تولید اتوماتیک تجزیه کننده بسته برنامه‌پذیر: بهبود سخت افزاری معماری سوئیچ OPEN FLOW با استفاده از FPGA

عباس یزدی نژاد[۱]، کمال جمشیدی[۲]، علی بهلولی[۳]

[۱] دانشکده مهندسی کامپیوتر دانشگاه اصفهان ، abbasyazdinejad@eng.ui.ac.ir

[۲] دانشکده مهندسی کامپیوتر دانشگاه اصفهان ، jamshidi@eng.ui.ac.ir

[۳] دانشکده مهندسی کامپیوتر دانشگاه اصفهان ، bohlooli@eng.ui.ac.ir

چکیده

امروزه زیرساخت‌های شبکه درکاربردهای صنعتی، خانگی و تجاری به یک موضوع بحرانی تبدیل شده در حالی که شبکه کنونی استاتیک و غیر قابل برنامه‌ریزی است. شبکه نرم افزارمحور به تازه‌گی قابلیت برنامه‌پذیری و نوآوری در شبکه را توسط OpenFlow به میان آورده که انعطاف‌پذیری و مدیریت شبکه را در پی داشته است به طوری که هوش شبکه در کنترلر سوئیچ OpenFlow قرار گرفته و ترافیک شبکه را در سطح داده مدیریت می‌کند. تاکنون پیاده سازی‌هایی از تجزیه کننده بسته بر روی FPGA در شبکه نرم افزارمحور مطرح شده‌اند که فاقد انعطاف‌پذیری لازم و برنامه‌پذیری بوده‌اند و تنها از یک گراف تجزیه پشتیبانی می‌کنند که این خود باعث بروز محدودیت در ایجاد پروتکل‌های جدید شبکه می‌شود. در واقع تجزیه کننده‌های بسته ثابت بوده‌اند و فاقد انعطاف پذیری لازم برای پشتیبانی از خصوصیات نسخه‌های مختلف سوئیچ OPEN FLOW می باشند. دراین مقاله به تولید اتوماتیک، تجزیه کننده بسته برنامه‌پذیر برای سوئیچ OPEN FLOW با استفاده از Genesis بر روی FPGA پرداخته شده است که علاوه بر ایجاد انعطاف پذیری بالا در سوئیچ، قابلیت برنامه‌پذیری را دارا است و امکان پشتیبانی از گراف‌های تجزیه متفاوت را در زمان اجرا فراهم می‌کند. با تولید این تجزیه کننده بسته برنامه‌پذیر بر روی FPGA موجب بهبود عملکرد سوئیچ و بالا بردن انعطاف‌پذیری در سطح داده شبکه نرم افزارمحور خواهیم شد. شبیه سازهای انجام شده از تجزیه کننده بسته نشان دهنده عملکرد صحیح تجزیه کننده و برنامه‌پذیری آن در سطح شبکه می‌باشد.

واژه‌های کلیدی

تجزیه کننده بسته، Genesis، شبکه نرم افزارمحور، سوئیچ OPEN FLOW، FPGA.

۱- مقدمه

زیرساخت‌های شبکه در اینترنت و شبکه‌های تجاری به یک موضوع بحرانی تبدیل شده‌اند با توجه به افزایش دستگاه‌های تلفن همراه، افزایش خدمات ابر و محدودیت‌های پهنای باند معماری شبکه به شکل پیچیده‌ای در آمده است که با ظرفیت شبکه فعلی و نیازهای کاربران مطابقت ندارد. تکنولوژی شبکه فعلی شامل محدودیت هایی از جمله پیچیدگی، سیاست‌های ناهماهنگ، عدم مقیاس‌پذیری و وابستگی به توسعه‌دهندگان می‌باشد که پاسخ‌گوی نیازهای کاربران شبکه در شرکت‌ها، خانه‌ها و مدارس نیست [۱]. در این زمان الگوی شبکه تغییر کرد و شبکه‌نرم‌افزارمحور[1] به میان آمد. SDN یک معماری جدید در دنیای شبکه ایجاد کرده که در آن قسمت کنترل شبکه از قسمت ارسال یا انتقال داده جدا شده است و رفتار شبکه را می‌توان تغییر داد. SDN توسط بنیاد شبکه‌های باز مطرح شد که توجه بسیاری از محققان دانشگاهی و فروشندگان را به خود جلب کرد. بنیاد شبکه‌های باز یک سازمان که مسئول کنترل و انتشار مشخصات OpenFlow می باشد. OpenFlow یک تکنولوژی جدید براساس مفهوم شبکه‌نرم‌افزارمحور است که در آن سوئیچ تصمیم می گیرد که چه اقداماتی باید انجام دهد. پیاده‌سازی های متعددی از سوئیچOpenFlow برروی ساختارهای مختلف صورت گرفته است[۲]. OpenFlow توسط شبکه‌نرم-افزارمحور برروی دستگاهای مختلف و نرم افزارهای شبکه اجرا شده که افزایش کارایی و کنترل ترافیک را برای ابزارهای شبکه به دنبال داشته است. به تازگی سوئیچ OpenFlow برروی ساختارهای مختلف نرم افزاری (Linux, Open-WRT) و سخت افزاری(NetFPGA) پیاده‌سازی شده است[۳]. بیشتر محققان علاقه دارند که سوئیچ OpenFlow را برروی پلت فرم هایی براساس FPGA پیاده‌سازی کنند به دلیل اینکهFPGA ها

