# پیاده‌سازی واحد IP-Lookup بهینه‌سازی شده برای مسیریاب‌های Access

*حدیث محسنی تکلو، *مرجان نادران طحان و **آرش طبیبی‌آذر

*دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف
**مرکز تحقیقات مخابرات ایران
mohseni@ce.sharif.edu, naderan@ce.sharif.edu, tabibi@itrc.ac.ir

## چکیده

یکی از موارد مهم طراحی در مسیریاب‌های IP نسل جدید، مکانیزم جستجوی مسیر است. به ازای هر بسته، بخش جستجوی مسیر، باید عمل تطبیق IP مقصد، با طولانی‌ترین پیشوند را انجام دهد تا پورت خروجی بسته مشخص شود. در این مقاله، یک روش جستجوی مسیر سریع ارائه شده است، که نیاز به یک حافظه کوچک دارد و می‌تواند به آسانی در سخت‌افزار پیاده‌سازی شود. در این روش اکثر جستجوها به یک دسترسی به حافظه نیاز دارند و هیچ جستجویی بیش از ۳ بار دسترسی نیاز ندارد، در صورت استفاده از روش خط لوله، همه جستجوها با یک‌بار دسترسی به حافظه انجام می‌شوند. این روش اگر در حافظه‌های امروزی با تأخیر ۱۰ نانوثانیه پیاده‌سازی شود، می‌تواند ۱۰۰ میلیون جستجوی مسیر در ثانیه انجام دهد،که بسیار سریعتر از روش‌های رایج پیاده‌سازی شده است. این مقاله همچنین به پیاده سازی توام سخت‌افزاری و نرم‌افزاری انجام شده این روش می پردازد که برای مسیریاب‌های Access بهینه و برروی FPGA پیاده‌سازی گردیده است که حجم حافظه و اندازه گیت مصرفی در آن پایین می باشد. این روش در مقایسه با روشهای مشابه در این سطح بسیار بهینه بوده و با سرعت پالس ساعت پایین، بیش از ۱۰ میلیون جستجو در ثانیه انجام می دهد.

## واژه‌های کلیدی

مسیریابی، مسیریاب IP، تطبیق، مسیرده، آرایه های برنامه پذیر

## ۱- مقدمه

ظهور وب جهانی (WWW) ترافیک شبکه اینترنت را هر چند ماه یک‌بار، دو برابر می‌کند و کاربردهایی که پهنای باند زیادی اشغال می‌کنند مانند آموزش از راه دور و کتابخانه‌های دیجیتالی، ترافیک بیشتری ایجاد می‌کنند. این موضوع طراحی مسیریاب‌هایی

با کیفیت سرویس (Quality of Service) بالا را با بهبود سه چالش سرعت اتصال‌ها، توان عملیاتی (Throughput) و سرعت ارسال بسته‌ها روبرو ساخته است. هم‌اکنون راه‌حل‌هایی برای دو مورد اول وجود دارد، این مقاله با راه‌حل‌های مشکل سوم سروکار دارد و روشی برای بالابردن سرعت ارسال بسته‌ها دربرابر با افزایش سرعت اتصال‌ها، ارائه می‌دهد. هرچند در سال‌های اخیر جستجوی MAC آدرس به سرعت‌های ۱۰۰ مگابیت در ثانیه رسیده است، اما این کار بر اساس تطبیق MAC آدرس ورودی با یکی از مدخل‌های جدول مسیریابی انجام می‌شود در حالی‌که مسیریاب‌های اینترنت برمبنای تطبیق طولانی‌ترین پیشوند بین آدرس مقصد و IP های جدول مسیریابی عمل می‌کنند[۱][۲]. بنابراین روش‌های تطبیق کامل و استفاده از CAM را در مورد آنها نمی‌توان بکار برد[۳]. بر این اساس، روش تطبیق پیشوند در سال ۱۹۹۰ پیشنهاد شد تا تعداد مدخل‌های جدول مسیریابی کمتر شود. در این روش فضای آدرس IP به کلاس‌های A، B و C تقسیم می‌شود که هرکدام به ترتیب ۲۴، ۱۶و ۸ بیت برای آدرس‌دهی دارند. البته این روش تقسیم‌بندی شبکه‌ها انعطاف‌پذیری ندارد و فضای آدرس‌دهی را هدر می‌دهد، بخصوص در مورد کلاس B. برای حل این مشکل روش CIDR (Classless Inter Domain Routing) پیشنهاد شد که تراکم تعداد دلخواهی شبکه را مجاز می‌شمارد و باعث کاهش تعداد مدخل‌های جدول مسیریابی می‌شود[۴]. روش CIDR هر مسیر بوسیله زوج <طول پیشوند، IP> مشخص می‌شود، که طول پیشوند بین ۰ تا ۳۲ بیت است. هنگامی که یک بسته IP دریافت می‌شود، مسیریاب بررسی می‌کند تا ببیند کدام‌یک از پیشوندها در جدولش طولانی‌ترین تطبیق را در مقایسه با IP مقصد در بسته ورودی دارد، سپس بسته از طریق پورت خروجی مربوط به این پیشوند ارسال می‌شود. در ادامه مقاله ابتدا مروری مختصر بر روش‌های موجود جستجوی IP کرده، بعد طرح کلی این روش موردنظر بیان می‌شود. سپس طریقه پیاده‌سازی آن ذکر خواهد شد و در آخر کارآیی روش مورد بررسی قرار می‌گیرد.

