نشتی در **cadence** ابتدا ورودی‌های هر ترانزیستور را در شماتیک سلول طوری تنظیم می‌کنیم که تمامی ترانزیستورها خاموش باشند و پین **VDD** را به منبع ولتاژ **DC** متصل خواهیم نمود. سپس از مسیر **tools>Analog environment** وارد محیط شبیه‌سازی خواهیم شد. از بخش **session** وارد **options** شده و بخش **waveformtools** را در حالت **AWD** قرار می‌دهیم. سپس از بخش **save all outputs** وارد شده و در سطر دوم ذخیره توان‌ها را فعال خواهیم کرد. پس از مرحله از بخش **Analysis** نوع شبیه‌سازی را انتخاب کرده و به دنبال آن **result** را اجرا خواهیم نمود. پس از اجرای برنامه وارد تب **tools** شده و **netlist&run** را انتخاب خواهیم کرد. در این مرحله پنجره‌ای باز شده که حاوی یک آدرس است. این آدرس را بدون تغییر **ok** کرده و وارد پنجره جدیدی، مطابق با شکل ۴ خواهیم شد. با طی مسیر نشان داده شده در شکل زیر، **pwr** را انتخاب کرده و وارد یک **calculator** می‌شویم که در آن توان مصرفی سلول تعريف شده است.



شکل پ-۴ مسیر دسترسی به توان نشتی محاسبه شده.

علاوه بر فایل‌های **.opt**، مربوط به هر سلول، یک فایل **.opt** کلی با نامی مشابه با کتابخانه اصلی نیز وجود دارد که در آن مشخصات کلی کتابخانه نظریه ولتاژهای اعمالی به هر پین، دما و ضرایب ثابت تکنولوژی(**k-factors**) تعريف شده و سلول‌های مختلف کتابخانه در آن فراخوانی می‌شوند. محتويات اين فاييل در شکل پ-۵ و پ-۶ آمده است.

```

* Generated by Liberty NCX vEngineering.red
technology : cmos ;
delay_model : table_lookup ;
date : 25-August-2015 ;
revision : 1.0000000 ;
time_unit : ns ;
leakage_power_unit : 1pW ;
voltage_unit : 1V ;
pulling_resistance_unit : 1kohm ;
current_unit : 1mA ;
capacitive_load_unit : 1.0000000 pf ;
input_threshold_pct_rise : 7.5800000 ;
output_threshold_pct_rise : 50.0000000 ;
input_threshold_pct_fall : 92.4200000 ;
output_threshold_pct_fall : 50.0000000 ;
slew_lower_threshold_pct_rise : 20.0000000 ;
slew_upper_threshold_pct_rise : 80.0000000 ;
slew_lower_threshold_pct_fall : 20.0000000 ;
slew_upper_threshold_pct_fall : 80.0000000 ;
slew_derrate_from_library : 1.0000000 ;
default inout_pin_cap : 0.0100000 ;
default_cell_leakage_power : 0.0000000e+00 ;
default_input_pin_cap : 0.0100000 ;
default_output_pin_cap : 0.0000000e+00 ;
default_famous_load : 1.0000000 ;
nom_voltage : 1.2000000 ;
nom_temperature : 25.0000000 ;
nom_process : 90nm ;
switching_power_split_model : true;
operating_conditions TYPICAL (
  process : 90nm ;
  temperature : 25.0000000 ;
  voltage : 1.2000000 ;
  tree_type : balanced_tree ;
)
* ادامه >>

input_voltage DC_46 {
  vil : 0.435 ;
  vih : 0.693 ;
  vimin : 0.0000000e+00 ;
  vimax : VDD ;
}
output_voltage DC_47 {
  vol : 0.15 ;
  voh : 1.05 ;
  vomin : 0.0000000e+00 ;
  vomax : VDD ;
}

scaling_factors Training average_factors {
  k_temp_cell_fail : -0.0014000 ;
  k_temp_cell_rise : -9.0000000e-04 ;
  k_volt_cell_fail : -0.5947000 ;
  k_volt_cell_rise : -0.5753000 ;
  k_volt_rise_transition : -0.6283000 ;
  k_volt_fail_transition : -0.8772000 ;
  k_temp_fail_transition : -5.0000000e-04 ;
}

include AUTILIBS.Indices :
do (
  AANDNAND
  AF1
  AF2
  AINVBUF
  AMUX
  AORNOR
  AXORNOR
  BANDNAND
  BF1
  BF2
  BINVBUF
  BMUX
  BORNOR
  BXORNOR
)