[1] **Software Defined Network(SDN)**

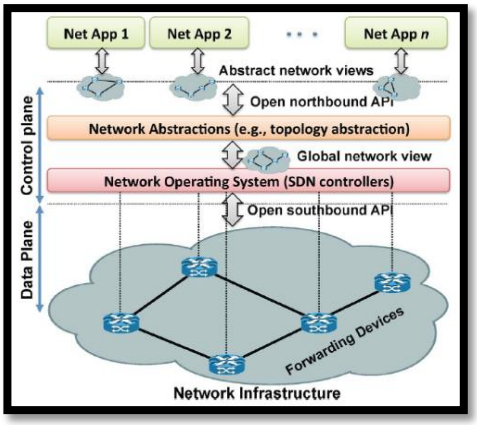
انعطاف پذیری، سرعت و قابلیت برنامه‌ریزی مجدد را دارند در مقایسه با ASICها که برای کاربردهای خاص استفاده می‌شوند و همه منظوره نیستند FPGA ها گزینه بهتری بوده و در ضمن اینکه ارزان تر و در دسترس تر هستند[۴]. آرایه‌های گیتی قابل برنامه ریزی میدانی[۲] شامل بلوک‌های منطقی پیکرپذیر[۳] به همراه اتصالات میانی برنامه‌پذیر هستند [۵]. بر خلاف مداهای مجتمع خاص منظوره[۴] که در تمام طول عمر خود تنها قادر به انجام یک عملیات هستند، یک تراشه‌ی بازپیکرپذیر می‌تواند در عرض چند میکرو ثانیه به یک پیکربندی جدید برنامه‌ریزی شود تا عملیات جدیدی را انجام دهد[۶]. بلوک‌های منطقی پیکرپذیر مهم‌ترین منابع منطقی برای پیاده‌سازی انواع مدارهای ترکیبی و ترتیبی هستند [۷]. معمولا برنامه‌نویسی یک FPGA با استفاده از زبان‌های توصیف سخت افزار نظیر VHDL و Verilog انجام می‌شود [۸،۹].در این مقاله به پیاده‌سازی و تولید اتوماتیک تجزیه کننده بسته برنامه پذیر توسط Genesis بر روی FPGA می پرداخته شده است که انعطاف پذیری و برنامه پذیری را در سطح سوئیچ OPEN FLOW و در نهایت در سطح داده شبکه به میان می آورد [۱].

۲- کارهای مربوطه

در زمینه طراحی و پیاده‌سازی سوئیچ بر روی بسترهای سخت افزاری و نرم افزاری کارهایی صورت گرفته و این موضوع مورد علاقه بسیاری از محققان بوده از جمله کارهای صورت گرفته می توان به Naous و همکاران[۱۰] اشاره کرد که سوئیچ OpenFlow را برروی NetFPGA پیاده سازی کرده اند. در این پیاده‌سازی تاخیر بالایی در وارد کردن یک جریان جدید به سوئیچ OpenFlow وجود دارد. با این حال این پیاده سازی شامل تاخیر بالا و انعطاف پذیری کمتری می‌باشد[4]. Bosshart [۱۱] و همکاران معماری از سوئیچ ارائه دادند کهRMT نامیده شده این معماری را برروی ASIC پیاده سازی کرده‌اند که از حافظه TCAM برای عمل مطابقت استفاده می‌کند که مدل OPEN FLOW را ارتقا داد ولی باز هم نتوانست محدودیت های OPEN FLOW را به طور کامل حل کند به طوری که نویسندگان آن عنوان کرده‌اند که طرحشان به واسطه تعداد مراحل پایپ لاین، جداول مطابقت، واحد عمل برای ساخت برروی تراشه ASIC محدود شده است و از این محدودیت ها نمی توان در ساخت یک طرح برروی ASIC اجتناب کرد. Liu به پیاده سازی سوئیچ SDN برروی FPGA پرداخته است[۳] که از تاخیر NetFPGA اجتناب کرده ولی بخش تجزیه کننده بسته آن ثابت می باشد که انعطاف پذیری و قابلیت بازپیکربندی برای پشتیبانی از سرآیندهای تعریف شده جدید بسته را ندارد و برای پشتیبانی از گراف‌های تجزیه جدید باید مجددا برنامه VHDL بخش تجزیه کننده آن نوشته شود در واقع این بخش فاقد برنامه پذیری لازم باتوجه به خصوصیات OPEN FLOW می باشد. در این مقاله هدف اصلی تولید اتوماتیک یک تجزیه کننده بسته برنامه پذیر توسط Genesis برای سوئیچOPEN FLOW برروی FPGA می باشد که انعطاف پذیری و قابلیت برنامه پذیری بالایی به سطح داده شبکه می‌دهد و اشکالات و نواقص کارهای قبلی را رفع می کند و از مزایای استفاده از FPGA بهره برده که در نهایت منجر به بهبود کارایی سوئیچ در شبکه می شود.

