Embedded EtherNet و مقدمات شبکه

فهرست:

مقدمه :

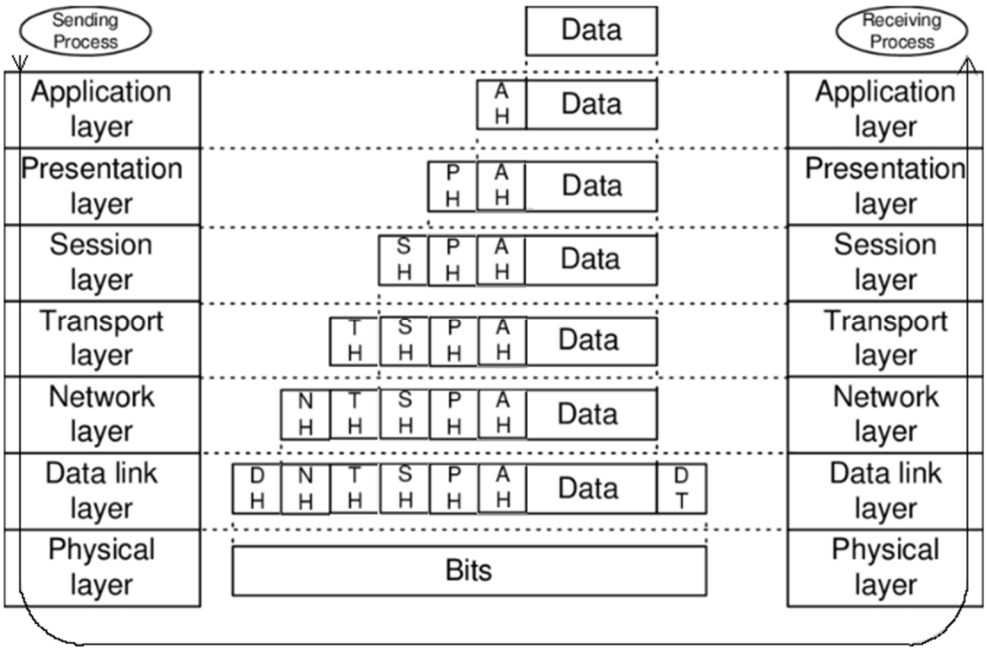
- در این یادداشت تصمیم داریم طریقه ارتباط یک برد الکترونیکی با یک کامپیوتر از طریق پورت اترنت یا آنطور که معمول است پورت شبکه ؛ به ساده ترین حالت نشان دهیم و ضمن آموزش آن با مفاهیم اصلی شبکه آشنا شویم. چنانچه شما با شبکه به طریق دیگری سروکار دارید (ادمین شبکه، فروشنده تجهیزات ؛ طراح یا نصاب شبکه ؛ برنامه نویس تحت وب و ....) ممکن است محتویات این سند منظور شما را برآورده نکند؛ هرچند اصول اولیه شبکه در آن آمده است. این سند در دو بخش ارایه میشه. بخش اول برقراری ارتباط اترنت یک برد الکترونیکی و کامپیوتر رو نشون میده و ساختار شبکه رو لایه لایه میریم بالا؛ در این مسیر از پکت اولیه اترنت شروع می کنیم؛ بعد به ترتیب پروتکل های HHTP,TCP,DHCP,UDP,ICMP,IP,ARP رو توضیح میدیم و اجرا می کنیم. در بخش دوم یا همون ضمایم، بعضی مفاهیم که یادگیریشون خوبه اما برای شروع ضروری نیستند رو بررسی می کنیم.
- بطور کلی، نوشته یا کتاب های فنی، با یک فهرست و بعد یک مقدمه عریض و طویل شروع می شوند. بخش ها جدا از هم و مفاهیم از یک نقطه شروع و تک تک مفاهیم ارایه می شوند. در مقدمه ممکن است مواردی مثل تاریخچه، تعاریف اولیه، اصطلاحات و... قرار داده شود که هرچند دانستن آنها مفید بنظر می رسد اما برای شروع و در ابتدا ضروری نیستند؛ مخصوصا اینکه این مبحث (شبکه)؛ اولا بقدری گسترده است که عملا محال است مسایل مرتبط با آن در یک کتاب گنجانده شود؛ ثانیا اصطلاحات و لغات مخفف و سرنام (abbreviation) در شبکه چنان گسترده است که با بمباران اطلاعاتی خواننده در طول مقدمه؛ ممکن است خواننده از ادامه راه مایوس شود (در صفحه ویکی پدیای یکی از پروتکل ها بیش از صد نام اختصاری آمده است!). ما با یک مثال شروع به یادگیری شبکه می کنیم و در طی این مسیر بخش های مختلف را همچون قطعات مجزای یک پازل کنار هم قرار می دهیم. این روش همون چیزیه که خودم تونستم ارتباط اترنت رو راه بندازم و با مابقی پروتکل ها آشنا بشم؛ امیدوارم برای شما هم همینطور باشه.
- در این نوشتار سعی شده تا حد امکان از کلمات فارسی استفاده بشه؛ هرچند خودمون رو محدود به این موضوع نکردیم و اولویت اولمون انتقال مفهوم بوده لذا در جای جای این نوشته ممکنه شما کلمات و مصطلاحات انگلیسی یا حتا تلفظ آن ها (مثل اترنت، کلاینت، سرور، پورت و ...) رو ببینید. سبک نوشتن هم به صورت محاوره ایه اما این هم حتمی نیست و بعضی جاها رسمی نوشتیم. تمام متن فی البداهه و در اوقات آزادم نوشته شده و احتمالا ویرایش آنچنانی نخواهد شد. از طرفی ممکنه متن دارای اشتباه و یا بی دقتی هایی باشه؛ امیدوارم بر من ببخشید.