```

للتازهای بکار رفته در فایل سلولها .opt
مقدار DC با تعیین مشخصه نک
وارونگر در HSPICE تغییر شده است

اندازه های بکار رفته در فایل سلولها .opt
مقادیر پیش فرض
قدر آستانه تغییرات
مقدار آستانه تغییرات
متغیر محاسبه توان با فرمول
0.5CV^{A2}
للتاز استاندارد، دما و تکنولوژی

این فایل حاوی ماتریس های ورودی
عملیات مشخصه پایی است

فرآخوانی سلولهای کتابخانه

شکل پ-۵ محتویات فایل .opt

```

* Generated by Liberty NCX vEngineering.red

ncx_input_transition_time_index : 0.03 0.1 0.4 0.9 1.5 2.2 3 ;
ncx_constrained_pin_transition_index : 0.03 0.9 2.2 ;
ncx_input_net_transition_index : 0.03 0.1 0.4 0.9 1.5 2.2 3;
ncx_related_pin_transition_index : 0.03 0.9 2.2 ;
ncx_total_output_net_capacitance_index : 0.00035 0.021 0.0385 0.084 0.147 0.231 0.3115;

```

شکل پ-۶ محتویات فایل .indices

محیط برنامه **NCX** در شکل پ-۷ نشان داده شده است. با اجرای این برنامه در **Terminal** لینوکس کتابخانه حاصل از مشخصه یابی سلولها با پسوند **.lib**. ایجاد خواهد شد. نرم افزار **NLDM**, **NLPM**, **CCS** قادر به ایجاد مدل های **NCX** برای هر کتابخانه است. در مدل **CCS** اندازه گیری توان با محاسبه جریان ترانزیستور صورت می گیرد در حالی که **NLPM** مستقیماً توان های بدست آمده را در کتابخانه ساخته شده گزارش می کند. همانطور که در شکل

۷ نیز مشاهده می‌شود، در این پروژه از مدل‌های غیرخطی برای ساخت کتابخانه استفاده شده است. جزئیات بیشتر کار با نرم افزار **NCX** در مرجع [۹] آمده است.

```

#!/bin/sh

ncx \
    -output_library AUTLIB90.lib \
    -model_file ../model90.typ \
    -netlist_dir ../netlists \
    -simulator_exec /opt/Synopsys/HSPICE/hspice/linux/hspice \
    -templates true \
    -input_template_dir config \
    -library_template_file AUTLIB90 \
    -driver_waveform_to_library false \
    -ccs_timing false \
    -compact_timing false \
    -timing true \
    -nldm true \
    -ccs_power false \
    -compact_power false \
    -nlpm true \
    -power true \
    -farm_type NoFarm

~
~
~
```

شکل پ-۷ محیط برنامه NCX

۴) ایجاد کتابخانه شبیه‌سازی با پسوند **.db**:

کتابخانه شبیه‌سازی برای تولید مدل‌های منطقی توصیف‌های **Verilog** یا **VHDL** یا **Library Compiler** در کتابخانه سلول‌های استاندارد بکاربرده می‌شود. این کتابخانه توسط ایجاد می‌گردد. ورودی این برنامه فایلی با پسوند **.lib** بوده و خروجی آن نیز کتابخانه‌ای با پسوند **.db** است که در مرحله سنتز قابل استفاده می‌باشد.