۳-معماری شبکه نرم افزار محور

در شکل۱ مولفه های اساسی SDN قرار دارد که از لایه کاربرد[۵]، لایه کنترل و لایه زیرساخت تشکیل شده است. در این شکل واسط های[۶] مختلفی برای ارتباطات سطوح مختلف دیده می شود نظیر واسط‌های شمالی و جنوبی که کنترلر اغلب از واسط جنوبی برای ارتباطات با APIها استفاده می‌کند. برنامه های کاربردی واسط شمالی شامل سیاست‌های تجاری و پیکربندی شبکه می باشد. واسط جنوبی شبکه همان openflow است که پروتکل اصلی برای ارتباطات کنترلر با سوئیچ های فیزیکی و مجازی می باشد[۱۲،۱۳]. کنترلر در قسمت میانی این معماری قرار دارد و مشابه یک میان افزار شبکه عمل می‌کندکه اجزای فیزیکی لایه فیزیکی شبکه نظیر مسیریاب‌ها، سوئیچ ها و دیواره آتش را از یکدیگر تفکیک می کند. شرکت‌ها با استفاده از SDN می توانند از طریق یک کنترلر مرکزی فارغ از سخت افزار و شرکت سازنده آن، شبکه را کنترل و مدیریت کنند[۱۴،۲].



شکل ۱: معماری سطوح SDN [۲]

همانطور که در شکل ۱ مشاهده می شود، کل شبکه از دو قسمت اصلی تشکیل شده است:

۱- سطح داده: در این قسمت المان‌های سخت افزاری و نرم افزاری همانند مسیریاب‌ها، سوئیچ ها و دیواره آتش قرار گرفته است و اتصال‌های بین این المان‌ها از طریق فیبر یا کابل مسی برقرار شده است [۱۴،۲].

۲- سطح کنترل: در این قسمت لایه کنترلی است و تمامی کنترلرها در این قسمت قرار می گیرد این قسمت متشکل از یکسری نرم افزارها و سرویس‌ها است که از لایه بالاتر خود یعنی لایه کاربرد دستور دریافت می کنند[۱۴،۲].

[2] FPGA
[3] CLB
[4] asic

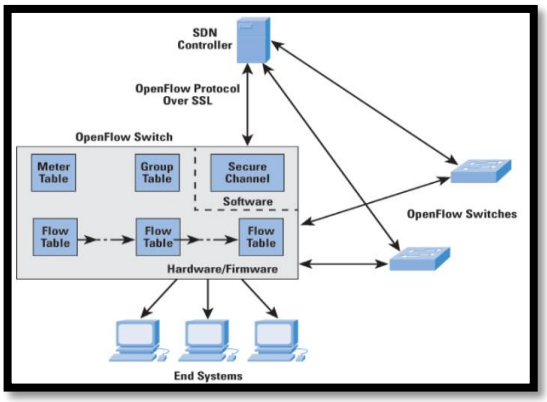
[5] Application
[6] Interface

۳-۱- پروتکل Open Flow

تکنولوژی OpenFlow سطح کنترل را از سطح داده جدا می‌کند و این اجازه را می‌دهد تا مدیران شبکه به منظور توسعه الگوریتم های خود برای کنترل جریان داده‌ها، بسته‌ها، مدیریت کارآمد تر شبکه و در نتیجه، انعطاف پذیری بیشتر در پاسخ به درخواست‌ها و نوآوری در شبکه بپردازند.چندین نسخه از خصوصیات OpenFlow تا کنون منتشر شده از جمله OpenFlow1.0 ارائه مدلی با یک جدول جریان و مجموعه ای ثابت از زمینه‌ها برای تطبیق،که از ۱۲ سرآیند بسته پشتیبانی میکند. OpenFlow1.1 گسترش یافته و سوئیچ را به حمایت از جداول جریان های متعدد، و پشتیبانی از MPLS، وا می دارد که از ۱۵ سرآیند پشتیبانی می‌کند. OpenFlow1.2 به پشتیبانی از IPv6 و توسعه بخش تطبیق می پردازد که از۳۶ سرآیند پشتیبانی میکند. OpenFlow 1.3 تونل زنی و انتزاع پورت منطقی، پشتیبانی سرویس های جدید، که از ۴۰سرآیند پشتیبانی می‌کند.OpenFlow 1.4 پشتیبانی از پورت ها نوری، گسترش نظارت بر وضعیت و توسعه پروتکل‌ها به آن اضافه شده است که از ۴۱ سرآیندپشتیبانی می‌کند. برای اطلاعات بیشتر در زمینه خصوصیات می توان به مراجع[۲,۳,۱۵,۱۶] رجوع کرد. باتوجه به خصوصیات مختلف OpenFlow و بروز رسانی آن سوئیچ همواره باید بتواند از این خصوصیات پشتیبانی کند در واقع اینجاست که نقش تجزیه کننده در سوئیچ برای پشتیبانی از سرآیندهای جدید و توالی سرآیندها در شبکه مشخص میشود که برنامه پذیری چه نقش به سزایی می تواند داشته باشد و انعطاف پذیری بالایی را به میان می آورد.

