



دانشگاه آزاد اسلامی

واحد تهران مرکزی

دانشکده فنی و مهندسی، گروه برق

پایان نامه برای دریافت درجه کارشناسی ارشد (M.Sc)

گرایش: الکترونیک

عنوان:

پیاده سازی الگوریتم سخت افزاری کلونی زنبورها بر روی FPGA

برای مسئله فروشنده دوره گرد

استاد راهنمای:

دکتر فرداد فرخی

استاد مشاور:

دکتر رضا صباغی ندوشن

پژوهشگر:

پوریا خداقلی پور

تابستان ۱۳۹۵

بِهِ نَامِ اِبْرَانِ

تقدیم به پدر و مادر عزیزم

به نام پدر و مادر که بی‌شک هرآنچه که هستم از سایه وجودشان است. پدرم، دستهایت را هر لحظه بر شانه‌هایم حس کردم که در این وادی پر از فراز و نشیب ترسی به دل راه ندادم و همواره دیوار استوار حضورت مطمئن‌ترین تکیه‌گاه من بوده است. مادرم، داشتن تو موهبتی است آسمانی. چه بگوییم که در وصف فدایکاری و مهربانی تو زبان قاصر است. عزیزانم، بزرگ‌ترین افتخارم زندگی در سایه آرامش بخش شماست.

تشکر و قدردانی

با تشکر از دانشگاه آزاد اسلامی، خالصانه‌ترین مراتب قدردانی خود را به استاد عزیز و گرانقدرم جناب آقای دکتر فرداد فرخی که با ممتاز و مهربانی فراوان راهنمایی‌های این تحقیق را بر عهده داشته تقدیم می‌دارم. همچنین از استاد مشاور جناب آقای دکتر رضا صباغی ندوشن کمال تشکر را داشته و از خداوند بزرگ برای این بزرگواران آرزوی بهروزی و سعادت را مسئلت دارم. همچنین از اساتید گرامی گروه برق الکترونیک واحد تهران مرکزی، جناب آقایان دکتر کنگرلو، دکتر کاشانی‌نیا و دکتر جوادی که در طول دوران تحصیل افتخار بهره‌مندی از محضر ایشان را داشته‌ام سپاسگزاری می‌نمایم.

فهرست مطالب

۱	چکیده
۲	۱. بیان مسئله
۲	۱-۱ فرضیه تحقیق
۲	۱-۱-۱ الگوریتم‌های بهینه‌سازی هوش جمعی
۵	۱-۱-۲ پیاده‌سازی الگوریتم‌های هوش جمعی بر روی FPGA
۵	۱-۲-۱ بهینه‌سازی کلونی مورچه‌ها بر روی FPGA
۵	۱-۲-۱-۱ بهینه‌سازی اجتماع ذرات
۷	۲-۱ اهمیت و ضرورت انجام تحقیق
۹	۲-۳-۱ اهداف تحقیق
۱۰	۲. مقدمه
۱۰	۱-۲ مسئله فروشنده دوره‌گرد
۱۳	۱-۲-۲ FPGA
۱۳	۱-۲-۲-۱ FPGA چیست؟
۱۴	۱-۲-۲-۲ جذابیت FPGA در چیست؟
۱۵	۱-۲-۲-۳ برای چه کاری می‌توان از FPGA استفاده کرد؟
۱۷	۱-۲-۲-۴ خواستگاه FPGA
۱۷	۱-۴-۲-۲ CPLD و SPLD
۱۸	۲-۴-۲-۲ PROM
۱۸	۳-۴-۲-۲ PLA
۱۹	۴-۴-۲-۲ GAL و PAL
۲۰	۵-۴-۲-۲ CPLD
۲۱	۶-۴-۲-۲ FPGA

۲۴	۵-۲-۲ معماری FPGA
۲۴	۱-۵-۲-۲ معماری درشت، متوسط و ریزدانه
۲۵	۲-۵-۲-۲ سلول منطقی XILINX
۲۶	۳-۵-۲-۲ المان منطقی Altera
۲۷	۴-۵-۲-۲ برش (Slice)
۲۷	۵-۵-۲-۲ LAB و CLB
۲۸	۳-۲ کلونی زنبورها
۲۹	۱-۳-۲ رفتارهای اجتماعی زنبورها
۲۹	۱-۱-۳-۲ رفتار جستجوی مکان کندو
۳۰	۲-۱-۳-۲ رفتار ازدواج
۳۰	۳-۱-۳-۲ رفتار جستجوی منابع غذایی
۳۶	۲-۳-۲ الگوریتم‌های کلونی زنبور
۳۶	۱-۲-۳-۲ الگوریتم زنبور
۳۷	۱-۱-۲-۳-۲ طرح ارائه شده
۳۷	۲-۱-۲-۳-۲ مقداردهی اولیه
۳۸	۳-۱-۲-۳-۲ رقص جنبشی
۳۸	۴-۱-۲-۳-۲ جستجوی محلی
۳۹	۵-۱-۲-۳-۲ جستجوی سراسری
۳۹	۶-۱-۲-۳-۲ بروزرسانی جمعیت
۴۰	۷-۱-۲-۳-۲ معیار توقف
۴۰	۸-۱-۲-۳-۲ الگوریتم زنبور توسعه یافته
۴۱	۲-۲-۳-۲ الگوریتم کلونی زنبور مصنوعی (ABC)
۴۶	۳-۲-۳-۲ الگوریتم بهینه‌سازی کلونی زنبور (BCO)

۵۲	۳. پیاده‌سازی نرم‌افزاری و ساخت‌افزاری الگوریتم زنبور
۵۲	۱-۳ پیاده‌سازی نرم‌افزاری
۵۲	۱-۱-۳ کدگزاری برای مسئله فروشنده دوره‌گرد
۵۳	۲-۱-۳ مسیر نزدیکترین همسایگی
۵۳	۳-۱-۳ روش‌های بهینه‌سازی محلی برای مسئله فروشنده دوره‌گرد
۵۴	۱-۳-۱-۳ ۲-opt روش
۵۴	۲-۳-۱-۳ روش جهش حریص تکه مسیر (GSTM)
۶۰	۴-۱-۳ الگوریتم زنبور برای حل مسائل ترکیبی
۶۲	۵-۱-۳ الگوریتم کلونی زنبور مصنوعی برای حل مسائل ترکیبی
۶۳	۶-۱-۳ الگوریتم بهینه‌سازی کلونی زنبور برای حل مسائل ترکیبی
۶۵	۷-۱-۳ نقاط ضعف و قوت سه الگوریتم کلونی زنبورها در حل مسائل ترکیبی
۶۷	۲-۳ الگوریتم کلونی زنبور پیشنهادی برای مسائل ترکیبی
۷۰	۳-۳ پیاده‌سازی ساخت‌افزاری
۷۰	۱-۳-۳ ساختار کلی پیاده‌سازی (واحد اصلی)
۷۳	۱-۱-۳-۳ نمودار ماشین حالت کراندار
۷۴	۲-۱-۳-۳ نمودار ماشین حالت واحد اصلی
۸۲	۲-۳-۳ واحد تولیدکننده مسیر نزدیکترین همسایگی
۸۷	۱-۲-۳-۳ واحد تشخیص نزدیکترین همسایه
۹۶	۲-۲-۳-۳ واحد تولیدکننده آدرس ترتیبی دو شهر
۱۰۳	۳-۲-۳-۳ واحد مسیر حرکت
۱۰۳	۱-۳-۲-۳-۳ رم توزیع شده
۱۰۴	۲-۳-۲-۳-۳ واحد رم توزیع شده
۱۰۵	۳-۳-۲-۳-۳ حالت‌های واحد مسیر حرکت

- ۱۰۸ ۴-۲-۳-۳ واحد رام بلوکی با یک درگاه
- ۱۰۸ ۱-۴-۲-۳-۳ RAM های بلوکی یا
- ۱۰۹ ۲-۴-۲-۳-۳ واحد رام بلوکی استفاده شده
- ۱۰۹ ۳-۴-۲-۳-۳ اطلاعات مسائل
- ۱۱۱ ۴-۴-۲-۳-۳ مقایسه دو نوع اطلاعات مسئله به منظور پیاده‌سازی سخت‌افزاری
- ۱۱۲ ۵-۴-۲-۳-۳ بررسی دقیق در پیاده‌سازی
- ۱۱۴ ۶-۴-۲-۳-۳ پیاده‌سازی اطلاعات مسئله بر روی FPGA
- ۱۲۰ ۵-۲-۳-۳ واحد محاسبه‌گر آدرس رام یا رم
- ۱۲۳ ۶-۲-۳-۳ واحد مقایسه و ذخیره (آدرس و مسافت تا نزدیکترین همسایه)
- ۱۲۶ ۳-۳-۳ واحد زنبور عسل
- ۱۳۰ ۴-۳-۳ واحد طبقه‌بندی زنبورها
- ۱۳۷ ۵-۳-۳ واحد محاسبات بهینه‌سازی محلی
- ۱۳۹ ۱-۵-۳-۳ تفسیر روش 2-opt برای پیاده‌سازی بر روی سخت‌افزار
- ۱۴۰ ۲-۵-۳-۳ نمودار ماشین حالت واحد محاسبات بهینه‌سازی محلی
- ۱۴۴ ۳-۵-۳-۳ واحد تولید تصادفی آدرس دو شهر
- ۱۴۶ ۱-۳-۵-۳-۳ شیفت رجیستر بازخورد خطی (LFSR)
- ۱۴۸ ۲-۳-۵-۳-۳ واحد تولید تصادفی آدرس در محدوده تعداد شهرها
- ۱۵۰ ۳-۳-۵-۳-۳ نمودار ماشین حالت واحد تولید تصادفی آدرس دو شهر
- ۱۵۳ ۶-۳-۳ واحد محاسبات بروزرسانی زنبور
- ۱۵۸ ۷-۳-۳ کاهش حجم بلوک‌های رم استفاده شده
- ۱۵۹ ۱-۷-۳-۳ واحد محاسبه‌گر جذر
- ۱۶۱ ۲-۷-۳-۳ واحد محاسبه‌گر فاصله اقلیدسی
- ۱۶۴ ۳-۷-۳-۳ واحد تولیدکننده آدرس و محتوا ماتریس فاصله‌ها

۱۶۹	۱-۳-۷-۳-۳ واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه
۱۷۱	۴-۷-۳-۳ واحد تولیدکننده ماتریس فاصله‌ها روی حافظه
۱۷۵	۴. نتایج
۱۷۵	۱-۴ پایگاه داده‌ها
۱۷۸	۲-۴ نحوه اعتبارسنجی
۱۸۱	۳-۴ نتایج پیاده‌سازی نرم افزاری
۱۸۳	۱-۳-۴ الگوریتم BA
۱۸۷	۲-۳-۴ الگوریتم ABC
۱۹۱	۳-۳-۴ الگوریتم BCO
۱۹۵	۴-۳-۴ الگوریتم پیشنهادی (HBA)
۱۹۹	۴-۴ نتایج پیاده‌سازی سخت افزاری
۱۹۹	۱-۴-۴ هسته‌های فرعی الگوریتم HBA
۲۰۱	۲-۴-۴ هسته اصلی الگوریتم HBA
۲۰۳	۵. نتیجه گیری
۲۰۳	۱-۵ بررسی نتایج پیاده‌سازی نرم افزاری
۲۰۶	۲-۵ بررسی نتایج پیاده‌سازی سخت افزاری
۲۰۸	۳-۵ جمع‌بندی
۲۰۹	۴-۵ پیشنهادات برای کارهای آتی
۲۱۱	۶. مراجع
۲۱۱	۱-۶ مراجع فارسی
۲۱۱	۲-۶ مراجع انگلیسی

فهرست شکل‌ها

- ۱۱ شکل ۱-۲ تور (c_1, c_2, \dots, c_n) در یک مسئله TSP
- ۱۸ شکل ۲-۲ تقسیم‌بندی قطعات منطقی برنامه‌پذیر
- ۲۱ شکل ۳-۲ ساختار عام یک CPLD
- ۲۲ شکل ۴-۲ المان‌های کلیدی تشکیل دهنده یک بلوک منطقی برنامه‌پذیر ساده
- ۲۳ شکل ۵-۲ نمای بالا به پایین معماری عام FPGA
- ۲۶ شکل ۶-۲ نمای ساده شده یک XILINX LC
- ۲۷ شکل ۷-۲ برش XILINX
- ۲۸ شکل ۸-۲ یک CLB حاوی چهار برش (تعداد برشها به خانواده FPGA وابسته است)
- ۳۳ شکل ۹-۲ رقص پیچشی زنبورهای عسل
- ۳۴ شکل ۱۰-۲ ارتباط بین زاویه مسیر حرکت زیگزاگ و موقعیت غذایی
- ۳۷ شکل ۱۱-۲ پارامترهای الگوریتم زنبور
- ۴۰ شکل ۱۲-۲ الگوریتم BA
- ۴۸ شکل ۱۳-۲ حل مسئله TSP با استفاده از الگوریتم BCO
- ۵۳ شکل ۱-۳ نحوه کدگزاری مسائل برای حل مسئله فروشنده دوره‌گرد
- ۵۴ شکل ۲-۳ روش 2-OPT
- ۵۷ شکل ۳-۳ اتصال حریص
- ۵۷ شکل ۴-۳ نزدیکترین همسایه شهرهای R_1 و R_2 که به صورت تصادفی از تکه مسیر انتخاب شدند
- ۵۸ شکل ۵-۳ چرخش شهر انتخاب شده تصادفی NL_{R_1} از فهرست همسایگی شهر R_1
- ۵۸ شکل ۶-۳ چرخش شهر انتخاب شده تصادفی NL_{R_2} از فهرست همسایگی شهر R_2
- ۷۱ شکل ۷-۳ طرح کلی الگوریتم ارائه شده بر روی سخت‌افزار
- ۷۴ شکل ۸-۳ واکنش بین کنترل و مسیر داده
- ۷۵ شکل ۹-۳ چارت FSM واحد اصلی

۸۳	شکل ۱۰-۳ واحد تولیدکننده مسیر نزدیکترین همسایگی
۸۳	شکل ۱۱-۳ نمودار ماشین حالت واحد تولید مسیر نزدیکترین همسایگی
۸۸	شکل ۱۲-۳ واحد تشخیص نزدیکترین همسایه
۸۹	شکل ۱۳-۳ نمای کلی ساختار واحد تشخیص نزدیکترین همسایه
۹۰	شکل ۱۴-۳ قسمت اول از ساختار تشخیص نزدیکترین همسایه
۹۲	شکل ۱۵-۳ قسمت دوم از ساختار تشخیص نزدیکترین همسایه
۹۳	شکل ۱۶-۳ قسمت سوم از ساختار تشخیص نزدیکترین همسایه
۹۴	شکل ۱۷-۳ قسمت چهارم از ساختار تشخیص نزدیکترین همسایه
۹۷	شکل ۱۸-۳ نحوه شمارش آدرس‌های رم برای تولید مسیر نزدیکترین همسایگی در سطح رفتار
۹۸	شکل ۱۹-۳ نحوه شمارش آدرس‌های رم برای تولید مسیر نزدیکترین همسایگی در سطح ساختار
۹۸	شکل ۲۰-۳ واحد تولیدکننده آدرس ترتیبی دو شهر
۱۰۰	شکل ۲۱-۳ نمودار FSM واحد تولیدکننده آدرس ترتیبی دو شهر
۱۰۲	شکل ۲۲-۳ پیاده‌سازی واحد تولید آدرس ترتیبی دو شهر در سطح ساختار
۱۰۲	شکل ۲۳-۳ واحد آشکارساز لبه
۱۰۴	شکل ۲۴-۳ یک CLB حاوی چهار برش
۱۰۴	شکل ۲۵-۳ واحد رم توزیع شده با دو درگاه
۱۰۵	شکل ۲۶-۳ واحد مسیر حرکت
۱۰۷	شکل ۲۷-۳ نمودار ماشین حالت واحد مسیر حرکت
۱۰۸	شکل ۲۸-۳ نمای تراشه با چند ستون از بلوک‌های RAM تعییه شده
۱۰۹	شکل ۲۹-۳ واحد رام بلوکی با یک درگاه
۱۱۰	شکل ۳۰-۳ اطلاعات یک مسئله فرضی برای مسئله فروشنده دوره گرد
۱۱۰	شکل ۳۱-۳ ماتریس فاصله‌ها برای مسئله فرضی با N شهر
۱۱۷	شکل ۳۲-۳ نحوه قرارگیری ماتریس فاصله‌ها بر روی رام و آدرس متناظر آنها

- شکل ۳-۳۳ نحوه پیاده‌سازی در سطح ساختار فرمول محاسبه آدرس رام ۱۱۹
- شکل ۳-۳۴ واحد محاسبه‌گر آدرس رم یا رام ۱۲۰
- شکل ۳-۳۵ ساختار واحد محاسبه‌گر آدرس رم یا رام ۱۲۱
- شکل ۳-۳۶ واحد محاسبه‌گر فرمول آدرس ۱۲۱
- شکل ۳-۳۷ نمودار FSM واحد محاسبه‌گر فرمول آدرس ۱۲۲
- شکل ۳-۳۸ واحد مقایسه و ذخیره ۱۲۴
- شکل ۳-۳۹ ساختار واحد مقایسه و ذخیره ۱۲۵
- شکل ۳-۴۰ واحد زنبور عسل ۱۲۶
- شکل ۳-۴۱ نمودار FSM زنبور عسل ۱۲۷
- شکل ۳-۴۲ ساختار قسمت اجازه جابه‌جایی از واحد طبقه‌بندی ۱۳۲
- شکل ۳-۴۳ هسته اجازه جابه‌جایی ۱۳۳
- شکل ۳-۴۴ سیگنال اتمام کار واحد طبقه‌بندی ۱۳۴
- شکل ۳-۴۵ تعیین اتصالات ورودی واحد زنبورها در واحد طبقه‌بندی ۱۳۶
- شکل ۳-۴۶ واحد محاسبات بهینه‌سازی محلی ۱۳۷
- شکل ۳-۴۷ نحوه اتصالات دو واحد زنبور و محاسبات بهینه‌سازی محلی ۱۳۸
- شکل ۳-۴۸ روش بهینه‌سازی محلی 2-OPT ۱۳۹
- شکل ۳-۴۹ نمودار FSM واحد محاسبات بهینه‌سازی محلی ۱۴۰
- شکل ۳-۵۰ واحد تولید آدرس تصادفی دو شهر ۱۴۵
- شکل ۳-۵۱ یک نمونه LFSR ۳۲ بیتی ۱۴۷
- شکل ۳-۵۲ واحد تولید تصادفی آدرس در محدوده تعداد شهرها ۱۴۸
- شکل ۳-۵۳ ساختار داخلی واحد تولید آدرس تصادفی در محدوده نزدیک به تعداد شهرها ۱۴۹
- شکل ۳-۵۴ نمودار FSM واحد تولید آدرس تصادفی دو شهر ۱۵۰
- شکل ۳-۵۵ واحد محاسبات بروزرسانی زنبور ۱۵۳

- شکل ۳-۵۶ نحوه اتصال واحدهای زنبور، محاسبات بهینه‌سازی محلی و محاسبات بروزرسانی
۱۵۴
- شکل ۳-۵۷ نحوه دوران تکه مسیر
۱۵۵
- شکل ۳-۵۸ نمودار FSM واحد محاسبات بروزرسانی زنبور
۱۵۶
- شکل ۳-۵۹ مولد هسته محاسبه گر جذر شرکت XILINX
۱۵۹
- شکل ۳-۶۰ مولد هسته محاسبه گر جذر شرکت ALTERA
۱۶۰
- شکل ۳-۶۱ واحد محاسبه گر فاصله اقلیدسی
۱۶۱
- شکل ۳-۶۲ نمودار FSM واحد محاسبه گر فاصله اقلیدسی برای FPGA های شرکت XILINX
۱۶۲
- شکل ۳-۶۳ نمودار FSM واحد محاسبه گر فاصله اقلیدسی برای FPGA های شرکت ALTERA
۱۶۴
- شکل ۳-۶۴ واحد تولیدکننده آدرس و محتوای ماتریس فاصله‌ها
۱۶۵
- شکل ۳-۶۵ ساختار واحد تولیدکننده آدرس و محتوای ماتریس فاصله‌ها با بهره از رام دو درگاهه
۱۶۵
- شکل ۳-۶۶ ساختار واحد تولیدکننده آدرس و محتوای ماتریس فاصله‌ها با بهره از رام تک درگاهه
۱۶۶
- شکل ۳-۶۷ قسمت کنترل واحد تولیدکننده آدرس و محتوای ماتریس فاصله‌ها با رام تک درگاهه
۱۶۸
- شکل ۳-۶۸ واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه
۱۷۰
- شکل ۳-۶۹ ساختار واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه
۱۷۰
- شکل ۳-۷۰ واحد تولیدکننده ماتریس فاصله‌ها روی حافظه
۱۷۲
- شکل ۳-۷۱ نمودار ماشین حالت واحد تولیدکننده ماتریس فاصله‌ها روی حافظه
۱۷۲
- شکل ۴-۱ مسئله EIL51 به همراه پاسخ بهینه سراسری آن
۱۷۶
- شکل ۴-۲ مسئله BERLIN52 به همراه همراه پاسخ بهینه سراسری آن
۱۷۷
- شکل ۴-۳ مسئله TSP225 به همراه پاسخ بهینه سراسری آن
۱۷۷
- شکل ۴-۴ نمای کلی تراشه با ستون‌های ضرب‌کننده تعییه شده و بلوك‌های RAM
۱۷۹
- شکل ۴-۵ توابع تشکیل دهنده یک MAC
۱۸۰
- شکل ۴-۶ پاسخ الگوریتم BA در ۳۰۰۰ تکرار با استفاده از 2-OPT برای مسئله EIL51
۱۸۴
- شکل ۴-۷ پاسخ الگوریتم BA در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله EIL51
۱۸۴

- شکل ۴-۴ پاسخ الگوریتم BA در ۳۰۰۰ تکرار با استفاده از 2-OPT برای مسئله BERLIN52
- شکل ۹-۴ پاسخ الگوریتم BA در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله BERLIN52
- شکل ۱۰-۴ پاسخ الگوریتم BA در ۶۰۰۰ تکرار با استفاده از 2-OPT برای مسئله KROA100
- شکل ۱۱-۴ پاسخ الگوریتم BA در ۶۰۰۰ تکرار با استفاده از GSTM برای مسئله KROA100
- شکل ۱۲-۴ پاسخ الگوریتم CABC در ۳۰۰۰ تکرار با استفاده از 2-OPT برای مسئله EIL51
- شکل ۱۳-۴ پاسخ الگوریتم CABC در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله EIL51
- شکل ۱۴-۴ پاسخ الگوریتم CABC در ۳۰۰۰ تکرار با استفاده از 2-OPT برای مسئله BERLIN52
- شکل ۱۵-۴ پاسخ الگوریتم CABC در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله BERLIN52
- شکل ۱۶-۴ پاسخ الگوریتم CABC در ۶۰۰۰ تکرار با استفاده از 2-OPT برای مسئله KROA100
- شکل ۱۷-۴ پاسخ الگوریتم CABC در ۶۰۰۰ تکرار با استفاده از GSTM برای مسئله KROA100
- شکل ۱۸-۴ پاسخ الگوریتم BCO در ۳۰۰۰ تکرار با استفاده از 2-OPT برای مسئله EIL51
- شکل ۱۹-۴ پاسخ الگوریتم BCO در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله EIL51
- شکل ۲۰-۴ پاسخ الگوریتم BCO در ۳۰۰۰ تکرار با استفاده از 2-OPT برای مسئله BERLIN52
- شکل ۲۱-۴ پاسخ الگوریتم BCO در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله BERLIN52
- شکل ۲۲-۴ پاسخ الگوریتم BCO در ۶۰۰۰ تکرار با استفاده از 2-OPT برای مسئله KROA100
- شکل ۲۳-۴ پاسخ الگوریتم BCO در ۶۰۰۰ تکرار با استفاده از GSTM برای مسئله KROA100
- شکل ۲۴-۴ پاسخ الگوریتم HBA در ۳۰۰۰ تکرار با استفاده از 2-OPT برای مسئله EIL51
- شکل ۲۵-۴ پاسخ الگوریتم HBA در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله EIL51
- شکل ۲۶-۴ پاسخ الگوریتم HBA در ۳۰۰۰ تکرار با استفاده از 2-OPT برای مسئله BERLIN52
- شکل ۲۷-۴ پاسخ الگوریتم HBA در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله BERLIN52
- شکل ۲۸-۴ پاسخ الگوریتم HBA در ۶۰۰۰ تکرار با استفاده از 2-OPT برای مسئله KROA100
- شکل ۲۹-۴ پاسخ الگوریتم HBA در ۶۰۰۰ تکرار با استفاده از GSTM برای مسئله KROA100
- شکل ۱-۵ ساختار پیشنهادی

فهرست جداول

۶۰	جدول ۱-۳ الگوریتم زنبور برای حل مسائل ترکیبی
۶۲	جدول ۲-۳ الگوریتم کلونی زنبور مصنوعی برای حل مسائل ترکیبی
۶۴	جدول ۳-۳ الگوریتم بهینه‌سازی کلونی زنبور گسسته برای حل مسائل ترکیبی
۶۸	جدول ۴-۳ الگوریتم HBA برای حل مسائل ترکیبی
۱۱۳	جدول ۵-۳ مقایسه چند مسئله و بیشینه و کمینه فاصله شهرهای آن‌ها
۱۳۱	جدول ۶-۳ ترتیب حرکت و تابع شایستگی مسیر نزدیکترین همسایگی برای مسئله EIL51
۱۷۸	جدول ۱-۴ مشخصات تراشه مورد استفاده در پیاده‌سازی
۱۸۳	جدول ۲-۴ مسافت مسیر نهایی الگوریتم BA
۱۸۷	جدول ۳-۴ مسافت مسیر نهایی الگوریتم CABC
۱۹۱	جدول ۴-۴ مسافت مسیر نهایی الگوریتم BCO
۱۹۵	جدول ۵-۴ مسافت مسیر نهایی الگوریتم HBA
۲۰۰	جدول ۶-۴ منابع سخت‌افزاری استفاده شده توسط زیرهسته‌ها
۲۰۲	جدول ۷-۴ پارامتر IS، نتایج و منابع سخت‌افزاری مصرف شده در پیاده‌سازی هسته HBA
۲۰۳	جدول ۱-۵ مقایسه نتایج چهار الگوریتم زنبور
۲۰۵	جدول ۲-۵ مقایسه زمان اجرای چهار الگوریتم زنبور برای مسئله EIL51
۲۰۸	جدول ۳-۵ فضای مصرفی برای پیاده‌سازی الگوریتم ABC بر روی FPGA

چکیده

الگوریتم‌های بهینه‌سازی هوش جمعی از جمله الگوریتم‌های فرالبتکاری هستند که در حل بسیاری از مسائل مهم بکار رفته‌اند و بسی شک مشهورترین آن‌ها، مسئله فروشنده دوره‌گرد (TSP) است. الگوریتم‌های کلونی زنبور عسل از جمله این الگوریتم‌ها هستند که از رفتار طبیعی زنبورهای عسل برای جستجوی منابع غذایی الهام گرفته شده‌اند و معروف‌ترین آن‌ها الگوریتم زنبور (BA)، کلونی زنبور مصنوعی (ABC) و بهینه‌سازی کلونی زنبور (BCO) است. در این تحقیق امکان پیاده‌سازی الگوریتم‌های کلونی زنبور عسل بر روی بستر سخت‌افزاری آرایه‌های دریچه‌ای برنامه‌پذیر در محل (FPGA) بررسی شده است. بدین منظور الگوریتم زنبور سخت‌افزاری (HBA) ارائه و این الگوریتم در بستر نرم‌افزار و سخت‌افزار برای حل مسئله فروشنده دوره‌گرد پیاده‌سازی شده است. ساختار الگوریتم پیشنهادی مناسب پیاده‌سازی بر روی بستر FPGA بوده و با هدف اشغال کمترین فضای سخت‌افزاری به همراه بیشترین سرعت، سخت‌افزار متناظر آن با روش‌های ابتکاری و در سه سطح گیت، رفتار و ساختار طراحی شده است. همچنین به منظور مقایسه کارایی و زمان اجرای الگوریتم پیشنهادی، سه الگوریتم BA، ABC و BCO بر روی بستر نرم‌افزاری پیاده‌سازی و نتایج این سه روش با پیاده‌سازی نرم‌افزاری و سخت‌افزاری الگوریتم HBA مقایسه شده‌اند. در کنار نتایج مناسب الگوریتم ارائه شده در نرم‌افزار، پیاده‌سازی سخت‌افزاری آن نیز نشان داد که می‌توان در عین مصرف کمینه‌ منابع سخت‌افزاری، به کاهش چشمگیری در زمان حل مسئله در کنار نتایج قابل قبول دست یافت.

کلمات کلیدی: الگوریتم‌های کلونی زنبور عسل، الگوریتم زنبور سخت‌افزاری، آرایه‌های دریچه‌ای برنامه‌پذیر در محل، مسئله فروشنده دوره‌گرد، پیاده‌سازی

۱. بیان مسئله

۱-۱ فرضیه تحقیق

۱-۱-۱ الگوریتم‌های بهینه‌سازی هوش جمعی

مفهوم بهینه‌سازی آن است که در بین پارامترهای یکتابع به دنبال مقادیری باشیم که تابع را کمینه یا بیشینه نماید. کلیه مقادیر مناسب جهت این امر را راه حل‌های ممکن و بهترین مقدار از این مقادیر را راه حل بهینه می‌نامند. الگوریتم‌های بهینه‌سازی هر دو نوع مسائل بیشینه‌سازی و کمینه‌سازی را پوشش می‌دهند. بهینه‌سازی کاربردهای زیادی در زمینه تخصیص منابع، زمان‌بندی‌ها و تصمیم‌گیری‌ها دارد. روش‌ها و الگوریتم‌های بهینه‌سازی به دو دسته تقسیم‌بندی می‌شوند:

- الگوریتم‌های دقیق^۱
- الگوریتم‌های تقریبی^۲

الگوریتم‌های دقیق قادر به یافتن جواب بهینه به صورت دقیق هستند، اما در مورد مسائل بهینه‌سازی سخت کارایی ندارند و زمان حل آن‌ها در این مسائل به صورت نمایی افزایش می‌یابد. الگوریتم‌های تقریبی قادر به یافتن جواب‌های خوب (نزدیک به بهینه) در زمان حل کوتاه برای مسائل بهینه‌سازی سخت هستند.

¹ Exact Algorithms

² Approximate Algorithms

الگوریتم‌های تقریبی خود به دو دسته طبقه‌بندی می‌شوند:

- الگوریتم‌های ابتکاری^۱
- الگوریتم‌های فرآبتكاری^۲

دو مشکل اصلی الگوریتم‌های ابتکاری، قرار گرفتن آن‌ها در بهینه‌های محلی و ناتوانی آن‌ها برای کاربرد در مسائل گوناگون است. الگوریتم‌های فرآبتكاری برای حل مشکلات الگوریتم‌های ابتکاری ارائه شده‌اند. در واقع الگوریتم‌های فرآبتكاری، یکی از انواع الگوریتم‌های بهینه‌سازی تقریبی هستند که دارای راهکارهای بروزرفت از بهینه محلی است و قابل کاربرد در طیف گسترده‌ای از مسائل هستند.

الگوریتم‌های بهینه‌سازی هوش جمعی از جمله الگوریتم‌های فرآبتكاری است که از طبیعت الهام گرفته شده‌اند تا روابط پیچیده را با شرایط اولیه و ساده درک نمایند. هوش جمعی نوعی هوش مصنوعی مبتنی بر رفتارهای ساده عامل‌ها که به طور محلی با یکدیگر و با محیط خود در تعامل هستند، است که بدون وجود یک کنترل کننده مرکزی منتهی به پاسخ‌های بهینه می‌شود. هر الگوریتم با ایجاد یک جمعیت اولیه از راه‌حل‌های کاربردی آغاز می‌گردد و به صورت مکرر از نسلی به نسلی دیگر در جهت بهترین راه‌حل حرکت می‌کند. از جمله موفق‌ترین روش‌های هوش جمعی که تاکنون ارائه شده‌اند می‌توان به موارد زیر اشاره کرد:

- بهینه‌سازی کلونی مورچه‌ها [۱]، [۲]، [۳]
- بهینه‌سازی اجتماع ذرات [۴]، [۵]
- بهینه‌سازی کلونی زنبور عسل [۶]، [۷]
- بهینه‌سازی کرم شب‌تاب [۸]، [۹]
- بهینه‌سازی سیستم دفاعی مصنوعی [۱۰]، [۱۱]

بهینه‌سازی کلونی مورچه‌ها یا به اختصار ACO، به مجموعه‌ای از الگوریتم‌های بهینه‌سازی مبتنی بر هوش دسته جمعی و احتمالات اطلاق می‌شود که ویژگی مشترک آن‌ها الهام گرفتن از روشی است

¹ Heuristic

² Metaheuristic

که مورچه‌ها در طبیعت از آن برای یافتن غذا استفاده می‌کنند. امروزه کاربرد اصلی الگوریتم بهینه‌سازی کلونی مورچه در یافتن مسیر بهینه (در اکثر موقع، همان کوتاهترین مسیر ممکن) بر روی گراف‌ها است که استفاده‌های فراوانی در حل مسائل پیچیده مهندسی از جمله در کنترل ترافیک دارد. بهینه‌سازی تراکم ذرات یا به اختصار PSO، یکی دیگر از روش‌های بهینه‌سازی الهام گرفته از طبیعت است که برای حل مسائل بهینه‌سازی عددی با فضای جستجوی بسیار بزرگ یا اصطلاحاً ابرفضاً، بدون نیاز به اطلاع از گرادیان تابع هدف ابداع شده است. این روش که اولین بار در سال ۱۹۹۵ توسط دو نفر به نام‌های کنی و ابرهارت ابداع شد، در آغاز برای شبیه‌سازی پرواز دسته جمعی پرندگان مورد استفاده قرار گرفت، ولی پس از ساده‌سازی الگوریتم اولیه مشاهده شد که این الگوریتم در واقع نوعی حل بهینه‌سازی را انجام می‌دهد و به همین دلیل می‌تواند برای حل سایر مسائل بهینه‌سازی هم مورد استفاده قرار گیرد.

الگوریتم‌های کلونی زنبور به طور کلی به مجموعه‌ای از الگوریتم‌های فراتکاری مبتنی بر هوش دسته جمعی اطلاق می‌شود که با الهام از زندگی اجتماعی زنبورها برای حل مسائل بهینه‌سازی ابداع شده‌اند. الگوریتم کلونی زنبور اساساً یک روش بهینه‌سازی مبتنی بر احتمالات است. امروزه از بهینه‌سازی کلونی زنبور در حل مسائل بهینه‌سازی پیوسته از قبیل آموزش شبکه‌های عصبی، طراحی بهینه اجزای مکانیکی و الکترونیکی و نیز در حل مسائل بهینه‌سازی ترکیبی از جمله بهینه‌سازی سرورهای اینترنت و مسئله فروشنده دوره‌گرد استفاده می‌شود.

الگوریتم کرم شبتاب یا به اختصار FA، یکی از جدیدترین الگوریتم‌های بهینه‌سازی مبتنی بر هوش دسته جمعی است که اولین بار در سال ۲۰۰۸ توسط فردی به نام یانگ معرفی شد. این الگوریتم برگرفته از نحوه جستجوی مکان بهینه در کرم‌های شبتاب بوده و در آن کرم‌های شبتاب از طریق انتشار نور به تبادل اطلاعات با یکدیگر به منظور پیدا کردن نقاط بهینه می‌پردازند.

سیستم‌های ایمنی مصنوعی رده‌ای از الگوریتم‌های تکاملی هستند که از سیستم ایمنی بدن جانداران الهام گرفته شده‌اند. سیستم ایمنی بدن سازوکاری است که با تکثیر به موقع سلول‌های تدافعی آنتی‌بادی، مانع رشد و تکثیر سلول‌های بیماری‌زا آنتی‌ژن در بدن می‌شود.

۲-۱-۱ پیاده‌سازی الگوریتم‌های هوش جمعی بر روی FPGA

تاکنون معروفترین الگوریتم‌های بهینه‌سازی هوش جمعی به منظور پیاده‌سازی بر روی FPGA استفاده شده است که از جمله آن‌ها می‌توان به موارد زیر اشاره کرد:

۱-۱-۲-۱ بهینه‌سازی کلونی مورچه‌ها بر روی FPGA

مرکل^۱ و همکاران کلیت پیاده‌سازی الگوریتم ACO بر روی FPGA را در مرجع [۱۲] ارائه داده‌اند. اولین پیاده‌سازی حقیقی الگوریتم ACO با نگاشت بر روی FPGA در مراجع [۱۳] و [۱۴] ارائه شده است که در آن نشان داده شده است که نوع متنکی بر جمعیت الگوریتم P-ACO نسبت به روش ACO هماهنگی بیشتری با FPGA با توجه به محدودیت‌های آن به منظور پیاده‌سازی نرم‌افزاری الگوریتم در کامپیوترهای ترتیبی بیشتر است. همچنین شورمن^۲ و همکاران در مرجع [۱۵] با تغییراتی بر روی الگوریتم P-ACO پیاده‌سازی آن بر روی FPGA را بهینه کرده‌اند. در مرجع [۱۶] هم معماری دیگری از پیاده‌سازی الگوریتم کلونی مورچه ارائه شده است که با تخصیص فضای سخت‌افزاری برای محاسبات اعشاری و اختصاص کد شناسایی برای ذخیره میزان فروممن تنها بر روی یک SRAM مشکلات پیشین نظری زمان زیاد اجرا و کندی سیر تکاملی را بهبود بخشیده است. یوشیکاوا^۳ و همکاران در مرجع [۱۷] با حذف محاسبات اعشاری و اختصاص یک جدول جستجو روش جدیدی برای پیاده‌سازی الگوریتم مورچه منطبق با سخت‌افزار ارائه داده‌اند که H-ACO نامیده می‌شود [۱۸].

۱-۱-۲-۲ بهینه‌سازی اجتماع ذرات

مرجع [۱۹] از الگوریتم اجتماع ذرات برای معکوس‌سازی شبکه عصبی و بدست آوردن پارامترهای بهینه شبکه عصبی در شبیه‌سازی محیط زیر آب بوسیله سیستم سونار استفاده نموده است و این الگوریتم را بر روی FPGA پیاده‌سازی کرده است. در مرجع [۲۰] به منظور بهینه کردن پارامترهای

¹ Merkle

² Scheuermann

³ Yoshikawa

دو تثیت‌کننده منبع تغذیه از الگوریتم PSO استفاده شده است و سپس الگوریتم بر روی یک DSP^۱ پیاده‌سازی شده است. در مرجع [۲۱] الگوریتم بهینه‌سازی اجتماع ذرات براساس رفتار کوانتوسومی^۲ ارائه شده است که از پارامترهای کمتری برخوردار است و همگرایی جهانی بهتری نسبت به الگوریتم بهینه‌سازی اجتماع ذرات استاندارد^۳ را دارد، سپس این الگوریتم بر روی FPGA به منظور کاربردهای بی‌درنگ یا زمان حقيقی^۴ پیاده‌سازی شده است. توبييني^۵ و همکاران در مرجع [۲۲] الگوریتم PSO را برای تعیین پارامترهای کنترل کننده PID یک روبات با چهار درجه آزادی استفاده و سپس آن را بر روی FPGA پیاده‌سازی کردند.

با توجه به مطالب ذکر شده در می‌یابیم که می‌توان الگوریتم کلونی زنبورها را بر روی FPGA پیاده‌سازی کرد تا که با دادن جنبه سخت‌افزاری به این الگوریتم بهینه‌سازی مهم، مشکلاتی همچون نیاز به کامپیوتر، حجم بالای سخت‌افزار مورد نیاز و ترتیبی بودن نحوه اجرای این الگوریتم بر روی نرم‌افزار را رفع نمود.

به منظور سنجش کارایی این الگوریتم‌ها از مسائل استاندارد و پرکاربرد استفاده می‌شود که فروشنده دوره‌گرد از جمله مهمترین و مورد مناقشه‌ترین مسائل مسیریابی است. فروشنده دوره‌گرد مسئله بهینه‌سازی است که در آن هدف گذر از تعداد مشخصی شهر به گونه‌ای است که این فروشنده کمترین مسیر را پیموده و از هر شهر تنها یک مرتبه گذر کند.

در مرجع [۲۳] نشان داده شده است که الگوریتم کلونی زنبورهای عسل و الگوریتم کلونی مورچه‌ها در حل مسئله TSP به بهترین نتایج دست یافته‌اند. پس با توجه به فرضیه‌های بیان شده نتیجه گرفته می‌شود که می‌توان الگوریتم کلونی زنبورها را بر روی بستر سخت‌افزاری قدرتمندی همچون FPGA برای مسئله‌ای با ساختار مناسب برای این الگوریتم یعنی مسئله فروشنده دوره‌گرد^۶ پیاده‌سازی کرد.

¹ Digital Signal Processor

² PQSO

³ PSO

⁴ Online

⁵ Thweny

⁶ Travelling Salesman Problem (TSP)

۱-۲ اهمیت و ضرورت انجام تحقیق

با توجه به پیشرفت دنیای دیجیتال و همچنین عدم پاسخ‌گویی مدارات مجتمع معمولی برای پیاده‌سازی توابع پیچیده، به مرور تراشه‌های جدیدی طراحی شده‌اند که با توجه به نیاز طراح، دارای تعداد زیادی گیت و بلوک منطقی هستند. از این میان PLA، PLD، PAL، SPLD، CPLD و FPGAها مشخص‌ترین نوع این تراشه‌ها هستند که تعداد گیت‌های به کار رفته در آن‌ها تا چندین میلیون گیت می‌رسد.

FPGAها^۱ که بعبارتی “آرایه دریچه‌ای برنامه‌پذیر میدانی” و به تعبیری ساده‌تر “آرایه گیت منطقی برنامه‌پذیر در محل” خوانده می‌شوند، تراشه‌هایی هستند با معماری داخلی از پیش تعیین شده توسط شرکت سازنده که قابلیت پیکربندی به منظورهای مختلف را توسط طراحان فراهم می‌آورند. در ابتدای ظهورFPGAها یعنی اواسط دهه ۱۹۸۰، از آن‌ها برای پیاده‌سازی ماشین‌هایی با پیچیدگی متوسط و پردازش داده‌های نسبتاً کم استفاده می‌شد. در اوائل دهه ۱۹۹۰ و با پیشرفت FPGAها، از آن‌ها برای ارتباط و شبکه، یعنی پردازش بلوک‌های بزرگ داده و فرستادن آن‌ها به اطراف استفاده می‌شد و در اواخر دهه ۱۹۹۰، بازار شاهد ورود آن‌ها به کاربردهای صنعتی، لوازم خانگی و خودروسازی بود. اما امروزه از FPGAها تقریباً برای پیاده‌سازی هر چیزی مانند دستگاه‌های مخابراتی، رادیوهای نرم‌افزاری، رادارها، پردازش تصویر و دیگر کاربردهای پردازش سیگنال^۲ و حتی قطعات SOC^۳ حاوی عناصر نرم‌افزاری و سخت‌افزاری استفاده می‌شود. در چند سال اخیر پیشرفت‌های زیادی در این زمینه صورت گرفته است که مهمترین آن‌ها عبارتند از: گنجایش بیشتر، سرعت بیشتر، راحتی استفاده و کم شدن هزینه.

هنگامی که یک تراشه برنامه‌پذیر از کارخانه تولیدی خارج می‌شود، هیچ عمل مشخصی را انجام نمی‌دهد. در عوض بوسیله خریدار تراشه برنامه‌ریزی می‌شود تا یک عملیات مورد نیاز را برای یک کاربرد مشخص انجام دهد. این قطعات دارای بلوک‌های منطقی برنامه‌پذیر و اتصالات بین بلوکی

¹ Field Programmable Logic Gate Array

² DSP

³ System On Chip

قابل پیکربندی هستند. برخی از آنها تنها یک بار قابلیت برنامه‌پذیری دارند که به^۱ OTP مشهورند و برخی دیگر چندین بار قابلیت برنامه‌پذیری را دارا هستند. این تراشه‌ها می‌توانند چند صد میلیون گیت منطقی قابل پیکربندی داشته باشند که همین ویژگی آنها را برای پیاده‌سازی توابع پیچیده و بسیار بزرگ مناسب می‌کند.

تعريفی دیگر که در FPGA به آن اشاره شد، "آرایه گیت منطقی برنامه‌پذیر در محل" بود، به عبارت دیگر می‌توان گفت FPGA‌ها^۲ هستند، یعنی برنامه‌پذیر درون سیستمی. ISP به قطعاتی گفته می‌شود که توان برنامه‌پذیری هنگام استقرار در سیستمی سطح بالاتر را داشته باشند. همین ویژگی، امکان تغییر طرح پیاده‌سازی شده را بصورت ساده برای ما فراهم می‌آورد، بدون آنکه نیاز به تولید نسخه جدیدی باشد.

برای برنامه‌نویسی و طراحی FPGA‌ها از دو روش زبان‌های توصیف سخت‌افزار (HDL-VHDL) و یا طراحی مدار استفاده می‌شود. این عمل توسط نرم‌افزارها و ابزارهای برنامه‌ریزی مختص به هر خانواده انجام می‌شود که توسط شرکت‌های سازنده در اختیار طراحان قرار می‌گیرد. به عنوان نمونه MAXPLUS و QUARTUS ابزارهای برنامه‌ریزی FPGA‌ها و CPLD‌های ساخت شرکت ALTERA و ISE ابزار برنامه‌ریزی CPLD‌ها و FPGA‌ها ساخت شرکت XILINX هستند. سال‌ها پیش که طراحی دیجیتال پا به عرصه وجود نهاد و تراشه‌های استانداردی چون گیت‌ها، فلیپ‌فلاب‌ها، لچ‌ها، شمارنده‌ها و ... ساخته شدند تصور روزی که فاصله نرم‌افزار و سخت‌افزار به حد کنونی برسد به طوری که تمامی مرزهای طراحی را درنوردیده و سخت‌افزار به انعطاف‌پذیری نرم‌افزار درآید بسیار دشوار بود. حال آنکه با وجود سخت‌افزارهایی قبل انعطاف همچون FPGA‌ها می‌توان این مرز را کمتر کرده و سهولت کاربری و قدرت نرم‌افزار را می‌توان بر روی سخت‌افزار داشت، نگاشت الگوریتم‌های پرکاربرد بر روی سخت‌افزار به امری مهم تبدیل شده است.

معمولًا الگوریتم‌های هوش جمعی بر روی کامپیوترهای ترتیبی و به صورت نرم‌افزاری پیاده‌سازی می‌شوند. ولیکن اگر زمان پردازش پارامتری تعیین کننده باشد، دو روش اصلی برای افزایش سرعت

¹ One-Time Programmable

² In-System Programmable

فرآیند وجود دارد. یکی اجرای الگوریتم‌های هوش جمعی بر روی چندین کامپیوتر که به صورت موازی کار می‌کنند و روش دیگر که بسیار مورد اطمینان است نگاشت مستقیم این الگوریتم‌ها بر روی سخت‌افزار است که به موجب آن می‌توان از ویژگی‌های موازی‌سازی و خط لوله معماري ساخت افزار بهره برد.

الگوریتم فراباتکاری کلونی زنبورها از جمله الگوریتم‌های مهم و بسیار پرکاربرد هوش جمعی در حل مسائل بهینه‌سازی است. با توجه به اینکه الگوریتم‌های هوش جمعی که از کارکرد ساده و موازی عامل‌های کلونی تشکیل یافته‌اند، می‌توان به این نتیجه دست یافت که پیاده‌سازی این الگوریتم بر روی بستر سخت‌افزاری که از مزیت موازی‌سازی بهره می‌برد امری مهم و مورد نیاز در تحقیقات و صنعت است.

۳-۱ اهداف تحقیق

- امکان پیاده‌سازی هر مسئله بهینه‌سازی بر روی سخت‌افزار با استفاده از الگوریتم کلونی زنبورها
- کاهش حجم سخت‌افزار مورد نیاز
- موازی‌سازی و حل مشکل اجرای ترتیبی در نرم‌افزار

۲. مقدمه

۱-۲ مسئله فروشنده دوره‌گرد

بدون شک معروفترین مسئله بهینه‌سازی ترکیبی مسئله است موسوم به مسئله فروشنده دوره‌گرد یا به اختصار TSP. در این مسئله، نقشه‌ای که در آن موقعیت تعدادی شهر (گاهی به همراه جاده‌های متصل کننده آنها و جهت هر یک) مشخص شده است به یک فروشنده دوره‌گرد داده و از او خواسته می‌شود که از یک شهر شروع به حرکت کرده و مرتباً از شهری به شهر دیگر برود به طوری که اولاً از هر شهر دقیقاً یک بار عبور کند و ثانیاً در نهایت به همان شهری باز گردد که در آغاز از آن شروع به حرکت کرده است. در این مسئله هدف از بهینه‌سازی یافتن کوتاهترین مسیر موجود از میان تمام مسیرهای ممکن است.

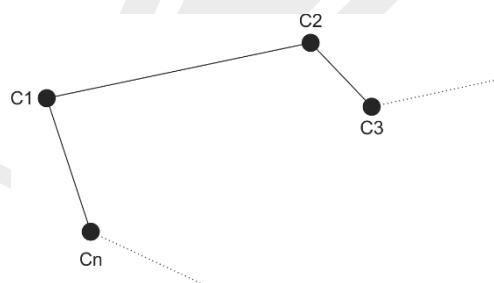
در عمل، مسئله TSP به اشکال مختلفی بیان می‌شود به عنوان مثال، از نظر ریاضی می‌توان مسئله TSP را بر روی گرافی مانند (E, n) تعریف کرد که در آن n مجموعه گره‌های گراف است که دارای n عضو (در واقع n شهر یا گره) است و E نیز مجموعه یال‌هایی است که این شهرها را به یکدیگر متصل می‌کنند. گراف مورد استفاده در این مسئله همیشه گرافی از نوع کامل فرض می‌شود، یعنی فرض بر این است که در این مسئله هر شهر اقلًاً با یک خط به شهر دیگری متصل شده است. همچنین در بیشتر مواقع خط یا خطوط واصل بین هر دو شهر دلخواه بدون جهت فرض می‌شود. معنی این گفته آن است که بر روی چنین خط یا خطوط واصلی فروشنده دوره‌گرد می‌تواند در هر دو جهت رفت و آمد نماید. البته مسئله TSP بر روی گراف‌های جهت‌دار نیز قابل تعریف است، هر چند که حالت اخیر کاربرد کمتری دارد. در هر صورت، هرگاه در مسئله TSP خطوط واصل بین شهرها از

پیش مشخص شده باشد فروشنده دوره‌گرد لزوماً باید با گذشتن از راههای داده شده مسیر خود را تکمیل نماید.

در اکثر مواقع در مسئله TSP فقط موقعیت شهرها (ونه خطوط واصل بین آنها) داده می‌شود. در این گونه موقع فروشنده دوره‌گرد برای رسیدن از شهری به شهر دیگر با گذر از روی خط مستقیم واصل بین آن دو شهر این کار را انجام می‌دهد. با کمی محاسبه می‌توان نشان داد که در این حالت در مسئله شامل n شهر تعداد کل مسیرهای ممکن برابر با $\frac{(n-1)!}{2}$ است که یکی از آنها جواب بهینه سراسری مورد نظر است. در این نوشته هر جا سخن از مسئله TSP به میان می‌آید منظور مسئله‌ای است که در آن فقط موقعیت شهرها (ونه خطوط واصل بین آنها) داده شده است مگر آنکه خلاف آن به صراحت ذکر شود.

جواب یک مسئله TSP مشتمل بر n شهر را می‌توان با یک تور^۱ نظیر (c_1, c_2, \dots, c_n) نمایش داد. معنی این طرز نمایش این است که فروشنده دوره‌گرد ابتدا از شهر c_1 به شهر c_2 ، سپس از شهر c_2 به شهر c_3 و به همین ترتیب جابجا می‌شود تا در نهایت به شهر c_n برسد. پس از آن فروشنده دوباره به شهر c_1 باز می‌گردد (شکل ۱-۲). طول^۲ این تور $(*)$ ، به صورت معادله ۱-۲ تعریف می‌شود:

$$TL(c_1, c_2, \dots, c_n) = \sum_{i=1}^n d(c_i, c_{i+1}) \quad \text{معادله ۱-۲}$$



شکل ۱-۲ تور $(*)$ در یک مسئله TSP

¹ Tour

² Length

که در آن $c_1 = c_{n+1}$ و $(*,*)^d$ فاصله (معمولًاً اقلیدسی) بین دو شهر است. هدف از بهینه‌سازی نیز پیدا کردن تور بهینه $T = (c_1, c_2, \dots, c_n)$ است به طوری که به ازای هر تور قابل قبول دیگری نظری T داشته باشیم:

$$TL(T^*) \leq TL(T) \quad \text{معادله ۲-۲}$$

مسئله TSP به وضوح یک مسئله بهینه‌سازی ترکیبی است، زیرا جواب‌های ممکن برای مسئله به صورت تورهایی هستند که ماهیت گستته و شمارش پذیر دارند و این جواب‌ها یا تورها باید یک تابع هزینه پیوسته معین را (که برابر با طول مسیر پیموده شده است) کمینه نمایند.

توجه داشته باشید که برای حل یک مسئله TSP معمولًاً نمی‌توان تمام تورهای ممکن را با استفاده از کامپیوتر محاسبه و طول‌های آنها را با هم مقایسه کرد. دلیل این گفته آن است که با افزایش تعداد شهرها (یعنی n) تعداد تورهای ممکن به سرعت افزایش می‌یابد. به عنوان مثال در حالت $n = 20$

فضای جستجو شامل

$$n! = 2432902008176640000 \approx 2.4 \times 10^{18} \quad \text{معادله ۳-۲}$$

حالت است (با فرض اینکه موقعیت شهر مبدا را ثابت در نظر نگیریم و از حالت‌های تکراری نیز چشم پوشی نکنیم). در این حالت اگر محاسبه و مقایسه هر تور تنها یک نانو ثانیه طول بکشد، محاسبه تمام تورهای ممکن به زمانی بیش از هفتاد سال نیاز خواهد داشت! بدیهیست که با اندکی افزایشی در تعداد شهرها وضع از این هم بدتر خواهد شد.

توجه شود که مسئله فروشنده دوره‌گرد در عمل نیز دارای کاربردهای مفید و جالبی است. به عنوان مثال اگر یک پستچی یا توزیع کننده مواد غذایی بخواهد بار خود را به طور بهینه در خانه‌ها یا مغازه‌های مختلفی در سطح شهر پخش کند. بدیهیست که در این حالت تحویل بار به خانه‌ها یا مغازه‌ها با یک ترتیب مناسب تاثیر قابل ملاحظه‌ای در کاهش مسافت طی شده، سوخت مصرفی و زمان تلف شده خواهد داشت. به عنوان یک کاربرد عملی دیگر، در هر خط تولید مدارات الکترونیکی پس از تهیه بورد مدار چاپی باید عملیات سوراخکاری و نصب قطعات بر روی بورد انجام گیرد. در مدارات پیچیده معمولًاً تعداد نقاطی که باید عملیات سوراخکاری و نصب قطعه بر روی آنها انجام گیرد بسیار

زیاد است. به همین دلیل در چنین مداراتی انجام عملیات سوراخکاری با یک ترتیب مناسب می‌تواند تاثیر قابل توجهی در کاهش زمان تلف شده برای انجام این کار داشته باشد. به عنوان یک کاربرد دیگر از مسئله TSP می‌توان به ربات جوشکار اشاره کرد؛ مثلاً در خط تولید اتومبیل، ربات جوشکار طبق برنامه‌ای از پیش تعیین شده به نقاط مختلفی بر روی بدن اتومبیل سر زده و در هر یک از آن‌ها یک جوش نقطه‌ای انجام می‌دهد. در اینجا نیز با مسئله‌ای از نوع فروشنده دوره‌گرد مواجه هستیم با این تفاوت که نقاط به جای صفحه در فضای سه بعدی قرار گرفته‌اند و محدودیت‌های ناشی از عدم برخورد ربات با بدن نیز به عنوان قیدهای مسئله عمل کرده و مسیرهای حرکت را به طور غیرمستقیم محدود می‌کنند (بیات، ۱۳۹۳).

FPGA ۲-۲

FPGA ۱-۲-۲ چیست؟

آرایه‌های گیت برنامه‌پذیر در محل^۱ مدارهای مجتمعی هستند که بلوک‌های منطقی برنامه‌پذیر و اتصالات بین بلوکی قابل پیکربندی دارند. طراحان می‌توانند این قطعات را برای انجام کارهای بسیار متنوعی پیکربندی یا برنامه‌ریزی کنند.

بسته به روش ساخت FPGA برخی از آن‌ها فقط یکبار و برخی دیگر چندین بار قابلیت برنامه‌ریزی دارند. قطعه‌ای که فقط یکبار قابلیت برنامه‌ریزی دارد OTP^۲ نامیده می‌شود. در نام FPGA بخش برنامه‌پذیر در محل به این حقیقت اشاره دارد که در مقایسه با قطعاتی که عملکرد داخلی آن‌ها توسط سازنده و به طور ثابت برنامه‌ریزی می‌شود، برنامه‌ریزی FPGA در محل صورت می‌پذیرد. یک مفهوم آن می‌تواند این باشد که FPGA در آزمایشگاه پیکربندی می‌شود و یا به اصلاح به عملکرد یک قطعه مستقر در سیستمی الکترونیکی اشاره دارد که قبلًا در محیط خارج از

¹ FPGA

² One-Time Programmable

آزمایشگاه نصب شده است. اگر بتوان قطعه‌ای را حین استقرار در سیستمی سطح بالاتر، برنامه‌ریزی کرد، به آن قطعه برنامه‌پذیر درون سیستمی^۱ گفته می‌شود (ماکسیلید، ۱۳۸۹).

۲-۲-۲ جذایت FPGA در چیست؟

در دنیای دیجیتال آی‌اسی‌های مختلفی از جمله منطق ژله‌ای (قطعات کوچکی که چند تابع منطقی ساده و ثابت دارند)، قطعات حافظه و ریزپردازنده‌ها وجود دارند. اما در این نوشتار، قطعات منطقی - برنامه‌پذیر^۲، مدارهای مجتمع با کاربرد خاص^۳ و البته قطعات FPGA مورد نظر است. PLD‌ها قطعاتی هستند که معماری داخلی آن از قبل توسط سازنده تعیین می‌شود، اما به گونه‌ای ساخته می‌شوند که طراحان بتوانند برای انجام کارهای مختلف آن‌ها را در محل پیکربندی کنند. هرچند این قطعات در مقایسه با FPGA‌ها تعداد نسبتاً محدودی گیت منطقی دارند و توابعی که پیاده می‌کنند بسیار ساده‌تر و کوچک‌تر است. در سوی دیگر این طیف، قطعات ASIC و FPGA قرار دارند که می‌توانند چند صد میلیون گیت منطقی داشته باشند و برای پیاده‌سازی توابع پیچیده و بسیار بزرگ استفاده می‌شوند.

اگرچه قطعات ASIC بیشترین تعداد ترانزیستور، پیچیدگی و کارایی (سرعت) را ارایه می‌دهند، اما فرآیند طراحی و ساخت آن‌ها بسیار زمان‌بر و پرهزینه است و افزون بر آن این عیب را دارد که طرح نهایی در سیلیکون ثابت می‌شود و بدون ایجاد نسخه جدیدی از قطعه قابل تغییر نیست. بنابراین، قطعات FPGA در جایگاهی بین PLD‌ها و ASIC‌ها قرار می‌گیرند؛ زیرا عملکردشان مانند PLD قابل سفارشی کردن در محل است، اما می‌توانند میلیون‌ها گیت منطقی داشته باشند و برای پیاده‌سازی توابع بسیار بزرگ و پیچیده‌ای که قبلاً فقط با استفاده از ASIC پیاده می‌شدند، بکار برده شوند.

هزینه یک طرح FPGA به مراتب پایین‌تر از طرح ASIC مشابه آن است، اما در تولید انبوه قیمت واحد ASIC بسیار کمتر از FPGA است. در عین حال، تغییر طرح پیاده شده در FPGA بسیار ساده-

¹ ISP

² PLD

³ ASIC

تر و زمان رسیدن به بازار برای چنین طرحی بسیار کوتاه‌تر است. از این رو FPGA‌ها موجب رشد و ترقی بسیاری از شرکت‌های طراحی کوچک و نو پا می‌شوند، زیرا طراحان می‌توانند بدون اجبار برای متحمل شدن هزینه‌های زیاد NRE¹ با خریداری مجموعه ابزارهای گران قیمت لازم برای طراحی ASIC، شخصاً² و یا در قالب گروههای کوچک مهندسی راهکارهای سختافزاری و نرمافزاری خود را در یک محیط آزمایشی مبتنی بر FPGA تحقق بخسند (ماکسفیلد، ۱۳۸۹).

۲-۲-۳- برای چه کاری می‌توان از FPGA استفاده کرد؟

با معرفی قطعات FPGA در اواسط دهه ۱۹۸۰، این قطعات در پیاده‌سازی منطق اتصالی³ ماشین‌های حالت با پیچیدگی متوسط و پردازش داده‌های نسبتاً کم بطور گستردگی استفاده می‌شدند. در اوایل دهه ۱۹۹۰، با آغاز رشد اندازه و پیچیدگی قطعات FPGA، بازار فروش بزرگ آن‌ها در عرصه‌های ارتباط و شبکه بود که با پردازش بلوک‌های بزرگ داده و فرستادن آن به اطراف سروکار دارند. در اوخر دهه ۱۹۹۰ با یک جهش ناگهانی، FPGA در کاربردهای صنعتی، لوازم خانگی و خودروسازی مورد استفاده قرار گرفت.

غالباً از قطعات FPGA برای نمونه‌سازی طرح‌های ASIC با فراهم آوردن محیطی برای بازیابی پیاده‌سازی فیزیکی یک الگوریتم استفاده می‌شود. البته، هزینه پایین توسعه و کوتاه بودن زمان رسیدن آن‌ها به بازار بدین معنیست که بطور فزاینده‌ای راه خود را بسوی محصولات نهایی باز می‌کنند. در واقع، برخی عرضه‌کنندگان اصلی FPGA قطعاتی دارند که به عنوان رقیب مستقیم قطعات ASIC فروخته می‌شود.

در اوایل قرن ۲۱، FPGA‌هایی با کارایی بالا و حاوی میلیون‌ها گیت ساخته شدند. برخی از این قطعات ویژگی‌های برجسته هسته‌های ریزپردازنده تعبیه شده⁴، واسطه‌های ورودی/خروجی سرعت بالا و مشابه آن را ارائه می‌دهند. نتیجه نهایی این است که FPGA‌های امروزی تقریباً برای پیاده‌سازی هر چیزی، از جمله دستگاه‌های مخابراتی و رادیوهای نرمافزاری، رادار، پردازش تصویر و دیگر

¹ Non-Recurring Engineering

² Glue Logic

³ Embedded Microprocessor Cores

کاربردهای پردازش سیگنال دیجیتال^۱ و قطعات SoC^۲ حاوی عناصر نرم افزاری و سخت افزاری، قابل استفاده‌اند.

در حال حاضر FPGA‌ها در چهار بخش اصلی به فروش می‌رسند: ASIC و سیلیکون سفارشی، پردازش سیگنال دیجیتال، میکروکنترولرهای تعبیه شده و تراشه‌های ارتباطی لایه فیزیکی. علاوه بر این در محاسبات قابل پیکربندی مجدد^۳ بازار جدیدی برای خود ایجاد کرده‌اند.

- ASIC و سیلیکون سفارشی: همانطور که در بخش قبل بیان شد، FPGA‌های امروزی به طور فرایندهای برای پیاده‌سازی طرح‌های متنوعی استفاده می‌شوند که پیش از این فقط با استفاده از ASIC‌ها و سیلیکون سفارشی پیاده‌سازی می‌شدند.

- پردازش سیگنال دیجیتال: پردازش سیگنال دیجیتال سرعت بالا معمولاً به کمک ریزپردازنده‌های خاصی بنام پردازنده‌های سیگنال دیجیتال^۴ انجام می‌شود، اما FPGA‌های امروزی می‌توانند ضرب کننده‌های تعبیه شده، مسیریابی حسابی اختصاصی و مقدار زیادی RAM بر روی تراشه داشته باشند که تمام آن‌ها عملیات‌های DSP را تسهیل می‌کنند. وقتی این ویژگی‌ها با موازات فراهم شده توسط FPGA‌ها ترکیب می‌شوند، نتیجه آن ایفای نقش سریع‌ترین تراشه‌های DSP با ضریب ۵۰۰ برابر یا بیشتر است.

- میکروکنترولرهای تعبیه شده: معمولاً توابع عملیاتی کوچک توسط پردازنده‌های تعبیه شده خاصی به نام میکروکنترولر اداره می‌شوند. بر روی تراشه این قطعات ارزان قیمت، حافظه‌های برنامه و دستورالعمل، زمان‌سنج‌ها و I/O‌های جانبی در اطراف هسته پردازنده قرار دارند. قیمت FPGA‌ها در حال کاهش است و حتی کوچک‌ترین FPGA‌های امروزی توانایی لازم برای پیاده‌سازی یک هسته پردازنده نرم و ترکیب شده با مجموعه‌ای از توابع ورودی/خروجی سفارشی را دارد. در نتیجه، جذایت FPGA برای کاربردهای کنترلی تعبیه شده رو به افزایش است.

¹ DSP

² System on Chip

³ Reconfigurable Computing

⁴ Digital Signal Processors

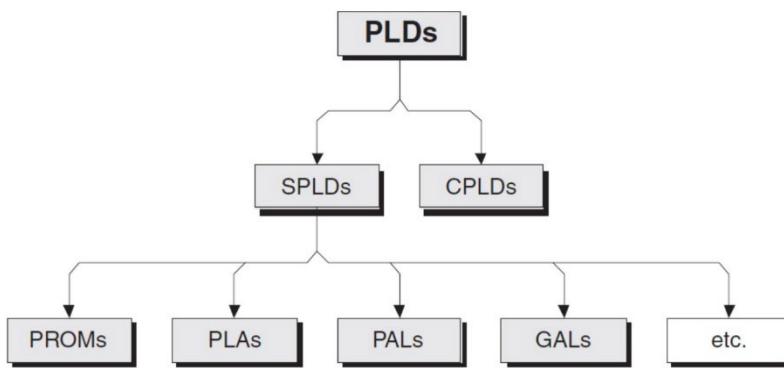
- ارتباطات لایه فیزیکی: FPGA ها سابقه‌ای طولانی در پیاده‌سازی منطق اتصالی دارند که تراشه‌های ارتباطی لایه فیزیکی و لایه‌های بالای پروتکل شبکه را به هم متصل می‌کند. اینکه FPGA های جدید امروزی می‌توانند چندین فرستنده/گیرنده سرعت بالا داشته باشند، بدین معنیست که توابع ارتباطی و شبکه‌ای را می‌توان در یک قطعه یکپارچه کرد.
- محاسبات قابل پیکربندی مجدد: محاسبه قابل پیکربندی به عملکرد موازی ذاتی و قابلیت پیکربندی مجدد فراهم شده توسط FPGA ها برای الگوریتم‌های نرم‌افزاری (شتاب سخت-افزاری) اشاره دارد. در حال حاضر شرکت‌های مختلفی موتورهای محاسباتی قابل پیکربندی بزرگی برای پیگیری فعالیت‌ها، از شبیه‌سازی سخت‌افزاری گرفته تا تحلیل رمزگاری برای کشف داروهای جدید می‌سازند (ماکسفلد، ۱۳۸۹).

٤-٢-٢ خواستگاه FPGA

١-٤-٢-٢ CPLD و SPLD

نخستین آی‌سی‌های برنامه‌پذیر را به طور کلی قطعات منطقی برنامه‌پذیر (PLD) می‌نامیدند. قطعات اولیه‌ای که در سال ۱۹۷۰ ظهور کردند نسبتاً ساده بودند و ارج و قرب زیادی داشتند، اما فقط تا اواخر دهه ۱۹۷۰ که نسخه‌های بسیار پیچیده‌تر در دسترس قرار گرفتند. برای تمایز آن‌ها از نسخه‌های قبلی ساده‌ترشان که همچنان تا به امروز نیز از آن‌ها استفاده می‌شود، این قطعات جدید PLD پیچیده (SPLD) نامیده شدند. شاید بهتر باشد نسخه‌های اولیه کم زرق و برق‌تر را PLD ساده (CPLD) نامید.

برخی افراد PLD و SPLD را هم معنی می‌انگارند و برخی دیگر PLD را شامل CPLD و SPLD می‌دانند. گاه طراحان از سرnamهای یکسانی برای قطعات متفاوت استفاده می‌کنند. برای مثال در مورد SPLD چندین معماری زیرین وجود دارد که سرnam آن‌ها از ترکیب مختلف سه یا چهار حرف یکسان تشکیل می‌شود (شکل ۲-۲).



شکل ۲-۲ تقسیم‌بندی قطعات منطقی برنامه‌پذیر

البته نسخه‌های FLASH E²PLD، EPLD و EPROM این قطعات نیز وجود دارند – برای مثال EPROM که برای وضوح بیشتر از شکل ۲-۲ حذف شده‌اند (ماکسفیلد، ۱۳۸۹).

PROM ۲-۴-۲-۲

نخستین PROM ساده PLD بود که در سال ۱۹۷۰ معرفی شد. برای آشنایی با طرز کار این قطعه می‌توان آن را حاوی یک آرایه ثابت از توابع AND در نظر گرفت که آرایه‌ای برنامه‌پذیر از توابع OR را راهاندازی می‌کند. پیوندهای برنامه‌پذیر آرایه OR را می‌توان به صورت پیوندهای فیوزی و یا در قطعات EPROM و E²PROM به ترتیب به صورت ترانزیستورهای EPROM و یا سلول‌های E²PROM پیاده‌سازی کرد.

قطعات PROM در آغاز به عنوان حافظه‌هایی برای ذخیره دستورالعمل‌های برنامه و مقادیر داده‌ای ثابت در نظر گرفته شده بودند. اما طراحان از آن‌ها برای پیاده‌سازی توابع منطقی ساده مانند جدول‌های جستجو و ماشین‌های حالت نیز استفاده می‌کردند. در حقیقت می‌توان برای پیاده‌سازی بلوک‌های منطقی ترکیبی که ورودی‌ها و خروجی‌های زیادی ندارند از PROM استفاده کرد (ماکسفیلد، ۱۳۸۹).

PLA ۳-۴-۲-۲

برای رفع محدودیت‌های تحمیلی توسط معماری PROM، آرایه منطقی برنامه‌پذیر (PLA) پله بعدی در نردنban ترقی PLD‌ها بود که برای نخستین بار در حدود سال ۱۹۷۵ در دسترس قرار گرفت. این

قطعات در مقایسه با PLDهای ساده قابلیت پیکربندی بالای داشتند، زیرا هر دو آرایه OR و AND برنامه‌پذیر بودند.

قطعات PLA هرگز نتوانستند سهم خوبی از بازار داشته باشند، اما چندین عرضه کننده فقط برای AND مدت کوتاهی جنبه‌های دیگر PLA را امتحان کردند. برای مثال، در PLA ها اجباراً آرایه‌های آرایه‌های OR را تغذیه نمی‌کردند و به ندرت در معماری‌های دیگری آرایه‌های NOR آرایه‌های AND را تغذیه می‌کردند. اگرچه از نظر تئوری داشتن معماری‌هایی از قبیل AND-NOR و NAND-NOR ممکن است، اما این انواع مختلف به ندرت در دسترس بودند و یا اصلاً ساخته نمی‌شدند. یکی از دلایل استفاده معماری AND-OR در این قطعات این بود که معادلات منطقی به شکل جمع حاصل ضرب‌ها را می‌توان مستقیماً به این معماری تصویر کرد. قالب-های دیگر نمایش معادلات مانند ضرب حاصل جمع‌ها را می‌توان به کمک روش‌های جبری استاندارد تطبیق داد.

استفاده از PLAها به خصوص در طرح‌های بزرگی که در معادلات منطقی آن‌ها جملات حاصل-ضرب مشترک زیادی بین چندین خروجی وجود داشت بسیار توصیه می‌شد. جنبه منفی PLA این است که در مقایسه با قطعات متناظرش که پیش از این تعریف شدند، زمان عبور سیگنال‌ها از پیوندهای فیوزی آن بیشتر است. بنابراین قابل برنامه‌ریزی بودن آرایه‌های AND و OR بدین معنی است که PLAها به میزان قابل توجهی کندر از PROMها هستند (ماکسفیلد، ۱۳۸۹).

GAL و PAL ۴-۴-۲-۲

برای برطرف کردن مشکلات سرعتی PLAها، در اوخر دهه ۱۹۷۰ گروه جدیدی از قطعات به نام منطق آرایه‌ای برنامه‌پذیر (PAL) معرفی شد. از نقطه نظر مفهومی، PAL دقیقاً در نقطه مخالف PROM است، زیرا آرایه AND برنامه‌پذیر و آرایه OR از پیش تعریف شده دارد. مزیت PAL در مقایسه با PLA این است که سریع‌تر است زیرا فقط یکی از آرایه‌های آن برنامه‌پذیر است. عیب PAL این است که این قطعه محدودتر شده، زیرا به تعداد محدودی جمله حاصل ضرب اجازه می‌دهد با هم OR شوند (ماکسفیلد، ۱۳۸۹).

CPLD ۵-۴-۲-۲

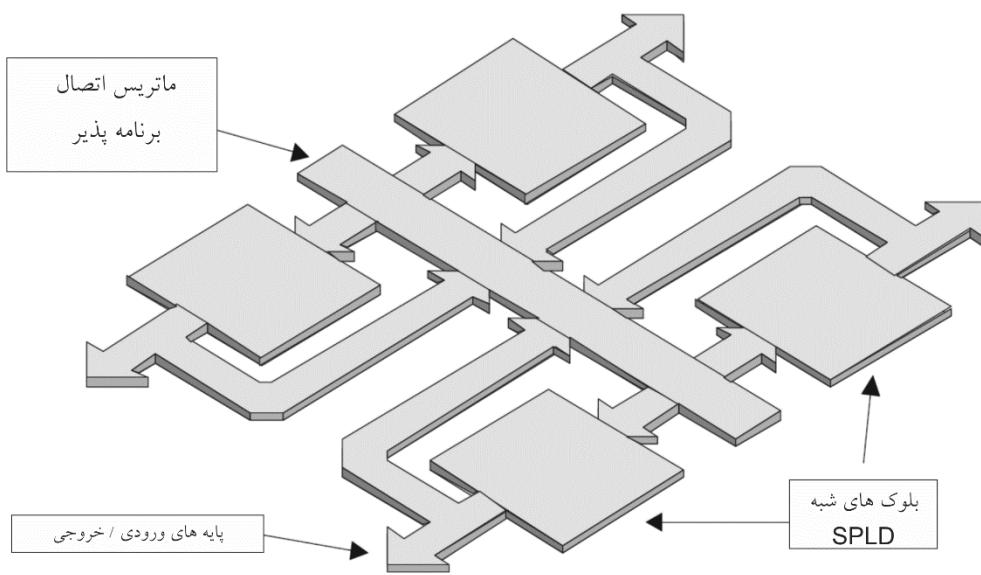
یک حقیقت غیر قابل انکار در الکترونیک این است که افراد همواره به دنبال چیزهای بزرگ‌تر (بر حسب توانایی‌های عملکردی)، کوچک‌تر (بر حسب اندازه فیزیکی)، سریع‌تر، قدرتمند‌تر و ارزان‌تر هستند. بنابراین، در اوخر دهه ۱۹۷۰ و اوایل ۱۹۸۰ قطعات PLD پیچیده‌تری ظهر کردند که PLD پیچیده یا CPLD نامیده می‌شوند.

پس از معرفی پر یاهوی قطعات PAL اولیه توسط MMI¹، این شرکت قطعه‌ای به نام MegaPAL معرفی کرد. قطعه جدید دارای ۸۴ پایه بود که چهار PAL استاندارد همراه با اتصالات پیوند دهنده آن‌ها به هم داشت. متاسفانه، مصرف توان MegaPAL مناسب نبود و در کل، استفاده از آن در مقایسه با چهار قطعه جداگانه مزایای کمی داشت.

در سال ۱۹۸۴، با معرفی یک CPLD بر مبنای ترکیبی از فناوری‌های CMOS و EEPROM توسط شرکت Altera گام بزرگ برداشته شد. به کارگیری CMOS به آلترا امکان داد به پیچیدگی و تراکم عملیاتی بسیار زیادی همراه با مصرف توان به نسبت کم دست یابد. قرار دادن قابلیت برنامه‌پذیری این قطعات بر مبنای سلول‌های EEPROM آن‌ها را برای استفاده در محیط‌های توسعه و نمونه‌سازی ایده‌آل ساخت.

البته شهرت آلترا فقط به خاطر ترکیب CMOS و EEPROM نیست. وقتی مهندسان معماری‌های SPLD را در قطعات بزرگ‌تری از قبیل MegaPAL رشد دادند، نخست می‌اندیشیدند که آرایه اتصال مرکزی (که ماتریس اتصال برنامه‌پذیر نیز نامیده می‌شود) پیوند دهنده بلوک‌های مجزای SPLD به اتصال ۱۰۰ درصد ورودی‌ها و خروجی‌های هر بلوک نیاز دارد. مشکل این بود که با دو برابر شدن اندازه بلوک‌های SPLD (معادل با دو برابر شدن ورودی‌ها و خروجی‌ها) اندازه آرایه اتصال چهار برابر می‌شد. به نوبه خود، چهار برابر شدن اندازه آرایه اتصال باعث افت بسیار زیادی در سرعت همراه با توان مصرفی بالاتر و افزایش قیمت قطعه می‌شد. آلترا آرایه اتصال مرکزی را با اتصال کمتر از ۱۰۰ درصد ساخت.

¹ Monolithic Memories Inc.