سید مجتبی داشخانه – پاییز 1403

شبکه و Embedded EtherNet (اترنت نهفته):

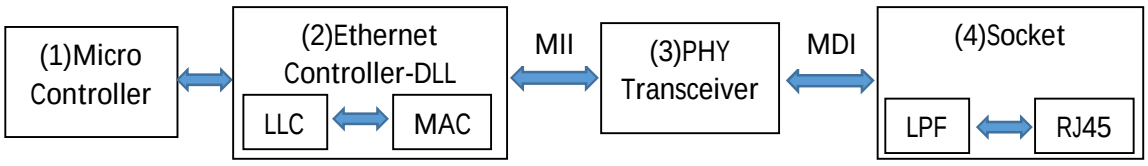
اگر بخواهیم در چند جمله، به مخاطب که احتمالا یک مهندس طراح الکترونیک هست؛ بگیم که چطور باید اترنت رو راه بندازه و در واقع نقشه مسیر یا سرنخ رو بهش بدیم؛ اینطور باید بگیم: " ابتدا باید سخت افزار اترنت رو راه بندازید؛ این سخت افزار از دو قسمت تشکیل شده. کنترلر اترنت و ترانسیور PHY. در نتیجه اگر میکروکنترلرتون، تنها بخش اترنت کنترلر داره، باید PHY رو جدا تهیه کنید. بعد از راه اندازی تراشه کنترلر اترنت به همراه PHY؛ باید بتونید با استفاده از میکروکنترلر و تراشه اترنت؛ فریم هایی به فرمت Ethernet ii که بهش IEEE 802.3 هم میگن؛ رو ارسال و دریافت کنید. از اینجا به بعد، مابقی پروتکل ها عموما در نرم افزار میکروکنترلر پیاده سازی میشه. پیشنهاد من اینه در مرحله اول ARP و IP کدنویسی بشوند. در مرحله بعد ICMP، که بتونید حداقل دستور پینگ رو اجرا کنید و از برقراری یک ارتباط درست اطمینان کسب کنید. در گام بعدی UDP و TCP و در گام های بعد پروتکل های لایه های بالاتر مثل DHCP,HTTP,DNS یا SMTP ". چون اولین و سخت ترین مرحله، راه اندازی تراشه هست؛ شاید نیاز باشه یک بار متن رو بخونید و بعد برید سراغ سخت افزار!

اگه قرار باشه چند وسیله با هم ارتباط داشته باشند ما یک شبکه داریم. واضحه که شبکه های مختلفی بر اساس نیاز طراحی شده ن و از یک جایی به بعد تصمیم گرفته شده مفاهیم و مدل های شبکه، استاندارد بشه تا هر شبکه و یا تجهیز جدیدی، با استفاده از این قالب استاندارد، دارای تعریف ساده ای باشه که براحتی بشه باهاش آشنا شد و اون رو راه انداخت. احتمالا شما هم نام دو مدل OSI و TCP/IP Stack رو شنیدید. در این دو مدل که شباهت بسیار زیادی هم با هم دارند؛ سعی شده تمام وظایف یک شبکه در قالب 7 (در دومی 5) لایه توضیح داده بشه. هر لایه وظیفه مشخصی داره و عملا شبکه به صورت ماژولار تعریف شده. برای شروع نیاز نیست که با تمامی این لایه ها آشنا باشید. اما اگر بخواهیم به طور خلاصه و در چند سطر رفتار شبکه رو توضیح بدیم؛ اینطوریه که داده اصلی یا نهایی که مورد استفاده کاربر یا سیستم هست؛ در هنگام ارسال، از بالاترین لایه مسیر خودش رو شروع میکنه؛ در هر لایه یک بخش به عنوان هدر (Header، سرآمد) به اون اضافه میشه و تحویل لایه پایین تر میشه. بالاخره داده ها به لایه دوم می رسند. در این لایه علاوه بر هدر؛ یک فوتر (footer) هم به داده ها اضافه میشه و به لایه اول داده میشه. لایه اول هیچ هدر یا فوتری اضافه نمیکنه و کارش، فقط تبدیل 0 و 1 های دیجیتال به سیگنال های فیزیکی (ولتاژ، موج، نور) هست یا به قولی " ارسال بیت های منطقی توسط رسانه فیزیکی (medium)". در سمت گیرنده عکس این عملیات اجرا میشه. این داده ها میتونن داده های چند سنسور از یک برد الکترونیکی باشند یا تگ های HTML برای نمایش در جستجوگر (Browser) شما. تفاوت دو مدل در اینه که لایه های 5 تا 7 در مدل OSI در مدل TCP/IP تنها در یک لایه قرار داده شده ن و وظایف 3 لایه بالایی در یک لایه خلاصه شده است(در ضمیمه [7] این دو مدل رو کامل تر بررسی کردیم). برد شما در ساده ترین حالت باید لایه های 1 و 2 رو پوشش بده؛ مابقی لایه ها میتونن در نرم افزار یا سخت افزار اجرا بشن.



بخش سخت افزاری اترنت:

یک برد با قابلیت برقراری ارتباط اترنت، در ساده ترین حالت به صورت شکل 1 بر روی برد الکترونیکی ایجاد می شود. همونطور که گفتیم "ساده ترین حالت" یعنی اینکه تنها لایه های 1 و 2 از مدل، درون سخت افزار هست.