۴- معماری OpenFlow

انعطاف پذیری از مزایای کلیدی OpenFlow در مقایسه با پروتکل های موجود مانند IP و اترنت است. به طور کلی، استفاده از OpenFlow مزایایی در مجازی سازی شبکه و توزیع شدگی داشته است[۱۶]. سوئیچ OpenFlow به طور عمده از جداول جریان و یک رابط برای اصلاح مدخل های جدول جریان تشکیل شده است[۳]. کنترلر OpenFlow در مورد مسیر بسته‌های تصمیم گیری می‌کند. و کنترلر به سوئیچ OpenFlow از طریق لایه سوکت امن (SSL) متصل شده و تغییر مدخل جدول جریان از طریق همین رابط می باشد.کنترلر نقش به روز رسانی جداول جریان را در سوئیچ را برعهده دارد[۳].

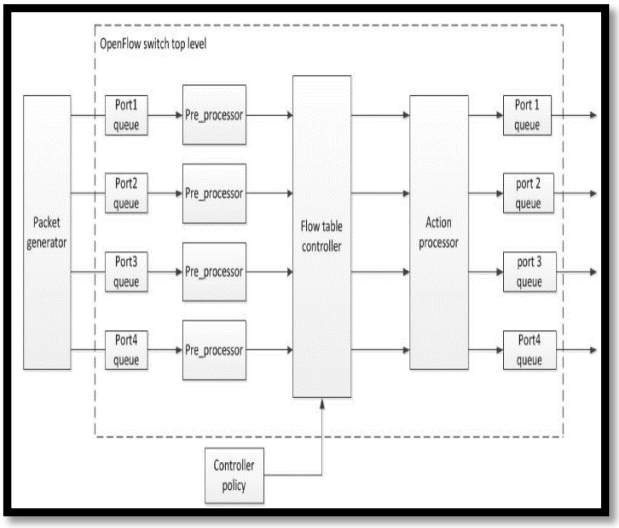


شکل ۲: معماری سوئیچ SDN [۳]

هر جدول جریان حاوی مدخل‌هایی برای شناسایی بسته های وارد شده است که براساس نوع سرآیند بسته که در جدول جریان می باشد اقدامات مربوط به آن انجام می‌شود. اگرسرآیند بسته ای با وارده های جدول مطابقت نداشته باشد به کنترلر فرستاده یا حذف می‌شود.

۵-معماری سوئیچ برروی FPGA

معماری سوئیچ OPENFLOW V1.1 که توسط Li طراحی شده را در شکل زیر می‌بینید[۳]. در این معماری سوئیچ دارای ۴ پورت ورودی برای دریافت بسته می باشد که پس از تجزیه یسته ها، آن ها را برروی پورت خروجی مناسب براساس قاعده مشخص شده در جداول مطابقت می‌فرستد که به معرفی اجزای سوئیچ پرداخته می‌شود:



شکل ۳: معماری سوئیچ OPEN FLOW V1.1[۳]

Packet_generator : به تولید بسته برای برای پورت‌های ورودی سوئیچ می پردازد.

Pre_Processor : بسته های تولیدی را از پورت ورودی دریافت می کند و عمل تجزیه را برروی بسته انجام می‌دهد. شناسایی، استخراج سرآیندها و زمینه‌های بسته در شبکه در این بخش است.

Flow Table Controller :جدول جریان سوئیچ می باشد که حاوی جریان‌هایی از بسته است که از این جداول برای انجام عمل مطابقت بسته-ها استفاده می‌شود.

Controller_policy : اعمال سیاست های لازم به Flow Table Controller از طریق قاعده و قانون هایی که به جداول مطابقت می-فرستد.

Action Processor : این بخش پس از این که عمل تطبیق در جداول جربان صورت می‌گیرد اقدامات مربوط به هر بسته در آن صورت می‌گیرد و بسته را به پورت خروجی مناسب می‌فرستد.