شکل ۲-۳ ساختار عام یک CPLD

گرچه هریک از سازندگان CPLD از معماری منحصر به فرد خود استفاده می‌کنند، اما هر قطعه عام چند بلوک SPLD (به طور معمول چند PAL) دارد که یک ماتریس اتصال برنامه‌پذیر مشترک دارند (شکل ۲-۳). علاوه بر برنامه‌ریزی بلوک‌های SPLD مجزا، توسط ماتریس اتصال برنامه‌پذیر نیز می‌توان اتصال بین بلوک‌ها را برنامه‌ریزی کرد (ماکسیلد، ۱۳۸۹).

FPGA ۶-۴-۲-۲

تقریباً در اوایل دهه ۱۹۸۰ مشهود بود که در زنجیره آی‌اسی‌های دیجیتال یک شکاف ایجاد شده است. از یک سو، قطعات برنامه‌پذیر مانند SPLD‌ها و CPLD‌ها بودند که قابلیت پیکربندی بالا و زمان‌های طراحی و اصلاح سریعی داشتند، اما توابع پیچیده یا بزرگ را پشتیبانی نمی‌کردند.

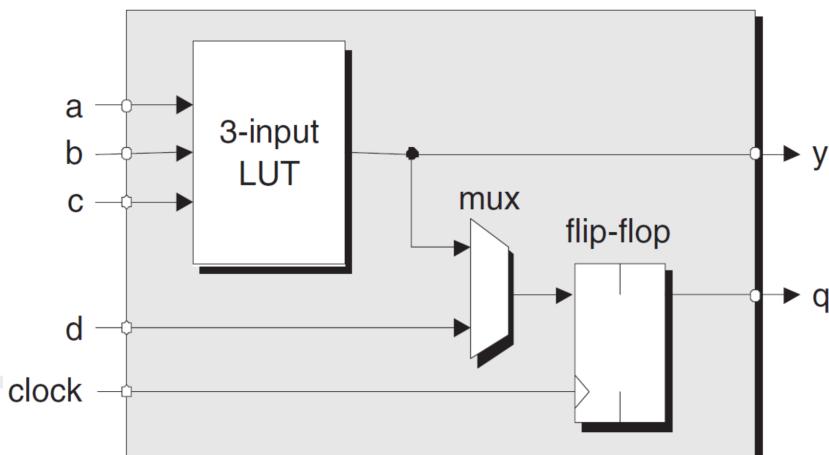
در سوی دیگر این طیف ASIC‌ها بودند. این قطعات می‌توانستند توابع بسیار بزرگ و پیچیده را پشتیبانی کنند، اما بسیار گران قیمت بوده و طراحی زمان‌بری داشتند. گذشته از این، وقتی طرحی به صورت ASIC پیاده شود در سیلیکون ثبیت شده است و قابل تغییر نیست.

به منظور پر کردن این فاصله، Xilinx یک کلاس جدید آی‌سی به نام آرایه گیت برنامه‌پذیر در محل FPGA توسعه داد که در ۱۹۸۴ به بازار عرضه شد.

نخستین FPGA مبتنی بر CMOS بودند و برای پیکربندی از سلول‌های SRAM استفاده می‌کردند.

اگرچه این قطعات ابتدایی نسبتاً ساده بودند و به نسبت استانداردهای امروزی تعداد کمی گیت داشتند، اما بسیاری از جنبه‌های معماری زیرین آن‌ها همچنان به کار گرفته می‌شوند.

قطعات اولیه مبتنی بر مفهوم بلوک منطقی برنامه‌پذیر بودند که یک جدول جستجوی^۱ ورودی^۲، یک ثبات که می‌توانست به صورت نگهدارنده^۲ یا فلیپ‌فلاب عمل کند و یک مالتی‌پلکسرا همراه با چند المان دیگر را شامل می‌شد. شکل ۴-۲ یک بلوک منطقی برنامه‌پذیر بسیار ساده را نشان می‌دهد (بلوک‌های منطقی دیگر در یک FPGA مدرن بسیار پیچیده‌تر هستند).



شکل ۴-۲ المان‌های کلیدی تشکیل دهنده یک بلوک منطقی برنامه‌پذیر ساده

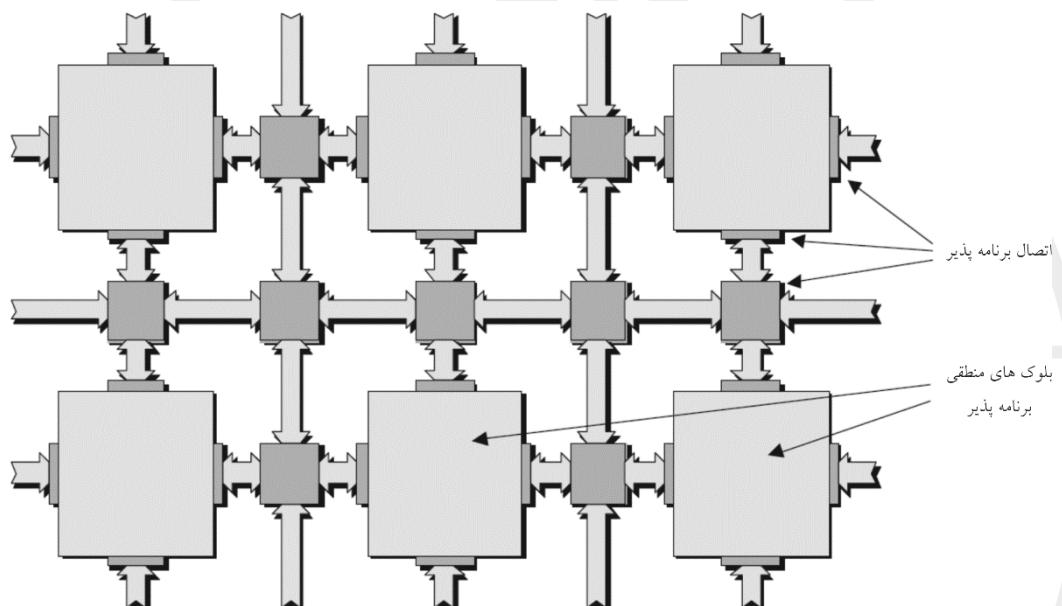
هر FPGA تعداد زیادی از این بلوک‌های منطقی برنامه‌پذیر دارد. بوسیله سلول‌های برنامه‌ریزی SRAM مناسب، هر بلوک منطقی را می‌توان برای انجام یکتابع متفاوت پیکربندی کرد. هر ثبات را می‌توان برای مقداردهی به ۰ یا ۱ منطقی و عمل کردن به صورت یک فلیپ‌فلاب یا نگهدارنده پیکربندی کرد. اگر گزینه فلیپ‌فلاب انتخاب شود، ثبات می‌تواند توسط یک لبه مثبت یا منفی ساعت تریگر شود (سیگنال ساعت برای تمام بلوک‌های منطقی یکسان است).

¹ LUT (Look Up Table)

² Latch

مالتی‌پلکسر تغذیه کننده فلیپ‌فلاپ می‌تواند برای قبول خروجی LUT یا یک ورودی جداگانه به بلوک منطقی پیکربندی شود و LUT را می‌توان برای نشان دادن تمام توابع منطقی^۳ ورودی پیکربندی کرد.

هر FPGA شامل تعداد زیادی بلوک منطقی برنامه‌پذیر است که همچون جزیره‌هایی توسط دریایی از اتصالات برنامه‌پذیر احاطه شده‌اند (شکل ۵-۲)



شکل ۵-۵ نمای بالا به پایین معماری عام FPGA

علاوه بر اتصالات محلی نشان داده شده در شکل ۵-۵، مسیرهای اتصال سراسری سرعت بالایی نیز وجود دارند که سیگنال را بدون نیاز به رفتن از طریق چند المان سویچینگ محلی در سراسر تراشه منتقل می‌کنند.

همچنین قطعه، پدها^۱ و پایه‌های I/O اصلی نیز دارد که در اینجا نشان داده نشده‌اند. بوسیله سلول‌های SRAM می‌توان اتصال را به گونه‌ای برنامه‌ریزی کرد که ورودی‌های اصلی قطعه به ورودی یک یا چند بلوک منطقی برنامه‌پذیر متصل شوند و خروجی‌های هر بلوک منطقی را نیز می‌توان برای درایو ورودی‌های یک بلوک منطقی دیگر، خروجی‌های اصلی قطعه و یا هر دو استفاده کرد.

¹ PAD

نتیجه نهایی این بود که FPGA‌ها به شکلی موفقیت‌آمیز فاصله بین PLD‌ها و ASIC‌ها را پر کردند. از یک نظر، کاملاً قابل پیکربندی بودند و زمان طراحی و اصلاح PLD‌ها را داشتند. از طرف دیگر، برای پیاده‌سازی توابع بزرگ و پیچیده که قبلًا فقط در حوزه ASIC بود قابل استفاده بودند (ASIC‌ها همچنان برای طرح‌های واقع‌بازگ و پیچیده، با کارآیی بالا لازم بودند، اما با افزایش پیچیدگی FPGA‌ها، هرچه بیشتر به فضای طراحی ASIC‌ها تجاوز کردند) (ماکسفلد، ۱۳۸۹).

۵-۲-۲ معماری FPGA

۱-۵-۲-۲ معماری درشت، متوسط و ریزدانه

به طور معمول قطعات FPGA به صورت ریزدانه یا درشت‌دانه طبقه‌بندی می‌شوند. برای اینکه مفهوم این طبقه‌بندی را درک کنیم، باید بدانیم که مهمترین ویژگی متمایز کننده FPGA‌ها از دیگر قطعات لایه زیرین ساخت آن‌هاست که غالباً تعداد زیادی بلوک منطقی برنامه‌پذیر نسبتاً ساده دارد که درون اتصالات برنامه‌پذیر بسیار زیادی تعییه شده‌اند (شکل ۲-۵).

در معماری ریزدانه، هر بلوک منطقی را فقط می‌توان برای پیاده‌سازی یک تابع بسیار ساده استفاده کرد. برای مثال، می‌توان بلوک را برای عمل کردن به صورت یک تابع^۱ ۳ ورودی، مانند یک گیت منطقی (NAND، OR، AND و غیره) یا یک المان ذخیره‌سازی (فلیپ‌فلاپ D، نگهدارنده D) پیکربندی کرد.

معماری‌های ریزدانه^۱ علاوه بر پیاده‌سازی منطق اتصالی و ساختارهای بسیار قاعده مانند ماشین‌های حالت، هنگام اجرای الگوریتم سیستولیکی نیز بازده خوبی دارند. این معماری‌ها برخی از مزایای فناوری ستز منطقی استاندارد را نیز ارائه می‌دهند که به سوی معماری‌های ریزدانه ASIC تمایل دارند.

در اواسط دهه ۱۹۹۰ علاقه زیادی به معماری‌های ریزدانه مشاهده شد، اما به مرور زمان فقط معماری‌های درشت‌دانه^۲ باقی ماندند. در معماری درشت‌دانه، هر بلوک منطقی در مقایسه با معماری-

¹ Fine-Grained

² Coarse-Grained

های متناظر ریزدانه، منطق نسبتاً زیادی دارد. برای مثال، هر بلوک منطقی می‌تواند حاوی چهار LUT چهار ورودی، چهار مالتی‌پلکسرا، چهار فلیپ‌فلاب D و مقداری منطق نقلی سریع باشد.

یک نکته مهم در ارتباط با دانه بودن معماری این است که پیاده‌سازی‌های ریزدانه، در مقایسه با عملکردی که توسط آن بلوک‌ها قابل پشتیبانی است، اتصالات ورودی/خروجی نسبتاً زیادی برای هر بلوک دارند. هر چه دانه بودن بلوک به سوی دانه متوسط^۱ و بالاتر پیش رود، در مقایسه با مقدار عملکردی که بلوک‌ها پشتیبانی می‌کنند، تعداد اتصالات آن‌ها کاهش می‌یابد. این نکته مهم است، زیرا اتصالات میان بلوک‌کی برنامه‌پذیر در سیگنال‌هایی که درون FPGA انتشار می‌یابند بیشترین تاخیر را ایجاد می‌کنند.

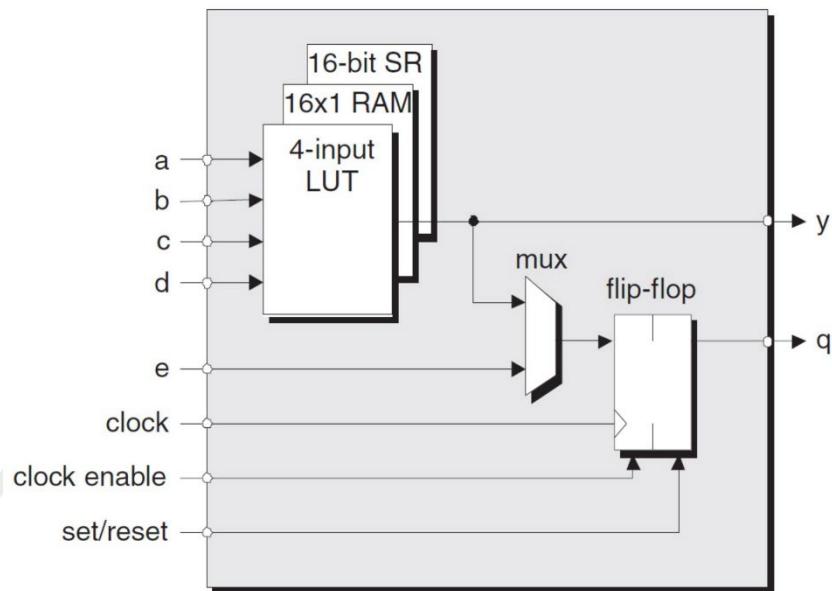
نکته دیگر این است که تعدادی از شرکت‌ها معماری‌های دانه درستی را تولید می‌کنند که آرایه‌ای از گره‌ها را شامل می‌شوند. هر یک از این گره‌ها یک المان پردازشی بسیار پیچیده است از یک تابع الگوریتمی مانند FFT گرفته تا یک هسته ریزپردازنده همه منظوره کامل. این قطعات در کلاس FPGA قرار نمی‌گیرند و به این دلیل معماری‌های FPGA مبتنی بر LUT غالباً به عنوان دانه متوسط طبقه‌بندی می‌شوند (ماکسفیلد، ۱۳۸۹).

۲-۵-۲-۲ سلول منطقی XILINX

در دنیای FPGA هر عرضه‌کننده از نام‌های خاصی برای اشیاء استفاده می‌کند و این کار کمی مشکل‌ساز است. جزء بنیادی هسته در یک FPGA پیشرفتۀ Xilinx سلول منطقی^۲ نامیده می‌شود. سلول منطقی شامل یک LUT چهار ورودی (که به صورت یک RAM 16×1 یا یک ثبات جابه‌جاوی ۱۶ بیتی عمل می‌کند)، یک مالتی‌پلکسر و یک ثبات است (شکل ۶-۲).

¹ Medium-Grained

² LC



شکل ۲-۶ نمای ساده شده یک Xilinx LC

تصویر نشان داده شده در شکل ۲-۶ بسیار ساده شده است، همچنین مشاهده می‌شود که ثبات را می‌توان به صورت یک فلیپ‌فلاپ یا یک نگهدارنده پیکربندی کرد. تریگر شدن با لبه بالارونده یا لبه پایین‌رونده کلک را می‌توان مانند پلاریته سیگنال‌های clock enable و set/reset (بالا فعال یا پایین فعال) پیکربندی کرد.

علاوه بر MUX، LUT و ثبات، المان‌های محدود دیگری از جمله مقداری منطق نقلی سریع ویژه برای استفاده در عملیات‌های حسابی دارد (ماکسفیلد، ۱۳۸۹).

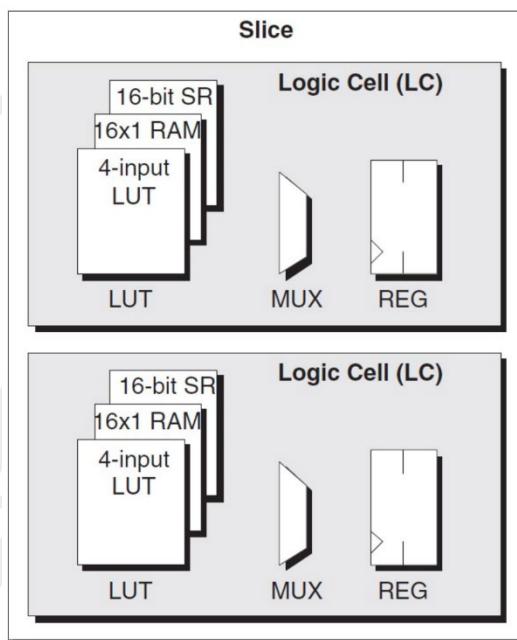
۲-۵-۳ المان منطقی Altera

جزء بنیادی هسته در FPGA‌های ساخت Altera المان منطقی^۱ نامیده می‌شود. بین سلول منطقی Altera و المان منطقی Xilinx تفاوت‌هایی وجود دارد، اما مفهوم کلی آن‌ها بسیار مشابه است (ماکسفیلد، ۱۳۸۹).

¹ LE

۴-۵-۲-۲ برش (Slice)

Xilinx مرحله بعدی در سلسله مراتب را برش می نامد (بی ترددی Altera و عرضه کنندگان دیگر نام های معادل خودشان را دارند). هر برش حاوی دو سلول منطقی است (شکل ۷-۲).



شکل ۷-۲ برش

برای سادگی سیم های داخلی از شکل حذف شده اند، توجه شود که اگرچه هر LUT، MUX و ثبات سلولی منطقی داده های ورودی و خروجی خود را دارند، برش یک مجموعه از سیگنال های clock^۱ و clock enable set/reset دارد که بین هر دو سلول منطقی مشترک هستند (ماکسفلد، ۱۳۸۹).

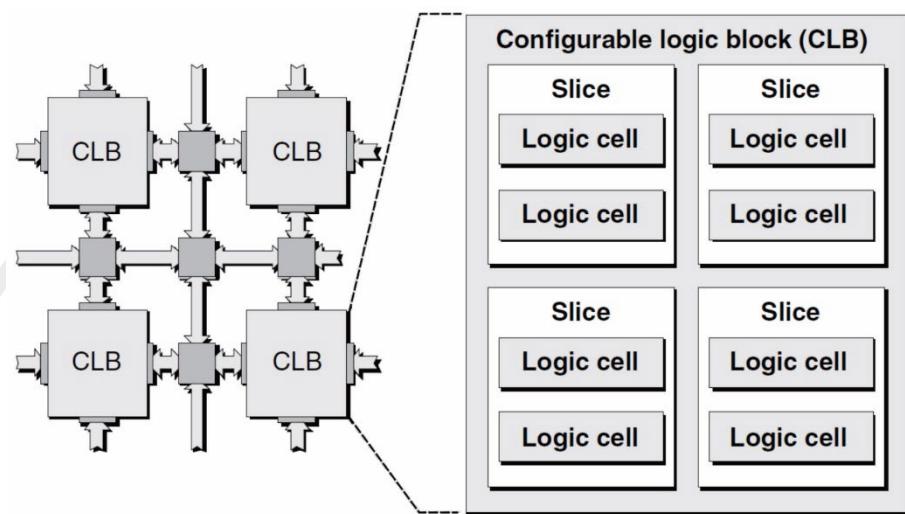
۵-۵-۲-۲ LAB و CLB

در یک مرحله بالاتر از سلسله مراتب به چیزی می رسیم که آن را بلوک منطقی قابل پیکربندی^۱ و Altera آن را بلوک آرایه منطقی^۲ می نامد. بی ترددی دیگر عرضه کنندگان FPGA نام های معادل خودشان را برای این موجودیت ها دارند.

¹ CLB

² LAB

برای مثال برخی از FPGA های Xilinx CLB در هر دو برش و برخی دیگر ۴ برش دارند. همچنین هر CLB یک تک بلوک منطقی برنامه پذیر است که در دریایی از اتصالات برنامه پذیر قرار دارد (شکل ۸-۲).



شکل ۸-۲ یک CLB حاوی چهار برش (تعداد برش ها به خانواده FPGA وابسته است) CLB اتصالات برنامه پذیر سریعی نیز دارد. از این اتصالات که در شکل ۸-۲ برای وضوح بیشتر نشان داده نشده اند، برای متصل کردن برش های مجاور استفاده می شود. دلیل این نوع سلسله مراتب بلوک منطقی ($LC \leq LC \leq LC \leq LC$) آن است که توسط یک سلسله مراتب معادل در اتصالات، تکمیل می شود. بنابراین، بین LC های یک برش یک اتصال سریع وجود دارد، سپس اتصال بین برش های یک CLB کمی کندتر است و به همین ترتیب اتصال بین CLB ها. هدف از این دستیابی به یک موازنۀ بهینه بین اتصال ساده اشیاء به هم بدون متحمل شدن تاخیر اتصالی بیش از اندازه است (ماکسفیلد، ۱۳۸۹).

۳-۲ کلونی زنبورها

همانطور که در فصل قبل بیان شد، الگوریتم های کلونی زنبور به طور کلی به مجموعه ای از الگوریتم های فرآیندی مبتنی بر هوش دسته جمعی اطلاق می شود که با الهام از زندگی اجتماعی زنبورها برای حل مسائل بهینه سازی ابداع شده اند. الگوریتم کلونی زنبور اساساً یک روش بهینه سازی

مبتنی بر احتمالات است. امروزه از بهینه‌سازی کلونی زنبور در حل مسائل بهینه‌سازی پیوسته از قبیل آموزش شبکه‌های عصبی، طراحی بهینه اجزای مکانیکی و الکترونیکی و نیز در حل مسائل بهینه‌سازی ترکیبی از جمله بهینه‌سازی سرورهای اینترنت و مسئله فروشنده دوره‌گرد استفاده می‌شود.

طی دهه گذشته الگوریتم‌های گوناگونی با الهام‌گیری از زندگی اجتماعی زنبورها برای حل انواع مختلفی از مسائل بهینه‌سازی ابداع شده‌اند که از میان آن‌ها می‌توان به سیستم زنبور^۱، الگوریتم BCO، الگوریتم ABC، MBO، الگوریتم زنبورها، الگوریتم HBMO، زنبور کندو، کلونی مصنوعی زنبور و الگوریتم VBA اشاره کرد. برخلاف بیشتر الگوریتم‌های فراابتکاری که در آن الگوریتم اصلی توسط یک فرد یا یک تیم از پژوهشگران ابداع و سپس توسط دیگران توسعه داده شده است، ایده استفاده از زندگی اجتماعی زنبورها برای حل مسائل بهینه‌سازی توسط چند تیم مستقل از پژوهشگران و به فواصل زمانی کوتاه ولی به شکل‌های مختلف ارائه شده است. در نتیجه این روش‌ها بعد‌ها نیز توسط محققان مختلف در مسیرهای متفاوتی توسعه یافته‌اند که همین امر باعث شده تا نتوان اکنون آن‌ها را در قالب یک الگوریتم واحد ارائه کرد (بیات، ۱۳۹۳).

۱-۳-۲ رفتارهای اجتماعی زنبورها

رفتار اجتماعی زنبورها در طبیعت که به یافتن جواب بهینه منجر می‌شود به سه دسته تقسیم می‌شوند. در ادامه این نوع رفتارها را مورد بررسی قرار می‌دهیم.

۱-۳-۱ رفتار جستجوی مکان کندو

در کندوهای زنبور در اوایل بهار، تخمک‌هایی برای تولد ملکه جدید گذاشته می‌شوند. قبل از تولید ملکه جدید، ملکه پیر کندو را با نیمی از اعضای آن برای تشکیل یک اجتماع جدید ترک می‌کند. آن‌ها به دنبال مکان جدیدی برای ساخت کندوی خود هستند. بدین منظور زنبورهای پیشاهنگ حدود دوازده محل جدید برای ساخت کندو را بررسی می‌کنند. مکان‌های کندوهای جدید به واسطه رقص‌های جنبشی این زنبورها معرفی می‌شوند. همچنین کیفیت رقص این زنبورها مستقیماً با کیفیت مکان

¹ BS

کندو مرتبط است. این زنبورهای پیشاهنگ پس از بازگشت از محل جستجو شده به رقص جنبشی پرداخته و در گذر زمان زنبورهایی که کیفیت محل پیدا شده توسط آنها کمتر است از رقص دست برداشته و در انتهای محل زنبور پیشاهنگ باقیمانده به عنوان کندوی جدید انتخاب می‌شود [۲۳].

۲-۱-۳-۲ رفتار ازدواج

پدیده تولیدمثل در اجتماع زنبور تحت مدیریت ملکه است. ملکه جوان پس از تولدش، در پروازهای جفتگیری شرکت خواهد کرد و در محل اجتماع به زنبورهای نر خواهد پیوست. ملکه با چندین زنبور نر در حال پرواز جفتگیری خواهد کرد که انتخاب زنبورهای نر توسط او بر اساس رقص زنبورهای نر یا به عبارتی کیفیت زنبور نر خواهد بود. سپس بعد از سه روز ملکه تخمگذاری می‌کند. تخمهای غیر بارور به زنبورهای نر تبدیل خواهند شد، در حالیکه تخم بارور بسته به کیفیت غذایی که به لارو داده می‌شود، به زنبور کارگر یا ملکه تبدیل می‌شوند. [۲۳].

۳-۱-۳-۲ رفتار جستجوی منابع غذایی

اولین مورد استفاده از الگوی غذایابی زنبورهای عسل برای حل مسائل مرتبط با بهینه‌سازی به سال ۱۹۹۷ باز می‌گردد که در آن ساتو و هاگیوارا سیستم زنبور^۱ را برای بهبود کارایی الگوریتم ژنتیک پیشنهاد دادند. مهم‌ترین رخداد پس از معرفی BS معرفی الگوریتم BCO در سال ۲۰۰۱ توسط لوشیچ و تئودورو ویچ بود که در آن این دو نفر از الگوی غذایابی زنبورهای عسل برای حل مسئله فروشنده دوره‌گرد و متعاقباً برای مدل‌سازی و حل مسائل مرتبط با مهندسی ترافیک استفاده کردند. اتفاق مهم بعدی در تاریخ بهینه‌سازی کلونی زنبور معرفی الگوریتم زنبورها توسط پم و همکاران در سال ۲۰۰۶ و ABC توسط باش ترکو کارابوگا در سال ۲۰۰۵ بود. در ابتدا از الگوریتم زنبورها برای بهینه‌سازی شبکه‌های عصبی و از ABC برای بهینه‌سازی عددی توابع استفاده می‌شد. از آن زمان تاکنون مرتبأ نمونه‌های کارامدتری از الگوریتم‌های بهینه‌سازی کلونی زنبور توسط محققان مختلف و به منظور حل مسائل گوناگون بهینه‌سازی پیشنهاد شده است که این رویه همچنان نیز ادامه دارد.

¹ BS

با توجه به وجود شباهت‌های زیاد بین الگوریتم‌های ذکر شده در قبل و عدم فراگیر شدن همه آن‌ها در محافل آکادمیک و نیز رشد و توسعه سریع این گونه الگوریتم‌ها، در ادامه این فصل تنها به الگوریتم‌های نسبتاً کلاسیک و جا افتاده یعنی BA، BCO و ABC خواهیم پرداخت. ولی قبل از آن در مورد نحوه جمع‌آوری گرده یا غذایابی توسط زنبورهای عسل در طبیعت توضیح خواهیم داد.

زنبورها با استفاده از نوعی همکاری جالب دسته‌جمعی موقعیت منابع غذایی را کشف و سپس از آن-ها بهره‌برداری می‌کنند. نحوه این همکاری در گونه‌های مختلف زنبور دارای تفاوت‌های اندکی است، هر چند که همگی آن‌ها از اصول تقریباً یکسانی پیروی می‌کنند. یکی از جالب‌ترین این همکاری‌های دسته‌جمعی به منظور کشف، جمع‌آوری و فرآوری شهد و گرده گل‌ها در زنبورهای عسل دیده می‌شود. ناگفته پیداست که چون منابع غذایی موجود در محیط پیرامون کندو¹ با هر دو عامل زمان و مکان تغییر می‌کنند و از طرفی زنبورها نیز از قدرت بینایی بسیار محدودی برخوردار هستند، لذا کشف آن‌ها نیاز به مهارت بالایی دارد. برای این منظور زنبورها نیز همانند بسیاری از دیگر جانداران در همکاری با یکدیگر سطحی از هوش را از خود بروز می‌دهند که هر یک به تنها یی فاقد آن هستند. در هر کندوی عسل هر زنبوری که قصد جمع‌آوری غذا یعنی شهد یا گرده را داشته باشد، ترجیحاً این کار را با دنبال کردن یکی از زنبورهایی که قبلاً موفق به پیدا کردن آن شده است انجام می‌دهد. برای این منظور هر یک از زنبورهای مسئول جمع‌آوری گرده، پس از یافتن و جمع‌آوری گرده بلاfacile به کندو بازگشته و سایر زنبورهای درون کندو را در مورد موقعیت، کیفیت و کمیت منبع گرده‌ای که موفق به کشف آن شده با استفاده از نوعی رقص موسوم به رقص جنبشی² آگاه می‌سازد. در واقع زنبورها بر خلاف مورچه‌ها بدون علامت‌گذاری در محیط پیرامون و به طور مستقیم به تبادل اطلاعات با یکدیگر می‌پردازند. رقص زنبورها در قسمت خاصی از کندو موسوم به سالن رقص و با هدف ترغیب بقیه زنبورها به پیروی از زنبور رقصنده انجام می‌شود. در نتیجه این عمل، هر زنبوری که به قصد جمع‌آوری گرده تصمیم به ترک کندو بگیرد، به احتمال زیاد یکی از این زنبورهای راقص را دنبال خواهد کرد.

¹ Hive

² Waggle Dance

در هر صورت، هر زنبور پس از رسیدن به منبع گرده مقداری گرده برداشته و به کندو باز می‌گردد.

پس از بازگشت هر زنبور به کندو یکی از سه حالت زیر ممکن است رخ دهد:

- زنبور از منبع گردهای که از آن بازگشته صرفنظر کند و به یک زنبور پیرو^۱ تبدیل شود.
- زنبور به جمع‌آوری گرده از همان منع ادامه دهد بدون آنکه تلاشی برای جذب سایر زنبورها به پیروی از خود کند.
- زنبور به جمع‌آوری گرده از همان منع ادامه دهد و در عین حال با رقصیدن خود سایر زنبورهای کندو را نیز به پیروی از خود ترغیب کند.

در واقع بسته به میزان جذابیت منبع غذا (فاصله تا کندو، کیفیت، مقدار و غیره) هر زنبور با یک احتمال معین تصمیم به انجام یکی از سه عمل فوق می‌گیرد. مکانیزمی که طی آن زنبورها انتخاب خود را انجام می‌دهند هنوز دقیقاً شناخته شده نیست، ولی آنچه که بدیهیست این است که احتمال انجام هر یک از سه عمل فوق بستگی به کیفیت منبع گردهای که زنبور با آن سروکار داشته است دارد. به هر حال، فرآیند فوق مرتباً تکرار می‌شود تا وقتی که شهد یا گرده موجود در منبع کشف شده به پایان برسد و زنبورها تصمیم به جستجو برای یافتن سایر منابع احتمالی غذا بگیرند. با توجه به بحث فوق، در هر کلونی زنبور نوعی فیدبک مثبت در فرآیند غذایابی وجود دارد. این فیدبک مثبت باعث تقویت مسیرهایی می‌شود که به منابع غذایی غنی‌تر متوجه می‌شوند. به طور مشابه، فیدبک منفی موجود در فرآیند غذایابی زنبورهای عسل نیز باعث کنار گذاشتن منابع غذایی کم ارزش‌تر می‌گردد.

یک دانشمند اکولوژیست و برنده‌ی نوبل اتریشی به نام کارل ون فریش^۲ اولین کسی بود که زبان رقص جنبشی زنبورهای عسل را به زبان قابل درک برای انسان‌ها ترجمه کرد. به طور کلی، مطابق شکل ۹-۲ رقص زنبورهای عسل کارگر در الگوهایی به شکل هشت انگلیسی انجام می‌شود. برای این زنبور ابتدا زنبور رقصندۀ به طور زیگزاگ و تابدار در یک مسیر تقریباً مستقیم (در واقع بر روی کمر هشت) می‌دود. این بخش از حرکت را اصطلاحاً فاز جنبش^۳ می‌نامند. پس از رسیدن به انتهای

¹ Follower

² Karl Von Frisch

³ Waggle Phase

مسیر زنبور به سمت راست چرخیده و با یک حرکت نیم دایره‌ای به ابتدای مسیر باز می‌گردد که این بخش از حرکت را اصطلاحاً فاز بازگشت^۱ می‌نامند. سپس دوباره زنبور به صورت زیگزاگ در همان مسیر مستقیم قبلی حرکت کرده ولی این بار پس از رسیدن به انتهای مسیر به سمت چپ می‌چرخد و با طی یک مسیر نیم دایره‌ای شکل به ابتدای مسیر باز می‌گردد. بسته به نوع اطلاعاتی که زنبور رقصنده قصد انتقال آن‌ها را به سایر زنبورها دارد این فرآیند ممکن است از یک بار تا بیش از صد بار تکرار شود (شکل ۹-۲).

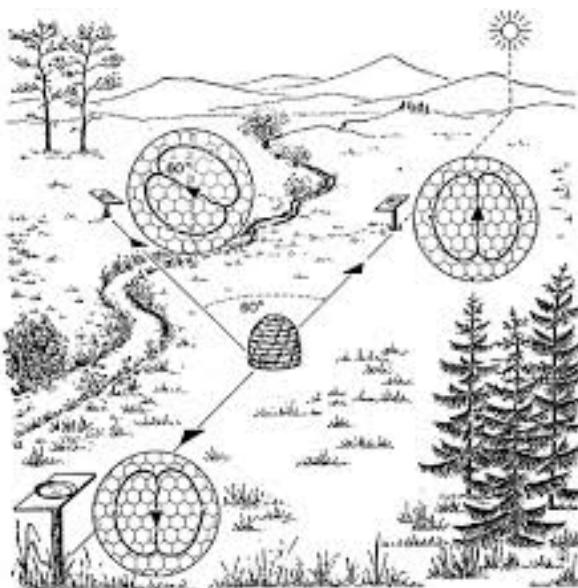


شکل ۹-۲ رقص پیچشی زنبورهای عسل

نکته بسیار جالب اینکه جهت و مدت زمان دویدن زیگزاگ هر زنبور رقصنده ارتباط نزدیکی با جهت و فاصله منبعی که زنبور در حال تبلیغ آن است دارد. زنبورهایی که درون حفره‌ها زندگی می‌کنند عمل رقصیدن را با حرکت بر روی شانه‌هایی که در جهت عمودی یا شاقول در داخل کندو قرار دارند انجام می‌دهند. به عبارت دقیق‌تر اگر یک گل یا منبع غذایی درست در امتداد خط مستقیمی که از در کندو تا خورشید امتداد دارد قرار گرفته باشد، زنبور رقصنده دقیقاً در جهت عمودی حرکت خواهد کرد (شکل ۱۰-۲). اما اگر منبع غذایی نسبت به خورشید در زاویه مشخصی قرار گرفته باشد، زنبور رقصنده نیز مطابق شکل با همان زاویه نسبت به مسیر قائم حرکت خواهد کرد. به عنوان مثال در شکل ۱۰-۲ حرکت زیگزاگ با زاویه ۶۰ درجه نسبت به راستای قائم، موقعیت یک منبع غذایی در راستای ۶۰ درجه نسبت به خط کشیده شده از در کندو تا موقعیت خورشید را آدرس‌دهی می‌کند. تا

^۱ Return Phase

به اینجای کار مربوط به نحوه آدرس دهی راستای منبع غذایی است. فاصله بین کندو و منبع غذایی نیز توسط طول مسیر و مدت زمان دویدن زیگزاگی شکل کدگزاری می شود؛ به طوری که هر چقدر نقطه هدف دورتر باشد فاز جنبش و به تبع آن فاز برگشت در موقع رقصیدن طولانی تر خواهد بود.



شکل ۱۰-۲ ارتباط بین زاویه مسیر حرکت زیگزاگ و موقعیت غذایی

زیبایی زندگی دسته جمعی زنبورها به اینجا محدود نمی شود. یکی دیگر از نکات جالب توجه در ارتباط با زندگی جمعی زنبورها این است که هر چقدر یک زنبور رقصیده در مورد منبعی که یافته هیجان زده تر باشد، شدت پیچشها و زیگزاگ هایش را بیشتر می کند تا زنبورهای بیشتری را به خود جلب کرده و بیشتر آنها را به پیروی از خود قانع کند. بدین ترتیب اگر چندین زنبور به طور همزمان در حال رقصیدن باشند، بین آنها رقابتی در جهت قانع کردن و جلب توجه سایر زنبورها درمی گیرد. در این رقابت ممکن است زنبورها مسیر یکدیگر را قطع کنند یا حتی با یکدیگر درگیر شوند (زیبایی این عمل در اینجاست که این درگیری به هر حال ریسکی را متوجه خود زنبور می کند، در حالیکه زنبور از آن نفع شخصی نمی برد). همچنین اگر زنبوری برای یک مدت طولانی در حال رقصیدن باشد، زاویه رقصیدنش را به آرامی و به گونه ای تغییر می دهد که تغییرات رخ داده در موقعیت خورشید در طول زمان رقصیدن در آن لحاظ شود. با این کار علیرغم تغییر جهت خورشید زنبورهای پیرو همچنان موفق به یافتن منبع غذایی خواهند شد.

برقراری فیدبک مثبت در جانوران دارای زندگی اجتماعی اگرچه دارای فوائد غیرقابل انکاری برای این جانوران است، اما همزمان می‌تواند خطراتی را نیز برایشان به دنبال داشته باشد. به عنوان مثال ممکن است یک شکارچی بالقوه نیز با کشف ساز و کار اجتماعی زنبورها، در محل غذاهای بهتر کمین کند و پس از آنکه زنبور رقصنده سایر زنبورها را به محل مورد نظر هدایت کرد به راحتی آن‌ها را شکار کند. به همین دلیل زنبورهای عسل باید در کنار فیدبک مثبت فیدبک منفی را نیز برقرار کنند. پژوهشگران مشاهده کرده‌اند که در صورت قرار دادن زنبور مرده بر روی یک گل، زنبورهای بازگشته از آن گل رقص‌های کمتری را از خود بروز می‌دهند. چنین به نظر می‌رسد که زنبورهای کارگر وجود زنبور مرده بر روی گل را به عنوان نشانه‌ای از یک شکارچی در پیرامون منبع غذایی تعبیر کرده و در نتیجه علاقهٔ کمتری به هدایت سایر زنبورها به آن نقطه از خودشان نشان می‌دهند.

اما شاید حیرت‌انگیزترین بخش از فرآیند غذایابی ذکر شده در اینجا مربوط به مکانیزم ارتباطی بین زنبور رقصنده و زنبورهای تماشاگر درون کندو باشد. شاید در نگاه اول چنین به نظر برسد که زنبورهای مستقر در اطراف زنبور رقصنده با نگاه کردن به زنبور راقص در مورد دنبال کردن یا نکردن او تصمیم‌گیری می‌کنند (ارتباط از طریق چشم و دیدن). اما مطالعات گسترده نشان می‌دهند که احتمالاً در بین حواس مختلف زنبورهای تماشاگر در این مرحله حس بینایی کمترین تاثیر را در فرآیند دارد (فضای درون کندوی مورد بحث عملاً تاریک و فاقد نور است). توضیح بهتر این است که زنبورها در موقع پرواز (یا تکان دادن و مالیدن اجزای بدنشان به یکدیگر، به هر دلیل) بار الکتریکی قابل ملاحظه‌ای را در بدن خود جمع می‌کنند که همین امر باعث می‌شود موقع انجام رقص جنبشی میدان‌های الکتریکی مدوله شده‌ای (متغیر با زمان، ترکیب فرکانس بالا و پایین) را از خود ساعت کنند. با توجه به قانون فاراده، این میدان‌های الکتریکی متغیر با زمان منتشر شده توسط زنبورهای رقصنده به زنبورهای تماشاگر اطراف ولتاژهای الکتریکی القا می‌کند. به عبارت دقیق‌تر، تاژکهای باردار موجود بر روی بدن زنبورهای اطراف همانند نوعی آتن عمل می‌کند، به طوری که هر چقدر شدت جنبش زنبور رقصنده بیشتر باشد به همان نسبت میزان ولتاژ القا شده به زنبورهای تماشاگر اطراف نیز بیشتر خواهد بود. در واقع هر دو عامل میدان الکتریکی متغیر با زمان و ارتعاشات

مکانیکی ناشی از تکان خوردن مولکول‌های هوا (به خاطر حرکت بال‌ها) باعث انتقال پیام می‌شوند. به همین دلیل بود که قبلاً گفته شد که زنبور در مورد منبع غذایی که پیدا کرده هیجان‌زده‌تر باشد در موقع رقص باشدت بیشتری بال‌ها و بدنش را تکان می‌دهد. با توجه به توضیحات فوق عده‌ای از پژوهشگران نقش میدان‌های الکتریکی را در برقراری نوعی نظام ارتباطات اجتماعی بین زنبورها بسیار مهم دانسته‌اند.

در پایان باید به این نکته نیز اشاره کنیم که در موقع شروع جستجو برای یافتن منابع جدید غذا، زنبورها به طور تصادفی و بدون استفاده از تجربه قبلی این کار را انجام می‌دهند. بنابراین، رفتار زنبورها در واقع ترکیبی از دو روش جستجوی محلی و جستجوی سراسری است. به عبارت دقیق‌تر، وقتی که هر زنبور پس از برگشتن به کندو اطلاعات مربوط به منبع گرده خود را با سایر زنبورها به اشتراک می‌گذارد نوعی جستجوی محلی انجام می‌شود، در حالی که جستجوی تصادفی زنبورها در محیط بیرون کندو یک جستجوی سراسری است (بیات، ۱۳۹۳).

۲-۳-۲ الگوریتم‌های کلونی زنبور

در این بخش به ترتیب الگوریتم زنبور، کلونی زنبور مصنوعی و بهینه‌سازی کلونی زنبور را معرفی خواهیم کرد.

۱-۲-۳-۲ الگوریتم زنبور

بدون از دست رفتن کلیت کار در ادامه فرض خواهد شد که مسئله بهینه‌سازی مسئله حداقل‌سازی است. همانطور که قبلاً ذکر شد، الگوریتم زنبور عسل از راه کار زنبور عسل در یافتن منابع غذایی الهام گرفته شده است که هدف آن جستجوی بهترین راه حل برای یک مسئله بهینه‌سازی است. هر نقطه از فضای جستجو (هر راه حل بالقوه) یک منبع غذایی در نظر گرفته می‌شود. "زنبورهای پیشاهنگ" بطور تصادفی در این فضا پخش می‌شوند (معادل راه حل‌هایی که بطور اتفاقی تولید می‌شوند) و از طریق تابع برازنده‌گی، کیفیت مکان‌های بازدید شده را گزارش می‌دهند (معادل ارزیابی راه حل‌ها). راه حل‌های نمونه طبقه‌بندی شده و سایر زنبورها استخدام شده تا به جستجو در مکان‌هایی

با بالاترین رتبه بپردازند (یعنی راه حل های بالقوه نزدیک به بهترین راه حل های تولید شده مورد بررسی قرار می گیرند). همسایگی یک راه حل محدوده گل^۱ نامیده می شود. الگوریتم زنبور راه حل هایی با بیشتر میزان تضمین را تعیین می کند و بصورت گزینشی همسایگی های آنها را برای کمینه جهانی تابع هدف کاوش می نماید. در ادامه این بخش، این روش به همراه جزئیات توضیح داده می شود. فلوچارت الگوریم و پارامترهای اصلی در شکل ۱۱-۲ و ۱۲-۲ نمایش داده شده اند [۲۴].

<i>ns</i>	number of scout bees
<i>ne</i>	number of elite sites
<i>nb</i>	number of best sites
<i>nre</i>	recruited bees for elite sites
<i>nrb</i>	recruited bees for remaining best sites
<i>ngh</i>	initial size of neighbourhood
<i>stlim</i>	limit of stagnation cycles for site abandonment

شکل ۱۱-۲ پارامترهای الگوریتم زنبور

۱-۲-۳-۲ طرح ارائه شده

فضای راه حل های امکان پذیر بصورت $\{x \in R^n ; \min_i < x_i < \max_i ; i = 1, \dots, n\}$ و تابع $f(x)$ داده شده و هر راه حل کاندید بصورت یک بردار n بعدی با متغیرهای برازنده‌گی $x = \{x_1, \dots, x_n\}$ نشان داده می شود [۲۴].

۲-۳-۲-۱ مقداردهی اولیه

جمعیتی ثابت از ns زنبور پیشاهنگ در نظر گرفته می شود و بطور تصادفی و با احتمال یکنواخت در فضای راه حل پراکنده می شوند. هر زنبور پیشاهنگ محل بازدید شده (راه حل) را از طریق تابع برازنده‌گی، ارزیابی می نماید. سپس الگوریتم وارد حلقه اصلی شده که مرکب از چهار مرحله است. تکرار الگوریتم تا زمانیکه معیار توقف دیده شود، ادامه می یابد [۲۴].

¹ Flower Patch

۴-۲-۳-۲ رقص جنبشی

ns محل مشاهده شده، براساس اطلاعات تابع برازنده‌گی که توسط زنبورهای پیشاہنگ جمع‌آوری شده، رتبه‌بندی و nb موقعیت با حداکثر برازنده‌گی (حداقل تابع هزینه) برای جستجوی محلی انتخاب می‌شوند. جستجوی محلی توسط سایر زنبورها صورت می‌پذیرد که به همسایگی محل‌های انتخاب شده توسط زنبورهای پیشاہنگ‌ها هدایت می‌شوند. برای هر محل منتخب تعداد پیروهای اختصاص داده شده به شرح ادامه است:

هر زنبور پیشاہنگ که از nb محل برتر باز می‌گردد، رقص جنبشی انجام داده که هم لانه‌های خود را برای جستجوی محلی به پیروی وا می‌دارد. زنبورهای پیشاہنگ که نخستین ne محل نخبه (بهترین) را در میان nb محل منتخب بازدید کرده‌اند، nre پیرو را برای جستجوی همسایگی‌ها بکار می‌گیرند. پیشاہنگ‌هایی که محل‌های باقی‌مانده ($nb - ne$) را بازدید کرده‌اند، nrb زنبور پیرو ($nrb \leq nre$) را به منظور جستجوی همسایگی بکار می‌گیرند.

براساس رویه گفته شده، زنبورهای بیشتری برای جستجوی فضای راه حل در مجاورت ne راه حل با بالاترین برازنده‌گی اختصاص داده می‌شوند. بنابراین جستجوهای محلی بیشتر در مجاورت محل‌های نخبه انجام می‌شود که به عنوان تضمین شده‌ترین موقعیت‌های فضای جستجو مورد بررسی قرار می‌گیرند. این روند نفرگیری تفاضلی مبتنی بر برازنده‌گی کلید عملکرد الگوریتم زنبور است، چراکه وسعت جستجو را تعریف می‌نماید [۲۴].

۴-۱-۲-۳-۲ جستجوی محلی

برای هر nb محل منتخب، زنبورهای پیرو بطور تصادفی و با احتمال یکنواخت در همسایگی با بالاترین برازنده‌گی که پیشتر توسط زنبور پیشاہنگ مشخص شد، قرار می‌گیرند. این همسایگی (محدوده گل) بصورت یک جعبه n بعدی با صفحه‌های a_1, \dots, a_n تعریف می‌شود که زنبور پیشاہنگ مرکز آن است. برای هر محدوده گل برازنده‌گی محل‌های بازدید شده توسط زنبورهای پیرو مورد ارزیابی قرار می‌گیرد. اگر یکی از زنبورهای پیرو در مکانی با برازنده‌گی بالاتر از زنبور پیشاہنگ قرار بگیرد، آن زنبور پیرو بعنوان زنبور پیشاہنگ جدید انتخاب می‌گردد. در نهایت اصلاح‌ترین زنبور

از هر محدوده نگاه داشته می‌شود. همچنین اصلاح ترین راه حل بازدید شده تاکنون به عنوان نماینده تمام محدوده گل‌ها شناخته می‌شود. این زبور زمانی که به کندو باز می‌گردد همچنان به رقص ادامه می‌دهد. در طبیعت ساز و کار بازخورد متفاوت است، از آنجایی که همه زبورهایی که در فرآیند جستجوی غذا شرکت می‌کنند به رقص جنبشی می‌پردازند [۲۴].

۵-۱-۲-۳-۲ جستجوی سراسری

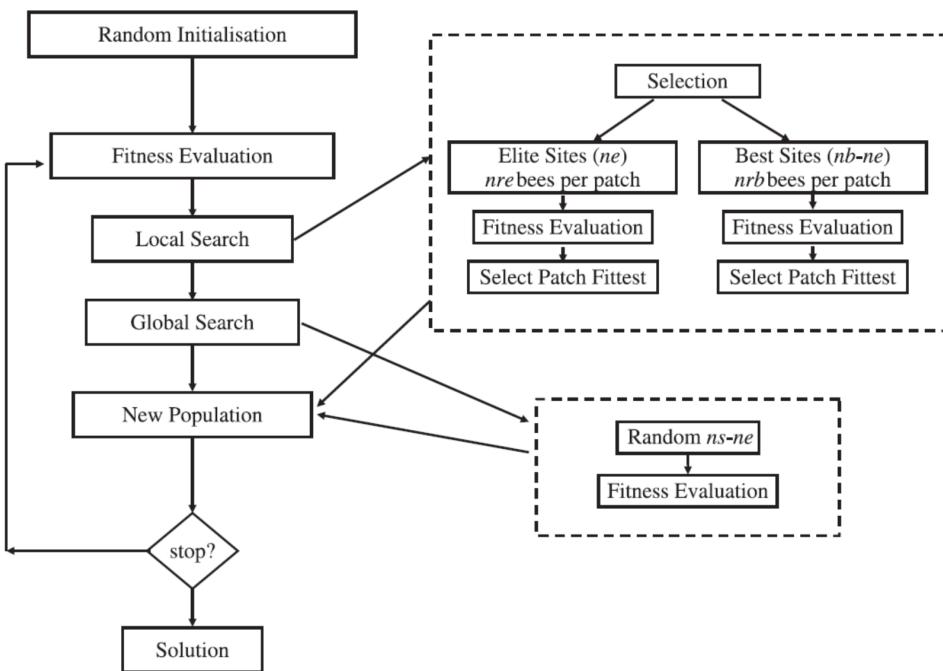
در مرحله جستجوی سراسری، $ns-nb$ زبور بطور تصادفی در فضای جستجوی برازنده‌گی قرار می‌گیرند، تا برای پیدا کردن محدوده‌های گل جدید کاوش کنند. کاوش اتفاقی تلاش برای جستجوی سراسری را در الگوریتم زبور نشان می‌دهد [۲۴].

۶-۱-۲-۳-۲ بروزرسانی جمعیت

پس از پایان هر تکرار، جمعیت جدید کلونی زبورها از دو گروه تشکیل می‌شود. گروه اول شامل nb زبور پیشه‌نگ که مرتبط به مرکز محدوده گل متناظر (بهترین راه حل محدوده) است و نتیجه جستجوی محلی را نشان می‌دهند. گروه دوم شامل $ns-nb$ زبور پیشه‌نگ است که با راه حل‌های اتفاقی مرتبط هستند و نمایانگر نتیجه جستجوی سراسری هستند [۲۴].

۷-۱-۲-۳-۲ معیار توقف

معیار توقف وابسته به حوزه مسئله است و می‌تواند موقعیت راه حل برازنده‌گی بیشتر از یک آستانه از پیش تعیین شده و هم تکمیل تعداد تکرارهای تکاملی باشد که از قبل تعیین شده‌اند [۲۴].



شکل ۱۲-۲ الگوریتم BA

۸-۲-۳-۲ الگوریتم زنبور توسعه یافته

فرآیند نشان داده شده در شکل ۱۲-۲، الگوریتم زنبور را در فرمول‌بندی پایه نشان می‌دهد. دو فرآیند دیگر نیز معرفی شده است ([۲۵] و [۲۶]) تا میزان صحت جستجوی الگوریتم را افزایش داده و از محاسبات اضافی اجتناب گردد.

- کوچک شدن همسایگی

براساس نخستین فرآیند، اندازه $\{a_1, \dots, a_n\} = a$ در محدوده گل در ابتدا یک عدد بزرگ در نظر گرفته می‌شود. برای هر متغیر a_i رابطه ۴-۲ صدق می‌کند:

$$a_i(t) = ngh(t) \times (\max_i - \min_i) \quad \text{معادله ۴-۲}$$

$$ngh(0) = 1.0$$

که در آن t مشخص کننده تامین تکرار از حلقه اصلی الگوریتم زنبور است. ابعاد محدوده گل تا زمانی که فرآیند جستجوی محلی نقاطی با برآزندگی بهتر را پیدا می‌کند، بدون تغییر نگه داشته می‌شود. اگر جستجوی محلی نتواند برآزندگی‌های بهتری را نتیجه دهد، ابعاد a کاهش می‌یابد. بروزرسانی ابعاد همسایگی از یک فرمول ابتکاری پیروی می‌نماید:

$$ngh(t+1) = 0.8 \times ngh(t)$$

بنابراین با پیروی از این سازوکار، جستجوی محلی ابتدائاً بر روی یک همسایگی بزرگ تعریف می‌شود که ویژگی اکتشافی وسیعی دارد. با پیشرفت الگوریتم، جستجو با جزئیات بیشتری مورد نیاز است تا مقدار بهینه محلی کنونی تصحیح گردد. بنابراین جستجو بسیار اکتشافی خواهد بود و نواحی اطراف مقدار بهینه بیشتر مورد اکتشاف قرار می‌گیرد.

- ترک محل^۱

دومین فرآیند زمانی اعمال می‌شود که روش نخست در کوچک کردن همسایگی، کمکی در بدست آوردن شایستگی بهتر نکند. هر زمان که جستجوی محلی نتواند راه حلی با تابع هزینه کمتر پیدا کند، ابعاد جستجوی محلی کاهش می‌یابد. پس از یک تعداد از پیش مشخص شده از تکرارهای راکد متوالی یعنی پارامتر $stlim$ ، فرآیند جستجوی محلی فرض را بر این می‌گذارد که قلئه برازنده‌گی محلی رسیده است و امکان بهبود بیشتر وجود ندارد. در نتیجه، جستجوی محدوده گل به اتمام می‌رسد و یک راه حل اتفاقی جدید تولید می‌شود. اگر محل رها شده مربوط به بهترین مقدار برازنده‌گی که تاکنون بدست آمده باشد، موقعیت آن ضبط می‌گردد. اگر در طول باقی جستجو، سایر محدوده گل‌ها مقدار بهتری برای تابع برازنده‌گی پیدا نکردند، موقعیت ذخیره شده به عنوان نتیجه نهایی در نظر گرفته می‌شود [۲۴].

۲-۲-۳-۲ الگوریتم کلونی زنبور مصنوعی (ABC)

در این بخش در مورد الگوریتم کلونی زنبور مصنوعی یا به اختصار ABC خواهیم پرداخت. از این الگوریتم بیشتر برای حل مسائل بهینه‌سازی پیوسته نظیر پیدا کردن نقطه اکسترمم یک تابع چند متغیره استفاده می‌شود.

در الگوریتم ABC نیز زنبورهای مصنوعی، که هر یک از آن‌ها در واقع یک عامل محاسباتی ساده است، با تقلید رفتار زنبورهای عسل در طبیعت به حل مساله بهینه‌سازی مورد نظر می‌پردازند. در الگوریتم ABC زنبورهای مصنوعی موجود در کلونی به سه دسته تقسیم و از آن‌ها می‌آورند؛

¹ Site Abandonment

زنبورهای تماشاگر که درون کندو بوده و به رقص زنبورهای مشغول نگاه میکنند تا با استفاده از آن-ها از موقعیت منبع غذا آگاه شوند و زنبورهای پیشاہنگ که به طور تصادفی در محیط پیرامون کندو به دنبال غذا می‌گردند. زنبورهای پیشاہنگ و تماشاگر را گاهی زنبورهای بیکار نیز می‌نامند. دلیل این نامگذاری آن است که در موقع اجرای برنامه فقط زنبورهای مشغول به کار به جستجو برای یافتن جواب‌های بهینه می‌پردازند. همانطور که در ادامه نیز خواهیم دید، زنبورهای تماشاگر وجود خارجی ندارند و زنبورهای پیشاہنگ نیز فقط هنگامی که نیاز به انتخاب تعدادی نقطه به طور تصادفی از دامنه مسئله داشته باشیم ظاهر می‌شوند. تنها دلیل استفاده از این زنبورها کمک به توضیح نحوه عملکرد الگوریتم ABC و ایجاد امکانی برای مقایسه آن با عملکرد زنبورهای عسل در طبیعت است.

الگوریتم ABC با ایجاد جمعیت اولیه‌ای از بردارهای تصادفی شروع به کار می‌کند. گاهی در این مرحله، زنبورهایی که مسئول یافتن تعدادی نقطه به طور تصادفی از دامنه مسئله هستند را زنبورهای پیشاہنگ می‌نامند. نحوه ادامه کار بدین صورت است که در هر تکرار از الگوریتم، زنبورهای مصنوعی (که این بار زنبورهای مشغول به کار نمیده می‌شوند) با انجام جستجوی تصادفی در اطراف جواب‌های به دست آمده در تکرار قبل به یافتن جواب‌های جدید می‌پردازند. بدیهیست که این جواب‌های جدید لزوماً همگی بهتر از جواب‌های بدست آمده در تکرار قبل نخواهند بود. پس از آنکه هر یک از زنبورهای مصنوعی مشغول به کار یک جواب جدید پیدا کردد، همه آنها دوباره به کندو باز گردانده می‌شوند و برای انتخاب مسیر حرکتشان در تکرار بعدی تصمیم‌گیری می‌کنند. این همان مرحله زنبورهای تماشاگر است. همانطور که دیده می‌شود، در الگوریتم ABC واقعاً هیچ زنبور مصنوعی تماشاگری در کار نیست؛ بلکه همان زنبورهایی که هر کدامشان یک جواب جدید از دامنه مسئله پیدا کرده‌اند، جواب‌هایشان را در این مرحله با یکدیگر به اشتراک می‌گذارند. برای این منظور ابتدا هر زنbor کیفیت موقعیت جدیدش را با موقعیت قبلی مقایسه کرده و اگر موقعیت جدید بهتر از موقعیت قبلی باشد، آن را جایگزین موقعیت قبلی می‌کند و در غیر این صورت آن را کنار گذاشته و همان موقعیت قبلی را در حافظه‌اش نگه می‌دارد. سپس میزان بهینگی (یا در واقع تناسب^۱) جواب-

¹ Fitness

های به دست آمده توسط زنبورها محاسبه شده و سپس هر یک از زنبورها با استفاده از چرخ رولت یکی از جواب‌های به دست آمده توسط زنبورهای تکرار قبلی را به عنوان مسیر جستجو در تکرار بعدی انتخاب می‌کند. بدین ترتیب بدینهیست که نخست نواحی اطراف جواب‌های بهینه‌تر توسط تعداد زنبورهای بیشتری در تکرار بعدی مورد جستجو قرار می‌گیرند و دوم امکان انجام جستجوی جدید در اطراف جواب‌های غیر بهینه به دست آمده در تکرار قبلی نیز فراهم می‌گردد. برآیند عملیات فوق جستجوی موثر تمام دامنه مسئله است. مراحل فوق تا زمان تامین شرط مورد نیاز برای خاتمه اجرای برنامه (یعنی مثلاً اجرای تعداد از پیش تعیین شده‌ای از تکرارها) انجام می‌گیرد. در الگوریتم ABC برای افزایش کارایی فرآیند فوق، در صورتی که پس از انجام تعداد از پیش تعیین شده‌ای از تکرارها هیچ جواب بهتری پیدا نشود، تعدادی از زنبورها (یا همگی آنها) راه حل‌های خود را کنار گذاشته و با تبدیل شدن به زنبورهای پیشاهنگ مجدداً به جستجوی تصادفی در دامنه مسئله می‌پردازند. انجام این کار می‌تواند به نحو قابل ملاحظه‌ای شанс یافتن پاسخ بهینه سراسری را افزایش دهد. بهترین پاسخ به دست آمده در طول تمام تکرارها را نیز می‌توان برابر با پاسخ مسئله بهینه‌سازی در نظر گرفت. با توجه به توضیحات فوق واضح است که الگوریتم ABC از برخی جهات شبیه به کلونی زنبورهای واقعی و از برخی جهات متفاوت با آن عمل می‌کند.

در ادامه، مراحل ذکر شده در قبل را با زبان ریاضی فرمول بندی می‌کنیم. برای این منظور مسئله کمینه‌سازی پیوسته:

$$\begin{cases} \min f(x) \\ s.t. X \in \Omega \end{cases} \quad \text{معادله ۶-۲}$$

را که در آن $[x_1, x_2, \dots, x_m]$ را یک تابع غیرخطی m متغیره و

$$\Omega \triangleq \left\{ x \in R^m \mid -\infty < l_k \leq x_k \leq u_k < \infty, k = 1, \dots, m \right\} \quad \text{معادله ۷-۲}$$

است را در نظر بگیرید. در موقع شروع به کار الگوریتم، موقعیت‌های تصادفی اولیه مناسبی را از دامنه مسئله به زنبورهای مصنوعی نسبت می‌دهیم. برای این منظور فرض کنید موقعیت زنبور n به صورت معادله ۸-۲ است.

$$X_n = [x_{n1}, x_{n2}, \dots, x_{nm}] \quad \text{معادله ۸-۲}$$

که m و SN به ترتیب بیانگر تعداد متغیرهای مسئله بهینه‌سازی و تعداد زنبورهای مصنوعی بوده و $x_{n1}, x_{n2}, \dots, x_{nm}$ نیز اعداد تصادفی انتخاب شده از دامنه مسئله هستند. سپس در هر تکرار از الگوریتم، زنبور کارگر i با جستجو در اطراف X_i (یعنی موقعیت ذخیره شده در حافظه اش از تکرار قبل) موقعیت احتمالاً بهینه‌تر جدید $V_n = [v_{n1}, v_{n2}, \dots, v_{nm}]$ را در همسایگی اش می‌یابد که مؤلفه‌های بردارهای X_i و V_n به صورت معادله ۹-۲ با یکدیگر مربوط‌اند:

$$v_{ni} = x_{ni} + \varphi_{ni} (x_{ni} - x_{ki}) , \quad i=1, \dots, m \quad \text{معادله ۹-۲}$$

در معادله (۹-۲)، $k \neq n$ یک عدد تصادفی صحیح در بازه $[1, SN]$ و φ_{ni} یک عدد تصادفی حقیقی با توزیع یکنواخت در بازه $[0, 1]$ است. در صورتی که موقعیت همسایه جدید بهتر از موقعیت قبلی باشد، یعنی داشته باشیم $(v_n f(x_n) < f(x_k))$ ، موقعیت جدید جایگزین موقعیت قبلی می‌شود ($V_n = X_i$) و در غیراینصورت موقعیت قبلی حفظ می‌گردد. توضیح چند نکته در ارتباط با معادله (۹-۲) ضروری است. اول اینکه φ_{ni} به جای بازه $[0, 1]$ می‌تواند از بازه $[-a, a]$ انتخاب شود که در اینصورت با استفاده از پارامتر a می‌توان شعاع همسایگی مورد جستجو را کنترل کرد (توجه داشته باشید که در موقع محاسبه هر یک از مقادیر v_i باید از یک مقدار تصادفی جدید برای متغیر k استفاده شود). همچنین باید مواظب بود که V_n به خارج از محدوده تعريف شده توسط Ω در معادله (۹-۲) وارد نشود. توجه کنید که در معادله (۹-۲) در واقع یکی از ابعاد هر یک از بردارهای جواب را انتخاب کرده و با توجه به مقدار پارامتر φ_{ni} به طور تصادفی به سمت آن یا در خلاف جهت آن حرکت می‌کنیم.

پس از محاسبه موقعیت‌های جدید از معادله (۹-۲) و انجام جایگزینی‌های لازم، مقدار تناسب زنبورها را از معادله ۱۰-۲ محاسبه می‌کنیم:

$$fit(X_n) = \begin{cases} \frac{1}{1+f(X_n)} & f(X_n) \geq 0 \\ 1+abs(f(X_n)) & f(X_n) < 0 \end{cases} \quad \text{معادله ۱۰-۲}$$

که $fit(X_n)$ بیانگر تناسب منبعت از موقعیت X_n واقع شده و $f(X_n)$ مقدارتابع هزینه در این نقطه است. با توجه به تعريف ارائه شده در قبل برای تابع تناسب بدیهی است که به ازای

(X_n)های منفی، مقدار تابع تناسب بزرگتر از یک و در غیر این صورت کوچکتر از یک خواهد بود. همچنین واضح است که در صورت استفاده از تعریف داده شده در معادله (۱۰-۲)، زنبورهای تماشاگر زنبورهای کارگر تکرار بعد (یا در واقع موقعیت‌های آنها) را با استفاده از قوانین احتمالاتی و بر اساس کیفیت منابع غذایی یافت شده توسط آنها انتخاب می‌کنند. برای این منظور احتمال انتخاب منبع غذایی واقع در موقعیت n یعنی P_n از معادله ۱۱-۲ محاسبه می‌شود:

$$P_n = \frac{fit(X_n)}{\sum_{k=1}^{SN} fit(X_k)} \quad \text{معادله ۱۱-۲}$$

زنبورهای کارگر تکرار بعد را می‌توان با استفاده از چرخ رولت انتخاب کرد. برای این منظور کافیست قطاع هر زنبور را بر روی چرخ رولت متناسب با احتمال به دست آمده از معادله (۱۱-۲) قرار دهیم. از آنجایی که در هر تکرار از الگوریتم دقیقاً به SN زنبور کارگر احتیاج داریم، لذا باید چرخ رولت را SN بار بچرخانیم و هر بار زنبور قرار گرفته در جلوی اشاره‌گر را به عنوان زنبور کارگر تکرار بعد انتخاب نماییم. بدیهیست که با این کار ممکن است برخی از زنبورها بیش از یک بار انتخاب شوند در حالی که برخی دیگر شанс انتخاب شدن را حتی برای یک بار هم پیدا نکرده‌اند.

روند فوق (یعنی انتخاب موقعیت همسایه، جایگزین کردن موقعیت‌های بهتر به جای موقعیت‌های بدتر قبلی، محاسبه تناسب موقعیت‌ها و انتخاب زنبورهای تکرار بعدی با استفاده از قوانین احتمالاتی) تا زمان تامین شرط یا شروط مورد نیاز برای پایان اجرای برنامه ادامه می‌یابد. نتیجه طبیعی استفاده از فرآیند فوق برقراری نوعی فیدبک مثبت و در نتیجه جذب تعداد بیشتری از زنبورها به سمت منابع غذایی غنی‌تر (یا به طور معادل نقاط بهینه‌تر) است. تنها نکته باقی مانده در ارتباط با این الگوریتم مربوط به نحوه تعریف و عملکرد زنبورهای پیشاہنگ است که در ادامه در مورد آن صحبت خواهیم کرد.

برای افزایش شانس یافتن پاسخ‌های بهتر و امکان خروج از نقاط بهینه محلی در نسخه اولیه الگوریتم ABC پیشنهاد شده بود که یکی از زنبورهای مورد استفاده در الگوریتم به نام زنبور پیشاہنگ نامگذاری شده و به کار گرفته شود [۲۷]. تعریف اولیه مورد استفاده برای زنبورهای پیشاہنگ چند

سال بعد در [۲۸] تا حدی تغییر داده شد که در ادامه فقط در مورد تعریف اخیر صحبت خواهد شد. در واقع مهم‌ترین تغییر رخ داده در تعریف زنبور پیشاہنگ در [۲۸] نسبت به [۲۷] مربوط به تعداد آن‌ها است که در تعریف اخیر این عدد به «یک» محدود نشده است. به عبارت دقیق‌تر، طبق مطالب گفته شده در [۲۸] از همان آغاز اجرای برنامه یکی یا چند تا از زنبورها را علامتگذاری می‌کنیم و اگر این زنبورها پس از چند تکرار متوالی معین (که مقدار آن از قبل مشخص شده) موفق به یافتن جوابی بهتر از آنچه از قبیل در ذهنشان دارند نشوند، آن‌ها را به زنبورهای پیشاہنگ تبدیل می‌کنیم که همین امر باعث می‌شود این زنبورها با رها کردن موقعیت ثبت شده در حافظه‌شان به جستجوی تصادفی در دامنه مسئله بپردازنند. معمولاً در مقالات و مراجع مرتبط با الگوریتم ABC، تعداد تکرارهای متوالی ای که در صورت نیافتن پاسخ بهتر توسط یک زنبور معین (که از پیش مشخص شده) آن را به زنبور پیشاہنگ تبدیل می‌کنیم با پارامتر limit نشان داده می‌شود. به ادعای [۲۸] انتخاب مناسب پارامتر limit تاثیر به سزایی در کارامدی الگوریتم ABC دارد. بدیهی است که با انجام این کار آن دسته از منابع غذایی که از همان آغاز ضعیف هستند یا پس از استفاده ضعیف می‌شوند با برقراری نوعی فیدبک منفی کنار گذاشته خواهند شد. همچنین توجه داشته باشید که معیار کنارگذاری معمولاً باید به گونه‌ای انتخاب شده باشد که زنبورهای پیشاہنگ پیش از پایان اجرای برنامه بتوانند به تعداد دفعات به اندازه کافی دوباره وارد دامنه مسئله شوند. در الگوریتم ABC بهترین موقعیت یافت شده توسط تمام زنبورها از آغاز اجرای برنامه تا پایان آخرین تکرار به عنوان پاسخ نهایی مسئله معرفی می‌شود. در پایان بد نیست به این نکته نیز اشاره کنیم که در حال حاضر هیچ روش نظاممندی برای تعیین مقادیر مناسب پارامترهای مورد استفاده در الگوریتم ABC وجود ندارد. با این حال چون تعداد پارامترهای مورد استفاده در این الگوریتم کم است، معمولاً مقادیر مناسب آن‌ها را می‌توان با یک آزمون و خطای ساده تعیین کرد (بیات، ۱۳۹۳).

۲-۳-۳ الگوریتم بهینه‌سازی کلونی زنبور (BCO)

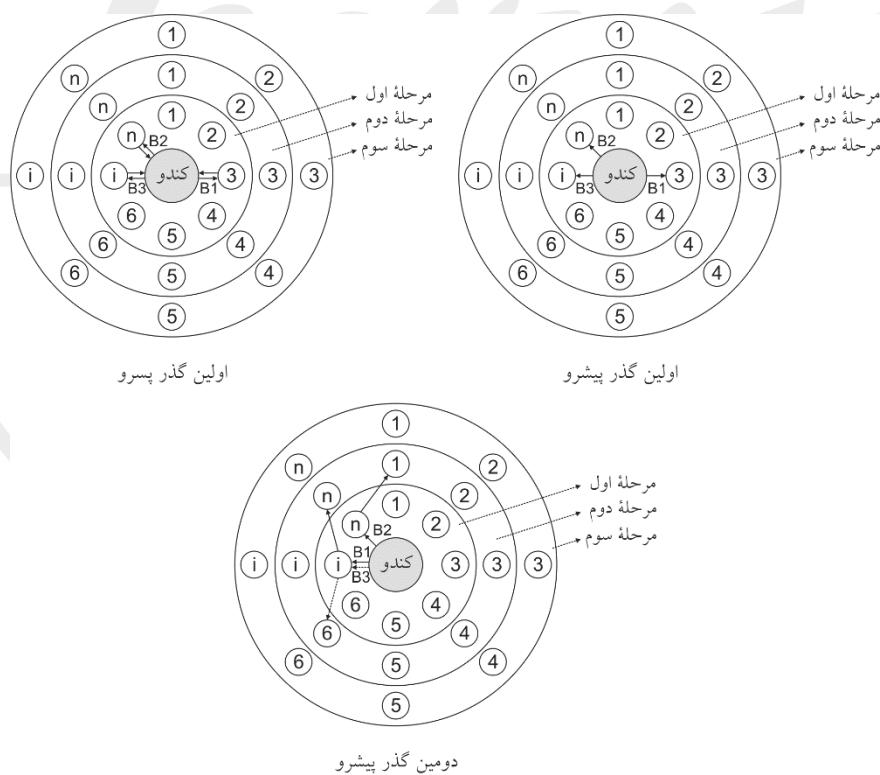
یکی از الگوریتم‌های نسبتاً کلاسیک و جا افتاده در حوزه کلونی زنبور الگوریتمی است موسوم به الگوریتم بهینه‌سازی کلونی زنبور یا به اختصار BCO. همانطور که در ادامه نیز خواهیم دید، با توجه

به ماهیت این الگوریتم از آن بیشتر برای حل مسائل بهینه‌سازی ترکیبی استفاده می‌شود، هرچند که به راحتی می‌توان از آن برای حل مسائل بهینه‌سازی پیوسته نیز استفاده کرد. به عنوان مثال از الگوریتم TSP نیز می‌توان برای یافتن کوتاهترین مسیر موجود بر روی یک گراف یا حل یک مسئله BCO استفاده نمود. در ادامه ابتدا کمی در مورد زنبورهای مصنوعی مورد استفاده در الگوریتم BCO صحبت شده و سپس به معرفی الگوریتم BCO پرداخته می‌شود.

الگوریتم BCO الگوریتمی مبتنی بر جمعیتی از زنبورهای مصنوعی است که همگی به دنبال جواب بهینه می‌گردند. بنابراین هر زنبور مصنوعی در واقع یک عامل محاسباتی ساده است که در همکاری با سایر زنبورهای مصنوعی یک مسئله پیچیده بهینه‌سازی (احتمالاً از نوع ترکیبی) را حل می‌کند. توجه به این نکته ضروری است که در الگوریتم BCO هر زنبور مصنوعی دقیقاً یک جواب برای مسئله بهینه‌سازی پیدا خواهد کرد که در نهایت بهترین آن‌ها می‌تواند به عنوان جواب نهایی مسئله بهینه‌سازی در نظر گرفته شود.

هر مرحله از الگوریتم BCO از دو فاز مجزا تشکیل شده است: گذر پیشرو و گذر پسرو. در هر گذر پیشرو، ابتدا هر یک از زنبورهای مصنوعی تمام مسیرهای پیش رو را وارسی کرده و سپس با انتخاب یکی از آن‌ها تعداد از پیش تعیین شده‌ای از حرکت‌ها را انجام می‌دهد. مثلاً در مسئله TSP هر زنبور با انتخاب تعدادی از گرهایی که قبلاً انتخاب نکرده می‌تواند یک مسیر برای ادامه حرکت خود ایجاد نماید. این حرکت‌ها یا مستقیماً منجر به یک پاسخ بهینه جدید می‌شوند و یا یک حل نیمه کاره موجود را که در نهایت منجر به یک حل جدید خواهد شد، بهبود می‌بخشند. به عبارت دقیق‌تر، در الگوریتم BCO زنبورهای مصنوعی مرحله به مرحله مسیر خود را تکمیل می‌کنند و پس از اضافه کردن هر تکه جدید به مسیر خود بلافصله نتایج را با هم مقایسه می‌نمایند. بدین منظور برای ایجاد یک حل نیمه کاره در گذر پیشرو، زنبورها دوباره به کندو باز می‌گردند و فاز دوم عملیات، که همان مرحله گذر پسرو است، آغاز می‌شود. در گذر پسرو زنبورهای مصنوعی اطلاعات مربوط به حل‌هایشان را با یکدیگر به اشتراک می‌گذارند.

در طبیعت، زنبورهای عسل با رقصیدن خود سایر زنبورها را در مورد موقعیت و کیفیت منبعی که یافته‌اند آگاه می‌سازند. برای شبیه‌سازی نرم‌افزاری این رفتار، در الگوریتم BCO تابع هزینه متناظر با هر یک از حل‌های نیمه‌کاره را محاسبه می‌کنیم (به عنوان مثال، در مسئله TSP مقدار این تابع برابر با طول مسیر پیموده شده است). سپس در مسیر برگشت، هر زنبور براساس مقادیر به دست آمده برای تابع هزینه توسط زنبورهای مختلف با یک احتمال معین تصمیم می‌گیرد که آیا از حل نیمه‌کارهای که تا این لحظه ایجاد کرده دست بکشد و به یک زنبور پیرو تبدیل شود یا قبل از بازگشتن و ادامه حل قبلی خود سایر زنبورها را نیز به استفاده از حل نیمه‌کارهای که تا این لحظه ایجاد کرده ترغیب کند. در هر صورت هر زنبور پیرو به تصادف یکی از زنبورهای رقاص را برای دنبال کردن انتخاب خواهد کرد (چنین زنبور رقاصی را زنبور پیشرو می‌نامیم). با توجه به بحث فوق هر چقدر که حل پیدا شده توسط یک زنبور بهتر باشد، احتمال دنبال کردن آن توسط سایر زنبورهای پیرو بیشتر خواهد بود.



شکل ۱۳-۲ حل مسئله TSP با استفاده از الگوریتم BCO

شکل ۱۳-۲ به ترتیب اولین گذر پیشرو و پسرو و دومین گذر پیشروی زنبورها را در الگوریتم BCO در موقع حل یک مسئله TSP با n گره نشان می‌دهند (در این شکل تعداد زنبورهای مصنوعی برابر با ۳ و تعداد گرههای اضافه شده در هر مرحله به مجموعه گرههای مرحله قبل برابر با ۱ در نظر گرفته شده است). در دومین گذر پیشرو، زنبورها مسیرهای ناقص قبلی را با انتخاب تعداد معینی از گره‌های مناسب (که مثلاً در مسئله TSP این گرههای مناسب برابر با مجموعه گرههای انتخاب نشده در مراحل قبل هستند) یک مرحله دیگر توسعه می‌دهند و پس از آن با انجام گذر پسرو به کندو باز می‌گردند. زنبورها دوباره در فرآیند تصمیم‌گیری شرکت می‌کنند، تصمیم خود را می‌گیرند و گذر پیشروی سوم را انجام می‌دهند. این رویه تا زمانی که حداقل یکی از زنبورها حل خود را تکمیل کند ادامه می‌یابد. توجه داشته باشید که مثلاً در مسئله یافتن کوتاهترین مسیر ممکن بین دو گره از یک گراف، زنبورها با انجام فرآیند فوق لزوماً همگی به طور همزمان موفق به تکمیل تور خود نخواهند شد، در حالی که مثلاً در مسئله TSP اگر یکی از زنبورها موفق به تکمیل تور خود شود، در آن صورت حتماً بقیه زنبورها نیز موفق به انجام این کار شده‌اند.