تا به حال چندین روش جستجوی مسیر با سرعت زیاد ارائه شده است [۵][۶][۷][۸][۹][۱۰][۱۱][۱۲]، مثلاً در روشی که توسط دگرمارک و همکاران[۵] ارائه شد، ساختار جدول‌های جستجو بسیار کوچک است تا جایی که یک جدول با ۴۰۰۰۰ مدخل می‌تواند تا اندازه ۱۵۰ تا ۱۶۰ کیلوبایت فشرده شود. البته این یک روش نرم‌افزاری است، اگر توسط سخت‌افزار پیاده‌سازی شود بین ۲ تا ۹ دسترسی به حافظه به ازای هر جستجو لازم است. گوپتا و همکاران [۷] نیز یک روش سریع ارائه دادند که از DRAM استفاده می‌کند و حداکثر نیاز به دو دسترسی به حافظه در یک جدول جستجوی ۳۳ مگابایتی دارد. با اضافه کردن یک جدول میانی اندازه جدول جستجو به ۹ مگابایت کاهش می‌یابد ولی تعداد دسترسی‌ها به ۳ تا افزایش پیدا می‌کند، اگر این روش در سخت‌افزار و با استفاده از Pipeline پیاده‌سازی شود، دسترسی‌ها به یکی کاهش می‌یابد. والدوگل و همکاران [۱۲] نیز روشی برای جستجوی سریع برمبنای جستجوی دودویی ارائه دادند که برای جدول‌های آدرس و مسیریابی بزرگ مقیاس‌پذیری خیلی خوبی دارد و به تعداد لگاریتم تعداد بیت‌های آدرس، جستجوی Hash لازم دارد. برای مثال در پایگاه‌داده‌ای با N تا پیشوند با طول آدرس W، روش جستجوی دودویی معمولی از مرتبه W*log(N) است، ولی این روش از مرتبه W+log(N) است. البته این روش هم نرم‌افزاری است و در پیاده‌سازی در سخت‌افزار جواب خوبی نمی‌دهد. روش‌های متعدد دیگری هم در [۱۳] ذکر گردیده است.

## ۲- طرح کلی روش جستجوی IP

ساده‌ترین روش در جستجوی IP داشتن یک جدول با $2^{32}$ مدخل است که هر مدخل آن نشان دهنده یکی از IP های ۳۲ بیتی باشد. در این صورت برای هر جستجوی IP یک دسترسی به حافظه نیاز است، ولی عیب این روش در اندازه بسیار بزرگ جدول جستجو است [۷].

برای کاهش اندازه جدول، روش جستجوی غیرمستقیم را بکار می‌بریم [۷]. هر IP ۳۲ بیتی به دو قسمت ۱۶ بیتی قطعه و آفست تقسیم می‌شود. جدول قطعه ۶۴ کیلو مدخل دارد ($۲^{۱۶}$) که هر مدخل یا شماره پورت خروجی را نگه می‌دارد (که در این صورت حاوی عددی کمتر از ۲۵۶ است، البته با این فرض که شماره پورت‌های خروجی بین ۰ و ۲۵۶ است) و یا یک اشاره‌گر است که به NHA(Next Hop Array) مربوط به این خانه اشاره می‌کند. هر NHA جدولی با ۶۴ کیلو ($۲^{۱۶}$) مدخل است که هر مدخل آن ۸ بیت است و شماره پورت خروجی مربوط به یک IP را نگه می‌دارد. بنابراین به ازای یک آدرس IP که به صورت a.b.x.y است، a.b به عنوان اندیس جدول قطعه استفاده می‌شود و x.y به عنوان اندیس NHA مربوطه (البته در صورتی که نیاز باشد). بنابراین به ازای قطعه a.b، اگر طول طولانی‌ترین پیشوند متعلق به این قطعه کمتر یا مساوی ۱۶ باشد، مدخل مربوطه در جدول، پورت خروجی را نگه می‌دارد و نیازی به نگهداری NHA نیست، در غیراین‌صورت، ۶۴ کیلو مدخل در NHA لازم است. در نتیجه روش جستجوی غیرمستقیم برای هر جستجوی IP حداکثر دو بار دسترسی به حافظه نیاز دارد.