۵-۱- بهبود معماری سوئیچ بر روی FPGA

اجزا و معماری داخلی سوئیچ با عملکرد مربوط به هر بخش را در قسمت قبل دیدید در این زیر بخش با توجه به معماری سوئیچ ارائه شده توسط Li ، در قسمت Pre_Processor که معماری تجزیه کننده بسته آن ثابت بوده و فاقد توانایی در پشتیبانی از پروتکل‌های جدید در شبکه می‌باشد در

این مقاله به تولید اتوماتیک معماری سخت افزاری تجزیه کننده بسته برنامه پذیر توسط Genesis پرداخته شده است که ضمن حفظ کارایی و بهبود عملکرد، دارای انعطاف‌پذیری و قابلیت برنامه پذیری بیشتر در سوئیچ می باشد در مقابل بخش تجزیه کننده بسته قبلی که به صورت ثابت بوده و تنها از یکسری تعداد مشخص سرآیند بسته براساس توالی در ماشین حالت پشتیبانی می‌کند معماری ارائه شده قادر به پشتیبانی از هرنوع گراف تجزیه برای سرآیندهای شبکه می باشد و این امکان را به سوئیچ OPENFLOW می‌دهد که بتواند پروتکل‌های جدید را تعریف کند. در مدل قبلی تنها با نوشتن برنامه VHDL مجدد برای تجزیه کننده می‌توان از گراف‌ها و خصوصیات جدید OPEN FLOW پشتیبانی کرد ولی در اینجا ما به صورت اتوماتیک تجزیه کننده برنامه‌پذیر را تولید کرده که قابلیت پشتیبانی از هر گرافی در زمان اجر را دارد.

۶- Genesis

Genesis [۱۷] یک chip generator است که توسط دانشگاه استنفورد توسعه داده شده است. در واقع Chip Generator (CG) برای طراحی اتوماتیک یک سیستم برای تولید ASIC می‌باشد. ژنراتورهایی از قبیل (Min-Area FFT Generator, Floating Point Mult-Acc Generator, Stencil Engine Generator) در معماری Genesis طراحی و پیاده سازی شده‌اند که براساس پارمترهای ورودی ، کد قابل سنتز خروجی طرح مورد نظر را تولید کرده و در طراحی معماری مورد نظر از آن می‌توان استفاده کرد.

شکل ۴: Genesis chip Generator [۱۷]

کارها و فعالیت های پژوهشی متعددی با استفاده از Genesis صورت گرفته است که می توان به مراجع [۱۸-۲۳] اشاره کرد.

۶-۱-معماری سخت افزاری تجزیه کننده بسته برنامه پذیر

تجزیه کننده بسته برنامه پذیر ما کاملا اتوماتیک توسط Genesis تولید می شود. که از آن برای سوئیچ OPEN FLOW بر روی FPGA استفاده شده است. در اینجا Genesis را بر روی یکی از نسخه های لینوکس نسخه Ubuntu 15.04 ۶۴ بیتی نصب شده در شکل زیر شروع Genesis را در ترمینال لینوکس مشاهده می‌شود.

```
ccc@ubuntu: ~
ccc@ubuntu:~$ export PERL5LIB="$GENESIS_HOME/PerlLibs/ExtrasForOldPerlDistributi
ons"
ccc@ubuntu:~$ Genesis2.pl


-----------------------------------------------
--- Genesis Is Starting Work On Your Design ---
-----------------------------------------------

-----------------------------------------------
|            Genesis Release Info             |
|  $Change: 11879 $ --- $Date: 2013/06/11 $   |
-----------------------------------------------

-------------------------------------------------------------------------------
    !! THIS VERSION OF GENESIS2 IS NOT FOR ANY COMMERCIAL USE !!
     FOR COMMERCIAL LICENSE CONTACT SHACHAM@ALUMNI.STANFORD.EDU
-------------------------------------------------------------------------------

Genesis2::Manager->execute: Starting Auxiliary File Generation Phase
Genesis2::Manager->execute: Done With Auxiliary File Generation Phase

ccc@ubuntu:~$
```