شکل 1: سخت افزار اترنت برای پیاده سازی لایه های 1 و 2 از مدل شبکه

سخت افزار مربوط به شبکه از قسمت های زیر تشکیل شده؛ در ابتدا پردازنده ما هست که برای این قسمت از یک میکروکنترلر استفاده کرده ایم. میکروکنترلر در واقع مبدا یا مقصد داده نهایی است و لایه های 3 تا 7 رو در نرم افزار پوشش میده. ارتباط بین بخش پردازنده (1) و بخش کنترلر اترنت (2) بستگی به تراشه انتخاب شده دارد (SPI, uart ,...). بخش های دیگه عبارتند از :

LLC: Logical Link Controller کنترلر ارتباط منطقی

امور مرتبط با تشکیل فریم (frame) اترنت اینجا انجام می شود.

MAC: Media(Medium) Access Controller کنترلر دستیابی به رسانه

امور مرتبط به آدرس دهی در اینجا انجام می شود.

DLL: Data Link Layer

به مجموع دو قسمت LLC و MAC در مدل شبکه OSI (ضمیمه [7]) اصطلاحا لایه لینک داده گفته می شود. این قسمت، لایه دوم از مدل های OSI و TCP/IP Stackرا تشکیل می دهد و گاهی به تراشه های آن فقط با عنوان "مک کنترلر" یا MAC یاد می شود. لایه یک در مدل های شبکه، قسمت PHY یا لایه فیزیکی است که در شکل بالا با شماره (3) مشخص شده است.

MII: Medium Independent Interface رابط مستقل از رسانه

چرا مستقل؟ چون فریم اترنت در حالت منطقی مستقل از لایه فیزیکی و نحوه ارسال بیت ها است. بیت ها ممکن است به صورت پالس های ولتاژ یا جریان، پالس های نور یا حتا به صورت بی سیم (امواج) ارسال شوند. در این نوع ارتباط؛ در هر کلاک؛ 4 بیت از MAC به PHY ارسال می شود؛ پس برای یک ارسال یک بایت (8 بیت) تنها به دو کلاک نیاز است. در سرعت 10Mb این بخش با کلاک 2.5MHz و در سرعت 100Mb با کلاک 25MHz کار می‌کند. اما از آنجایی که در این ارتباط، به 16 پین نیاز است؛ برای کاهش تعداد پین (به 9 پین)، از نوع ارتباط RMII (Reduced MII) استفاده می شود که در هر کلاک دو بیت ارسال/دریافت می کند و به کلاک 50MHz نیاز دارد. نوع تک بیتی آن به نام SNI(Serial Network Interface) یا SMII نیز وجود دارد که 7 پین نیاز دارد و فقط در حالت 10Mb استفاده می شود (کمتر هم مورد استفاده قرار می گیرد). اطلاع در مورد MII، وقتی حایز اهمیت است که بخش کنترلر اترنت از بخش ترانسیور PHY جدا باشد (مبحث گسترده ای هم هست). توجه داشته باشید که ارتباط MII برای ارسال و دریافت داده های روی لایه فیزیکی و تحویل به /از مک کنترلر استفاده می شود. برای ارتباط با خود تراشه (منظور ارتباط MAC و PHY) و تنظیم ثبات های داخلی آن از ارتباط سریال جداگانه ای با دو پین برای کلاک و داده استفاده می شود. از آنجاییکه عملکرد بخش PHY در اسناد IEEE به صورت استاندارد تعریف شده است؛ تراشه های PHY، ساختار داخلی و عملکرد تقریبا مشابهی دارند و کدهای پیاده سازی شده برای هر تراشه را می توان با کمترین تغییر، برای تراشه های دیگر به کار گرفت. (برای 10BaseT و 100BaseTX دو سند IEEE802.3i, IEEE802.3u رو ببینید)

PHY : PHYsical transceiver فرستنده/گیرنده فیزیکی

وظیفه این بخش تبدیل بیت های منطقی به ماهیتی (سطوح ولتاژ، جریان الکتریکی، نور) است که سخت افزار باید بفهمه تا بتونه ارسال یا دریافت کنه. در حالت ارسال، بخش PHY از طرف مک کنترلر بایت هایی را دریافت

میکنه (به پهنای 4 ؛ 2 یا 1 بیت) و اون ها رو به پالس هایی با ماهیت مشخص تبدیل میکنه. در حالت دریافت نیز عکس عمل فوق انجام می گیرد. همچنین این بخش مسوولیت پیاده سازی قابلیت هایی مثل AutoNegotiation, Cillision detect, LTP و ... رو بر عهده داره؛ در نتیجه بخش PHY با روش ارسال و دریافت داده ها؛ بررسی خطا؛ تست برقراری لینک و تعیین خودکار نوع ارتباط (سرعت؛ دوطرفه بودن و..) ارتباط مستقیم داره. گاهی به این بخش، تنها Transceiver یا فقط PHY گفته میشه.

MDI: Medium Dependent Interface رابط وابسته به رسانه

چرا وابسته؟ چون به دلیل نوع رسانه (Media یا Medium) انتخابی و ارتباط فیزیکی؛ باید بیت ها به گونه ای به لایه فیزیکی و ملحقات آن مثل فیلتر، داده شوند که قابل تفسیر و اجرا باشند. مجددا یادآور می شویم که بیت ها ممکن است به هر نحوی (ولتاژ ، جریان، نور یا امواج الکترومغناطیسی) ارسال شوند و همونطور که در شکل مشخصه، MDI رابط بین PHY و فیلتر (یا به طور کلی سوکت) هست. گاهی به این بخش PMD (Physical Medium Dependent) هم گفته میشه.