برای تولید این کتابخانه، ابتدا **dc_shell** را اجرا کرده و فایل **.lib** را با نوشتن عبارت زیر در ترمینال باز شده، به آن معرفی خواهیم کرد.

dc_shell>read_lib/[address]/file.lib

سپس با نوشتن عبارت زیر در کتابخانه **db**. تولید می شود.

dc_shell>write_lib library –format db –output/[address]/file.db

پیوست ۲

اطلاعات موجود در کتابخانه **.lib**

جدول عب-۱ اطلاعات موجود در فایل **.lib**

library and delay_model	Provide a library name and the delay model to use.
nom_process property	Specifies the reference points for process scaling used for the characterization of the cells (typical, worst and best)
nom_temperature and nom_voltage	Specifies the temperature and voltage reference points
operating_conditions	Defines the process, temperature and voltage values at the library level along with the default_operating_conditions
slew and delay threshold points	Low and high threshold values for slew calculation (10% - 90% points) and the threshold for delay calculations (50% points)
default values for fanout, capacitance, slew	Specifies the limits on maximum input slew on an input pin, input/inout pin capacitance and the maximum output capacitance on any output pin
units	Specifies the units used for time, capacitance, power, voltage, current etc.
Lookup table templates	Define templates of common information to us in lookup tables. These are defined for timing arcs, power and timing checks that will be included in the cell definitions
Cell Definitions	
Cell(cell_name)	The cell name
Area	Specifies the cell area, used during logic synthesis and timing analysis, no units
<lookup tables>(lookup_table_template_name)	Specifies the timing models, power to use for the particular path in the circuit One, two or three dimensional models are used depending on the lookup-table being created

```
cell_fall(fall_template_name n x m)
    index_1 (value1, value2, ..., value n)
    index_2 (value1, value2, ..., value m)
    values (
        data_max11:data_typ11:data_min11, ..., data_max1m:data_typ1m:data_min1m \
        .... \
        data_maxn1:data_typn1:data_minn1, ..., data_maxnm:data_typerpnm:data_minnm);

pin(pin_name)
    direction : input, output, inout, internal
    clock_pin
    function(expression)
        Used for output or bidirectional pins. The expression defines the value of the output pin as a function of input pins
    max_capacitance
        The maximum output capacitive load that an output pin can drive
    capacitance
        The capacitive load of an input, inout, output or internal pin. Usually defined as 0 for output pins
    internal_power()
        Output pins in combinational cells, define the rise_power and fall_power to a related input pin. Input and clock pins also define this in sequential cells
    timing()
        Output pins in combinational cells, define the rise_delay, fall_delay, rise_transition and fall_transition to a related input pin
```

پیوست ۳

جدول ب-۲ قطعه کد globals.py

```
class Variables:
    varlist = []
    strlist = []
    def __init__(self):
        pass

class inputVars:
    ptrlist = []
    strlist = []
    def __init__(self):
        pass

def init():
    global varlist, inputs, VLSIlist, table
    varlist = Variables()
    inputs = inputVars()
    VLSIlist = []
    table = []
```

جدول ب-۳ تابع make_script_files در functions.py

```
def make_script_files():
    for t in globals.VLSIlist:
        file_name = 'S' + t.packname.replace("#", '')
        verilog_name = 'F' + t.packname.replace("#", '')
        script_file=open('scripts/' + file_name + '.scr','w')
        script_file.write('set power_preserve_rtl_hier_names "true"\n')
        script_file.write('analyze -format verilog { ./source/Verilogs/' + veri-
log_name + '.v' + '}\n')
        script_file.write('elaborate ' + verilog_name+'\n')
        script_file.write('link\n')
        script_file.write('uniquify -force\n')
        script_file.write('compile -map_effort medium\n')
        script_file.write('change_names -rules verilog -hierarchy\n')
        script_file.write('write -format verilog -hierarchy -output '+
'./netlist/' + verilog_name + '.v\n')
        script_file.write('uplevel #0 { report_power -analysis_effort low } >
./Power/' + verilog_name + '.txt\n')
        script_file.write('uplevel #0 { report_area -nosplit } > ./Area/' + veri-
log_name + '.txt\n')
        script_file.close()
```