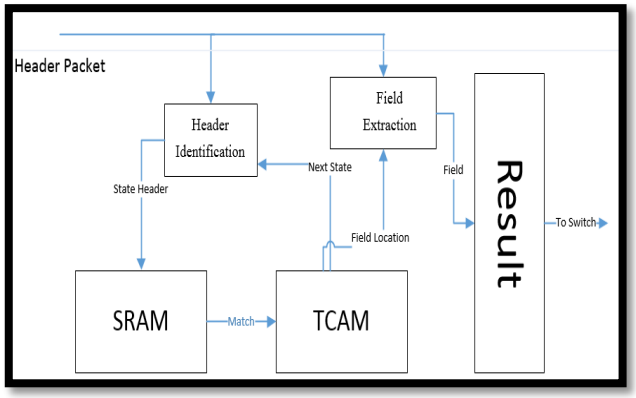
شکل ۵: Genesis start

Genesis در واقع ابزاری است که نمونه طراحی را با استفاده از معماری template ها و مجموعه‌ای از پارامترهای پیکربندی تولید می‌کند. Templatesها شامل کدهای Verilog و perl می‌باشند. کدهای Perl معماری انتخاب شده را به صورت کدهای Verilog در می آورند که در این مقاله معماری از تجزیه کننده، توسط Genesis به صورت اتوماتیک تولید شده است و از پارامترهای ورودی برای تولید تجزیه کننده برنامه پذیر توسط Genesis می‌توان از گراف تجزیه، عرض پردازش، عرض بافر، نام برد. خروجی Genesis تجزیه کننده بسته برنامه پذیر با کدهای قابل سنتز Verilog می باشد با توجه به خصوصیات دستورات ورودی، Genesis عملیات تولید تجزیه کننده را تماما اتوماتیک انجام میدهد در شکل زیر تولید موفقیت آمیز توسط Genesis را می‌بینند.

```
ccc@ubuntu: ~/Desktop/GENESIS/parser-gen-master4/build
        INFO:   Command line parameter override: path=top.parser, param name=Max
RdAmt, override val='15'
        INFO:   Command line parameter override: path=top.parser, param name=Pro
gBufWordWidth, override val='16'
        INFO:   Command line parameter override: path=top.parser, param name=Num
ExtractVectorInputs, override val='4'
        INFO:   Command line parameter override: path=top.parser, param name=TCA
MTableFile, override val='../examples/headers-simple-w016-ms015-l02-lw02-flaz0-e
1-eb04.tcam'
Genesis2::Manager->gen_verilog: Starting code generation from module top
Genesis2::Manager->execute: Done With Verilog Code Generation Phase

Genesis2::Manager->execute: Starting Auxiliary File Generation Phase
Genesis2::ConfigHandler->WriteXml: Now writing output XML file: parser.xml
Genesis2::Manager->execute: Done With Auxiliary File Generation Phase

-----------------------------------------------
--- Genesis Finished Generating Your Design ---
-----------------------------------------------

++++++++++++++++++++++++++++++++++++
Checking gen2.log in checkRun.pl
++++++++++++++++++++++++++++++++++++
ccc@ubuntu:~/Desktop/GENESIS/parser-gen-master4/build$
```

شکل۶: Genesis finished

ساختار این معماری تولید شده را در شکل زیر می‌بینید. تجزیه کننده بسته های ورودی را دریافت می‌کند.



شکل ۶: Programmable packet parser

پس از ورود بسته‌ها به تجزیه کننده، بخش Header Identification براساس گراف تجزیه به شناسایی سرآیند بسته‌های ورودی می پردازد. این بخش از حافظه TCAM، که حافظه‌هایی برای مطابقت سه تایی می باشد استفاده می‌کند یعنی علاوه بر ۰ و ۱ از don't care هم پشتیبانی می‌کند در واقع عمل مطابقت سرآیندها در TCAM که حافظه هایی برحست محتوا می‌باشند صورت می‌گیرد و بعد در SRAM که براساس hash table می باشد مطابقت دقیق صورت می‌گیرد و حالت بعدی و نوع سرآیندهای بعد از این سرآیند مشخص میگردد و در نهایت Field Extraction به خارج کردن زمینه‌هایی از جمله آدرس مبدا و مقصد ، آفست، نوع سرآیند مشخص شده می‌پردازد و در نهایت در Result بافر می‌شوند تا به جداول مطابقت در سوئیچ فرستاده شوند و اقدامات مربوط به هر بسته براساس فرمان کنترلر که به سوئیچ داده شده صورت بپذیرد.