LPF: Low Pass Filter فیلتر پایین گذر

این قسمت شامل یک فیلتر پایین گذر؛ مختص ارسال بیت ها به صورت پالس های الکتریکی دیفرانسیلی است و معمولا به صورت ترانسفورماتوری پیاده سازی می شود اما امکان ساخت آن با روش های دیگر مثلا با خازن نیز وجود دارد (در حالتی که ارتباط در یک محیط آزمایشگاهی و احتمالا در یک فاصله کم صورت می گیرد). این قسمت ممکن است در انواع دیگر، طور دیگری باشد یا حتا نباشد. همچنین به یاد داشته باشید که تحریک ترانس ها در دو مد جریانی یا ولتاژی ممکنه انجام بشه.

RJ45: Registered Jack 45

در انتها نیز سوکت رو داریم که در حالت استاندارد از سوکت RJ45 استفاده می شود اما بنا به سلیقه طراح؛ انتخاب هر سوکتی که حداقل دارای 4 پین باشد؛ آزاد است (مثلا در دوربین های صنعتی یا قطعات محرک، از انواع سوکت قفل شونده استفاده می شود).

پیشنهاد می کنم، خیلی خودتون رو درگیر این لغات اختصاری و اصطلاحات نکنید. به علت پیچیدگی ارتباط، سعی شده طراحی، کاملا ماژولار باشه و هر بخش کار مخصوصی رو انجام بده. شاید در ابتدا گیج کننده به نظر برسه اما کم کم با این کلمات مانوس شده و متوجه می شویم که بودنشان مفید (و گاهی الزامی) هست. به عنوان مثال اگر بخواهیم از فیبر نوری استفاده کنیم؛ کافیه قسمت PHY تغییر کنه و همچنین سوکت اختصاصی فیبرنوری استفاده بشه و مابقی بخش ها مثل سابق به کارشون ادامه میدن. خب حالا بریم سر وقت این بخش ها:

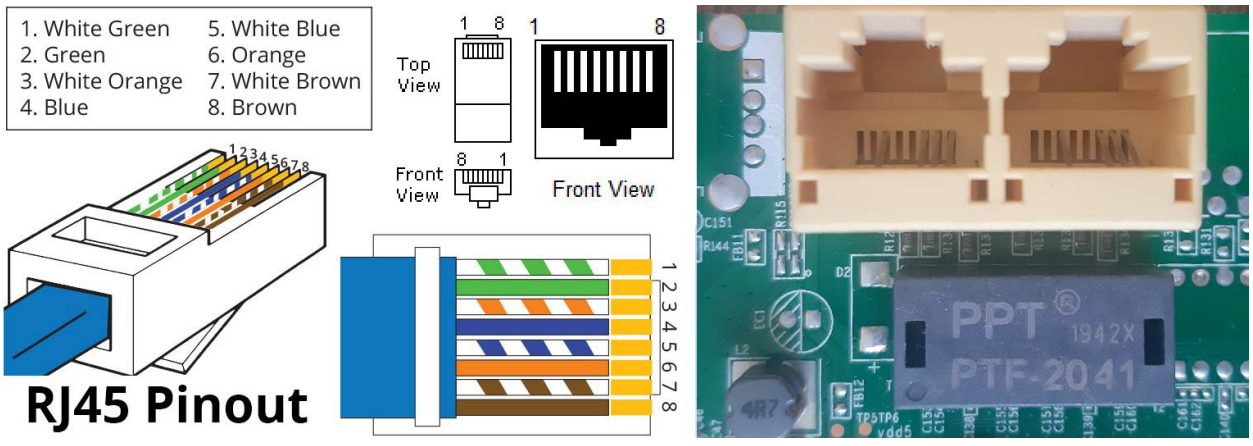
بخش 1: پردازنده یا میکروکنترلر.

بخش 2: کنترلر اترنت.

بخش 3: فرستنده/گیرنده PHY.

بخش 4: فیلتر و سوکت

در تجهیزات صنعتی و کارت های شبکه قدیمی، بخش 4 واقعا دارای دو قسمت مجزای سوکت و فیلتر بود (قطعه سیاه رنگ در تصویر سمت راست از شکل 2)؛ اما در حال حاضر عموما بخش فیلتر درون سوکت جاسازی میشه (کمتر شدن فضای اشغال شده روی PCB ؛کم شدن فاصله سوکت تا فیلتر و همچنین زیباتر شدن برد) و در نتیجه سوکت شبکه، دارای فیلتر نیز هست. سوکت HR911105A که ما در بردمون استفاده کردیم، هر دو بخش را داراست. لازم به ذکر است نوع نری روی کابل و مادگی روی برد قرار می گیرد. همچنین در شکل 2 پین اوت و رنگبندی استاندارد یک کابل اترنت را مشاهده می کنید. دو استاندارد برای رنگ بندی سیم ها تعریف شده اما توجه داشته باشید که بسته به نوع کابل؛ straghit (مستقیم یا یک به یک) یا cross (ضربدری یا متقاطع؛ اتصال TX یک سمت به RX طرف دیگر و بالعکس) و همچنین با توجه به سلیقه شرکت سازنده، ممکن است رنگبندی متفاوت باشد. به علت قابلیت تشخیص خودکار نوع کابل در تراشه (IC, Chip) های جدید، استفاده از هر دو نوع کابل (مستقیم یا ضربدری) میسر است. در ضمیمه[9] توضیحات بیشتری در این مورد داده خواهد شد.