هرچند که روش جستجوی غیرمستقیم سریع است ولی چگونگی توزیع پیشوندهای یک قطعه را درنظر نمی‌گیرد، یعنی اگر طول طولانی‌ترین پیشوندهای تطبیق شده با یک قطعه بیشتر از ۱۶ باشد، ناچار به نگهداری یک جدول ۶۴ کیلویی هستیم. می‌توان با توجه به توزیع پیشوندهای یک قطعه، اندازه جدول را کاهش داد. در این روش جدول ۶۴ کیلویی قطعه را داریم ولی هر مدخل این جدول به دو فیلد تقسیم می‌شود : فیلد اول یا یک اشاره‌گر است (اگر مقدارش بیشتر از ۲۵۶ باشد) یا پورت خروجی را نگه می‌دارد (اگر مقدارش کمتر از ۲۵۶ باشد) و فیلد دوم طول آفست را معلوم می‌کند (K) و عددی بین ۰ تا ۱۵ است. به ازای یک آفست K بیتی NHA مربوطه $۲^K$ مدخل خواهد داشت که در طرح غیرمستقیم قبلی K همواره ۱۶ بود(شکل ۱). طول آفست به طول پیشوندهای آن قطعه بستگی دارد، مثلاً اگر قطعه a.b، M تا پیشوند داشته باشد که طول طولانی‌ترین آنها L باشد (۱۶ < L <= ۳۲) ، طول آفست برای این قطعه برابر L−۱۶ است، می‌توان به این صورت تعبیر کرد که برای IP به صورت a.b.x.y قسمت a.b به عنوان اندیس جدول قطعه بکار می‌رود و K بیت سمت چپ x.y (یعنی از بیت ۱۶ام تا بیت K – L +۱۶ام) NHA مربوطه را مشخص می‌کنند. اندازه NHA به طول و مجموعه پیشوندها بستگی دارد. فرض کنید که $L_i$ و $H_i$ به ترتیب نشان‌دهنده طول و پورت خروجی پیشوند $P_i$ باشند، در این‌صورت اگر P مجموعه مرتب شده پیشوندهای یک قطعه با فرض اینکه در آن $P=\{P_0, P_1, \ldots, P_{m-1}\}$, $K = \max\{o_i \mid \text{belongs to set P}, o_i=L_i-16\}$ باشد، اندازه NHA ، $۲^K$ می‌شود. همچنین به ازای هر پیشوند $P_i$، داده ساختاری برای نگه‌داری نقطه ابتدا و نقطه انتهای $P_i$ به‌نام‌های $S_i^0$ و $E_i^0$ وجود دارد (IP های این محدوده به پورت خروجی $H_i$ فرستاده می‌شوند) به‌طوری‌که $ma(S_i^0)$ و $ma(E_i^0)$ به‌ترتیب آدرس‌های حافظه $S_i^0$ و $E_i^0$ در NHA اند و $op(S_i^0)$ و $op(E_i^0)$ پورت‌های خروجی آدرس‌های ابتدا و انتها هستند. فرض کنید Pi = a.b.x.y و فرم باینری x.y به‌صورت $x_0, x_1, \ldots, x_{15}$ و $s_0, s_1, \ldots, s_{k-1}$ پوشش‌های نقطه ابتدا باشد، (به‌طوری که $s_j=1$ اگر $j<o_i$ و $s_j=0$ اگر $j>=o_i$) و $e_0, e_1, \ldots, e_{k-1}$ پوشش نقطه انتها باشد (به‌طوری که $e_j=0$ اگر $j<o_i$ و $e_j=1$ اگر $j>=o_i$). در این‌صورت $ma(S_i^0) = (x_0, x_1, \ldots, x_{k-1}$ AND $s_0, s_1, \ldots, s_{k-1})$ و $ma(E_i^0) = (x_0, x_1, \ldots, x_{k-1}$ OR $e_0, e_1, \ldots, e_{k-1})$ و همچنین $NHA_j = H_i$ خواهد بود. به‌ازای هر $P_i$ در P یک $S_i^0$ و $E_i^0$ به دست می آید که بازه $[ma(S_i^0), ma(E_i^0)]$ شامل همه آدرس‌های بین $ma(S_i^0)$ و $ma(E_i^0)$ است. پورت خروجی تمام آدرسهایی که در این بازه قرار می‌گیرند، برابر پورت خروجی $P_i$ است(یعنی $H_i$). البته این در صورتی درست است که بازه‌های $P_i$ های متفاوت، اشتراک نداشته باشند. در اکثر مواقع این درست نیست و بازه‌ها با هم اشتراک دارند، اشتراک بازه‌ها نشان‌دهنده این است که برای یک IP مقصد بیش از یک مدخل تطبیق وجود دارد که البته این روش باید طولانی‌ترین آدرس تطبیق شده را پیدا می‌کند. بنابراین اگر آدرسی مثل j به‌طور همزمان به یک مجموعه از بازه‌ها تعلق داشته باشد $NHA_j = H_i$ و $P_i$