جدول ٦-٤ سایر توابع مازول functions.py

```
import string
import globals
valid_input=[1,0,True,False] # Valid Input For Boolean Functions
def input_num(input_str):
    """
    (VLSI_Object)-> (Input_list, Numer Of Inputs)
    """
    try:
        input_list=[]
        for i in input_str:
            if (i in string.ascii_letters) and (i not in input_list):
                input_list.append(i)
        return len(input_list)
    except:
        print("Please Pass VLSI Object To Function")
        return None
# This Function Count Boolean Operation In Input String Separately
def input_op(input_str):

    try:
        counter=[0,0,0,0]
        input_list=[]
        for i in input_str:
            if i==".":
                counter[0]=counter[0]+1
            elif i=="+":
                counter[1]=counter[1]+1
            elif i=="^":
                counter[2]=counter[2]+1
            elif i=="!":
                counter[3]=counter[3]+1
        return counter # Return List
    except:
        print("Error In Input")
        return None

def check_valid(input_string):
    """
    (Str)->Boolean
    """
    try:
        for i in input_string:
            if i not in string.ascii_letters+ string.digits +"+^.() #!":
                return False
        return True
    except:
        print("Problem In Input")
        return None
def table_maker(obj):
    pass
def func_creator(obj):
    pass
```

جدول ٦-٥ توابع مازول generator.py

```
import string
from random import * # This Random Moudle Added For Rand Int And Other
import os # This Module Added For Get List Directory
variable=list(string.ascii_letters) # Convert Ascii Letter To List
operation=["+","-","*","^"] # Valid Operation

# This Function Create Gate Generation
def Gate_Gen(gate_number,var_num,not_off=True):
    gate_index=0 # Gate Index That Increase In Loop
    gate_list=[] # Empty List Of Gate
    while(gate_index<gate_number):
        coin_1=randrange(0,2) # Coin_1 For Var_1 Not Gate
        coin_2=randrange(0,2) # Coin_2 For Var_2 Not Gate
        var_1="" # Empty Var_1
        var_2="" # Empty Var_2
        while(var_1==var_2): # Generate Random Until Two Variable Be Different
            var_1=variable[randrange(0,var_num)]
            var_2=variable[randrange(0,var_num)]
        if not_off==False: # Check Not_False Argument
            if coin_1==1:
                var_1=var_1+"!"
            if coin_2==1:
                var_2=var_2+"!"
        op_index_not=randrange(1,4) # Operation Factor
        #coin_3=randrange(0,2)
        gate_list.append("(" + var_1 + operation[op_index_not] + var_2 + ")") # Create And Append One Gate At The End OF List
        gate_index=gate_index+1 # increase Gate index
    return gate_list

# Generator Final String
def generator(gate_list,not_off=True):
    temp=gate_list # Copy OF The Input Gate List
    temp_2=[] # Empty List
    while(len(temp)>2): # Do This While Until We Have Two Object Process
        i=0 # Counter
        coin_not=randrange(0,2) # Random Number For Add Not At The End Of Each Parantece
        if coin_not==1: # Condition Of Not Adding
            temp[i]=temp[i]+"!" # Add Not To The First
        if len(temp)%2==0: # Even Length
            while((i+2)<len(temp)):
                op_index=operation[randrange(1,4)] # Generate Random Operation (-Not In This Version)
                temp_2.append("(" + temp[i] + op_index + temp[i+1] + ")")
                i=i+2
        else: # Odd Length
            while((i+2)<len(temp)):
                op_index=operation[randrange(1,4)] # Generate Random Operation (- Not In This Version)
                temp_2.append("(" + temp[i] + op_index + temp[i+1] + ")") # merge tuple Of Two From Original List
                i=i+2 # Jump To Next Two
        temp_2.append(temp[len(temp)-1]) # Create New Gate List
        temp=temp_2 # Copy To First Temp
        temp_2=[] # Clear List
    return temp[1]+operation[randrange(1,4)]+temp[0] # merge Last Two Object And Return Last Object
```

```
if __name__=="__main__":
    file_list=os.listdir()
    for i in range(len(file_list)):
        print(str(i+1)+" - "+file_list[i]+"\n")
    input_str=input("Please Enter One Of This File For Generate : \n")
    if int(input_str)<=len(file_list):
        file=open(file_list[int(input_str)-1], 'w')
        gate_number=input("Please Enter Number Of 2-Input Gates in File : ")
    var_number=input("Please Enter Total Number Of Var in File")
    try:
        gate_form=Gate_Gen(int(gate_number),int(var_number))
        print(gate_form)
        gen_function=generator(gate_form)
        file.write(gen_function)
        file.close()
        print("Function : "+gen_function)
        print("Done!!!")
    except:
        print("Wrong Number Inserted")
        file.close()

else:
    "Wrong Files Number"
```