در هر صورت پس از آنکه حداقل یکی از زنبورها موفق به تکمیل حل خود شود، اصطلاحاً می‌گوییم که یک تکرار از الگوریتم BCO به پایان رسیده است (بنابراین، مثلاً در مسئله TSP هر تکرار دقیقاً شامل n مرحله و هر مرحله شامل یک گذر پیشرو و یک گذر پسرو است). فرآیند تکراری فوق تا آنجا ادامه می‌یابد که شرط لازم برای پایان اجرای الگوریتم فراهم شود؛ یعنی مثلاً تعداد از پیش تعیین شده‌ای از تکرارها و یا تعداد از پیش تعیین شده‌ای از تکرارهای متوالی که طی آن هیچ بهبودی در مقدار تابع هزینه مشاهده نمی‌شود، انجام گیرد.

با این توضیحات الگوریتم BCO را در ادامه بیان می‌نماییم. پارامترهایی که مقدار آن‌ها باید پیش از اجرای الگوریتم BCO به طور مناسب تنظیم گردند عبارتند از:

B : تعداد زنبورهای کندو،
 NC : تعداد حرکت‌های انجام شده در هر گذر پیشرو (مثلاً در مسئله TSP، NC برابر با تعداد گره‌هایی است که هر زنبور در هر گذر پیشرو آن‌ها را انتخاب خواهد کرد).

همچنین همیشه در موقع شروع به کار الگوریتم BCO فرض می‌کنیم که تمامی زنبورها در داخل کندو قرار دارند. با توجه به توضیحات فوق الگوریتم BCO به صورت زیر بیان می‌شود:

۱ - مقداردهی اولیه: به هر یک از زنبورها یک جواب تهی نسبت دهید.

۲ - برای هر یک از زنبورهای کندو، گذر پیشرو را به صورت زیر انجام دهید:

$$k = 1-2 \text{ قرار دهید}$$

۲-۱ تمام حرکت‌های ممکن برای ادامه مسیر فعلی ساخته شده توسط زنبور را شناسایی و امتیازدهی کنید (امتیازدهی به گونه‌ای انجام می‌شود که جواب‌های بهینه‌تر امتیاز بیشتری به دست آورند).

۳-۱ با توجه به امتیازدهی انجام شده در بند قبل، یکی از حرکت‌های ممکن برای ادامه مسیر را به طور تصادفی با استفاده از چرخ رولت انتخاب کنید (بدین ترتیب احتمال انتخاب هر یک از حرکت‌ها به کیفیت آن بستگی خواهد داشت).

۴-۱ قرار دهید $k = k + 1$ و اگر $k \leq NC$ آنگاه به مرحله (۲-۲) بروید.

۳ - تمام زنبورها را به کندو بازگردانید.

۴ - هر زنبوری با یک احتمال معین تصمیم می‌گیرد که آیا به تکمیل حل نیمه‌کاره خود ادامه داده و سایر زنبورها را نیز به پیروی از خود ترغیب کند و یا اینکه به یک زنبور پیرو تبدیل گردد (بدینهیست که در مسائل کمینه‌سازی، زنبورهایی که مقدار تابع هزینه‌شان عدد کوچکتری باشد از شанс بیشتری برای ادامه حل نیمه‌کاره قابلی برخوردار خواهند بود و بالعکس).

۵ - برای هر یک از زنبورهای پیرو یکی از زنبورهای پیشاهنگ را به طور تصادفی با استفاده از چرخ رولت انتخاب کنید.

۶ - در صورتی که شرط پایان اجرای الگوریتم هنوز فراهم نشده است به قدم دوم بروید.

۷ - بهترین نتیجه را اعلام کنید.

با توجه به توضیحات فوق واضح است که انتخاب $1 = NC$ به این معناست که در هر رهگذر پیشرو، هر یک از زنبورهای مصنوعی تنها یک گره به مجموع گرههای حل نیمه‌کاره خود اضافه

خواهد کرد. همچنین توجه شود که قدم سوم از الگوریتم فوق، یعنی بازگرداندن زنبورها به کندو، عملاً فاقد هر گونه بار محاسباتی است. در عمل برای پیاده‌سازی گام‌های چهارم و پنجم نیز کافیست که تمام زنبورها را دوباره با استفاده از چرخ رولت از میان زنبورهای مرحله قبل انتخاب نماییم. در گام ششم منظور از «شرط پایان اجرای الگوریتم» همان شروطی است که پیشتر نیز به آن اشاره کرده بودیم؛ یعنی مثلاً انجام تعداد از پیش تعیین شده‌ای از تکرارها یا انجام تعداد از پیش تعیین شده‌ای از تکرارها که طی آن هیچ بهبودی در مقدار تابع هزینه مشاهده نشود (بیات، ۱۳۹۳).

۳. پیاده‌سازی نرم‌افزاری و سخت‌افزاری الگوریتم زنبور

در این فصل در ابتدا به نحوه پیاده‌سازی نرم‌افزاری سه الگوریتم بسیار معروف غذایابی زنبورها یعنی الگوریتم زنبور^۱، کلونی زنبور مصنوعی^۲ و بهینه‌سازی کلونی زنبور^۳ خواهیم پرداخت. در ادامه در مورد الگوریتم پیشنهاد شده توضیح خواهیم داد و تغییراتی که برای پیاده‌سازی در سخت‌افزار بر روی آن انجام شده است. پس از آن به تشریح نحوه پیاده‌سازی آن بر روی سخت‌افزار خواهیم پرداخت.

۳-۱ پیاده‌سازی نرم‌افزاری

همواره بهتر است که قبل از پیاده‌سازی یک الگوریتم پیچیده بر روی سخت‌افزار، آن را بر روی نرم-افزار پیاده کرد، چراکه در نرم‌افزار با محدودیت‌های سخت‌افزاری همانند زمان‌بندی، مدیریت حافظه و محدودیت منابع روبه‌رو نبوده و می‌توان قبل از آنکه با مشکلات پیاده‌سازی در سطوح پایین سخت‌افزاری دست و پنجه نرم کرد، از عملکرد درست الگوریتم در نرم‌افزاری سطح بالا مطمئن شد. توضیح پیاده‌سازی نرم‌افزاری در زیرقسمت‌های بعد آمده است.

۳-۱-۱ کدگزاری برای مسئله فروشنده دوره‌گرد

همانطور که در شکل ۱-۳ دیده می‌شود، برای اینکه بتوان مسئله فروشنده دوره‌گرد را حل نمود به یک مدل کدگزاری نیاز است که بدین منظور هر یک از شهرهای مسئله را که شامل یک مختصات طول و عرض هست را به یک عدد یا به عبارتی یک نشانی اختصاص داده؛ بدین معنی که مختصات طول و عرض شهر شماره یکم را با شماره یک در نظر گرفته و مختصات طول و عرض شهر دوم را

¹ Bee Algorithm (BA)

² Artificial Bee Colony (ABC)

³ Bee Colony Optimization (BCO)

با شماره دو متناظر کرده و تا آخرین شهر که شهر n است (تعداد کل شهرها در یک مسئله فرضی است که در قسمت‌های سخت‌افزاری نیز ما آن را با عبارت مختصر CITIES به عنوان ورودی واحدها خواهیم شناخت). همچنین اگر این شماره‌ها را به دنبال هم بچینیم یک مسیر بدست آمده که مسیر حرکت فروشنده خواهد بود. به عنوان مثال همانند شکل ۱-۳ وقتی شهر شماره یکم اول صفت قرار می‌گیرد یعنی شروع مسیر از شهر اول بوده، سپس فروشنده به شهر دوم رفته تا به آخرین شهر در صفت برسد و سپس از شهر اول بازگشته تا مسیر خود را کامل کند.



شکل ۱-۳ نحوه کدگذاری مسائل برای حل مسئله فروشنده دوره‌گرد

۲-۱ مسیر نزدیکترین همسایگی

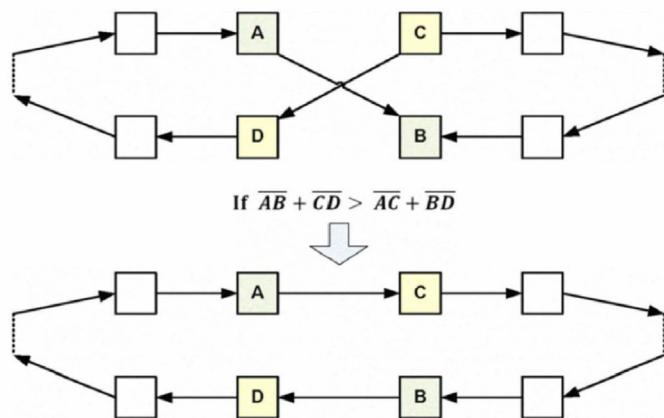
یکی از روش‌های مرسوم برای به دست آوردن یک حل تقریبی برای مسئله TSP روشی است موسوم به روش نزدیکترین همسایگی. در این روش ابتدا یکی از گره‌های مسئله به طور تصادفی انتخاب شده و سپس نزدیکترین گره به گره انتخاب شده به عنوان شهر مقصد بعدی در نظر گرفته می‌شود. با تکرار این فرآیند مسیری بدست می‌آید که آن را مسیر حاصل از روش نزدیکترین همسایگی می‌نامیم. بدیهیست که با شروع از شهرهای متفاوت و حرکت طبق فرآیند فوق ممکن است مسیری متفاوت از مسیر به دست آمده در جستجوهای قبل به دست آید (بیات، ۱۳۹۳).

۳-۱ روش‌های بهینه‌سازی محلی برای مسئله فروشنده دوره‌گرد

حل مسئله فروشنده دوره‌گرد با هر یک از الگوریتم‌های کلونی زنبور بدنی گونه است که پس از اختصاص دادن مقادیر اولیه مسیر به هر زنبور، این زنبورها حول مسیر ذخیره شده در حافظه خود که همان جواب مسئله است به دنبال جواب بهتری برای مسئله می‌گردند که این جستجو به صورت محلی انجام می‌شود. در ادامه دو روش بهینه‌سازی محلی آورده شده است.

۱-۳-۱-۳ ۲-opt روش

روش بهینه‌سازی 2-opt [۲۹] یک روش جستجوی محلی است که اولین بار برای حل مسئله فروشنده دوره‌گرد ارائه شد. ایده اصلی این روش آن است که قسمتی از مسیر را که از روی خود گذر می‌کند را در نظر گرفته و مسیر را به گونه‌ای تغییر دهیم که دیگر آن تقاطع وجود نداشته باشد.



شکل ۲-۳ ۲-Opt روش

در این روش، هر جفت از لبه‌ها در توالی مسیر با دیگری مورد مقایسه قرار می‌گیرند. در شکل ۳-۲، می‌توان دید که اگر مجموع فاصله حرکت از A به B و از C به D بیشتر از مجموع فاصله حرکت از A به C و از B به D باشد، دو یال اولیه را حذف می‌کنیم (A به B و C به D) و دو یال آخر را می‌افزاییم (A به C و B به D). این امر توالی جدید نشان داده شده در نیمه پایین شکل را ارائه می‌دهد [۳۰].

۱-۳-۲-۳ روش جهش حریص تکه مسیر^۱ (GSTM)

هدف از جهش در الگوریتم زنگنه، امکان اجتناب از بهینه‌های محلی و ساخت جمعیتی با مقادیر غیرتکراری است. بیشتر عملگرهای جهش، نمونه‌های جدید را بوسیله ایجاد اعوجاج در هر عضو بصورت تصادفی تولید می‌کنند. بنابراین در طول فرآیند بهینه‌سازی، احتمال برخورد با راه حل‌های محلی کاهش می‌یابد. ولی این عملگرها در طی پیاده‌سازی عملیات اعوجاج، هیچ تلاشی برای توسعه

^۱ Greedy Sub Tour Mutation

مسیر انجام نمی‌دهند. عملگرهایی که توسعه را انتخاب و اعمال می‌کنند و حداکثر بهره را تنها برای همان نمونه تولید می‌کنند، روش‌های حریص^۱ نامیده می‌شوند. همچنین الگوریتم‌های حریص از جمله تبادل جهشی حریص (GSM) [۳۱] نتایج بهتری را انتخاب می‌کنند، بنابراین سریعتر به راه حل نزدیک می‌شود. هرچند زمانی که روش‌های حریص به یک راه حل محلی می‌رسند، تغییر به راه حل جدید دیگر برایشان بسیار مشکل است. در نتیجه، روش‌های حریص راه حل‌هایی محلی تولید می‌کنند که از مقدار بهینه بسیار دور هستند.

عملگر GSTM [۳۲] که برای حل مسئله TSP و به وسیله الگوریتم ژنتیک توسعه یافته است، از هر دوی تکنیک‌های حریص و سنتی با کمک پارامترهای مختلف استفاده می‌کند. ساختار پارامتری GSTM مانع از گیر افتادن در راه حل‌های محلی همانند سایر عملگرهای جهش حریص می‌شود و دسترسی سریعتری را به راه حل‌ها فراهم می‌آورد. عملگر GSTM با استفاده از روش‌های حریص خیلی سریعتر به راه حل‌های محلی رسیده و یک اعواجاج تصادفی همانند عملگر سنتی جهش را بر روی این راه حل‌ها اعمال می‌کند و نمونه‌های جدید زود هنگامی تولید می‌کند و سپس دوباره با استفاده از روش‌های حریص آنها را امتحان می‌کند.

شش پارامتر در عملگر GSTM هستند که قالب کار و جهت عملگر جهش را مشخص می‌سازند. الگوریتم GSTM در قالب شبه کد^۲ در ادامه توضیح داده شده است.

انتخاب پارامترهای P_{RC} (احتمال اتصال مجدد)، P_{CP} (احتمال اختلال و اصلاح)، P_L (احتمال خطی بودن)، L_{MIN} (حداقل طول تکه مسیر)، L_{MAX} (حداکثر طول تکه مسیر)، NL_{MAX} (طول فهرست همسایگی)

گام ۱: تعریف تصادفی نقاط شروع (R_1) و پایان (R_2) تکه مسیر

گام ۲: انتخاب تصادفی مقدار Rnd بین بازه صفر و یک

اگر ($Rnd \leq P_{RC}$ پس {

¹ Greedy

² Pseudocode

کم کردن $[R_1 - R_2]$ تکه مسیر از مسیر $(T^{\#} \leftarrow T - [R_1 - R_2]; T^* \leftarrow [R_1 - R_2])$ تکه مسیر T^* را بگونه‌ای در مسیر $T^{\#}$ قرار دهید که موجب کمینه توسعه گردد

بر اساس عدد تصادفی انتخاب شده از بازه صفر تا یک،

اگر $(Rnd \leq P_{CP})$ سپس {

تکه مسیر $[R_1 - R_2]$ در مسیر T قرار می‌گیرد (

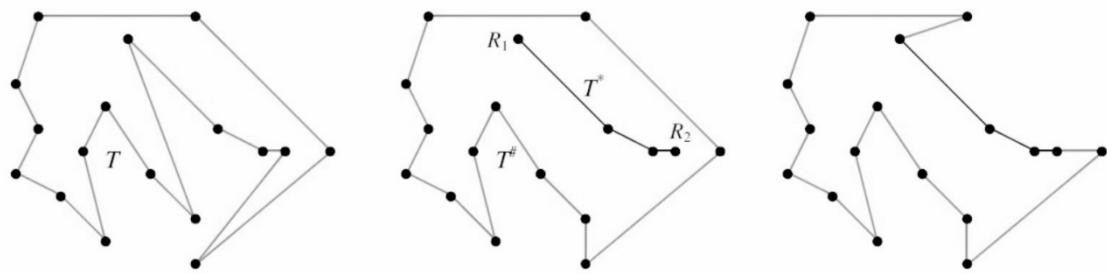
P_L با شروع از R_1 هر عنصر از تکه مسیر T^* به مسیر T با استفاده از چرخش یا ترکیب با احتمال اضافه می‌شود)

در غیراينصورت {

انتخاب تصادفی یکی از همسایه‌ها از فهرست همسایه‌ها برای نقاط R_1 و R_2 (NL_{R_1} و NL_{R_2})؛
برگرداندن نقاط NL_{R_1} یا NL_{R_2} به گونه‌ای که بهره حداکثر حاصل شود و این نقاط با نقاط R_1 یا R_2 همسایه باشند
اگر وارونگی رخ نداده باشد تکرار صورت می‌پذیرد {

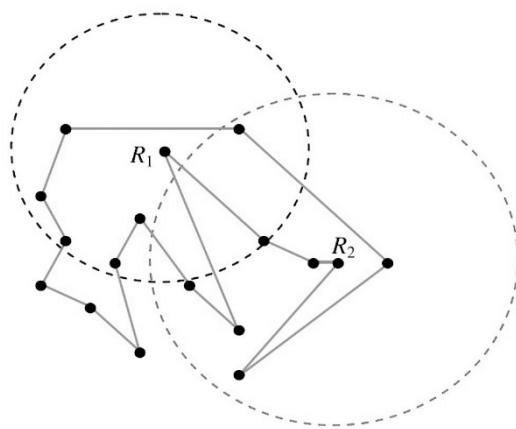
پارامترهای L_{MIN} و L_{MAX} برای تعریف طول تکه مسیری که لازم است در طول عملیات جهش انتخاب شود، استفاده می‌شوند. L_{MIN} و L_{MAX} بترتیب کوچکترین و بزرگترین طول تکه مسیری که می‌تواند انتخاب شود، را نشان می‌دهد. تکه مسیر $(T^* \leftarrow R_1 - R_2)$ که بر روی قسمتی از مسیر که عملیات جهش روی آن اجرا می‌شود، انتخاب می‌شود و باید در فاصله L_{MIN} و L_{MAX} انتخاب شود $L_{MIN} \leq |R_1 - R_2| \leq L_{MAX}$ باشد. حداکثر فاصله تکه مسیری که می‌تواند با L_{MAX} انتخاب شود بستگی به تعداد شهرها (n) در TSP دارد.

احتمال اتصال مجدد (P_{RC}) احتمال اجرای قسمتی است که تکه مسیر T^* انتخاب شده را از مسیر T حذف کرده و مجدداً آن را به مسیر $T^{\#}$ متصل کند، بگونه‌ای که حداقل توسعه رخ دهد (شکل ۳-۳).



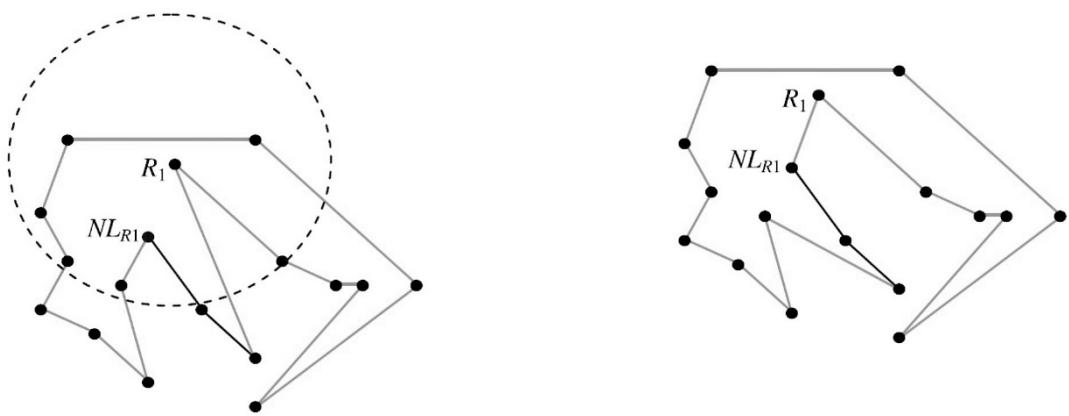
شکل ۳-۳ اتصال حریص

اگر عملیات اتصال مجدد که یک عملگر حریص است انجام نشود، به منظور ایجاد نمونه‌های جدید در جمعیت یا وارونگی محدود شده تکه مسیر با استفاده از جستجوی فهرست‌های فضای همسایه و یا اعوجاج تصادفی بر اساس احتمال اصلاح و اعوجاج (P_{CP}) انجام خواهد گرفت. در طی فرآیند چرخش تکه مسیر که از فهرست همسایه‌ها استفاده می‌کند، در مرحله نخست، هر یک از همسایه‌های (NL_{R_1} و NL_{R_2}) از فهرست همسایه‌ها انتخاب می‌شوند که متعلق به هر دو نقطه (R_1 و R_2) تکه مسیر است و همچنین ابعاد آن یعنی NL_{MAX} پیش از این تعریف شده است. سپس وارونگی (G_{R_1} و G_{R_2}) به این همسایگی‌ها اعمال می‌شوند، بطوریکه آن‌ها همسایه‌های نقاط تکه مسیر باشند و حداقل بیشتر در میان مسیرهای مختلف رخ داده باشد (شکل ۴-۳ تا ۶-۳). فهرست‌های همسایه مجزا برای هر شهر c_i ذخیره شده و شامل تعداد NL_{MAX} نزدیکترین همسایه به شهر c_i هستند.

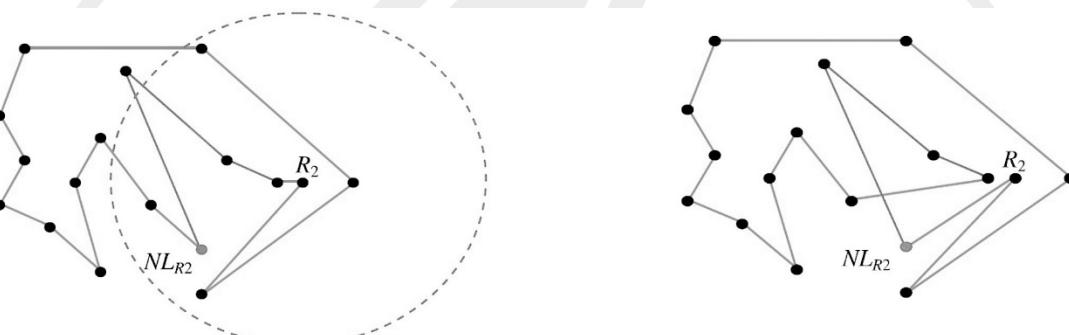


شکل ۴-۳ نزدیکترین همسایه شهرهای R_1 و R_2 که به صورت تصادفی از تکه مسیر انتخاب شدند.

$$(NL_{MAX} = 7)$$



شکل ۳-۵ چرخش شهر انتخاب شده تصادفی NL_{R_1} از فهرست همسایگی شهر R_1



شکل ۳-۶ چرخش شهر انتخاب شده تصادفی NL_{R_2} از فهرست همسایگی شهر R_2

لازم است که یک اعوجاج تصادفی بر روی هر عضو که در حال جهش هست به عنوان پیشگیری از احتمال گیر افتادن در راه حل های محلی، انجام شود. اعوجاجی که به هر عضو اعمال خواهد شد، بسته به P_{CP} و احتمال خطی بودن (P_L) دارد. در مسیر T ، تکه مسیر (T^*) بین نقاط R_1 و R_2 در خود اعوجاج پیدا می کند. P_L قاعدة عملیات اعوجاجی که اعمال می شود را تعریف می کند. هرچه P_L به یک نزدیکتر شود، احتمال ترکیب تصادفی هر شهر افزایش خواهد یافت همانند عملگر جهش درهم^۱ که زمانی که به صفر نزدیک می شود، احتمال انجام یک عملیات وارونگی معمولی افزایش می یابد، مانند عملگر روش اعوجاج. در ادامه این الگوریتم آورده شده است.

$$T^* \leftarrow T [R_1 - R_2]$$

$$w = R_1$$

¹ Scramble

تکرار عملیات‌های بعدی تا زمانیکه شهری در تکه مسیر^{*} T^* باقی نماند {

بر اساس عدد اتفاقی تولید شده Rnd در بازه صفر تا یک؛

اگر $\{ (Rnd \leq P_L)$

یک شهر تصادفی از تکه مسیر^{*} T^* به موقعیت w مسیر، اضافه گردد {

در غیراینصورت {

آخرین شهر تکه مسیر^{*} T^* به موقعیت w مسیر اضافه گردد {

شهرهای اضافه شده در مسیر T از تکه مسیر^{*} T کم گردد؛

[۳۲]. { w = w + 1

خلاصه روش [۳۲] که در اینجا آورده شد، بدین صورت است که یک عدد تصادفی انتخاب می‌کنیم، اگر این عدد از احتمال P_{RC} کوچکتر بود، تکه مسیری را که از مسیر اصلی جدا کردیم در جایی از مسیر اصلی قرار می‌دهیم که بهترین تابع شایستگی را نتیجه دهد که این قسمت زیرمجموعه جستجوهای جامع^۱ است. اگر بزرگتر از P_{RC} و کوچکتر از احتمال P_{CP} باشد، این مسیر در جای خود قرار می‌گیرد با این تفاوت که یا دوران یافته^۲ یا ترتیب شهرهای تکه مسیر بهم خورده است.^۳ اینکه کدام یک از این دو اتفاق بیفتند به انتخاب عدد تصادفی دوم و احتمال P_L وابسته است. اگر عدد تصادفی اول بزرگتر از P_{CP} باشد، نوع دیگری از دوران اعمال می‌شود که این نوع دوران را بر حسب تابع هزینه پیشنهادی تعریف کرده‌اند. مقادیر پارامترهای پیشنهادی در این الگوریتم بدین صورت است که در آن nodes تعداد گره‌های مسیر و Round به معنای گرد شدن عدد است:

$$L_{MAX} = Round(\sqrt{nodes}) , L_{MIN} = 2 , NL_{MAX} = 5 , P_L = 0.2 , P_{CP} = 0.8 , P_{RC} = 0.5$$

بدلیل آنکه در [۳۲] نحوه جستجوی جامع مشخص نشده است و احتمال P_{RC} هم بزرگ در نظر گرفته شده است (۰/۵)، در نتیجه جستجوی کامل یعنی قرار دادن تکه مسیر در بین هر دو گره مسیر

¹ Exhaustive

² Rolling

³ Mixing

و ارزیابی تابع شایستگی آن و انتخاب بهترین نتیجه از میان آن‌ها بطول خواهد انجامید. لذا در اینجا جستجوی جامع بدین صورت پیاده شده است که میانه هر یال مسیر اصلی که تکه مسیر از آن مسیر جدا شده است را محاسبه می‌کنیم $(T^{\#} \leftarrow T - [R_1 - R_2])$ ، سپس فاصله گره‌های R_1 و R_2 را از هر یک از این میانه‌ها حساب کرده و فاصله‌های متناظر با هر میانه را با هم جمع می‌کنیم. پس مجموعه‌ای نتیجه می‌شود که هر عضو آن مجموع فاصله دو گره R_1 و R_2 تا یکی از میانه‌هاست. حال کمترین فاصله که معادل نزدیکترین میانه است را انتخاب کرده و تکه مسیر را بین دو گره تشکیل دهنده این یال قرار می‌دهیم. جواب روش ارائه شده معادل جواب جستجوی جامع نبوده، چراکه لزوماً^۱ بهترین تابع هزینه را نتیجه نمی‌دهد، ولیکن از سرعت بیشتری برخوردار است.

۴-۱-۳ الگوریتم زنبور برای حل مسائل ترکیبی

این الگوریتم برای حل هر دو نوع مسائل پیوسته و ترکیبی مناسب است و تفاوت در نحوه جستجوی محلی است. در جدول ۴-۳ این الگوریتم مرحله‌بندی شده است:

جدول ۴-۳ الگوریتم زنبور برای حل مسائل ترکیبی

تعداد زنبورهای کلونی ^۱ (nc)، تعداد زنبورهای منتخب ^۲ (ns)، تعداد زنبورهای نخبه ^۳ (ne)، تعداد پیروهای زنبورهای منتخب ^۴ (nrs)، تعداد پیروهای زنبورهای نخبه ^۵ (nre)، تعداد تکرارها برای رها کردن یک بهینه محلی ^۶ (nl)، تعداد کل تکرارها (iterations)	تعیین پارامترها	.
اختصاص جواب اولیه به زنبورهای کلونی	مسیر اولیه	۱
محاسبه تابع شایستگی زنبورها	ارزیابی	۲
مرتب کردن زنبورها بر اساس تابع شایستگی آن‌ها	طبقه‌بندی	۳

¹ Colony Size

² Selected Bees

³ Elite Bees

⁴ Selected Bees Recruiters

⁵ Elite Bees Recruiters

⁶ Site Abandonment

۴	زنبورهای نخبه	ارزیابی زنبورهای پیرو انتخاب حریصانه ^۱ بین زنبورهای پیرو و زنبور نخبه متناظر آنها اختصاص جواب اولیه به زنبورهای بهبود نیافته در nl تکرار	nre جستجوی محلی در اطراف هر زنبور نخبه (اختصاص nre زنبور پیرو)
۵	زنبورهای منتخب	ارزیابی زنبورهای پیرو انتخاب حریصانه بین زنبورهای پیرو و زنبور منتخب متناظر آنها اختصاص جواب اولیه به زنبورهای بهبود نیافته در nl تکرار	nrs جستجوی محلی در اطراف هر زنبور منتخب (اختصاص nrs زنبور پیرو)
۶	ذخیره	ذخیره بهترین جواب	
۷	زنبورهای بیکار	اختصاص جواب اولیه به باقیمانده زنبورها nc - ns - ne ارزیابی زنبورها با جواب جدید	
۸	تکرارها	بازگشت به مرحله ۳ در صورت عدم اتمام تکرارها (iterations)	

خلاصه مراحل ذکر شده بدین صورت است که کلونی به روش مرتب‌سازی^۲ به سه دسته تقسیم شده است، زنبورهای نخبه، زنبورهای منتخب و باقی زنبورها. در مرحله جستجوی محلی، در اطراف زنبورهای نخبه به تعداد nre جستجو و در اطراف زنبورهای منتخب به تعداد nrs جستجو انجام می‌شود ($nre > nrs$). حال بهترین جواب ناشی از جستجو در اطراف هر زنبور با جواب قبلی مقایسه شده و در صورت بهبود یافتن، جایگزین آن می‌شود. سپس بهبود جواب هر یک از زنبورهای نخبه و منتخب نسبت به تکرارهای قبل بررسی می‌شود و در صورتی که در nl تکرار متواالی بهبودی حاصل نشده باشد یا به عبارتی در یک بهینه محلی افتاده باشند، با مقدار اولیه جدیدی جایگزین می‌شوند. در پی آن بهترین جواب در کل زنبورها ذخیره شده که بهترین جواب کل تکرارها نیز هست. در انتها برای بقیه زنبورهایی که در روند جستجو شرکت داده نشده‌اند، مقدار اولیه جدیدی در نظر گرفته

¹ Greedy Selection

² Sorting

می شود که این زنبورها نقش جستجوی سراسری را ایفاء می کنند. حال به مرحله طبقه بندی بازگشته تا زنبورهای نخبه و منتخب را از میان تمامی زنبورها دوباره مشخص کند. این روند تا آخرین تکرار ادامه می یابد.

۳-۱-۵ الگوریتم کلونی زنبور مصنوعی برای حل مسائل ترکیبی

الگوریتم ABC ذاتاً برای حل مسائل پیوسته مناسب است؛ ولیکن با اعمال تغییراتی در آن می توان آن را برای حل مسائل ترکیبی هم استفاده نمود. مقالات متعددی بدین منظور ارائه شده اند که چندین روش برای حل مسئله فروشنده دوره گرد راه هم شامل می شوند. این روش ها از عملگرهای ژنتیکی برای جستجوی محلی استفاده کرده اند و از جمله این روش ها می توان به CABC^۱ [۳۳] اشاره کرد که از عملگر GSTM برای جستجوی محلی استفاده کرده است. این روش توسط کارابوگا^۲ ^۳ بنیان گذار این الگوریتم ارائه شده است. همانند قل به منظور ساده سازی، این الگوریتم مرحله بندی شده و در

جدول ۲-۳ آورده شده است:

جدول ۲-۳ الگوریتم کلونی زنبور مصنوعی برای حل مسائل ترکیبی

تعداد زنبورهای کلونی (nc)، تعداد زنبورهای کارگر ^۳ (ne)، تعداد زنبورهای تماشاگر ^۴ (no)، تعداد زنبورهای پیشاہنگ ^۵ (ns)، تعداد تکرارها برای رها کردن یک بهینه محلی (nl)، تعداد کل تکرارها (iterations)	تعیین پارامترها	۰
اختصاص جواب اولیه به زنبورهای کارگر	مسیر اولیه	۱
محاسبه تابع شایستگی زنبورهای کارگر	ارزیابی	۲
جستجوی محلی زنبورهای کارگر در اطراف جواب متناظر آنها ارزیابی جواب جدید زنبورهای کارگر انتخاب حریصانه زنبورهای کارگر بین جواب جدید و قدیم	زنبورهای کارگر	۳

^۱ Combinatorial ABC

^۲ Karaboga

^۳ Employed Bees

^۴ Onlooker Bees

^۵ Scout Bees

no جستجوی محلی در اطراف زنبورهای کارگر بر اساس میزان شایستگی آنها ارزیابی زنبورهای تماشاگر	زنبورهای تماشاگر	۴
انتخاب حریصانه بین زنبورهای تماشاگر و زنbor کارگر متناظر آنها	ذخیره	۵
اختصاص مقادیر اولیه به ns زنbor بهبود نیافته در nl تکرار ارزیابی زنبورهای پیشahnگ	زنبورهای پیشahnگ	۶
بازگشت به مرحله ۳ در صورت عدم اتمام تکرارها (iterations)	تکرارها	۷

بیان ساده‌تر این مراحل بدین صورت است که در واقع به تعداد زنبورهای کارگر جواب مجزا داریم، ابتدا برای هر جواب یک جستجوی محلی انجام داده و نتیجه را با مقدار قبلی مقایسه کرده و در صورت بهبود جایگزین مقدار قبلی می‌کنیم. حال زنبورهای تماشاگر را بسته به شایستگی بین زنبورهای کارگر تقسیم می‌کنیم. با توجه به اینکه هر یک از این زنبورها معادل یک جستجوی محلی است، در واقع به تعداد زنبورهای تماشاگر جستجوی محلی انجام می‌شود، ولیکن این جستجوها در قالب احتمالاتی بین تمام جواب‌ها پخش می‌شوند. در این پیاده‌سازی بدین منظور از چرخ رولت^۱ استفاده شده است. سپس جواب‌ها با تکرارهای قبلی مقایسه شده و حداقل ntsای آنها (زنبورهای پیشahnگ) که در nl تکرار متوالی بهبود پیدا نکرده باشند با مقادیر اولیه جایگزین می‌شوند. همانطور که مشخص است زنبورهای پیشahnگ مسئول جستجوی سراسری هستند. حال به مرحله زنbor کارگر بازگشته و این روند در هر تکرار و تا انتها ادامه می‌یابد.

۳-۱-۶ الگوریتم بهینه‌سازی کلونی زنbor برای حل مسائل ترکیبی

به طورکلی، روش‌های حل مسئله فروشنده دوره‌گرد در دو مقوله گسترده طبقه‌بندی می‌شوند: الگوریتم‌های دقیق و الگوریتم‌های تقریبی [۳۴]. الگوریتم‌های دقیق روش‌هایی هستند که از مدل‌های ریاضی بهره‌برداری می‌کنند، در حالی که الگوریتم‌های تقریبی از بهبودهای ابتکاری^۲ و تکراری^۳ به

¹ Roulette Wheel

² Heuristic

³ Iterative

عنوان فرآیند حل مسئله استفاده می‌کنند. الگوریتم‌های تقریبی می‌توانند در دو گروه طبقه‌بندی شوند: ابتکاری سازنده^۱ و ابتکاری بهبوددهنده^۲ [۳۵]. الگوریتم بهینه‌سازی کلونی زنبور از جمله روش‌های سازنده است و ذاتاً برای حل مسائل ترکیبی بکار می‌آید. همانطور که در فصل دوم مشاهده شد از این الگوریتم برای حل مسئله فروشنده دوره‌گرد نیز استفاده شده است. [۳۵] در الگوریتم BCO اصلی تغییراتی اعمال کرده که به عملکردی همانند الگوریتم مورچه که مبتنی بر فرمون است، منجر شده است. بدین معنی که زنبورها این امکان را داشتند که تعداد دفعاتی که زنبورهای دیگر، یال‌های مشخصی را بازدید کرده‌اند را به خاطر بسپارند. در سال ۲۰۱۱ نوع بهبوددهنده این الگوریتم با نام BCOi^۳ ارائه شد [۳۶] که برای حل مسائل ترکیبی همچون p-مرکز^۴ بکار گرفته شد. در این پیاده‌سازی روش بهبوددهنده برای حل مسئله فروشنده دوره‌گرد با تغییراتی به منظور اضافه کردن جستجوی سراسری به الگوریتم، ارائه شده است. در جدول ۳-۳ مراحل این الگوریتم آمده است:

جدول ۳-۳ الگوریتم بهینه‌سازی کلونی زنبور گستته برای حل مسائل ترکیبی

تعیین پارامترها		
تعداد زنبورهای کلونی (nc)، تعداد گذرهای پیشرو (nfp)، تعداد کل تکرارها (iterations)		.
اختصاص جواب اولیه به زنبورها	مسیر اولیه	۱
محاسبه تابع شایستگی زنبورها	ارزیابی	۲
جستجوی محلی در اطراف هر زنبور ارزیابی جواب جدید انتخاب حریصانه بین جواب جدید و قدیم	گذر پیشرو	۳
بازگرداندن زنبورها به کندو و سربازگیری مجدد	گذر پیشرو ^۵	۴

¹ Constructive Heuristics

² Improvement Heuristics

³ BCO Improvement Concept

⁴ P-Center

⁵ Backward Pass

بازگشت به گذر پیشرو در صورت عدم اتمام nfp تکرار متوالی از دو گذر	تکرار گذرها	۵
ذخیره بهترین جواب	ذخیره	۶
اختصاص مقادیر اولیه به زنبور با بدترین تابع شایستگی ارزیابی زنبور	بدترین زنبور	۷
بازگشت به مرحله 3 در صورت عدم اتمام تکرارها (iterations)	تکرارها	۸

بيان جامع تر اين الگوريتم بدین شكل است که به تعداد زنبورها جواب اولیه در نظر می گيريم. در گذر پیشرو^۱ برای هر جواب يك جستجوی محلی انجام داده و نتيجه را با مقدار قبلی مقایسه کرده و در صورت بهبود جایگزين مقدار قبلی می کنيم. حال بر اساس تابع شایستگی جوابها و فرآيندی احتمالاتی همانند چرخ رولت برای جوابهای حاصل سربازگری می کنيم، بدین معنی که زنبورها را به جوابها اختصاص می دهیم. در واقع تصمیم گرفته می شود که هر جواب چند بار در جمعیت جدید (nc) تکرار شود. همانند الگوريتم اصلی اين دو گذر به ترتیب و به تعداد nfp بار تکرار خواهند شد. بعد از ذخیره بهترین جواب تا تکرار اخیر، بدترین جواب مسئله را با يك مقدار اولیه جدید جایگزين می کنيم و به مرحله گذر پیشرو بازگشته و اين روند تا آخرین تکرار دنبال می شود.

۷-۱-۳ نقاط ضعف و قوت سه الگوريتم کلونی زنبورها در حل مسائل ترکیبی

نقاط قوت الگوريتم زنبور عسل:

- تمام زنبورها (منتخب و نخبه) در هر تکرار برای اجتناب از محصور شدن در بهینه های محلی بررسی می شوند.
- در پیاده سازی نرم افزاری که تمامی مراحل به ترتیب و يك اجرا می شوند، تمیز زنبورهای نخبه از زنبورهای منتخب و اختصاص تعداد دفعات جستجوی محلی بیشتر به زنبورهای نخبه باعث کاهش زمان اجرای الگوريتم می شود.

¹ Forward Pass

- الگوریتم بسیار ایستا بوده بدین معنی که تعداد بهینه‌سازی‌های محلی و نحوه سربازگیری به فرآیندهای مبتنی بر احتمالات همانند چرخ رولت وابسته نیست که این ویژگی الگوریتم را مناسب بسترها ساختافزاری که قابلیت موازی‌سازی دارند، می‌سازد.

نقاط ضعف الگوریتم زنبور عسل:

- در هر تکرار زنبورهایی که نه جزء زنبورهای نخبه هستند نه زنبورهای منتخب با جواب جدید جایگزین می‌شوند. در حل مسائل پیوسته این ویژگی یک امتیاز به حساب آمده، ولی در حل مسائل ترکیبی که تعداد جواب‌های مختلف و نزدیک به بهینه زیاد نیستند (همانند مسیر نزدیکترین همسایگی که به تعداد شهرها محدود می‌شود)، این ویژگی یک محدودیت است؛ چراکه همواره مقادیر تکراری را وارد چرخه کرده و این مقادیر تکراری در مرحله طبقه‌بندی در انتهای صفحه قرار گرفته و حذف می‌شوند. به عبارتی بدون اینکه این روند تاثیری در نتایج داشته باشد مرتباً تکرار شده که در نتیجه سبب کند شدن اجرای الگوریتم می‌شود.

- در هر تکرار الگوریتم، جواب متناظر با زنبورهای باقی‌مانده جایگزین جواب‌های در حال بررسی می‌شود. بالطبع اگر این جواب‌های جدید تابع شایستگی بهتری نسبت به جواب‌های قبلی داشته باشند، تعدادی از جواب‌هایی که هنوز به طور کامل مورد بررسی و بهینه‌سازی محلی قرار نگرفته‌اند، از دور خارج می‌شوند. در حالی که چه بسا بررسی بیشتر این جواب‌ها می‌توانست به نتایج نهایی بهتری منجر شود. همچنین خود این جواب‌ها که جواب نامناسب‌تری نسبت به بقیه دارند، هرگز مورد بررسی قرار نمی‌گیرند. در واقع این ضعف به الگوریتم زنبور وارد است که تعدادی از جواب‌ها را بدون هرگونه بررسی و تعدادی از جواب‌ها را با بررسی‌های کمتر از مقدار لازم از مجموعه جواب‌های مسئله خارج می‌کند.

نقاط قوت الگوریتم کلونی زنبور مصنوعی:

- امکان کنار گذاشتن جواب‌های در حال بررسی نسبت به الگوریتم زنبور کمتر است و پس از بررسی در صورت عدم بهبود در $n!^{th}$ تکرار با جواب‌های جدید تعویض می‌شوند.

- تمامی زنبورها مراحل بهینه‌سازی محلی را طی می‌کنند و یک جواب تصادفی به صرف اینکه در ابتدا تابع شایستگی مناسبی نداشته کنار گذاشته نمی‌شود.

نقاط ضعف الگوریتم کلونی زنبور مصنوعی:

- بر خلاف الگوریتم زنبور ساختار پویا داشته یعنی تعداد بهینه‌سازی‌های محلی به روندهای احتمالاتی همانند چرخ رولت وابسته است. در پیاده‌سازی‌های سخت‌افزاری که می‌توان آن را هدف نهایی در نظر گرفت، این ویژگی منجر به کاهش موازی‌سازی خواهد شد.

نقاط قوت الگوریتم بهینه‌سازی کلونی زنبور:

- تعداد پارامترهای کم
- تنها بدترین زنبور در هر تکرار جایگزین می‌شود که مزیتی است نسبت به الگوریتم ABC که در آن تنها جوابی که بهبود نمی‌یابد، جایگزین می‌شود و در نتیجه فرآیند جستجوی تصادفی بسیار کند است. همچنین نسبت به الگوریتم BA که جواب‌هایی را بدون بررسی کنار می‌گذارد یا به عبارتی فرآیند جستجوی تصادفی آن بسیار سریع است، برتری محسوب می‌شود.

نقاط ضعف الگوریتم بهینه‌سازی کلونی زنبور:

- به واسطهٔ نحوه سربازگیری اجرای این الگوریتم نسبت به دو الگوریتم دیگر کندتر است.
- به دلیل نوع فرآیند سربازگیری، جواب‌های تکراری در کل جمعیت زیاد بوده که در تکرارهای بالاتر منجر به توقف بهینه‌سازی و محصور شدن در بهینهٔ محلی می‌شود.

۲-۳ الگوریتم کلونی زنبور پیشنهادی برای مسائل ترکیبی

روش جدیدی که ارائه شده است تلفیقی از سه روش الگوریتم زنبور عسل (BA)، الگوریتم کلونی زنبور مصنوعی (ABC) و بهینه‌سازی کلونی زنبور (BCO) است. نقاط ضعف و قوت هر یک از سه روش در نظر گرفته شده است و حاصل آن ارائهٔ روشنی است که ترکیبی از نقاط قوت این الگوریتم‌ها است. همچنین در کنار یک بستر قدرتمند همانند FPGA که توانایی بالایی در موازی‌سازی دارد، این روش مناسب‌ترین الگوریتم برای پیاده‌سازی بر روی سخت‌افزار است. این روش الگوریتم زنبور

سخت افزاری (HBA) نام نهاده شده است. چراکه از نظر ساختار بسیار به الگوریتم BA نزدیک بوده و همچنین برای اهداف سخت افزاری مناسب تر است. در جدول ۴-۳ مراحل این الگوریتم مشاهده می شود:

جدول ۴-۳ الگوریتم HBA برای حل مسائل ترکیبی

تعداد زنبورهای تماشاگر (no)، تعداد زنبورهای کارگر (ne)، ، تعداد زنبورهای پیشاهنگ (ns)، تعداد تکرارها برای رها کردن یک بهینه محلی (nl)، تعداد تکرارها برای جایگزینی زنبورهای پیشاهنگ (is)، تعداد کل تکرارها (iterations)	تعیین پارامترها	.
اختصاص جواب اولیه به زنبورهای تماشاگر	مسیر اولیه	۱
محاسبه تابع شایستگی زنبورهای تماشاگر	ارزیابی	۲
ne جستجوی محلی در اطراف هر زنبور تماشاگر (اختصاص ne زنبور کارگر) ارزیابی زنبورهای کارگر انتخاب حریصانه بین زنبورهای کارگر و زنبور تماشاگر متناظر آنها اختصاص جواب اولیه به زنبورهای تماشاگر بهبود نیافته در nl تکرار	زنبورهای کارگر	۳
ذخیره بهترین جواب	ذخیره	۴
بازگشت به مرحله ۳ در صورت عدم اتمام is تکرار	تکرار ۱	۵
مرتب کردن زنبورهای تماشاگر بر اساس تابع شایستگی	طبقه‌بندی	۶
اختصاص مقدار اولیه به زنبورهای پیشاهنگ ارزیابی زنبورهای پیشاهنگ	زنبورهای پیشاهنگ	۷
بازگشت به مرحله ۳ در صورت عدم اتمام تکرارها (iterations)	تکرار ۲	۸

حال به توضیح روش پیشنهادی می پردازیم. جوابهای اولیه به تعداد زنبورهای تماشاگر در نظر گرفته شده و ارزیابی می شوند. در واقع زنبورهای تماشاگر وظیفه نگهداری بهترین جوابها را به

¹ Hardware Bee Algorithm

عهده دارند. در الگوریتم حال حاضر وظیفه زنبور تماشاگر نسبت به واقعیت و نحوه غذایابی زنبور عسل در طبیعت تغییر داده شده است. حال به هر زنبور تماشاگر ne زنبور کارگر اختصاص داده می‌شود که معادل ne جستجوی محلی در اطراف زنبور تماشاگر است. بهترین جواب بین زنborهای کارگر و زنبور تماشاگر متناظر انتخاب شده و به عنوان جواب جدید زنبور تماشاگر در نظر گرفته می‌شود. در صورتی که هر یک از زنborهای تماشاگر در nl تکرار بهبودی حاصل نکرده باشد، جواب آنها با مقادیر جدید جایگزین می‌شود. حال بهترین جواب را ذخیره کرده که این امر تضمین می‌کند که بهترین نتیجه حاصل تا تکرار اخیر از دست نمی‌رود، اگرچه زنبور متناظر آن به دلیل عدم بهبود در nl تکرار جایگزین شده باشد. در این مرحله تعداد تکرارها بررسی شده و اگر به تعداد تکرار is نرسیده باشیم به دو مرحله قبل باز می‌گردد. در صورت اتمام تکرارها شمارنده متناظر آن بازنمانی شده و زنborهای تماشاگر به هدف تعیین زنborهای پیشاهمگ طبقه‌بندی می‌شوند ($ns < no$). در مرحله بعد زنborهای پیشاهمگ را با جواب‌های جدیدی جایگزین کرده، آنها را ارزیابی کرده و به مرحله زنborهای کارگر باز می‌گردد. این روند تا جایی که تکرارهای کل الگوریتم پایان بیابد تکرار می‌شود.

وجود زنborهای کارگر یا به عبارتی تعداد جستجوهای محلی بیشتر در هر تکرار همانند الگوریتم زنبور باعث سرعت بیشتر جستجوی جواب بهینه نسبت به الگوریتم کلونی زنبور مصنوعی یا الگوریتم بهینه‌سازی کلونی زنبور می‌شود.

در الگوریتم ارائه شده در نظر گرفتن is تکرار قبل از جایگزینی زنborهای پیشاهمگ از این روست که جواب‌ها بدون بررسی‌های لازم همانند الگوریتم زنبور از چرخه خارج نشوند. همچنین وجود زنborهای پیشاهمگ منجر به جستجوی تصادفی سریعتر در فضای مسئله می‌شود و ضعف کندي جستجو در الگوریتم ABC بدین شکل مرتفع می‌شود. این جایگزینی همانند الگوریتم BCO بر روی زنborهایی با بدترین میزان شایستگی در نظر گرفته می‌شود و تفاوت این زنborها با زنborهای بیکار در الگوریتم BA در بررسی بیشتر این جواب‌ها در تکرارهای بعد است.

۳-۳ پیاده‌سازی سخت‌افزاری

در این قسمت نحوه پیاده‌سازی الگوریتم ارائه شده بر روی FPGA، توضیح کلیت واحدهای متفاوت آن و ویژگی‌های اصلی هر یک از این واحدها ارائه خواهد شد. در ادامه به توضیح بخش‌های اصلی این پیاده‌سازی خواهیم پرداخت.

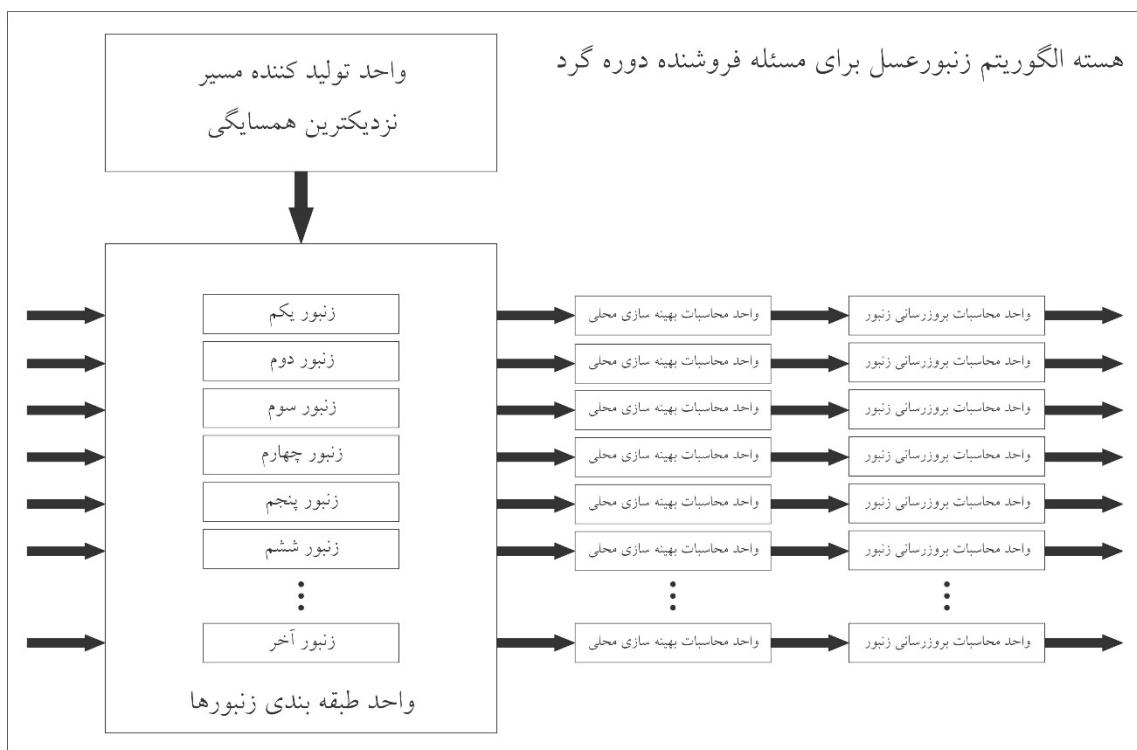
۱-۳-۳ ساختار کلی پیاده‌سازی (واحد اصلی^۱)

به منظور پیاده‌سازی بر روی FPGA، از روش ارائه شده استفاده شده است که تغییراتی در آن در نظر گرفته شده است. این تغییرات به دو دلیل است: اول آنکه روش ارائه شده مناسب پیاده‌سازی بر روی سخت‌افزار شود و دلیل دوم امکان تبدیل ساده‌تر الگوریتم ارائه شده به سه الگوریتم دیگر به منظور مقایسه در کارهای آینده است. تغییرات ایجاد شده بدین شکل است:

- زنبورهایی که در $n!$ تکرار بهبود نیافته‌اند با مقادیر جدید جایگزین نمی‌شوند؛ چراکه تعداد جواب‌های جدید محدود بوده و بررسی بیشتر جواب حال حاضر زنبور روند بهتری به حساب می‌آید. همچنین باعث پیاده‌سازی بسیار پیچیده‌تر سخت‌افزاری و مصرف منابع موجود FPGA می‌شود.
- مرحله طبقه‌بندی قبل از جستجوی بهینه‌های محلی قرار گرفته است. دلیل این امر تبدیل ساده‌تر این روش به الگوریتم زنبور و همچنین قرار دادن بهترین نتیجه در اولین زنبور و مشاهده روند حل مسئله بدون ایجاد پیچیدگی بیشتر و مصرف منابع سخت‌افزار اضافی به منظور جستجوی بهترین جواب در هر تکرار است. ولیکن این تغییر منجر به افزایش زمانی اندک در بدست آوردن جواب بهینه می‌شود.
- تعداد زنبورهای پیشاهنگ یکی در نظر گرفته شده‌اند، چراکه تعداد مسیرهای نزدیکترین همسایگی محدود بوده و در نتیجه تنها به تعداد شهرها، مسیر نزدیکترین همسایگی جدید موجود است.

در ادامه به توضیح نحوه پیاده‌سازی این الگوریتم بر روی سخت‌افزار می‌پردازیم.

¹ Top Module



شکل ۷-۳ طرح کلی الگوریتم ارائه شده بر روی سخت‌افزار

در شکل ۷-۳ طرح کلی این پیاده‌سازی مشاهده می‌شود. همانطور که مشخص است ساختار کلی این طرح از چندین زیربخش یا به عبارتی هسته تشکیل شده است که خود این هسته‌ها دارای زیرهسته‌های دیگری هستند که به صورت مژو در قسمت‌های بعدی این فصل به آن‌ها خواهیم پرداخت.

این ساختار از پنج هسته^۱ زیرمجموعه اصلی تشکیل شده است:

- هسته تولید کننده مسیر نزدیکترین همسایگی^۲
- هسته زنبور عسل
- هسته طبقه‌بندی^۳ زنبورها
- هسته محاسبات بهینه‌سازی محلی^۴
- هسته محاسبات بروزرسانی^۵ زنبور

¹ Core

² Nearest Neighbor Tour (NNT)

³ Sorting

⁴ Local Optimization Search

⁵ Update

کارکرد کلی این ساختار بدین شکل است که در شروع پروسه، سیستم به مدت کوتاهی در حال بازنشانی^۱ خواهد ماند. پس از گذار از این حالت اگر به هسته اصلی سیگنال فعالسازی واحد داده شود، واحد تولیدکننده مسیر نزدیکترین همسایگی داده‌ها را به صورت تک تک بر روی زبورها بارگذاری می‌کند. پس از بارگذاری کامل تمام زبورها که معادل جواب اولیه و تصادفی هستند، الگوریتم این زبورها را از نظر میزان شایستگی بررسی کرده و این زبورها را مرتب کرده؛ به عبارتی اطلاعات آن‌ها را جایه‌جا کرده تا جایی که تمامی زبورها بر اساس تابع شایستگی‌شان به صورت صعودی چیده شوند. حال در این مرحله جستجوی محلی در هسته‌های مربوطه شروع شده و پس از اینکه یک زبور به جواب بهتری نائل شد، اطلاعات موجود بر روی آن بروزرسانی می‌شود. بعد از آن دوباره به مرحله طبقه‌بندی برگشته و اطلاعات درون زبورها بروزرسانی می‌شود و دوباره مرحله بهینه‌سازی محلی و بروزرسانی اجرا خواهد شد. این روال تا جایی ادامه خواهد داشت که الگوریتم به واحد تولید مسیر نزدیکترین همسایگی فرمان بارگذاری مقدار جدید در زبور با بدترین میزان شایستگی را صادر کند. سپس دوباره تمامی مراحل با در نظر گرفتن مقدار جدید بارگذاری شده تکرار خواهد شد. این توالی یعنی جستجوی محلی، بروزرسانی، جایگزینی بدترین زبور در تکرارهای مشخص شده تا آخرین تکرار الگوریتم ادامه خواهد داشت و نتیجه بدست آمده در آخرین تکرار، نتیجه نهایی خواهد بود.

لازم به ذکر است که واحد مسیر نزدیکترین همسایگی، یکتا در ساختار در نظر گرفته شده است و این واحد اطلاعات مسیر حرکت فروشنده دوره‌گرد را به صورت متوالی و تک تک برای واحد زبورها می‌فرستد. دلیل اینکه این واحد یکتا در نظر گرفته شده است آنست که نیازی به ایجاد خروجی در تمام مدت کارکرد برای این واحد نبوده و تنها در زمان‌های مشخصی درخواست بارگذاری داده‌های جدید بر روی زبور به این هسته داده می‌شود؛ ولیکن در عین حال در تمام مدت این واحد مشغول محاسبات تولید مسیرهای جدید بوده و پس از اتمام تولید یک مسیر جدید منتظر درخواست ارسال می‌ماند؛ به عبارتی موازی با دیگر هسته‌ها عمل می‌کند.

¹ Reset

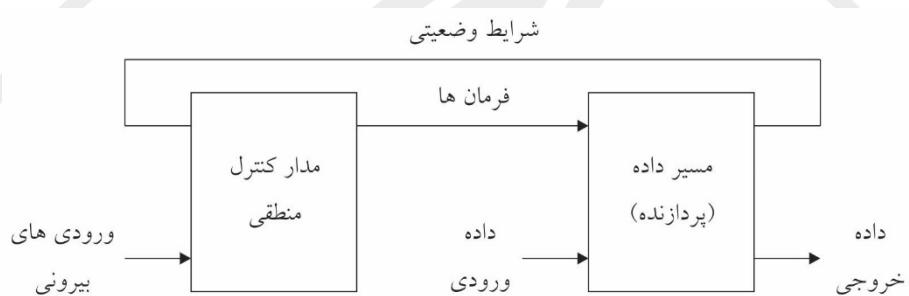
۱-۱-۳-۳ نمودار ماشین حالت کراندار^۱

اطلاعات دودویی ذخیره شده در یک سیستم دیجیتال را می‌توان به دو صورت اطلاعات داده و کنترل دسته‌بندی کرد، داده عناصر گستره‌ای از اطلاعات است که برای انجام اعمال حسابی، منطقی، جابجایی و دیگر عملیات پردازش داده، دستکاری می‌شود. این عملیات با عناصر دیجیتالی چون جمع کننده‌ها، دیکدرها، مولتی پلکسراها، شمارنده‌ها و شیفت رجیسترها پیاده‌سازی می‌شوند. اطلاعات کنترلی، سیگنال‌های فرمانی که انواع عملیات را در بخش داده مدیریت می‌کنند، تولید می‌نمایند تا کار پردازش داده مورد نظر انجام شود. طراحی منطقی یک سیستم دیجیتال را می‌توان به دو بخش تقسیم کرد. یک بخش متعلق به مدارهای دیجیتال است که عملیات پردازش داده را انجام می‌دهند، بخش دیگر مربوط به طراحی مدارهای کنترلی است که عملیات مختلف و توالی آن‌ها را مدیریت می‌نماید. رابطه بین منطق کنترل و پردازش داده در یک سیستم دیجیتال در شکل ۸-۳ مشاهده می‌شود. مسیر پردازش داده، که به آن عموماً مسیر داده می‌گویند، بر حسب نیازهای سیستم، داده را در ثبات دستکاری می‌کند. مدار کنترل یک رشته از فرامین را به مسیر داده صادر می‌نماید. مدار کنترل شرایط وضعیت را از مسیر داده دریافت کرده و برای تعیین رشته سیگنال‌های کنترل از آن‌ها استفاده می‌کند. مدار کنترلی که سیگنال‌هایی را برای توالی عملیات در مسیر داده تولید می‌کند، یک مدار ترتیبی است که حالات داخلی اش فرامین کنترل را به سیستم دیکته می‌نماید. در هر زمان، حالت کنترل ترتیبی مجموعه‌ای از فرامین از پیش تعیین شده‌ای را آغاز می‌کند. بسته به شرایط وضعیت و دیگر ورودی‌های بیرونی، کنترل ترتیبی به حالت بعدی می‌رود تا اعمال دیگری را در مسیر داده آغاز کند و نیز حالت بعدی خود زیرسیستم کنترل را معین نماید.

وظایف رشته کنترل و مسیر داده سیستم دیجیتال با الگوریتم‌های سخت‌افزاری مشخص می‌گردد. یک الگوریتم متشكل است از تعداد معینی از مراحل اجرایی یا رویه‌ای، که چگونگی تهیه یک حل را برای یک مسئله با یک مقدار تجهیزات پیاده‌سازی می‌کند. ابتکاری‌ترین و مشکل‌ترین بخش از طراحی، فرموله کردن الگوریتم‌های سخت‌افزاری برای رسیدن به اهداف است.

^۱ Finite State Machine (FSM)

فلوچارت روشنی مناسب برای مشخص کردن رشته مراحل اجرایی و مسیرهای تصمیم‌گیری در یک الگوریتم است. فلوچارت برای الگوریتم سخت‌افزاری جملات را به یک نمودار اطلاعاتی تبدیل می‌کند که رشته عملیات آن‌ها را به همراه شرایط لازم‌الاجراء یک به یک می‌شمارد. یک فلوچارت خاص که فقط برای تعریف الگوریتم‌های سخت‌افزاری به وجود آید چارت ماشین حالت نام دارد. ماشین حالت نیز نام دیگری برای یک مدار ترتیبی است که ساختار مبنا برای یک سیستم دیجیتال است.



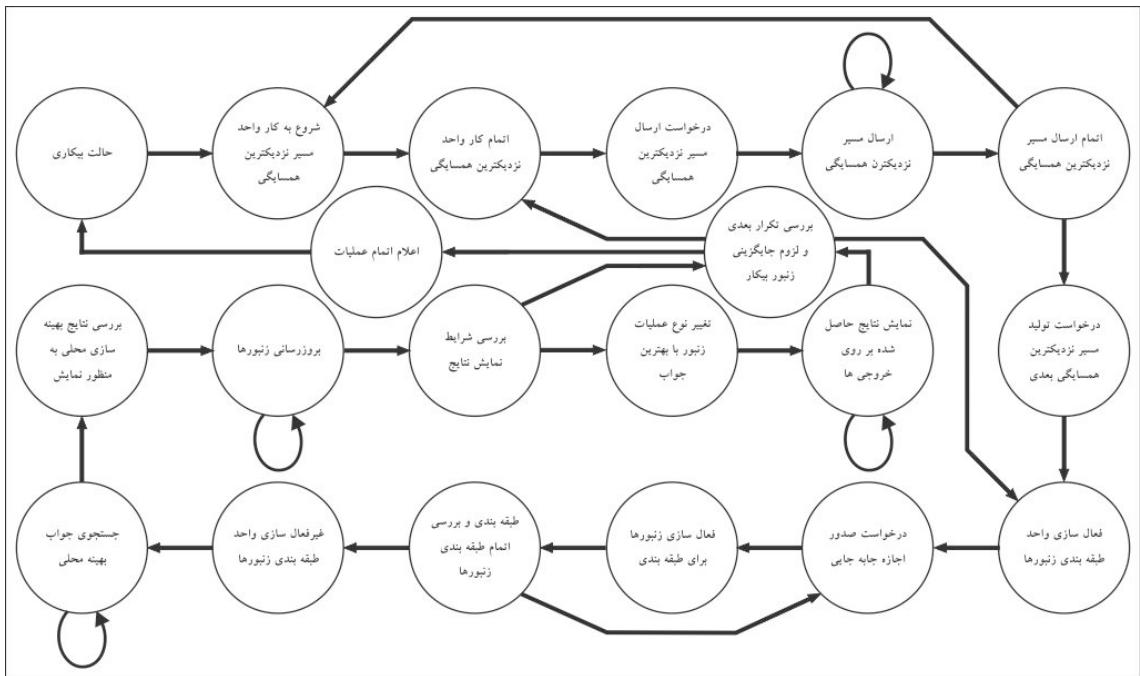
شكل ٨-٣ واکنش بین کنترل و مسیر داده

چارت ماشین حالت ظاهراً همان چارت معمولی است، ولی تفسیر متفاوتی دارد. فلوچارت معمولی رشته مراحل اجرایی و مسیرهای تصمیم‌گیری را برای یک الگوریتم به ترتیب و بدون ملاحظه ارتباط زمانی آن‌ها، توصیف می‌کند. چارت FSM رشته وقایع به همراه ارتباط زمانی بین حالات یک کنترل-گر و وقایعی که ضمن رفتن از یک حالت به حالت بعدی رخ می‌دهند را توصیف می‌نماید. این چارت خصوصاً برای مشخص کردن دقیق رشته کنترل و عملیات مسیر داده (واحد پردازش) در یک سیستم دیجیتال انتخاب شده است، ضمن این که محدودیت‌های سخت‌افزار دیجیتال نیز لحاظ شده است (مانو، ۱۳۸۱).

٣-١-٢ نمودار ماشین حالت واحد اصلی

در شکل ۹-۳، نمودار FSM^۱ این واحد نشان داده شده است و به صورت مسروچ به توضیح نحوه کنترل هسته‌های زیرمجموعه پرداخته شده است.

1 Finite State Machine



شكل ٩-٣ نمودار FSM واحد اصلی

- حالت بیکاری: در این مرحله سیگنال فعال‌سازی هسته اصلی بررسی شده و در صورت دریافت آن، سیگنال فعال‌سازی به هسته مسیر نزدیکترین همسایگی ارسال و مقادیر ثبات‌های شمارنده نیز بازنمانی خواهد شد.
 - شروع به کار واحد نزدیکترین همسایگی: در این حالت هسته اصلی درخواست تولید مسیر جدید را به واحد تولید مسیر نزدیکترین همسایگی ارسال می‌کند.
 - اتمام کار واحد مسیر نزدیکترین همسایگی: هسته اصلی منتظر سیگنال اتمام عملیات از هسته مسیر نزدیکترین همسایگی خواهد ماند، پس از آماده شدن این مسیر و اعلام آمادگی واحد برای ارسال این داده‌ها، زبور متناظر را در حالت ذخیره‌سازی یا به عبارتی ذخیره اطلاعات ورودی‌ها قرار می‌دهد.
 - درخواست ارسال مسیر نزدیکترین همسایگی: در این حالت دستور ارسال مسیر نزدیکترین همسایگی و فعال‌سازی زبور متناظر برای ذخیره اطلاعات ارسال شده صادر خواهد شد.
 - ارسال مسیر نزدیکترین همسایگی: در این حالت مقادیر ذخیره شده در واحد نزدیکترین همسایگی که همان توالی شهرها برای مسیر فروشنده دوره‌گرد است، به صورت سری از این

واحد ارسال شده و به صورت سری و در هر کلاک پالس یک مقدار بر روی زنبور متناظر ذخیره می‌شود. توجه شود که اگرچه طول قابل ارسال و قابل دریافت هر دو واحد یعنی مسیر نزدیکترین همسایگی و زنبور بیشتر از تعداد شهرهاست (۲۵۶ سلول)، ولیکن تنها به تعداد شهرهای موجود در مسئله اطلاعات جابه‌جا خواهد شد. دلیل اینکه این طول به این میزان در نظر گرفته شده است امکان تغییر راحت‌تر بین مسائل متفاوت است. همچنین این طول از واحد اصلی قابل تغییر بوده و امکان افزایش یا کاهش آن به منظور حل مسائلی با شهرهای بیشتر از ۲۵۵ عدد و یا کاهش مصرف منابع سخت‌افزاری متناظر با تعداد شهرهای هر مسئله وجود دارد.

- اتمام ارسال مسیر نزدیکترین همسایگی: در مرحله حال حاضر اتمام عملیات ذخیره‌سازی زنبور مورد بررسی قرار می‌گیرد و در صورت اتمام کار و باقی‌ماندن زنبورهایی بدون جواب اولیه به مرحله اول یعنی جایی که دستور ساخت مسیر بعدی برای واحد مسیر نزدیکترین همسایگی صادر می‌شود، انتقال خواهد یافت. با طی مراحل اول تا این مرحله زنبور بعد مقادیر خود را دریافت خواهد کرد. این روال تا اتمام تمامی زنبورها ادامه خواهد داشت. این مراحل و ذخیره اولیه بر روی زنبورها تنها در اولین تکرار انجام خواهد شد و در مراحل بعدی واحد تولیدکننده مسیر نزدیکترین همسایگی فقط یک مسیر و تنها برای زنbor با بدترین میزان شایستگی را تولید خواهد کرد.
- درخواست تولید مسیر نزدیکترین همسایگی بعدی: به منظور موازی‌سازی هرچه بیشتر پس از اتمام ذخیره‌سازی مسیرهای اولیه و مقادیر شایستگی متناظر آنها بر روی تمامی زنبورها، در این مرحله دستور ساخت مسیر بعدی به واحد تولید مسیر نزدیکترین همسایگی و در یک کلاک پالس داده خواهد شد. در نتیجه در تمامی مراحل اجرای الگوریتم، واحد تولید مسیر نزدیکترین همسایگی به صورت موازی با بقیه واحدها فعالیت خواهد کرد.
- فعال‌سازی واحد طبقه‌بندی زنبورها: پس از ذخیره مقادیر اولیه مسیرها و توابع شایستگی متناظر آنها روی زنبورها، نوبت به طبقه‌بندی این مسیرها یا به عبارتی زنبورها بر اساس

مقادیر تابع شایستگی آن‌ها می‌رسد. به منظور طبقه‌بندی زنبورها و با فرض ثابت نگه داشتن جایگاه آن‌ها، باید اطلاعات زنبورها به گونه‌ای تعویض شوند که در نهایت تابع شایستگی آن‌ها به صورت صعودی مرتب شده باشد. بدین صورت که زنبور اول دارای کمترین تابع شایستگی (کوتاه‌ترین مسیر) برای مسئله فروشنده دوره‌گرد باشد. مقدار تابع شایستگی زنبور دوم بیشتر از زنبور اول ولی کمتر از زنبور سوم باشد و به همین روال تا آخرین زنبور که دارای بیشترین مقدار تابع شایستگی (طولانی‌ترین مسیر) باشد. پس از مقایسه مقادیر تابع شایستگی زنبورها، اگر دو زنبور اجازه جابه‌جایی مسیرهای خود را داشته باشند، این مسیرها به همراه مقدار تابع شایستگی متناظر آن‌ها به صورت متوالی و همزمان بین دو زنبور جابه‌جا خواهد شد. همزمان بودن این جابه‌جایی بدین معنی است که یک زنبور با ارسال یک خانه حافظه خود برای زنبور دیگر، مقدار دریافتی از زنبور دیگر را در همان خانه حافظه خود ذخیره می‌کند و احتیاجی به حافظه واسط^۱ وجود ندارد. پس از اتمام یک مرحله جابه‌جایی دوباره مقادیر تابع شایستگی زنبورها با یکدیگر مقایسه شده و مرحله دیگری از جابه‌جایی شروع خواهد شد. این مقایسه‌ها و جابه‌جایی‌ها تا جایی ادامه پیدا می‌کند که زنبورها بر اساس مقادیر تابع شایستگی آن‌ها به صورت صعودی مرتب شوند. به منظور بیشترین بازدهی، در هر یک از مراحل تعویض اطلاعات زنبورها، بیشینه تعداد زنبورها امکان عملیات جابه‌جایی اطلاعات را خواهند داشت. بدین معنی که چندین زنبور این امکان را دارند که به صورت موازی و جفت جفت اطلاعات خود را با هم تعویض کنند. برای مثال همزمان با زنبور اول و دوم، زنبور چهارم و پنجم هم امکان تعویض اطلاعات خود را خواهند داشت. لازم به توضیح است که امکان جابه‌جایی سه زنبور مجاور وجود ندارد، یعنی مثلاً امکان جابه‌جایی دو زنبور اول و دوم و دو زنبور دوم و سوم در یک زمان نیست. حال برای شروع عملیات طبقه‌بندی و با توضیحات داده شده، در این حالت سیگنال فعال‌سازی به واحدهای صدور اجازه جابه‌جایی ارسال خواهد شد.

¹ Buffer

- درخواست صدور اجازه جابه‌جایی: در این حالت، درخواستی به هسته‌های صدور اجازه جابه‌جایی برای صدور اجازه‌های جدید ارسال خواهد شد. هسته‌های صدور اجازه جابه‌جایی برای زنبورها با توجه به مقادیر تابع شایستگی آن‌ها اجازه جابه‌جایی صادر خواهند کرد. ولیکن باید در نظر داشت که این اجازه‌ها نباید در میانه فرآیند جابه‌جایی اطلاعات زنبورها تغییر کند. با توجه به اینکه در طی مراحل طبقه‌بندی، دو زنبور مقادیر مسیر و تابع شایستگی خود را با هم تعویض می‌کنند؛ با تغییر مقادیر تابع شایستگی زنبورها، اجازه جابه‌جایی آن‌ها نیز دستخوش تغییرات می‌شود. به همین دلیل صدور اجازه جابه‌جایی باید از خارج از هسته‌های صدور اجازه جابه‌جایی قابل کنترل باشد.
- فعال‌سازی زنبورها برای طبقه‌بندی: در این مرحله برای زنبورهایی که اجازه جابه‌جایی اطلاعات را دارند، سیگنال فعال‌سازی ارسال می‌شود. برخلاف آنچه که در این مرحله و مرحله بعدی اتفاق می‌افتد، قاعده‌تاً باید اول نوع عملیات هسته زنبور عسل (در اینجا ارسال اطلاعات ذخیره شده به خروجی‌ها و ذخیره اطلاعات ورودی‌ها) را مشخص کرد و سپس سیگنال فعال‌سازی را ارسال کرد. ولیکن هسته زنبور به گونه‌ای طراحی شده است که یک کلک پالس بعد از دریافت سیگنال فعال‌سازی، نوع عملیات درخواستی را بررسی می‌کند. پس این امکان وجود دارد که در این مرحله سیگنال فعال‌سازی ارسال شود و در حالت بعدی نوع عملیات را مشخص کرد.
- طبقه‌بندی و بررسی اتمام طبقه‌بندی زنبورها: در این حالت نوع عملیات روی زنبورها که همان ارسال اطلاعات ذخیره شده به خروجی‌ها و ذخیره اطلاعات ورودی‌ها است، مشخص می‌شود. همچنین در این مرحله، شرط اتمام این جابه‌جایی‌ها بررسی می‌شود؛ در صورتی که هنوز تمام زنبورها به ترتیب مورد نظر نرسیده باشند، به مرحله درخواست صدور اجازه جابه‌جایی برگشته و دوباره شروط جابه‌جایی بررسی شده و دستور جابه‌جایی مربوطه صادر خواهد شد. این روند تا رسیدن به ترتیب مورد نظر و در حالت بیشینه همزمانی انجام خواهد شد. اگر سیگنال اتمام طبقه‌بندی دریافت شود به حالت بعد منتقل خواهد شد.