طولانی‌ترین پیشوند از آن مجموعه است. این یک الگوریتم کلی، برای ساختن NHA و جستجوی IP در آن است، ولی ساختار NHA را می‌توان بهینه کرد، بهینه سازی این الگوریتم به این صورت است که بجای NHA یک CWA (Code Word Array) و یک CHNA (Compressed NHA) نگهداری شود. برای ساخت CWA از تکنیک CBM (Compressed Bit Map) استفاده می‌کنیم که در آن به ازای هر مدخل در NHA، یک بیت در نظر گرفته می‌شود. فرض کنید $a_i$ مقدار i امین مدخل در NHA (یعنی پورت خروجی) و $b_i$ مقدار متناظر آن در آرایه CBM و $c_j$ پورت خروجی متناظر در آرایه CNHA باشد، در ابتدا با $c_0 = a_0$ و $b_0 = 1$ و $j = 1$، NHA را به‌ترتیب از چپ به راست می‌پیماییم و اگر $a_{i+1} = a_i$ آنگاه $b_{i+1} = 0$ در غیراین‌صورت $b_{i+1} = 1$ و $c_j = a_{i+1}$ و $j = j+1$. به این ترتیب اندازه CWA که شامل CBM و CNHA است، بسیار کمتر از اندازه یک NHA می‌شود، با استفاده از الگوریتم زیر، می‌توان بدون ساخت NHA اولیه، آرایه‌های CBM و CNHA را ساخت، البته باید توجه کرد که برای هر دو پیشوندی که در یک قطعه هستند یا یکی کاملاً درون دیگری قرار می‌گیرد و یا اینکه هیچ اشتراکی با هم ندارند [۷].

## ۲-۱- الگوریتم ساخت CBM/CNHA

ورودی این الگوریتم بهینه‌سازی، مجموعه $P = \{P_0, P_1, \ldots, P_{m-1}\}$ (پیشوندهای مرتب شده مربوط به یک قطعه) و $L = \{S_0^0, E_0^0, S_1^0, E_1^0, \ldots, S_{m-1}^0, E_{m-1}^0\}$ و خروجی آن CBM و CNHA می باشد. گامهای ساخت آن به صورت زیر است.

- L را برحسب آدرس‌های حافظه مرتب کنید، به‌ازای آدرس‌های یکسان ترتیب اولیه‌شان حفظ می‌شود.
- مجموعه A و پشته C را برابر تهی قرار دهید.
- عناصر A را به‌ترتیب از چپ به راست پیمایش می‌کنیم و به‌ازای هر عنصر داریم :
- اگر عنصر از نوع $S_i^0$ باشد در این صورت $S_i^0$ را در پشته C، وارد کنید و آن را به A اضافه کنید، در غیر این‌صورت عنصر از نوع $E_i^0$ است.
- عنصر بالای پشته را حذف کنید:

۱. اگر عنصر بالای پشته در این لحظه از نوع $S_j^k$ است، در این صورت $S_j^{k+1}$ را به A اضافه کنید، به طوری که $op(S_j^{k+1}) = op(S_j^k)$ و $ma(s_j^{k+1}) = ma(E_i^0)+1$ و عنصر بالای پشته را با $S_j^{k+1}$ عوض کنید.