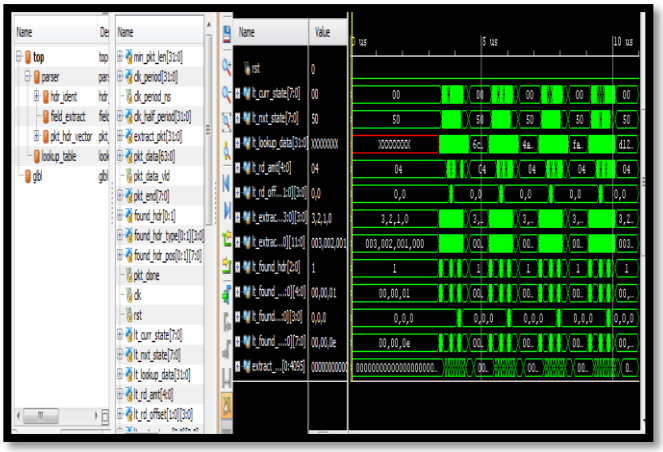
۷-شبیه سازی

در این قسمت ما به شبیه سازی معماری تجزیه کننده بسته تولید شده توسط Genesis می پردازیم عملیات شناسایی سرآیندها در تجزیه کننده به صورت ترتیبی می باشد و هر سرآیند در شبکه اطلاعات منحصر بفرد خودش را دارد در هر سرآیندی یکسری اطاعات در مورد سرآیند فعلی ازجمله طول سرآیند و نوع سرآیند بعد از آن در اختیار ما می‌گذارد در شکل زیر اطلاعات یک بسته را می‌بینید که ازسرآیندهای اترنت، VLAN ، IPV4 و TCP تشکیل شده است.

```
            //Packet Data in Network
// Header Packet: ethernet@0(14) ieee802-1q@14(4) ipv4@18(28) tcp@46(24)
    assign pkts[31] = {
        8'hc4, 8'hbb, 8'hde, 8'h8b, 8'h41, 8'hc3, 8'hb8, 8'h6a,
        8'h55, 8'h72, 8'h47, 8'h8a, 8'h81, 8'h00, 8'hd9, 8'hd1,
        8'h08, 8'h00, 8'h07, 8'he2, 8'h63, 8'he9, 8'h3f, 8'h83,
        8'h00, 8'h00, 8'h15, 8'h06, 8'hc2, 8'hc5, 8'hf2, 8'h93,
        8'ha9, 8'h70, 8'h44, 8'h86, 8'hd4, 8'h1c, 8'h4a, 8'h65,
        8'h89, 8'h97, 8'h55, 8'h36, 8'h8f, 8'h94, 8'h95, 8'h28,
        8'hd5, 8'ha6, 8'hcd, 8'hca, 8'h88, 8'hb5, 8'h01, 8'h92,
        8'h98, 8'h6f, 8'h60, 8'h7b, 8'h14, 8'hf1, 8'hfb, 8'h6b,
        8'h99, 8'hd4, 8'hf6, 8'hb3, 8'hbf, 8'h47
    };
```

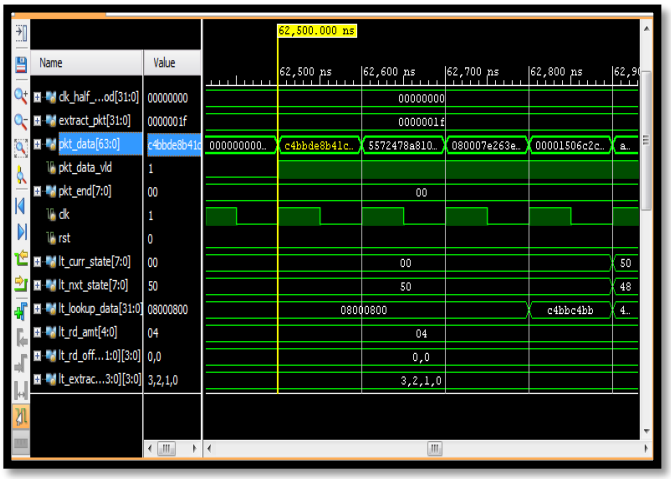
شکل ۷: Packet E/V/ip4/tcp

که پس از ورود بسته به معماری تجزیه کننده برنامه پذیر عملیات پردازش بر روی بسته صورت گرفته و اطلاعات مربوط به سرآیندها و زمینه‌ها برای خارج شدن بدست می آید در شکل زیر یکسری نتایج از آن آمده است. برای پیاده سنتز و پیاده سازی معماری تجزیه کننده بسته از Xilinx 870HT FPGA Virtex-7 استفاده شده است و نتایج بدست آمده از شبیه‌سازی این معماری توسط نرم افزار - Vivado Design Suite Xilinx نسخه ۲۰۱۵,۲ می‌باشد.



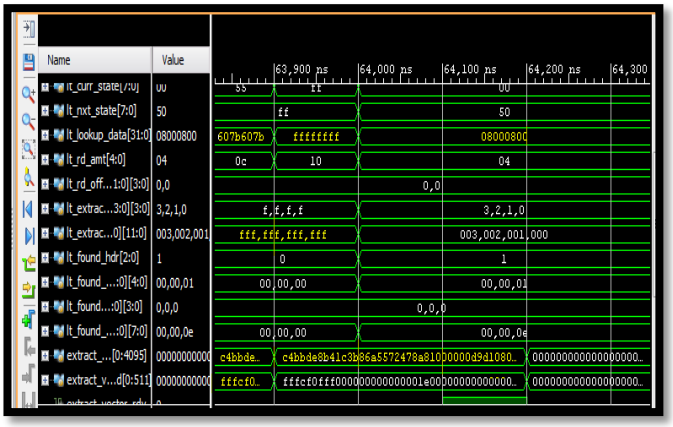
شکل ۹: Simulation programmable packet parser in vivado

نتایج شبیه سازی مربوط به بسته در شکل ۸ می‌باشدکه بسته با C4bb هگز شروع می‌شود که در زمینه Pkt_data شکل زیر آمده است.