- غیرفعال‌سازی واحد طبقه‌بندی زنبورها: پس از اتمام روند طبقه‌بندی، باید واحدهای صدور اجازه جابه‌جایی غیرفعال شوند. در این حالت و در یک کلاک پالس سیگنال غیرفعال‌سازی ارسال می‌شود.
- جستجوی بهینه‌سازی محلی: در این حالت سیگنال فعال‌سازی به واحدهای جستجوی بهینه-سازی محلی ارسال خواهد شد. بهینه‌سازی محلی به روش 2-opt انجام خواهد شد که در مورد آن در قسمت‌های قبلی توضیح داده شده است. واحد بهینه‌سازی محلی در این روش برای هر بار بررسی امکان بهینه‌سازی، به اطلاعات ۴ شهر نیاز دارد. اطلاعات این ۴ شهر در دو مرحله و در هر مرحله اطلاعات ۲ شهر از زنبور متناظر دریافت خواهد شد. به همین دلیل در این حالت زنبورها در وضعیت ارسال محتوای آدرس روی ورودی‌ها به خروجی‌ها قرار داده خواهند شد که امکان ارسال اطلاعات شهرها را به واحد بروزرسانی داشته باشند. در این حالت همچنین اتصالات ورودی و خروجی لازم بین واحد بهینه‌سازی محلی و زنبور هم برقرار خواهد شد. در صورتی که کار جستجوی جواب بهینه محلی برای تمامی زنبورها به اتمام رسیده باشد، کنترل سیستم به مرحله بعد انتقال خواهد یافت. لازم به ذکر است که هر یک از واحدهای بهینه‌سازی محلی، به صورت متوالی و به تعداد دفعات مشخص شده عمل بهینه‌سازی محلی را انجام خواهد داد و در نهایت بهترین جواب پیدا شده از اولین تکرار تا آخرین تکرار را بر روی خروجی‌های خود قرار خواهد داد. این معادل اختصاص تعدادی زنبور کارگر به زنبور تماشاگر و انتخاب حریصانه بین جواب آن‌ها است.
- بررسی نتایج بهینه‌سازی محلی به منظور نمایش: اگر واحدهای بهینه‌سازی محلی به نتایج بهتری دست یافته باشند، در این مرحله زنبورهای متناظر آن‌ها علامت‌گذاری می‌شوند که در صورت تمایل نتایج آن‌ها را بتوان در روی خروجی‌ها یا شبیه‌ساز مشاهده کرد.
- بروزرسانی زنبورها: در این حالت سیگنال فعال‌سازی به واحدهای بروزرسانی ارسال خواهد شد. همچنین زنبورها در وضعیت عملیات جابه‌جایی محتوای آدرس ورودی‌ها (تعویض اطلاعات دو خانه حافظه) قرار خواهند گرفت و ارتباطات لازم برای عملیات بروزرسانی بین

- واحد بروزرسانی، واحد بهینه‌سازی و زنبور برقرار خواهد شد. اگر واحدهای بهینه‌سازی محلی به نتایج بهتری دست پیدا کرده باشند، واحدهای بروزرسانی متناظر آن‌ها مقادیر مورد نظر برای تغییرات در مسیر زنبور عسل متناظر را تولید خواهند کرد. این مقادیر به همراه سیگنال فعال‌سازی به زنبور متناظر ارسال خواهد شد و به دلیل اینکه زنبورها در حالت جابه‌جایی دو شهر هستند، دو شهر مورد نظر با هم تعویض خواهند شد. این روند تعویض شهرها تا انتهای بروزرسانی هر زنبور ادامه خواهد داشت. با در نظر گرفتن اینکه انتخاب چهار شهر در روش opt-2 به صورت تصادفی است، تعداد دفعات عملیات تعویض برای هر زنبور با زنبور دیگر متفاوت بوده و در نتیجه دریافت سیگنال اتمام بروزرسانی زنبورها همزمان نخواهد بود. به علاوه، در این مرحله سیگنال اتمام عملیات واحدهای بروزرسانی هم بررسی شده و در صورت اتمام بروزرسانی همه زنبورها، به مرحله بعدی منتقل خواهد شد.
- بررسی شرایط نمایش نتایج: در این مرحله دو شرط بررسی می‌شود. اول اینکه آیا واحد بهینه‌سازی محلی زنبور با بهترین جواب به نتایج بهتری دست یافته است یا خیر؟ دوم اینکه آیا الگوریتم به انتهای خود رسیده است یا خیر؟ در صورت تحصیل هر یک از این شرایط، یعنی برای مشاهده اطلاعات زنبور با بهترین جواب و یا جواب نهایی مسئله بر روی خروجی‌ها، سیگنال فعال‌سازی به آن زنبور ارسال خواهد شد و سپس به مرحله تغییر نوع عملیات زنبور با بهترین جواب منتقل می‌شود. در صورتی که این شرایط برقرار نباشد، به مرحله بررسی تکرار بعدی و لزوم جایگزینی زنبور آخر انتقال می‌یابد.
 - تغییر نوع عملیات زنبور با بهترین جواب: در این حالت نوع عملکرد زنبور به ارسال اطلاعات ذخیره شده به خروجی‌ها تغییر خواهد کرد. همانطور که در قبل هم دیده شد، یک کلک پالس بعد از سیگنال فعال‌سازی نوع عملیات درخواستی روی زنبور بررسی می‌شود و در کلک پالس بعدی عملیات درخواستی روی زنبور شروع خواهد شد. پس در این مرحله نوع عملیات مشخص شده و به مرحله بعد منتقل خواهد شد.

- نمایش نتایج حاصل شده بر روی خروجی‌ها: در این مرحله نتایج بهترین زنبور (زنبور اول) بر روی خروجی قرار خواهد گرفت تا که بتوان پیشرفت کار و یا نتیجه نهایی را مشاهده کرد. نمایش این اطلاعات بر روی خروجی‌ها در هر بار گذراز این مرحله به تعداد شهرها تاخیر ایجاد می‌کند که به منظور بدست آوردن زمان دقیق حل مسئله فروشنده دوره‌گرد توسط طرح پیشنهادی می‌توان از فرمول ۱-۳ استفاده کرد:

$$\text{معادله ۱-۳} = \text{زمان اجرای الگوریتم}$$

طول یک کلک پالس \times تعداد دفعات نمایش اطلاعات \times تعداد شهرها - کل زمان اجرا

- بررسی تکرار بعدی و لزوم جایگزینی زنبور آخر: در این الگوریتم، سه نوع تکرار در نظر گرفته شده است. اولین نوع تکرار، تکرار داخلی واحد بهینه‌سازی محلی است که در نتیجه آن بهترین جواب از بین تمام تکرارها در روی خروجی‌های واحد بهینه‌سازی محلی قرار خواهد گرفت. این تکرارها معادل ne بهینه‌سازی محلی توسط زنبورهای کارگر در اطراف زنبور تماشاگر و انتخاب حریصانه از میان جواب‌های حاصله است. دومین نوع تکرار، تعداد دفعات رخداد مراحل طبقه‌بندی، بهینه‌سازی و بروزرسانی است که پس از آن زنبور با بدترین جواب باید با مقادیر اولیه جدید جایگزین شود. این نوع تکرار معادل ns تکرار برای جایگزینی زنبور پیشاهنگ است. تکرار سوم هم تعداد دفعات تکرار کل فرآیند الگوریتم از ابتدا تا انتها است. در نتیجه تعداد دفعات کل تکرار الگوریتم، حاصل ضرب این سه نوع تکرار است. در این مرحله تکرار نوع دوم و سوم بررسی می‌شود. اگر تکرار نوع دوم به انتها نرسیده باشد، یعنی لزومی به جایگزینی زنبور آخر (پیشاهنگ) نباشد، به مرحله طبقه‌بندی انتقال می‌یابد تا تمام مراحل طبقه‌بندی، بهینه‌سازی محلی و بروزرسانی برای هر یک از زنبورها دوباره طی شود. در صورتی که نوع تکرار دوم به مقدار مورد نظر برای بارگذاری مقدار اولیه جدید روی زنبور آخر رسیده باشد و هنوز مسیر نزدیکترین همسایگی بررسی نشده‌ای باقی مانده باشد، حالت مدار به مرحله‌ای منتقل خواهد شد که اتمام کار هسته تولید مسیر نزدیکترین همسایگی را بررسی کرده و در صورت اتمام این مرحله دستور ارسال و

ذخیره این مقادیر بر روی زنیور آخر داده خواهد شد و در نتیجه تمامی مراحل از ابتدا تا انتهای پیموده خواهد شد. لازم به اشاره است که زمان اجرای هسته نزدیکترین همسایگی برای تولید یک مسیر جدید بیشتر از مراحل دیگر است. به همین دلیل تعداد تکرارها به گونه‌ای در نظر گرفته می‌شود که مرحله تولید مسیر جدید به پایان رسیده باشد؛ که بدین صورت کل هسته‌ها همواره در حال کار بوده و زمان بیکاری یا انتظار نخواهند داشت. در صورتی که الگوریتم به انتهای تکرارهای کل رسیده باشد، هسته به حالت اعلام اتمام عملیات انتقال می‌یابد.

- اعلام اتمام عملیات: در این حالت سیگنال اتمام عملیات به خروجی متناظر ارسال خواهد شد و این سیگنال تا زمان بازنشانی سیگنال فعال‌سازی هسته اصلی بر روی خروجی باقی خواهد ماند. پس از بازنشانی سیگنال فعال‌سازی، به مرحله بیکاری منتقل خواهد شد و منتظر دستور فعال‌سازی مجدد خواهد ماند.

حال به توضیح قسمت‌های اصلی و زیرقسمت‌های متناظر آنها در این پیاده‌سازی خواهیم پرداخت.

۲-۳-۳ واحد تولیدکننده مسیر نزدیکترین همسایگی

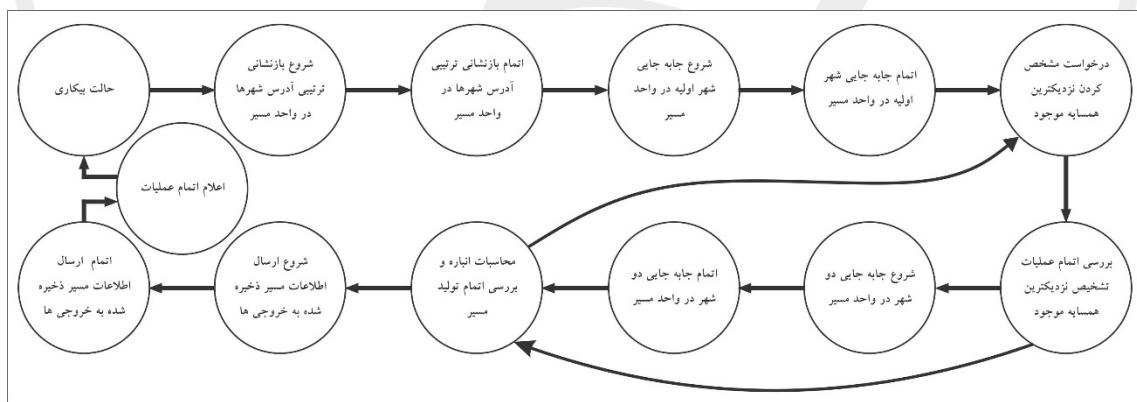
این واحد وظیفه ساخت مسیر نزدیکترین همسایگی را که در بخش پیاده‌سازی نرم‌افزاری توضیح آن داده شد را به عهده دارد. چگونگی انجام آن نیز بدین سان است که با فعال‌سازی واحد و پس از دریافت سیگنال مربوط به ساخت مسیر جدید در روی ورودی NEXT-TOUR-REQUEST از شهر دریافت سیگنال ارسال مسیر و مقدار مسافت مربوطه می‌ماند. بعد از دریافت سیگنال ارسال، متضطر دریافت سیگنال ارسال مسیر و اطلاعات مسیر حرکت، آدرس مسیر در رم و مسافت طی خروجی DONE را به صفر تغییر داده و اطلاعات مسیر حرکت، آدرس مسیر در رم و مسافت طی شده را به ترتیب بر روی خروجی‌های TOUR-LENGTH و INDEX-OUT و TOUR-OUT قرار می‌دهد و سپس به مدت یک کلاک پالس خروجی DONE را یک کرده و متضطر دریافت سیگنال ساخت مسیر جدید می‌ماند. در صورت دریافت این سیگنال شهر دوم را به عنوان شهر بعدی در نظر گرفته و مسیر نزدیکترین همسایگی را برای آن ساخته و مراحل قبل را دوباره طی می‌کند. سپس برای

شهر سوم و الی شهر آخر. در صورتی که به شهر نهایی رسیدیم، دوباره به شهر اول برگشته و از شهر اول، مسیر نزدیکترین همسایگی را تولید خواهد کرد. در شکل ۱۰-۳ ورودی و خروجی‌های این واحد ترسیم شده است.



شکل ۱۰-۳ واحد تولیدکننده مسیر نزدیکترین همسایگی

در شکل ۱۱-۳ نمودار FSM این واحد مشاهده می‌شود، که در آن مراحل مختلف کنترل این واحد آمده است. در این نمودار داریم:



شکل ۱۱-۳ نمودار ماشین حالت واحد تولید مسیر نزدیکترین همسایگی

- حالت بیکاری: در این حالت اعمال بازنشانی، بررسی سیگنال فعال‌سازی و سیگنال مربوط به ساخت مسیر بعدی بررسی می‌شود. در صورتی که واحد فعال شده باشد و سیگنال ساخت مسیر بعد نیز دریافت شده باشد، به حالت بعد خواهد رفت. همچنین در این مرحله مقادیر

شمارندها بازنشانی شده و مقدار شهر اول مسیر نیز تحقیق شده که اگر به ماکسیمم تعداد شهرها رسیده باشد، به آدرس شهر اول تغییر می‌یابد.

- شروع بازنشانی ترتیبی آدرس شهرها در واحد مسیر: واحد مسیر حرکت از واحد نزدیکترین همسایه، را در وضعیت بازنشانی ترتیبی آدرس شهرها قرار داده و سیگنال فعال‌سازی به این واحد ارسال می‌کند. این کار باعث خواهد شد که واحد مسیر حرکت، از مقدار اول تا آخرین شهر را به ترتیب بر روی خود بازنویسی کند. به عبارتی مسیر ذخیره شده قبلی بر روی خود را پاک کرده و با مقادیر اولیه و به ترتیب شهر اول تا آخر جایگزین می‌کند.
- اتمام بازنشانی ترتیبی آدرس شهرها در واحد مسیر: سیگنال خروجی واحد نزدیکترین همسایه بررسی شده که در صورت اتمام بازنویسی مقادیر اولیه به مرحله بعد منتقل شود.
- شروع جابه‌جایی شهر اولیه در واحد مسیر: در این حالت واحد مسیر از واحد نزدیکترین همسایه را در حالت جابه‌جایی محتوای آدرس ورودی‌ها (دو شهر) قرار داده و شهر منتخب برای جایگاه اولین شهر را مشخص کرده و سیگنال فعال‌سازی به واحد مسیر داده می‌شود تا که این شهر در جایگاه اول مسیر قرار گیرد.
- اتمام جابه‌جایی شهر اولیه در واحد مسیر: سیگنال اتمام عملیات واحد نزدیکترین همسایه بررسی شده که یک شدن آن به معنی اتمام عمل جابه‌جایی و انتقال شهر مورد نظر به جایگاه اول مسیر است.
- درخواست مشخص کردن نزدیکترین همسایه موجود: در این مرحله واحد مسیر حرکت از واحد نزدیکترین همسایه در وضعیت ارسال محتوای آدرس روی ورودی‌ها به خروجی‌ها قرار خواهد گرفت. همچنین واحد شمارنده از واحد نزدیکترین همسایه را فعال کرده که نتیجه آن تشخیص نزدیکترین همسایه بعدی در مسیر خواهد بود. در واقع این مرحله شروع تمامی تکرارها تا ساخت کامل مسیر نزدیکترین همسایگی است که با دادن سیگنال شروع پیدا کردن همسایه بعدی آغاز می‌شود.

- بررسی تمام عملیات تشخیص نزدیکترین همسایه موجود: اگر عملیات تشخیص به اتمام رسیده باشد، واحد نزدیکترین همسایه خروجی DONE خود را به مدت یک کلاک پالس یک خواهد کرد. با دریافت این سیگنال در این مرحله، بر اساس اینکه الگوریتم در کجای روند تولید مسیر نزدیکترین همسایگی باشد، تصمیم گرفته خواهد شد که به حالت جابه-جایی دو شهر برود یا به حالت محاسبات انباره. این شرط از آنروスト که بعد از انتخاب شهر یکی مانده به آخر، برای شهر بعد هیچ انتخابی وجود ندارد و تنها فاصله شهر یکی مانده به آخر و شهر آخر باید از واحد نزدیکترین همسایه دریافت شود و نیازی به هیچگونه جابه-جایی در مسیر تولید شده نیست. همچنین برای اضافه کردن فاصله شهر اول و آخر هم از همین روش استفاده شده و تنها از واحد نزدیکترین همسایه فاصله این دو شهر دریافت شده و سپس به مرحله محاسبات انباره منتقل خواهد شد. به بیان دیگر، تنها دو بار از این مرحله به مرحله محاسبات انباره جهش می‌شود. این جهش در دو تکرار آخر و برای شهر یکی مانده به آخر و شهر آخر و همچنین شهر اول و آخر است. در باقی تکرارها به مرحله بعدی یعنی شروع جابه‌جایی دو شهر در واحد مسیر انتقال خواهد یافت.
- شروع جابه‌جایی دو شهر در واحد مسیر: حال که نزدیکترین شهر به شهر حاضر یعنی شهری که محاسبات الگوریتم برای آن انجام شده است، مشخص شده است؛ باید جای آن با شهر بعدی در توالی مسیر تغییر یابد. در بخش‌های بعد توضیح داده خواهد شد ولی در اینجا باید به این نکته اشاره شود که واحد مسیر، از رم توزیع شده برای ذخیره‌سازی توالی مسیر حرکت فروشنده دوره‌گرد استفاده می‌کند و شماره هر یک از شهرها در یک آدرس رم ذخیره می‌شود. پس به منظور جابه‌جایی تنها کافیست که محتوای آدرس شهر منتخب و شهر بعدی در توالی مسیر عوض شود. به عنوان مثال؛ اگر شهر شماره ۲ اولین شهر در مسیر و شهر منتخب بعدی شهر شماره ۳۳ باشد، پس شهر ۳۳ را باید به آدرس بعدی شهر ۲ در رم منتقل کرد و شهری که در آن آدرس قرار دارد را باید به آدرس شهر ۳۳ در رم انتقال داد. با این روش همواره تمام شهرهای موجود یا کاندید برای نزدیکترین همسایه در آدرس‌های بعدی

شهر حال حاضر در رم قرار خواهند داشت. بدین منظور در این حالت، واحد مسیر حرکت از واحد نزدیکترین همسایه در وضعیت جابه‌جایی محتوای آدرس ورودی‌ها قرار داده می‌شود. آدرس دو شهری که باید با هم تعویض شوند، روی ورودی‌های واحد نزدیکترین همسایه قرار خواهد گرفت و سیگنال فعال‌سازی به واحد مسیر حرکت از واحد نزدیکترین همسایه ارسال خواهد شد.

- تمام جابه‌جایی دو شهر در واحد مسیر: در این مرحله سیگنال خروجی واحد نزدیکترین همسایه بررسی شده تا که تمام عملیات جابه‌جایی مشخص شود.
- محاسبات انباره و بررسی تمام تولید مسیر: در این حالت برای مسیر نزدیکترین همسایگی که تاکنون محاسبه شده، مسافت طی شده محاسبه می‌شود. همچنین در این مرحله تعداد شهرهای مسیر در حال ساخته بررسی می‌شود و در صورتی که مسیر هنوز کامل نشده باشد به حالت درخواست مشخص کردن نزدیکترین همسایه موجود انتقال خواهد یافت. به دلیل اینکه برای شهر یکی مانده به آخر مسیر، هیچ انتخاب دیگری جز شهر آخر نیست یا به عبارتی تنها همسایه موجود شهر یکی مانده به آخر همان شهر آخر است، پس الگوریتم تا آنجا پیش می‌رود که نزدیکترین همسایه به شهر دو تا مانده به آخر را مشخص کند. بدین شکل مسیر کامل محاسبه شده است و فقط مسافت مسیر کامل هنوز محاسبه نشده است. بدین منظور با دوبار انتقال به حالت درخواست مشخص کردن نزدیکترین همسایه موجود، چشم‌پوشی از دو مرحله مربوط به جابه‌جایی و بازگشت مجدد به این حالت، مسافت بین دو شهر یکی مانده به آخر و آخر و دو شهر آخر و اول هم به این مسافت اضافه می‌شود. در نتیجه مسافت حاصل شده، مسافت کامل حرکت فروشنده دوره‌گرد پس از بازگشت به نقطه شروع حرکت است. اگر هم که مسیر کامل شده باشد و مسافت کامل مسیر هم محاسبه شده باشد، به حالت بعدی رفته و منتظر دریافت سیگنال مربوط به ارسال مسیر ذخیره شده خواهد ماند.

- شروع ارسال اطلاعات مسیر ذخیره شده به خروجی‌ها: در صورت عدم دریافت سیگنال ارسال، مقدار خروجی DONE را به یک تغییر داده، در غیراینصورت خروجی DONE را صفر کرده، وضعیت واحد مسیر از واحد نزدیکترین همسایه را به ارسال اطلاعات ذخیره شده به خروجی‌ها تغییر داده و سیگنال فعال‌سازی را به واحد مسیر ارسال می‌کند. بدین صورت در دو کلک پالس بعد از دریافت سیگنال ارسال، مسیر به همراه آدرس آن‌ها در رم و همچنین مقدار تابع شایستگی متناظر بر روی خروجی‌ها قرار خواهد گرفت. این اطلاعات مسیر و آدرس آن‌ها در رم به صورت سری و با هر کلک پالس یک مقدار بر روی خروجی‌ها قرار خواهد گرفت.
- اتمام ارسال اطلاعات مسیر ذخیره شده به خروجی‌ها: در این مرحله اتمام کار واحد نزدیکترین همسایه بررسی شده و در صورتی که روند ارسال مسیر به اتمام رسیده باشد، به مرحله اعلام اتمام عملیات منتقل خواهد شد. همچنین آدرس شهر اولیه بعدی نیز محاسبه می‌شود.
- اعلام اتمام عملیات: در این حالت خروجی DONE به مدت یک کلک پالس یک خواهد شد و سپس به مرحله بیکاری بازگشته و منتظر سیگنال ساخت مسیر بعدی خواهد ماند.

۱-۲-۳-۳ واحد تشخیص نزدیکترین همسایه

این واحد وظیفه مشخص کردن نزدیکترین همسایه به شهر حال حاضر که قبلًاً به عنوان قسمتی از مسیر انتخاب نشده است را برعهده دارد. همچنین با قرار دادن ورودی و خروجی‌های مناسب به هسته‌های خارج از این واحد اجازه دسترسی به واحد مسیر حرکت داده شده است که امکان تغییر در واحد مسیر حرکت را می‌دهد. این نوع دسترسی از ایجاد چندین واحد مسیر حرکت، افزایش حجم سخت‌افزاری و مشکلات مربوط به نقل و انتقالات محتوای این واحدها به هم جلوگیری به عمل می‌آورد. در شکل ۱۲-۳ نمای خارجی هسته تشخیص نزدیکترین همسایه آورده شده است.

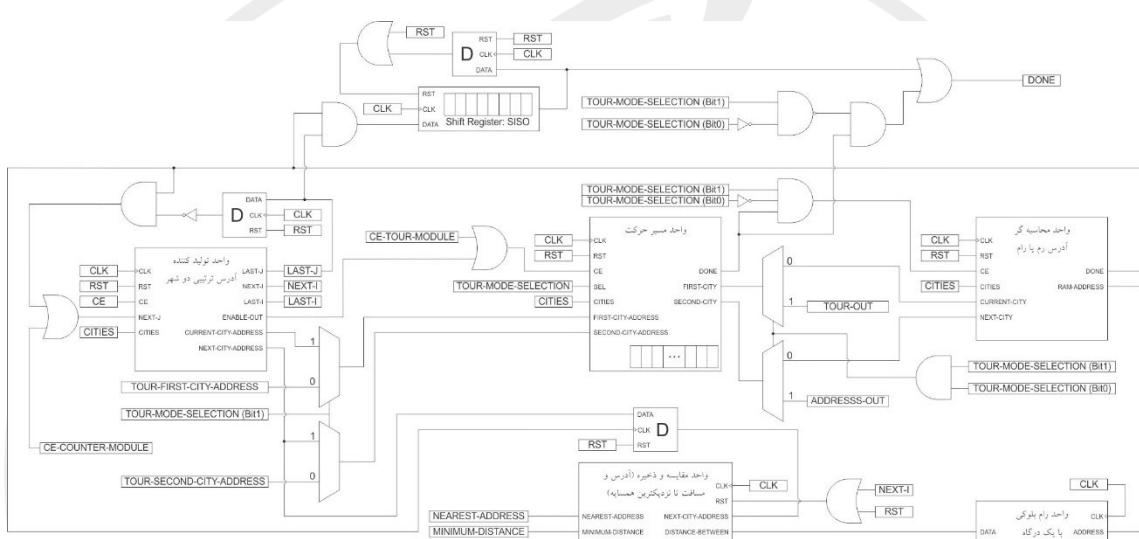


شکل ۱۲-۳ واحد تشخیص نزدیکترین همسایه

در این واحد با استفاده از ورودی CE-TOUR-MODULE امکان ارسال سیگنال فعالسازی به واحد مسیر حرکت وجود داشته و با استفاده از ورودی TOUR-MODE-SELECTION می‌توان حالت یا وضعیت واحد مسیر حرکت را مشخص کرد. همچنین در هر یک از دو مرحله جابه‌جایی در واحد تولید مسیر نزدیکترین همسایگی یعنی جابه‌جایی شهر اولیه یا جابه‌جایی دو شهر در طول مسیر با استفاده از دو ورودی TOUR-SECOND-CITY-ADDRESS و TOUR-FIRST-CITY-ADDRESS می‌توان آدرس دو شهر مورد نظر را به واحد مسیر ارسال کرد. به عبارتی با استفاده از این چهار ورودی امکان دسترسی کامل به واحد مسیر حرکت داده شده است. همچنین دو خروجی ADDRESS و INDEX-OUT برای ارسال اطلاعات واحد مسیر حرکت به خارج از هسته در نظر گرفته شده است. ورودی CE-COUNTER-MODULE به منظور درخواست تشخیص نزدیکترین شهر بعدی به کار می‌رود که سیگنال فعالسازی واحد تولیدکننده آدرس ترتیبی دو شهر نیز هست. سه خروجی LAST-I، NEXT-I، LAST-J و NEAREST-ADDRESS هم مقادیر آدرس نزدیکترین همسایه و مقدار مسافت از شهر حاضر تا آن را مشخص می‌کنند. البته این خروجی‌ها در تمام مدت محاسبات دارای مقدار بوده و تنها

باید در انتهای تشخیص نزدیکترین همسایه و بعد از دریافت سیگنال از خروجی DONE خوانده شوند.

طراحی این واحد در سطح ساختار انجام گرفته و به جای استفاده از ماشین حالت از سطح گیت برای کنترل این واحد استفاده شده است که حجم سخت افزاری کمتری نسبت به کنترل در سطح رفتار یا به عبارتی ماشین حالت اشغال می کند. نمای کلی این ساختار در شکل ۱۳-۳ آورده شده است.



شکل ۱۳-۳ نمای کلی ساختار واحد تشخیص نزدیکترین همسایه

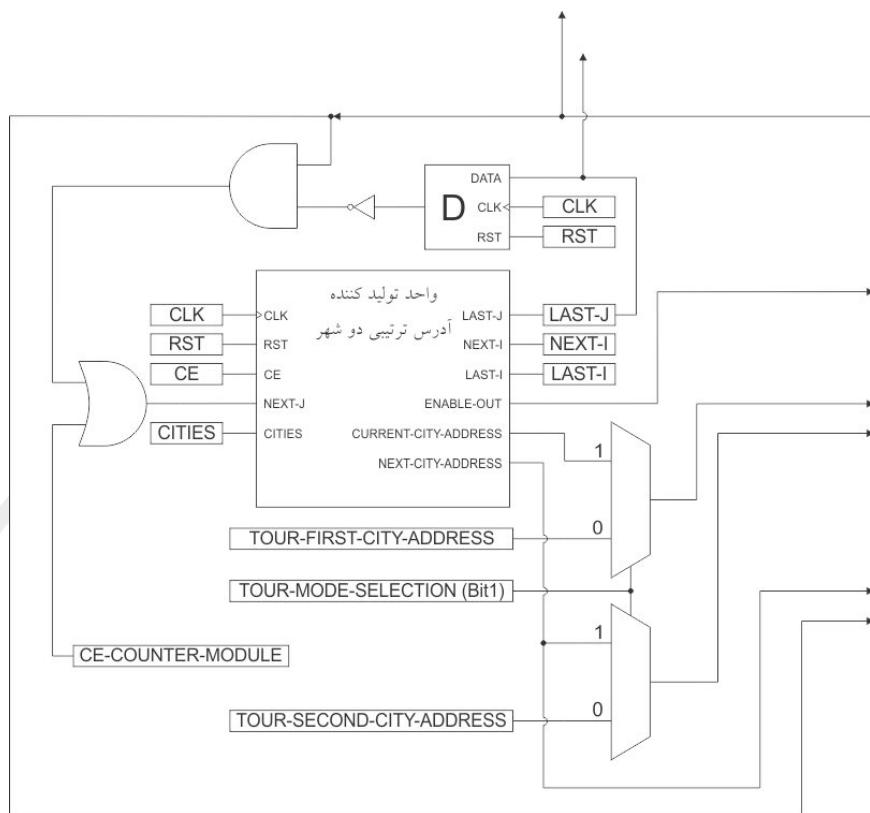
بدون در نظر گرفتن قسمت های کنترلی، زیر واحدهای ادامه در این ساختار استفاده شده اند:

- واحد تولید کننده آدرس ترتیبی دو شهر
- واحد مسیر حرکت
- واحد محاسبه گر آدرس رم^۱ یا رام
- واحد رام
- واحد محاسبه و ذخیره (آدرس و مسافت تا نزدیکترین همسایه)

در ادامه به توضیح این ساختار خواهیم پرداخت و نحوه کنترل قسمت های متفاوت آن را بیان خواهیم کرد.

¹ RAM (Random-Access Memory)

² ROM (Read-Only Memory)



شکل ۱۴-۳ قسمت اول از ساختار تشخیص نزدیکترین همسایه

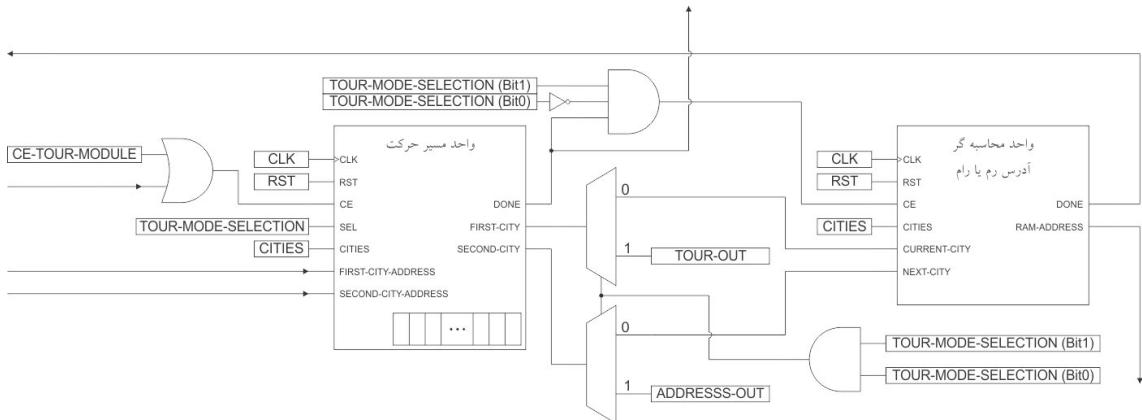
در شکل ۱۴-۳ واحد تولیدکننده آدرس ترتیبی دو شهر با قسمت‌های کنترلی آن آمده است. همانطور که مشاهده می‌شود ورودی CE^۱ واحد تشخیص نزدیکترین همسایه مستقیماً به ورودی CE این CE-COUNTER-MODULE به واحد متصل شده است. همچنین مشاهده می‌شود که ورودی محاسبه‌گر آدرس رم همراه سیگنال دیگری که تلفیقی از سیگنال LAST-J و سیگنال DONE واحد محاسبه‌گر آدرس رم یا RAM است، به ورودی شهر بعدی این واحد داده شده است. با استفاده از ورودی مذکور می‌توان شروع یک جستجوی جدید حول شهر حال حاضر را برای شهرهای موجود تحریک کرد و با استفاده از سیگنال تلفیقی، دستور تولید آدرس شهر بعدی موجود به این واحد داده خواهد شد. سیگنال دوم به صورت خودکار در هر کدام از جستجوهای نزدیکترین همسایه در داخل خود هسته تولید می‌شود. در قسمت‌های بعدی توضیح خواهیم داد که واحد تولیدکننده آدرس ترتیبی دو شهر مقادیر ترتیبی از دو آدرس یعنی آدرس شهر حال حاضر و شهر بعدی مورد بررسی را تولید خواهد کرد و منتظر

^۱ Chip Enable

سیگنال برای تولید دو آدرس بعدی خواهد ماند؛ یعنی به عنوان مثال، با در نظر گرفتن شهر اول با آدرس صفرم، شهر بعدی مقادیر یک را خواهد داشت، حال پس از دریافت سیگنال مذکور مقدار آدرس شهر بعدی به دو تبدیل خواهد شد، همانطور بعد از سیگنال بعدی آدرس شهر بعدی به سه تغییر خواهد کرد، الى تعداد شهرها منهای یک. در انتهای این روند، آدرس شهر حال حاضر یکی اضافه شده و شهر بعدی به ترتیب از آدرس شهر حال حاضر به اضافه یک تا آخرین شهر منهای یک تغییر خواهد کرد. به عبارتی در هر جستجوی نزدیکترین همسایه تنها مقدار یک شهر تغییر کرده و شهری که مقدار آدرس آن ثابت باقی مانده شهر حال حاضر مسیر است. حال با فرض اینکه شهر حال حاضر را شهر A و شهر Mورد بررسی بعدی را Zام در نظر بگیریم، این واحد در انتهای هر بار محاسبه آدرس رام نیاز به دریافت یک سیگنال داشته تا آدرس شهر بعدی را یک واحد افزایش دهد. ولی تنها زمانی که آخرین شهر موجود را بررسی می‌کند نباید این سیگنال داده شود؛ چراکه مرحله بعدی جستجو با شهر حال حاضر در آدرس بعدی رم شروع خواهد شد و این مشخصه مورد نظر از این هسته نیست؛ زیرا که قبل از جستجوی بعدی باید مراحل جابه‌جایی ذکر شده در قبل صورت پذیرد. همانطور که قبلاً بیان شد واحد تشخیص نزدیکترین همسایه برای جستجوی مجدد منتظر دریافت سیگنال از طرف ورودی CE-COUNTER-MODULE باقی می‌ماند. از فلیپ‌فلاب D به منظور ایجاد یک کلک پالس تاخیر استفاده شده است، چراکه در زمان بررسی آخرین شهر از مجموعه شهرهای موجود، اگر این تاخیر وجود نداشته باشد خروجی ENABLE-OUT سیگنالی مبنی بر مجاز بودن شهر آخر ایجاد نخواهد کرد.

دو مالتی‌پلکسر موجود در شکل ۱۴-۳ برای آنست که ورودی واحد مسیر حرکت به ورودی‌های واحد نزدیکترین همسایه متصل شود یا به خروجی‌های واحد تولیدکننده آدرس ترتیبی دو شهر. در شکل دیده می‌شود که سیگنال کنترلی این دو مالتی‌پلکسر بیت دوم سیگنال دو بیتی TOUR- MODE-SELECTION است که بیانگر آنست که در حالت ارسال محتوای آدرس روی ورودی‌ها به خروجی‌ها به واحد تولیدکننده آدرس ترتیبی دو شهر متصل خواهد شد و در حالت جابه‌جایی

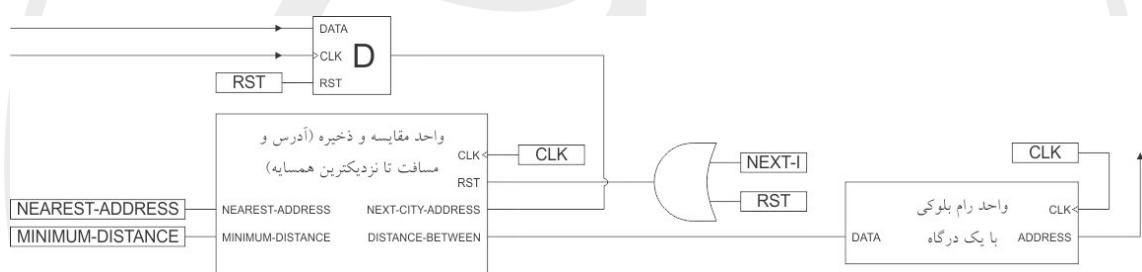
محتوای آدرس ورودی‌ها به ورودی‌های واحد متصل خواهد بود. حالت‌های متفاوت واحد مسیر حرکت در قسمت‌های بعدی توضیح داده خواهد شد.



شکل ۳-۱۵ قسمت دوم از ساختار تشخیص نزدیکترین همسایه

در شکل ۳-۱۵ دو واحد مسیر حرکت و واحد محاسبه‌گر آدرس رم یا رام به همراه سیگنال‌های کنترلی آن‌ها نمایش داده شده است. به منظور فعال‌سازی واحد مسیر حرکت از OR دو سیگنال CE-TOUR-MODULE واحد تولیدکننده آدرس ترتیبی دو شهر و ورودی ENABLE-OUT استفاده شده است. سیگنال اول زمانی تولید شده که آدرس شهر بعدی بزرگتر از آدرس شهر حال حاضر باشد؛ به عبارتی در محدوده مجاز آدرس‌دهی باشد و سیگنال دوم از ورودی واحد و به دلیل دسترسی بیشتر به این واحد از خارج از واحد دریافت می‌شود. ورودی دو بیتی حالت یا وضعیت واحد مسیر حرکت، مستقیماً از ورودی‌های واحد اعمال شده و همچنین با استفاده از بیت صفرم و اول این ورودی و دو دی‌مالتی‌پلکسرا مشخص می‌شود که خروجی‌های این واحد به خروجی‌های واحد تشخیص نزدیکترین همسایه متصل شوند یا که به واحد محاسبه‌گر آدرس رم یا رام متصل شوند. این اتصال به گونه‌ای است که فقط در حالت ارسال اطلاعات ذخیره شده به خروجی‌ها، خروجی‌های واحد مسیر حرکت به خروجی‌های اصلی متصل می‌شوند. همانطور که قبل ام بیان شد دو ورودی SECOND-CITY-ADDRESS و FIRST-CITY-ADDRESS از خروجی‌های مالتی-پلکسرا دریافت می‌شوند.

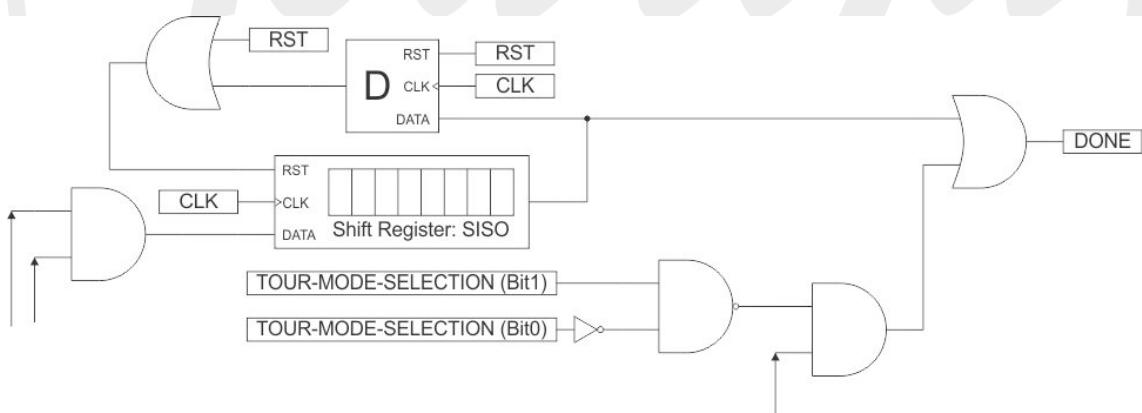
سیگنال فعالساز واحد محاسبه‌گر آدرس رم یا رام هم تنها زمانی مهیا می‌شود که واحد مسیر حرکت در وضعیت ارسال محتوای آدرس روی ورودی‌ها به خروجی‌ها قرار داشته باشد؛ یا به عبارتی در حال محاسبه نزدیکترین همسایه باشد و واحد مسیر حرکت اطلاعات دو شهر مورد نظر را در خروجی خود قرار داده باشد؛ یا به وصفی سیگنال اتمام خواندن محتوای آدرس‌ها را ارسال کرده باشد. از دو سیگنال اتمام عملیات دو واحد محاسبه‌گر آدرس رم یا رام و واحد مسیر حرکت برای ایجاد خروجی DONE واحد استفاده شده است. همچنین همانطور که در قبل توضیح داده شد، سیگنال DONE واحد محاسبه‌گر آدرس رم یا رام خود یکی از سیگنال‌های مورد استفاده به منظور کنترل واحد تولیدکننده آدرس ترتیبی دو شهر است. با در نظر گرفتن شکل کامل ساختار مشخص می‌شود که خروجی RAM-ADDRESS به ورودی واحد رام متصل است.



شکل ۳-۱۶ قسمت سوم از ساختار تشخیص نزدیکترین همسایه

شکل ۳-۱۶ واحد رام که بر روی رم بلوکی FPGA نگاشت می‌شود و واحد مقایسه و ذخیره (آدرس و مسافت تا نزدیکترین همسایه) را به همراه سیگنال‌های کنترلی آن‌ها نمایش می‌دهد. همانطور که دیده می‌شود، خروجی‌های واحد مقایسه و ذخیره مستقیماً به خروجی واحد تشخیص نزدیکترین همسایه متصل شده است. در صورتی که این اطلاعات بعد از سیگنال اتمام اتمام عملیات واحد مذکور خوانده شود، مقادیر نهایی را در خود خواهد داشت. در غیراینصورت آدرس و مسافت طی شده در آن تا آخرین وضعیت محاسبات خواهد بود؛ به عبارتی در این حالت تمامی شهرهای موجود هنوز بررسی نشده‌اند. به منظور محاسبه درست مقادیر آدرس نزدیکترین همسایه و فاصله متناظر آن تا شهر حال حاضر، این واحد در هر بار درخواست جدید تشخیص نزدیکترین همسایه بازنشانی شده، تا مقادیر آدرس و فاصله‌های قدیمی ذخیره شده پاک شده و مقادیر اولیه خود را اختیار کنند که بیشینه

ممکن برای دقت مورد نظر است؛ یعنی تمام بیت‌ها با مقدار یک به جای صفر بارگذاری می‌شوند. سیگنال اتمام عملیات برای بازن Shanی واحد مقایسه و ذخیره، از خروجی NEXT-I دریافت می‌شود. این سیگنال بعد از دریافت سیگنال شروع تشخیص نزدیکترین همسایه بعدی روی ورودی CE- COUNTER-MODULE واحد تولیدکننده آدرس ترتیبی دو شهر به عرض یک کلاک پالس یک خواهد بود که بیانگر شروع بررسی شهر جدیدی به عنوان شهر حال حاضر است. ورودی آدرس این واحد از خروجی شهر بعدی واحد تولیدکننده آدرس ترتیبی دو شهر با یک کلاک پالس تاخیر می‌آید؛ چراکه پس از قرار گرفتن آدرس روی ورودی رام، یک کلاک پالس طول خواهد کشید که محتوای رام بر روی خروجی قرار بگیرد و در این فاصله مقادیر شهر بعدی در خروجی واحد تولیدکننده آدرس ترتیبی دو شهر تغییر کرده است. پس این تاخیر به منظور همزمانی آدرس شهر بعدی و فاصله شهر حال حاضر تا این شهر ضروری است. در قسمت‌های بعدی توضیح داده خواهد شد که از رام برای ذخیره ماتریس فاصله‌ها استفاده شده است و با قرار دادن آدرس مورد نظر بر روی ورودی آن، خروجی این واحد فاصله دو شهر مورد نظر خواهد بود.



شکل ۱۷-۳ قسمت چهارم از ساختار تشخیص نزدیکترین همسایه

سیگنال DONE واحد تشخیص نزدیکترین همسایه در شکل ۱۷-۳ باید چندین پیغام را در خود داشته باشد:

- در ابتدا باید پس از اتمام هر جستجوی همسایگی یک سیگنال اتمام عملیات صادر کند، که این سیگنال باید در زمانی رخ دهد که آخرین شهر موجود بررسی شده باشد و این شرط

زمانی اتفاق می‌افتد که سیگنال LAST-J یک باشد؛ یعنی در حال بررسی آخرین شهر باشد و خروجی واحد محاسبه‌گر آدرس رم یا رام هم دریافت شده باشد؛ بدین معنی که شهر آخر هم بررسی شده است. تنها محدودیت آنست که در لحظه‌ای که خروجی محاسبه‌گر آدرس رم یا رام برای شهر یکی مانده به آخر ارسال می‌شود، به محض دریافت این خروجی در لبه بالارونده کلاک پالس، واحد تولیدکننده آدرس ترتیبی دو شهر خروجی J LAST را یک کرده و به عبارتی این دو سیگنال با هم تلاقی پیدا کرده، در حالی که هنوز آخرین شهر موجود بررسی نشده است. بعد از اینکه آخرین شهر هم در نظر گرفته شد، خروجی DONE واحد محاسبه‌گر آدرس رم یا رام دوباره یک پالس ارسال می‌کند که با توجه به اینکه خروجی J LAST یک باقی مانده است، این دو خروجی باز پالس دیگری تولید خواهد کرد، در واقع AND این دو واحد شامل دو پالس اتمام عملیات خواهد بود. به منظور حذف این پدیده از یک شیفت رجیستر ورود نوبتی، خروج نوبتی^۱ بهره گرفته شده است. با در نظر گرفتن طول مناسب برای این شیفت رجیستر، دو سیگنال تولیدی به طور کامل در داخل شیفت رجیستر قرار می‌گیرند و پس از آنکه اولین سیگنال روی خروجی شیفت رجیستر قرار گیرد، با تاخیر یک کلاک پالس محتویات شیفت رجیستر پاک می‌شود. در نتیجه با استفاده از این روش از مجموع دو سیگنال تولیدی تنها یک سیگنال با عرض یک کلاک پالس خواهیم داشت. فاصله لبے بالارونده این دو سیگنال ۶ کلاک پالس است و باید قبل از ایجاد خروجی هر دو سیگنال به طور کامل داخل شیفت رجیستر قرار بگیرند. در نتیجه به شیفت رجیستری با حداقل طول ۷ بیت نیاز هست. ولیکن اگر این تعداد بیت در نظر گرفته شود، سیگنال اتمام عملیات زودتر از موعد مورد نظر در روی خروجی ظاهر خواهد شد. زمانی که سیگنال اتمام عملیات واحد محاسبه‌گر رم یا رام ظاهر می‌شود، هنوز به یک کلاک پالس دیگر نیاز است که اطلاعات رام خوانده شود و آخرین مقایسه برای تعیین نزدیکترین همسایگی انجام شود. همچنین ترجیح بر اینست که خروجی DONE یک کلاک پالس بعد از اتمام عملیات یک

¹ Serial-In, Serial-Out (SISO)

شود. با توجه به این دو نکته و به دلیل کاهش حجم سخت‌افزار استفاده شده، حداقل طول ۹ بیت برای این شیفت رجیستر باید در نظر گرفته شود. واضح است که افزایش طول این شیفت رجیستر تنها در ایجاد سیگنال خروجی تاخیر ایجاد می‌کند. از فلیپ‌فلاب D به منظور ایجاد یک کلاک پالس تاخیر استفاده شده است و در صورت نیاز به طول بیشتر سیگنال DONE، باید شیفت رجیستر SISO دیگری را به جای آن در نظر گرفت. دقیق شود که اگر این تاخیر استفاده نمی‌شد، خروجی شیفت رجیستر به سرعت پس از یک شدن، صفر می‌شد؛ به عبارتی خروجی DONE فقط به اندازه تاخیر انتشار شیفت رجیستر یک می‌شد و نه به اندازه طول یک کلاک پالس.

• به دلیل اینکه در خارج از این واحد به واحد مسیر نیز دسترسی وجود دارد، باید در انتهای هر بار عملیات بازنمانی اولیه مسیر حرکت، جابه‌جایی یا ارسال اطلاعات به خارج از هسته، سیگنال اتمام عملیات بر روی خروجی ظاهر شود که همانند قبل، پالسی به عرض یک کلاک پالس خواهد بود. این سیگنال باید در زمانی ایجاد شود که واحد مسیر حرکت خروجی DONE مربوط به خود را یک کلاک پالس تحریک کرده باشد و همچنین در وضعیتی غیر از تشخیص نزدیکترین همسایه باشد. اگر در وضعیت تشخیص نزدیکترین همسایه باشد، با هر بار بررسی دو آدرس موجود روی ورودی‌های واحد مسیر حرکت یک کلاک پالس ایجاد خواهد شد، که مورد نظر نیست. گیت NOR تضمین می‌کند که تنها در حالت‌هایی غیر از این حالت، کلاک پالسی از خروجی DONE واحد مسیر حرکت به خروجی DONE واحد تشخیص نزدیکترین همسایگی ارسال خواهد شد.

۲-۲-۳-۳ واحد تولیدکننده آدرس ترتیبی دو شهر

در روند پیدا کردن مسیر نزدیکترین همسایگی، این واحد وظیفه تولید دو آدرس برای بررسی شهر حال حاضر و مجموعه شهرهای موجود دیگر را عهده‌دار است. در قبل بیان شد که برای مشخص شدن مسیر حرکت فروشنده، شماره شهرها بر روی یک ردیف یا صف چیده می‌شود. به اینصورت اولین شماره در صف، اولین شهر انتخابی برای شروع حرکت است و آخرین شهر روی صف، شهری

هست که قبل از پیمودن کامل مسیر و برگشت به شهر اول باید از آن گذر شود. به منظور اینکه در این پیاده‌سازی صفت مذکور، مجموعه شهرهای موجود برای آن و عملیات روی آن‌ها کمترین میزان حافظه یا حجم سخت‌افزاری را اشغال کند، این صفت و مجموع شهرهای موجود تلفیق شده است. بدین معنی که حافظه مجازی دیگری برای شهرهای موجود یا شهرهای طی نشده در نظر گرفته نشده است و تنها یک صفت وجود داشته و از این صفت اطلاعات لازم استخراج خواهد شد. ضمن اینکه به منظور پیاده‌سازی هرچه بهینه‌تر، این صفت بر روی رم توزیع شده نگاشت شده است. در ادامه نحوه استفاده و کارکرد این پیاده‌سازی شرح داده خواهد شد.

فرض می‌کنیم که در مسئله مورد بررسی n شهر داشته باشیم و در اولین قدم از تولید نزدیکترین مسیر همسایگی هم باشیم؛ یعنی در آدرس صفر از رم توزیع شده؛ جایی که صفت شهرها در آن ذخیره می‌شود. شهرهای موجود برای بررسی در آدرس‌های یک تا آدرس $n-1$ هستند، اگر شهر دوم مسیر حرکت را انتخاب کرده باشیم، پس در آدرس یکم از رم هستیم؛ در نتیجه مجموعه شهرهای موجود برای بررسی، شهرهای آدرس ۲ تا $n-1$ در رم هستند. به همین منوال تا آدرس $n-2$ که انتخابی جز آدرس $n-1$ برای آن وجود ندارد. با این روش نیاز به دو مجموعه که یکی شامل شهرهای طی شده و دیگری شامل شهرهای انتخاب نشده باشند، متفاوت شده و حافظه استفاده شده به نصف تقليق می‌یابد. ولی موردنی که باید در نظر گرفته شود، روشی نظاممند برای بررسی این آدرس‌ها است. واحد تولیدکننده آدرس ترتیبی دو شهر این وظیفه را تقبل کرده و این آدرس‌ها را بر روی خروجی‌های خود به همراه تعدادی سیگنال کنترلی قرار می‌دهد. یکی از این خروجی‌ها آدرس شهر حال حاضر (CURRENT-CITY-ADDRESS) و یکی دیگر آدرس شهر بعدی در صفت یا شهر بعدی موجود (NEXT-CITY-ADDRESS) است. به عنوان مثال؛ با فرض وجود ۵ شهر، نحوه شمارش و خروجی‌های تولید شده این واحد در شکل ۱۸-۳ نمایش داده شده است.

NEXT-CITY-ADDRESS:

1	2	3	4	2	3	4	3	4	4	0
---	---	---	---	---	---	---	---	---	---	---

CURRENT-CITY-ADDRESS:

0	1	2	3	4
---	---	---	---	---

شکل ۱۸-۳ نحوه شمارش آدرس‌های رم برای تولید مسیر نزدیکترین همسایگی در سطح رفتار

همانطور که در شکل ۱۸-۳ دیده می‌شود، اول آدرس صفرم و تمامی آدرس‌های بعد از آن در رم در نظر گرفته می‌شود. بعد از مشخص شدن شهر بعدی در آدرس یکم، آدرس متناظر به یک افزایش یافته و بقیه شهرهای موجود که شامل شهرها در آدرس ۲ تا ۴ است بر روی خروجی قرار خواهد گرفت. همینطور تا آدرس شهر یکی مانده به آخر یعنی ۳ و شهر آخر یعنی آدرس ۴. بعد از این آدرس‌دهی نیاز به تولید آدرسی برای در نظر گرفتن فاصله شهر آخر و شهر اول در محاسبات است که این مقدار با قرار دادن مقدار آدرس ۴ بر روی خروجی مربوط به شهر حال حاضر و آدرس صفر بر روی خروجی آدرس شهر بعدی ایجاد می‌شود. این نحوه شمارش مربوط به پیاده‌سازی در سطح رفتار است. اگر بخواهیم در سطح ساختار این واحد را پیاده‌سازی کنیم، نحوه شمارش کمی متفاوت خواهد بود که در شکل ۱۹-۳ این نوع شمارش ترسیم شده است.

NEXT-CITY-ADDRESS:	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0
CURRENT-CITY-ADDRESS:	0	1	2	3	4											

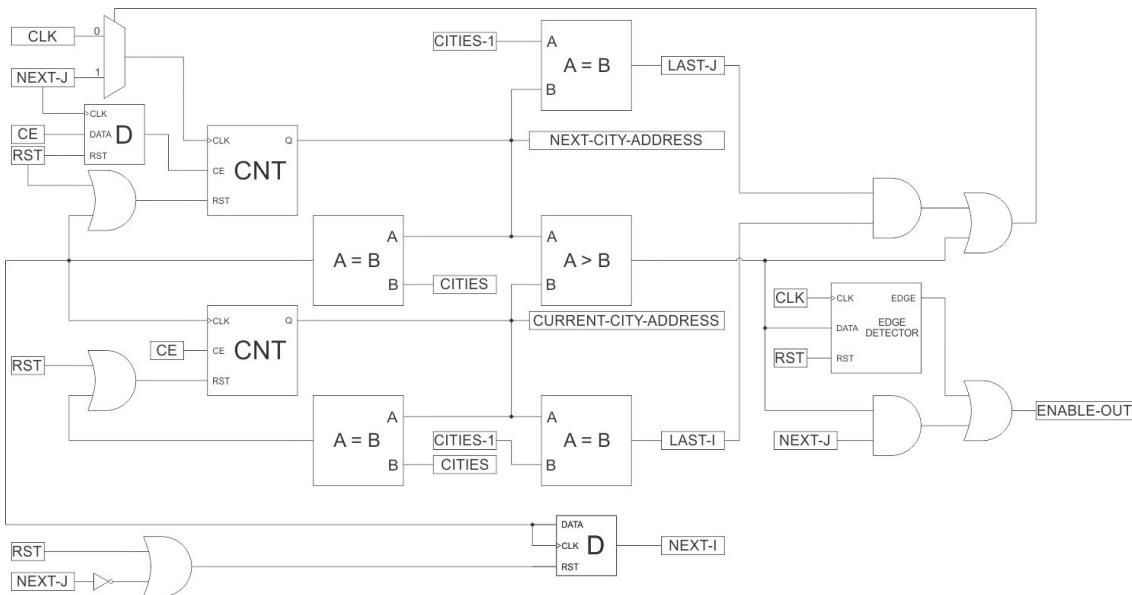
شکل ۱۹-۳ نحوه شمارش آدرس‌های رم برای تولید مسیر نزدیکترین همسایگی در سطح ساختار همانطور که دیده می‌شود نحوه شمارش در سطح ساختار همانند سطح رفتار است با این تفاوت که بین مقادیر مجاز آدرس‌ها بر روی خروجی NEXT-CITY-ADDRESS شماره‌های میانی یا غیرمجاز هم با هر کلاک پالس قرار داده می‌شوند که با وجود خروجی‌های کنترلی مقادیر مجاز به راحتی قابل تشخیص خواهند بود. در شکل ۲۰-۳ واحد تولیدکننده آدرس ترتیبی دو شهر به همراه ورودی و خروجی‌های آن نمایش داده شده است.



شکل ۲۰-۳ واحد تولیدکننده آدرس ترتیبی دو شهر

این واحد در ورودی CITIES تعداد شهرها را که همان n است دریافت کرده و شمارش را شروع می‌کند. به منظور دریافت مقادیر مجاز بعدی بر روی خروجی‌ها باید ورودی J-NEXT تحریک شود (لبه بالارونده یک سیگنال کنترلی). دلیل وجود این ورودی آنست که شروع انجام محاسبات و درخواست بعدی برای دو آدرس جدید، باید از خارج از این هسته کنترل شود؛ چراکه زمان انجام آن بیش از یک کلاک پالس است و باید در کنترل هسته بزرگتری باشد که این هسته زیرمجموعه آن هسته قرار بگیرد. همانطور که گفته شد آدرس شهر حال حاضر بر روی خروجی CURRENT-CITY-ADDRESS و آدرس مجموعه شهرهای انتخاب نشده بر روی خروجی NEXT-CITY-ADDRESS قرار خواهد گرفت. به منظور اطلاع از شرایط و جایگاه آدرس دهی چندین خروجی کنترلی در نظر گرفته شده است. خروجی اول مربوط به ENABLE-OUT است که در زمانی که خروجی‌ها در محدوده مجاز هستند، به مدت یک کلاک پالس مقدار یک را اتخاذ می‌کند. این مقادیر مجاز زمانی است که آدرس شهر حال حاضر کوچکتر از آدرس شهر بعدی باشد و یا آدرس شهر اول و آخر در نظر گرفته شود. همانطور که از دو شکل ۱۸-۳ و ۱۹-۳ پیداست، این خروجی در پیاده‌سازی در سطح ساختار این خروجی در مواردی که مجاز شمرده می‌شود، دارای پالس خروجی خواهد بود. سیگنال LAST-I در زمان بررسی آخرین آدرس شهر حال حاضر یعنی مورد شهر آخر و اول یک خواهد شد و تا انتهای بررسی در همان وضعیت باقی خواهد ماند. خروجی NEXT-I در ابتدای بررسی هر شهر حال حاضر جدید مقدار یک را به مدت یک کلاک پالس بر روی خود قرار داده و سپس صفر می‌شود. خروجی LAST-J هم در هنگام بررسی آخرین عنصر از مجموعه شهرهای موجود یک شده و همانند خروجی LAST-I تا انتهای بررسی این آدرس یک باقی خواهد ماند. به عنوان مثال در مورد ۵ شهر، زمانی که آدرس شهر بعدی ۴ باشد یا شهر اول و آخر در حال بررسی باشد.

در شکل ۲۱-۳ نحوه پیاده‌سازی این واحد در سطح ساختار بدون شمارش شهر اول و آخر به تصویر کشیده شده است.



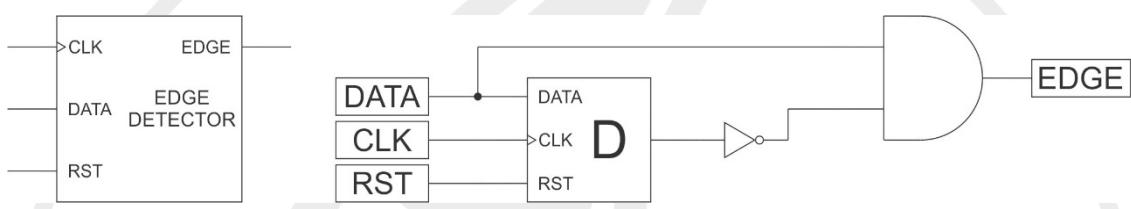
شکل ۲۱-۳ پیاده‌سازی واحد تولیدکننده آدرس ترتیبی دو شهر بدون برگشت در سطح ساختار

این طرح با استفاده از دو شمارنده^۱، یک آشکارساز لبه، پنج واحد مقایسه‌گر، دو فلیپ‌فلاپ D، یک مالتی‌پلکسر دو به یک و هشت گیت منطقی پیاده شده است. در ابتدای هر سری شمارش شهر بعدی، ورودی انتخاب‌گر مالتی‌پلکسر صفر بوده، پس ورودی کلاک شمارنده بالایی به کلاک پالس اصلی FPGA متصل است، این همان قسمتی است که شماره آدرس‌های شهر بعدی به سرعت و هر یک در یک کلاک پالس شمرده شده تا به مقادیر مجاز برسد؛ این حالت در ابتدای هر آدرس جدید برای شهر حال حاضر تکرار خواهد شد. پس از شمارش از صفر تا مقدار مجاز اول و به محض اینکه یک مقدار مجاز مشاهده شود، سیگنال خروجی تنها مقایسه‌گر عدد بزرگتر یک شده، ورودی شمارنده بالایی به ورودی J NEXT-J واحد متصل شده و منتظر دریافت سیگنال از آن ورودی برای آدرس‌های بعدی باقی خواهد ماند. خروجی شمارنده بالایی، همان خروجی NEXT-CITY-ADDRESS است. همچنین خروجی این شمارنده به دو واحد مقایسه‌گر تساوی هم متصل شده که یکی برای مشخص کردن سیگنال کنترلی LAST-J است و دیگری برای زمانی است که یک دور شمارش کامل آدرس شهرهای موجود به اتمام رسیده است، یا به عبارتی مجموعه شهرهای بررسی نشده، تهی است و زمان انتقال به آدرس شهر حال حاضر جدید است. در این حالت خروجی این مقایسه‌کننده تساوی،

¹ COUNTER (CNT)

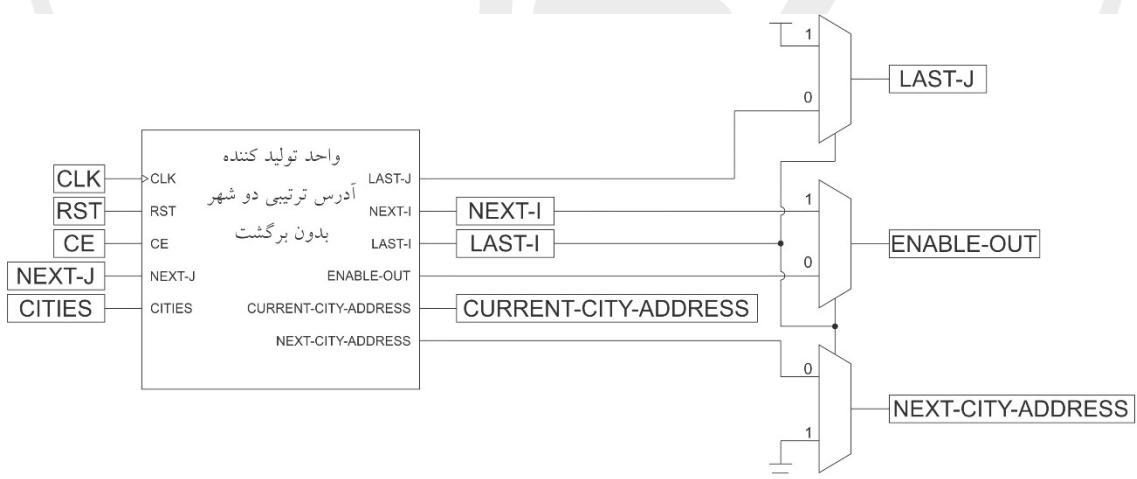
ورودی کلاک پالس شمارنده پایینی که شمارنده آدرس شهر حال حاضر است را تحریک کرده و باعث افزایش یک واحدی این شمارنده می‌شود. همچنین شمارنده بالایی را بازن Shanai کرده که آدرس‌های بیشتر از حد نیاز را نشمارد و خروجی I-NEXT را هم به مدت یک کلاک پالس یک خواهد کرد. حال شمارنده پایینی به مقدار بعدی رفته و شمارنده بالایی از مقدار صفر شروع کرده تا که با در نظر گرفتن مقدار جدید آدرس شهر حال حاضر به مقدار مجاز برسد. خروجی شمارنده پایینی به دو مقایسه‌گرتساول متصل است که یکی برای بازن Shanai این شمارنده بوده و اجازه شمارش بیشتر از حد نیاز را به این واحد نمی‌دهد. مقایسه‌گر دیگر برای سیگنال خروجی I-LAST به کار برده شده است. همانطور که از شکل ۲۱-۳ پیداست، خروجی کنترلی ENABLE-OUT از AND دو سیگنال J-NEXT و خروجی مقایسه‌گر مقدار بزرگتر ساخته می‌شود؛ پس در ابتدای هر شمارش مجاز به طول سیگنال J-NEXT یک خواهد بود و تنها محدودیت آن در اولین مقدار مجاز است. با فرض اینکه طول سیگنال J-NEXT محدود بوده و در این پیاده‌سازی یک کلاک پالس در نظر گرفته شده است، زمانی که مقدار سیگنال خروجی این مقایسه‌گر یک شود، دیگر اثری از سیگنال J-NEXT وجود ندارد که AND منطقی این دو مقدار خروجی یک را نتیجه بدهد. فرض کنیم بررسی آخرین شهر هم تمام شده است و هسته متظر دریافت ورودی J-NEXT بوده که مقدار شمارنده بالایی را بازن Shanai کرده و مقدار خروجی شمارنده پایینی را یک واحد افزایش دهد. در این لحظه هنوز سیگنال خروجی مقایسه‌گر مقدار بزرگتر یک است. با دریافت ورودی J-NEXT این سیگنال داخلی صفر شده و ورودی کلاک شمارنده بالایی به کلاک اصلی سیستم متصل می‌شود، حال شمارش شروع شده و در طی این شمارش سیگنال J-NEXT به طور کامل از بین می‌رود. این شمارش تا جایی ادامه خواهد یافت که به مقدار مجاز اول بررسیم؛ به عنوان مثال، آدرس ۴ بر روی خروجی شهر بعدی و آدرس ۳ بر روی خروجی شهر حال حاضر. در این لحظه مقدار سیگنال خروجی مقایسه‌گر مذکور یک شده، ولی دیگر اثری از ورودی J-NEXT نیست. بنابراین در این شرایط، از لبۀ بالارونده سیگنال خروجی این مقایسه‌گر برای ایجاد پالسی بر روی خروجی استفاده شده است. در شکل ۲۲-۳ واحد لبۀ یاب به همراه ساختار درونی آن که در این پیاده‌سازی به کار رفته است، مشاهده می‌شود. این طرح

یکی از روش‌های معروف برای مشخص کردن لب بالارونده یک سیگنال است. حداکثر عرض این سیگنال نزدیک به یک کلاک پالس است و آن زمانی است که لب بالارونده سیگنال مورد بررسی درست بعد از لب بالارونده کلاک پالس اعمال شود. همچنین هرچه لب بالارونده سیگنال از لب بالارونده کلاک پالس فاصله بگیرد و به لب بالارونده کلاک پالس بعدی نزدیک‌تر شود، طول سیگنال تولید شده کوتاه‌تر خواهد بود؛ تا جایی که اگر همزمان با لب بالارونده کلاک پالس بعدی باشد، طول این سیگنال تنها به اندازه تاخیر انتشار فلیپ‌فلاب و گیت نقیض خواهد بود.



شکل ۲۲-۳ واحد آشکارساز لب

حال با اضافه کردن دو شهر اول و آخر به مجموعه آدرس‌های تولید شده، این ساختار کامل می‌شود. بدین منظور از سه مالتی‌پلکسر برای خروجی‌های LAST_J، NEXT_CITY_ADDRESS و LAST_I استفاده شده است. این ساختار در شکل ۲۳-۳ نمایش داده شده است.



شکل ۲۳-۳ پیاده‌سازی واحد تولیدکننده آدرس ترتیبی دو شهر در سطح ساختار

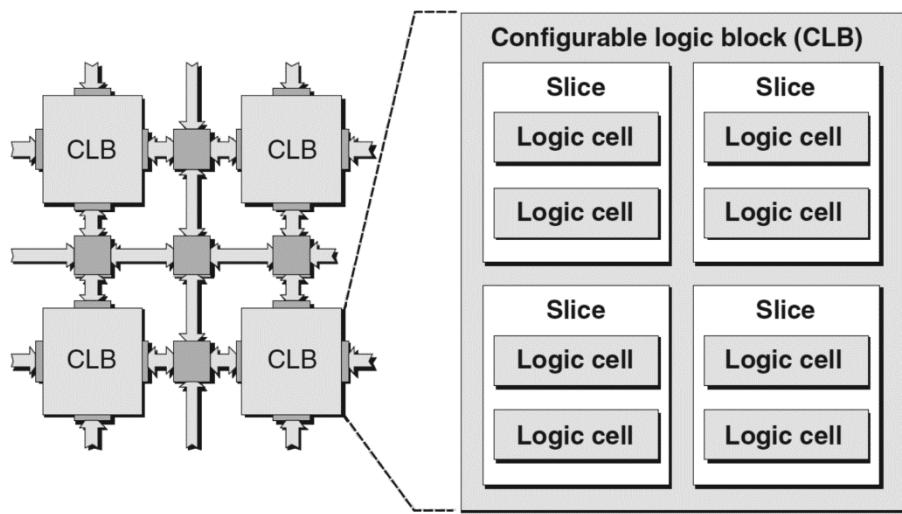
۳-۲-۳ واحد مسیر حرکت

در این واحد مسیر حرکت فروشنده دوره‌گرد ذخیره شده است و با قرار دادن حالت‌های متفاوت برای این واحد، امکان تغییرات در مسیر ذخیره شده فراهم شده است. همانطور که در قبل هم بیان شد، این مسیر بر روی رم توزیع شده ذخیره می‌شود. به منظور افزایش کارایی این واحد، رم در نظر گرفته شده به صورت دو درگاه پیاده‌سازی شده است که امکان بیشینه بهره‌برداری از این واحد را مهیا می‌کند.

۳-۲-۳-۱ رم توزیع شده

برخی از عرضه‌کنندگان امکان استفاده از سلول‌های تشکیل دهنده‌ی LUT به صورت یک بلوک کوچک RAM را فراهم می‌آورند. برای مثال، ۱۶ سلول تشکیل دهنده یک LUT چهار ورودی می‌تواند در نقش یک 16×1 عمل کند. به این قابلیت، RAM توزیع شده گفته می‌شود؛ زیرا LUT‌ها در سطح تراشه توزیع می‌شوند و به این ترتیب از بلوک‌های RAM بزرگ‌تر متمایز می‌شوند. در پیکربندی چهاربرشی CLB شکل ۲۴-۳، تمام LUT‌های درون CLB را می‌توان برای پیاده‌سازی‌های زیر با هم پیکربندی کرد:

۸×۱۶ RAM بیت تک درگاهه
۴×۳۲ RAM بیت تک درگاهه
۲×۶۴ RAM بیت تک درگاهه
۱×۱۲۸ RAM بیت تک درگاهه
۴×۱۶ RAM بیت دو درگاهه
۲×۳۲ RAM بیت دو درگاهه
۱×۶۴ RAM بیت دو درگاهه (ماکسفیلد، ۱۳۸۹).



شکل ۲۴-۳ یک CLB حاوی چهار برش (تعداد برش‌ها به خانواده FPGA وابسته است)

۲-۳-۲-۳-۳ واحد رم توزیع شده

در شکل ۲۵-۳ ورودی و خروجی‌های واحد رم توزیع شده نشان داده شده است. خواندن محتوای ADDRESS A این رم به صورت همزمان بوده؛ یعنی زمانی که آدرس جدیدی بر روی ورودی‌های A و B قرار گیرد، محتوای آن آدرس‌ها پس از لب بالاروند کلاک پالس، روی خروجی‌ها قرار خواهد گرفت. به منظور نوشتن محتوا بر روی این واحد، آدرس مورد نظر را بر روی ورودی ADDRESS A و محتوا را بر روی ورودی DATA-IN قرار داده و سپس ورودی WE^۱ را یک کرده و بدین ترتیب در لب بالاروند کلاک پالس بعدی این مقدار در آدرس مشخص شده ذخیره خواهد شد.

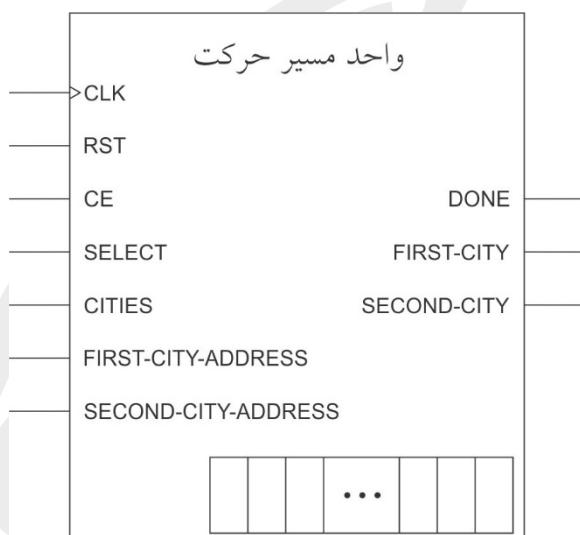


شکل ۲۵-۳ واحد رم توزیع شده با دو درگاه

^۱ Write Enable

۳-۳-۲-۳-۳ حالت‌های واحد مسیر حرکت

نمای کلی این واحد در شکل ۲۶-۳ آورده شده است. از ورودی CE برای فعال‌سازی و شروع عملیات مورد نظر بر روی رم استفاده شده و با استفاده از ورودی SELECT می‌توان نوع این عملیات را مشخص کرد. همچنین یک کلاک پالس بعد از اتمام هر عملیاتی، خروجی DONE به مدت یک کلاک پالس یک خواهد بود.



شکل ۲۶-۳ واحد مسیر حرکت

این عملیات شامل حالت‌های زیر است:

- بازنشانی ترتیبی آدرس شهرها: دو کلاک پالس بعد از دریافت سیگنال فعال‌سازی و انتخاب این حالت، شماره شهرها به ترتیب از یک تا مقدار n (تعداد شهرهای موجود در مسئله مورد بررسی) بر روی رم و در آدرس صفر تا $n-1$ چیده می‌شود؛ به عبارتی این مسیر بازنشانی می‌شود. بعد از پرسه بازنشانی، خروجی DONE در کلاک پالس بعدی و به مدت یک کلاک پالس یک خواهد بود.

- جابه‌جایی محتوای آدرس ورودی‌ها: در این حالت هسته مسیر حرکت، دو آدرس موجود بر روی ورودی‌های خود را خوانده و در صورتی که به یک آدرس یکسان اشاره نکند، محتوای این دو آدرس را با هم جابه‌جا می‌کند. به منظور جابه‌جایی نیاز به یک بافر هست که

محتوای یکی از این آدرس‌ها را در طول جابه‌جایی در خود نگه دارد. به منظور بهره گرفتن به صورت بهینه از این واحد، همواره آدرس n از رم توزیع شده به عنوان این بافر استفاده می‌شود. توجه شود که کلیه هسته‌ها به صورت نگاشت عمومی^۱ پیاده‌سازی شده‌اند؛ در نتیجه می‌توان از هسته اصلی^۲ تغییرات عمدہ‌ای ایجاد کرد. از جمله این تغییرات، طول رم توزیع شده است. در هر حال پیاده‌سازی به صورت نگاشت عمومی نیاز به مقدار اولیه داشته و اینجا طول اولیه این رم ۲۵۶ خانه در نظر گرفته شده است. با این توضیحات درمی‌یابیم که بدون تغییرات در واحد هسته اصلی می‌توان از مسائلی حداکثر با ۲۵۵ شهر استفاده کرد، چراکه خانه آخر رم به بافر اختصاص یافته است. به علاوه نرم‌افزار سیستم‌اسایزر XST در شرایطی که تعداد شهرها کمتر از تعداد خانه‌های در نظر گرفته شده برای رم باشد، آن را به صورت بهینه پیاده‌سازی کرده و خانه‌هایی که مورد استفاده قرار گرفته نمی‌شوند را به سخت‌افزار تبدیل نخواهد کرد. عملیات جابه‌جایی در سه کلک پالس انجام خواهد شد و خروجی DONE که نشان از اتمام عملیات دارد، در کلک پالس بعدی یک خواهد بود. دو کلک پالس هم برای خواندن سیگنال فعال‌سازی و بررسی نوع عملکرد انتخابی نیاز است. در نتیجه تمام عملیات جابه‌جایی شش کلک پالس به طول خواهد انجامید.

- ارسال محتوای آدرس روی ورودی‌ها به خروجی‌ها: در این حالت پس از دریافت سیگنال فعال‌سازی، آدرس‌های موجود بر روی ورودی‌های واحد خوانده شده و محتوای متناظر این آدرس‌ها در رم بر روی خروجی‌ها قرار داده می‌شوند. لازم به ذکر است که به منظور تنظیم زمانی در هسته تشخیص نزدیکترین همسایه، باید این محتوا تا شرایط بازنشانی یا سیگنال فعال‌سازی بعدی بر روی خروجی‌ها باقی بماند. به همین دلیل این حالت با سه حالت دیگر متفاوت است و حتی حالت اعلام اتمام عملیات مخصوص به خود را دارد. این عملیات سه کلک پالس به طول خواهد انجامید و در کلک پالس بعد سیگنال اتمام عملیات بر روی خروجی DONE ظاهر خواهد شد.

¹ Generic

² Top Module

• ارسال اطلاعات ذخیره شده به خروجی‌ها: دو کلک پالس پس از ارسال سیگنال فعالسازی و انتخاب این حالت، محتوای رم بر روی خروجی FIRST-CITY و آدرس آن بر روی خروجی SECOND-CITY قرار داده می‌شود. شروع این آدرس‌ها و محتوای منتظر آنها از آدرس صفرم بوده و تا آدرس $n-1$ ادامه خواهد داشت. پس از آنکه محتوای آخرین آدرس بر روی خروجی قرار گرفت، در کلک پالس بعدی سیگنال اتمام عملیات که به عبارتی همان اتمام ارسال اطلاعات هست، بر روی خروجی DONE قرار خواهد گرفت.