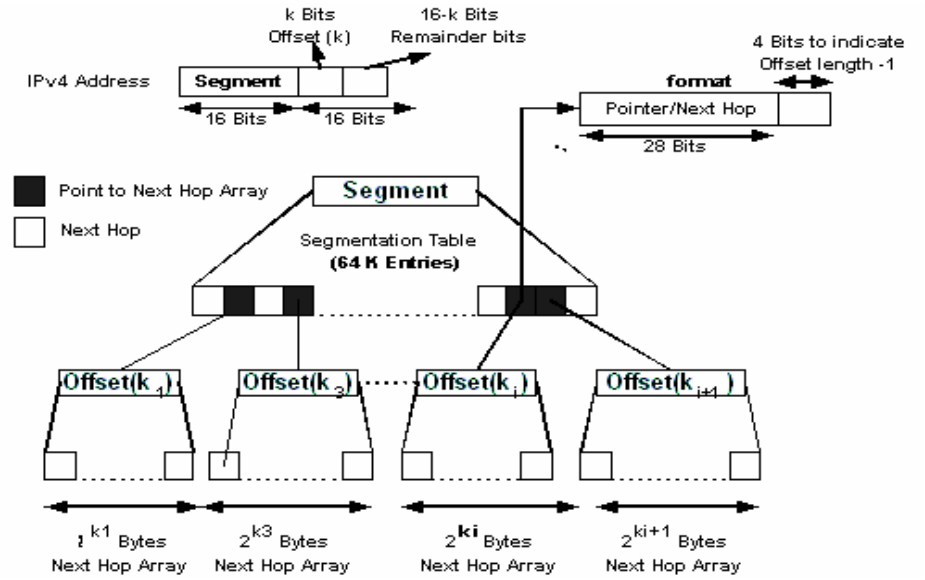
۲. در غیر این‌صورت پشته C خالی است.

- مجموعه A را به‌صورت زیر فشرده کنید، به‌ازای هر دو عضو متوالی $S_j^k$ و $S_p^q$ در A:

۱. عضو $S_j^k$ را از A حذف کنید اگر $ma(S_j^k) = ma(S_p^q)$.

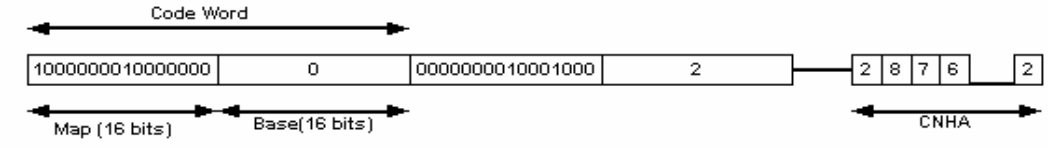
۲. عضو $S_p^q$ را از A حذف کنید اگر $op(S_j^k) = op(S_p^q)$.

- هر عضو A به‌صورت $S_j^k$ را حذف کنید در صورتی که $ma(S_j^k) >= ma(E_0^0)$ باشد.
- به ازای $i = 0$ تا $i = |A|-1$ که $|A|$ اندازه مجموعه A است، اگر $S_j^k$، iامین عنصر A باشد، $CBM_p = 1$ است به طوری‌که $p = ma(S_j^k)$ و $CNHA_i = op(S_j^k)$.




**شکل۱- جدول جستجو در روش غیرمستقیم** [۱]

در نهایت می توان آرایه CBM را به صورت آرایه ای از کلمات کد (CWA) درآورد، هر کلمه کد از دو قسمت دو بایتی map و base تشکیل شده است. CBM را به دسته‌های ۱۶ بیتی تقسیم کرده، هر دسته را در قسمت map یکی از کلمه‌های کد قرار می‌دهیم، base هر کدام از این کلمه کدها هم نشان دهنده تعداد یک‌هایی است که در map کلمه کدهای قبلی وجود دارد، مثالی از این مورد در شکل ۲ دیده می‌شود.