شکل 2: سوکت RJ45

و اما نکته مهم در هنگام طراحی سخت افزار از دید یک مهندس طراح الکترونیک، ترکیب بخش های 1 تا 3 در شکل 1 هست؛ که چندین حالت را بوجود می آورد.

یک حالت اینه که هر سه قسمت، درون یک تراشه باشند؛ یعنی میکروکنترلر ، هم کنترلر اترنت رو داشته باشه و هم فرستنده/گیرنده PHY رو؛ مثل بعضی از میکروکنترلرهای خانواده PIC همانند PIC18F87J60. این میکروکنترلرها عموما قیمتی بیشتر از میکروهای ساده دارند و کاربر رو در انتخاب میکروکنترلر محدود می کنند.

حالت دیگه اینه که میکروکنترلر ما فقط دارای کنترلر اترنت باشه؛ لذا ما باید بخش PHY رو جداگانه به مدار اضافه کنیم. مثل میکروهای STM32f105/107 یا LPC1763~69 و جهت بخش PHY نیز می توان از تراشه هایی مثل W3100A,DP83848,LAN8742 استفاده نمود.

حالت سوم اینه که هر سه بخش، جدا باشند. این حالت در حال حاضر به ندرت استفاده می شود؛ چون عموما بخش کنترلر اترنت، یا درون پردازنده قرار می گیرد یا همراه با قسمت PHY ارایه می شود. در حالت های دوم و سوم؛ شما علاوه بر اینکه باید به ارتباط با مک کنترلر مسلط باشید؛ باید به ارتباط MII و بخش PHY نیز تسلط کافی داشته باشید تا بتونید برد رو اصطلاحا راه اندازی کنید. راه اندازی این دو بخش خودش نیاز به وقت و اطلاعات دو چندان داره.

و حالت نهایی که مقصود ما نیز هست (به این دلیل که بتوانیم از اترنت، روی هر میکروکنترلری حتی میکروهای ضعیفتری مثل خانواده AVR هم استفاده کنیم) اینه که از تراشه ای استفاده کنیم که دارای هر دو بخش کنترلر اترنت و فرستنده/گیرنده PHY باشه. در این سند، از تراشه ENC28J60 تولید شرکت میکروچیپ استفاده می کنیم. میکروکنترلر نیز تراشه STM32f103RBT6 استفاده شده که براحتی میتونه با یه میکروی ARM دیگه جایگزین بشه و با کمی تغییر در کد، می توانیم از هر میکروی دیگه ای استفاده کنیم. فقط کافیه میکروکنترلر ما دارای پورت SPI برای برقراری ارتباط با تراشه ENC28j60 باشه. جهت اطلاع؛ برنامه هم در محیط KEIL نوشته شده که زیاد مهم نیست. هدفمون اینه که برد ما بتونه بدون واسطه و با استفاده از ارتباط اترنت و پروتکل های آن با یک کامپیوتر در ارتباط باشه. لازم به ذکره که یه شبکه واقعی بسیار پیچیده تر از این حالته و تعداد زیادی وسایل میانی مثل سوییچ ها (switch) و روترها (مسیریاب ، Router) و ... داخل شبکه وجود خواهند داشت. در ادامه متوجه میشیم که میشه در حضور سوییچ (مودم های خانگی معمولا سوییچ هم هستند) هم، همین کار رو انجام داد؛ ولی چون در شبکه ما، فقط دو دستگاه (هاست، host) وجود دارد؛ ما از سوییچ استفاده نکردیم و اتصال بین کامپیوتر و برد، مستقیم هست. بعد از بخش ابتدایی و اصلی این سند (در ضمایم) ، مفاهیم شبکه رو دوباره بررسی می کنیم و این قطعات و تجهیزات رو از دید یک مهندس الکترونیک معرفی می کنیم. دوباره یادآوری می کنیم که نوع ارتباط شما با شبکه (مثلا اینکه شما فروشنده تجهیزات هستید؛ طراح و نصاب یک

شبکه اید؛ مدیر یا ادمین شبکه هستید یا برنامه نویس تحت وب یا ...)، در اینکه چه بخش هایی رو باید بیشتر بهش توجه کنید، حایز اهمیته. ما از دید یک مهندس الکترونیک که قصد داره بین برد خودش و تجهیزات دیگه، از ارتباط اترنت و پروتکل هاش استفاده کنه؛ به مبحث شبکه نگاه می کنیم.

احتمالا شنیدید که در هر مدل از طراحی یک شبکه؛ لایه های زیادی وجود دارند. امروزه، تراشه های کنترلر اترنت، قابلیت های گوناگونی دارند و گاهی پروتکل های واقع در چند لایه رو پشتیبانی می کنند. اما یک تراشه کنترلر اترنت، حداقل باید بتونه لایه دوم، یعنی دریافت و ارسال فریم های اصلی اترنت مبتنی بر پروتکل Ethernet ii رو انجام بده (لایه اول توسط بخش PHY اجرا می شود). ما، برای یادگیری مفاهیم اصلی؛ مابقی پروتکل ها در لایه های بالاتر رو در نرم افزار پیاده سازی می کنیم اما اگر تراشه، قابلیت اجرای لایه های بالاتر رو داشته باشه (مثل W5500, W5100)؛ میشه از ظرفیت های تراشه هم استفاده کرد.

- اگر از تراشه یا کد آماده ی دیگه ای استفاده می کنید و شما قادر به دریافت یا ارسال فریم های اصلی اترنت مبتنی بر استاندارد Ethernet ii یا اونطور که بعضا گفته میشه IEEE802.3 هستید؛ می تونید از این بخش (ارتباط با ENC28j60) بگذرید.