در شکل ۲۷-۳ نمودار ماشین حالت این واحد آورده شده است. این واحد پس از دریافت سیگنال فعالسازی، از حالت بیکاری به حالت بررسی نوع عملیات رفته و بسته به عملیات انتخاب شده به یکی از حالت‌های موجود انتقال می‌یابد. حالت‌هایی که تا اتمام انجام عملیات تغییر حالت نداده و تنها تکرار می‌شوند، پیکان برگشت به خود دارند. یک کلک پالس پس از اتمام هر عملیات، خروجی DONE برای تایید اتمام عملیات یک خواهد شد. جز در یکی از حالت‌ها، مرحله بعدی حالت بیکاری خواهد بود که در این حالت الگوریتم دوباره منتظر سیگنال فعالسازی بعد باقی خواهد ماند تا که با توجه به حالت انتخاب شده، عملیات مورد نظر را دوباره آغاز کند. ولیکن در یک حالت، پس از دریافت سیگنال فعالسازی مستقیماً به مرحله بررسی نوع عملیات منتقل خواهد شد.

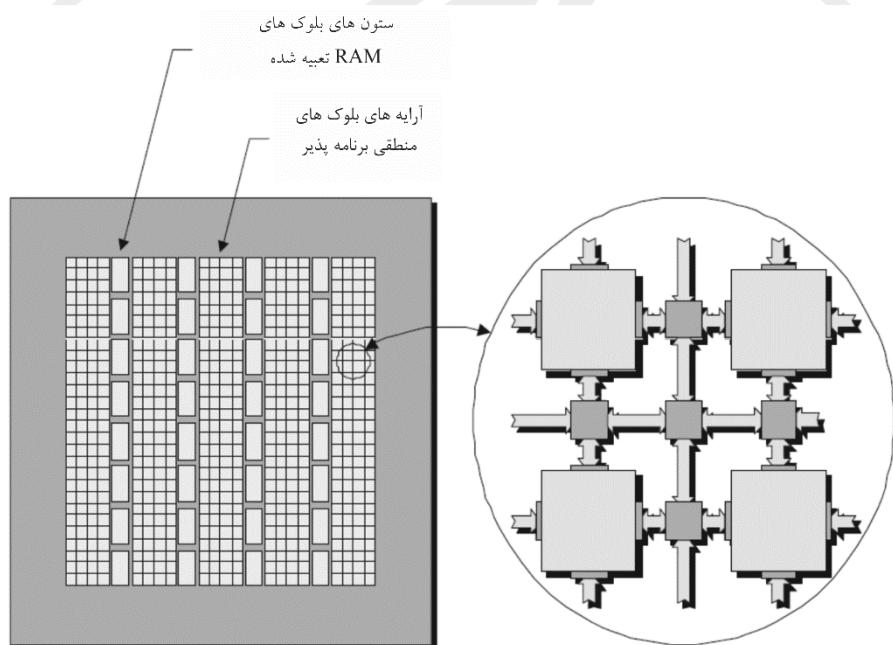


شکل ۲۷-۳ نمودار ماشین حالت واحد مسیر حرکت

۴-۲-۳-۳ واحد رام بلوکی با یک درگاه

۱-۴-۲-۳-۳ RAMهای بلوکی^۱ یا RAMهای تعییه شده

در بسیاری از کاربردها به حافظه نیاز است، از اینرو FPGAهای امروزی مقادیر نسبتاً زیادی RAM تعییه شده به نام e-RAM یا RAM بلوکی دارند. بسته به معماری قطعه، این بلوک‌ها می‌توانند در کناره‌های قطعه قرار گیرند، بر روی تراشه پراکنده شوند، یا مانند شکل ۲۸-۳ در چند ستون سازماندهی شوند.



شکل ۲۸-۳ نمای تراشه با چند ستون از بلوک‌های RAM تعییه شده

چنین بلوکی، بسته به قطعه می‌تواند چند هزار یا چند ده هزار بیت را نگهداری کند. علاوه بر این، هر قطعه می‌تواند چند صد بلوک RAM این چنینی داشته باشد، در نتیجه در کل ظرفیت ذخیره-سازی چند صد هزار بیت تا چندین میلیون بیت داشته باشد.

هر بلوک RAM را می‌توان به صورت جداگانه استفاده کرده و یا چندین بلوک را برای پیاده‌سازی بلوک‌های بزرگ‌تر با هم ترکیب کرد. از این بلوک‌ها برای مقاصد مختلفی از قبیل پیاده‌سازی RAMهای تک یا دو درگاهه استاندارد، توابع FIFO^۲، ماشین‌های حالت و غیره استفاده می‌شود (ماکسفیلد، ۱۳۸۹).

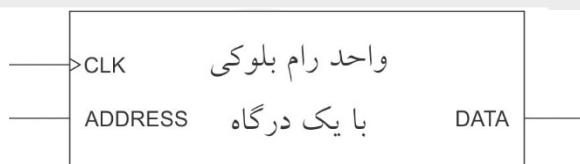
¹ Block RAM (Read–Access Memory)

² First-In, First-Out

یک RAM بلوکی را می‌توان یک SRAM سریع که به وسیله یک واسط همزمان آرایش پذیر احاطه شده تصور نمود. هر بلوک RAM از $16K$ (۲^{۱۴}) بیت داده به علاوه $2K$ بیت توازن تشکیل شده است. می‌توان آن را با عرض‌های متفاوتی از $1 \times 16K$ (یعنی $2^0 \times 2^{14}$) یا 32×512 (یعنی $2^5 \times 2^9$) سازماندهی نمود. (پونگ، ۱۳۹۰). به عنوان مثال بر روی تراشه VIRTEX5 مدل XC5VLX330 به میزان ۱۰۳۶۸ کیلوبیت فضای رم بلوکی در دسترس است.

۲-۴-۲-۳-۳ واحد رام بلوکی استفاده شده

همانطور که می‌توان از رم‌های تعبیه شده به صورت رم‌های بلوکی استفاده کرد، می‌توان این رم‌ها را نیز به صورت ROM پیاده‌سازی کرد. تنها تفاوت این دو پیاده‌سازی بر روی FPGA آن است که امکان نوشتن بر روی رام وجود نداشته و تنها می‌توان اطلاعات داخل آن را خواند. در شکل ۲۹-۳ واحد رام پیاده‌سازی شده مشاهده می‌شود. این واحد با هر لبۀ بالاروندۀ کلاک پالس، محتوای آدرس ورودی را بر روی خروجی قرار خواهد داد. پر واضح است که اگر محاسبات در لبۀ پایین‌رونده صورت پذیرد و آدرس بر روی ورودی واحد قرار بگیرد، تا لبۀ بالاروندۀ کلاک پالس بعدی خروجی جدید را دریافت نخواهیم کرد.



شکل ۲۹-۳ واحد رام بلوکی با یک درگاه

۳-۴-۲-۳-۳ اطلاعات مسائل

حل هر مسئله‌ای احتیاج به اطلاعاتی از شرایط، محدودیت‌ها و پارامترهای اولیه داشته که این اطلاعات برای هر مسئله متفاوت است. در مورد مسئله فروشنده دوره‌گرد این اطلاعات به صورت مختصات طول و عرض هر یک از شهرهای مسئله، ارائه شده است. به منظور استفاده از این مقادیر دو رویکرد وجود داشته:

- استفاده مستقیم از مختصات طول و عرض شهرها: در این حالت با داشتن مختصات طول و عرض و با استفاده از فاصله اقلیدسی فاصله دو شهر محاسبه می‌شود.

$$D(T[i], T[j]) = \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2}$$

معادله ۲-۳ فاصله اقلیدسی

در شکل ۳۰-۳ دو بردار مختصات شهرها برای مسئله فرضی با n شهر به همراه شماره شهرها مشاهده می‌شود.

X Coordinate:	X1 X2 X3 X4 X5 X6 X7 X8 ... Xn	TRAVEL SALES PERSON DATABASE
Y Coordinate:	Y1 Y2 Y3 Y4 Y5 Y6 Y7 Y8 ... Yn	
CITY ADDRESS:	1 2 3 4 5 6 7 8 ... n	TOUR

شکل ۳۰-۳ اطلاعات یک مسئله فرضی برای مسئله فروشنده دوره‌گرد

- استفاده از ماتریس فاصله‌ها: اگر تعداد دفعات محاسبه فاصله اقلیدسی زیاد باشد، بهتر است که یکبار این فواصل را برای تمامی شهرها محاسبه کرد و در یک ماتریس قرار داد و در هر بار نیاز به فاصله دو شهر، به این ماتریس رجوع شود و مقدار متناظر خوانده شود. این ماتریس مربعی بوده و هر یک از ابعاد آن به تعداد شهرهاست (n). به منظور بدست آوردن فاصله دو شهر کافیست شماره شهر اول و دوم را سطر و ستون این ماتریس در نظر گرفته و محل تلاقی این سطر و ستون حاوی فاصله این دو شهر است. این ماتریس حول قطر اصلی خود متقارن بوده و تمام عناصر سطر اصلی آن صفر است؛ چراکه فاصله هر شهر از خود صفر است. در شکل ۳۱-۳ ماتریس فاصله‌ها مشاهده می‌شود:

CITY ADDRESS		1 2 3 4 5 6 7 8 ... n									
		1	2	3	4	5	6	7	8	...	n
1		0	D ₁₂	D ₁₃	D ₁₄	D ₁₅	D ₁₆	D ₁₇	D ₁₈	...	D _{1n}
2		D ₂₁	0	D ₂₃	D ₂₄	D ₂₅	D ₂₆	D ₂₇	D ₂₈	...	D _{2n}
3		D ₃₁	D ₃₂	0	D ₃₄	D ₃₅	D ₃₆	D ₃₇	D ₃₈	...	D _{3n}
4		D ₄₁	D ₄₂	D ₄₃	0	D ₄₅	D ₄₆	D ₄₇	D ₄₈	...	D _{4n}
5		D ₅₁	D ₅₂	D ₅₃	D ₅₄	0	D ₅₆	D ₅₇	D ₅₈	...	D _{5n}
6		D ₆₁	D ₆₂	D ₆₃	D ₆₄	D ₆₅	0	D ₆₇	D ₆₈	...	D _{6n}
7		D ₇₁	D ₇₂	D ₇₃	D ₇₄	D ₇₅	D ₇₆	0	D ₇₈	...	D _{7n}
8		D ₈₁	D ₈₂	D ₈₃	D ₈₄	D ₈₅	D ₈₆	D ₈₇	0	...	D _{8n}
⋮		⋮									
n		D _{n1}	D _{n2}	D _{n3}	D _{n4}	D _{n5}	D _{n6}	D _{n7}	D _{n8}	...	0

شکل ۳۱-۳ ماتریس فاصله‌ها برای مسئله فرضی با n شهر

در این ماتریس منظور از D_{ij} فاصله شهر i و j است، یعنی D_{34} فاصله شهر سوم و چهارم است. با توجه به اینکه این ماتریس متقارن بوده پس به منظور پیدا کردن فاصله شهر سوم و چهارم می‌توان از عنصر D_{43} نیز استفاده کرد.

۳-۲-۴ مقایسه دو نوع اطلاعات مسئله به منظور پیاده‌سازی سخت‌افزاری

حال با دانستن مطالب قبلی می‌توان به ضعف و قوت هر یک از دو روش ذکر شده برای پیاده‌سازی سخت‌افزاری اشاره کرد:

- حجم سخت‌افزاری: حجم سخت‌افزاری خود دو قسمت است، یکی حجم بلوک‌های رم استفاده شده و دیگری حجم فضای مصرف شده برش‌ها. در روش مختصات شهرها، حجم بلوک‌های رم کمتری استفاده شده؛ چراکه فقط لازم است مختصات طول و عرض شهرها را ذخیره کنیم و بدین صورت به $n \times n$ سلول برای ذخیره‌سازی بر روی رم نیاز است؛ ولیکن برای پیاده‌سازی سخت‌افزار محاسبه فاصله اقلیدسی، تعداد برش‌های زیادی مصرف خواهد شد. در روش ماتریس فاصله‌ها، بلوک‌های رم بیشتری مصرف شده؛ چراکه ماتریس فاصله‌ها دارای n^2 عنصر است که این عناصر را بدین ترتیب بر روی رم قرار می‌دهیم که اول تمام عناصر سطر اول، سپس عناصر سطر دوم و الی آخر. پس برای دستیابی به فاصله دو شهر تنها کافیست که آدرس آن بر روی رم محاسبه شود که فرمول آن به صورت معادله ۳-۳ است:

$$\text{معادله ۳-۳} \quad \text{RAM ADDRESS} = (i - 1) \times n + j - 1$$

در این فرمول i و j آدرس دو شهر مورد نظر است و خانه‌های رم از آدرس صفر شروع می‌شود. ملاحظه می‌شود که فرمول آدرس رم بسیار ساده‌تر از فرمول محاسبه فاصله اقلیدسی با پیچیدگی محاسبه توان دو و رادیکال است که در نتیجه تعداد برش‌های کمتری از سخت-افزار را اشغال می‌کند.

- موازی‌سازی: به دلیل اینکه روش ماتریس فاصله‌ها حجم بلوک‌های رم را به سرعت استفاده کرده، در نتیجه از این ماتریس به تعداد محدودی می‌توان ایجاد کرد. مثلاً اگر تراشه XC5VLX330 و مسئله eil51 با ۵۱ شهر را در نظر بگیریم، به ۲۶۰۱ سلول رم برای

ماتریس فاصله‌ها نیاز است. با توجه به اینکه تعداد سلول‌های بلوک‌های رم باید توانی از دو باشد، پس حداقل تعداد سلول‌های بلوک رم باید 4096 در نظر گرفته شود. حال اگر دقت هر یک از این سلول‌ها را 32 بیت در نظر بگیریم و بیت‌های توازن بلوک‌های رم را نیز حذف کنیم، می‌توان حداقل 70 ماتریس فاصله بر روی این تراشه داشته باشیم. در حالیکه امکان پیاده‌سازی تعداد بیشتری از سخت‌افزار محاسبه فاصله اقلیدسی وجود دارد؛ چراکه همین تراشه دارای 51840 برش است که امکان پیاده‌سازی‌های بیشتری را نسبت به حالت ماتریس فاصله‌ها مهیا خواهد کرد. در نتیجه امکان موازی‌سازی بیشتری برای حالت مختصات شهرها وجود دارد.

• سرعت عملکرد: همانطور که در قبل هم بیان شد، محاسبه فاصله اقلیدسی بسیار زمان‌برتر از محاسبه آدرس رم یا رام است؛ چراکه به محاسبات بیشتر و پیچیده‌تری نیازمند است. با توجه به اینکه روش ماتریس فاصله‌ها، امکان موازی‌سازی کمتری نسبت به روش دیگر دارد، ولیکن سرعت بیشتری در انجام محاسبات و دسترسی به مقادیر فاصله‌ها داشته، محدودیت موازی‌سازی آن جبران می‌شود. همچنین با توجه به اینکه این روش بیشتر از منابع سخت‌افزاری در قالب بلوک‌های رم استفاده کرده و برش‌های تراشه را برای اجرای هسته‌های دیگر آزاد خواهد گذاشت، در نتیجه روش بهتری برای پیاده‌سازی بر روی سخت‌افزار است.

۳-۴-۵ بررسی دقت در پیاده‌سازی

رسیدن به نتایج درست و بهینه در هر مسئله‌ای نیازمند مشخص کردن دقت لازم برای حل مسئله است. برای رسیدن به دقت لازم در مسئله فروشنده دوره‌گرد، اگرچه مختصات طولی و عرضی شهرها مقادیر صحیحی هستند، ولی فاصله بین این شهرها، مقادیری دارای قسمت اعشاری خواهند بود که قسمت اعشاری آنها در بیشتر مسائل قابل چشم‌پوشی نبوده و در بدست آوردن نتایج نهایی موثر خواهند بود. در جدول ۳-۵ مقایسه چند مسئله و مقادیر کمینه و بیشینه فاصله شهرها در آنها آمده است.

جدول ۳-۵ مقایسه چند مسئله و بیشینه و کمینه فاصله شهرهای آنها

نام مسئله	تعداد شهر	کمینه فاصله دو شهر	بیشینه فاصله دو شهر
eil51	۵۱	۲/۲۲۶۱	۸۵/۶۳۲۹
eil76	۷۶	۲/۲۳۶۱	۸۵/۲۷۶
eil101	۱۰۱	۱/۴۱۴۲	۹۱/۸۳۱۴
pr76	۷۶	۳۰۰	۲۲۶۷۴/۲
pr107	۱۰۷	۲۰۰	۱۰۶۹۱/۲
pr136	۱۳۶	۱۷۰	۱۴۰۷۶/۴
pr226	۲۲۶	۱۰۰	۱۷۴۲۲/۸
kroA100	۱۰۰	۱۳/۰۳۸۴	۴۱۴۹/۸
kroB150	۱۵۰	۸/۰۶۲۳	۴۱۸۷/۳
kroB200	۲۰۰	۵	۴۱۷۰/۳
berlin52	۵۲	۱۵	۱۷۱۶/۱
st70	۷۰	۱	۱۲۸/۷۲۵
rat99	۹۹	۴/۱۲۳۱	۲۱۸/۲۲
lin105	۱۰۵	۳۱	۳۱۸۹/۴
bier127	۱۲۷	۱۱۶	۱۹۴۴۱/۳
ch150	۱۵۰	۱/۸۴۹۲	۸۴۹/۴۴۹
tsp225	۲۲۵	۶/۵	۵۱۸/۷۰۱
a280	۲۸۰	۰	۳۰۲/۳۳۸
pcb442	۴۴۲	۵۰	۴۸۴۱/۵

حتی اگر فقط برای مسائل خاصی مقادیر اعشاری را در نظر نگیریم، پیاده‌سازی به گونه‌ای اختصاصی شده است و قابل بسط دادن برای مسائل معروف دیگر، همانند مسئله کوله‌پشتی، مسائل مهندسی و به طور کلی مسائل دنیای واقعی^۱ که هدف غایی تولید علم است، خواهد بود.

در پیاده‌سازی‌های سخت‌افزاری همواره معضل ممیز شناور^۲ نیز مطرح بوده و مخصوصاً در روش محاسبه فاصله اقلیدسی به آن نیاز مبرم احساس می‌شود که در غیراینصورت نتایج متفاوت خواهند بود. اگر به روش ماتریس فاصله‌ها پیاده‌سازی کنیم، می‌توان از ممیز ثابت^۳ بهره برد که حجم سخت-افزاری کمتری مصرف کرده و پیچیدگی کمتری هم نسبت به روش ممیز شناور دارد.

یکی از تکنیک‌های استفاده شده در پیاده‌سازی‌های FPGA به منظور حل مشکل ممیز ثابت، ضرب عدد ممیزدار در یک مقدار صحیح است؛ به گونه‌ای که دیگر دارای ممیز نبوده و به یک عدد صحیح تبدیل شود. در این روش بعد از ضرب، عملیات مورد نظر را بر روی عدد صحیح بدست آمده اعمال کرده و در نهایت خروجی را تقسیم بر این عدد می‌کنیم تا به نتیجه درست برسیم. به دلیل اینکه در FPGA عمل تقسیم تعریف نشده و برای استفاده از عملگر تقسیم باید هسته تقسیم را به هسته‌های موجود اضافه کرد که خود باعث هدر رفتن منابع سخت‌افزاری است، مقدار ضرب شده در عدد را توانی از دو در نظر می‌گیرند، چراکه هر بیت شیفت به سمت چپ معادل ضرب در دو و هر بیت شیفت به سمت راست معادل تقسیم بر دو است. به عبارتی بسته به دقت مورد نیاز، عدد را به تعداد بیت‌های دلخواه به سمت چپ شیفت داده و در انتها نیز بسته به عملیات انجام شده، عدد حاصل را به تعداد بیت‌های لازم به سمت راست شیفت می‌دهند، یعنی در عمل جمع به همان تعداد بیت و در عمل ضرب دو برابر تعداد بیت‌ها به سمت راست شیفت می‌دهند.

۳-۴-۶ پیاده‌سازی اطلاعات مسئله بر روی FPGA

همانطور که در بخش‌های قبلی به آن اشاره شد، ماتریس فاصله‌ها روش بهتری برای پیاده‌سازی بر روی FPGA است، هم از نظر حجم سخت‌افزاری و تعداد برش‌هایی که استفاده کرده، هم از نظر

¹ Real World Problems

² Floating-Point

³ Fixed-Point

سرعت. تنها دو محدودیت وجود دارد، یکی مشکل ممیز و دیگری محدودیت تعداد بلوک‌های رم اشغال شده است.

- ممیز: به منظور حل این مشکل از تکنیک بیان شده در بخش قبل استفاده شده است با این تفاوت که به جای ضرب در توانی از دو، در توانی از ده ضرب شده است. فرض کنیم که هر یک از مختصات شهرها را در عدد ثابتی ضرب کنیم، این کار به منزله آن است که محورهای مختصات را به همراه مختصات شهرها بسط^۱ داده‌ایم، یعنی تنها فاصله شهرها بیشتر می‌شود، ولی در نتیجه حاصل شده تغییری ایجاد نشده و همچنان مسیر بهینه ثابت خواهد بود. در ادامه به صورت ریاضی این موضوع بررسی شده است. برای مسافت طی شده در مسئله فروشنده دوره‌گرد داریم:

$$F = \sum_{i=1}^{n-1} D(T[i], T[i+1]) + D(T[n], T[1]) \quad \text{معادله ۴-۳}$$

$$D(T[i], T[i+1]) = \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}$$

که در آن $T(i)$ شهر نام بوده و از فاصله اقلیدسی برای محاسبه فاصله شهرها استفاده شده است. جمله آخر در معادله ۴-۳ مربوط به فاصله شهر اول و آخر است.

$$D(T^*[i], T^*[i+1]) = \sqrt{(10^k \times x_i - 10^k \times x_{i+1})^2 + (10^k \times y_i - 10^k \times y_{i+1})^2} =$$

$$\sqrt{10^{2k} \times (x_i - x_{i+1})^2 + 10^{2k} \times (y_i - y_{i+1})^2} = 10^k \times \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} =$$

$$10^k \times D(T[i], T[i+1]) \quad \text{معادله ۵-۳}$$

$$F^* = \sum_{i=1}^{n-1} D(T^*[i], T^*[i+1]) + D(T^*[n], T^*[1]) =$$

$$\sum_{i=1}^{n-1} 10^k \times D(T[i], T[i+1]) + 10^k \times D(T[n], T[1]) =$$

$$10^k \times \left[\sum_{i=1}^{n-1} D(T[i], T[i+1]) + D(T[n], T[1]) \right] = 10^k \times F \quad \text{معادله ۶-۳}$$

^۱ Expand

در معادلات ۳-۵ و ۶-۷، T بیانگر مختصات یک شهر بعد از ضرب توان ده در آن است.

همانطور که در این دو معادله دیده می‌شود، اگر مختصات هر یک از شهرها را در توانی از ده ضرب کنیم، معادل آن است که فاصله آن‌ها را در همان ضریب، ضرب کرده‌ایم و در نهایت هم مسافت کل طی شده، در همان توان ده ضرب خواهد شد.

در نتیجه برای ساخت ماتریس فاصله‌ها هر یک از مقادیر را در توانی از ده ضرب کرده و به منظور دستیابی به دقت بیشتر حاصل را به نزدیکترین عدد صحیح گرد^۱ کرده و سپس نتیجه را در واحد رام ذخیره می‌کنیم. پس با توجه به مطالب بیان شده و استفاده از ماتریس فاصله‌ها در پیاده‌سازی، تنها عملیات لازم برای پیدا کردن کوتاهترین مسیر جمع و مقایسه خواهد بود که در نتیجه آن ضرب در توانی از ده در نتیجه نهایی تاثیری نخواهد داشت، جز مقدار نهایی نمایش داده شده برای مسافت طی شده که ضریبی از ده خواهد بود و به همین ترتیب نیاز به محاسبات عدد اعشاری در تمام هسته‌ها و تمام مراحل مختلف محاسبات متنفی خواهد شد.

• بلوک‌های رم استفاده شده: همانطور که در قبل به آن اشاره شد، ماتریس فاصله‌ها متقارن بوده و عناصر قطر اصلی هم صفر است؛ به عبارتی قسمت بالامثلی یا پایین‌مثلثی آن تمام فواصل شهرها را شامل خواهد شد. با این روش تعداد سلول‌های مورد نیاز بلوک‌های رم برای ذخیره این ماتریس از n^2 به $\frac{n^2-n}{2}$ تقلیل خواهد یافت. به عنوان مثال برای مسئله ۱۵۱ ذخیره این ماتریس از ۱۲۷۵ سلول رام نیاز است و با توجه به توان دو بودن تعداد سلول‌ها، رام پیاده‌سازی شده ۲۰۴۸ سلول خواهد داشت که نصف حال قبل است و در نتیجه تعداد حداقل ۱۴۰ ماتریس داده را می‌توان بر روی تراشه مذکور پیاده‌سازی کرد. ولیکن دیگر از فرمول قبلی برای محاسبه آدرس رام نمی‌توان استفاده کرد و به فرمول جدیدی نیاز است.

به منظور قرار دادن ماتریس فاصله‌های بالامثلی شده بر روی رام ابتدا سطر اول را از ستون دوم تا انتها بر روی رم قرار داده، سپس سطر دوم را از ستون سوم تا آخر، بعد از آن سطر سوم را از

^۱ Round

ستون چهارم تا آخر و به همین صورت تا سطر $n - 1$ با دو ستون n و n و نهایتاً سطر $n - 1$ و ستون n . نمایش سلول‌های رم پس از قرار دادن ماتریس بالامثلثی فاصله‌ها در آن، در شکل ۳۲-۳ آمده است. به دلیل اینکه قسمت‌هایی از پیاده‌سازی به صورت ترکیبی بوده (همانند واحد ذخیره و محاسبه) که به محض قرار گرفتن ورودی بر روی آن‌ها محاسبات آغاز خواهد شد و همچنین اینکه واحد محاسبه‌گر آدرس رم یا رام در حالت بیکاری آدرس صفر را در خروجی خود قرار خواهد داد، آدرس صفرم رام به مقدار بیشینه ممکن برای دقت مورد نظر (۳۲ بیت) اختصاص داده شده است و فواصل شهرها از آدرس یکم رام شروع می‌شوند.

I=1 J=2	11111111111111...11111111111111	ADDRESS 0
I=1 J=3		ADDRESS 1
I=1 J=4		ADDRESS 2
⋮		ADDRESS 3
I=1 J=n		⋮
I=2 J=3		ADDRESS n-1
I=2 J=4		ADDRESS n
I=2 J=5		ADDRESS n+1
⋮		ADDRESS n+2
I=2 J=n		⋮
I=3 J=4		ADDRESS 2n-3
I=3 J=5		ADDRESS 2n-2
I=3 J=6		ADDRESS 2n-1
⋮		ADDRESS 2n
I=3 J=n		⋮
I=4 J=5		ADDRESS 3n-6
⋮		ADDRESS 3n-5
I=k J=k+1		⋮
I=k J=k+2		
I=k J=k+3		
⋮		
I=k J=n		
I=k+1 J=k+2		
⋮		
I=n-3 J=n		
I=n-2 J=n-1		
I=n-2 J=n		
I=n-1 J=n		

شکل ۳۲-۳ نحوه قرارگیری ماتریس فاصله‌ها بر روی رام و آدرس متناظر آن‌ها

در معادله ۷-۳ نحوه بدست آوردن فرمول محاسبه آدرس نشان داده شده است:

معادله ۷-۳

$$\begin{aligned}
 ADDRESS[D(T[i], T[j])] &\Rightarrow i < j \\
 i = 1 \rightarrow ADDRESS &= j - i \\
 i = 2 \rightarrow ADDRESS &= (n - 1) + j - i \\
 i = 3 \rightarrow ADDRESS &= (n - 1) + (n - 2) + j - i \\
 i = 4 \rightarrow ADDRESS &= (n - 1) + (n - 2) + (n - 3) + j - i \\
 i = 5 \rightarrow ADDRESS &= (n - 1) + (n - 2) + (n - 3) + (n - 4) + j - i \\
 \dots \\
 i = i \rightarrow ADDRESS &= (n - 1) + (n - 2) + \dots + (n - (i - 1)) + j - i
 \end{aligned}$$

زمانی که $i = 1$ است (یعنی سطر اول ماتریس بالامثلی) و آدرس رام از شماره یک شروع شود، آدرس فاصله دو شهر از تفاضل شماره دو شهر بدست می‌آید. بعد از طی شدن کامل سطر اول که تعداد عنصرهای آن $n - 1$ است، برای محاسبه آدرس فواصل در سطر دوم این مقدار باید به تفاضل شماره دو شهر اضافه شود. همینطور در محاسبه آدرس سطر سوم ماتریس، عناصر سطر اول $(n - 1)$ و دوم $(n - 2)$ نیز باید در نظر گرفته شوند و همینطور تا آخر. در سطر آخر معادله ۷-۳ فرمول آدرس برای یک سطر و ستون دلخواه نشان داده شده است. همانطور که در معادله‌های ۷-۳ آورده شده است و از ماتریس بالامثلی فاصله‌ها نیز قابل استنتاج است، مقدار i همواره باید از مقدار j کوچکتر باشد. فرمول نهایی و ساده شده آدرس رام به شکل معادله ۸-۳

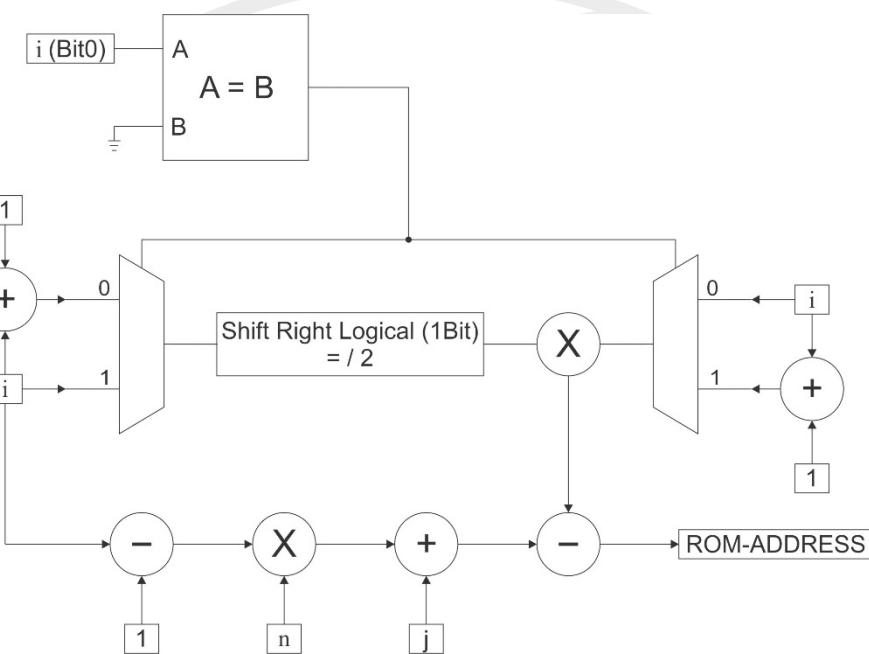
خواهد بود:

معادله ۸-۳

$$\begin{aligned}
 ROM - ADDRESS &= \\
 (n - 1) + (n - 2) + \dots + (n - (i - 1)) + j - i &= (i - 1) \times n + j - 1 - 2 - \dots - (i - 1) - i = \\
 (i - 1) \times n + j - (1 + 2 + \dots + i) &= (i - 1) \times n + j - \frac{i \times (i + 1)}{2} \Rightarrow \\
 ROM - ADDRESS &= (i - 1) \times n + j - \frac{i \times (i + 1)}{2}
 \end{aligned}$$

در معادله ۸-۳ نتیجه دنباله عددی که از ۱ شروع شده و تا i ادامه دارد، توسط ریاضی دان بزرگ گوس ارائه شده است. در این فرمول یک تقسیم، دو ضرب و سه جمع وجود دارد که تقسیم آن

بر دو بوده و مقادیر صورت کسر $i+1$ است و این بیانگر آن است که یکی از این دو مقدار همواره زوج است. در نتیجه در پیاده‌سازی این معادله به جای تقسیم از یک شیفت به راست بهره گرفته خواهد شد. اگر بخواهیم نحوه پیاده‌سازی ساختاری این فرمول را نشان دهیم به شکل ۳۳-۳ است.



شکل ۳۳-۳ نحوه پیاده‌سازی در سطح ساختار فرمول محاسبه آدرس رام

در شکل ۳۳-۳ از یک مقایسه‌گر و بیت صفر i برای مشخص کردن زوج و فرد بودن آن استفاده شده است، i چه فرد باشد چه زوج، به وسیله دو مالتیپلکسر مقدار زوج مشخص شده و یک واحد به سمت راست شیفت داده خواهد شد که معادل تقسیم بر دو است و مقدار فرد در نتیجه ضرب خواهد شد.

همانطور که در قبل هم اشاره شد، تعداد خانه‌های رم‌ها چه توزیع شده و چه بلوکی باید توانی از دو باشند. برای محاسبه تعداد بیت‌های لازم برای آدرس دهی رم‌ها با فرض اینکه n تعداد شهرها، x تعداد بیت‌های لازم برای آدرس دهی شماره شهرها و y تعداد بیت‌های لازم برای آدرس دهی رم باشد، خواهیم داشت:

$$x = \lceil \log_2 n \rceil$$

$$y = \lceil \log_2 \frac{n^2 - n}{2} \rceil$$

معادله ۹-۳

در معادله ۹-۳ بیانگر گرد کردن نتیجه لگاریتم دو به سمت مثبت بینهایت است. از x در ساخت بهینه رم توزیع شده و از y در ساخت بهینه رم بلوکی استفاده می‌شود. این فرمول در پیاده‌سازی‌های روی FPGA به صورت گستردگی کاربرد دارد. اثبات فرمول ۹-۳ در ادامه آورده شده است:

$$2^x \geq n \Rightarrow \log_2 2^x \geq \log_2 n \Rightarrow x \times \log_2 2 \geq \log_2 n \Rightarrow x \geq \log_2 n \quad \text{معادله ۱۰-۳}$$

۳-۲-۵-۵ واحد محاسبه‌گر آدرس رام یا رم

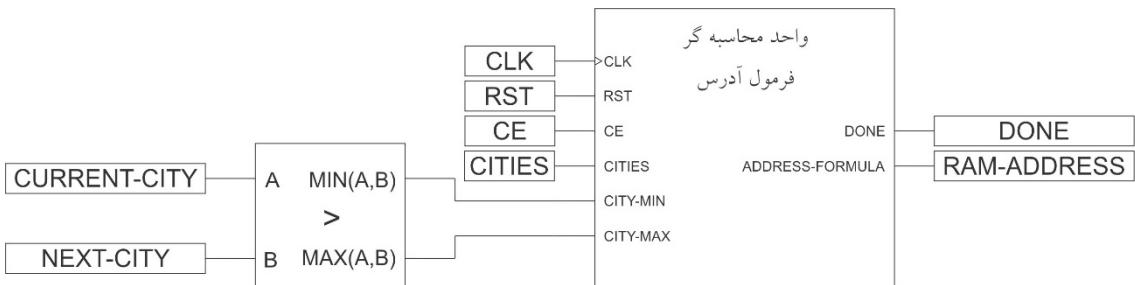
در بخش قبل به نحوه محاسبه آدرس رام پرداخته شد، در این بخش به واحد محاسبه می‌پردازیم. در شکل ۳۴-۳ واحد مذکور نمایش داده است، این واحد شماره دو شهر را از واحد مسیر حرکت یا واحد زنبور که بعداً به آن خواهیم پرداخت، از ورودی‌های CURRENT-CITY و NEXT-CITY دریافت و آدرس رام را محاسبه کرده و بر روی خروجی RAM-ADDRESS قرار داده و یک سیگنال اتمام عملیات هم بر روی خروجی DONE به مدت یک کلاک پالس ایجاد می‌کند.



شکل ۳۴-۳ واحد محاسبه‌گر آدرس رام یا رام

همانطور که قبلاً به آن اشاره شد، در محاسبه آدرس، بزرگ و کوچک بودن شماره دو شهر مهم بوده و به همین علت ساختار این واحد به شکل ۳۵-۳ طراحی شده است. در این ساختار مقادیر کوچک

و بزرگ شماره شهرها مشخص شده و بر روی ورودی‌های متناظر واحد محاسبه‌گر فرمول آدرس قرار خواهند گرفت. خروجی‌های واحد محاسبه‌گر فرمول آدرس مستقیماً به خروجی‌های این واحد متصل است.



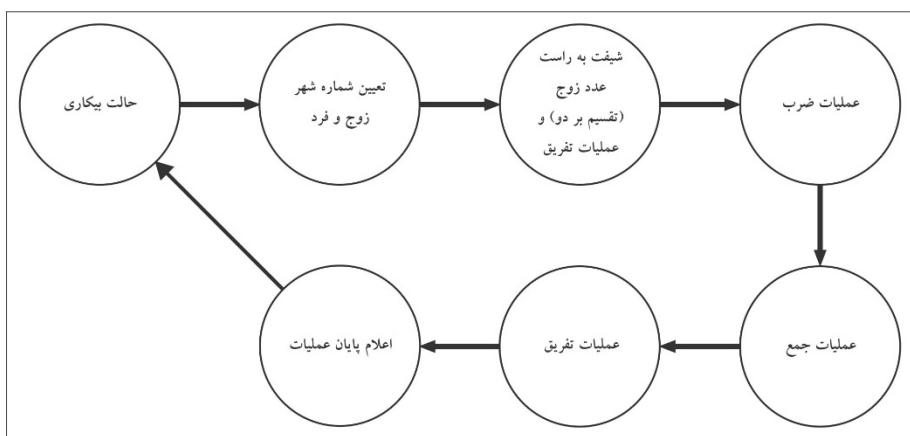
شکل ۳۵-۳ ساختار واحد محاسبه‌گر آدرس رم یا رام

در شکل ۳۶-۳ واحد محاسبه‌گر فرمول آدرس نشان داده شده است که شماره کوچکتر یا همان i را در ورودی CITY-MIN و شماره بزرگتر یا همان j را از ورودی CITY-MAX دریافت خواهد کرد.



شکل ۳۶-۳ واحد محاسبه‌گر فرمول آدرس

برای پیاده‌سازی فرمول محاسبه آدرس رام از ماشین حالت استفاده شده است که نمودار آن در شکل ۳۷-۳ نشان داده شده است. در این نمودار داریم:



شکل ۳-۳۷ نمودار FSM واحد محاسبه‌گر فرمول آدرس

- حالت بیکاری: در این وضعیت سیگنال فعال‌سازی و مجاز بودن شماره شهرها (بزرگتر از صفر) بررسی شده و در صورتی که این شرایط مهیا شود، عملیات محاسبه آدرس شروع می‌شود.
- تعیین شماره شهر زوج و فرد: در این حالت زوج یا فرد بودن شماره شهر کوچکتر بررسی شده و مقادیر ثبات‌های نگهدارنده اعداد زوج و فرد که یکی از دو مقدار $i+1$ یا i هستند، تعیین می‌شوند.
- شیفت به راست عدد زوج (تقسیم بر دو) و عملیات تفریق: در این حالت ثبات حاوی مقدار زوج یک بیت به سمت راست شیفت داده می‌شود که همان پیاده‌سازی تقسیم بر دو است. همچنین در این حالت مقدار شهر کوچکتر منهای یک $(1-i)$ در فرمول نیز محاسبه می‌شود.
- عملیات ضرب: در این مرحله دو ضرب موجود در معادله آدرس محاسبه خواهند شد، این دو ضرب عبارتند از:
 ۱. ضرب حاصل تقسیم بر دو و مقدار فرد
 ۲. ضرب $1-i$ و تعداد شهرهای مسئله (n)
- عملیات جمع: در این مرحله نتیجه حاصل از ضرب دوم و شماره شهر بزرگتر با هم جمع می‌شوند و به مرحله بعد می‌روند.

• عملیات تفریق: در این حالت نتیجه جمع حالت قبل از مقدار ضرب اول کم می‌شود؛ در واقع آدرس نهایی رام محاسبه می‌شود.

• اعلام پایان عملیات: در این مرحله، سیگنال اتمام عملیات یعنی DONE بر روی خروجی ایجاد می‌شود. همچنین سیگنال فعال‌سازی نیز بررسی شده و در صورت صفر بودن به حالت بیکاری برخواهد گشت. به عبارتی تا زمانی که سیگنال فعال‌سازی بر روی ورودی این واحد برقرار باشد، آدرس محاسبه شده را بر روی خروجی خود نگه خواهد داشت.

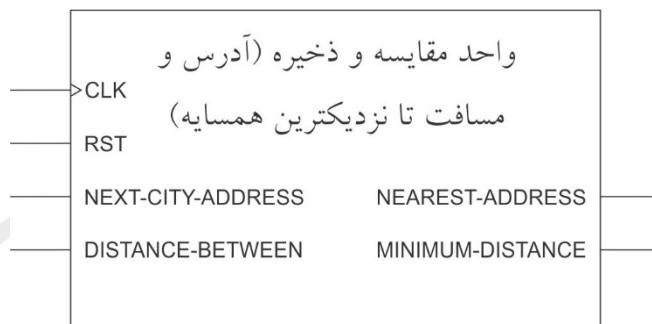
۳-۲-۶ واحد مقایسه و ذخیره (آدرس و مسافت تا نزدیکترین همسایه)

این واحد وظیفه مشخص کردن نزدیکترین همسایه و ذخیره آدرس متناظر آن در واحد مسیر حرکت را عهده‌دار است. در قبل بیان شد که در محاسبه نزدیکترین همسایه، یک شهر به عنوان شهر حال حاضر در نظر گرفته می‌شود و شهرهای بعد از آن در صفت به عنوان همسایه‌های موجود برای این شهر در نظر گرفته می‌شوند. در هر بار که یکی از این همسایه‌ها برای مقایسه در نظر گرفته می‌شوند و مقدار فاصله این شهر تا شهر حال حاضر از رام نیز استخراج می‌شود، باید واحد وجود داشته باشد که این فواصل را مقایسه کرده و در صورتی که از همسایه قبلی ذخیره شده فاصله کمتری با

شهر حال حاضر داشته باشد، فاصله جدید را به همراه آدرس آن ذخیره کند. روند انتخاب یک شهر از مجموعه شهرهای موجود و مقایسه فاصله آن نسبت به شهرهای دیگر با شهر حال حاضر تا جایی ادامه می‌یابد که دیگر شهری برای مقایسه نماند. حال بر روی خروجی‌های این واحد، آدرس نزدیکترین همسایه و فاصله متناظر آن تا شهر حال حاضر وجود خواهد داشت. این واحد در شکل

۳-۳۸ نشان داده شده است. ورودی NEXT-CITY-ADDRESS آدرس یک شهر همسایه و ورودی DISTANC-BETWEEN فاصله آن شهر را تا شهر حال حاضر را دریافت می‌کند و نزدیکترین همسایه بین همسایه‌هایی که تاکنون مورد بررسی قرار گرفته‌اند را بر روی خروجی‌های خود قرار خواهد داد. آدرس این همسایه بر روی خروجی NEAREST-ADDRESS و فاصله آن تا شهر حال حاضر بر روی MINIMUM-DISTANCE قرار خواهد گرفت. این واحد همواره دارای خروجی

است و برای آنکه نزدیکترین همسایه موجود را پیدا کنیم، خروجی‌های این واحد پس از اتمام بررسی همه شهرهای موجود باید خوانده شوند.

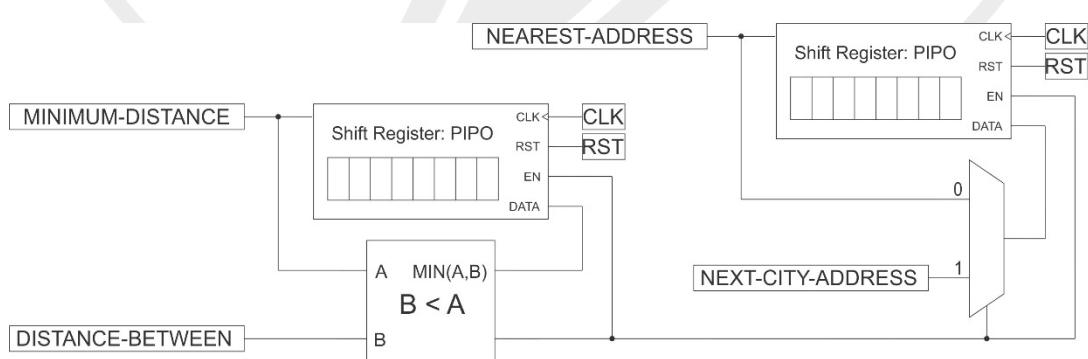


شکل ۳-۳۸ واحد مقایسه و ذخیره

پیاده‌سازی هسته مذکور در سطح ساختار بوده و از دو شیفت رجیستر ورود موازی، خروج موازی^۱، یک مقایسه‌کننده و یک مالتی‌پلکسر تشکیل شده است. این طرح در شکل ۳-۳۹ نمایش داده شده است. نحوه عملکرد آن بدین صورت است که وقتی فاصله جدیدی بر روی ورودی قرار بگیرد با مسافت قبلی مقایسه شده و در صورتی که این فاصله کمتر از مقدار قبلی باشد، خروجی مقایسه‌گر یک می‌شود. خروجی مقایسه‌گر ورودی انتخاب‌گر مالتی‌پلکسر و سیگنال فعال‌سازی شیفت رجیستر را یک کرده که در نتیجه آدرس شهر جدید را برای ذخیره به ورودی‌های ثبات متناظر آن می‌فرستد، تا در لب بالارونده کلاک پالس بعدی بر روی آن ذخیره شوند. همچنین این مقایسه‌گر مقدار فاصله کمینه را بر روی خروجی دیگر خود به منظور ذخیره‌سازی قرار خواهد داد. اگر شهر جدید فاصله بیشتری از شهر ذخیره شده قبلی داشته باشد، هم ورودی شیفت رجیسترها به مقادیر قبلی خود متصل و هم سیگنال فعال‌سازی شیفت رجیسترها صفر خواهد بود؛ در نتیجه تغییری در مقدار ذخیره شده بر روی آن‌ها ایجاد نخواهد شد. نکته حائز اهمیتی که در این ساختار باید در نظر گرفته شود آنست که در زمان شروع جستجو برای همسایگی یک شهر جدید، این واحد همچنان مقادیر قبلی ناشی از مقایسه‌های قبلی را بر روی خود نگاه خواهد داشت؛ در نتیجه باید برای آن راه حلی پیدا کرد. این راه حل را می‌توان در مقدار اولیه یا بعد از بازنشانی شیفت رجیستر مربوط به ذخیره فاصله جست.

^۱ Parallel-In, Parallel-Out (PIPO)

اگر مقدار بازنشانی این شیفت رجیستر بیشینه ممکن در نظر گرفته شود و در ابتدای هر روند پیدا کردن نزدیکترین همسایه برای شهر حال حاضر بازنشانی شود، مشکل مرتفع خواهد شد. بدین ترتیب در ابتدای یک جستجو وقتی اولین شهر و مسافت متناظر آن بر روی ورودی‌های واحد قرار می‌گیرند، به دلیل اینکه فاصله دریافت شده بی‌شک از مقدار بیشینه کوچکتر است، جایگزین آن خواهد شد و این مقایسه تا انتها به درستی انجام خواهد گرفت. اگرچه لازم نیست، ولیکن بهتر است به منظور هماهنگ‌سازی و استفاده کمتر از واحدهای زیرمجموعه، فلیپ‌فلاب‌های شیفت رجیستر مربوط به ذخیره آدرس نیز در زمان بازنشانی با یک پر شوند. این مقدار بیشینه آدرس، یک مقدار غیر مجاز بوده و اشاره به آخرین آدرس رم توزیع شده دارد و همانطور که در قبل بیان شد، این سلول از رم توزیع شده به عنوان حافظه واسط در نظر گرفته شده است. در قبل هم اشاره شد که خروجی واحد محاسبه‌گر آدرس رم یا رام، در موقعی که فعال نیست، مقدار صفر را بر روی خود قرار داده و در نتیجه رام هم مقدار ذخیره شده در آدرس صفرم خود را به ورودی این واحد ارسال می‌کند. حال می‌توان دریافت که چرا آدرس صفرم رام همواره مقدار بیشینه در نظر گرفته شده است. فرض کنیم که مقدار کوچکتری از فاصله ذخیره شده بر روی واحد مقایسه و ذخیره، بر روی آدرس صفرم رام قرار گرفته باشد. پس شهری که آدرس آن بر روی ورودی این واحد است به همراه فاصله موجود در آدرس صفرم رام ذخیره خواهد شد که یک گیر¹ در سیستم به حساب می‌آید و نتایج غلطی را در پی خواهد داشت.



شکل ۳۹-۳ ساختار واحد مقایسه و ذخیره

¹ Bug

۳-۳-۳ واحد زنبور عسل

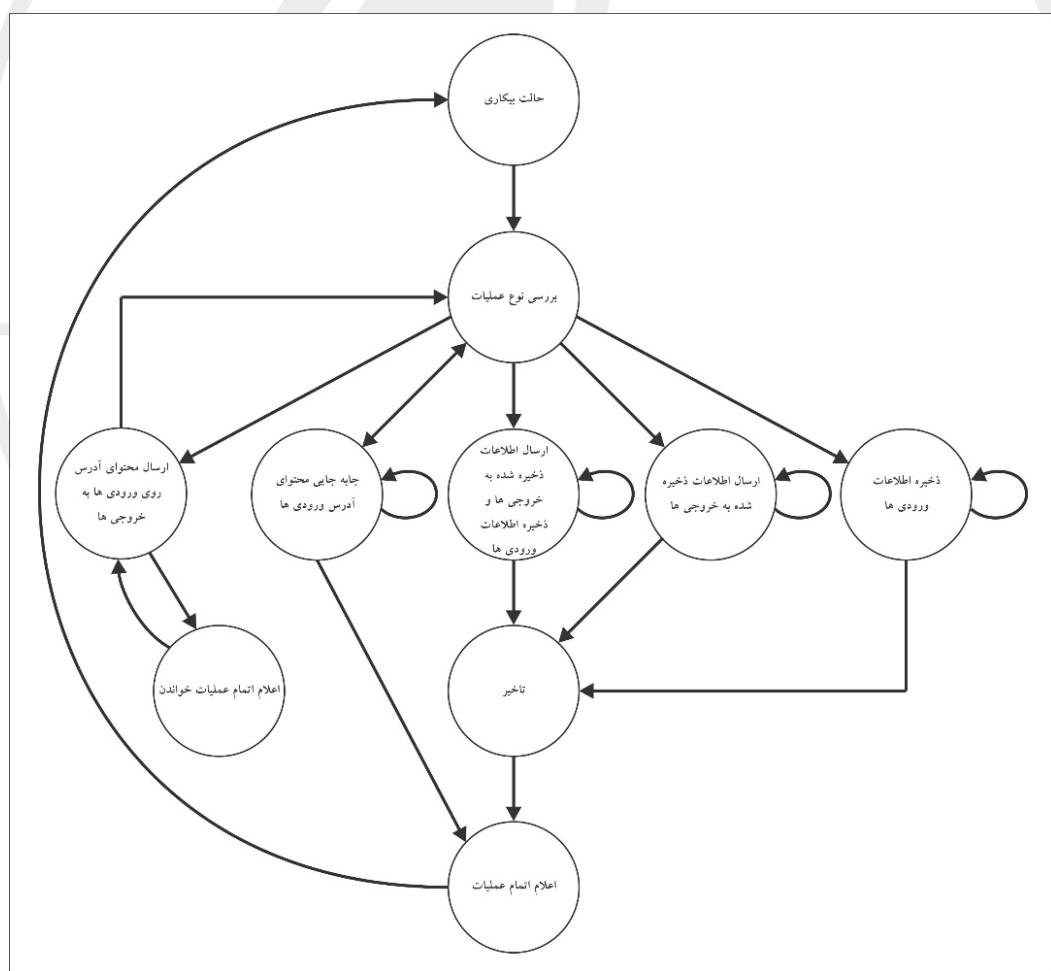
در هر یک از تکرارهای الگوریتم، وظیفه نگهداری جواب پیدا شده، بر عهده واحد زنبور است. این واحد بسیار شبیه واحد مسیر حرکت است؛ یعنی برای نگهداری صفت یا مسیر حرکت فروشنده دوره‌گرد از یک رم توزیع شده با طول مساوی با رم توزیع شده واحد مسیر حرکت بهره می‌برد. این واحد سه حالت عملکردی مشترک با واحد مسیر حرکت نیز دارد؛ به بیانی دیگر تنها تفاوت‌های این دو واحد در دو نوع عملکرد و نگهداری مسافت طی شده برای مسیر متناظر است. در شکل ۴۰-۳، واحد زنبور نشان داده شده است. همانند واحد مسیر از ورودی SELECT نوع عملکرد مشخص می‌شود. از ورودی FITNESS-IN برای دریافت طول مسافت طی شده استفاده می‌شود و از خروجی‌های DATA1-OUT و DATA2-OUT برای قرار دادن اطلاعات ذخیره شده مسیر چه به صورت تکی و چه به صورت گروهی بهره برد می‌شود. همچنین مقدار مسافت متناظر مسیر ذخیره شده، نیز همواره در خروجی FITNESS-OUT قابل دسترس خواهد بود. دلیل این امر، ساختار واحد طبقه‌بندی است که باید مستقل از سیگنال فعال‌سازی زنبور به مسافت مسیر دسترسی داشته باشد.



شکل ۴۰-۳ واحد زنبور عسل

نمودار FSM این واحد در شکل ۴۱-۳ نشان داده شده است که شامل ۵ حالت عملکردی متفاوت است. تمامی این حالت‌ها دو کلک پالس پس از دریافت سیگنال فعال‌سازی شروع می‌شوند و پس از

اتمام عملیات مورد نظر، سیگنال DONE یا اتمام عملیات را بر روی خروجی قرار می‌دهند. جز در یکی از حالت‌ها، مرحله بعدی پس از حالت اعلام اتمام عملیات حالت بیکاری خواهد بود. در این حالت‌های مشابه، الگوریتم دوباره منتظر سیگنال فعال‌سازی بعد باقی خواهد ماند تا که با توجه به حالت انتخاب شده عملیات مورد نظر را دوباره آغاز کند. ولیکن در تک حالت استثناء، پس از دریافت سیگنال فعال‌سازی مستقیماً به مرحله بررسی نوع عملیات متقل خواهد شد. توجه شود که حالت‌هایی که تا اتمام انجام عملیات تغییر حالت نداده و تنها تکرار می‌شوند، پیکان برگشت به خود دارند. همچنین وجود حالت تاخیر در روند برخی از عملکردها بدان دلیل است که سیگنال اتمام عملیات یک کلاک پالس پس از تخصیص مقادیر به خروجی‌ها صادر شود.



شکل ۴۱-۳ نمودار FSM زنبور عسل

این حالت‌ها عبارتند از:

- ذخیره اطلاعات ورودی‌ها: در این حالت مقداری را که در ورودی DATA1-IN دریافت می‌کند، در آدرسی که در ورودی DATA2-IN وجود دارد می‌نویسد و مقدار روی ورودی FITNESS-IN را به عنوان مسافت طی شده متناظر آن در نظر می‌گیرد. این همان ویژگی است که می‌توان با استفاده از آن مسیر نزدیکترین همسایگی ارسال شده را در زنبور ذخیره کرد. در این عملکرد در هر کلک پالس یک شماره شهر در زنبور ذخیره می‌شود.
- جابه‌جایی محتوای آدرس ورودی‌ها: همانند واحد مسیر حرکت، در صورتی که آدرس روی ورودی‌ها به یک آدرس یکسان اشاره نکنند، جای دو شهر را در صف عوض می‌کند؛ با این تفاوت که مقدار مسافت جدید روی ورودی خود را نیز ذخیره کرده و روی خروجی نشان می‌دهد. طول انجام خود عملیات جابه‌جایی ۳ کلک پالس است. حال با در نظر گرفتن دو کلک پالس برای بررسی سیگنال فعال‌سازی و نوع عملیات انتخابی و همچنین یک کلک پالس برای صدور سیگنال اتمام عملیات، کل این عملیات شش کلک پالس به طول خواهد انجامید.
- ارسال محتوای آدرس روی ورودی‌ها به خروجی‌ها: مشابه واحد مسیر حرکت، شماره دو شهر ذخیره شده در آدرس ورودی‌ها را بر روی خروجی‌ها قرار خواهد داد. لازم به ذکر است که همانند واحد مسیر حرکت و به منظور تنظیم زمانی با دیگر هسته‌ها، باید این محتوا تا بازنشانی یا دریافت سیگنال فعال‌سازی بعدی بر روی خروجی‌ها باقی بماند. به همین دلیل این حالت با چهار حالت دیگر متفاوت است و حتی حالت اتمام عملیات مخصوص به خود را دارد. این عملیات سه کلک پالس به طول خواهد انجامید و در کلک پالس بعد سیگنال اتمام عملیات بر روی خروجی DONE ظاهر خواهد شد.
- ارسال اطلاعات ذخیره شده به خروجی‌ها: این حالت مشابه واحد مسیر حرکت عمل کرده و مسیر ذخیره شده به همراه آدرس آنها را بر روی خروجی‌ها قرار می‌دهد. در این حالت

شماره هر شهر و آدرس متناظر آن در یک کلاک پالس به خروجی فرستاده می‌شود و مقدار مسافت هم که همواره بر روی خروجی FITNESS-OUT قابل دسترس است.

ارسال اطلاعات ذخیره شده به خروجی‌ها و ذخیره اطلاعات ورودی‌ها: یکی از عملکردهای متفاوت این واحد با واحد مسیر حرکت است که این اجازه را به واحد می‌دهد که همزمان یک آدرس رم و محتوای آن را به خروجی‌ها ارسال کند و مقدار روی یکی از ورودی‌ها را در همان آدرس (یا آدرس روی ورودی دیگر) ذخیره کند. بدین صورت که یک آدرس و محتوای آن را به خروجی‌ها اختصاص داده و در همین زمان ورودی‌های واحد را به ورودی‌های رم توزیع شده اختصاص داده و یک سیگنال اجازه نوشتن^۱ هم به رم توزیع شده ارسال می‌کند. به این روش در کلاک پالس بعدی مقادیر آدرس و محتوای قبلی آن روی خروجی‌ها نشسته و مقدار روی یک از ورودی‌ها در همان آدرس (یا آدرس روی ورودی دیگر) ذخیره می‌شود. لازم به توجه است که همانند واحد مسیر در این واحد هم از رم توزیع شده همزمان بهره گرفته شده است؛ که در نتیجه استفاده از این نوع رم، این نوع عملکرد همزمان و بدون خطأ امکان‌پذیر شده است. این عملکرد آنجا موثر است که زنبورها نیاز به طبقه‌بندی داشته و پس از تشخیص نیاز به جابه‌جایی مسیر و مسافت ذخیره شده بر روی آنها، بتوانند همزمان این جابه‌جایی را انجام دهند که در نتیجه آن زمان این جابه‌جایی به نصف تقلیل می‌یابد.

با در نظر گرفتن اینکه هدف استفاده بهینه از حجم سخت‌افزاری است، در می‌باییم که بهتر است که یک واحد زنبور در تمام مراحل حل مسئله یعنی طبقه‌بندی، جستجوی محلی و بروزرسانی ثابت باشد و تنها در قسمت‌های متفاوت فراخوانی شود و با در نظر گرفتن ویژگی‌های متنوع واحد زنبور، این امر امکان‌پذیر شده است.

¹ Write Enable

۴-۳-۳ واحد طبقه‌بندی زنبورها

در ابتدای کار یعنی وقتی که مسیر اولیه به زنبورها اختصاص می‌یابد یا در زمان بازگشت از مرحله بروزرسانی، باید زنبورها بر اساس تابع شایستگی آن‌ها مرتب شوند؛ به صورتی که در نهایت زنبور اول دارای بهترین مقدار تابع شایستگی باشد که در اینجا مقدار کمینه مسافت طی شده برای فروشنده دوره‌گرد است و زنبور آخر نیز دارای بیشترین مسافت طی شده باشد. پس به روندی برای این مرحله از الگوریتم نیاز است که بیشترین سرعت یا به عبارتی موازی‌سازی را دارا باشد.

در بخش قبل بیان شد که زنبورها مقدار تابع شایستگی را همواره در خروجی خود نگه می‌دارند. این ویژگی این امکان را مهیا می‌سازد که بدون فعال‌سازی واحد زنبور میزان شایستگی آن را مورد بررسی قرار داد و در صورت لزوم دستور جابه‌جایی را صادر کرد. همچنین به دلیل محدودیت منابع سخت-افزاری، زنبورها یکتا در نظر گرفته شده‌اند و همگی در هسته اصلی بوده و تنها در قسمت‌های مختلف فراخوانی می‌شوند. با این توضیحات مشخص است که واحد طبقه‌بندی باید در کنار زنبورها قرار گرفته و امکان مجزا بودن آن نیست؛ به عبارتی باید این واحد در خود هسته اصلی، جایی که تمام زنبورها را شامل می‌شود، پیاده‌سازی شود.

این واحد در سطح ساختار است و به دلیل نحوه طراحی کمترین میزان سخت‌افزار را در کنار بیشترین میزان موازی‌سازی ارائه می‌دهد. این ساختار به سه بخش تقسیم می‌شود:

- اجزاء جابه‌جایی

در این بخش مشخص می‌شود که کدام یک از زنبورها اجازه دارند که محتویات مسیر و تابع شایستگی خود را با هم تعویض کنند. نحوه عملکرد این بخش بدین صورت است که دو زنبور با جایگاه $i - 1$ و i اجازه جابه‌جایی محتوای خود را دارند، اگر و تنها اگر مقدار تابع شایستگی زنبور با جایگاه i کمتر از زنبور با جایگاه $i - 1$ و کمتر از تابع شایستگی زنبور با جایگاه $i + 1$ باشد. به بیانی دیگر دو زنبور i و $i - 1$ باید جابه‌جا شوند، ولی دو زنبور i و $i + 1$ نیازی به جابه‌جایی ندارند. در این طرح اولویت جابه‌جایی با زنبور با جایگاه پایین‌تر (شماره بیشتر) است. به عنوان یک مثال فرض کنیم که بعد از قرار دادن مسیرهای نزدیکترین همسایگی برای مسئله

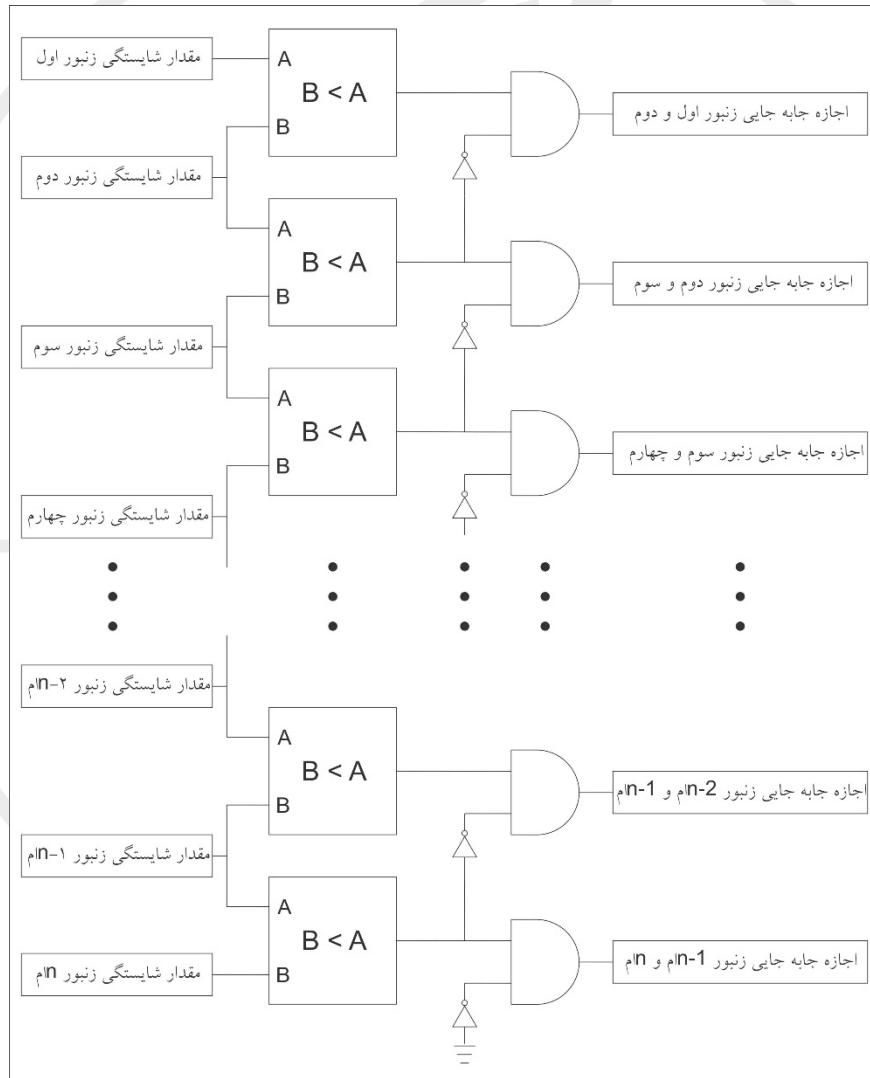
eil51 در ۱۱ زنبور، حال می‌خواهیم این زنبورها را طبقه‌بندی کنیم. مقادیر تابع شایستگی و مسیرهای این زنبورها در جدول ۳-۶ آورده شده است:

جدول ۳-۶ ترتیب حرکت و تابع شایستگی مسیر نزدیکترین همسایگی برای مسئله eil51

مسیر حرکت	مسافت	زنبور	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
۱۳۲ ... ۲۲۴۳	۵۱۴	I	۱	۱	۱	۱	۱	۱	۱	۱	۱	۱
۲۱۶ ... ۲۲۴۰	۶۰۱	II	۹	۹	۲	۲	۲	۲	۲	۲	۲	۲
۳۲۰ ... ۳۷۴۰	۵۷۵	III	۷	۲	۹	۹	۳	۳	۳	۳	۳	۳
۴۱۷ ... ۲۴۴۳	۵۳۱	IV	۲	۷	۷	۳	۹	۴	۴	۴	۴	۴
۵۳۸ ... ۴۵۴۰	۵۰۷	V	۴	۴	۳	۷	۴	۹	۵	۵	۵	۵
۶۴۸ ... ۴۰۴۳	۵۶۸	VI	۶	۳	۴	۴	۷	۵	۹	۶	۶	۶
۷۲۳ ... ۵۴۰	۵۴۹	VII	۳	۶	۶	۵	۵	۷	۶	۹	۷	۷
۸۲۶ ... ۳۹۴۰	۵۷۷	VIII	۸	۸	۵	۶	۶	۶	۷	۷	۹	۸
۹۵۰ ... ۳۶۴۰	۶۰۶	IX	۱۰	۵	۸	۸	۸	۸	۸	۸	۸	۹
۱۰۴۹ ... ۴۰۴۳	۵۵۹	X	۵	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰
۱۱۳۲ ... ۳۶۴۰	۶۰۹	XI	۱۱	۱۱	۱۱	۱۱	۱۱	۱۱	۱۱	۱۱	۱۱	۱۱

در سمت چپ جدول ۳-۶ مسیر نزدیکترین همسایگی (شماره شهرهایی که فروشنده دوره‌گرد از آن‌ها به ترتیب از چپ به راست گذر می‌کند) و مقادیر تابع شایستگی متناظر آن‌ها آورده شده است. در سمت راست آن شماره زنبورها به یونانی و نحوه جابه‌جایی هر کدام از این مسیرها براساس شماره اولویت نهایی آن‌ها نشان داده شده است. خانه‌های تیره شده بیانگر امکان جابه‌جایی بین دو زنبور در مرحله بعد است. مشاهده می‌شود که در ابتدا، مقدار تابع شایستگی زنبور چهارم کمتر از زنبور سوم و پنجم بوده، پس اجازه جابه‌جایی به زنبور سوم و چهارم داده می‌شود. در همین حال زنبور شماره هفتم مقدار تابع شایستگی کمتری از زنبور ششم و هشتم داشته، پس زنبور ششم و هفتم هم اجازه جابه‌جایی دارند. به همین ترتیب دو زنبور نهم و دهم

هم اجازه جابه‌جایی اطلاعات را دارند. به عبارتی در یک مرحله شش زنبور مقادیر ذخیره شده خود را با هم تعویض می‌کنند. پس از اتمام این جابه‌جایی مرحله بعد شروع می‌شود، در این مرحله اجازه جابه‌جایی بین زنبورهای دوم و سوم، پنجم و ششم و همچنین هشتم و نهم داده می‌شود. با تکرار این روال در نه مرحله ($S_0 \sim S_9$) کلیه زنبورها به طور کامل طبقه‌بندی می‌شوند.



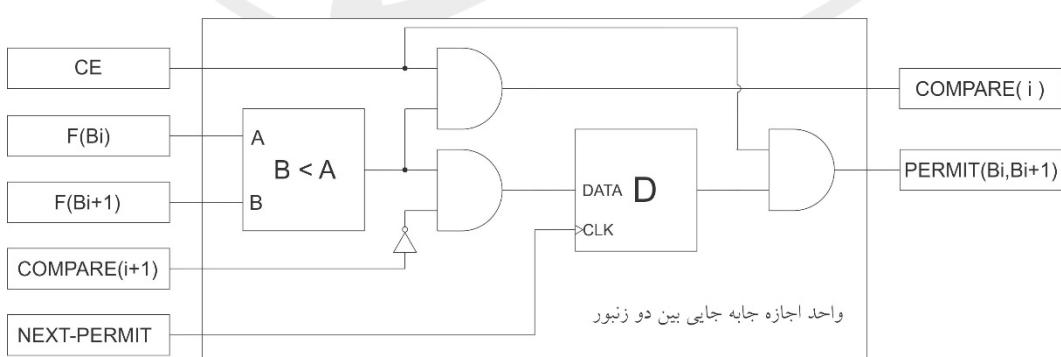
شکل ۴۲-۳ ساختار قسمت اجازه جابه جایی از واحد طبقه‌بندی

نمای کلی و ساده شده واحد اجازه جابه جایی در شکل ۴۲-۳ آورده شده است. همانطور که دیده می‌شود به وسیله یک مقایسه‌گر و یک گیت NOT و AND امکان پیاده‌سازی این قسمت فراهم

شده است. ورودی گیت NOT از نتیجه مقایسه مقدار تابع شایستگی دو زنبور با جایگاه پایین تر (شماره بیشتر) گرفته شده است؛ که تضمین می کند که در بین سه زنبور، اگر زنبور دوم و سوم امکان جابه جایی داشته باشند، دو زنبور اول و دوم این اجازه را نخواهند داشت و این به دلیل آن است که امکان جابه جایی همزمان برای زنبور دوم یعنی جابه جایی با زنبور اول و جابه جایی با زنبور سوم وجود ندارد. همانطور که از شکل ۴۲-۳ پیداست، زنبور آخر (بیشترین شماره) ورودی NOT خود را از مقدار صفر گرفته که نمایانگر آنست که دارای بالاترین اولویت جابه-جایی است و همچنین نیازی به خروجی مقایسه گر زنبور اول هم نیست. این طرح را می توان به صورت بالاترین اولویت با زنبور اول هم پیاده سازی کرد. به صورت ریاضی معادله اجازه جابه-جایی به صورت معادله ۱۱-۳ خواهد بود:

$$PERMIT(B_{i-1}, B_i) = \begin{cases} 1 & PERMIT(B_i, B_{i+1}) = 0 \\ 0 & PERMIT(B_i, B_{i+1}) = 1 \end{cases} \quad \text{معادله ۱۱-۳}$$

به منظور ساده سازی در پیاده سازی، سه عنصر از ساختار اجازه جابه جایی در شکل ۴۲-۳، شامل یک مقایسه گر، یک گیت AND و یک گیت NOT به صورت مجزا و در قالب یک هسته در نظر گرفته شده است. همچنین به منظور کنترل زمان صدور اجازه جابه جایی و فعال سازی هسته، چند عنصر منطقی به آن اضافه شده است. در نتیجه این بهبود، این هسته با هسته اصلی هماهنگ تر عمل خواهد کرد. حال این واحد شامل یک مقایسه گر، سه گیت AND، یک گیت NOT و یک فلیپ فلاب پ می شود. نمای این هسته، ساختار داخلی و ورودی و خروجی های آن در شکل ۴۳-۳ نمایش داده شده است.

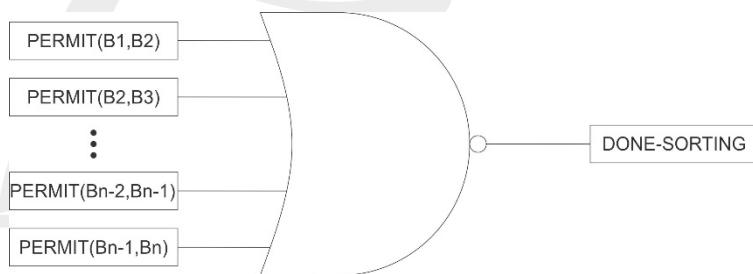


شکل ۴۳-۳ هسته اجازه جابه جایی

در شکل ۴۳-۳، $F(B_i)$ تابع شایستگی زنبور i ، $F(B_{i+1})$ تابع شایستگی زنبور $i+1$ ، $COMPARE(i)$ نتیجه مقایسه تابع شایستگی زنبور i ام و زنبور $i+1$ ام، $COMPARE(i+1)$ نتیجه مقایسه تابع شایستگی زنبور $i+1$ ام و زنبور $i+2$ ام، $NEXT-PERMIT$ دستور صدور اجازه جابه‌جایی و $PERMIT(B_i, B_{i+1})$ اجازه جابه‌جایی زنبور i ام و زنبور $i+1$ ام است. همچنین ورودی CE برای کنترل فعال‌سازی این واحد در نظر گرفته شده است.

- اتمام جابه‌جایی

مراحل جابه‌جایی زمانی به اتمام می‌رسد که دیگر هیچ اجازه جابه‌جایی وجود نداشته باشد، به عبارتی تمامی زنبورها در جایگاه درست و ترتیبی خود قرار گرفته باشند. این شرط با یک گیت NOR چند ورودی قابل پیاده‌سازی است که در شکل ۴۴-۳ نشان داده شده است.



شکل ۴۴-۳ سیگнал اتمام کار واحد طبقه‌بندی

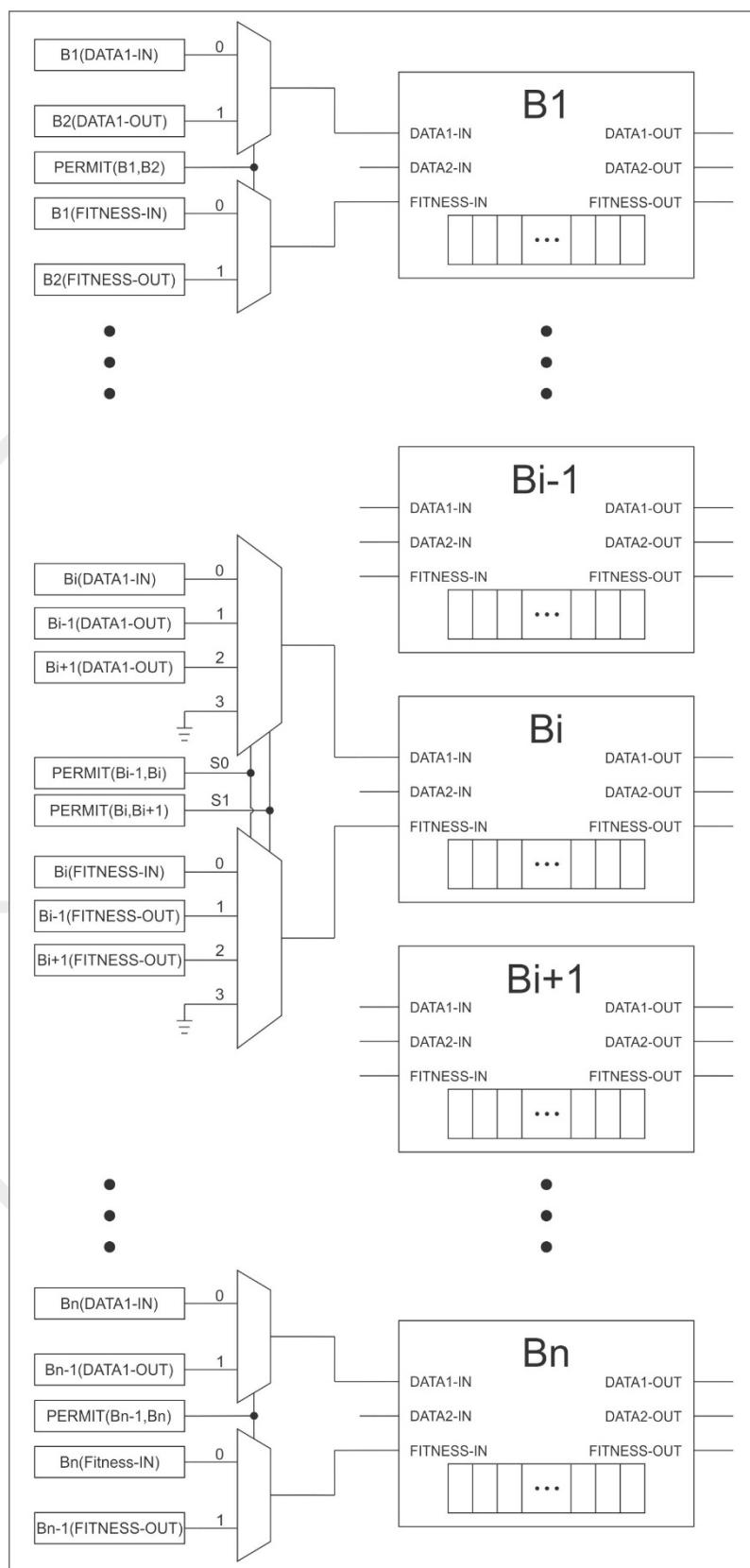
- تعیین اتصالات ورودی زنبورها

بعد از مشخص شدن و صدور اجازه جابه‌جایی زنبورها و قبل از اینکه به دو زنبور مجاز دستور جابه‌جایی ارسال شود، باید اتصال بین ورودی و خروجی‌های زنبورها مشخص شوند. این اتصالات با استفاده از سیگنال کنترلی اجازه جابه‌جایی و به شکل ۴۵-۳ مشخص می‌شوند. به منظور وضوح تصویر و جلوگیری از پیچیدگی، ورودی و خروجی‌های غیر لازم زنبورها نمایش داده نشده است. همچنین اتصالات به DATA2-IN زنبورها هم نمایش داده نشده است. این اتصالات همانند اتصالات به DATA1-IN زنبورها است؛ با این تفاوت که به جای نوشته‌های DATA2-X سمت چپ مالتی‌پلکسرهای مربوطه در شکل، باید مقادیر DATA1-X را با

جایگرین کرد. پس در این پیاده‌سازی برای هر زنبور سه مالتی‌پلکسر در نظر گرفته شده است، دو تا برای مسیر حرکت (یکی برای شماره شهرها و دیگری برای آدرس متناظر آن در رم) و دیگری برای مسافت طی شده متناظر آن. برای تمامی زنبورها جز زنبور اول و آخر مالتی‌پلکسراها ۴ به ۱ هستند؛ ولیکن چون زنبور دیگری در بالا یا پایین این دو زنبور وجود ندارد، در نتیجه برای این دو زنبور دو مالتی‌پلکسر ۲ به ۱ کافیست.