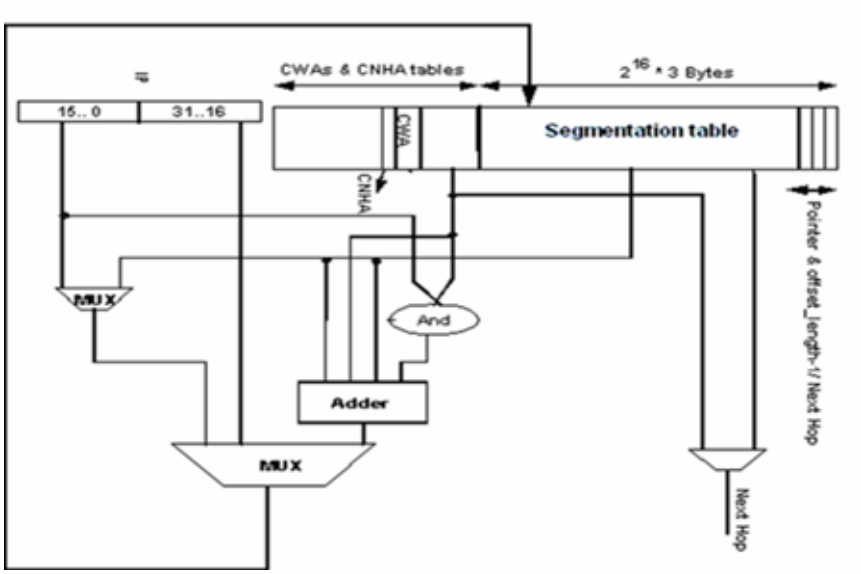

**شکل۲- ساختار آرایه کلمه‌های کد** [۱]

جستجو با این ساختار داده به این‌صورت است که به ازای مقدار آفست q، $cw_s = map_s + base_s$ است، که در آن $s = q \text{ div } 16$ می‌باشد. همچنین $w = q \bmod 16$ بیت متناظر q در $map_s$ است و |w| تعداد یک‌هایی است که از ابتدا تا wامین بیت $map_s$ وجود دارد، در این‌صورت پورت خروجی آفست q برابر است با $op_q = CNHA_t$ به‌طوری‌که $t = base_s+|w|-1$ باشد. با توجه به اینکه ساخت این جدول بسیار سریع است برای به روزکردن آن، می‌توان آن را دوباره ساخت.

## ۳-پیاده‌سازی

در پیاده‌سازی سیستم و همچنین برای تست بلوک سخت‌افزاری نیاز به یک مدل نرم‌افزاری مرجع (Golden Model) هست که پیاده سازی گردید. برای افزایش سرعت اجرا تایپ‌های vector و stack و توابع sort از STL موجود در C++ مورد استفاده قرار گرفته است. توابع اصلی پیاده سازی نرم‌افزاری در زیر آورده شده است. در تست سخت‌افزار پیاده‌سازی شده نیز، از این توابع بهره جسته شده است که در نهایت بصورت PLI در مدلسازی سخت‌افزاری می‌توانند مورد استفاده قرار گیرند.

- **CalculateS** و **CalculateE:** توابعی که $S_i^0$ و $E_i^0$ را برای هر IP ورودی محاسبه می‌کنند.
- **Insert:** این تابع مجموعه L را می‌سازد.
- **Process**: این تابع مجموعه L را به عنوان ورودی می‌گیرد و پشته C و مجموعه A را می سازد (گام سوم).
- **Compact:** این تابع مجموعه A را به عنوان ورودی گرفته آن را بهینه می‌کند (گام چهارم و پنجم).
- **CWAMaker:** این تابع مجموعه A و مقدار K را به عنوان ورودی گرفته و جدولی شامل دو فیلد CWA و CNHA را برمی‌گرداند (گام ششم).
- **TableMaker:** این تابع یک اشاره‌گر که شامل یک گروه از IP (با دو بایت اول یکسان) و تعداد این IP ها را به عنوان ورودی گرفته و خانه مربوط به این گروه از IP ها را در جدول قطعه درست می‌کند (گام‌های ۲ تا ۶).
- **ArrayMaker:** این تابع توسط برنامه اصلی صدا زده می‌شود و جدول قطعه را می‌سازد.
- **MemoryMaker:** این تابع جدول قطعه ساخته‌شده توسط این الگوریتم به یک حافظه خارجی نگاشت می کند. در ابتدای این حافظه آرایه ثابت ۶۵۵۳۶ تایی جدول قطعه قرار می‌گیرد که هر خانه آن ۳ بایت است. اگر بزرگ‌ترین پوشش موجود در بین IP های مربوط به این خانه بیشتر از ۱۶ باشد، ۴ بیت بالا صرف نگه‌داشتن مقدار "۱-(بزرگترین پوشش−۳۲)" می‌شود که عددی بین ۰ تا ۱۵ است. ۲۰ بیت باقی‌مانده هم حاوی آدرس اولین خانه CWA مربوط به این خانه جدودل قطعه در حافظه است. در غیراین‌صورت مستقیماً شماره پورت خروجی در خانه آرایه نگهداری می‌شود. قسمت دوم حافظه شامل CWA و CNHA های خانه‌هایی از جدول قطعه است که به دلیل داشتن پوشش بزرگتر از ۱۶ دارای جدول آفست هستند. در اینجا ترتیب ذخیره‌سازی این آرایه‌ها به ترتیب شماره IP ها در جدول قطعه است.
- **Search:** این تابع یک IP و اشاره‌گری به ابتدای حافظه را به عنوان ورودی می‌گیرد و بزرگترین پیشوند موجود که بر این IP منطبق است، پیدا کرده و شماره پورت خروجی را برمی‌گرداند.