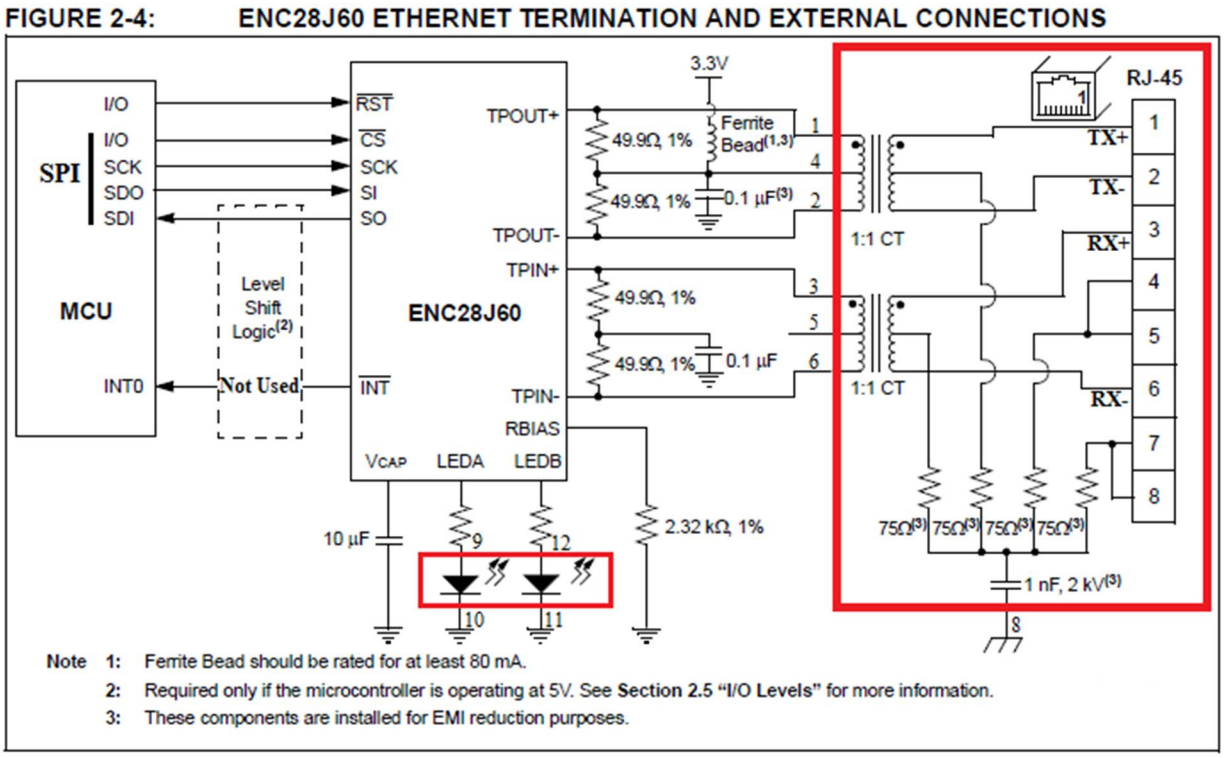
شکل 3 که از دیتاشیت ENC برداشته شده؛ نحوه اتصال این تراشه به میکروکنترلر و همچنین سوکت رو نشون میده. چون میکروکنترلر 3.3 ولتی هست؛ Level Shifter بین میکروکنترلر و ENC28J60 نیاز نیست. از پایه INT هم استفاده نکردیم. پس تنها چهار پین برای ارتباط SPI و یک پین برای Reset استفاده شده و اتصال، مستقیم هست. سوکت های اترنت هم، از سمت بیرون (RJ45 در شکل 3) با چهار پین به کابل شبکه متصل میشن. این چهار پین عبارتند از:

1 = TX+

2 = TX-

3 = RX+

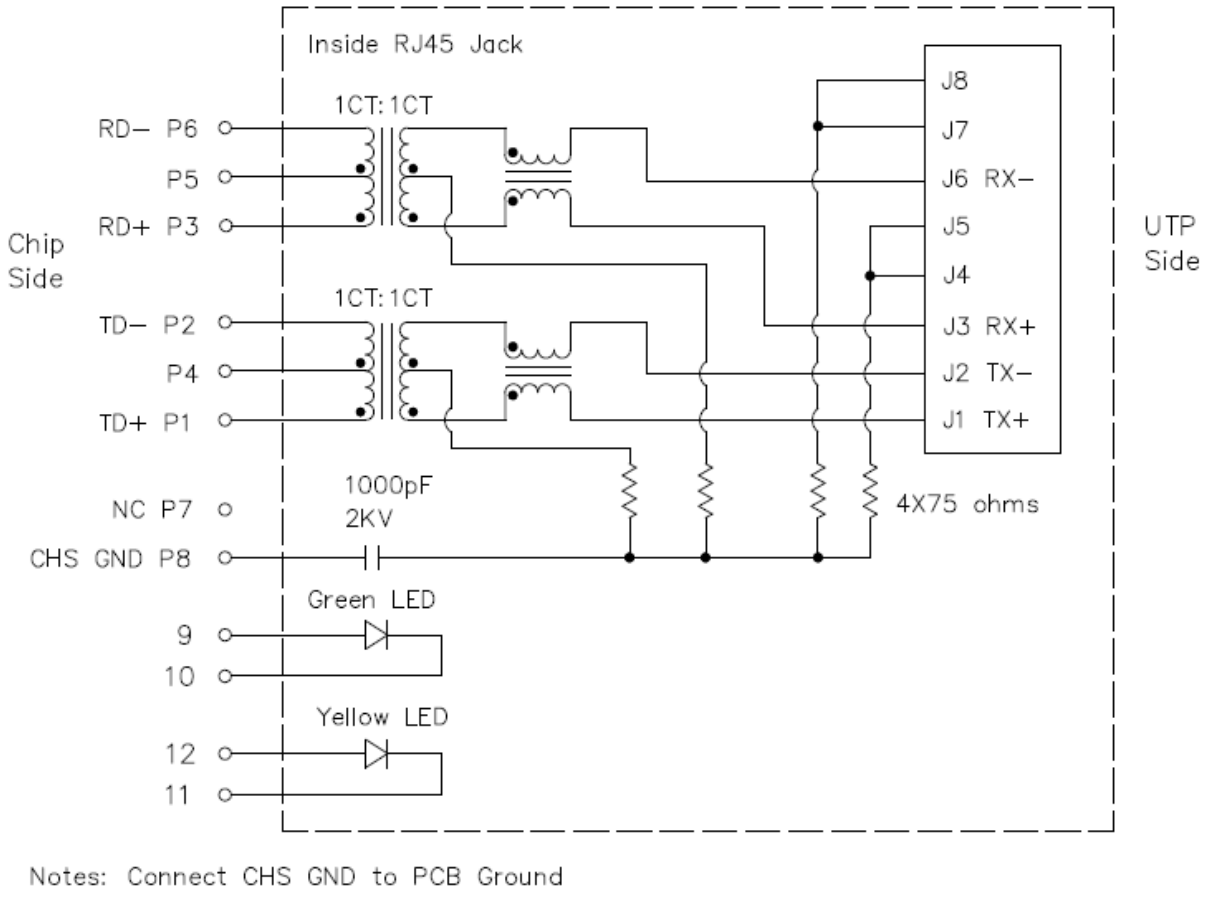
6 = RX-



شکل 3: ارتباط تراشه ENC28J60 با سوکت حاوی فیلتر و پردازنده

توجه داشته باشید؛ سوکت اترنتی ای که ما استفاده می کنیم با شماره HR911105A از شرکت HanRun، دارای فیلتر داخلی و LED های وضعیت هم هست، پس قسمت هایی که داخل کادر در شکل 3 مشخص شده اند؛ داخل سوکت قرار دارند (با رنگ قرمز اگر فایل pdf رو میخونید). شماره های کنار آنها نیز شماره پین در پشت سوکت HR911105A رو نشون میده که در شکل 4 جداگانه ترسیم شده ن.

از پشت سوکت (chip side در شکل 4) هم 12 پین در اختیار ماست که ما از 6 تای اون برای سیگنال های ارسال و دریافت استفاده می کنیم؛ 4 پین برای سیگنال های دیفرانسیلی TX,RX؛ به علاوه 2 پین که سروسط ترانسفورماتورهای فیلترینگ هستند. چهار پین هم سرهای LED های روی سوکت هستند. این LED ها برای نمایش بصری وضعیت ارتباط همچون برقراری لینک (عموما به رنگ سبز) و ارسال/دریافت بیت ها (رنگ زرد) هستند. قابلیت تعریف نوع عملکرد و زمان روشن ماندن آن ها نیز از طریق ثبات های داخلی (register) وجود دارد. شکل 4 از دیتاشیت سوکت HR911105A برداشته شده است.



شکل 4: پین اوت سوکت HR911105A

همانطور که در تصویر 3 مشخص است؛ هر دو پین TX با مقاومت 49.9 اهمی پول آپ (Pull UP) شده اند. سروسط ترانسفورماتور ارسال نیز مستقیما به Vcc=3.3V متصل است. خازن و فریت بید، جهت صافی ولتاژ تغذیه به کار رفته اند. دو پین سمت گیرنده با مقاومت 49.9 اهمی به هم متصل شده و با یک خازن 100n به زمین متصل شده اند اما سروسط ترانسفورماتور گیرنده رها شده است.

- چون برد ما در یک محیط آزمایشگاهی قراره تست بشه؛ نیاز نیست که مقاومت های 49.9 اهمی و همینطور مقاومت 2.32K روی پین Rbias ؛ خیلی با دقت انتخاب بشن و برد با مقادیری مثل 50 اهم یا 2.2k هم کار می کنه. اما چنانچه میخواهید که برد در سرعت های بالا و فواصل زیاد به درستی کار کنه؛ از مقادیر توصیه شده در دیتاشیت استفاده کنید. برای کابل هم از کابل آماده با طول بین یک تا دو متر استفاده کنید.
- اگر از تراشه دیگری استفاده می کنید؛ حتما به نحوه اتصال پین های TX,RX بین تراشه و (فیلتر) سوکت در دیتاشیت تراشه مورد استفاده تون، توجه داشته باشید. چون بسته به نوع تحریک ترانسفورماتورها،

ممکنه متفاوت از شکل 3 باشه؛ مخصوصا در اتصال سروسط ترانسفورماتور گیرنده که در ارتباط با ENC28J60 استفاده نشده است (P5 از سوکت HR911105A).

- فاصله سوکت تا تراشه بایست کمترین مقدار ممکن باشد. در صورتی که بنابر مقتضیات PCB این امکان وجود ندارد؛ مقاومت و خازن ها رو در نزدیکی تراشه قرار بدهید.
- همونطور که در تصاویر مشخص هست؛ سوکت ها در سمت اتصال کابل، 8 پین دارند. در اینجا تنها از دو زوج آنها استفاده می شود؛ یک زوج برای ارسال و یک زوج برای دریافت؛ اما در بعضی از استانداردها برای سرعت های بالاتر، از تمام 8 سیم استفاده می شود. دو زوج برای ارسال و دو زوج برای دریافت.