جدول ٦-٦ توابع مازول logic.py

```
def And(a,b):
    if (a in [0,1] and b in [0,1]):
        return a and b
    else:
        print("Input Is Not Boolean")
        return None

def Or(a,b):
    if (a in [0,1] and b in [0,1]):
        return a or b
    else:
        print("Input Is Not Boolean")
        return None

def Xor(a,b):
    if (a in [0,1] and b in [0,1]):
        if (a == b):
            return 0
        else:
            return 1
    else:
        print("Input Is Not Boolean")
        return None

def Not(a):
    if a in [0,1]:
        if (a == 0):
            return 1
        else:
            return 0
    else:
        print("Input Is Not Boolean")
        return None

def Nand(a,b):
    return Not(And(a,b))

def Nor(a,b):
    return Not(Or(a,b))

def Xnor(a,b):
    return not(Xor(a,b))
```

جدول ٦-٧ توابع مازول parser.py

```
from VLSI import *
import globals

def make_table():
    for counter in range(0,2**len(globals.inputs.strlist)):
        temp = counter
        for i in range(0,len(globals.inputs.strlist)):
            globals.varlist.varlist[globals.inputs.ptrlist[i]] = int(temp%2)
            temp =int(temp/ 2)

        globals.table.append(calculate_result())
        counter+=1

def print_result():
    outstr = ''
    for i in globals.inputs.strlist:
        outstr += i
    print (outstr+ '    Out')

    for counter in range(0,2**len(globals.inputs.strlist)):
        #print('number of inputs: ' + str(len(globals.inputs.strlist)))
        temp = counter
        outstr = ''
        for i in range(0,len(globals.inputs.strlist)):
            outstr += str(temp%2)
            temp = int(temp/2)

        outstr += '    ' + globals.table[counter]
    print(outstr)

def calculate_result():
    for temp in globals.VLSIlist:
        while (not temp.function()):
            #
            output = temp.func[len(temp.func)-1].output
            pass
    temp = globals.VLSIlist[len(globals.VLSIlist)-1]
    return str(globals.varlist.varlist[temp.variables[temp.func[len(temp.func)-1].output]])
```

```

def parse_string(inputstring):

    start = -2
    end = -1
    indexcounter = 0
    i = 0
    j = 0

    temp = inputstring
    temp.replace(" ", '')

    while (i < len(temp)):

        if (temp[i] == ')'):
            end = i
            j = i
            while (j >= 0):
                if (temp[j] == '('):
                    start = j
                    break
                else:
                    j-=1

            if (start < 0):
                break
            elif (i == len(temp) -1):
                start = 0
                end = len(temp) - 1
                indexcounter += 1
                packname = ('#%d#' % indexcounter)
                tempobject = VLSI(temp,packname,len(globals.VLSIlist))
                globals.VLSIlist.append(tempobject)
                break

        if (start >= 0):
            indexcounter+=1
            temp1 = temp[start+1:end]
            packname = ('#%d#' % indexcounter) #      (a+b) -> #indexcounter# and
is saved as a var
            tempobject = VLSI(temp1,packname,len(globals.VLSIlist))
            globals.VLSIlist.append(tempobject)
            temp = temp[:start]+('#%d#' % (indexcounter)) + temp[end+1:]
            start = -2
            end = -1
            i = 0

    i+=1