**شکل۳- شمای کلی پیاده‌سازی سخت‌افزاری**

### ۳-۲-پیاده‌سازی سخت افزاری

در پیاده‌سازی سخت‌افزاری برای همگام‌سازی کل سیستم با پالس ساعت از یک ماشین حالت برای انجام مراحل مختلف جستجو استفاده می‌شود. اساس کار جستجو در سخت‌افزار بسیار شبیه به نرم‌افزار است، با این تفاوت که در سخت‌افزار دسترسی به حافظه از نوع دسترسی سخت‌افزاری و در گام‌های پالس ساعت است. برای پیاده‌سازی موتور جستجو (Search-Engine) از زبان توصیف سخت افزاری Verilog استفاده شده است، که دارای گذرگاه آدرس ۲۴ بیتی و داده ۳۲ بیتی می باشد که برای حافظه های همگام (SSRAM) طراحی شده است. شمای کلی این پیاده سازی در شکل۳ نشان داده شده است.

سخت افزار طراحی شده برای ارزیابی برروی FPGA های شرکت Xilinx پیاده سازی شده، که نتایج آن در جدول ۱ مشخص شده است. این نتایج با استفاده از نرم‌افزار Leonardo2002c به دست آمده‌است که بهترین نتیجه چه از جهت مساحت و چه از جهت فرکانس کاری در استفاده از Virtex-E به دست می آید که در آن می توان کلاک مدار را با فرکانسی در حدود MHZ ۷۰ تنظیم کرد. در شبیه سازی کد سخت افزاری مشخص می شود که با فرض اینکه هر دسترسی به حافظه یک کلاک طول بکشد، در هر ۷ کلاک می توان یک عمل جستجو را انجام داد که در نتیجه در هر ثانیه ۱۰M جستجو قابل انجام است که برای یک مسیریاب از نوع Access که هدف این پیاده‌سازی بوده جوابگو خواهد بود. همچنین برای اطمینان از صحت کد پیاده سازی شده در سخت‌افزار(Verification)، نتایج شبیه‌سازی تحت نرم‌افزار ModelSim5.5se، با نتایج به دست آمده از تابع جستجوی نرم افزاری (Golden-Model) مقایسه شدند.

**جدول۱- نتایج پیاده‌سازی روش در سخت‌افزار**

| Target Device (FPGA) | Area (LUTs) | Frequency (MHz) | Throughput (MLookup/s) |
|---|---|---|---|
| Spartan-II | ۳۹۷ | ۳۲/۵ | ۴/۶ |
| Virterx | ۴۵۲ | ۵۶ | ۸ |
| Virtex-E | ۴۴۷ | ۷۳/۳ | ۱۰/۴ |

## ٤- تخمین زمان اجرا و حافظه مصرفی

زمان اجرای الگوریتم جستجو، ثابت و حداکثر برابر سه بار دسترسی به حافظه است. در بهترین حالت با یک بار دسترسی پورت خروجی مشخص می‌شود. در نتیجه تعداد جستجوها در ثانیه با سرعت دسترسی به حافظه محدود می‌شود.

برای ساخت حافظه‌ای که عمل جستجو در آن انجام شود ابتدا باید جدول قطعه را با استفاده از تابع ArrayMaker ساخت و بعد با استفاده از تابع MemoryMaker آن را به صورت یک حافظه یکپارچه که عمل جستجو در آن قابل انجام باشد درآورد. زمان مورد نیاز برای ساخت جدول قطعه به تعداد IP هایی بستگی دارد که قرار است جدول با آنها ساخته شود. اگر فرض کنیم که تعدادIP های ورودی N باشد، بدترین توابع در O(N) عملیات مربوط به خود را انجام می‌دهند. البته در خلال ساخت جدول نیاز به عملیات مرتب‌سازی است که این کار در زمان O(N*log(N)) انجام می‌شود. خود تابع ArrayMaker هم در ابتدا همه خانه‌های جدول قطعه را مقداردهی می‌کند که این کار در زمان ثابت $O(۲^{۱۶})$ انجام می‌شود. در تابع MemoryMaker هم در ابتدا برای محاسبه حافظه مورد نیاز ابتدا کل جدول قطعه بررسی می‌شود که این کار در هم زمان ثابت $O(۲^{۱۶})$ انجام می‌شود. سپس محتوای