شکل ۸: programmable packet parser in vivado

در شکل بعدی اطاعات مربوط به سرآیند بعدی یعنی IPv4(۰۸۰۰)، استخراج آدرس مبدا و مقصد، آفست و سایر اطلاعات مربوط به سرآیند فعلی صورت می‌گیرد.



شکل ۹: programmable packet parser in vivado

۸-نتیجه گیری

با توجه به سرعت پیشرفت شبکه‌های کامپیوتری، ایجاد قابلیت‌های جدید و پاسخ به درخواست های کاربران نیاز به برنامه پذیری در سایر زیر ساخت ها و تجهیزات شبکه حس می‌گردد باتوجه به اینکه تجزیه کننده بسته یک عنصر حیاتی و کلیدی برای سوئیچ OPEN FLOW می باشد لازم است که از انعطاف پذیری بالا برخوردار بوده ضمن اینکه با برنامه پذیری به پشتیبانی از OPEN FLOW در جهت تعریف پروتکل‌های جدید بپردازد و موجب افزایش عملکرد و کارایی در سطح داده SDN شود. در این مقاله نشان داده شد که با تولید اتوماتیک تجزیه کننده بسته برنامه‌پذیر بر روی FPGA منجر به بهبود عملکرد سوئیچ OPEN FLOW می‌شود.

۹-منابع

[1] Nunez-Martinez, Jose, Jorge Baranda, and Josep Mangues-Bafalluy. "A service-based model for the hybrid software defined wireless mesh backhaul of small cells." In *Network and Service Management (CNSM), 2015 11th International Conference on*, pp. 390-393. IEEE, 2015.

[2] D. Kreutz, F. M. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, pp. 14-76, 2015.

[3] Liu, Ting. "Implementing Open Flow Switch Using FPGA Based Platform." Information Technology, Mathematics and Electrical Engineering, pp. 1-138,2014.

[4] M. Wielgosz, M. Panggabean, J. Wang, and L. A. Rønningen, "An FPGA-based platform for a network architecture with delay guarantee," *Journal of Circuits, Systems, and Computers*, vol. 22, p. 1350045, 2013.

[5] Maxfield, Clive. *FPGAs: Instant Access: Instant Access*. Newnes, 2011.

[6] R. Wain et al, "An overview of FPGAs and FPGA programming; Initial experiences at Daresbury," Technical report, CCLRC Daresbury Laboratory, Daresbury, Warrington, Cheshire, UK, 2006.

[7] Virtex-6 FPGA Configurable Logic Block User Guide, Xilinx Corporation, February 3, 2012.

[8] R. Lipsett, E. Marschner, and M. Shahdad, "VHDL – The Language," IEEE Design & Test of Computers, pp. 28–41, April 1986.

[9] S. Palnitkar, "Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition", Prentice Hall PTR, February 2003.

[10] Naous, Jad, et al. "Implementing an OpenFlow switch on the NetFPGA platform." Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. ACM, 2008.

[11] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, et al., "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in ACM SIGCOMM Computer Communication Review, pp. 99-110, 2013.

[12] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, et al., "Onix: A Distributed Control Platform for Large-scale Production Networks," in OSDI, pp. 1-6, 2010.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, et al., "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol. 38, pp. 69-74, 2008.

[14] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-defined Internet architecture: Decoupling architecture from infrastructure," in Proceedings of the 11th ACM Workshop on Hot Topics in Networks, pp. 43-48, 2012.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, et al., "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol. 38, pp. 69-74, 2008.

[16] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using openflow: A survey," Communications Surveys & Tutorials, IEEE, vol. 16, pp. 493-512, 2014.

[17] Shacham, Ofer, et al. "Rethinking digital design: Why design must change." Micro, IEEE 30.6 (2010): 9-24.

[18] Richardson, Stephen, et al. "An area-efficient minimum-time FFT schedule using single-ported memory." *Very Large Scale Integration (VLSI-SoC), 2013 IFIP/IEEE 21st International Conference on*. IEEE, 2013.

[19] Shacham, Ofer, et al. "Avoiding game over: Bringing design to the next level." Proceedings of the 49th Annual Design Automation Conference. ACM, 2012.

[20] Wachs, Megan A. Specifying and Validating Memory Protocols for Chip Generators. Diss. STANFORD UNIVERSITY, 2013.

[21] Shacham, Ofer, et al. "Rethinking digital design: Why design must change." Micro, IEEE 30.6 (2010): 9-24.

[22] Zhu, Qiuling, et al. "Polar format synthetic aperture radar in energy efficient application-specific logic-in-memory." Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on. IEEE, 2012.

[23] Richardson, Stephen, et al. "An area-efficient minimum-time FFT schedule using single-ported memory." Very Large Scale Integration (VLSI-SoC), 2013 IFIP/IEEE 21st International Conference on. IEEE, 2013.