```

جدول ٨-٦ توابع مازول test.py

```
from VLSI import *
from parser2 import *
import platform
import globals

#input the string to pass to VLSI object list
input_command=input("Please Enter One Of This : \n 1.Test_Case1 \n 2.Test_Case2 \n 3.Test_Case3 \n Any Other : Manual Input \n")
if int(input_command)==1:
    file=open("Test_Case1.txt","r")
    file_name=file.name
    inputstring=file.read()
    file.close()
elif int(input_command)==2:
    file=open("Test_Case2.txt","r")
    file_name=file.name
    inputstring=file.read()
    file.close()
elif int(input_command)==3:
    file=open("Test_Case3.txt","r")
    file_name=file.name
    inputstring=file.read()
    file.close()
else:
    inputstring=input("Please Enter Function : ")
    file_name="Manual Function"
input_var_num=input_num(inputstring)
input_op_num=input_op(inputstring)

globals.init()

timer_1=time.perf_counter()      # Start Time Of Analysis

parse_string(inputstring)
timer_2=time.perf_counter()
make_table()
timer_3=time.perf_counter()
# End Of Time Performance Counter
today_date=datetime.datetime.today() # Today Date And Local Time For Saving
In Performance Text File
parser_perf_time=str(timer_2-timer_1)+" Sec"
table_perf_time=str(timer_3-timer_2)+" Sec"
print("Parser Performance Time: "+parser_perf_time) # Print Time Performance
print("Table Maker Performance Time: "+table_perf_time) # Print Time Performance

make_script_files()

print_result()
```

```

#print('Variables used:')
#for k in range(0,len(globals.varlist.varlist)):
#    print(str(k))
#    print('value:' + str(globals.varlist.varlist[k]))
#    print('string token:' + globals.varlist.strlist[k])
#    print()

#print('VLSI objects:')
#for temp in globals.VLSIlist:
#    for funcs in temp.func:
#        print('first: ' + str(temp.variables[funcs.firstvarptr]))
#        print('second: ' + str(temp.variables[funcs.secondvarptr]))
#        print('out: ' + str(temp.variables[funcs.output]))
#        funcs.printobj()
#        print('\n')
#    print('next object\n')

#for k in range(0,len(globals.inputs.strlist)):
#    print(str(k))
#    print('value:' + str(globals.inputs.ptrlist[k]))
#    print('string token:' + globals.inputs.strlist[k])
#    print()

result_file=open("Parser_Perf_Result.txt","a") # Open Result File
result_file_2=open("Table_Perf_Result.txt","a")
result_file.write(file_name+ " : ,"+ "Input Var Number:
"+str(input_var_num)+" Input Operation Number: "+str(sum(input_op_num))+"
Elapsed Time: "+parser_perf_time+" "+str(today_date)+" CPU:
"+platform.processor()+"\n") # Write Result In File
result_file_2.write(file_name+ " : ,"+ "Input Var Number:
"+str(input_var_num)+" Input Operation Number: "+str(sum(input_op_num))+"
Elapsed Time: "+table_perf_time+" "+str(today_date)+" CPU:
"+platform.processor()+"\n") # Write Result In File
result_file.close() # Close Parser Text File
result_file_2.close() # Close Table Maker File

#print('Variables used:')
#for k in range(0,len(varlist.varlist)):
#    print(str(k))
#    print('value:' + str(varlist.varlist[k]))
#    print('string token:' + varlist.strlist[k])
#    print()

#for k in range(0,len(inputs.strlist)):
#    print(str(k))
#    print('value:' + str(inputs.ptrlist[k]))
#    print('string token:' + inputs.strlist[k])
#    print()