خانه‌های جدول به حافظه جدید منتقل می‌شود. پس ساخت حافظه جستجو در زمان $O(\max\{N*\log(N)$و$2^{16}\})$ انجام می‌شود. چون نسبت به تعداد IP ها ساخت حافظه در زمان کمی انجام می‌شود، به روز کردن جدول با ساخت مجدد جدول انجام می‌شود که برای افزایش سرعت و کارآیی می‌توان از دو حافظه استفاده کرد که در زمان به‌روزسازی حافظه جدید جایگزین حافظه قدیمی می شود. حافظه مصرفی به این صورت قابل محاسبه است، اندازه جدول قطعه ثابت و $3*2^{16}$ بایت (یعنی ۱۹۶/۶۰۸ کیلو بایت) است، به ازای هر خانه این جدول که نیاز به جدول آفست دارد یک آرایه برای CWA و یک آرایه برای CNHA نیاز است که اندازه آن‌ها بستگی به تعداد IPهایی دارد که در این خانه جدول قطعه قرار می‌گیرند. در بدترین حالت که همه $2^{16}$ IP مربوط به یک خانه جدول قطعه وجود داشته باشند و هر دو IP متوالی دارای پورت‌های خروجی متفاوت باشند، اندازه CWA برابر ۳۲ کیلو بایت و اندازه CNHA برابر ۶۴ کیلو بایت است. در نتیجه در بدترین حالت هر خانه جدول قطعه، ۹۶ کیلو بایت حافظه نیاز دارد. در حالت متوسط، ۴۵۰ تا ۴۷۰ کیلوبایت حافظه مورد نیاز است[۱].

## ۵- نتیجه‌گیری

در این مقاله ، الگوریتم ارائه شده در [۱] برای مسیریاب‌های Access تغییر و بهینه گردیده است. علاوه بر پیاده‌سازی سخت‌افزاری، مدل نرم‌افزاری آن نیز تهیه گردیده است که به عنوان بخشی از هسته نرم‌افزار مسیریاب می‌تواند مورد استفاده قرار بگیرد. سخت‌افزار طراحی شده در فرکانس کاری پایین، دارای کارایی بالایی بوده و براحتی قابل پیاده‌سازی بر روی قطعات معمول برنامه‌پذیر می‌باشد. سخت‌افزار پیاده‌سازی شده نیز در مسیریاب مورد مطالعه با نرخ ورودی STM-4 مورد استفاده قرار گرفته است.

## ٦-مراجع


[1] Nen-Fu Huang and Shi-Ming Zhao, “A Novel IP-Routing Lookup Scheme and Hardware Architecture for Multigigabit Switching Routers,” *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, VOL. 17, NO. 6, JUNE 1999.

[2] F. Baker, Ed. “Requirements for IP version 4 routers.” RFC 1812, June 1995.

[3] A. McAuley and P. Francis, “Fast routing lookup using CAM’s,” in *Proc. IEEE INFOCOM*, 1993, pp. 1382–1391.

[4] Y. Rekhter and T. Li, “An architecture for IP address allocation with CIDR,” RFC 1518, Sept. 1993.

[5] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, “Small forwarding tables for fast routing lookups,” in *Proc. ACM SIGCOMM’97*, France, pp. 3–14.

[6] W. Doeringer, G. Karjoth, and M. Nassehi, “Routing on longestmatching prefixes,” *IEEE/ACM Trans. Networking*, vol. 4, pp. 86–97, Feb. 1996.

[7] P. Gupta, S. Lin, and N. McKeown, “Routing lookups in hardware at memory access speeds,” in *Proc. IEEE INFOCOM’98*, Session 10B-1, San Francisco, CA, pp. 1240–1247.

[8] C. Labovitz, G. R. Malan, and F. Jahanian, “Internet routing instability,” in *Proc. ACM SIGCOMM’97*, France, pp. 115–126.

[9] B. Lampson, V. Srinivasan, and G. Varghese, “IP lookups using multiway and multicolumn search,” in *Proc. IEEE INFOCOM’98*, San Francisco, CA, pp. 1248–1256.

[10] S. Nilsson and G. Karlsson, “Fast address lookup for Internet routers.”, Available at http://www.nada.kth.se/˜snilsson/public/papers.html/.

[11] T. Pei and C. Zukowaki, “Putting routing tables in sillicon,” *IEEE Network*, Jan. 1992, pp. 42–50.

[12] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, “Scalable high speed IP routing lookups,” in *Proc. ACM SIGCOMM’97*, France, pp. 25–36.

[13] Stanford University Workshop on Fast Routing and Switching (Dec.1996). Available at http://tinytera.standford.edu/Workshop_Dec96/.