اگر الگوریتم در هر مرحله‌ای بغیر از طبقه‌بندی باشد، چون سیگنال فعال‌سازی واحدهای اجازه جابه‌جایی صفر است، در نتیجه هیچ اجازه جابه‌جایی داده نشده و تمامی مالتی‌پلکسراها ورودی صفرم خود را به ورودی زنبور متصل می‌کنند که نتیجتاً امکان استفاده واحدهای دیگر از زنبور فراهم می‌شود. اگر الگوریتم در مرحله طبقه‌بندی باشد و اجازه جابه‌جایی دو زنبور B_{i-1} و B_i وجود داشته باشد، خروجی DATA1-IN زنبور B_{i-1} به ورودی DATA1-OUT زنبور B_i ، خروجی DATA1-OUT زنبور B_{i-1} به ورودی DATA2-IN زنبور B_i ، خروجی DATA2-OUT زنبور B_i به ورودی OUT زنبور B_{i-1} و خروجی DATA2-OUT زنبور B_i به ورودی DATA1-IN زنبور B_{i-1} و خروجی OUT زنبور B_i به ورودی FITNESS-IN زنبور B_{i-1} وصل خواهد بود. همین روند برای خروجی FITNESS-OUT و ورودی FITNESS-IN دو زنبور هم برقرار است. به عبارتی دیگر، خروجی‌های هر زنبور به ورودی‌های متناظر زنبور دیگر متصل می‌شود.

حال که اجازه جابه‌جایی‌ها مشخص شد و ورودی‌های زنبورها نیز تعیین شدند، فقط کافیست که دو زنبور را در حالت ارسال اطلاعات ذخیره شده به خروجی‌ها و ذخیره اطلاعات ورودی‌ها قرار داده و سیگنال فعال‌سازی به آن‌ها داده شود تا محتوای آن‌ها در $n+3$ کلک پالس (n تعداد شهرهای مسئله) تعویض شوند.



شکل ۳-۴۵ تعیین اتصالات ورودی واحد زنبورها در واحد طبقه‌بندی

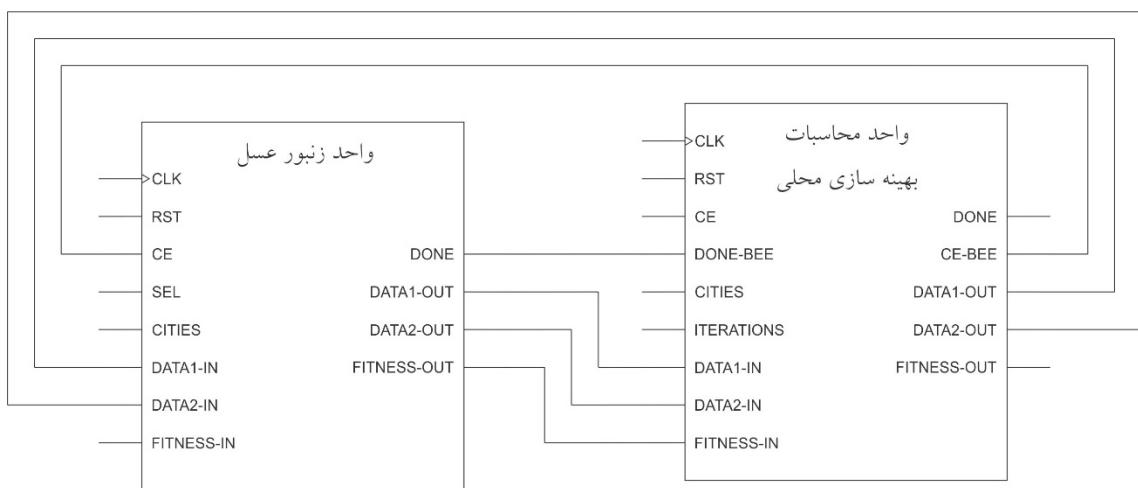
۵-۳-۳ واحد محاسبات بهینه‌سازی محلی

بعد از اینکه زنبورها طبقه‌بندی شدند، حال نوبت به جستجوی محلی می‌رسد. باید واحدی وجود داشته باشد که متعلق به یک زنبور بوده و حول جواب آن زنبور جستجو کرده و به دنبال نتایجی بهتر برای آن زنبور باشد. این واحد محاسبات بهینه‌سازی محلی نامیده می‌شود و این نامگذاری بدان دلیل است که این واحد در کنار واحد زنبور عمل کرده و تنها تعدادی شهر و تابع شایستگی را از زنبور دریافت کرده و بعد از انجام عملیات روی آن‌ها، نتیجه نهایی را به واحد بروزرسانی ارائه داده تا در صورت نائل شدن به نتایج بهتر، بروزرسانی‌های لازم روی زنبور انجام شود. در شکل ۴۶-۳ ورودی و خروجی‌های واحد مذکور نشان داده شده است.



شکل ۴۶-۳ واحد محاسبات بهینه‌سازی محلی

یکی از ویژگی‌های این واحد آنست که در صورت تمایل می‌تواند جستجو را چندین بار تکرار کند. با قرار دادن تعداد تکرارهای مدنظر روی ورودی ITERATIONS، در انتهای جستجو بهترین نتیجه در کل تکرارها روی خروجی‌های واحد قرار داده خواهد شد. در شکل ۴۷-۳ نحوه اتصال این واحد به واحد زنبور در مرحله جستجوی محلی نمایش داده شده است.



شکل ۳-۴۷ نحوه اتصالات دو واحد زنیبور و محاسبات بهینه‌سازی محلی

مشاهده می‌شود که خروجی‌های زنیبور به ورودی‌های این واحد و خروجی‌های این واحد به ورودی‌های واحد زنیبور متصل است. در طول روند بهینه‌سازی این واحد به اطلاعات شهرها و مسافت کل مسیر نیازمند بوده که این اطلاعات را از زنیبور بعد از قرار گرفتن آن در نوع عملکرد ارسال محتوای آدرس روی ورودی‌ها به خروجی‌ها، دریافت می‌کند. به این ترتیب که آدرس‌های دو شهر را بر روی ورودی زنیبور قرار داده و سپس یک سیگنال فعال‌سازی برای زنیبور ارسال می‌کند. زنیبور هم بعد از قرار دادن شماره شهرها بر روی خروجی خود یک سیگنال اتمام عملیات را به ورودی این واحد داده تا بقیه مراحل جستجوی جواب بهینه‌تر پیگیری شود. همانطور که قبلاً هم اشاره شد، زنیبور این دو شهر را تا سیگنال بازنشانی یا فعال‌سازی بعدی در خروجی‌های خود نگه داشت. این واحد شامل سه زیرواحد است:

- واحد تولید تصادفی آدرس دو شهر
- واحد محاسبه‌گر آدرس رام یا رم
- واحد رام بلوکی با یک درگاه

توضیح دو واحد آخر در قبل آمده است و در زیربخش‌های بعدی به واحد اول خواهیم پرداخت. حال قبل از آنکه به توضیح نمودار ماشین حالت یا به عبارتی مراحل کنترلی این واحد پردازیم، ابتدا به توضیحاتی در مورد روش انتخابی به منظور جستجوی محلی یعنی روش 2-opt می‌پردازیم.

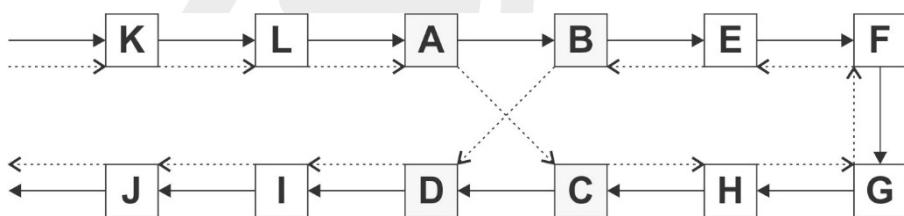
۱-۵-۳-۳ تفسیر روش 2-opt برای پیاده‌سازی بر روی سخت‌افزار

فرض کنید که تکه مسیر فرضی زیر را داریم و دو یال به عنوان مثال AB و CD انتخاب شده‌اند. در این حالت مسیر اصلی طی شده در شکل ۴۸-۳ که با پیکان‌های سیاه رنگ مشخص شده، به صورت زیر است:

$$K \rightarrow L \rightarrow A \rightarrow B \rightarrow E \rightarrow F \rightarrow G \rightarrow H \rightarrow C \rightarrow D \rightarrow I \rightarrow J$$

در بهینه‌سازی به روش 2-Opt بیان می‌شود که اگر شرط $AC + BD < AB + CD$ برقرار بود، مسیر بهینه جدید در شکل ۴۸-۳ که با پیکان‌های نقطه‌چین نشان داده شده است، بدین ترتیب خواهد بود:

$$K \rightarrow L \rightarrow A \rightarrow C \rightarrow H \rightarrow G \rightarrow F \rightarrow E \rightarrow B \rightarrow D \rightarrow I \rightarrow J$$



شکل ۴۸-۳ روش بهینه‌سازی محلی 2-Opt

با کمی دقت ملاحظه می‌شود که تکه مسیر تغییر یافته $C \rightarrow H \rightarrow G \rightarrow F \rightarrow E \rightarrow B$ حاصل دوران تکه مسیر $B \rightarrow E \rightarrow F \rightarrow G \rightarrow H \rightarrow C$ است. به عبارتی روش 2-opt دوران تکه مسیرهایی با طول‌های متفاوت را مورد بررسی قرار می‌دهد و اگر حاصل این دوران تابع شایستگی بهتری را نتیجه دهد، جایگزین خواهد شد.

مطلوب دیگری که از این تحلیل نتیجه می‌شود آنست که برای بررسی‌های مختلف و انتخاب تکه مسیرهای متفاوت نیازی به انتخاب دو یال شامل چهار گره نیست، بلکه تنها کافیست که دو گره ابتدایی و انتهایی را انتخاب کنیم؛ یعنی در اینجا A و D. تنها این شرط باید رعایت شود که دو شهر انتخابی باید حداقل دو شهر از هم فاصله داشته باشند؛ چراکه در غیراینصورت دوران مفهومی پیدا نمی‌کند. به بیانی دیگر در انتخاب آدرس دو شهر، باید شرط $|ADDR_A - ADDR_D| \geq 3$ را در نظر گرفت.

همچنین با فرض اینکه مسیر جدید مسیر کوتاهتری نسبت به قبل باشد، برای محاسبه مسافت مسیر جدید احتیاجی به محاسبه دوباره طول کل مسیر نیست، چراکه تنها طول دو یال در مسیر تغییر کرده و تنها جهت حرکت در بقیه مسیر عوض شده است و در نتیجه می‌توان با استفاده از معادله ۱۲-۳ به راحتی مسافت مسیر جدید را بدست آورد:

$$= \text{طول مسافت پیموده شده جدید} \quad \text{معادله ۱۲-۳}$$

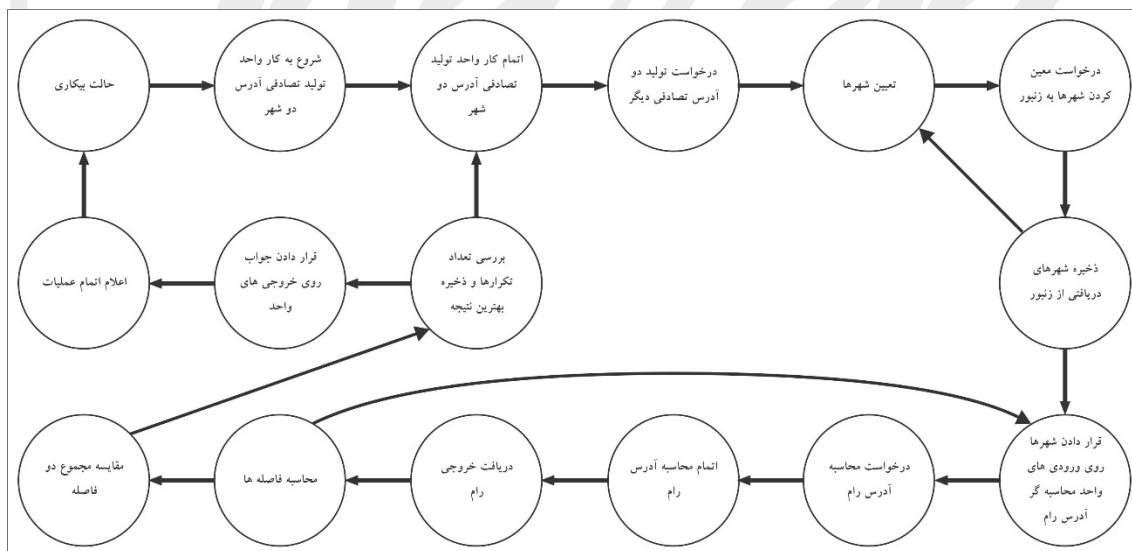
$$\text{مجموع دو یال اصلی} - \text{مجموع دو یال جدید جایگزین} + \text{مسافت مسیر سابق}$$

این مسافت در مثال اخیر بدین صورت است:

$$D_{NEW} = D_{OLD} + (AC + BD) - (AB + CD) \quad \text{معادله ۱۳-۳}$$

۴۹-۳-۳ نمودار ماشین حالت واحد محاسبات بهینه‌سازی محلی

در شکل ۴۹-۳ نمودار ماشین حالت این واحد نمایش داده شده است و در ادامه به توضیحات مربوط به آن پرداخته شده است.



شکل ۴۹-۳ نمودار FSM واحد محاسبات بهینه‌سازی محلی

- **حالت بیکاری:** در این حالت مقادیر سیگنال‌ها بازنمانی شده و همچنین سیگنال فعال‌سازی بررسی می‌شود. در صورت اینکه به واحد سیگنال فعال‌سازی داده شود، به حالت بعد منتقل می‌شود.

- شروع به کار واحد تولید تصادفی آدرس دو شهر: در این مرحله به واحد تولید تصادفی آدرس دو شهر دستور شروع داده می‌شود. این دو شهر همان شهرهای اول و آخر در روش 2-opt هستند.
- اتمام کار واحد تولید تصادفی آدرس دو شهر: خروجی اتمام عملیات واحد تولید تصادفی آدرس دو شهر بررسی شده و در صورتی که انتخاب آدرس دو شهر اول و آخر تکه مسیر به پایان رسیده باشد، آنها را در دو ثبات ذخیره کرده و به مرحله بعد می‌رود.
- درخواست تولید دو آدرس تصادفی دیگر: در این مرحله به منظور موازی‌سازی هرچه بیشتر هسته‌ها، اگر تعداد تکرارها بزرگتر از یک باشد و آخرين تکرار هم نباشد، فرمان تولید آدرس تصادفی دو شهر دیگر به واحد تولید تصادفی آدرس دو شهر داده می‌شود. این کار باعث می‌شود که در تکرارهای بعدی این روند، آدرس‌های جدید آماده باشند و زمانی برای این مراحل صرف نشود.
- تعیین شهرها: هدف روش 2-opt انتخاب دو یال از مسیر است؛ در نتیجه بعد از انتخاب دو شهر اول و آخر مسیر، باید چهار شهر در نظر گرفته شود. این چهار شهر شامل شهر انتخابی اول با آدرس R_1 و شهر بعد از آن با آدرس $R_1 + 1$ ، شهر انتخابی آخر با آدرس R_2 و شهر قبل از آن با آدرس $R_2 - 1$ است. حال لازم است که محتوای این چهار آدرس یعنی شماره شهرها از واحد زنبور دریافت شود. بدین منظور و با وجود دو ورودی و دو خروجی در واحد زنبور، تنها لازم است که دو بار به زنبور درخواست داده شود. به همین دلیل مرحله حال حاضر دو بار تکرار شده و در تکرار اول دو آدرس $R_1 + 1$ و در تکرار دوم (برگشت از حالت ذخیره شهرهای دریافتی از زنبور) آدرس $R_2 - 1$ را روی خروجی واحد قرار می‌دهد.
- درخواست معین کردن شهرها به زنبور: در این مرحله پس از آنکه آدرس‌ها بر روی ورودی-های واحد زنبور قرار گرفته‌اند، پیغام شروع عملیات یا فعال‌سازی برای واحد زنبور ارسال می‌شود.

- ذخیره شهرهای دریافتی از زنیور: بعد از اینکه زنیور پیغام اتمام عملیات را از طریق ورودی BEE-DONE برای این واحد ارسال کرد، دو شهر دریافتی بر روی دو ثبات ذخیره می‌شوند. این حالت نیز دو بار تکرار شده که با بررسی اتمام کار زنیور در اولین اجرا، شماره دو شهر اول و در دومین اجرا شماره دو شهر دوم را ذخیره می‌کند. در اولین تکرار به مرحله تعیین شهرها و در اجرای دوم به حالت بعد می‌رود.
- قرار دادن شهرها روی ورودی‌های واحد محاسبه‌گر آدرس رام: همانطور که در قبل اشاره شد، هدف انتخاب چهار یال است. اولین یال مربوط به شهر اول با آدرس R_1 و شهر یکی پس از آن با آدرس R_1+1 است، یال دوم بین شهر آخر با آدرس R_2 و شهر قبلی از آن با آدرس R_2-1 است. یال سوم بین دو شهر با آدرس‌های R_1 و R_2-1 و بالاخره یال چهارم بین شهرها با آدرس‌های R_1+1 و R_2 است. بدین ترتیب این مرحله چهار بار تکرار شده و در هر تکرار به ترتیب شماره دو شهر متناظر یکی از یال‌ها را بر روی ورودی‌های واحد محاسبه‌گر آدرس رام قرار می‌دهد. باید توجه شود که واحد محاسبه‌گر آدرس رام به گونه‌ای طراحی شده است که تا زمانی که سیگنال فعال‌سازی آن برقرار باشد، آدرس محاسبه شده و سیگنال اتمام عملیات را روی خروجی‌های خود نگه میدارد. این ویژگی در هسته تشخیص نزدیکترین همسایه بسیار حیاتی است. پس باید در این مرحله سیگنال فعال‌سازی واحد محاسبه‌گر آدرس رام بازنشانی شود تا مقادیر روی خروجی‌های این واحد بازنشانی و آماده شروع محاسبات آدرس جدید شود.
- درخواست محاسبه آدرس رام: پس از قرار دادن شماره دو شهر روی ورودی‌های واحد محاسبه‌گر آدرس رام و بازنشانی سیگنال فعال‌سازی آن، با ارسال درخواست به آن واحد یا به عبارتی سیگنال فعال‌سازی در این مرحله، آدرس فاصله این دو شهر بر روی رام مشخص می‌شود.
- اتمام محاسبه آدرس رام: در اینجا اتمام محاسبات آدرس رام بررسی می‌شود و در صورت پایان یافتن به مرحله بعد می‌رود. این مرحله وظیفه ذخیره مجموع دو یال اصلی را نیز به

عهده دارد. بدین صورت که اگر در تکرار سوم خود باشد، یعنی مسافت دو یال اصلی از رام دریافت شده و مجموع آنها نیز محاسبه شده باشد، این مقدار را ذخیره کرده و ثبات انباره را به منظور محاسبه مجموع دو یال دیگر بازنشانی می‌کند.

- دریافت خروجی رام: بعد از محاسبه آدرس رام به یک کلاک پالس تاخیر نیاز است تا نتایج رام بر روی خروجی آن واحد قرار بگیرد. این تاخیر با یک مرحله میانی مهیا شده است.
- محاسبه فاصله‌ها: در این مرحله، خروجی رام با ثبات انباره‌ای که مسئول نگهداری مجموع دو یال است جمع می‌شود. به عبارتی مجموع دو یال در این مرحله محاسبه می‌شود. اگر روند محاسبه آدرس چهار یال و محاسبه دو فاصله مورد نیاز برای بهینه‌سازی 2-opt به اتمام رسیده باشد، به حالت بعد که مقایسه این دو مقدار است انتقال می‌یابد، در غیراینصورت به مرحله قرار دادن شهرها بر روی ورودی واحد محاسبه‌گر آدرس رام منتقل می‌شود.
- مقایسه مجموع دو فاصله: دو مقدار حاصل یعنی مجموع طول دو یال اصلی و دو یال جدید حاصل از دوران شهرهای میانی بررسی می‌شود. اگر مجموع طول دو یال جدید کوچکتر بود، آدرس دو شهر اول و آخر به همراه مسافت مسیر جدید که در همین مرحله محاسبه می‌شود، بر روی ثبات‌ها ذخیره خواهد شد.
- بررسی تعداد تکرارها و ذخیره بهترین نتیجه: قبل ذکر شد که این واحد امکان چندین بار تکرار را دارد و در انتهای تمامی تکرارها بهترین نتیجه حاصله را روی خروجی‌های خود قرار می‌دهد. در اینجا، نتیجه حاصل شده در حالت قبل با بهترین نتیجه کل تکرارها مقایسه می‌شود و اگر الگوریتم در حالت قبل به نتیجه بهتری نائل شده باشد، این نتیجه در ثبات‌های مربوطه ذخیره خواهد شد. همچنین در این مرحله تعداد تکرارها نیز بررسی می‌شود و در صورتی که هنوز به تعداد تکرارهای مشخص شده نرسیده باشد، به مرحله اتمام کار واحد تولید تصادفی آدرس دو شهر، یعنی جایی که وجود آدرس‌های تصادفی جدید بررسی می‌شود، انتقال می‌یابد؛ در غیراینصورت به حالت بعد می‌رود.

- قرار دادن جواب روی خروجی‌های واحد: اگر تکرارها به اتمام رسیده باشد، بهترین نتیجه در این مرحله روی خروجی‌ها قرار می‌گیرد. این نتیجه عبارت است از آدرس دو شهر اول و آخر بهترین بهینه‌سازی در طول تکرارها و مقدار جدید طول مسیر متناظر آن.
- اعلام اتمام عملیات: در این حالت خروجی DONE یک شده و همچنین سیگنال فعال‌سازی هم بررسی می‌شود. این واحد هم مانند واحد محاسبه‌گر آدرس رام به گونه‌ای طراحی شده است که تا زمان برقراری سیگنال فعال‌سازی، نتایج را روی خروجی‌های خود نگه می‌دارد. پس در این حالت اگر سیگنال فعال‌سازی بازنشانی شود، به مرحله بیکاری انتقال می‌یابد. لازم به ذکر است که این واحد اگر جواب بهینه‌ای پیدا نکند، مقادیر را روی خروجی‌های مربوط به آدرس شهرها را صفر و مسافت طی شده را بیشینه ممکن قرار می‌دهد که این مقادیر برای دوران مسیر غیرمجاز شمرده می‌شوند. بدین روش واحد بهینه‌سازی به واحد بروزرسانی اعلام می‌کند که جواب بهتری حاصل نشده است. به منظور بررسی این شرایط کافیست صفر بودن آدرس R_2 که آدرس بزرگتر است و روی خروجی DATA2-OUT قرار می‌گیرد، سنجیده شود.

۳-۵-۳ واحد تولید تصادفی آدرس دو شهر

در زیربخش قبل به واحدی با عملکرد تولید آدرس دو شهر به صورت تصادفی اشاره شد که این شهرها نقش دو شهر اول و آخر در روش Opt-2 را دارند. این دو شهر را نمی‌توان به صورت کاملاً تصادفی انتخاب کرد و باید شرایطی داشته باشند. شرط اول فاصله مجاز دو شهر است که باید حداقل دو شهر در مسیر فروشند از هم فاصله داشته باشند که این شرط را با نام کران پایین می‌شناسیم. همچنین این آدرس‌ها باید در محدوده آدرس‌دهی شهرها باشند؛ یعنی بیشینه تعداد شهرها منهای یک که در اینجا به این شرط کران بالا اطلاق می‌شود. در شکل ۳-۵۰ واحد تولید تصادفی آدرس دو شهر آورده شده است.



شکل ۳۵۰ واحد تولید آدرس تصادفی دو شهر

این واحد تعداد شهرها را از ورودی گرفته و با دریافت سیگنال فعالسازی، شهر اول مسیر (R_1) را بر روی خروجی CITY-ADDRESS-RAND1 قرار داده و شهر آخر مسیر (R_2) را بر روی خروجی CITY-ADDRESS-RAND2 قرار می‌دهد. سپس سیگنال اتمام عملیات را بر روی خروجی DONE قرار داده و منتظر سیگنال فعالسازی بعدی شده تا دو آدرس شهر دیگر را تولید کند.

این واحد تا زمانی که سیگنال بازنشانی را دریافت نکرده یا تمام جفت ترکیب‌های موجود و مجاز آدرس دو شهر را یک بار بر روی خروجی‌های خود قرار نداده باشد، به مقادیر اولیه خود باز نخواهد گشت. طراحی این واحد به گونه‌ای است که تکرار یک جفت ترکیب یا جابه‌جایی دو شهر مجاز شمرده شده است. ولیکن جابه‌جایی دو شهر در یک حالت در نظر گرفته نمی‌شود و آن حالتی است که آدرس یکی از شهرها صفر باشد. برای مسئله فرضی با n شهر، تعداد جفت ترکیب‌ها قبل از رسیدن به مقادیر اولیه، $(n-3) \times (n-2)$ است. این معادله حاصل فرمول دنباله عددی گوس برای یک دنباله عددی است که از یک شروع شده و تا $n-3$ ادامه می‌یابد؛ با این تفاوت که در دو ضرب شده است؛ به عبارتی تکرار یا جابه‌جایی دو شهر در آن مجاز است. در قبل هم از فرمول دنباله عددی گوس در ساده‌سازی محاسبه آدرس رام استفاده شده بود. بزرگترین عدد دنباله یعنی $3-n$ نتیجه شرط کران پایین است. همانطور که ذکر شد برای آدرس صفر ترکیب دو آدرس تکرار نمی‌شود و تعداد این جفت ترکیب‌ها $3-n$ است. بنابراین با کسر این مقدار از فرمول بالا به معادله ۱۴-۳ می‌رسیم که بیشترین تعداد جفت ترکیب‌های مجاز این واحد است. بر اساس این فرمول برای مسئله ei151 تعداد ۲۳۰۴ جفت ترکیب آدرس مجاز وجود خواهد داشت.

معادله ۱۴-۳

$$\text{لذا } (n-3) \times (n-2) - (n-3) = (n-3)^2$$

لازم به ذکر است که اعداد تولید شده کاملاً تصادفی نیستند و برای تولید آنها از یک روش معروف که حجم سخت‌افزاری کمی را اشغال کرده، استفاده شده است. در اینجا از LFSR^۱ برای تولید اعداد نیمه تصادفی بر روی FPGA بهره گرفته شده است.

۱-۳-۵-۳-۳ شیفت رجیستر بازخورد خطی (LFSR)

شیفت رجیستر بازخورد خطی یا تولیدکننده تاوزورث^۲ توسط مارویاما^۳ مورد استفاده قرار گرفته است [۳۷]. آنها تولیدکننده را یک توالی m تایی یا توالی حداکثر^۴ در نظر گرفته‌اند. این بدان معنی است که تولیدکننده با طول $n-1$ ^۵ عدد تولید می‌نماید [۳۸]. به بیانی دیگر یک LFSR یک شیفت رجیستر است که بیت ورودی آن ترکیب خطی از حالت قبلی خود آن است. محتوای این ثبات در هر لبۀ کلاک پالس یک بیت به سمت راست جابه‌جا می‌شود. همچنین یک بازخورد از بیت‌های از قبل مشخص شده^۶، به با ارزشترین بیت به واسطه دریچه یا انحصاری^۷ یا دریچه نقیض یا انحصاری داده می‌شود. در زمان استفاده از دریچه نقیض یا انحصاری، یک بودن تمام بیت‌ها غیرمجاز شمرده شده یا به بیانی نباید مقدار اولیه ثبات باشد یا هرگز اتفاق نمی‌افتد. همین مورد در هنگام استفاده از دریچه یا انحصاری نیز برقرار است با این تفاوت که همه بیت‌ها نباید صفر باشند [۳۹]. در شکل ۳۲ یک نمونه LFSR ۳۲ بیتی نشان داده شده است [۳۸].

¹ Linear Feedback Shift Register

² Tauseworth

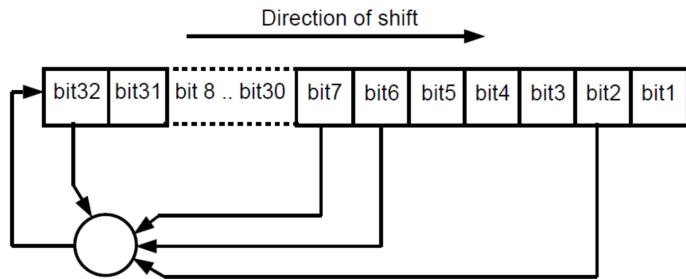
³ Maruyama

⁴ Maximal Sequence (M-Sequence)

⁵ Taps

⁶ XOR Gate

⁷ XNOR Gate



شکل ۳-۵۱ یک نمونه ۳۲ بیتی LFSR

به مقدار اولیه این شیفت رجیستر که شمارش از آن شروع می‌شود، دانه یا بذر^۱ گفته می‌شود و به دلیل آنکه عملکرد ثبات جبری است، مقادیر تولید شده کاملاً به وسیلهٔ حالت حال حاضر (حالت قبل) ثبات مشخص می‌شود. بعلاوهٔ چون ثبات دارای تعداد حالات ممکن کرانداری است، عاقبت وارد یک حلقهٔ تکرار خواهد شد. در هر صورت یک LFSR با تابع بازخوردی که به درستی انتخاب شده است، می‌تواند ردیفی از اعداد را تولید کند که به نظر تصادفی آمده و حلقهٔ تکرار مقادیر آن طولانی باشد [۴۰].

همچنین تولیدکنندگان FPGA طراحی‌های نمونه‌ای از LFSR‌ها را فراهم کردند تا به عنوان تولیدکننده‌های اتفاقی توالی، مورد استفاده قرار بگیرند. برای مثال یادداشت نحوهٔ کاربرد^۲ شرکت Xilinx [۳۹]. در این یادداشت، از دریچهٔ نقیص یا انحصاری استفاده شده است و TAP‌ها هم برای LFSR‌های سه تا ۱۶۸ بیتی مشخص شده‌اند. همچنین بازخورد به کم ارزشترین بیت داده شده و جایه‌جایی محتوای ثبات به سمت چپ است.

در پیاده‌سازی الگوریتم‌های فراتکاری بر روی FPGA به صورت گسترده‌ای از LFSR‌ها استفاده شده است. در پیاده‌سازی الگوریتم ABC پیوسته از LFSR ۲۰ بیتی [۴۱] ، الگوریتم ACO از ۱۶ بیتی [۴۲] و در پیاده‌سازی الگوریتم PSO از LFSR‌های ۱۶ بیتی [۴۳] ، ۲۰ بیتی [۴۴] و [۴۵] ، ۳۲ بیتی [۴۶] استفاده شده است. همچنین در دیگر مقالات پیاده‌سازی PSO، از LFSR‌هایی با طول نامشخص استفاده شده است ([۱۹]، [۲۱]، [۴۷]، [۴۸]، [۴۹]، [۵۰]، [۵۱]).

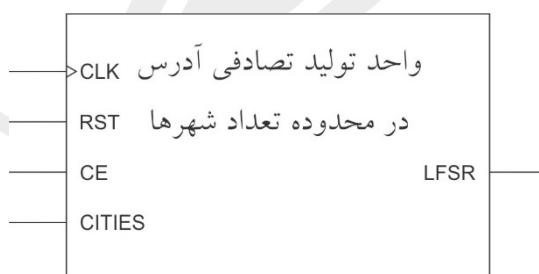
¹ Seed

² Application Note

۳-۵-۲-۳ واحد تولید تصادفی آدرس در محدوده تعداد شهرها

تولید آدرس نیمه تصادفی دو شهر، به عهده LFSR‌ها قرار داده شده است و در این پیاده‌سازی از یادداشت شرکت Xilinx بهره گرفته شده است که در آن از دریچه نقیض یای انحصاری و بازخورد به کم ارزشترین بیت استفاده می‌شود.

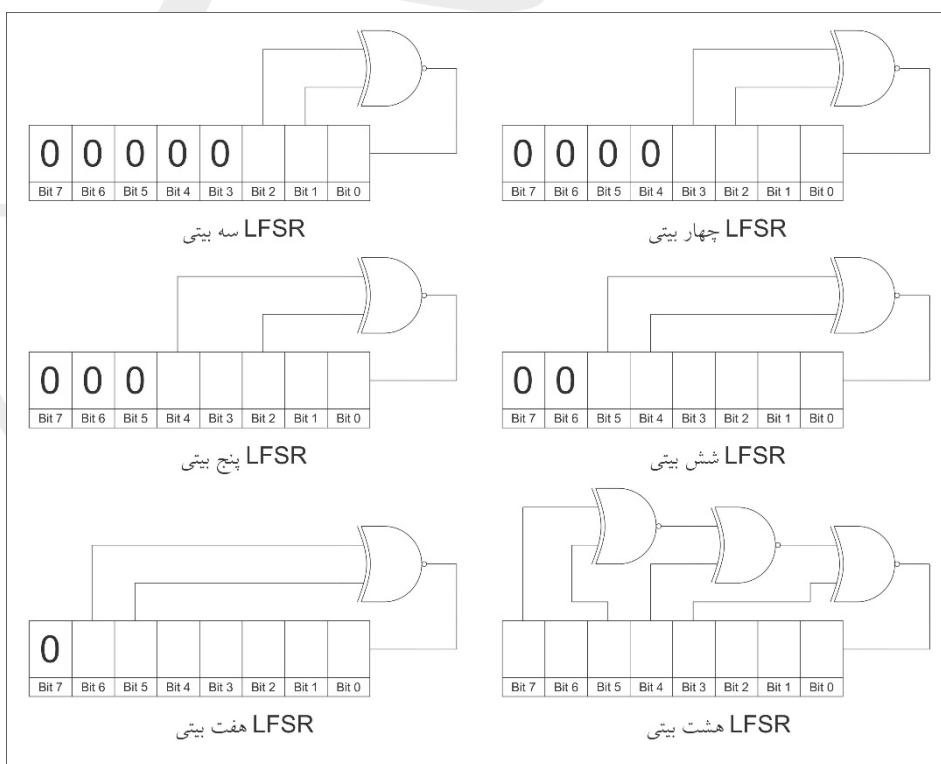
می‌دانیم که آدرس‌ها اعدادی بزرگتر مساوی صفر هستند، همچنین در قبل گفته شد که اگر طول آدرس‌دهی واحد مسیر حرکت یا واحد زنبور عسل را هشت بیت در نظر بگیریم، مسائلی با حداقل ۲۵۵ شهر را می‌توان مورد بررسی قرار داد؛ چراکه یک واحد از حافظه رم توزیع شده به عنوان متغیر میانی^۱ برای عملیات جابه‌جایی در نظر گرفته شده است. حال مشخص می‌شود که چرا از دریچه نقیض یای انحصاری استفاده کردیم؛ چراکه استفاده از این دریچه در ساخت یک LFSR هشت بیتی اعدادی بین صفر و ۲۵۴ را نتیجه می‌دهد (در قسمت قبل بیان شد که عدد ۲۵۵ مقدار غیرمجاز این تولید کننده اعداد نیمه تصادفی است). با این فرض که یک LFSR هشت بیتی داریم و به عنوان مثال مسئله eil51 را هم انتخاب کرده باشیم که ۵۱ شهر را داراست، این تولید کننده تعداد آدرس‌های زیادی را بر روی خروجی خود قرار می‌دهد که غیرمجاز هستند؛ یعنی بزرگتر از ۵۰ هستند. به منظور حل این مشکل هسته‌ای پیاده شد که این آدرس‌ها را در محدوده‌ای کمی بزرگتر یا مساوی تعداد شهرها در اختیار هسته‌ای تولید تصادفی آدرس دو شهر قرار می‌دهد. این واحد در شکل ۵۲-۳ نشان داده شده است.



شکل ۵۲-۳ واحد تولید تصادفی آدرس در محدوده تعداد شهرها

^۱ Buffer

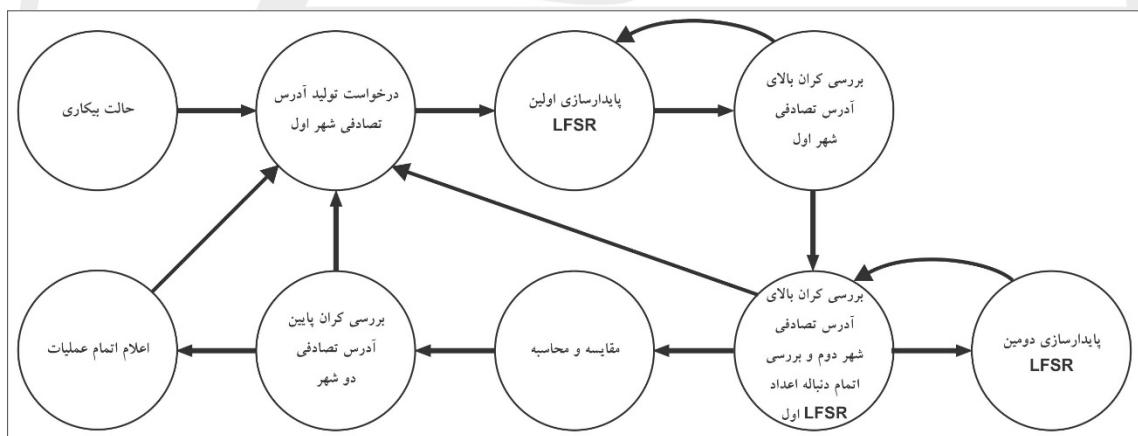
با هر بار دریافت سیگنال فعالسازی این واحد یک مقدار جدید بر روی خروجی خود قرار خواهد داد و با داشتن تعداد شهرها بر روی ورودی، مقادیر تولیدی LFSR محدود به تعداد بیت‌های لازم خواهد شد. به عنوان مثال برای مسئله ۵۱ شهر، LFSR شش بیتی در نظر گرفته می‌شود و خروجی LFSR عددی بین صفر تا ۶۲ خواهد بود که در نتیجه تعداد کلاک پالس‌های کمتری نسبت به LFSR هشت بیتی برای بررسی مجاز بودن خروجی این LFSR نیاز است. خروجی این هسته همواره یک عدد هشت بیتی است؛ ولیکن بیشینه این اعداد با بکار گرفتن LFSR مناسب محدود می‌شود. تنها محدودیتی که در این هسته وجود دارد، مربوط به مقدار اولیه یا بذر است، این مقدار باید عددی کمتر از ۷ باشد (سه بیت) که توالی اعدادی همانند یک LFSR معمولی با بیت‌های متناظر را نتیجه دهد. ساختار LFSR‌های استفاده شده به منظور محدود کردن خروجی واحد در شکل ۵۳-۳ نمایش داده شده است.



شکل ۵۳-۳ ساختار داخلی واحد تولید آدرس تصادفی در محدوده شهرها

۳-۳-۵ نمودار ماشین حالت واحد تولید تصادفی آدرس دو شهر

در این واحد از دو واحد تولید تصادفی آدرس در محدوده شهرها بهره گرفته شده است. نحوه عملکرد بدین گونه است که LFSR مربوط به شهر دوم یک مقدار را اختیار کرده و LFSR مربوط به شهر اول تمامی آدرس‌های ممکن را به صورت نیمه تصادفی اختیار می‌کند، سپس LFSR شهر دوم مقدار نیمه تصادفی بعدی را اختیار کرده و دوباره LFSR شهر اول تمامی مقادیر مجاز را اختیار می‌کند و این روند تا آخرین ترکیب ممکن دو شهر ادامه می‌یابد. سپس حلقة تکرار از اولین ترکیب دو شهر شروع می‌شود. همانطور که در قبل هم اشاره شد، این حلقة تکرار مثلاً برای مسئله eil51 بعد از ۲۳۰۴ ترکیب دو شهر تکرار می‌شود. در شکل ۳-۵، نمودار ماشین حالت این واحد آورده شده است. این نحوه پیاده‌سازی با دو واحد زیرمجموعه در زمانی که مرحله بهینه‌سازی محلی با چندین تکرار اجرا می‌شود و بهترین نتیجه حاصل در خروجی‌ها قرار می‌گیرد، مزیت به حساب می‌آید.



شکل ۳-۵ نمودار FSM واحد تولید تصادفی آدرس دو شهر

در این نمودار داریم:

- حالت بیکاری: در این حالت سیگنال فعال‌سازی بررسی می‌شود و در صورت دریافت این سیگنال به مرحله بعد منتقل شده و عملیات تولید دو آدرس نیمه تصادفی شروع می‌شود.
- درخواست تولید آدرس تصادفی شهر اول: سیگنال فعال‌سازی به LFSR مربوط به آدرس شهر اول فرستاده می‌شود؛ به عبارتی درخواست آدرس شهر جدیدی ارسال و وارد مرحله بعد می‌شود.

- پایدارسازی اولین LFSR: خروجی واحد حاوی LFSR یک کلاک پالس بعد از دریافت سیگنال فعال‌سازی تغییر می‌کند؛ به همین دلیل به یک کلاک پالس تاخیر نیاز است که با این حالت تامین می‌شود.
- بررسی کران بالای آدرس تصادفی شهر اول: در این مرحله شرط کران بالای آدرس شهر اول بررسی شده و در صورتی که این شرط برآورده نشود، درخواست تولید آدرس جدیدی به تولید کننده متناظر آن داده شده و با استفاده از تاخیر مرحله قبل، حلقه‌ای را ایجاد کرده که تا پیدا شدن آدرسی مجاز ادامه خواهد یافت.
- بررسی کران بالای آدرس تصادفی شهر دوم و بررسی اتمام دنباله اعداد LFSR اول: در این حالت شرط کران بالای آدرس شهر دوم بررسی می‌شود و در صورتی که در محدوده مجاز نباشد، درخواست تولید آدرس جدیدی به تولیدکننده متناظر آن داده می‌شود و به مرحله پایدارسازی دومین LFSR منتقل می‌شود. همانطور که بیان شد، برای دریافت خروجی‌های صحیح از واحد تولید اعداد تصادفی، بعد از درخواست مقدار جدید به یک کلاک پالس تاخیر نیاز است که بدین منظور به مرحله پایدارسازی دومین LFSR رفته و پس از بازگشت کران بالای این آدرس جدید، مجدداً بررسی می‌شود و در صورت غیرمجاز بودن، درخواست تولید مجدد داده خواهد شد و دوباره به مرحله پایدارسازی دومین LFSR باز می‌گردد. این روند تا انتخاب یک آدرس در محدوده مجاز ادامه خواهد داشت. اگر شرط قبلی برقرار بود، آدرس شهر اول بررسی می‌شود و اگر به مقدار صفر رسیده باشد، یعنی LFSR متناظر شهر اول حلقة تکرار بعدی را شروع کرده باشد، دوباره درخواست تولید آدرس جدیدی برای شهر دوم را به تولید کننده متناظر آن ارسال خواهد کرد و به مرحله درخواست تولید آدرس تصادفی شهر اول منتقل خواهد شد. به عبارتی وقتی که حلقة تکرار اعداد برای آدرس شهر اول به اتمام رسیده باشد (مقدار صفر را اختیار کرده باشد)، الگوریتم در ابتدا آدرس جدیدی را برای شهر دوم درخواست می‌دهد و سپس با بازگشت به نقطه شروع، درخواست آدرس جدیدی برای شهر اول را می‌دهد. بدلیل استفاده از آدرس صفرم برای تشخیص اتمام دنباله

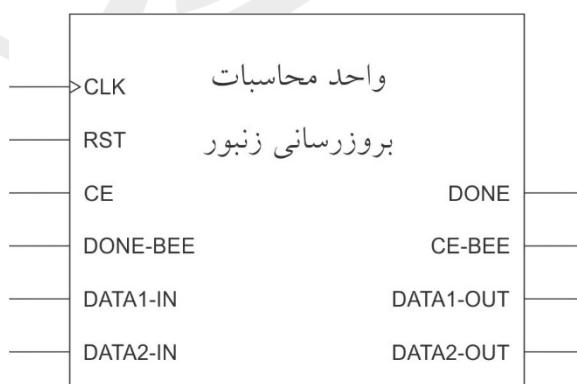
یکی از آدرس‌های تصادفی، آدرس صفرم در ترکیب دو شهر فقط یکبار تکرار می‌شود؛ یعنی جابه‌جایی دو شهر که یکی از آن‌ها آدرس صفرم را اختیار کرده باشد، در مجموعه جفت ترکیب‌های آدرس دو شهر در نظر گرفته نمی‌شود. لازم به ذکر است که اگر یکی از این دو آدرس مقدار صفر را اختیار کند، کفايت می‌کند. یعنی حذف انتخاب صفر برای یکی از شهرها، حالتی از ترکیب دو آدرس شهر را حذف نمی‌کند و فقط ترکیب دو شهر در دنباله یکبار تکرار می‌شود. در صورتی که آدرس تولید شده دارای دو شرط قبل باشد، به مرحله مقایسه و محاسبه منتقل خواهد شد.

- پایدارسازی دومین LFSR: عملکرد این مرحله همانند مرحله مشابه قبلی است و تنها یک کلک پالس تاخیر ایجاد می‌کند.
- مقایسه و محاسبه: در این مرحله به منظور بررسی شرط کران پایین، قدر مطلق اختلاف آدرس تصادفی دو شهر محاسبه می‌شود.
- بررسی کران پایین آدرس تصادفی دو شهر: همانطور که از اسم مرحله مشخص است در اینجا شرط کران پایین بررسی می‌شود؛ یعنی تفاضل دو آدرس باید بزرگتر یا مساوی ۳ باشد. در صورتی که این شرط برقرار نباشد به مرحله درخواست تولید آدرس تصادفی شهر اول بازگشته و الگوریتم تا پیدا کردن دو آدرس مجاز (شروط کران بالا و کران پایین) تکرار می‌شود. اگر شرط کران پایین برقرار بود، مقدار کوچکتر را روی خروجی متناظر با R_1 و مقدار بزرگتر را روی خروجی متناظر با R_2 قرار می‌دهد و به مرحله بعد می‌رود.
- اعلام اتمام عملیات: خروجی DONE را یک کرده و همچنین سیگنال فعالسازی را نیز بررسی می‌کند. این هسته تا درخواست دو عدد تصادفی بعدی (دريافت سیگنال فعالسازی بعدی) مقادیر دو آدرس تصادفی را روی خروجی‌های خود نگه داشته و همچنین مقدار خروجی DONE را هم یک نگاه می‌دارد. به شرطی که سیگنال فعالسازی دریافت شده باشد، مقدار خروجی DONE صفر شده و به مرحله درخواست تولید آدرس تصادفی شهر اول منتقل می‌شود.

پس از دریافت سیگنال فعال‌سازی (لبه بالاروند سیگنال) توسط واحد تولید تصادفی آدرس دو شهر، حداقل ۷ کلاک پالس برای قرار دادن دو آدرس بر روی خروجی و نتیجتاً ۸ کلاک پالس تا تحریک خروجی DONE زمان نیاز است. حال لزوم کارکرد موازی این واحد به صورت مجرزا مشخص خواهد شد.

۶-۳-۳ واحد محاسبات بروزرسانی زنبور

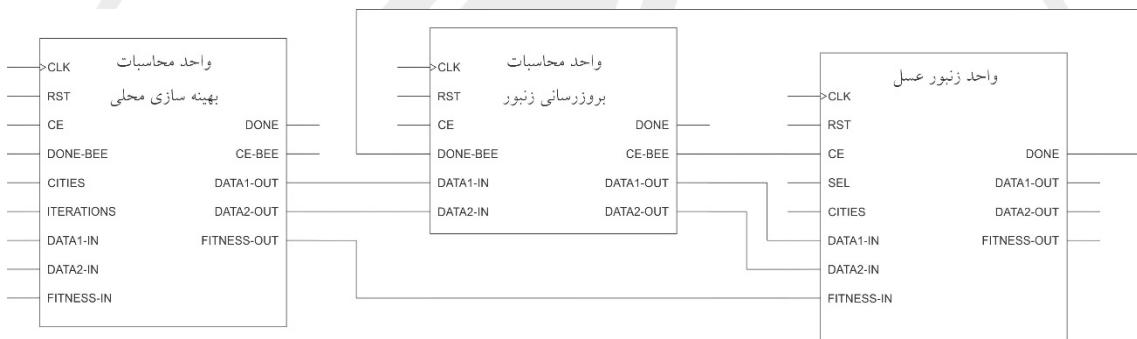
هنگامی که مرحله بهینه‌سازی محلی با نائل شدن به نتایج بهتری خاتمه می‌یابد و قبل از اینکه تکرار بعدی شروع شود، باید اطلاعات ذخیره شده روی زنبور اعم از مسیر حرکت و مسافت طی شده بروز شود؛ این کار وظیفه واحد بروزرسانی زنبور است. ورودی و خروجی‌های این واحد در شکل ۵۵-۳ نمایش داده شده است. نحوه عملکرد این واحد بدین صورت است که دو مقدار آدرس شهر اول و آخر را از واحد بهینه‌سازی محلی به ترتیب بر روی ورودی‌های DATA1-IN و DATA2-IN و دریافت کرده و در صورتی که نتیجه بهتری حاصل شده باشد، تکه مسیر بین این دو شهر را دوران می‌دهد. بدین منظور آدرس دو شهری که باید با هم تعویض شوند را بر روی خروجی‌های DATA1-OUT و DATA2-OUT قرار داده که به ورودی‌های واحد زنبور متصل‌اند. سپس یک سیگنال فعال‌سازی با استفاده از خروجی CE-BEE به زنبور ارسال می‌کند و تا اتمام کار زنبور متظر شده و بعد درخواست جایه‌جایی دو شهر بعد را می‌دهد. این روند تا دوران کامل تکه مسیر بین دو آدرس R_1 و R_2 ادامه می‌یابد و سپس سیگنال اتمام بروزرسانی را بر روی خروجی DONE قرار می‌دهد.



شکل ۵۵-۳ واحد محاسبات بروزرسانی زنبور

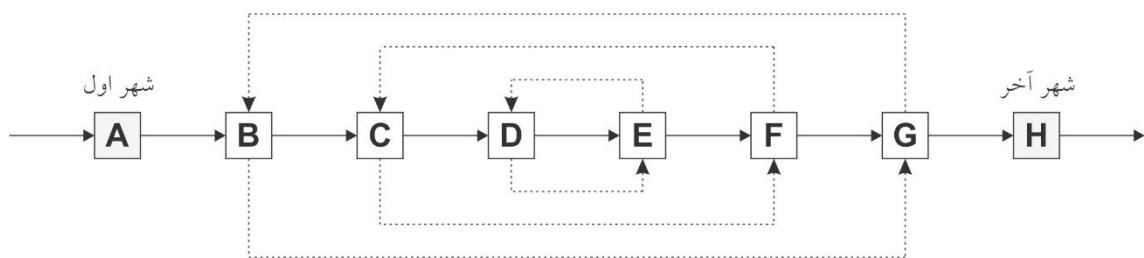
قبل‌گفته شد که این واحد برای بررسی اینکه آیا زنبور متناظر نیاز به بروزرسانی دارد یا به عبارتی واحد بهینه‌سازی محلی به نتیجه بهتری رسیده است یا نه، مقدار آدرس R_2 را خوانده و اگر صفر بود یعنی احتیاجی به تغییر نیست. در این شرایط پیغام اتمام کار را بر روی خروجی خود قرار می‌دهد که مشخص می‌کند، زنبور متناظر آماده طی مراحل بعدی است.

نحوه اتصال سه واحد زنبور، محاسبات بهینه‌سازی محلی و محاسبات بروزرسانی به شکل ۵۶-۳ است. مشاهده می‌شود که خروجی FITNESS-OUT واحد بهینه‌سازی مستقیماً به ورودی متناظر واحد زنبور متصل شده است.

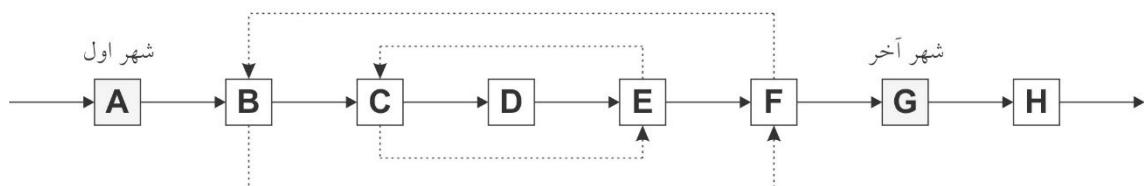


شکل ۵۶-۳ نحوه اتصال واحدهای زنبور، محاسبات بهینه‌سازی محلی و محاسبات بروزرسانی

کنترل این هسته در سطح رفتار و با ماشین حالت انجام شده است. الگوریتم کلی که برای این ماشین حالت در نظر گرفته شده است بدین صورت است که با داشتن آدرس شهر اول و آخر (R_2, R_1), برای دوران تکه مسیر در هر تکرار باید دو شهر جابه‌جا شوند. همانند شکل ۵۷-۳ در تکرار اول شهر اول تکه مسیر و شهر آخر تکه مسیر انتخاب شده و جای این دو عوض می‌شود. سپس شهر دوم تکه مسیر و شهر یکی مانده به آخر تکه مسیر جابه‌جا می‌شوند. این روند تا جایی ادامه می‌یابد که تکه مسیر به طور کامل دوران پیدا کرده باشد.



نحوه دوران تکه مسیری با تعداد شهرهای زوج



نحوه دوران تکه مسیری با تعداد شهرهای فرد

شکل ۳-۵۷ نحوه دوران تکه مسیر

تعداد دورانها از معادله ۱۵-۳ محاسبه می‌شود:

$$ITERATIONS < \frac{ADDRESS_{LAST\ CITY} - ADDRESS_{FIRST\ CITY}}{2} \quad \text{معادله ۱۵-۳}$$

در شکل ۳-۵۷ اگر شهر A را در آدرس ۱ در نظر بگیریم، شهر H در آدرس ۸ و شهر G در آدرس ۷

است، پس داریم:

$$ITERATION < \frac{8-1}{2} = 3.5 \Rightarrow ITERATIONS = 3 \quad \text{تکه مسیر با تعداد شهرهای زوج}$$

$$ITERATION < \frac{7-1}{2} = 3 \Rightarrow ITERATIONS = 2 \quad \text{تکه مسیر با تعداد شهرهای فرد}$$

دو راه برای تشخیص تعداد تکرارهای لازم وجود دارد:

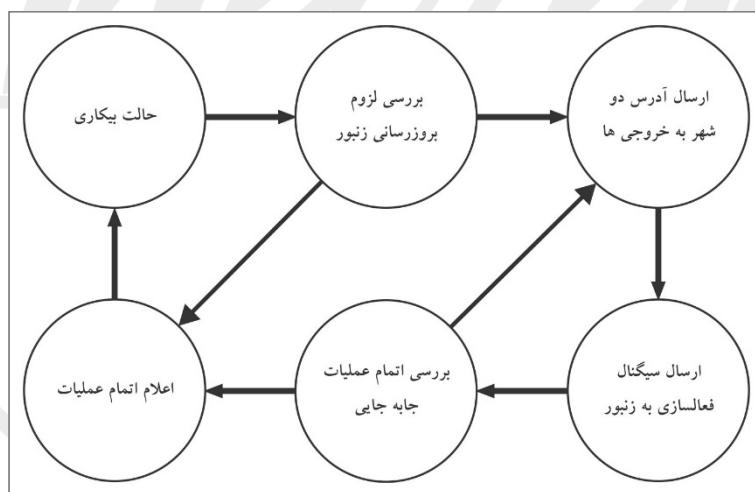
- اولین راه استفاده از فرمول ۱۵-۳ مناسب با پیاده‌سازی روی سخت‌افزار است. بدین صورت

که در ابتدا اختلاف آدرس دو شهر R_1 و R_2 محاسبه می‌شود. در صورتی که این اختلاف فرد بود، یعنی تعداد شهرهای تکه مسیر زوج باشد، آن را تقسیم بر دو کرده (شیفت به راست یک بیتی) که نتیجه این شیفت مقدار اعشار را نیز حذف کرده و جواب نهایی را نتیجه می-

دهد. اگر اختلاف دو آدرس زوج بود، یعنی تکه مسیر دارای تعداد فردی شهر است، تعداد تکرار مساوی حاصل شیفت به راست منهای یک است.

- دو مین راه چشم‌پوشی از فرمول و استفاده از شرط دیگری است. بدین صورت که در هر مرحله از تکرار الگوریتم، اختلاف آدرس آخرین دو شهری که با هم تعویض شده‌اند، محاسبه می‌شود. اگر این اختلاف کمتر یا مساوی دو باشد، فرآیند جابه‌جایی به اتمام رسیده است. این شرط بدان معنی است که بین دو شهری که تعویض شده‌اند، یا فقط یک شهر مانده و یا اینکه به هم چسبیده‌اند؛ در نتیجه نیاز به دوران جدیدی نیست.

فارغ از اینکه کدامیک از این دو روش استفاده شود، نمودار FSM آن‌ها شبیه به هم است. چراکه محاسبه تعداد دوران‌ها و یا تفاضل آدرس آخرین دو شهر تعویض شده، قبل از ورود به حالت‌های الگوریتم انجام می‌شود. نمودار FSM این واحد به همراه توضیحات مربوطه در شکل ۵۸-۳ مشاهده می‌شود:



شکل ۵۸-۳ نمودار FSM واحد محاسبات بروزرسانی زنیور

- حالت پیکاری: در این مرحله سیگنال فعالسازی بررسی می‌شود و در صورتی که دریافت شده باشد، برای شروع عملیات به مرحله بعد منتقل می‌شود.

- بررسی لزوم بروزرسانی زنبور: در این مرحله، آدرس R_2 بررسی می‌شود. اگر این آدرس صفر بود، به حالت اعلام اتمام عملیات انتقال می‌یابد. در غیراینصورت جواب بهتری پیدا شده است و برای شروع فرآیند جابه‌جایی به حالت بعد منتقل می‌شود.
- ارسال آدرس دو شهر به خروجی‌ها: در این حالت آدرس دو شهری که باید با هم جابه‌جا شوند بر روی خروجی‌ها قرار می‌گیرند.
- ارسال سیگنال فعال‌سازی به زنبور: قبل‌اً بیان شد که هسته اصلی زنبورها را در زمان بروزرسانی در حالت جابه‌جایی محتوای آدرس ورودی‌ها قرار می‌دهد. بنابرانی در این حالت با ارسال سیگنال فعال‌سازی به زنبور متناظر، روند جابه‌جایی دو شهر آغاز می‌شود.
- بررسی اتمام عملیات جابه‌جایی: در این مرحله، سیگنال اتمام عملیات زنبور بررسی می‌شود. در صورت اتمام، یکی از شروط اتمام تعداد دوران‌ها یا تفاضل آدرس آخرین دو شهر تعویض شده، بررسی می‌شود. اگر به بیشینه تعداد دوران رسیده باشد و یا اختلاف آدرس‌ها کوچکتر و یا مساوی دو باشد، به مرحله اعلام اتمام عملیات منتقل می‌شود. در غیراینصورت برای شروع جابه‌جایی جدید به دو حالت قبل یعنی ارسال آدرس دو شهر به خروجی‌ها بازخواهد گشت.
- اعلام اتمام عملیات: این حالت سیگنال خروجی DONE را یک کرده و سیگنال فعال‌سازی را بررسی می‌کند. اگر این سیگنال بازنشانی شود، به حالت بیکاری منتقل می‌شود. به عبارتی تا زمان بازنشانی سیگنال فعال‌سازی، خروجی DONE را یک نگه می‌دارد. چراکه اگر چندین واحد بروزرسانی در یک محدوده زمانی سیگنال فعال‌سازی دریافت کنند، به دلیل تفاوت در تعداد دوران‌ها، زمان اتمام عملیات آن‌ها متفاوت خواهد بود. در نتیجه برای تشخیص پایان کار تمام این واحدها باید خروجی اتمام عملیات آن‌ها تا زمان دلخواه بالا (یک منطقی) نگه داشته شود.

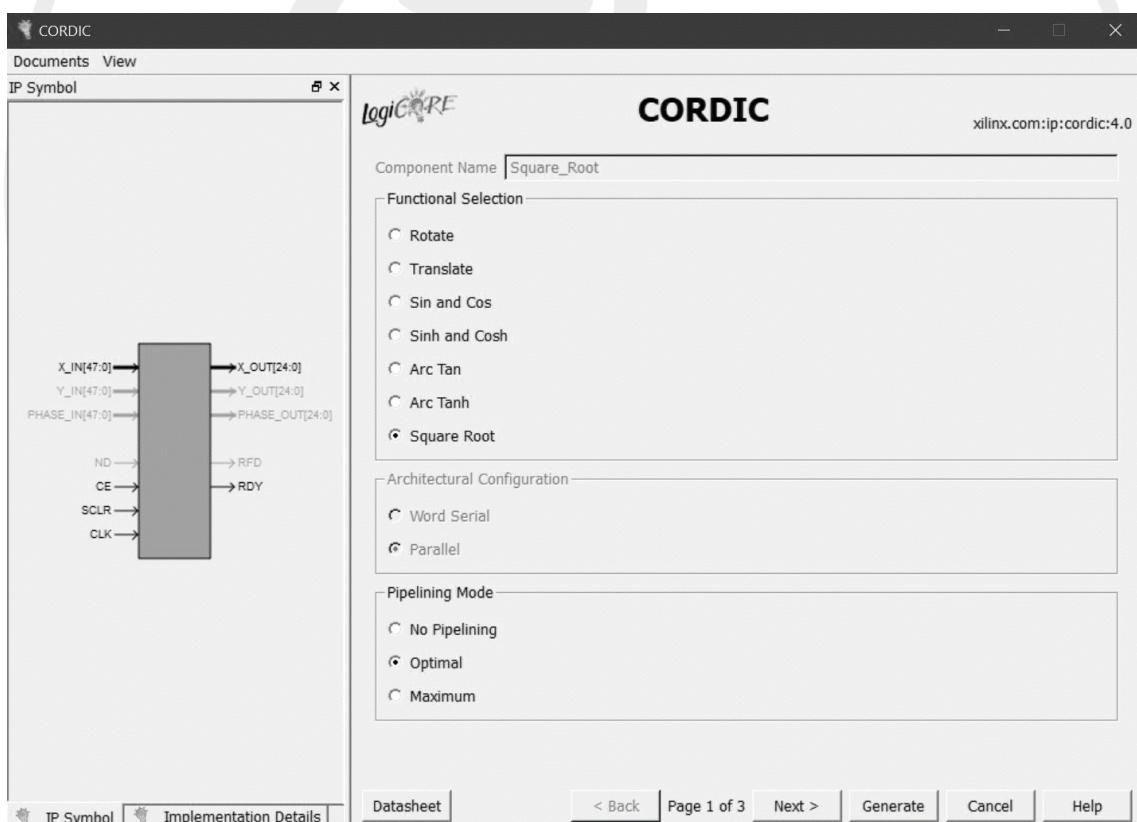
۷-۳-۳ کاهش حجم بلوک‌های رم استفاده شده

در این پیاده‌سازی از ماتریس فاصله‌ها برای نگهداری اطلاعات مربوط به فاصله شهرها استفاده شده است که این اطلاعات بر روی بلوک‌های رم FPGA نگاشت می‌شود. در قسمت‌های قبلی بیان شد که روش دیگر پیاده‌سازی آنست که مختصات طولی و عرضی شهرها را روی بلوک‌های رم ذخیره کرد و هر مرتبه فاصله اقلیدسی دو شهر را محاسبه کرد. همچنین معاوی این روش در مقایسه با روش استفاده شده ذکر شد. ولیکن به عنوان یک روش جایگزین می‌توان از ترکیب این دو روش استفاده کرد؛ بدین ترتیب که در کنار FPGA از رم خارجی بهره جست. در این روش محاسبات فواصل ماتریس فاصله‌ها در ابتدای الگوریتم انجام می‌شود و در آدرس‌های مناسب بر روی رم خارجی ذخیره می‌شود. حال از ماتریس فاصله‌های ایجاد شده در رم خارجی می‌توان در طول روند الگوریتم استفاده کرد. مزیت این روش نسبت به دو روش قبلی آنست که حجم بسیار کمتری از بلوک‌های رم تعییه شده در FPGA که محدود نیز هستند، استفاده می‌شود. در نتیجه در ازای صرف منابع منطقی بیشتر روی FPGA و زمان بیشتر برای اجرای الگوریتم، امکان کوچکسازی این هسته مهیا می‌شود. این بدان معنی است که می‌توان از این هسته در کنار هسته‌های بیشتری در یک FPGA بهره جست. لازم به توجه است که اگرچه حجم عناصر منطقی بیشتری در این روش مصرف می‌شود؛ ولی با داشتن یک واحد تولیدکننده ماتریس فاصله‌ها، می‌توان به سرعت و تعداد دلخواه ماتریس فاصله‌ها را ایجاد کرد. این ماتریس‌ها می‌توانند روی یک و یا چند رم خارجی قرار بگیرند. در این روش محتوای رم (فاصله اقلیدسی) تنها یکبار محاسبه شده و فقط در آدرس‌های متفاوت از حافظه ذخیره می‌شود.

دقت شود که از مطالب این قسمت در پیاده‌سازی نهایی الگوریتم HBA استفاده نشده است و تنها این طراحی برای شرایطی ارائه می‌شود که محدودیتی برای استفاده از بلوک‌های رم داخلی FPGA وجود داشته و امکان بکارگیری رم خارجی نیز وجود دارد. لازم به ذکر است که اکثر قریب به اتفاق سخت‌افزارهای حاوی FPGA با وجود یک یا چند رم خارجی طراحی و تولید می‌شوند که مزیت یا ضرورت ارائه این روش را مشهود می‌کند. حال به بیان نحوه پیاده‌سازی این طرح خواهیم پرداخت.

۱-۷-۳-۳ واحد محاسبه‌گر جذر^۱

می‌دانیم که برای محاسبه فاصله اقلیدسی به عملیات جذر نیاز است. این واحد نیازی به پیاده‌سازی ندارد و می‌توان از هسته‌های دارایی فکری^۲ موجود بهره جست. این هسته‌ها توسط تولیدکنندگان FPGA و تولیدکنندگان فرعی^۳ ارائه می‌شوند. تولیدکنندگان FPGA این هسته‌های IP را در نرم-افزارهای IDE^۴ خود ضمیمه می‌کنند که تعدادی از آن‌ها رایگان است. هسته محاسبه‌گر جذر از جمله این هسته‌ها است که به صورت رایگان در اختیار توسعه‌دهندگان قرار داده شده است. این هسته‌ها در قالب مولدهای هسته IP ارائه می‌شوند که به واسطه این مولدها می‌توان تغییراتی در هسته مورد نظر ایجاد کرد؛ به عبارتی این هسته‌ها را بسته به نیاز توسعه‌دهندگان سفارشی کرد. در شکل ۵۹-۳ و ۶۰-۳ به ترتیب مولد هسته IP محاسبه‌گر جذر شرکت Xilinx و Altera نمایش داده شده است.



شکل ۵۹-۳ مولد هسته محاسبه‌گر جذر شرکت Xilinx

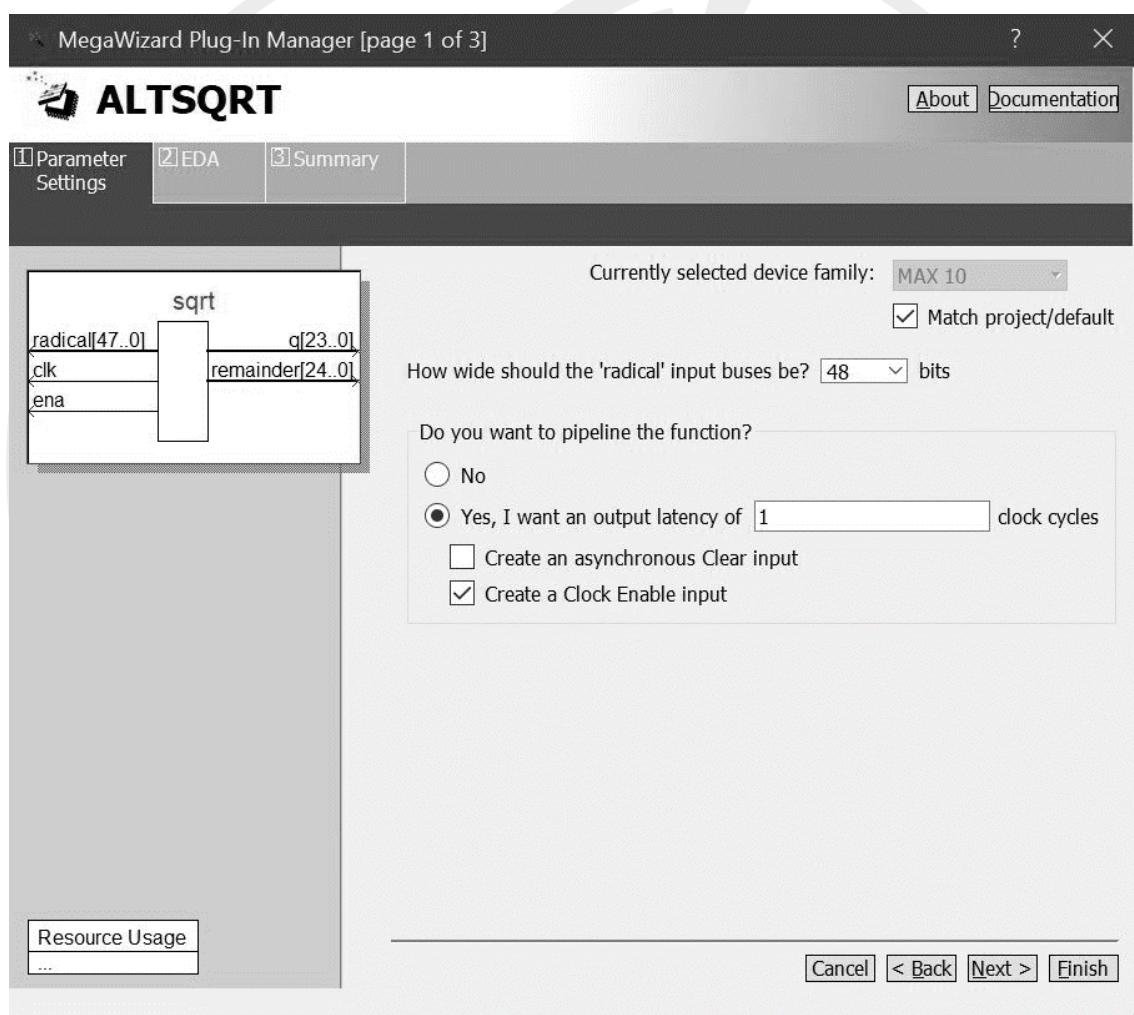
^۱ Square Root

^۲ IP Core (Intellectual Property Core)

^۳ Third Parties

^۴ Integrated Development Environment

هسته استفاده شده شرکت Xilinx دارای چهار ورودی و دو خروجی است. این هسته عدد زیر رادیکال را در قالب یک متغیر ۴۸ بیتی (X_IN) دریافت کرده و جذر آن را به صورت یک متغیر صحیح ۲۵ بیتی (X_OUT) روی خروجی قرار می‌دهد. یکی از چهار ورودی آن به کلک پالس اختصاص یافته است. به منظور کنترل این واحد ورودی CE به سیگنال فعال‌سازی و ورودی SCLR^۱ به بازنشانی همزمان اختصاص یافته است. همچنین خروجی RDY^۲ اتمام عملیات را اعلام می‌کند.



شکل ۳-۶۰ مولد هسته محاسبه‌گر جذر شرکت Altera

^۱ Synchronous Clear

^۲ Ready

هسته استفاده شده شرکت Altera دارای سه ورودی و دو خروجی است. این هسته عدد زیر رادیکال ۲۴ را در قالب یک متغیر ۴۸ بیتی (radical) دریافت کرده و جذر آن را به صورت یک عدد صحیح ۹ بیتی روی خروجی remainder هم باقیمانده نتیجه محاسبات جذر است؛ به عبارتی مقداری که اگر رادیکال آن محاسبه شود، قسمت اعشاری جذر را نتیجه می‌دهد. یکی از سه ورودی این واحد به کلاک پالس اختصاص یافته است. همچنین به منظور کنترل این واحد ورودی ena هم به سیگنال فعال‌سازی اختصاص یافته است.

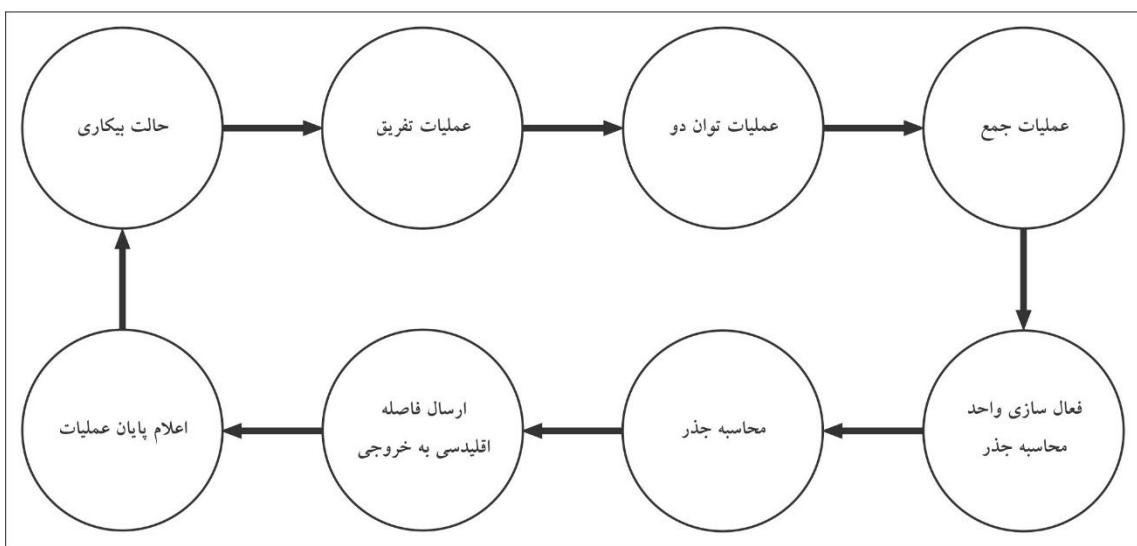
۲-۷-۳-۳ واحد محاسبه‌گر فاصله اقلیدسی

حال که هسته IP محاسبه‌گر جذر توضیح داده شد، به توضیح هسته محاسبه‌گر فاصله اقلیدسی می‌رسیم. این هسته برای هر دو تولیدکننده بزرگ FPGA ارائه شده است. در شکل ۶۱-۳ ورودی و خروجی‌های این واحد به تصویر کشیده شده است.



شکل ۶۱-۳ واحد محاسبه‌گر فاصله اقلیدسی

پس از دریافت سیگنال فعال‌سازی، این واحد مجبور دو مقداری که مختصات طولی و عرضی آنها روی ورودی‌های آن قرار گرفته است را روی خروجی خود قرار می‌دهد و یک کلاک پالس بعد خروجی DONE را به یک تغییر داده و در این حالت باقی می‌ماند تا سیگنال فعال‌سازی بازنشانی شود. در شکل ۶۲-۳ نمودار ماشین حالت پیاده‌سازی این الگوریتم برای FPGA‌های شرکت Xilinx نمایش داده شده است.



شکل ۳-۶۲ نمودار ماشین حالت واحد محاسبه‌گر فاصله اقلیدسی برای FPGA‌های شرکت Xilinx

در این نمودار داریم:

- حالت بیکاری: در این مرحله سیگنال فعال‌سازی بررسی می‌شود. در صورت دریافت آن، محاسبه فاصله اقلیدسی ($Euclidean\ Dist = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$) شروع می‌شود.
- عملیات تفریق: تفاضل دو مختصات طولی و عرضی محاسبه می‌شود.
- عملیات توان دو: مربع دو تفاضل محاسبه می‌شود.
- عملیات جمع: مربع‌های محاسبه شده با هم جمع می‌شوند.
- فعال‌سازی واحد محاسبه‌گر جذر: در این مرحله سیگنال اتمام عملیات^۱ واحد محاسبه‌گر جذر بررسی می‌شود؛ اگر این سیگنال بازنشانی شده باشد، سیگنال فعال‌سازی^۲ به این واحد ارسال می‌شود. دلیل این بررسی آنست که سیگنال اتمام عملیات واحد محاسبه‌گر جذر خروجی به ورودی بازنشانی همزمان^۳، بازنشانی شود.