```

جدول ۶-۹ متدهای عمومی کلاس VLSI

```
from functions import * # Outside Of The Class Functions
from logic import *
import globals
import time # Import Time Module For Counter Performance
import datetime # Import Date And Time For Saving Result
class funcObject:
    def __init__(self,firstvar,outputvar,secondvar=-1,funct=Not):
        self.func = funct
        self.firstvarptr = firstvar
        self.secondvarptr = secondvar
        self.output = outputvar

    def function(self,c,d):
        if (self.func != Not):
            return self.func(c,d)
    def Notfunction(self,ar):
        if (self.func == Not):
            return self.func(ar)
    def printobj(self):
        toprint = ''
        if (self.func == And):
            toprint = "And"
        elif (self.func == Or):
            toprint = "Or"
        elif (self.func == Xor):
            toprint = "Xor"

        if (self.func == Not):
            toprint = "Not"
        print('gate: ' + toprint)

class VLSI:
#    variables = []
#    func = []
#    inputs = []
#    wires = []
    packname = ''
    firstvar = -1
    secondvar = -1
    outputvar = -1
```

```

def __init__(self, string, output, index):
    if check_valid(string):
        self.inputs = []
        self.wires = []
        self.func = []
        self.variables = []

        self.input=string # Input String
        self.inputnum=input num(string) # Add Input Number As Attribute
        #print(str(self.outputvar))
        #checking for variables and functions in the passed string

    self.varcounter = 0

    start = 0
    end = -1
    #start and end of each section

    cursor = 0
    clone = string
    clone.replace(" ", "")
    #cloning input string

    ## !a + !(c+a.d)

    ## detecting and replacing NOTs with variables
    while (1):
        symbolcounter = 0
        for i in range(0, len(clone)):
            if clone[i] in "+.^!" :
                symbolcounter +=1
            if cursor == len(clone)-1:
                break
            elif (clone[cursor] == "!" ):
                start = cursor+1
                tempcursor = cursor +1
                while (1):
                    if (tempcursor < len(clone)):
                        if clone[tempcursor] in "+.^":
                            break
                        tempcursor += 1
                    if (tempcursor == len(clone)):
                        break

                end = tempcursor

```

```

tonot = self.handle_variable(clone[start:end])
self.inputs.append(clone[start:end])
self.packname = ('%d' % index) + ('%d#' % self.varcounter)
self.varcounter += 1
if (symbolcounter == 1):
    self.outputvar = self.handle_variable(output)
    out = self.outputvar
    self.packname = output
else:
    out = self.handle_variable(self.packname)
self.wires.append(out)
tempfuncobject = funcObject(firstvar = tonot, outputvar = out)
self.func.append(tempfuncobject)
clone = clone[0:start-1] + self.packname + clone[end:]
cursor += 1

cursor = 0
start = 0
while (1):
    symbolcounter = 0
    for i in range(0,len(clone)):
        if clone[i] in "+.^":
            symbolcounter +=1
        if cursor == len(clone) - 1:
            break
    elif clone[cursor] in "+.^":
        if (clone[cursor] == '+'):
            tempfunc = Or
        elif (clone[cursor] == '.'):
            tempfunc = And
        elif (clone[cursor] == '^'):
            tempfunc = Xor
        else:
            tempfunc = And

    end = cursor
    #detect variable
    temp1 = clone[start:end]
    temp = clone[start:end]
    if (self.count_sharps(temp) <= 2):
        self.inputs.append(temp)
    tempcursor = cursor + 1
    while (clone[tempcursor] not in "+.^" and tempcursor != len(clone)-1):
        tempcursor += 1
    if (tempcursor == len(clone) -1):
        tempcursor += 1

```

```

temp2 = clone[end+1:tempcursor]
        temp = temp2
        if (self.count_sharps(temp) <= 2):
            self.inputs.append(temp)
            self.packname = ('#%d' % index)+('#%d#' % 
self.varcounter)
            self.varcounter+=1
            if symbolcounter == 1:
                self.outputvar = self.handle_variable(output)
                out = self.outputvar
                self.packname = output
            else:
                out = self.handle_variable(self.packname)
                self.wires.append(out)

            first = self.handle_variable(temp1)
            second = self.handle_variable(temp2)

            tempfuncobject = funcObject(first,out,second,tempfunc)
            self.func.append(tempfuncobject)
            clone = clone[0:start] + self.packname +
clone[tempcursor:]

cursor = 0

cursor += 1
#if (output[0] == '#'):
#    output = 'V'+output
self.inputs.append(output)

##else:
##    print("Please Enter Valid String")
def handle_variable(self,string):
    for i in range(0,len(globals.varlist.strlist)):
        if (string == globals.varlist.strlist[i]):
            self.variables.append(i)
            return len(self.variables)-1
    if (not('#' in string)):
        globals.inputs.ptrlist.append(len(globals.varlist.strlist))
        globals.inputs.strlist.append(string)
        globals.varlist.strlist.append(string)
        globals.varlist.varlist.append(0)
        self.variables.append(len(globals.varlist.varlist) - 1)
    return len(self.variables) - 1

```

```
def count_sharps(self,string):
    counter = 0
    for i in range(0,len(string)):
        if (string[i] == '#'):
            counter += 1
    return counter

def __str__(self):
    #This Function Is For Show Object In Print Format
    return "VLSI ("+self.input+")"
def __repr__(self):
    # Representing VLSI Object
    return "VLSI_Object(Input_String="+self.input+ ")"
```

جدول ٦٠-١ متد ساخت فایل Verilog کلاس VLSI

```
def make_verilog(self):
    file_name = 'F' + self.packname.replace("#", '')
    verilog_file=open('../source/Verilogs/' + file_name + '.v', "w")
    verilog_file.write('module ' + file_name + ' (')

    for v in range(0,len(self.inputs)):
        if (v != 0):
            verilog_file.write(', ')
        varname = globals.varlist.strlist[self.variables[v]]
        varname = self.inputs[v]
        if ('#' in varname):
            varname = 'V' + varname.replace('#', 'V')
        # if (varname[0] == '#'):
        #     varname = 'V' + varname.replace('#', '')
        verilog_file.write(varname)
    verilog_file.write('); \n')

    for v in range(0,len(self.inputs)-1):
        if (v == 0):
            verilog_file.write('input ')
        else:
            verilog_file.write(' , ')
        varname = self.inputs[v]
        if ('#' in varname):
            varname = 'V' + varname.replace('#', 'V')
        verilog_file.write(varname)

    verilog_file.write('; \noutput ' + 'V' + self.inputs[len(self.inputs)-1].replace('#', 'V') + '; \n')
    if (len(self.wires) != 0):
        verilog_file.write('wire ')
        for w in range(0,len(self.wires)):
            if (w != 0):
                verilog_file.write(', ')
            wirename = globals.varlist.strlist(self.wires[w])
            verilog_file.write(wirename)
        log_file.write('W'+globals.varlist.strlist[self.variables[self.wires[w]]].replace('#', 'W'))

    verilog_file.write('; \n\n')

    for i in range(0,len(self.func)):

        firstflag = -1
        secondflag = -1
        outflag = -1

        f = self.func[i]
        outstr = globals.varlist.strlist[self.variables[f.output]]
        firststr = globals.varlist.strlist[self.variables[f.firstvarptr]]
```

```

secondstr = f.secondvarptr
    if (secondstr != -1):
        secondstr = globals.varlist.strlist[self.variables[secondstr]]
    else:
        secondstr = "~"
    for j in self.inputs:
        if (outstr == j):
            outstr = 'V'+outstr.replace('#', 'V')
            outflag = 1
        if (firststr == j):
            if ('#' in firststr):
                firststr = 'V'+firststr.replace('#', 'V')
            firstflag = 1
        if (secondstr == j):
            if ('#' in secondstr):
                secondstr = 'V'+secondstr.replace('#', 'V')
            secondflag = 1

    if (firstflag != 1):
        firststr = 'W' + firststr.replace('#', 'W')
    if (secondflag != 1):
        secondstr = 'W' + secondstr.replace('#', 'W')
    if (outflag != 1):
        if (outstr == self.inputs[len(self.inputs)-1]):
            outstr = 'V' + outstr.replace('#', 'W')
        else:
            outstr = 'W' + outstr.replace('#', 'W')

    if (f.func == And):
        verilog_file.write((and f%d ('% i) + outstr + ' , ' + firststr + ' , ' + secondstr +
') ;\n')
    elif (f.func == Or):
        verilog_file.write((or f%d ('% i) + outstr + ' , ' + firststr + ' , ' + secondstr +
') ;\n')
    elif (f.func == Xor):
        verilog_file.write((xor f%d ('% i) + outstr + ' , ' + firststr + ' , ' + secondstr +
') ;\n')
    elif (f.func == Not):
        verilog_file.write((not f%d ('% i) + outstr + ' , ' + firststr + ') ;\n')

verilog_file.write('endmodule')
verilog_file.close()

```



**Amirkabir University of Technology
(Tehran Polytechnic)**

Electrical Engineering Department

BSc Thesis

Design a logic recommender system for optimization of digital CMOS integrated circuits by means of artificial intelligence algorithm

**By
Sepand Haghghi**

**Supervisor
Dr. M. Shalchian**

October 2016