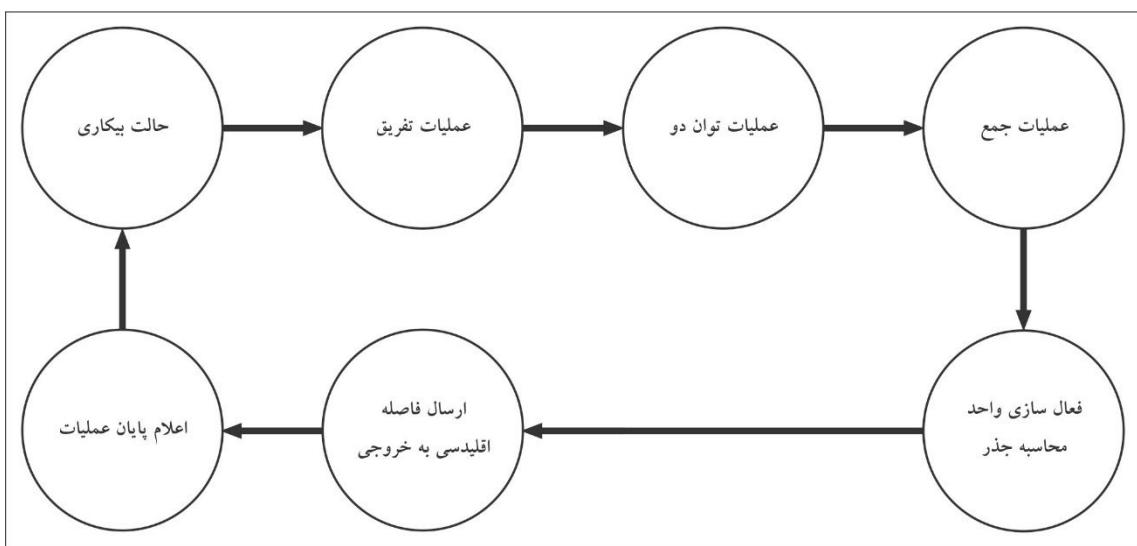
¹ RDY

² CE

³ SCLR

- محاسبه جذر: در این حالت خروجی تمام عملیات واحد محاسبه‌گر جذر بررسی می‌شود. هدف از این کار تشخیص آماده بودن مقدار مجدور است. در صورت آماده بودن به مرحله بعد منتقل می‌شود.
- ارسال فاصله اقلیدسی به خروجی: در این پیاده‌سازی طول یا دقت متغیر فاصله ۳۲ بیت در نظر گرفته شده است. در این مرحله مقدار مجدور محاسبه شده که یک عدد ۲۵ بیتی است در قالب یک عدد ۳۲ بیتی به خروجی واحد ارسال می‌شود. توجه شود که چون از هسته‌های IP موجود استفاده شده است، امکان سفارشی کردن ۱۰۰ درصدی وجود ندارد و باید در محدوده گزینه‌هایی که مولد هسته در اختیار توسعه‌دهنده قرار می‌دهد، انتخاب کرد.
- اعلام پایان عملیات: در این حالت سیگنال تمام عملیات به خروجی ارسال می‌شود و سیگنال فعال‌سازی واحد هم بررسی می‌شود. اگر سیگنال فعال‌سازی همچنان برقرار باشد، مقدار مجدور و سیگنال تمام کار را روی خروجی‌های واحد نگه می‌دارد. در صورت بازنشانی سیگنال فعال‌سازی و با استفاده از ورودی بازنشانی همزمان واحد محاسبه‌گر جذر، خروجی مربوط به مقدار جذر را بازنشانی می‌کند و به مرحله بیکاری منتقل می‌شود. لازم به ذکر است که برای آنکه سیگنال بازنشانی همزمان درست عمل کند، باید واحد محاسبه‌گر جذر فعال نگه داشته شود؛ به عبارتی پایه CE این واحد باید بالا نگه داشته شود.

با نحوه عملکرد این واحد برای FPGA‌های شرکت Xilinx آشنا شدیم. در شکل ۶۳-۳ نمودار ماشین حالت واحد محاسبه‌گر فاصله اقلیدسی برای FPGA‌های شرکت Altera آورده شده است.



شکل ۶۳-۳ نمودار ماشین حالت واحد محاسبه‌گر فاصله اقلیدسی برای FPGA‌های شرکت Altera

طراحی این واحد بسیار شبیه واحد مشابه آن برای شرکت Xilinx است با چند تفاوت جزیی. اول آنکه در مرحله فعال‌سازی واحد محاسبه جذر هیچ سیگنالی بررسی نمی‌شود و تنها سیگنال فعال‌سازی برای واحد محاسبه‌گر جذر ارسال می‌شود. دوم آنکه حالت محاسبه جذر حذف شده است؛ چراکه یک کلاک پالس پس از ارسال سیگنال فعال‌سازی، نتیجه جذر روی خروجی واحد محاسبه‌گر جذر قرار می‌گیرد و نیازی به بررسی سیگنال اتمام عملیات نیست. سوم آنکه در مرحله اعلام پایان عملیات، خروجی‌های واحد بازنمانی نمی‌شوند و فقط در صورت بازنمانی سیگنال فعال‌سازی، به حالت بیکاری منتقل می‌شود. در حالت بیکاری سیگنال فعال‌سازی برای واحد محاسبه‌گر جذر ارسال نمی‌شود که خود باعث بازنمانی خروجی این واحد می‌شود.

۳-۷-۳ واحد تولیدکننده آدرس و محتوای ماتریس فاصله‌ها

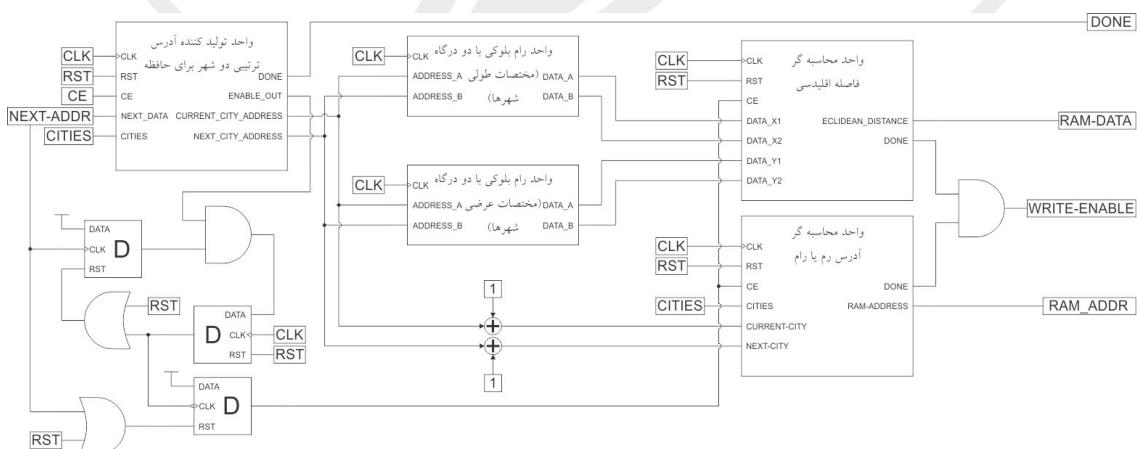
اکنون که نحوه محاسبه فاصله اقلیدسی بیان شد، به واحدی می‌پردازیم که با داشتن مختصات طولی و عرضی شهرها، فاصله اقلیدسی و آدرس متناظر آن بر روی حافظه را محاسبه کند. از این واحد برای تولید ماتریس فاصله‌ها به روشی که در قبل بیان شد، استفاده می‌شود. به عبارتی تنها قسمت بالامثلی ماتریس فاصله‌ها بر روی حافظه ذخیره می‌شود. ورودی و خروجی‌های این واحد در شکل ۶۴-۳ نمایش داده شده است.



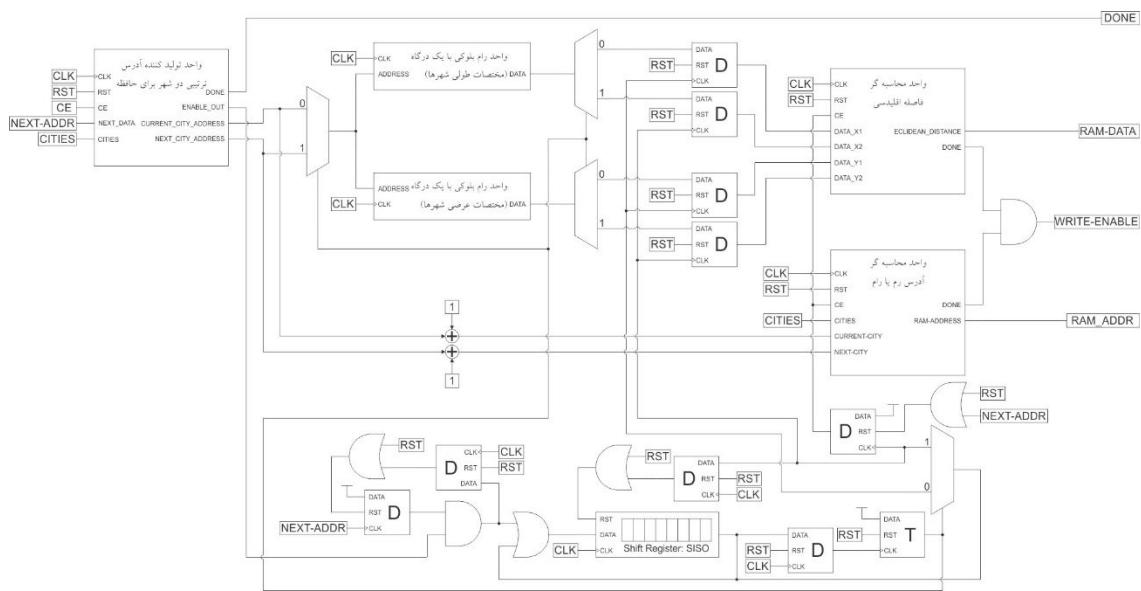
شکل ۶۴-۳ واحد تولید کننده آدرس و محتوای ماتریس فاصله ها

پس از فعال سازی، این واحد با هر بار دریافت سیگنال از ورودی NEXT_ADDR یک عنصر از ماتریس فاصله ها را به همراه آدرس متناظر آن روی حافظه تولید کرده و روی خروجی های خود قرار می دهد. سپس یک سیگنال به خروجی WRITE_ENABLE ارسال می کند که معادل اتمام محاسبات برای یک سلول از حافظه است. از این سیگنال می توان به عنوان سیگنال اجازه نوشتن روی حافظه هم استفاده کرد. همچنین پس از محاسبه تمام آدرس ها و فاصله های متناظر آنها، سیگنال اتمام عملیات را روی خروجی DONE قرار خواهد داد.

این هسته به دو صورت و در سطح ساختار پیاده شده است. این ساختار در اولین روش با استفاده از رام دو درگاهه و در دومین با بهره از رام تک درگاهه پیاده شده است. روش اول و دوم به ترتیب در شکل ۶۵-۳ و ۶۶-۳ به تصویر کشیده شده اند.



شکل ۶۵-۳ ساختار واحد تولید کننده آدرس و محتوای ماتریس فاصله ها با بهره از رام دو درگاهه



شکل ۶۶-۳ ساختار واحد تولیدکننده آدرس و محتوای ماتریس فاصله‌ها با بهره از رام تک درگاهه

هر دو ساختار در کنار عناصر پایه شامل چهار زیر واحد می‌شوند:

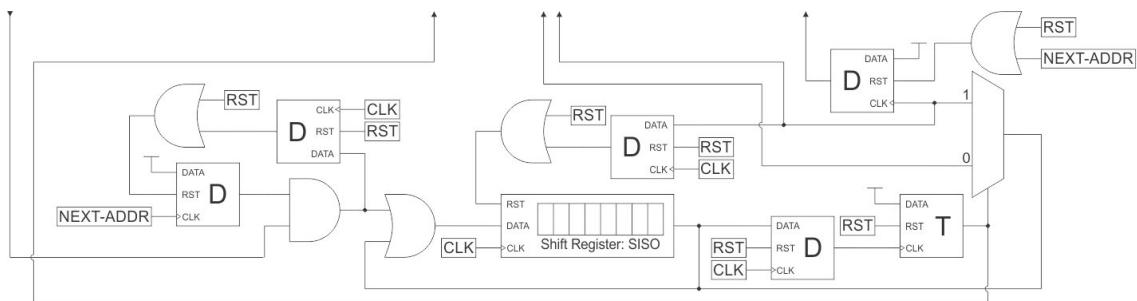
- واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه
- واحد رام بلوکی تک درگاهه / دو درگاهه
- واحد محاسبه‌گر فاصله اقلیدسی
- واحد محاسبه‌گر آدرس رم یا رام

جز زیر واحد اول بقیه زیر واحدها در مباحث قبل بررسی شده‌اند و در قسمت بعد به توضیح زیر واحد اول خواهیم پرداخت. ولی در اینجا به این نکته بسته خواهد شد که زیرذ واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه شماره سطر و ستون بالامثلی ماتریس فاصله‌ها را تولید می‌کند. این شماره‌ها از صفر شروع شده و تا آخرین شهر منهای یک ادامه می‌یابد.

حال به توضیح دو نوع طراحی این واحد می‌پردازیم. در ساختار اول یعنی طرح دارای رام بلوکی دو درگاهه (شکل ۶۵-۳)، بعد از اینکه سیگنال تحریک به ورودی NEXT_ADDR داده شد، یک شماره سطر و ستون از ماتریس بالامثلی روی خروجی‌های واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه قرار می‌گیرد. این شماره‌ها آدرس مختصات طولی و عرضی دو شهر بر روی رام است. پس در کلاک پالس بعد، مختصات دو شهر بر روی خروجی‌های رام‌ها قرار می‌گیرد. همچنین دو خروجی

واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه بعد از اینکه با یک جمع شدند به ورودی‌های واحد محاسبه‌گر آدرس رم یا رام داده می‌شوند. دلیل اضافه شدن یک به این دو آدرس آنست که آدرس شهرها روی حافظه از صفر شروع می‌شود، ولی شماره شهرها از یک شروع می‌شود و واحد محاسبه‌گر آدرس رم یا رام به شماره شهرها برای محاسبات خود نیاز دارد. بعد از قرار گرفتن اطلاعات روی ورودی‌های دو واحد محاسبه‌گر فاصله اقلیدسی و محاسبه‌گر آدرس رم یا رام، سیگنال فعال‌سازی به این دو واحد ارسال خواهد شد. در نتیجه، این دو واحد فاصله اقلیدسی دو شهر و آدرس آن بر روی حافظه را روی خروجی‌های خود قرار خواهند داد. همچنین یک کلک پالس بعد، سیگنال اتمام عملیات را روی خروجی DONE خود قرار خواهند داد و این سیگنال را تا زمان بازنشانی سیگنال فعال‌سازی روی خروجی‌های خود نگه خواهند داشت. خروجی واحد از AND خروجی اتمام عملیات این دو واحد حاصل می‌شود. سیگنال WRITE_ENABLE فعال‌سازی این دو واحد هم تلفیقی از خروجی ENABLE_OUT واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه و ورودی NEXT_ADDR است. این سیگنال تضمین می‌کند که این دو واحد بعد از قرار گرفتن آدرس مجاز روی خروجی تولیدکننده آدرس ترتیبی دو شهر برای حافظه و همچنین اطلاعات مورد نیاز روی ورودی‌های آن‌ها فعال شوند.

ساختار دوم یعنی طرح دارای رام بلوکی تک درگاهه (شکل ۳-۶۶)، عملکردی شبیه ساختار اول دارد با چند تفاوت که در ادامه توضیح داده خواهد شد. اولین تفاوت آنست که به دلیل استفاده از رام تک درگاهه، در دو مرحله مختصات مورد نیاز واحد محاسبه‌گر فاصله اقلیدسی مهیا می‌شود. اولین بار مختصات طولی و عرضی یک شهر و دومین بار مختصات طولی و عرضی شهر دوم به رام درخواست داده می‌شود. این چهار مختصات روی فلیپ‌فلاب‌های D متصل به ورودی‌های واحد محاسبه‌گر فاصله اقلیدسی ذخیره می‌شوند. قسمت کنترل‌کننده این طرح نیاز به توضیح بیشتر داشته که در ادامه نحوه عملکرد آن شرح داده شده است. شکل ۳-۶۷ قسمت کنترل‌کننده این طرح را نمایش می‌دهد.



شکل ۳-۶۷ قسمت کنترل ساختار تولیدکننده آدرس و محتوای ماتریس فاصله‌ها با رام تک درگاهه

همانند قبل، این قسمت هم تلفیقی از خروجی ENABLE_OUT واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه و ورودی NEXT_ADDR است. با این تفاوت که نه تنها مسئول تولید سیگنال فعل سازی دو واحد محاسبه‌گر فاصله اقلیدسی و محاسبه‌گر آدرس رم یا رام بوده؛ بلکه عهده‌دار ارسال سیگنال‌های کنترلی به مالتی‌پلکسرا، دی‌مالتی‌پلکسرا و فلیپ‌فلاب‌های D متصل به ورودی‌های واحد محاسبه‌گر فاصله اقلیدسی نیز هست. در سمت چپ شکل ۳-۶۷، از تلفیق خروجی واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه و ورودی NEXT_ADDR و ENABLE_OUT همچنین با بهره گرفتن از دو فلیپ‌فلاب D و دو گیت منطقی، سیگنالی را خواهیم داشت که در شروع هر جفت آدرس مجاز یک پالس ایجاد می‌کند. به منظور ایجاد تاخیر، این پالس به یک شیفت رجیستر داده می‌شود. از خروجی این شیفت رجیستر بازخورده به ورودی خود آن داده شده است که دلیل آن نیاز به دو پالس بعد از هر جفت آدرس مجاز است. پس از آنکه یک جفت آدرس مجاز روی خروجی‌های واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه نشست و قبل از اولین پالس کنترلی، ورودی‌های رام‌ها از طریق مالتی‌پلکسر به خروجی CURRENT_CITY_ADDRESS واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه متصل است. پس مختصات طولی و عرضی شهر اول در زمان دریافت این پالس بر روی خروجی‌های رام و ورودی‌های داده فلیپ‌فلاب‌های D متناظر آن‌ها مهیا شده است. در نتیجه با رسیدن این پالس به این فلیپ‌فلاب‌ها، این مختصات روی آن‌ها ذخیره می‌شود. همچنین انشعابی از این پالس در قسمت کنترل کننده پس از گذر از یک فلیپ‌فلاب D به یک فلیپ‌فلاب T می‌رسد و موجب می‌شود که ورودی دی‌مالتی‌پلکسر متصل به فلیپ‌فلاب T به خروجی شماره یک آن متصل شود. بعلاوه با یک شدن خروجی فلیپ‌فلاب T، اتصالات مالتی‌پلکسر

و دو دی مالتی پلکسر متصل به رامها هم تغییر می کند. این تغییر باعث می شود که خروجی NEXT_CITY_ADDRESS واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه به ورودی رامها وصل شود. پس در کلاک پالس بعدی، محتوای جدید رام روی خروجی آنها آماده خواهد بود و از طریق دی مالتی پلکسراها روی ورودی داده دو فلیپفلاب D متناظر آنها قرار خواهد گرفت. پس از چند کلاک پالس، پالس دوم (پالس بازخوردنی به ورودی شیفت رجیستر) به خروجی آن می رسد. این پالس از طریق دی مالتی پلکسر قسمت کنترل کننده به ورودی کلاک پالس این دو فلیپفلاب D متصل شده و باعث می شود که مختصات طولی و عرضی شهر دوم در این دو فلیپفلاب ذخیره شود. مضافاً با یک کلاک پالس تاخیر و از طریق دو فلیپفلاب D در قسمت راست بخش کنترل-کننده، انشعابی از پالس دوم سیگنال فعال سازی دو واحد محاسبه گر فاصله اقلیدسی و محاسبه گر آدرس رم یا رام و همچنین سیگنال بازنشانی شیفت رجیستر را مهیا می کند. انشعابی دیگر از همین پالس دوم پس از گذر از یک فلیپفلاب D در قسمت کنترل کننده، ورودی کلاک پالس فلیپفلاب T را تحریک کرده و باعث می شود که تمام تنظیمات به شروع این روند باز گردد. یعنی جایی که ورودی رامها به خروجی CURRENT_CITY_ADDRESS واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه وصل بود و خروجی رامها هم به فلیپفلاب های D متناظر با ذخیره مختصات شهر اول و متصل به واحد محاسبه گر فاصله اقلیدسی متصل بود. چون سیگنال فعال سازی دو واحد محاسبه گر فاصله اقلیدسی و آدرس رام باید تا درخواست جدید محاسبات فعال بماند، ورودی بازنشانی فلیپفلاب D متصل به این سیگنال با استفاده از سیگنال NEXT_ADDR بازنشانی می شود.

۱-۳-۷-۳ واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه

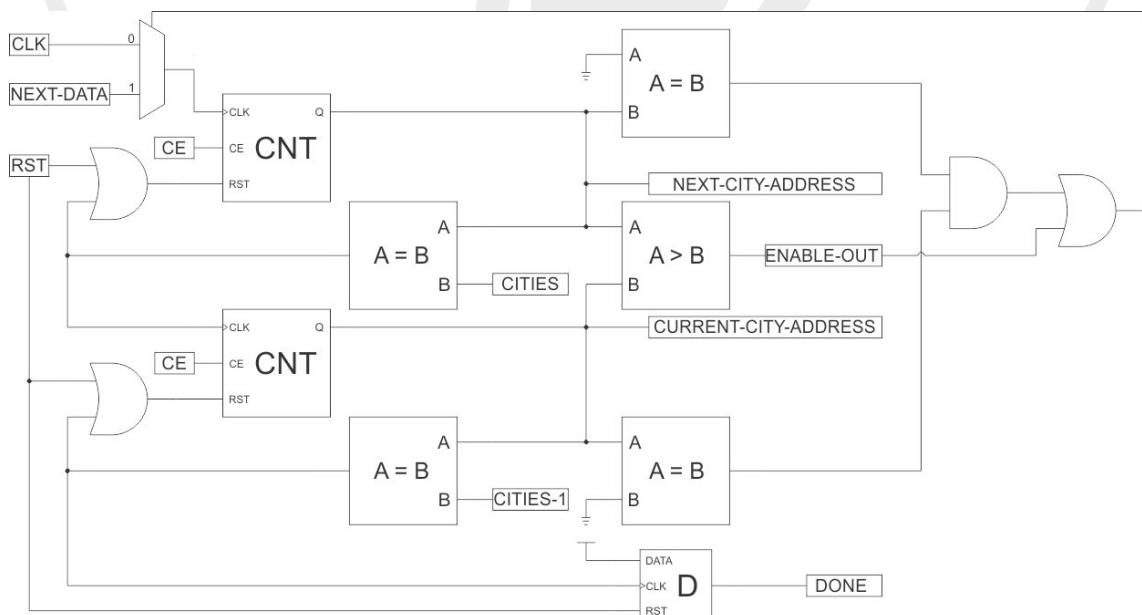
نحوه عملکرد این واحد همانند واحد تولیدکننده آدرس ترتیبی دو شهر بدون برگشت است. این واحد تولید شماره سطر و ستون بالامثلی ماتریس فاصله ها را به عهده دارد. این شمارش از شهر ۱ و ۲ در آدرس ۰ و ۱ شروع شده و تا شهر یکی مانده به آخر و شهر آخر ادامه می یابد. این واحد همانند هسته مشابه اگر فعال باشد، با دریافت سیگنال روی ورودی NEXT_DATA شماره سطر و ستون بالامثلی ماتریس فاصله ها را به ترتیب با شروع از صفر و یک روی خروجی های خود یعنی

آدرس‌های تولید شده در محدوده آدرس‌های مجاز ماتریس بالامثلی باشد، این واحد خروجی NEXT_CITY_ADDRESS و CURRENT_CITY_ADDRESS قرار می‌دهد. تا زمانی که پیاده‌سازی شده است. خروجی DONE برای اعلام اتمام شمارش سطر و ستون ماتریس بالامثلی در نظر گرفته شده است که معادل آدرس فاصله شهر یکی مانده به آخر و شهر آخر است. نمای ورودی و خروجی‌های این واحد در شکل ۶۸-۳ ترسیم شده است.



شکل ۶۸-۳ واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه

همانند واحد مشابه، این واحد هم در سطح ساختار پیاده شده است و نحوه پیاده‌سازی آن در شکل ۶۹-۳ مشاهده می‌شود.



شکل ۶۹-۳ ساختار واحد تولیدکننده آدرس ترتیبی دو شهر برای حافظه

نحوه عملکرد این واحد بدین گونه است که با فرض دریافت سیگنال فعالسازی، در ابتدا که مقادیر آدرس سطر و ستون صفر است، واحد متظر دریافت سیگنال از ورودی NEXT_DATA می‌ماند. به محض دریافت این سیگنال، مقدار ورودی کلک پالس شمارنده بالایی به کلک پالس اصلی سیستم وصل شده و شمارش شروع شده تا که به اولین مقدار مجاز ستون در ماتریس بالامثلی برسد یعنی یک. در این حالت ورودی کلک این شمارنده دوباره به ورودی NEXT_DATA متصل شده و برای تولید جفت آدرس بعدی متظر سیگنال از این ورودی می‌ماند. با هر بار رسیدن شمارنده بالایی به تعداد شهرها این شمارنده بازنشانی شده و شمارنده پایینی یک واحد افزایش می‌یابد. پس از بازنشانی شمارنده بالایی، کلک پالس آن مجدداً به کلک پالس اصلی متصل شده تا که در هر کلک پالس و به سرعت تا مقادیر مجاز ستون ماتریس بالامثلی شمارش شود. مقادیر مجاز با شرط بزرگتر بودن از شماره سطر مشخص می‌شوند. این روند تا رسیدن به آخرین جفت یعنی سطر یکی مانده به آخر ماتریس بالامثلی که معادل شهر یکی مانده به آخر و ستون آخر ماتریس بالامثلی که معادل شهر آخر است، ادامه می‌یابد. البته یک واحد از شماره سطر و ستون کم می‌شود؛ چراکه این شماره‌ها باید معادل آدرس مختصات شهر مورد نظر روی رام باشد؛ یا به عبارتی شماره سطر و ستون ماتریس باید از صفر شروع شوند. همچنین یک کلک پالس پس از آخرین جفت آدرس، سیگنال اتمام عملیات به خروجی ارسال می‌شود.

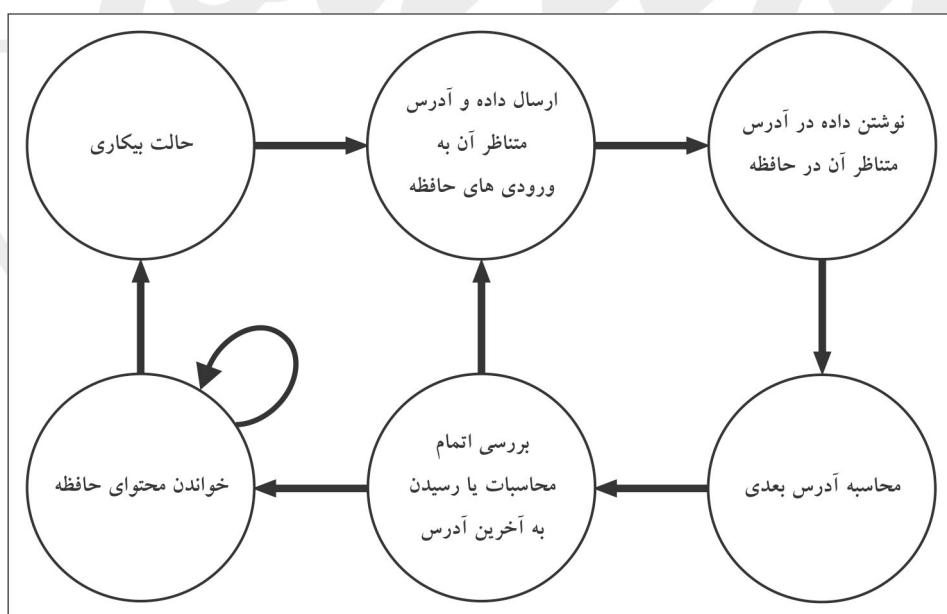
۴-۷-۳ واحد تولیدکننده ماتریس فاصله‌ها روی حافظه

پس از پیاده‌سازی تولیدکننده آدرس و محتوای ماتریس فاصله‌ها، برای ثبت مقادیر ماتریس فاصله‌ها روی حافظه باید واحد دیگری وجود داشته باشد. به منظور ایجاد چندین ماتریس فاصله‌ها، این واحد می‌تواند این مقادیر را یکبار محاسبه کرده و در مکان‌های مختلف حافظه ذخیره کند. ولیکن در اینجا فقط واحدی بررسی می‌شود که در یک محدوده آدرس از یک حافظه این مقادیر را ذخیره می‌کند. در صورتی که نیاز به ماتریس‌های بیشتری باشد، با تغییرات جزئی روی واحد ارائه شده این امکان فراهم خواهد شد. در شکل ۳-۷۰ ورودی و خروجی‌های این واحد مشاهده می‌شود.



شکل ۳-۷۰ واحد تولیدکننده ماتریس فاصله‌ها روی حافظه

با دریافت سیگنال فعال‌سازی، این واحد شروع به نوشتن عناصر ماتریس فاصله‌ها بر روی حافظه می‌کند و در انتهای کار خروجی **DONE** را یک می‌کند. پس از اتمام نوشتن مقادیر روی حافظه، برای بررسی صحت مقادیر نوشته شده می‌توان با استفاده از ورودی **RAM_ADDRESS**، مقادیر نوشته شده را بر روی خروجی **RAM_DATA** خواند. در شکل ۳-۷۱ نمودار ماشین حالت این واحد آورده شده است.



شکل ۳-۷۱ نمودار ماشین حالت واحد تولیدکننده ماتریس فاصله‌ها روی حافظه

این واحد شامل حالت‌های زیر است:

- حالت بیکاری: در این حالت سیگنال فعال‌سازی به منظور شروع عملیات ذخیره بررسی می‌شود. در صورت دریافت این سیگنال برای شروع روند به حالت بعد منتقل می‌شود.
- ارسال داده و آدرس متناظر آن به ورودی‌های حافظه: در این مرحله فاصله و آدرس متناظر محاسبه شده را روی ورودی‌های رم قرار می‌دهد. شروع این روند همزمان با حالت بیکاری تولیدکننده آدرس و محتوای ماتریس فاصله‌ها نیز هست؛ در نتیجه حداکثر مقدار فاصله و آدرس صفرم بر روی خروجی‌های این واحد قرار دارند. همچنین از قبل به یاد داریم که در اولین سلول حافظه یعنی آدرس صفرم حافظه باید حداکثر مقدار فاصله ذخیره شود. پس با شروع از این مرحله در اولین سلول مقدار حداکثر فاصله ذخیره می‌شود.
- نوشتن داده در آدرس متناظر آن در حافظه: در این حالت سیگنالی برای ورودی اجازه نوشتن¹ رم ارسال می‌شود که نتیجتاً رم در لبے بالاروندۀ کلک پالس پالس بعدی فاصله فرستاده شده به ورودی متناظر را روی خود ذخیره می‌کند.
- محاسبۀ آدرس بعدی: در این مرحله درخواست محاسبۀ آدرس و محتوای دیگری از ماتریس فاصله‌ها به واحد تولیدکننده آدرس و محتوای ماتریس فاصله‌ها ارسال می‌شود.
- بررسی اتمام محاسبات یا رسیدن به آخرین آدرس: در این حالت خروجی‌های WE و DONE واحد تولیدکننده آدرس و محتوای ماتریس فاصله‌ها بررسی می‌شود. اگر از خروجی WE سیگنالی دریافت شود؛ یعنی محاسبات مربوط به فاصله و آدرس به اتمام رسیده است؛ در این شرایط به مرحله ارسال داده و آدرس متناظر آن به ورودی‌های حافظه باز خواهد گشت. اگر از خروجی DONE سیگنالی دریافت کند؛ یعنی تمام مقادیر و آدرس‌های متناظر آن‌ها را محاسبه کرده است و کار ذخیره مقادیر روی حافظه هم به اتمام رسیده است؛ در این صورت به حالت بعد رفته تا بتوان مقادیر نوشته شده بر روی حافظه را خواند.

¹ Write Enable (WE)

- خواندن محتوای حافظه: در این حالت سیگنال اتمام عملیات را بر روی خروجی DONE واحد قرار خواهد داد. مقدار روی ورودی RAM_ADDRESS را بر روی ورودی رم قرار می‌دهد. همچنین سیگنال فعال‌سازی واحد را بررسی می‌کند؛ اگر سیگنال فعال‌سازی بازنمانی شود به حالت بیکاری باز می‌گردد. اگر هم که این سیگنال برقرار باشد، خروجی اتمام عملیات را یک نگاه داشته و امکان خواندن مقادیر نوشته شده رم را مهیا می‌سازد.



۴. نتایج

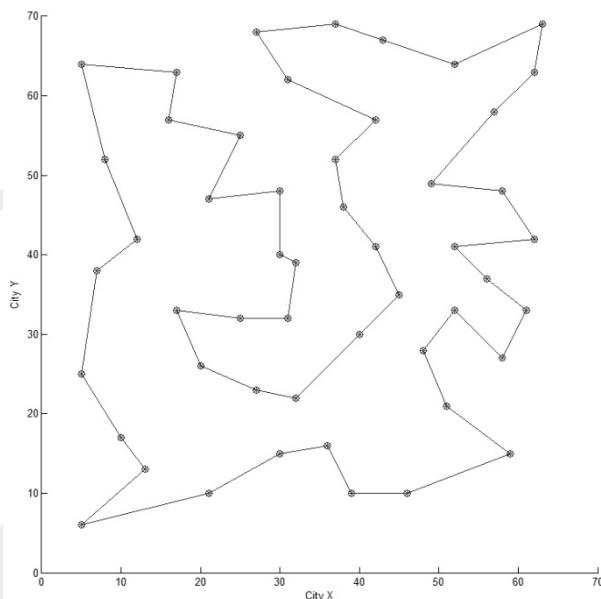
در این فصل ابتدا به توضیح پایگاه داده‌ای که از اطلاعات آن برای امتحان الگوریتم‌ها استفاده شده است می‌پردازیم و همچنین نرم‌افزارهایی که الگوریتم‌ها در آن‌ها پیاده و شبیه‌سازی شده‌اند. سپس نتایج حاصل از پیاده‌سازی‌های نرم‌افزاری و سخت‌افزاری ارائه می‌شود.

۱-۴ پایگاه داده‌ها

باقotope به بزرگ بودن فضای جستجو در اکثر مسائل TSP و در نتیجه دشوار بودن حل آن‌ها، از این گونه مسائل معمولاً برای آزمودن میزان کارایی الگوریتم‌های مختلف بهینه‌سازی ترکیبی و نیز مقایسه عملکرد آن‌ها استفاده می‌شود. مثلاً اگر فردی الگوریتم بهینه‌سازی تازه‌ای را کشف کند و مدعی کارایی بالای آن باشد، یک راه برای اثبات این ادعا اعمال آن به مسائل گوناگون TSP و مقایسه نتایج (از جنبه‌های مختلف) با سایر الگوریتم‌های جاافتاده و مقبول است.

ممکن است در نگاه اول چنین به نظر برسد که برای تولید یک مسئله TSP می‌توان تعداد دلخواهی نقطه (یا همان شهر) را به طور تصادفی بر روی صفحه قرار داد. اشکال این روش تولید در مسئله در این است که در حالت کلی راهی برای آگاهی از پاسخ بهینه سراسری مسئله تولید شده وجود ندارد و در نتیجه عملاً نمی‌توان در مورد کارایی الگوریتمی که جوابی برای چنین مسئله‌ای به دست می‌آورد، نیز قضاوت کرد. به همین دلیل در عمل از مسائل خاص و استانداردی که پاسخ بهینه سراسری‌شان از قبل معلوم است، استفاده می‌شود که در این گونه مسائل که با اسمی استانداردی نظیر eil76، eil51، kroA100 و مشخص می‌شوند، تعداد معینی شهر (با الهام‌گیری از پدیده‌های طبیعی و فیزیکی) در نقاط معینی از صفحه قرار داده شده‌اند. شکل‌های ۱-۴ تا ۳-۴ به ترتیب مسائل

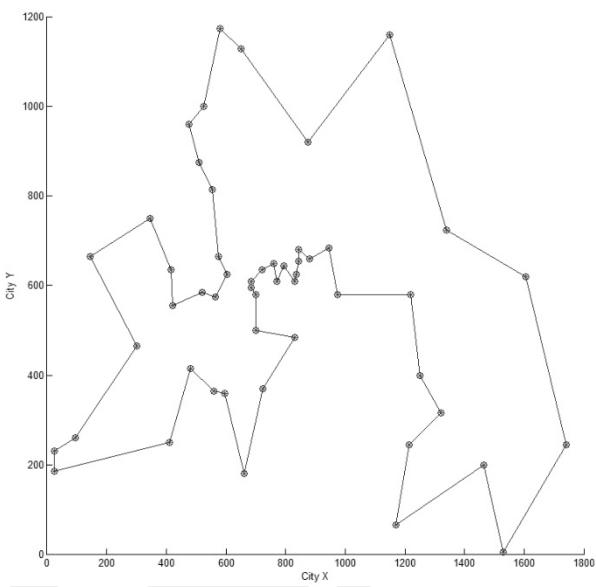
tsp225, berlin52 را به همراه جواب بهینه سراسری شان نشان می‌دهند. همانطور که پیشتر نیز اشاره کردیم این گونه مسایل معمولاً براساس موقعیت پدیده‌های طبیعی طراحی شده‌اند.



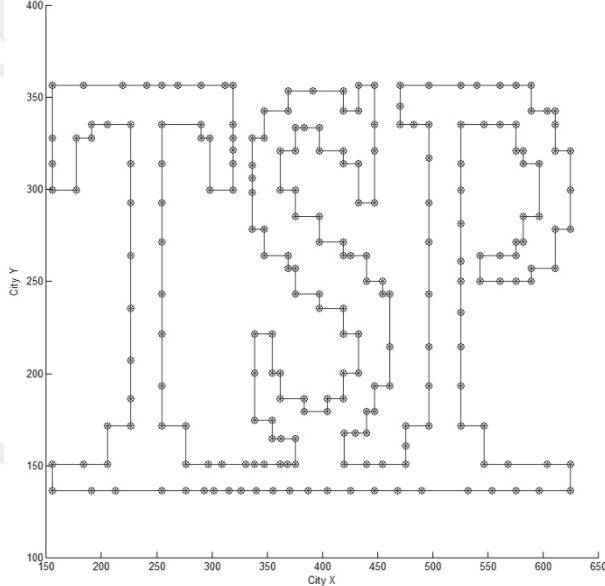
شکل ۱-۴ مسئله eil51 به همراه پاسخ بهینه سراسری آن

به عنوان مثال مسئله berlin52 موقعیت ۵۲ نقطه را در شهر برلین و مسئله pcb442 موقعیت ۴۴۲ نقطه را بر روی یک مدار چاپی جهت انجام عملیات سوراخکاری نشان می‌دهد (عدد موجود در انتهای اسم هر مسئله بیانگر تعداد نقاط موجود در آن مسئله است) (بیات، ۱۳۹۳).

تمامی مسائل استفاده شده از [۵۲] دریافت شده‌اند. این سایت موثق‌ترین منبعی است که انواع داده‌های مربوط به فروشنده دوره‌گرد را به صورت رایگان در اختیار محققین قرار داده است. در منبع مذکور مسائل متفاوت مسئله فروشنده دوره‌گرد شامل متقارن یا نامتقارن موجود است که در این تحقیق از نوع متقارن که در آن فاصله رفت و برگشت بین دو شهر برابر است، استفاده شده است.



شکل ۲-۴ مسئله berlin52 به همراه همراه پاسخ بهینه سراسری آن



شکل ۳-۴ مسئله tsp225 به همراه پاسخ بهینه سراسری آن

۲-۴ نحوه اعتبارسنجی

- ارزیابی نرمافزاری با بهره‌گیری از نرمافزار MATLAB انجام شده است. به عنوان یک زبان برنامه‌نویسی و ابزاری برای نمایش داده‌ها، MATLAB مجموعه‌ای از قابلیت‌های مفیدی است که از آن‌ها برای رشته‌های مختلف فنی و مهندسی، علوم، حسابداری و ریاضی استفاده می‌شود.
- به منظور پیاده‌سازی سخت‌افزاری، FPGA‌های ساخت شرکت Xilinx در نظر گرفته شده است. همچنین در اکثر مقالات پیاده‌سازی الگوریتم‌های فراتکاری، بیشتر از محصولات این شرکت و خانواده‌های Spartan [۱۹] و Virtex [۴۲]، [۴۶]، [۵۱] استفاده شده است. در این بین بیشترین زیرمجموعه‌های استفاده شده از این دو خانواده Spartan-3 [۲۱]، [۴۷] و Virtex-5 [۴۱]، [۴۳]، [۴۴]، [۴۵]، [۴۸]، [۴۹] بوده است. به همین دلیل در این پیاده‌سازی از این خانواده و تراشه XC5VLX330 بهره گرفته شده است. مشخصات این تراشه در جدول ۱-۴ آمده است.

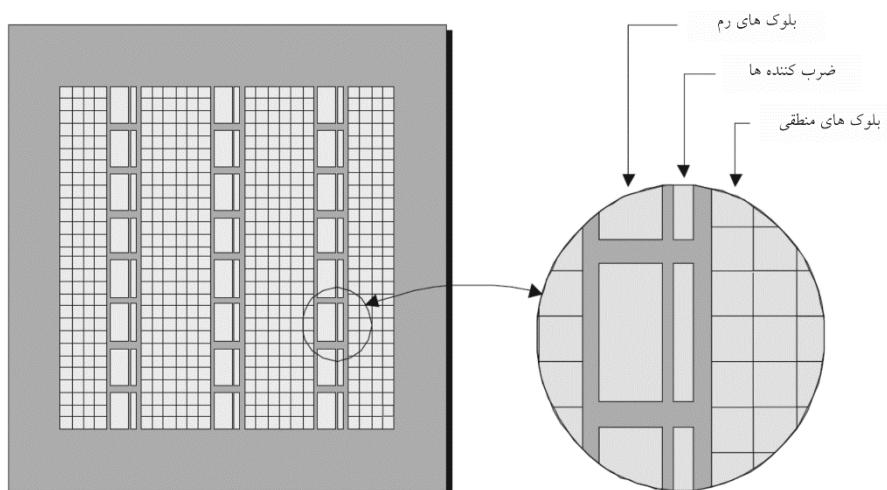
جدول ۱-۴ مشخصات تراشه مورد استفاده در پیاده‌سازی [۵۳]

بلوک‌های رم (کیلو بیت)	پردازنده سیگنال دیجیتالی (DSP48E)	رم‌های توزیع شده (کیلو بیت)	برش‌ها (Slices)	بلوک‌های منطقی قابل پیکربندی (CLBs)	تراشه
۱۰۳۶۸	۱۹۲	۳۴۲۰	۵۱۸۴۰	۲۵۹۲۰	XC5VLX330

در خانواده Virtex-5، هر برش شامل چهار جدول جستجو و چهار فلیپ‌فلاب می‌شود. همچنین آنطور که از جدول ۱-۴ مشخص است، هر CLB در این تراشه از دو برش تشکیل شده است.

در قبل بیان شد که بلوک‌های رم در واحدهای ۱۸ کیلوبیتی موجود هستند که در این تراشه این بلوک‌های رم در واحدهای ۳۶ کیلوبیتی در دسترس هستند، ولیکن می‌توان هر بلوک را به صورت دو واحد ۱۸ کیلوبیتی نیز استفاده کرد.

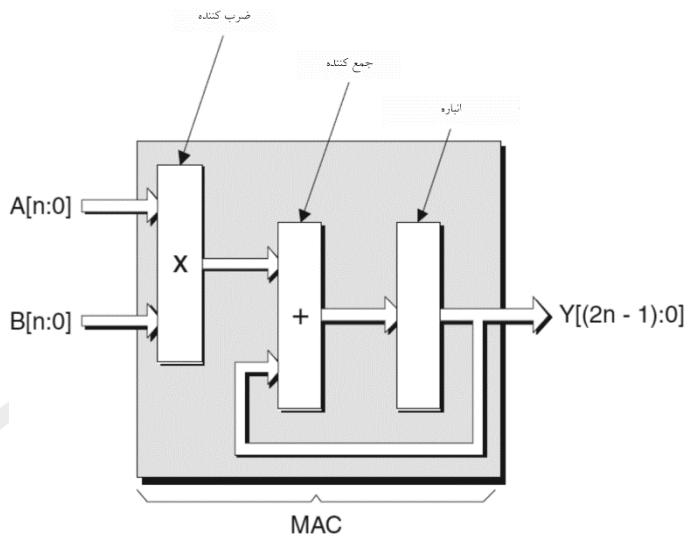
برخی توابع، از قبیل ضرب‌کننده‌ها، اگر با اتصال چندین بلوک منطقی برنامه‌پذیر به هم پیاده‌سازی شود، ذاتاً سرعت پایینی دارند. چون این توابع در کاربردهای زیادی استفاده می‌شوند، بسیاری از FPGA‌ها بلوک‌های ضرب‌کننده سخت‌افزاری خاصی دارند. معمولاً این بلوک‌ها در مجاورت بلوک‌های RAM تعییه شده قرار دارند؛ زیرا غالباً این توابع همراه با یکدیگر استفاده می‌شوند (شکل ۴-۴) (ماکسفیلد، ۱۳۸۹).



شکل ۴-۴ نمای کلی تراشه با ستون‌های ضرب‌کننده تعییه شده و بلوک‌های RAM همچنین برخی از FPGA‌ها بلوک‌های جمع‌کننده اختصاصی دارند. یکی از عملیات‌هایی که در کاربردهایی از نوع DSP متداول است ضرب و انباشتن^۱ نامیده می‌شود (شکل ۵-۴). همانطور که از نام آن برمی‌آید، این تابع دو عدد را در هم ضرب کرده و نتیجه را به جمع کل در یک انباره^۲ اضافه می‌کند (ماکسفیلد، ۱۳۸۹).

¹MAC

²Accumulator



شکل ۵-۴ توابع تشکیل دهنده یک MAC

اگر FPGA مورد استفاده فقط ضرب کننده های تعییه شده دارد مجبوری دارد این تابع را با ترکیب ضرب کننده و یک جمع کننده مشکل از چند بلوک منطقی برنامه پذیر پیاده کنید در حالی که در چند فلیپ فلاب مربوطه، RAM های توزیع شده یا در یک RAM بلوکی ذخیره می شود. اگر جمع کننده های تعییه شده نیز داشته باشد کارها ساده تر می شود. برخی از FPGA های کاملی را به صورت توابع تعییه شده دارند (ماکسفلد، ۱۳۸۹).

این تراشه هم همانند بیشتر تراشه های دیگر شامل ابزاره، ضرب کننده و جمع کننده می شود که در قالب پردازنده سیگنال دیجیتالی در جدول ۱-۴ آورده شده است. این DSP ها هر کدام دارای یک ابزاره، یک ضرب کننده و یک جمع کننده هستند.

- نرم افزار مخصوص FPGA ها و CPLD های شرکت Xilinx ISE نامیده می شود که با هر دو زبان VHDL و Verilog می تواند کدهای سخت افزاری را نوشته، غلطگیری نوشتاری، سرززه، شبیه سازی و تراشه را برنامه ریزی کند. همچنین زبان سخت افزاری VHDL نیز در نظر گرفته شده است.

- تست نتایج در نرم افزار شبیه سازی ISIM انجام شده است. نرم افزار شبیه ساز در حین شبیه سازی، عملکرد و زمان بندی طرح یا قسمت هایی از آن را بازبینی می کند. شبیه ساز، کد VHDL را به عملکردهای مداری تفسیر کرده و برای تعیین درستی کار کرد مدار، نتایج

منطقی HDL توصیف شده را نمایش می‌دهد. شبیه‌سازی در چند مرحله انجام می‌شود. در نخستین مرحله پس از وارد کردن طرح و آخرين مرحله بعد از پیاده‌سازی، جهت بازبینی عملکرد نهایی و کارآیی طرح، شبیه‌سازی انجام می‌شود. شبیه‌سازی یک فرآیند تکراریست که ممکن است برای رسیدن به عملکرد و زمان‌بندی مطلوب طرح، به تکرار نیاز داشته باشد.

شبیه‌سازی از مراحل کلی زیر تشکیل می‌شود:

کامپایل کردن کتابخانه‌های شبیه‌سازی

ایجاد طرح و آزمونه

شبیه‌سازی عملکردی

پیاده‌سازی طرح و ایجاد نتیجه‌سازی زمان‌بندی

شبیه‌سازی زمان‌بندی

نرم‌افزار ISE به گونه‌ای طراحی شده است که با چند ابزار سنتز و شبیه‌سازی قابل استفاده باشد و به این علت برای پیاده‌سازی طرح‌های منطقی برنامه‌پذیر از آغاز تا پایان مناسب است. این نرم‌افزار چنان با شبیه‌ساز (ISE ISIM) یکپارچه است که از واسط کاربر گرافیکی هدایت‌گر پروژه آن می‌توان شبیه‌سازی را انجام داد (ماکسفیلد، ۱۳۸۹).

۳-۴ نتایج پیاده‌سازی نرم‌افزاری

در این زیرقسمت نتایج حاصل از پیاده‌سازی نرم‌افزاری سه الگوریتم معروف غذاخوابی زنبور عسل یعنی BA، CABC یا همان ABC ترکیبی و BCO به همراه روش ارائه شده که هدف از آن پیاده‌سازی سخت‌افزاری است، به ترتیب آورده شده است.

به منظور مقایسه صحیح و منصفانه بین این الگوریتم‌ها، فاکتور اصلی تعداد بهینه‌سازی‌های محلی ملاک قرار گرفته است. بدین معنی که جستجوهای محلی در هر تکرار و در نتیجه کل تکرارها یکسان باشد. بدین منظور در هر تکرار هر یک از الگوریتم‌ها تعداد بیست جستجوی محلی را انجام می‌دهند. حال که تعداد جستجوها مشخص شد باید تعداد تکرارهای کل به صورتی در نظر گرفته شود که نتایج حاصل در اجراهای متفاوت از یک الگوریتم تفاوت ناچیزی داشته باشند. بدین هدف

برای مسائلی که کمتر از ۷۵ شهر دارند، تعداد ۳۰۰۰ تکرار و برای مسائلی با ۷۵ شهر یا بیشتر ۶۰۰۰ تکرار برای هر بار اجرا مدنظر قرار گرفته است.

در سه الگوریتم از چهار الگوریتم از پارامتر $n!$ استفاده شده است که همان تعداد تکرارهایی است که اگر جواب یک زنبور هیچگونه بهبودی نیابد، این زنبور با جواب‌های تصادفی جدید جایگزین شده و جستجو را حول جواب جدید شروع خواهد کرد. این پارامتر بر اساس [۳۳] انتخاب شده است که همانطور که در فصل سوم بیان شد، در این مرجع الگوریتم CABC برای مسئله فروشنده دوره‌گرد ارائه شده است.

در بیشتر روش‌های ارائه شده برای حل مسئله فروشنده دوره‌گرد جواب‌های اولیه به صورت تصادفی در نظر گرفته نمی‌شوند، چراکه به نتایج مناسبی ختم نخواهند شد. این جواب‌ها حاصل مسیر نزدیکترین همسایگی هستند، به عنوان نمونه [۳۳]. بدین دلیل در تمام پیاده‌سازی‌های انجام شده نیز از مسیر نزدیکترین همسایگی برای این مهم بهره گرفته شده است.

همچنین برای اینکه مقایسه خوبی در مورد روش انتخابی برای پیاده‌سازی سخت‌افزاری داشته باشیم، نتایج در حوزه نرم‌افزار با هر دو روش جستجوی محلی که در فصل قبل بیان شد یعنی 2-opt و GSTM ارائه شده است. GSTM یک روش پیچیده و دارای چهار روش متفاوت جستجوست که این کاوش‌ها شامل روش‌های حریص و سنتی هستند. در حالیکه روش 2-opt تنها با عملگر دوران جستجوی محلی را انجام داده که یک عملگر سنتی به حساب می‌آید.

در این قسمت نتایج در قالب شکل و نمودار برای سه مسئله معروف آورده شده است. شکل‌ها و نمودارها ذهنیت مناسبی در نحوه پیشرفت الگوریتم و جواب نهایی ایجاد می‌کنند. شکل سمت چپ در هر قسمت نمایانگر مسیر نهایی بدست آمده است. در این شکل هر ستاره که درون یک دایره قرار گرفته نمایانگر یک شهر بوده و دو محور افقی و عمودی مختصات شهرها را در صفحه فرضی نمایش می‌دهند. همچنین خطوط مشخص کننده ترتیب شهرها در مسیر حرکت فروشنده هستند. نمودار سمت راست بهترین پاسخ پیدا شده توسط الگوریتم تا تکرار متناظر را نمایش می‌دهد.

۱-۳-۴ الگوریتم BA

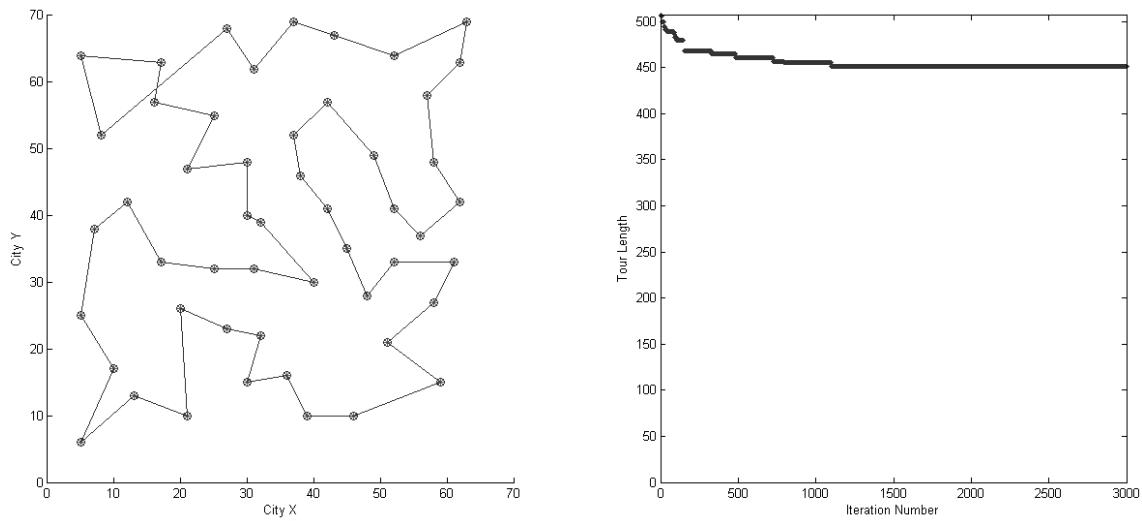
پارمترها در این الگوریتم بدین صورت است:

$$nc = 16, ns = 8, ne = 2, nrs = 2, nre = 4, nl = \frac{nc \times n}{3}$$

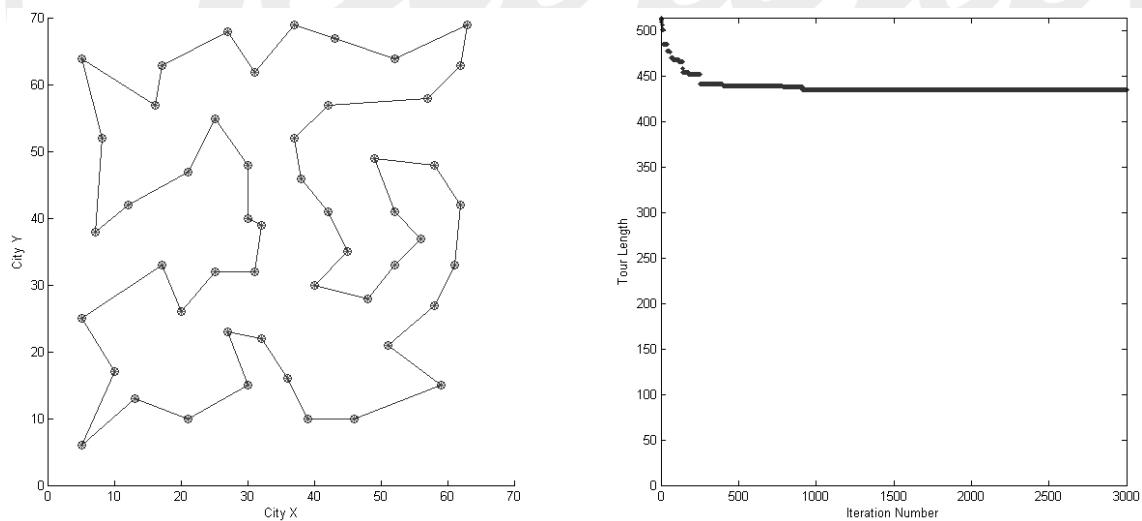
تعداد هشت زنبور منتخب که دو تای آن‌ها نخبه هستند برای این الگوریتم انتخاب شده است که هر کدام از این دو زنبور چهار جستجوی محلی انجام می‌دهند. زنborهای منتخب نیز هر کدام دو جستجو انجام داده که در هر تکرار تعداد جستجوها به بیست بار می‌رسد. به دلیل اینکه نحوه جستجوی جهانی یا به عبارتی تصادفی در این الگوریتم یکی از کلیدی‌ترین ویژگی‌های آن است، زنborهای کلونی دو برابر زنborهای منتخب در نظر گرفته شده‌اند. همچنین برای پرهیز از تکرار بی‌حاصل و کاهش سرعت اجرا پس از آنکه تمامی مسیرهای نزدیکترین همسایگی در روند الگوریتم قرار گرفت و مسیر جدیدی باقی نماند، هشت زنborی که وظیفه جستجوی تصادفی را به عهده داشتند کnar گذاشته شده و جستجوهای محلی روی هشت جواب برتر انجام خواهد شد. در جدول ۲-۴ مسافت هر یک از جواب‌های حاصل با دقت اعشاری آورده شده است. سپس این نتایج به صورت شهودی در شکل‌های ۶-۴ تا ۱۱-۴ نمایش داده شده است.

جدول ۲-۴ مسافت مسیر نهایی الگوریتم BA

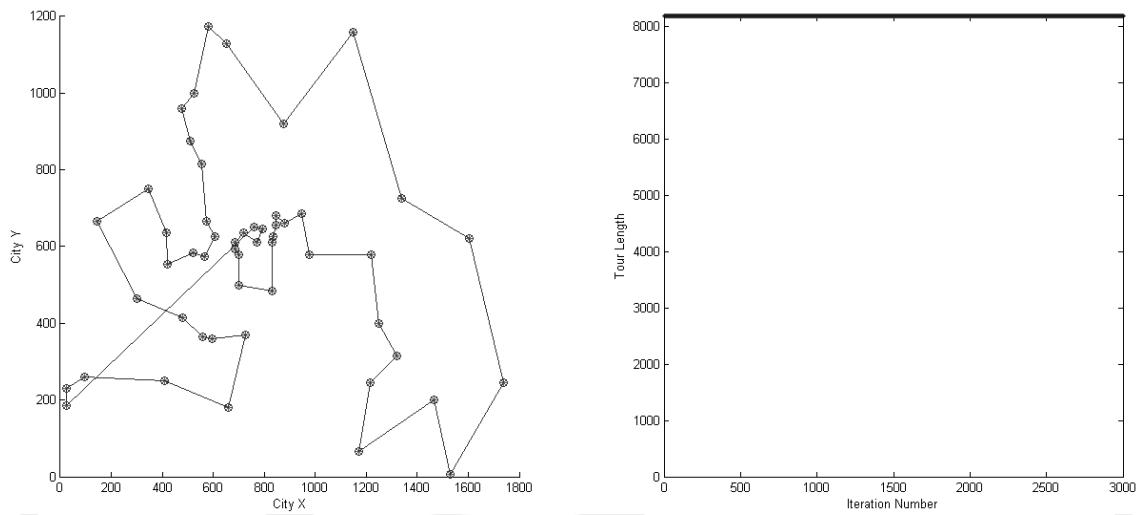
نام مسئله	BA	
	2-opt	GSTM
eil51	۴۵۰/۹۹۳	۴۳۵/۱۸۹۸
berlin52	۸۱۸۲/۱۹	۷۶۰۶/۹۵
kroA100	۲۲۲۴۲/۹	۲۱۶۱۴/۵



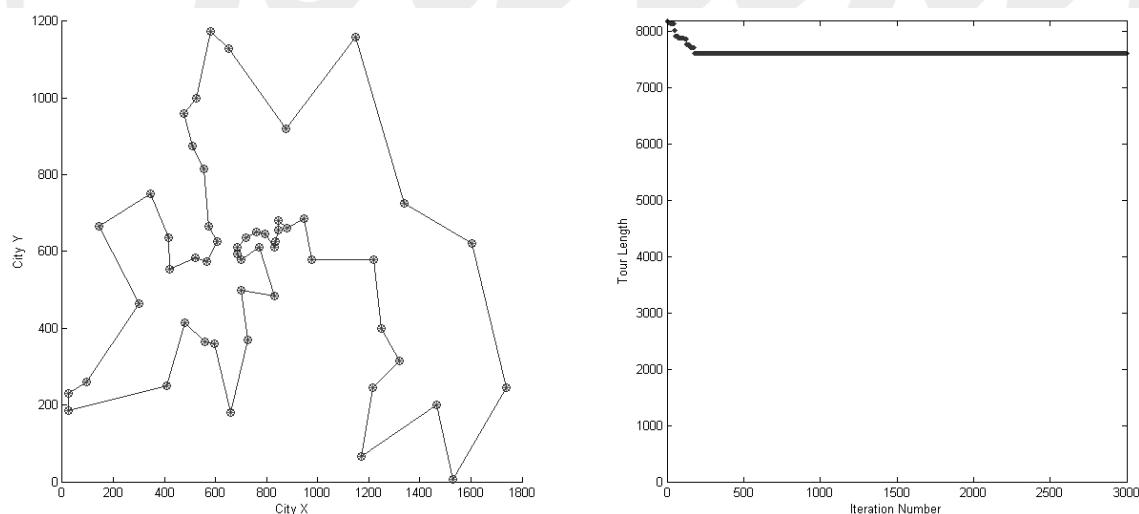
شکل ۴-۶ پاسخ الگوریتم BA در ۳۰۰۰ تکرار با استفاده از 2-opt برای مسئله eil51



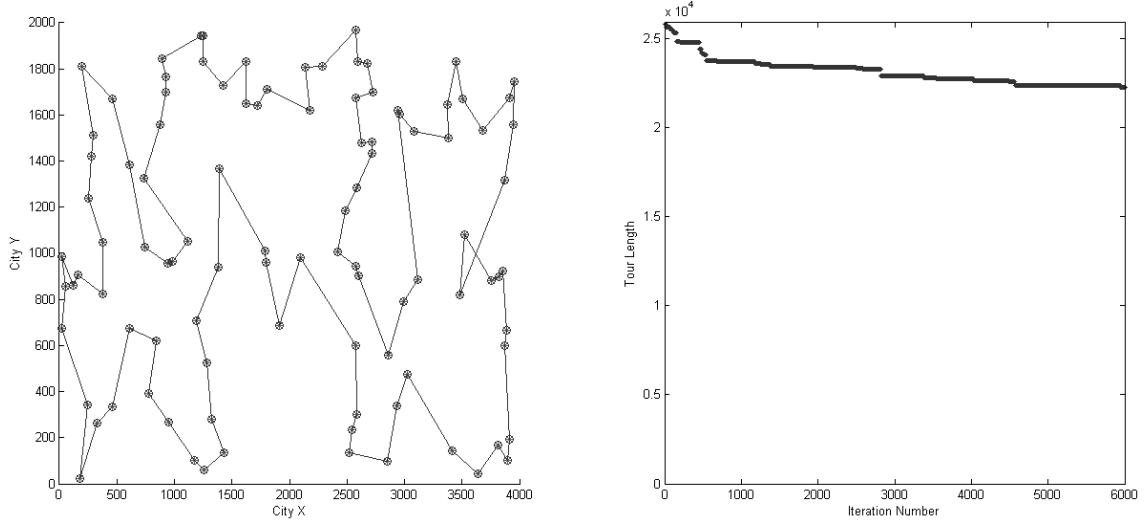
شکل ۴-۷ پاسخ الگوریتم BA در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله eil51



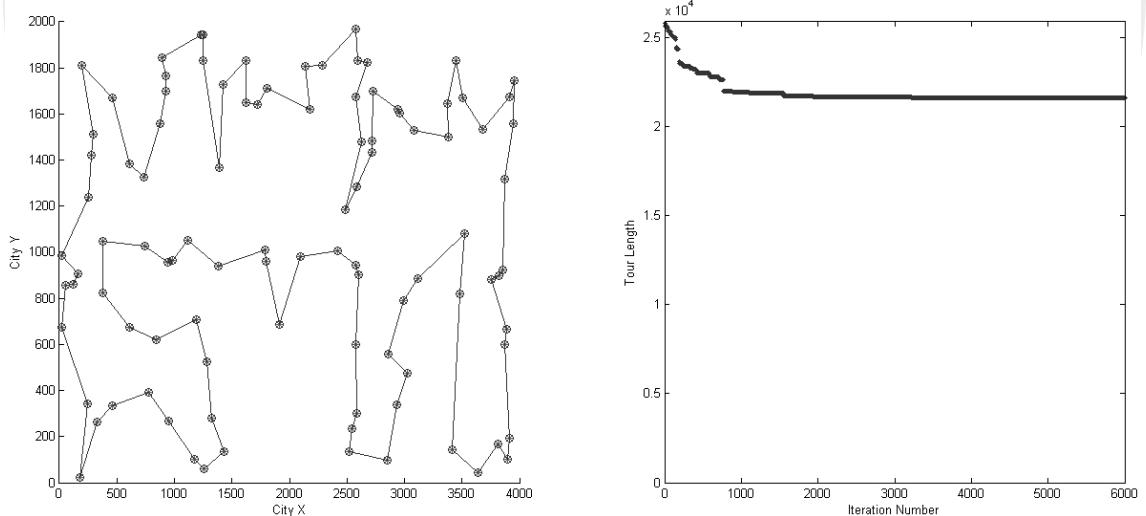
شکل ۸-۴ پاسخ الگوریتم BA در ۳۰۰۰ تکرار با استفاده از 2-opt برای مسئله berlinc52



شکل ۹-۴ پاسخ الگوریتم BA در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله berlinc52



شکل ۱۰-۴ پاسخ الگوریتم BA در ۶۰۰۰ تکرار با استفاده از ۲-opt برای مسئله kroA100



شکل ۱۱-۴ پاسخ الگوریتم BA در ۶۰۰۰ تکرار با استفاده از GSTM برای مسئله kroA100

۲-۳-۴ الگوریتم ABC

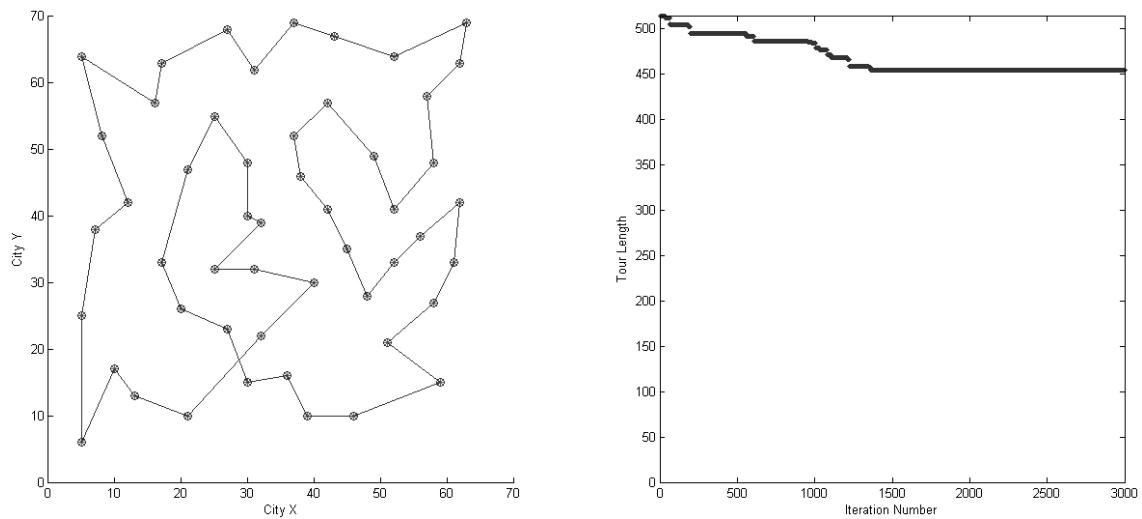
پارامترهای این الگوریتم به صورت زیر است:

$$nc = 20, ne = 10, no = 10, ns = 1, nl = \frac{nc \times n}{3}$$

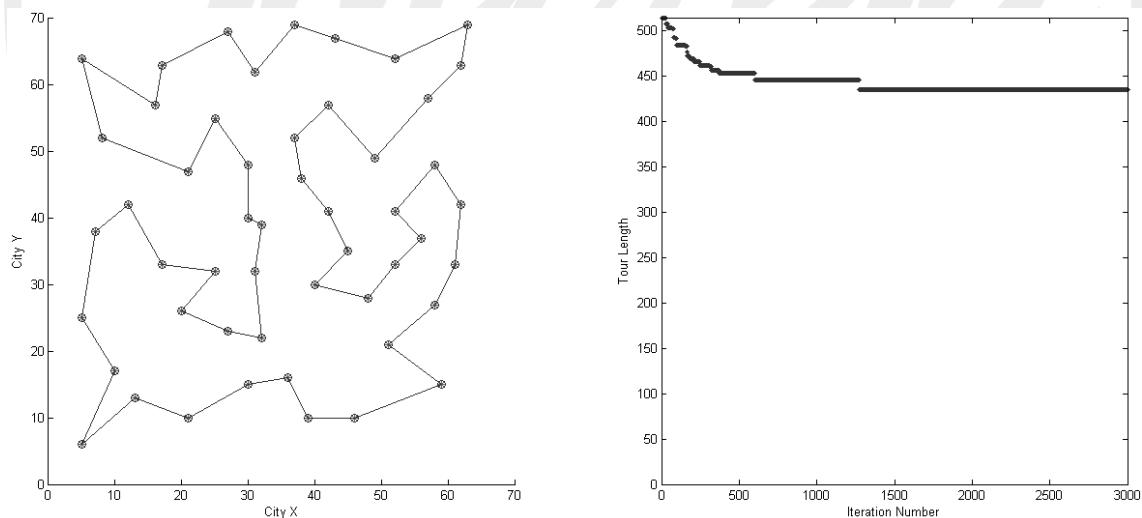
در این روش جمعیت کل کلونی ۲۰ زنبور قرار داده شده است که از آن‌ها ۱۰ زنبور به زنبورهای کارگر و ۱۰ زنبور به زنبورهای تماشاگر اختصاص داده شده است. هر کدام از این زنبورها نماینده یک جستجوی محلی هستند، بنابراین در هر تکرار ۲۰ جستجوی محلی صورت می‌پذیرد. همینطور زمانی که nl تکرار بگذرد و هیچ بهبودی در پاسخ یک زنبور اتفاق نیفتد آن زنبور به یک زنبور پیشاهنگ تبدیل می‌شود. ولیکن در هر تکرار تنها یک زنبور امکان جایگزینی جواب خود با مقدار مسیر نزدیکترین همسایگی جدید را در صورت برقراری این شرط دارد. در جدول ۳-۴ نتایج با دقت اعشاری برای مسائل مذکور نشان داده شده است و نحوه عملکرد الگوریتم در شکل‌های ۱۲-۴ تا ۱۷-۱۴ مشهود است.

جدول ۳-۴ مسافت مسیر نهایی الگوریتم CABC

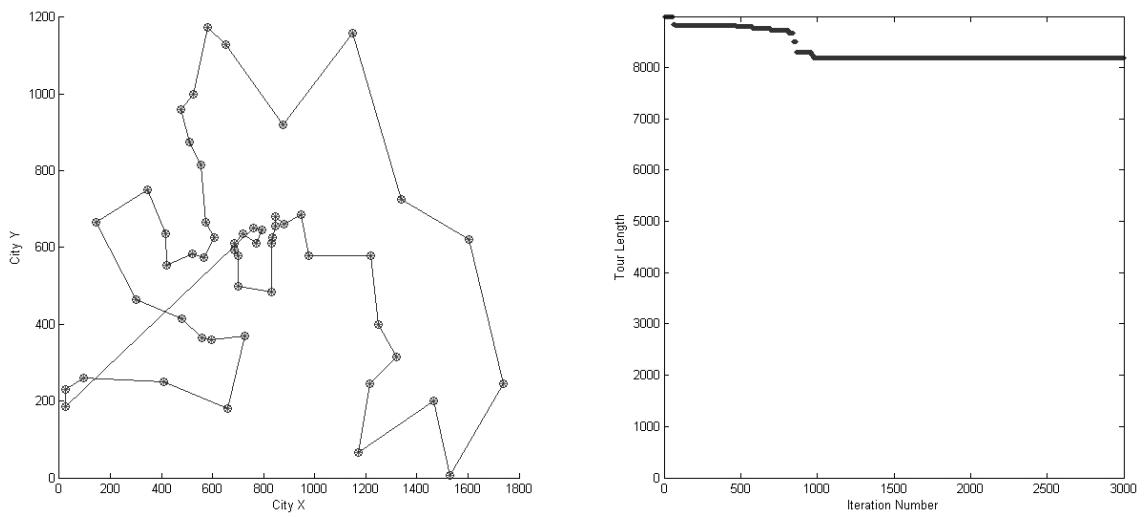
نام مسئله	CABC	
	2-opt	GSTM
eil51	۴۵۴/۸۳	۴۳۵/۵۱۲
berlin52	۸۱۸۲/۱۹	۷۶۸۶/۰۵
kroA100	۲۳۸۹۹/۲	۲۱۶۵۰/۸



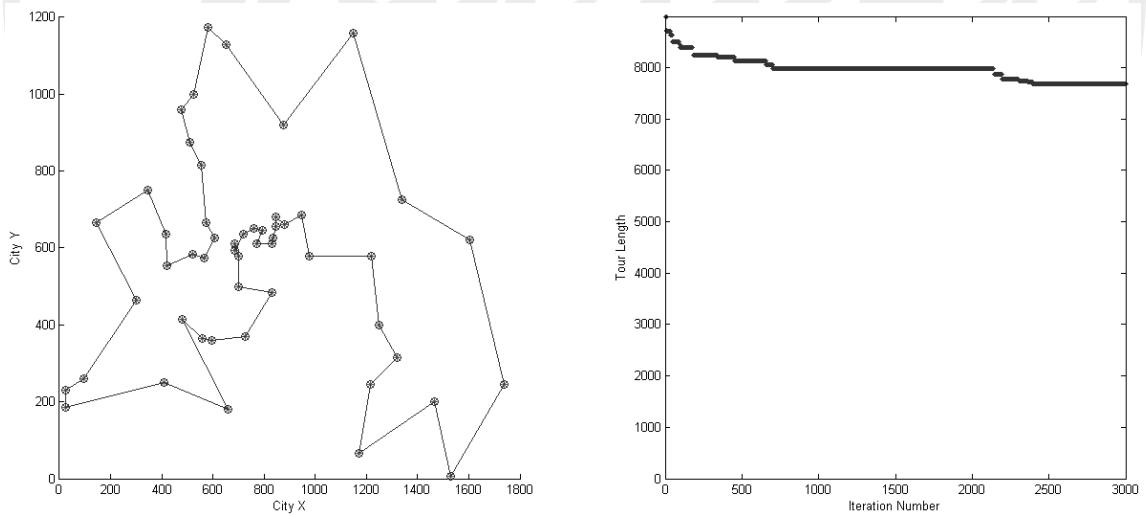
شکل ۱۲-۴ پاسخ الگوریتم CABC در ۳۰۰۰ تکرار با استفاده از ۲-opt برای مسئله eil51



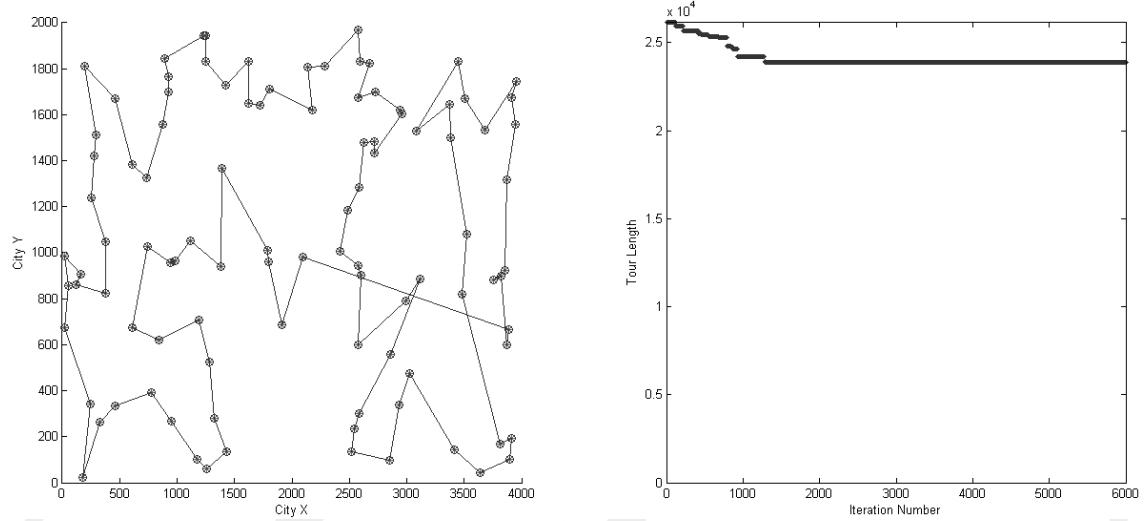
شکل ۱۳-۴ پاسخ الگوریتم CABC در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله eil51



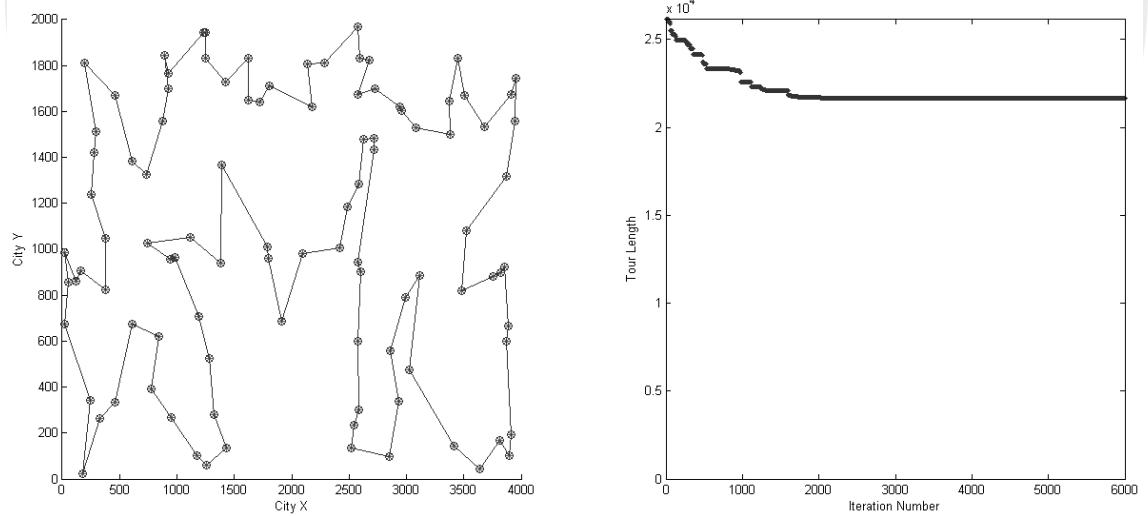
شکل ۱۴-۴ پاسخ الگوریتم CABC در ۳۰۰۰ تکرار با استفاده از 2-opt برای مسئله berlinc52



شکل ۱۵-۴ پاسخ الگوریتم CABC در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله berlinc52



شکل ۱۶-۴ پاسخ الگوریتم CABC در ۶۰۰۰ تکرار با استفاده از opt-2 برای مسئله kroA100



شکل ۱۷-۴ پاسخ الگوریتم CABC در ۶۰۰۰ تکرار با استفاده از GSTM برای مسئله kroA100

BCO ۳-۳-۴ الگوریتم

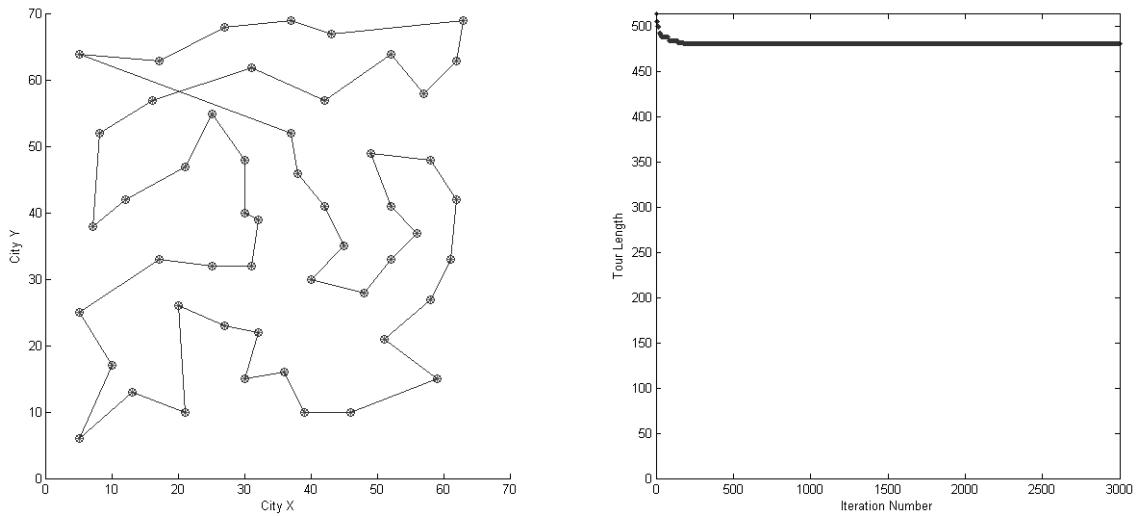
در میان چهار الگوریتم دیگر این الگوریتم کمترین تعداد پارامتر را داراست که خود یک مزیت به حساب می‌آید. این پارامترها در زیر نشان داده شده است:

$$nc = 10, nfp = 2, ns = 1$$

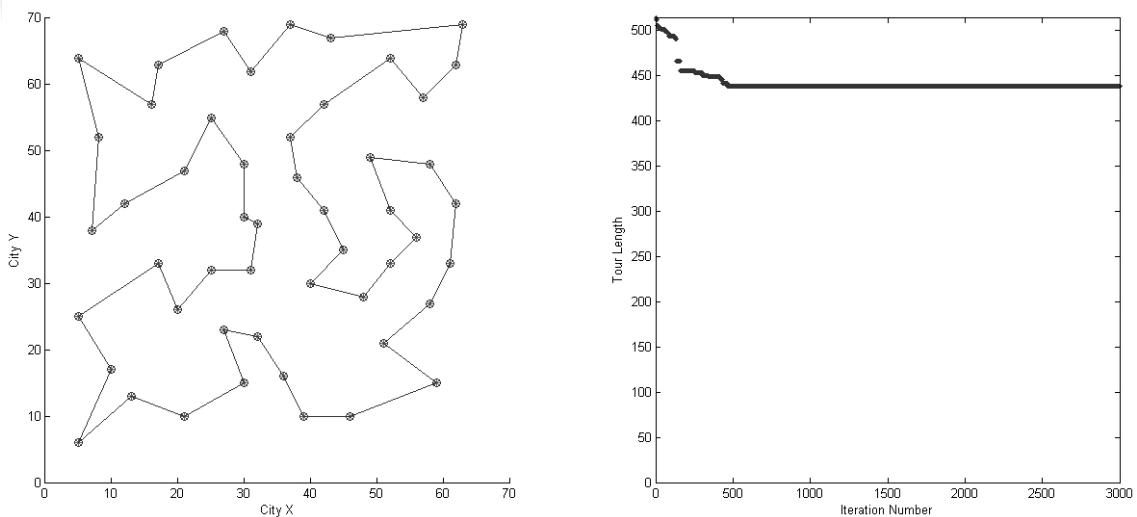
تعداد زنبورهای کلونی ۱۰ عدد در نظر گرفته شده است که هر زنبور معادل یک جستجوی محلیست. تعداد گذر پیشرو و پسرو هم دو بار در نظر گرفته شده است که در مجموع در هر تکرار ۲۰ جستجوی محلی شکل می‌گیرد. می‌توان تعداد زنبورها را ۲۰ عدد در نظر گرفت و با یک گذر پیشرو و پسرو به همین نتیجه رسید؛ ولیکن ویژگی کلیدی این الگوریتم این دو گذر و نحوه سربازگیری آن است. در هر تکرار زنبور با بدترین مقدار تابع شایستگی با یک مقدار جدید مسیر نزدیکترین همسایگی جایگزین می‌شود که این زنبور نقش جستجوی جهانی را ایفاء می‌کند. پر واضح است که تعداد مسیرهای جدید نزدیکترین همسایگی محدود بوده و این مسیرهای غیرتکراری به تعداد شهرها محدود می‌شوند. در نتیجه بعد از اینکه همه مسیرهای جدید را یکبار در روند اجرا قراردادیم این زنبور با جوابهای جدید جایگزین نخواهد شد و تنها به جستجو در اطراف آخرین مسیر جایگزین خواهد پرداخت. در کنار این زنبور، ۹ زنبور دیگر با بهترین پاسخ نیز به جستجوی محلی مشغول هستند. نتایج حاصل در جدول ۴-۴ و شکل‌های ۱۸-۴ تا ۲۳-۴ مشاهده می‌شود.

جدول ۴-۴ مسافت مسیر نهایی الگوریتم BCO

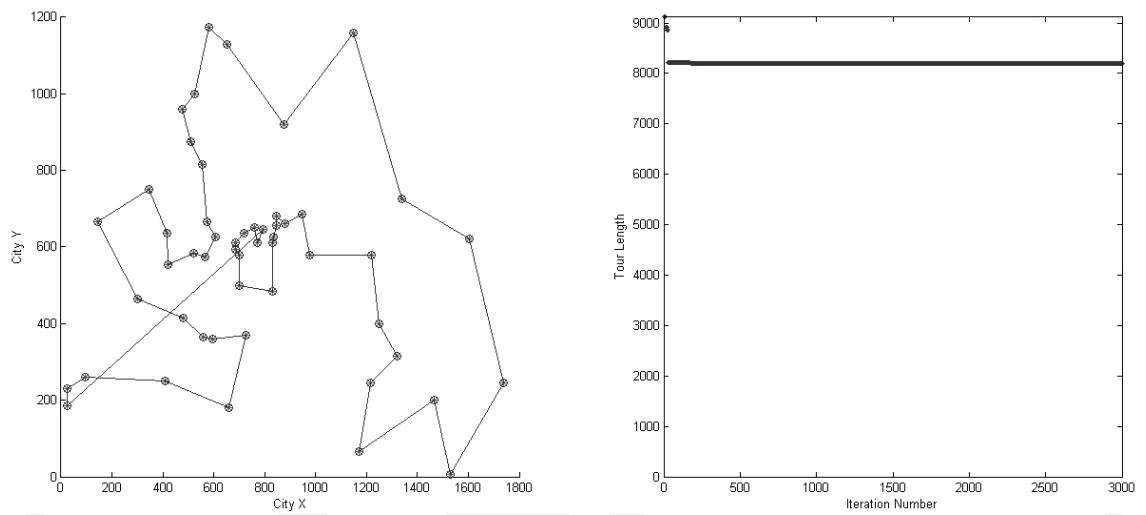
نام مسئله	BCO	
	2-opt	GSTM
eil51	۴۸۰/۹۰۲	۴۳۸/۷۰۳
berlin52	۸۲۰۲/۰۸	۷۶۷۶/۸۳
kroA100	۲۴۶۹۸/۵	۲۴۳۳۰/۵



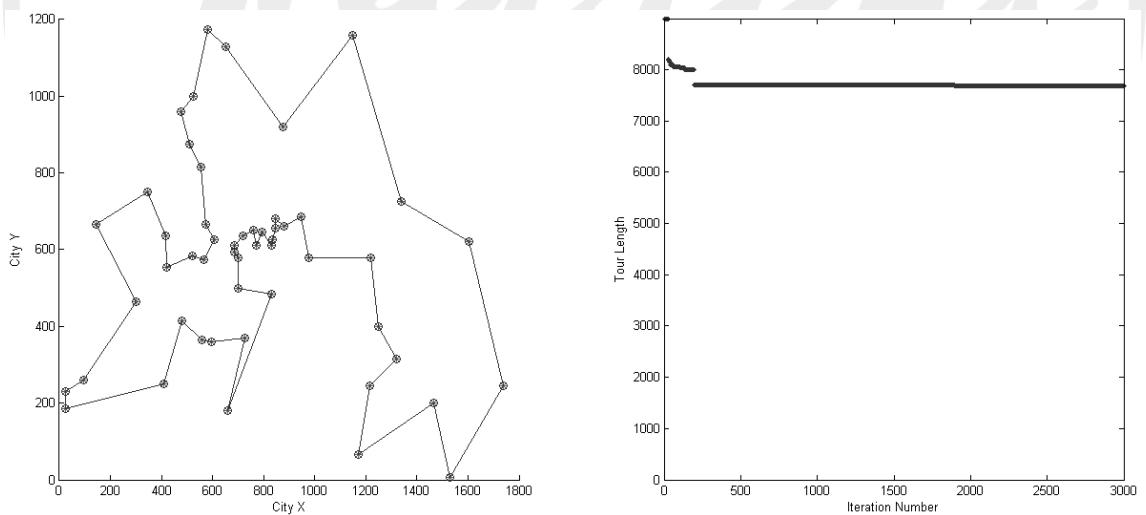
شکل ۱۸-۴ پاسخ الگوریتم BCO در ۳۰۰۰ تکرار با استفاده از 2-opt برای مسئله eil51



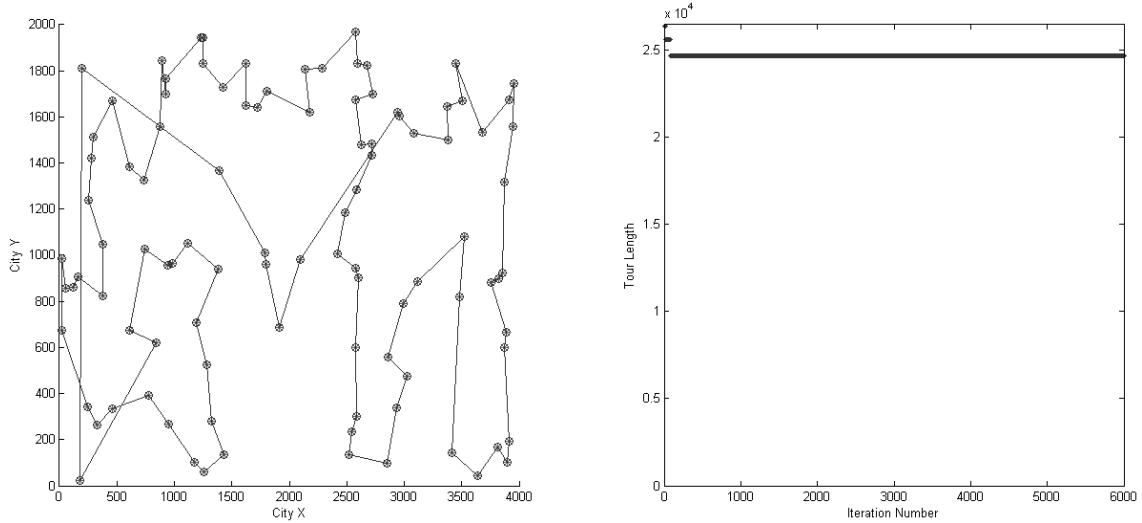
شکل ۱۹-۴ پاسخ الگوریتم BCO در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله eil51



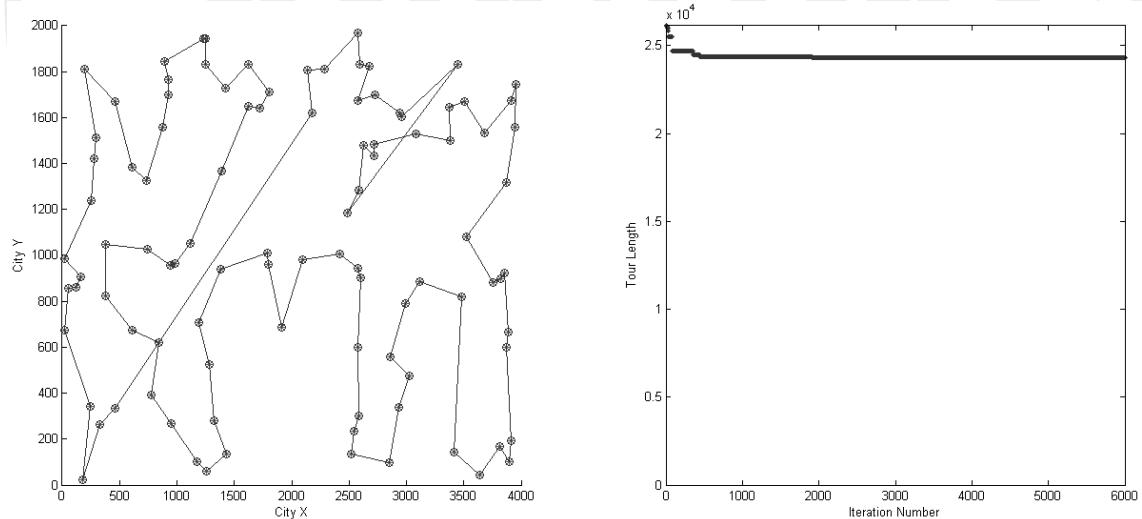
شکل ۲۰-۴ پاسخ الگوریتم BCO در ۳۰۰۰ تکرار با استفاده از 2-opt برای مسئله berlin52



شکل ۲۱-۴ پاسخ الگوریتم BCO در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله berlin52



شکل ۲۲-۴ پاسخ الگوریتم BCO در ۶۰۰۰ تکرار با استفاده از 2-opt برای مسئله kroA100



شکل ۲۳-۴ پاسخ الگوریتم BCO در ۶۰۰۰ تکرار با استفاده از GSTM برای مسئله kroA100

۴-۳-۴ الگوریتم پیشنهادی (HBA)

پارامترهای الگوریتم پیشنهادی به صورت زیر در نظر گرفته شده است:

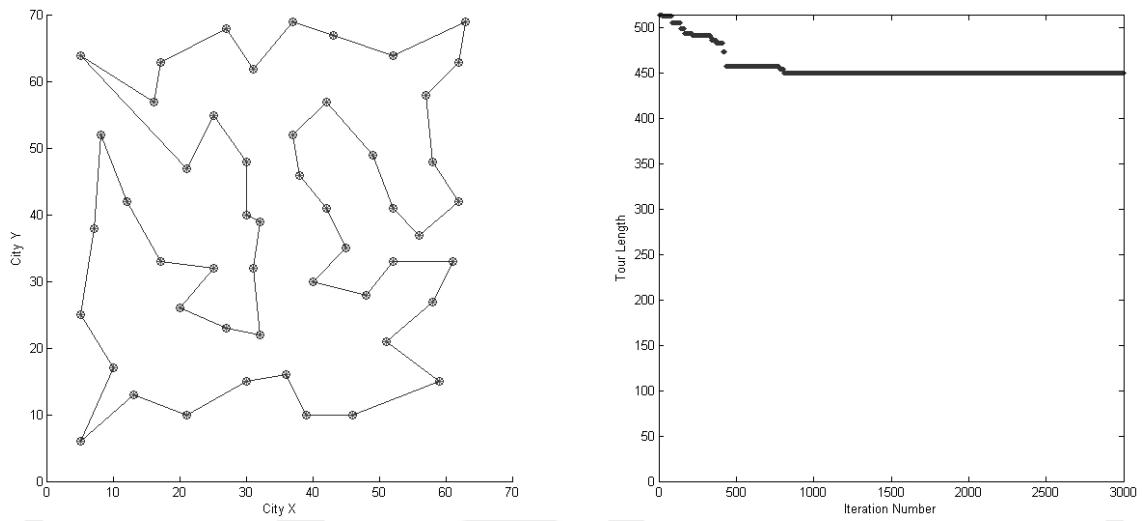
$$no = 10, ne = 2, ns = 1, nl = \frac{nc \times n}{3}, is = \frac{\text{iterations}}{n - no + 1}$$

زنبورهای تماشاگر که وظیفه نگهداری پاسخ را به عهده دارند ۱۰ عدد در نظر گرفته شده است و به هر زنبور تماشاگر دو زنبور کارگر اختصاص داده شده است که با احتساب آن در هر تکرار ۲۰ پویش محلی به واسطه زنبورهای کارگر انجام می‌پذیرد.

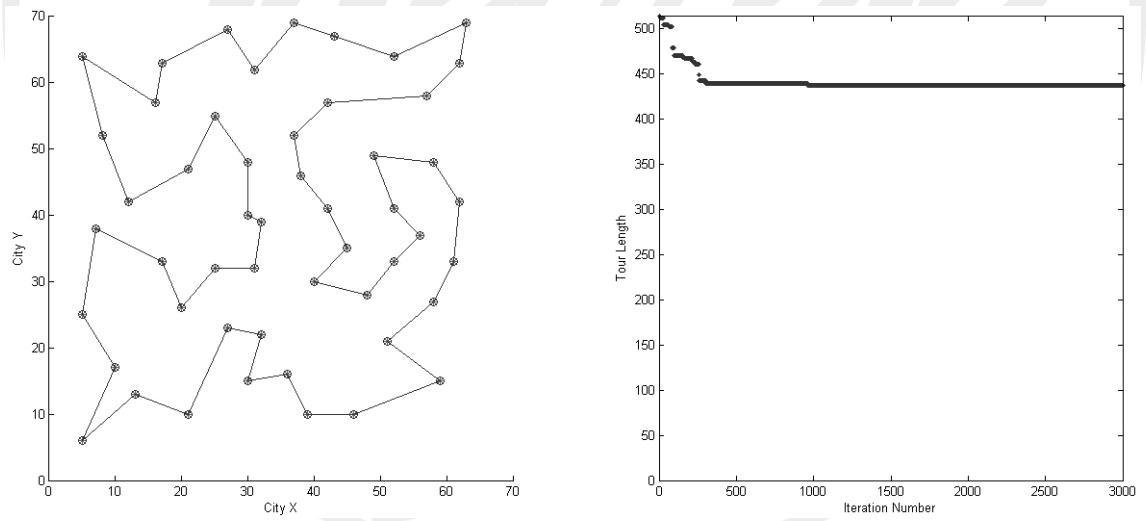
همانطور که قبلاً هم توضیح داده شد، زنبورها با بدترینتابع شایستگی در تعداد از پیش تعیین شده-ای تکرار با جواب جدیدی جایگزین می‌شوند. تعداد زنبورهای پیشنهادی یکی در نظر گرفته شده است. هدف اصلی آنست که قبل از اینکه جواب جدید نادیده گرفته شود، بر روی آن بررسی‌های لازم به عمل آید. به همین دلیل is باید مقداری را اختیار کند که تضمین کند اولاً تمامی شهرها یکبار در روند اجرا وارد شوند و دوماً بیشترین تعداد بررسی ممکن بر روی آن صورت می‌گیرد. در این پارامتر مقدار n تعداد شهرها بوده و صورت این کسر تکرارهای کل الگوریتم است. در ادامه جدول ۴-۵ و شکل‌های ۲۹-۴ تا ۲۹-۶ نتایج حاصل را به تصویر کشیده‌اند.

جدول ۴-۵ مسافت مسیر نهایی الگوریتم HBA

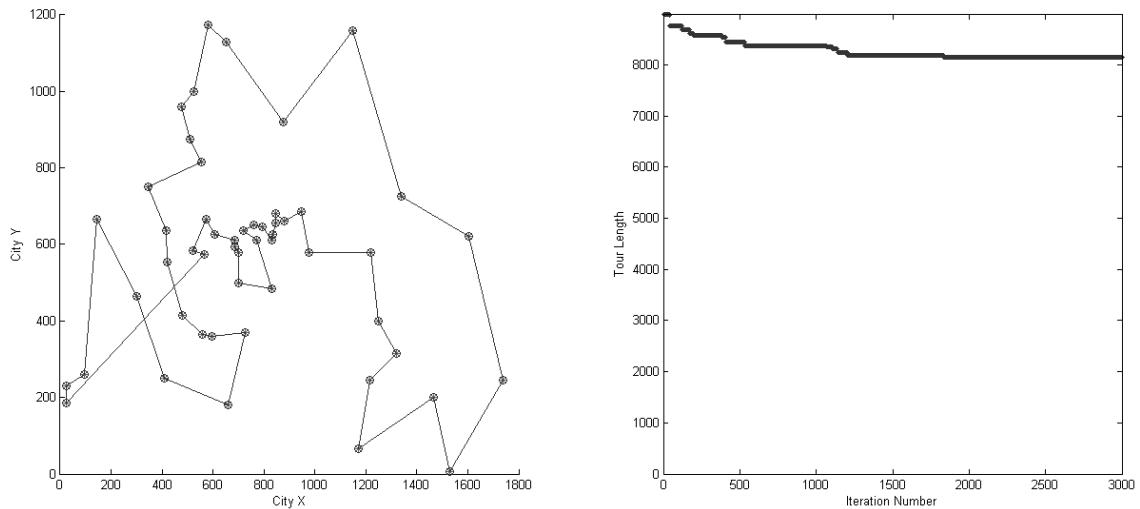
نام مسئله	HBA	
	2-opt	GSTM
eil51	۴۵۰/۳۹۳	۴۳۷/۰۶۲
berlin52	۸۱۵۹/۵۶	۷۹۷۴/۵۴
kroA100	۲۳۱۳۴/۸	۲۲۳۸۸/۱



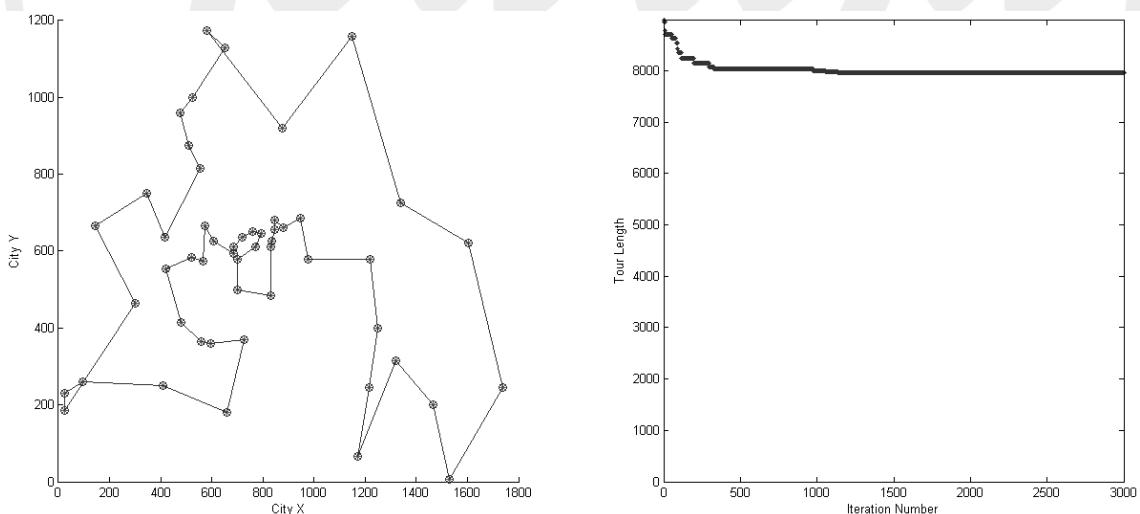
شکل ۲۴-۴ پاسخ الگوریتم HBA در ۳۰۰۰ تکرار با استفاده از 2-opt برای مسئله eil51



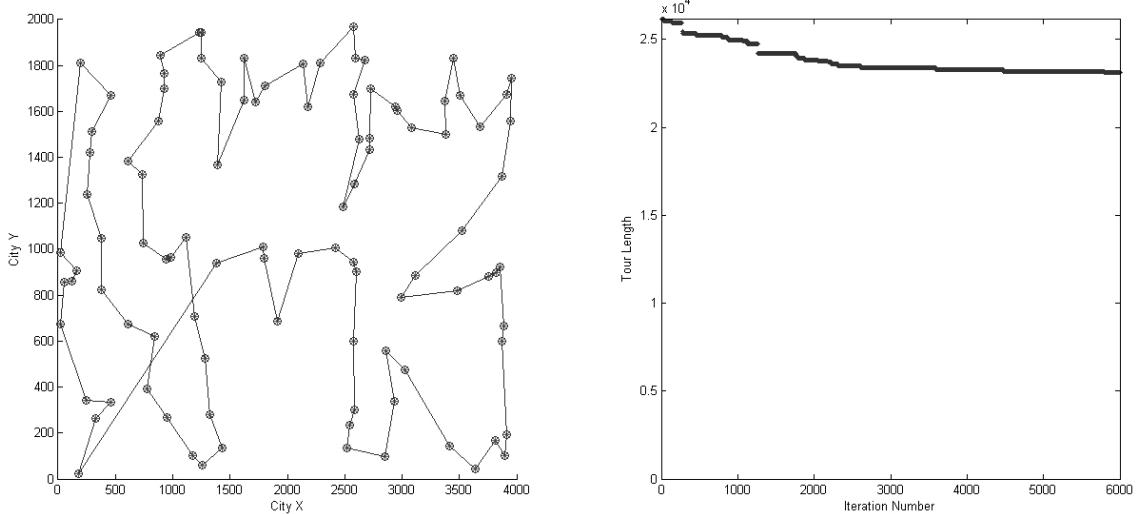
شکل ۲۵-۴ پاسخ الگوریتم GSTM HBA در ۳۰۰۰ تکرار با استفاده از GSTM برای مسئله eil51



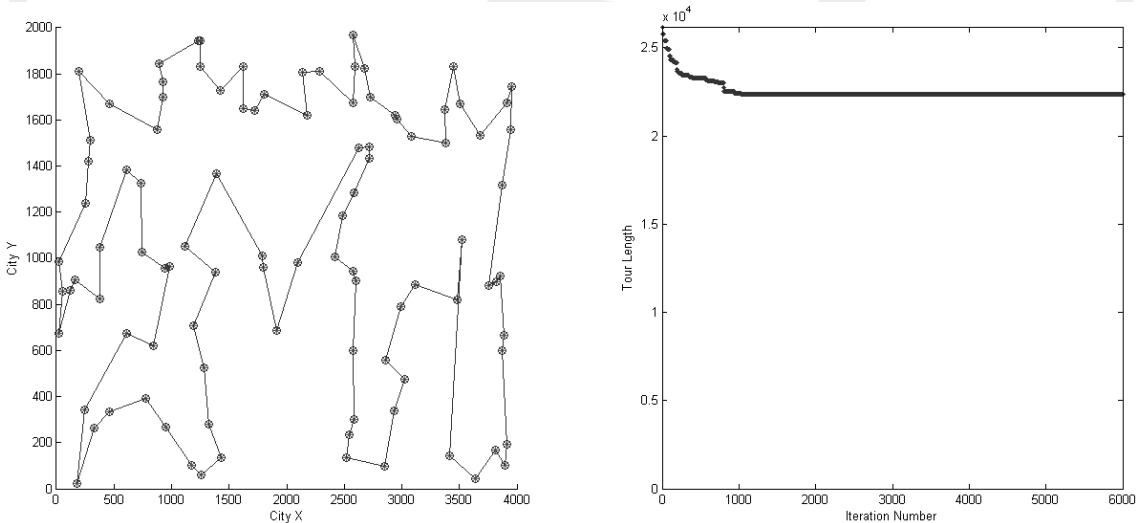
شکل ۲۶-۴ پاسخ الگوریتم HBA در ۳۰۰۰ تکرار با استفاده از 2-opt برای مسئله ۲



شکل ۲۷-۴ پاسخ الگوریتم GSTM HBA در ۳۰۰۰ تکرار با استفاده از 2-opt برای مسئله ۲



شکل ۲۸-۴ پاسخ الگوریتم HBA در ۶۰۰۰ تکرار با استفاده از 2-opt برای مسئله kroA100



شکل ۲۹-۴ پاسخ الگوریتم HBA در ۶۰۰۰ تکرار با استفاده از GSTM برای مسئله kroA100

۴-۴ نتایج پیاده‌سازی سخت‌افزاری

در زیر قسمت‌های بعدی نتایج حاصل از پیاده‌سازی سخت‌افزاری مشاهده می‌شود. در ابتدا حجم سخت‌افزار مصرف شده برای پیاده‌سازی هر یک از هسته‌های زیرمجموعه و پس از آن عملکرد کلی الگوریتم نشان داده شده است.

۱-۴-۴ هسته‌های فرعی الگوریتم HBA

در جدول ۶-۴ واحدهای اصلی تر این پیاده‌سازی برای مسئله eil51 به همراه جزئیات منابع سخت-افزاری مورد نیاز هر یک آورده شده است. می‌دانیم که یک برش از جداول جستجو و فلیپ‌فلاب‌ها تشکیل شده است، ولی ممکن است در ساخت یک هسته تمام قسمت‌های یک برش مصرف نشود؛ به عنوان مثال تنها از جداول جستجوی آن استفاده شود. به همین دلیل نرم‌افزار ISE نتایج حاصل از حجم مصرفی را دقیق‌تر و در قالب تعداد عناصر مصرف شده ارائه می‌دهد. در جدول ۶-۴، ثبات‌ها مشخص‌کننده تعداد کل ثبات‌هایی است که برای پیاده‌سازی مصرف شده است، این ثبات‌ها در برش‌های متفاوتی پخش شده‌اند. همچنین جداول جستجو اشاره به تعداد کل جداول جستجو دارد و این جداول هم در برش‌های مختلفی هستند. اگر این جداول و فلیپ‌فلاب‌ها در ترکیب با هم و در یک برش بکار برد شوند، زوج جدول ثبات را نتیجه می‌دهند. به بیانی دیگر اگر این زوج‌ها را تقسیم بر تعداد جداول و فلیپ‌فلاب‌های موجود در هر برش کنیم، نتیجه تعداد کل برش‌هایی خواهد بود که کاملاً استفاده شده‌اند. پردازنده سیگنال دیجیتالی هم مشخص کننده تعداد واحدهای DSP48E تعبیه شده است که این واحدها هر یک شامل یک انباره، یک ضرب‌کننده و یک جمع‌کننده می‌شوند. در انتهای جدول ۶-۴ هم تعداد بلوک‌های رم مصرف شده نشان داده شده است.

مشاهده می‌شود که از منابع پردازنده سیگنال دیجیتالی تنها در هسته‌هایی استفاده شده است که شامل واحدهای محاسبه‌گر فرمول آدرس رم یا رام و محاسبه‌گر فاصله اقلیدسی هستند. به همین ترتیب تنها واحدهایی از بلوک‌های رم استفاده کرده‌اند که شامل واحد رم یا رام می‌شوند. در جدول ۶-۴ بیشترین حجم منابع سخت‌افزاری مربوط به ینج واحد تولید کننده مسیر نزدیکترین همسایگی، تشخیص نزدیکترین همسایگی، بهینه‌سازی محلی، تولید کننده ماتریس فاصله‌ها روی حافظه و

تولیدکننده آدرس و محتوای ماتریس فاصله‌ها است که شامل بیشترین تعداد زیرهسته نیز می‌شوند.

دلیل حجم بالای دو هسته آخر مذکور وجود هسته محاسبه‌گر جذر است که یک هسته IP است.

جدول ۶-۴ منابع سخت‌افزاری استفاده شده توسط زیرهسته‌ها برای مسئله ۱۵۱

نام کارهای نمود	نمود	نام	نام	نام	نام	نام هسته
۳	۲	۱۸۴	۶۲۲	۲۱۷		تولید کننده مسیر نزدیکترین همسایگی
۳	۲	۱۲۷	۴۵۲	۱۵۷		تشخیص نزدیکترین همسایه
۰	۰	۱۹	۸۷	۱۹		تولید کننده آدرس ترتیبی دو شهر
۱	۰	۲۴	۱۴۱	۲۴		مسیر حرکت
۰	۲	۳۳	۱۰۴	۵۳		محاسبه‌گر آدرس رم یا رام
۰	۲	۲۳	۷۸	۵۳		محاسبه‌گر فرمول آدرس
۰	۰	۴۰	۷۲	۴۰		مقایسه و ذخیره
۱	۰	۵۰	۱۸۵	۵۰		زنبور عسل
۲	۲	۳۲۶	۶۸۷	۳۶۹		محاسبات بهینه‌سازی محلی
۰	۰	۴۱	۱۰۷	۴۹		تولید تصادفی آدرس دو شهر
۰	۰	۶	۱۰	۸		تولید تصادفی آدرس در محدوده تعداد شهرها
۰	۰	۱۳	۱۰۲	۱۴		محاسبات بروزرسانی
۴	۶	۳۶۷	۸۴۲	۷۶۱		تولیدکننده ماتریس فاصله‌ها روی حافظه
۲	۶	۳۶۳	۸۰۳	۷۵۴		تولیدکننده آدرس و محتوای ماتریس فاصله‌ها
۰	۴	۲۷۴	۶۴۰	۶۸۲		محاسبه‌گر فاصله اقلیدسی
۰	۰	۱۶	۳۹	۱۶		تولیدکننده آدرس ترتیبی دو شهر برای حافظه

۴-۴-۴ هسته اصلی الگوریتم HBA

همانطور که در قبل هم گفته شد در پیاده‌سازی روش پیشنهادی بر روی سخت‌افزار پارامتر nl حذف شده است و پارامترهای باقیمانده به صورت زیر در نظر گرفته شده‌اند:

$$no = 10, ne = 2, ns = 1$$

همانند قسمت نرم‌افزاری تعداد جستجوهای محلی در هر تکرار با در نظر گرفتن ۱۰ زنبور تماشاگر و دو زنبور کارگر برای هر یک از این زنبورها به ۲۰ جستجو می‌رسد. همینطور پارامتر is به صورتی محاسبه شده است که تعداد کل تکرارها برای مسائل دارای کمتر از ۷۵ شهر از ۳۰۰۰ تکرار و مسائل دارای ۷۵ شهر و بیشتر به ۶۰۰۰ تکرار برسد و در عین حال همه مسیرهای نزدیک‌ترین همسایگی موجود و غیرتکراری مورد بررسی قرار بگیرند و این بررسی‌ها تا حد ممکن به بیشینه نزدیک باشند. همانند قسمت قبل میزان منابع سخت‌افزاری مورد نیاز با جزئیات برای هر مسئله در جدول ۷-۴ آورده شده است. به علاوه در این جدول نتیجهٔ نهایی حاصل از سنتز و شبیه‌سازی هر یک از این مسائل یعنی مسافت بهترین مسیر پیدا شده و زمان اجرای آن نیز آورده شده است.

در مورد زمان اجرا باید محدودیت فرکانسی تراشهٔ انتخاب شده و مهم‌تر از آن محدودیت فرکانسی^۱ هسته‌های پیاده‌سازی شده در نظر گرفته شود. این محدودیت فرکانسی با استفاده از تحلیل زمانی استاتیک^۲ هسته‌ها مشخص می‌شود.

- تراشهٔ بکار برده شده XC5VLX330 است و حداکثر فرکانس این تراشه با استفاده از ضرب‌کننده کلک دیجیتالی^۳ تعییه شده در تراشه به ۵۵۰ مگاهرتز می‌رسد که معادل کلای پالسی با طول تقریبی $1/8$ نانوثانیه می‌شود.
- در جدول ۷-۴ حداکثر فرکانس کاری الگوریتم HBA برای پنج مسئله آورده شده است. با توجه به تأخیر انتشار در سخت‌افزار، این فرکانس‌ها حداکثر فرکانسی هستند که این الگوریتم می‌تواند بدون خطا یرای آن مسئله به جواب نائل شود. مشاهده می‌شود که کمترین آن‌ها

¹ Timing Closure

² Static Timing Analysis

³ Digital Clock Multiplier (DCM)

۱۱۸ مگاهرتز و مربوط به مسئله eil51 است. این کلاک پالس طول تقریبی ۸/۵ نانوثانیه را

داراست.

با توجه به دو نکته بالا، از فرکانس ۱۱۱ مگاهرتز برای شبیه‌سازی استفاده شده است که معادل کلاکی

پالسی به طول ۹ نانوثانیه است.

لازم به ذکر است که نتایج موجود در جدول ۷-۴ حاصل تنظیم نرم‌افزار سیتاسایزر بر روی موازن‌های

از سرعت و حجم منابع مصرفی در روند سیتاسایز است.

جدول ۷-۴ پارامتر is، نتایج و منابع سخت‌افزاری مصرف شده در پیاده‌سازی هسته HBA

ردیف ردیف ردیف	نام نام نام	دیزاین دیزاین دیزاین	نام نام نام							
۲۸	۲۲	۳۸۶۳	۱۰۳۰۰	۴۳۹۷	۶/۶۳	۱۱۸	۴۴۹	۳۰	eil51	
۲۸	۲۲	۳۸۳۴	۹۹۱۶	۴۳۹۸	۶/۸۶	۱۲۰	۸۰۰۸	۳۲	berlin52	
۵۰	۲۲	۴۰۲۱	۱۰۵۹۳	۴۵۲۹	۱۶/۲۲	۱۲۹	۷۱۲	۲۳	st70	
۵۰	۲۲	۳۹۹۲	۱۰۱۲۲	۴۵۳۰	۲۰/۶۵	۱۲۰	۵۷۵	۲۱	eil76	
۵۰	۲۲	۳۹۹۲	۱۰۱۲۲	۴۵۳۰	۲۰/۶۵	۱۲۰	۱۱۷۶۶۹	۲۱	pr76	

۵. نتیجه‌گیری

در این بخش در ابتدا به بررسی نتایج حاصل از پیاده‌سازی نرم‌افزاری پرداخته، سپس نتایج حاصل از قسمت سخت‌افزار تحلیل می‌شود. در بخش جمع‌بندی خلاصه نتایج حاصل بیان می‌شود و بخش آخر هم به پیشنهادات کارهای آتی و توسعه اختصاص داده شده است.

۱-۵ بررسی نتایج پیاده‌سازی نرم‌افزاری

جدول ۱-۵ مقایسه نتایج چهار الگوریتم زنبور

نام مسئله	بهترین	HBA		BA		CABC		BCO	
		2-opt	GSTM	2-opt	GSTM	2-opt	GSTM	2-opt	GSTM
eil51	۴۲۶	۴۴۹	۴۳۷	۴۵۴	۴۳۵	۴۵۶	۴۳۶	۴۵۷	۴۳۹
berlin52	۷۵۴۲	۷۷۳۱	۷۹۷۵	۷۸۸۷	۷۶۰۷	۸۰۹۲	۷۶۸۶	۸۱۸۲	۷۶۷۷
st70	۶۷۵	۷۱۱	۷۳۰	۷۱۱	۶۹۲	۷۱۶	۷۰۲	۷۱۳	۷۳۱
eil76	۵۳۸	۵۷۴	۵۶۵	۵۶۶	۵۵۶	۵۷۸	۵۶۳	۵۹۲	۵۹۶
pr76	۱۰۸۱۵۹	۱۱۳۰۱۵	۱۱۰۸۶۹	۱۱۲۲۲۱	۱۰۹۵۵۳	۱۱۳۶۷۰	۱۱۰۷۸۵	۱۱۳۳۶۶	۱۱۰۴۳۹
rat99	۱۲۱۱	۱۳۲۴	۱۳۰۷	۱۲۵۹	۱۲۴۵	۱۳۱۵	۱۲۵۸	۱۳۵۶	۱۳۰۹
kroA100	۲۱۲۸۲	۲۳۱۳۵	۲۲۳۸۸	۲۲۲۴۳	۲۱۶۱۵	۲۳۸۹۹	۲۱۶۰۱	۲۴۶۹۹	۲۴۳۳۱
eil101	۶۲۹	۶۷۹	۷۰۰	۶۷۱	۶۶۳	۷۱۱	۶۶۶	۶۹۵	۶۶۹
lin105	۱۴۳۷۹	۱۷۵۶۶	۱۶۰۳۴	۱۵۰۸۷	۱۴۵۷۴	۱۷۶۵۵	۱۵۲۵۵	۱۵۳۵۶	۱۴۷۰۶
pr107	۴۴۳۰۳	۴۵۳۳۸	۴۴۹۸۲	۴۵۳۳۸	۴۴۶۲۳	۴۵۶۶۷	۴۵۰۹۸	۴۵۳۳۸	۴۴۷۰۵

جدول ۱-۵ حاوی نتایج پیاده‌سازی چهار الگوریتم زنبور برای ۱۰ مسئله و دو نوع جستجوی محلی است. در این جدول بهترین نتیجه حاصل بین تمامی الگوریتم‌ها برای روش 2-opt با خاکستری کم رنگ و برای روش GSTM با خاکستری پررنگ نشان داده شده است. همچنین بهترین جواب موجود برای هر مسئله هم در این جدول قرار داده شده است.

همانطور که از جدول ۱-۵ مشخص است در مسائلی با تعداد شهرهای کمتر از ۷۵ و با استفاده از روش جستجوی محلی 2-opt نتایج حاصل از الگوریتم پیشنهادی در صدر جدول قرار گرفته است. همچنین الگوریتم BA با استفاده از روش 2-opt و در مسائلی با تعداد شهرهای بیشتر از ۷۵ بهترین نتیجه را داشته است. ضمن اینکه این الگوریتم با استفاده از روش جستجوی محلی GSTM بهینه‌ترین پاسخ‌های مسئله را نیز پیدا کرده است.

دلیل اینکه الگوریتم BA بهترین نتیجه را می‌دهد مربوط به جستجوی جهانی بسیار قوی‌تر این الگوریتم بوده، چراکه در چند تکرار اول تمامی مسیرهای نزدیکترین همسایگی را بررسی کرده و بهترین آن‌ها را انتخاب کرده و بقیه تکرارها را به بهینه‌سازی همان پاسخ‌ها می‌پردازد. این روند در مسائلی که تعداد شهرها افزایش یافته و بالاتر از ۷۵ قرار می‌گیرد کاملاً باعث تفاوت و برتری این الگوریتم نسبت به بقیه الگوریتم‌ها خواهد شد. ولی در مسائلی که تعداد شهرها کمتر بوده و در استفاده از روش بهینه‌سازی سنتی انتخاب شده (2-opt) موثر نیست، چراکه در ابتدا اعمال این روش (دوران) بر روی یک مسیر تا حد مشخصی امکان بهینه‌سازی را مهیا می‌کند و بیشتر از آن نیاز به عملگرهای پیچیده‌تری همانند GSTM دارد. همینطور حذف سریع تعدادی از راه حل‌ها نتیجه‌ای جز محدود شدن میزان بهینه‌سازی را به دنبال خواهد داشت.

جایگاه بعد از الگوریتم زنبور، مربوط به کلونی زنبور مصنوعی بوده که بر خلاف BA تمامی جواب‌ها را بررسی کرده ولی جستجوی جهانی ضعیفتری را نسبت به آن دارد. این مشکل سبب می‌شود که نتایج حاصل از CABC ضعیفتر از الگوریتم زنبور باشند.

جدای چند مورد الگوریتم BCO دارای بدترین نتایج در بین چهار الگوریتم بوده، چراکه اولاً این الگوریتم ذاتاً دارای جستجوی جهانی مناسبی نبوده و بدین منظور در اینجا روشی برای آن در نظر

گرفته شد که در غیراینصورت این الگوریتم به جواب بهینه همگرا نمی‌شود. همچنین به دلیل نوع سربازگیری در این روش، بعد از تعدادی تکرار بیشتر زنبورهای آن دارای جواب یکسانی بوده و در واقع خیلی سریع در یک بهینه محلی محصور شده و امکان خروج آن نیز مهیا نیست.

دلیل اینکه نتایج حاصل از الگوریتم پیشنهادی با استفاده از روش GSTM پایین‌تر از نتایج دو الگوریتم دیگر یعنی BA و CABC قرار می‌گیرد، نحوه اختصاص جستجوهای هر تکرار است. در این دو الگوریتم جستجوهای محلی در اطراف جواب بهینه‌تر بیشتر بوده که باعث نائل شدن به نتایج بهتر برای آن جواب می‌شود. در حالیکه در الگوریتم پیشنهادی برای تمامی زنبورها تعداد جستجوهای محلی یکسانی در نظر گرفته شده است که باعث افت نتایج در زمان بکارگیری یک روش بهینه‌سازی پیچیده‌تر ولی موثرتر مثل GSTM می‌شود. ولیکن باید در نظر داشت که این روش به منظور پیاده‌سازی موثر بر روی سخت‌افزار ارائه شده است و رسیدن به نتایجی حدود نتایج دیگر الگوریتم‌ها و گاهی بهتر از آن‌ها نشان از کاربردی بودن روش پیشنهادی دارد.

در جدول ۲-۵ مقایسه‌ای بین چهار الگوریتم از نظر زمانی انجام شده است. زمان درج شده در این جدول برای حل مسئله eil51 با ۳۰۰۰ تکرار است. همانند جدول ۱-۵ بهترین نتیجه از روش 2-opt با خاکستری کم‌رنگ و بهترین نتیجه از روش GSTM با خاکستری پررنگ نشان داده شده است.

جدول ۲-۵ مقایسه زمان اجرای چهار الگوریتم زنبور برای مسئله eil51

روش	HBA		BA		CABC		BCO	
بهینه‌سازی	2-opt	GSTM	2-opt	GSTM	2-opt	GSTM	2-opt	GSTM
زمان (ثانیه)	۱۶/۲۹	۱۸/۴۹	۱۶/۱۵	۱۸/۱۵	۱۴/۸۵	۱۶/۸۳	۱۶/۷۴	۱۸/۴۴

بهترین زمان اجرا بین این الگوریتم‌ها، مربوط به روش CABC بوده و دلیل آن تعداد جستجوی جهانی کمتر یا به عبارتی استفاده کمتر از محاسبات مسیر نزدیکترین همسایه است. نتایج حاصل از سه الگوریتم دیگر بسیار نزدیک به هم بوده و تفاوت چندانی نمی‌کنند. الگوریتم HBA زمان بسیار ناچیزی از همتای خود یعنی BA بیشتر طول می‌کشد که دلیل آن تعداد دفعات بیشتر فراخوانی تابع نزدیکترین همسایگی است.

۲-۵ بررسی نتایج پیاده‌سازی سخت‌افزاری

می‌دانیم که پیاده‌سازی توابع پیچیده بر روی FPGA به راحتی امکان‌پذیر نیست و در صورت پیاده‌سازی هم منابع بسیار زیادی از تراشه انتخابی را مصرف کرده که نتیجتاً هسته حاصل کاربرد مناسبی نخواهد داشت. همانطور که در فصل سوم بیان شد روش opt-2 برای پیاده‌سازی در نظر گرفته شده است که منابع سخت‌افزاری کمتری را اشغال کرده، سرعت بیشتری داشته و هم واحد بروزرسانی کوچکتر با محاسبات کمتر و سرعت بالاتر را نتیجه می‌دهد. در جدول ۱-۵ مشاهده شد که نتایج حاصل از روش opt-2 در مسائلی با تعداد شهرهای کمتر از ۷۵، نتایج مناسبی داشته است و اگرچه نسبت به بهترین راه حل ممکن فاصله دارد، ولی در میان دیگر الگوریتم‌ها نتایج قابل قبولی را ارائه داده است.

بعضی از نتایج حاصله از سخت‌افزار نسبت به نتایج نرم‌افزار کمی پایین‌تر قرار می‌گیرد که می‌توان دو دلیل برای آن شمرد. اول آنکه در پیاده‌سازی سخت‌افزاری به منظور جلوگیری از ایجاد پیچیدگی، کاهش موازی‌سازی و افزایش زمان نتیجه‌دهی پارامتر n حذف شد که این خود کمی باعث افت نتایج می‌شود. دلیل دوم آنست که تولید اعداد تصادفی در نرم‌افزار بسیار نزدیک به اعداد تصادفی واقعی بوده؛ ولیکن در پیاده‌سازی انجام شده از یکی از روش‌های تولید اعداد نیمه تصادفی بهره گرفته شده است [۳۸]. در این مرجع کمترین حجم استفاده شده و همچنین کم کیفیت‌ترین نوع تولید اعداد تصادفی به روش LFSR نسبت داده شده است. ولیکن با توجه به هدف اصلی از پیاده‌سازی یعنی کاهش زمان به وسیله موازی‌سازی و کاهش مصرف منابع این نوع اعداد نیمه تصادفی مناسب این پیاده‌سازی است و البته باید در نظر داشت که جواب‌های حاصل تفاوت معناداری با جواب‌های حاصل از نرم‌افزار ندارند و بسیار نزدیک به هم هستند.

اگر بخواهیم به صورت شهودی زمان اجرای دو نوع پیاده‌سازی نرم‌افزاری و سخت‌افزاری را مقایسه کنیم، به دلیل موازی‌سازی ۱۰ زنبور زمان حاصل از پاسخ سخت‌افزار باید ۱/۰ زمان حاصل از نرم‌افزار باشد، ولیکن با در نظر گرفتن اینکه پردازنده استفاده شده فرکانس کاری بیش از چهار برابر

تراشهٔ Virtex-5 را داشته و از دو هسته نیز تشکیل شده است، زمان حاصل تنها باید به ۰/۸ کاوش یابد. ولی در واقع کاوش زمانی سخت‌افزار نسبت به نرم‌افزار رقمی به مراتب کمتر است. چراکه در این پیاده‌سازی بیشینه موافقی سازی در نظر گرفته شده است. به عنوان مثال هستهٔ مسیر نزدیکترین همسایگی بعد از قرار دادن ۱۰ مسیر اول در زنبورها به صورت کاملاً موافق عمل کرده و با اینکه زمان اجرای آن نسبت به بقیه هسته‌ها بیشتر بوده، ولیکن با استفاده از موافقی سازی این محدودیت رفع شده است و همینطور برای دیگر هسته‌ها همانند تولید تصادفی آدرس دو شهر. لازم به ذکر است که محاسبات سخت‌افزار در بهینه‌ترین شکل بوده؛ بدین معنی که در واحد محاسبات بهینه‌سازی به منظور محاسبه نتیجهٔ حاصل از اعمال روش opt-2 تنها چهار فاصلهٔ مورد نیاز از محل رام خوانده می‌شود. ولیکن در نرم‌افزار به منظور ساده‌سازی طول کل مسیر با استفاده از فاصلهٔ اقلیدسی دوباره محاسبه می‌شود. لازم به توجه است که به دلیل تاخیر انتشار بسیار کم در مدارهای FPGA و ذات سخت‌افزاری، کلاً سرعت سخت‌افزار در مقایسه با نرم‌افزار بالاتر است. به عنوان یک مثال، بررسی یک شرط در FPGA و با مدارهای ترکیبی به اندازهٔ تاخیر انتشار مدارات ترکیبی طول می‌کشد؛ ولی بررسی همین شرط بر روی نرم‌افزار ممکن است به ۵ یا ۶ کلک پالس نیاز داشته باشد.

در عین بیان مطالب قبل، ولیکن مقایسهٔ درست باید در شرایط یکسان و حداقل در بستری یکسان انجام گیرد؛ بدین معنی که با پیاده‌سازی این الگوریتم بر روی هسته‌های ریزپردازندهٔ نرم یا سخت بر روی FPGA و سپس مقایسه نتایج، به مقایسه‌ای دقیق و درست برسیم.

در بررسی‌های انجام شده تاکنون هیچیک از الگوریتم‌های کلونی زنبور برای مسئلهٔ فروشنده دوره‌گرد و در واقع هیچ مسئلهٔ ترکیبی دیگری بر روی FPGA پیاده نشده است. تنها مرجع نزدیک به این پیاده‌سازی که الگوریتم ABC را برای مسائل پیوسته بر روی FPGA پیاده کرده است، مرجع [۴۱] است. HPOABC روش پیشنهادی برای سخت‌افزار بوده که تفاوت آن با الگوریتم ABC در نحوه جستجوی زنبور پیشانگ است. همچنین برای امتحان روش و پیاده‌سازی انجام شده از چهار تابع معروف معادلات ۱-۵ تا ۴-۵ با شش بعد استفاده شده است:

$$f_1(\vec{x}) = \sum_{i=1}^N x_i^2 \quad \text{معادله ۱-۵ کره}$$

$$f_2(\vec{x}) = \sum_{i=1}^N \left(\sum_{j=1}^i x_j \right)^2$$

معادله ۲-۵ چهارتایی

$$f_3(\vec{x}) = \sum_{i=1}^N (x_i^2 - 10 \cos(2\pi x_i) + 10)$$

معادله ۳-۵ راستریگین

$$f_4(\vec{x}) = \sum_{i=1}^{N/2} 100(x_{2i} - x_{2i-1})^2 + (1 - x_{2i-1})^2$$

معادله ۴-۵ روزن بروک

این پیاده‌سازی همانند روش HBA با استفاده از ۱۰ زنبور انجام شده است. در جدول ۳-۵ حجم سخت‌افزاری استفاده شده از تراشه XC5VLX110T که زیرمجموعه تراشه‌های خانواده ۵ Virtex است نشان داده شده است. در این جدول حجم بالای تعداد جداول جستجو جلب توجه می‌کند.

جدول ۳-۵ فضای مصرفی برای پیاده‌سازی الگوریتم ABC بر روی FPGA

Implemented core	FF	LUTs	DSP48E	Freq. MHz
HPOABC f_1	8734(12.6%)	33558(48.5%)	46(71.9%)	83.40
HPOABC f_2	8384(12.1%)	33212(48.1%)	46(71.9%)	83.51
HPOABC f_3	18863(27.3%)	99263(143%)	6(9.37%)	82.50
HPOABC f_4	8405(12.2%)	33716(48.8%)	26(40.6%)	83.40

۳-۵ جمع‌بندی

با توجه به مطالب گفته شده قبلی، درمی‌یابیم که پیاده‌سازی انجام شده نتایج قابل قبولی برای جستجوی مسیر با کمترین مسافت ارائه می‌دهد که در کنار نتایج بسیار خوب مربوط به حجم منابع سخت‌افزاری استفاده شده و زمان ناچیز برای این جستجو آن را به یک کاندیدای خوب در کاربردهای برخط و در کنار هسته‌های دیگر تبدیل می‌کند.

دیده شد که برای مسائل با تعداد شهرهای کم الگوریتم پیشنهادی با روش 2-opt عملکرد بهتری نسبت به بقیه الگوریتم‌ها نشان داده است. همچنین با توجه به اینکه الگوریتم BA به دلیل داشتن زنبورهای بیشتر حجم سخت‌افزاری بیشتری را اشغال می‌کند و زمان اجرای بیشتری نیز دارد، الگوریتم ارائه شده در بقیه مسائل با شهرهای بیشتر، جز در پاسخ نهایی در بقیه معیارها از الگوریتم BA برتری دارد. مضافاً روش 2-opt به نسبت حجم مصرفی منابع و سرعت عملکرد، نتایج مقبولی را

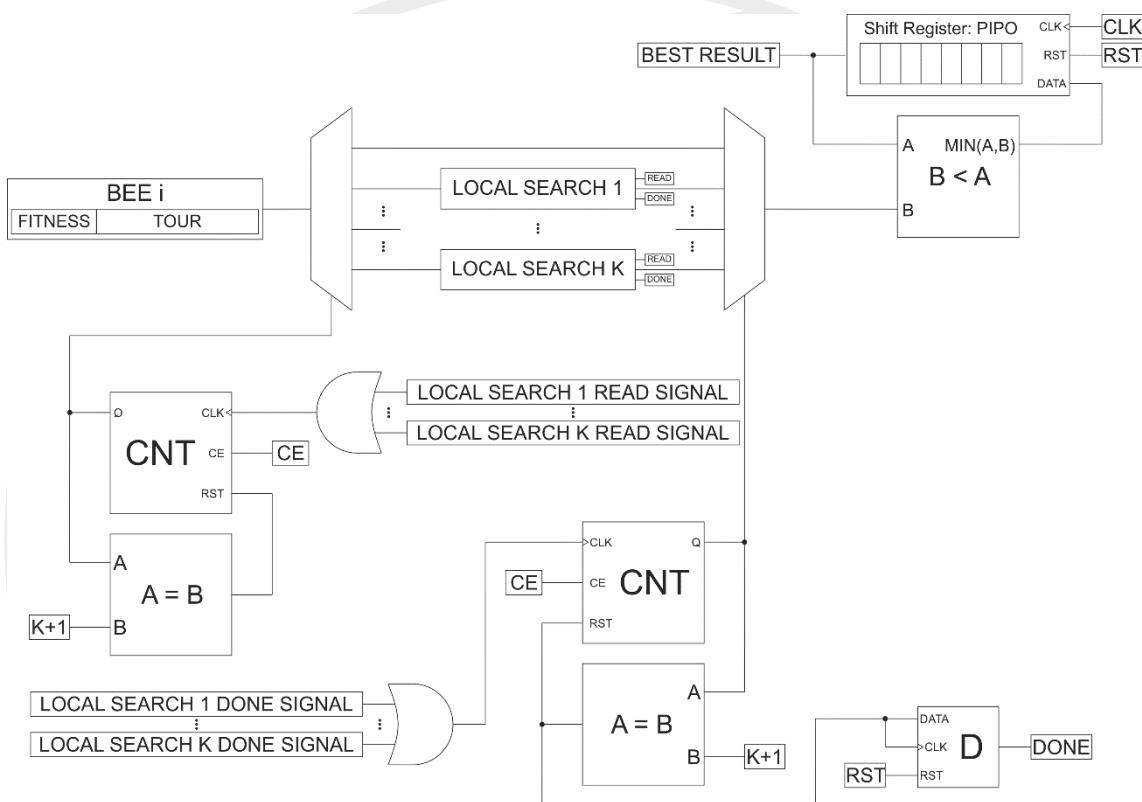
در مقایسه با یک روش پیچیده همانند GSTM ارائه می‌دهد که سبب می‌شود که روش 2-opt برای پیاده‌سازی سخت‌افزاری مناسب باشد.

۴-۵ پیشنهادات برای کارهای آتی

به منظور توسعه و بسط تحقیق انجام شده پیشنهاد می‌شود که:

- به منظور موازی‌سازی بیشتر می‌توان برای هر زنبور بیشتر از یک واحد محاسبات بهینه‌سازی در نظر گرفت. نحوه اتصالات و ساختار کلی این طرح در شکل ۱-۵ آمده است. در این طرح، هر واحد محاسبات محلی به واسطه یک دی‌مالتی‌پلکسر به زنبور و یک مالتی-پلکسر به مقایسه‌گر و ثباتی که نگهدارنده بهترین جواب واحدها در یک تکرار است، متصل شده است. این مالتی‌پلکسر و دی‌مالتی‌پلکسر به وسیله خروجی‌های واحدهای بهینه‌سازی کنترل می‌شوند. سیگنال اول (READ) برای کنترل اتصال یک واحد به زنبور برای دریافت اطلاعات شهرها و سیگنال دوم برای اعلام اتمام عملیات و مقایسه نتایج این واحدها در نظر گرفته شده است (DONE). نحوه عملکرد کلی بدین صورت است که در ابتدا مقادیر ثبات‌های بهترین مسیر و مسافت متناظر آن بازنگشانی می‌شود. سپس واحد محاسبات بهینه‌سازی محلی اول درخواست ۴ شهر را به واحد زنبور می‌دهد. بعد از دریافت این چهار شهر شروع به انجام محاسبات کرده و در این حالت واحد دوم به زنبور متصل شده و اطلاعات ۴ شهر دیگر را دریافت می‌کند. به همین ترتیب تا آخرین واحد محاسبه بهینه‌سازی. چون واحد اول زودتر شروع کرده در نتیجه زودتر هم به نتایج می‌رسد و در زمانی که واحد دوم هنوز در حال محاسبات است، نتایج خود را برای مقایسه به خروجی می‌فرستد. بعد از این نوبت به واحد دوم می‌رسد و به ترتیب تا آخرین واحد نتایج برای مقایسه و ذخیره به خروجی فرستاده می‌شوند. در انتها این ثبات شامل بهترین نتیجه از تمامی جستجوهای محلی خواهد بود. این طرح سرعت را افزایش داده ولی به ازای مصرف بیشتر تعداد بلوک‌های منطقی و رم. در صورت نیاز به بلوک‌های رم در هسته‌های دیگر می‌توان برای محاسبه فواصل مستقیماً از فاصله اقلیدسی استفاده کرد که در نتیجه آن تنها نیاز به ذخیره‌سازی مختصات افقی و عمودی شهرها بر روی

بلوک‌های رم است. ولیکن واحد محاسبه فاصله اقلیدسی منابع ثبات، جداول جستجو و پردازنده‌های سیگنال دیجیتالی تعبیه شده قابل توجهی را اشغال خواهد کرد. همچنین می‌توان از روش سوم که تلفیقی از دو روش قبل استفاده کرد. در روش سوم می‌توان به تعداد دلخواه ماتریس فاصله‌ها را در ابتدای الگوریتم روی حافظه جانبی ذخیره کرد.



شکل ۱-۵ ساختار پیشنهادی

- پیاده‌سازی الگوریتم HBA بر روی پردازنده‌های نرم یا سخت تعبیه شده بر روی FPGA به منظور مقایسه صحیح زمانی و حجم منابع مصرفی بین پیاده‌سازی نرم‌افزاری و سخت‌افزاری.
- پیاده‌سازی واحد تولید کننده تصادفی آدرس دو شهر با استفاده از انواع دیگر تولیدکننده‌های اعداد نیمه تصادفی و مقایسه نتایج با هم.
- پیاده‌سازی روش‌های دیگر بهینه‌سازی محلی از قبیل 3-opt و مقایسه نتایج با روش 2-opt
 - نظر زمانی و منابع مصرفی.

۶. مراجع

۱- مراجع فارسی

(ماکسفیلد، ۱۳۸۹) ماکسفیلد، کلیو، مرجع کامل طراحی با FPGA، ۱۳۸۹، ترجمه فرزاد شکاری زاده، چاپ اول، تهران، انتشارات نص.

(پونگ، ۱۳۹۰) پونگ پی. چو، نمونه‌سازی FPGA با مثال‌هایی از VHDL، ۱۳۹۰، ترجمه دکتر قدرت‌اله سپیدنام، چاپ اول، بابل، انتشارات علوم رایانه.

(بیات، ۱۳۹۳) مریخ بیات، فرشاد، الگوریتم‌های بهینه‌سازی فرآیندکاری، ۱۳۹۳، چاپ اول، تهران، انتشارات جهاد دانشگاهی واحد زنجان.

(مانو، ۱۳۸۱) مانو، موریس، طراحی دیجیتال (مدار منطقی)، ۱۳۸۱، ترجمه دکتر قدرت‌اله سپیدنام، چاپ اول، مشهد، انتشارات خراسان.

(ثابت، ۱۳۹۰) ثابت، شیما سادات، "انتخاب الگوریتم تکاملی بهینه بر مبنای ساختار"، پایان نامه کارشناسی ارشد دانشگاه آزاد اسلامی واحد تهران مرکزی، استاد راهنمای: دکتر فرداد فرخی، استاد مشاور: دکتر کاوه کنگرلو، زمستان ۱۳۹۰.

۲- مراجع انگلیسی

- [1] Colorni A., Dorigo M., and Maniezzo V., "Distributed optimization by ant colonies," in Proc. ECAL91—Eur. Conf. Artificial Life. New York: Elsevier, pp. 134–142, 1991.
- [2] Dorigo M. and Stützle T., "Ant Colony Optimization", MIT Press, Cambridge, MA, 2004.
- [3] Dorigo M., Birattari M. and Stützle T., "Ant Colony Optimization, Artificial Ants as a Computational Intelligence Technique", IEEE Computational Intelligence Magazine, Vol 1, Issue 4, pp. 28-39, 2006.
- [4] Kennedy J., and Eberhart R. C., "Particle swarm optimization", In Proceedings of the IEEE International Conference on Neural Networks, Vol. 4, pp. 1942-1948, 1995.
- [5] Yang X. S., Nature-Inspired Metaheuristic Algorithms: Second Edition, Luniver Press, 2010.
- [6] Karaboga. D, "An Idea Based on Honey Bee Swarm for Numerical Optimization", Technical Report- TR06, Erciyes University, Computer Engineering Department, 2005.
- [7] Karaboga. D and Basturk. B, "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm", Journal of Global Optimization 39, pp. 459–471, 2007.
- [8] Krishnanand K.N. and D. Ghose "Detection of multiple source locations using a glowworm metaphor with applications to collective robotics". IEEE Swarm Intelligence Symposium, Pasadena, California, USA, pp. 84–91, 2005.
- [9] Krishnanand, K.N.; Ghose, D. "Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions". Swarm Intelligence, Vol 3, Issue 2: pp. 87–124, 2009.
- [10] D. Dasgupta (Editor), "Artificial Immune Systems and Their Applications", Springer-Verlag, Inc. Berlin, 1999.
- [11] Cutello V., Nicosia G., Pavone M., Timmis J., "An Immune Algorithm for Protein Structure Prediction on Lattice Models", IEEE Transactions on Evolutionary Computation, Vol. 11, Issue. 1, pp. 101–117, 2007.
- [12] Merkle D., Middendorf M., "Fast ant colony optimization on runtime reconfigurable processor arrays", Genetic Programming and Evolvable Machines Vol 3, Issue 4, pp. 345–361, 2002
- [13] Diessel O., ElGindy H., Middendorf M., Guntsch M., Scheuermann B., Schmeck H., So K., "Population based ant colony optimization on FPGA", In IEEE International Conference on Field-Programmable Technology (FPT), pp. 125–132, 2002.

- [14] Scheuermann B., So K., Guntsch M., Middendorf M., Diessel O., ElGindy H., Schmeck H., "FPGA implementation of population-based ant colony optimization", *Applied Soft Computing* 4, pp. 303–322, 2004.
- [15] Scheuermann B., Guntsch M., Middendorf M., Schmeck H., "Time-Scattered Heuristic for the Hardware Implementation of Population-Based ACO", *Ant Colony Optimization and Swarm Intelligence Lecture Notes in Computer Science*, Vol 3172, pp. 250-261, 2004.
- [16] Yoshikawa, M. and Terai, H., "Architecture for high-speed ant colony optimization", *Proceedings of the IEEE International Conference on Information Reuse and Integration*, Las Vegas, NV, pp. 1–5, 2007.
- [17] Yoshikawa, M., "Hardware-oriented ant colony optimization considering intensification and diversification", *Advances in greedy algorithms*, I-Tech, pp. 359–68, 2008.
- [18] Duan H., Yu Y., Zou J., Feng X., "Ant colony optimization-based bio-inspired hardware: survey and prospect", *Transactions of the Institute of Measurement and Control* 34, pp. 318–333, 2012.
- [19] Reynolds P.D., Duren R.W., Trumbo M.L., Marks II R.J., "FPGA implementation of particle swarm optimization for inversion of large neural networks", *IEEE Swarm Intelligence Symposium*, pp. 389-392, 2005.
- [20] Palangpour P., Mitra, P., Ray, S., Venayagamoorthy G.K., "DSP-Based PSO Implementation for Online Optimization of Power System Stabilizers", *NASA/ESA Conference on Adaptive Hardware and Systems*, 2008.
- [21] Chai Z., Univ J., Sun J., Cai R., Xu W., "Implementing Quantum-Behaved Particle Swarm Optimization Algorithm in FPGA for Embedded Real-Time Applications", *Fourth International Conference on Computer Sciences and Convergence Information Technology*, pp. 24-26, 2009.
- [22] Thweny F.A., Alhammad A.A., "Implementation of FPGA Based PSO PID Controller for Feedback IVAX SCARA Robot Manipulator", *IJCSET*, Vol 3, Issue 10, pp. 367-372, 2013.
- [23] Bitam S., Batouche M., Talbi E., "A survey on Bee Colony Algorithms", *IEEE International Symposium on Parallel and Distributed Processing*, pp. 1-8, 2010.
- [24] Pham, D.T., Castellani, M., "The bee algorithm: modelling foraging behaviour to solve continuous optimization problems". *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, Vol 223 Issue 12, pp. 2919-2938, 2009.
- [25] Pham D.T., Otri S., Haj Darwish A., "Application of the Bees Algorithm to PCB assembly optimization", In *Proceedings of the Third International Virtual Conference on Intelligent production machines and systems (IPROMS)*, Whittles, Dunbeath, Scotland, pp. 511–516, 2007.
- [26] Pham D. T., Castellani M., Fahmy A., "A Learning the inverse kinematics of a robot manipulator using the Bees Algorithm", In *Proceedings of the INDIN*, pp. 493–498, 2008.,
- [27] Karaboga D., Basturk B., "On the performance of artificial bee colony (ABC) algorithm", *Applied soft computing*, Vol 8, Isuue 1, pp.687-697, 2008.
- [28] Akay B., Karaboga D., "A modified artificial bee colony algorithm for real-parameter optimization", *Information Sciences*, 192, pp. 120-142, 2012.
- [29] Croes G.A., "A method for solving traveling salesman problems", *Operations Research.*, vol. 6, pp. 791-812, 1958.
- [30] Banharnsakun A., Achalakul T., Sirinaovakul T.B., "ABC-gsx a hybrid method for solving the traveling salesman problem", *Nature and Biologically Inspired Computing*, pp. 7-12, 2010.
- [31] Louis S. J., Tang R. "Interactive genetic algorithms for the traveling salesman problem". In *Proceedings of the genetic and evolutionary computing conference (GECCO)*, pp. 385–392, 1999.
- [32] Albayrak M., Allahverdi N., "Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms", *Expert Systems with Applications*, vol. 38, no. 3, pp. 1313-1320, 2011.
- [33] Karaboga D., Gorkemli B., "A combinatorial artificial bee colony algorithm for traveling salesman problem", *International Symposium on Innovations in Intelligent Systems and Applications (INISTA)*, pp. 50-53, 2011.
- [34] Laporte G., "The traveling salesman problem: An overview of exact and approximate algorithms", *European Journal of Operational Research*, vol. 59, no. 2, pp. 231-247, 1992.
- [35] Wong L.P., Hean Low M.Y., Chong C.S., "A Bee Colony Optimization Algorithm for Traveling Salesman Problem", *Second Asia International Conference on Modelling & Simulation*, pp. 818-823, 2008.
- [36] Davidovic T., Ramljak D., Selmic M., Teodorovic D., "Bee colony optimization for the pp-center problem". *Computers & Operations Research*. Volume 38, Issue 10, pp. 1367–1376 2011.

- [37] Maruyama T., Funatsu T., Seki M., Yamaguchi Y., Hoshino T., "A Field-Programmable Gate-Array system for Evolutionary Computation", Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications, From FPGAs to Computing Paradigm, pp. 356-365, 1998.
- [38] Martin P., "An analysis of random number generators for a hardware implementation of genetic programming using FPGAs and Handel-C", in Proceedings of the Genetic and Evolutionary Computation Conference, pp. 837-844. 2002.
- [39] Xilinx Co., "Linear Feedback Shift Registers in Virtex Devices", 2007.
- [40] Kumar J., Shukla S., Prakash D., Mishra P., Kumar S., "Random Number Generator Using Various Techniques through VHDL", International Journal of Computer Applications in Engineering Sciences, pp. 127-129, 2011.
- [41] Munoz D.M., Llanos C.H., Coelho L.D.S., Ayala-Rincon M., "Accelerating the artificial bee colony algorithm by hardware parallel implementations", *Circuits and Systems 2012 IEEE Third Latin American Symposium on Circuits and Systems* (LASCAS), pp. 1-4, 2012.
- [42] Juang C.F., Lu C.M., Lo C., Wang C.Y., "Ant colony optimization algorithm for fuzzy controller design and its fpga implementation", *IEEE Transactions on Industrial Electronics*, vol. 55, no. 3, pp. 1453-1462, 2008.
- [43] Munoz D.M., Llanos C., Coelho L., Ayala-Rincon M., "Hardware particle swarm optimization with passive congregation for embedded applications", *IEEE VII Southern Conference on Programmable Logic* (SPL), pp. 173-178, 2011.
- [44] Munoz D.M., Llanos C., Coelho L., Ayala-Rincon M., "Comparison between two FPGA implementation of the particle swarm optimization algorithm for high-performance embedded applications", *The IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications* (BIC-TA 2010), vol. 1, pp. 1637-1645, 2010.
- [45] Munoz D.M., Llanos C., Coelho L.D.S., Ayala-Rincon M., "Hardware opposition-based PSO applied to mobile robot controllers", *Engineering Applications of Artificial Intelligence*, pp.64-77, 2014
- [46] Rathod A., Thakker R.A., "FPGA Realization of Particle Swarm Optimization Algorithm using Floating Point Arithmetic", *IEEE International Conference on High Performance Computing and Applications* (ICHPCA), pp. 1-6, 2014.
- [47] Mehmood S., Cagnoni S., Mordonini M., "Hardware oriented adaptation of a particle swarm optimization algorithm for object detection", *IEEE 11th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, pp. 904-911, 2008.
- [48] Munoz D.M., Llanos C.H., Coelho L.D.S., Ayala-Rincon M., "Hardware architecture for particle swarm optimization using floating-point arithmetic", *IEEE Ninth International Conference on Intelligent Systems Design and Applications*, pp. 243-250, 2009.
- [49] Yamada T., Maeda Y., Miyoshi. S., Hikawa H., "Simultaneous Perturbation Particle Swarm Optimization and FPGA Realization", *Proceedings of SICE Annual Conference*, pp. 1462-1465, 2010.
- [50] Li S.A., Hsu J., Wong C.C., Yu C.J., "Hardware/software co-design for particle swarm optimization algorithm. ELSEVIER, Special Issue on Interpretable Fuzzy Systems. Volume 181, Issue 20, pp. 4582-4596, 2011.
- [51] Calazan R.M., Nedjah N., Mourelle L.M., "A hardware accelerator for Particle Swarm Optimization", *Journal of Applied Soft Computing*, Volume 14, Part C, pp. 347-356, 2014.
- [52] <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/>
- [53] www.xilinx.com/support/documentation/data_sheets/ds100.pdf

Abstract

Swarm intelligence optimization algorithms are kinds of metaheuristic algorithms that have been used to solve well-known problems. Without any doubt, the most famous problem among them is the Traveling Salesman Problem (TSP). One category of these algorithms is honey bee colony algorithms which have been inspired by the natural behavior of honey bees foraging. The most popular types of these algorithms are Bee Algorithm (BA), Artificial Bee Colony (ABC), and Bee Colony Optimization (BCO). This research has investigated the possibility of implementing the bee colony algorithms on the Field Programmable Gate Array (FPGA). For this purpose, a Hardware Bee Algorithm (HBA) approach has been proposed and implemented on the software and hardware basis for solving the traveling salesman problem. The structure of the proposed algorithm is suitable for implementation on the FPGA. In addition, the hardware has been designed with heuristic methods on the gate, behavioral, and structural levels to minimize the hardware resource occupation and maximize the speed. Moreover, to compare the performance and execution time of the proposed algorithm, BA, ABC, and BCO algorithms have been implemented on the software, and their results have been compared with HBA's software and hardware implementations. In addition to good software results of the HBA algorithm, hardware implementation has shown that it is possible to obtain acceptable results with the minimum usage of hardware resources and a noticeable reduction in execution time.

Keywords: Honey Bee Colony Algorithms, Hardware Bee Algorithm (HBA), Field Programmable Gate Array (FPGA), Traveling Salesman Problem (TSP), Implementation



ISLAMIC AZAD UNIVERSITY

Central Tehran Branch

Faculty of Technical and Engineering

Department of Electrical Engineering

(M.Sc) Thesis

On Electronic

Subject:

**Implementation of Hardware Bee Algorithm (HBA) on FPGA
for Travelling Salesman Problem**

Advisor:

Dr. Fardad Farokhi

Reader:

Dr. Reza Sabbaghi Nadoushan

By:

Pourya Khodagholtipour

September 2016