بدنه تراشه (chasis) و همچنین سر آزاد خازن های 1nF,2KV که برای اتصال دو زمین جداگانه سمت ارسال و سمت دریافت(زمین مدار خودمان) داخل سوکت تعبیه شده (پین 8 سوکت) به زمین مدار متصل می شوند.

کاتد LED ها رو به زمین وصل کنید و آند اون ها رو با یک مقاومت مناسب در حد 1K یا کمتر به ENC وصل کنید(در دیتاشیت گفته برای تنظیم ارتباط به صورت HalfDouplex اینطوری متصل کنیم؛ اما اندازه ای برای مقاومت تعیین نکرده. من 1k گذاشتم اما در نت تا مقدار 180 اهم هم دیدم که استفاده شده؛ اهمیت خاصی هم نداره چون تنها محدود کننده جریان هست و وابسته به جریان مطلوب LED هاست).

با مراجعه به دیتاشیت ENC28j60 (که از این به بعد ممکنه به اختصار ENC نامیده بشه) می بینیم که این تراشه قابلیت برقرار ارتباط با استاندارد 10Base-T رو داره. این استاندارد در سند IEEE802.3i توسط IEEE تنظیم و انتشار پیدا کرده و شامل تمام الزامات و پیشنهاد های این نوع ارتباط هست. حالا این استاندارد چی هست؟ اجازه بدید خیلی خلاصه وار یه مروری بکنیم:

در استاندارد شبکه اترنت، سه سرعت 10,100,1000 Mbps تعریف شده اند. به این سه سرعت به ترتیب Regular, Fast , Gigabit Ethernet میگن. البته در حال حاضر سرعت های بسیار بالاتر هم در حال توسعه و استفاده هست. عدد 10 نشون میده که ENC میتونه بیت ها رو با سرعت 10 مگابیت بر ثانیه منتقل کنه. یعنی برای انتقال یک بایت؛ سرعت ماکزیمم، یک هشتم این عدد یا 1.25MBps هست (در این سند b برای بیت و B برای بایت استفاده شده) اما به خاطر وجود سربار (overload) در پروتکل های مختلف؛ عدد واقعی در ارسال داده های اصلی، بسیار کمتر از این هست. نگران نباشید؛ این سرعت برای بسیاری از کارهای ما کافیه. مثلا وقتی بردی طراحی میکنید که مقدار چند سنسور رو میخونه و با TCP (یکی از پروتکل های لایه چهارم هست که به موقع توضیح داده خواهد شد) به کامپیوتری ارسال میشه و بعد از پردازش در کامپیوتر، پیغام هایی میگیره که باید یک تعداد نمایشگر LED ، رله یا شیر برقی (Electric Valve) رو راه اندازی کنه. در صورتی که این سرعت کم باشه، با فهم اصول شبکه و جایگزینی ENC با یک تراشه سریعتر مثل ENC424J600، مشکل حل میشه.

مورد بعدی در 10Base-T کلمه Base هست. که به معنای خروجی Base band این تراشه است . Base band یا باند پایه در مقابل Broad band یا باند گسترده استفاده میشه و بیانگر اینه که داده های ما گسسته (دیجیتال) شده اند. به عبارت دیگر در هر لحظه از زمان فقط یک بیت از داده روی خط قرار دارد. وقتی شما یه سیگنال مثلا 5 ولتی رو دیجیتال میکنید؛ بازه صفر تا 5 ولت رو به دو (در حالت باینری) یا مثلا 4و 8 و.... حالت جدا تقسیم می کنید و سیگنال دریافت شده رو اینطور تفسیر میکنید که مثلا زیر 0.8 ولت نمایانگر سطح منطقی low یا '0' و ولتاژ بالاتر از 2.5 ولت بیانگر سطح منطقی High یا '1' هست؛ و سیگنال نباید در حد فاصل این دو منطقه یعنی بین 0.8 تا 2.5 قرار بگیره. در این حالت در هر بازه نمونه برداری dt شما یک بیت رو ارسال می کنید؛ پس برای ارسال یک بایت به 8 کلاک نمونه برداری نیاز دارید.

حالا فرض کنید که سطح ولتاژ 5 ولت رو به 4 بخش تقسیم کنید و این چهار بخش رو به چهار حالت ,"00" "11","10","01" تفسیر کنید. لذا در این حالت در هر dt شما دو بیت رو ارسال کردید؛ پس برای ارسال یک بایت، تنها به 4 کلاک نیاز خواهید داشت. معمول اینه که فقط از حالت باینری (دو دویی) استفاده میشه. Base band یعنی این! اما در مقابل Broad band رو داریم و به این معناست که با سیگنال به عنوان یک سیگنال آنالوگ برخورد میشه. اجازه بدید با یک مثال، موضوع رو روشن تر کنیم. فرض کنید برای نمایش یک تصویر بر روی مانیتور، شما حالت باینری استفاده می کنید. برای نمایش هر پیکسل به صورت رنگی با فرمت RGB 8bit ، جهت تعیین رنگ یک پیکسل، شما باید سه بایت یا 24 بیت ارسال کنید. اما در حالت Broad در هر dt شما رنگ یک پیکسل رو مشخص می کنید؛ مثلا سیگنال 2.4v نشاندهنده رنگ سبز (و ولتاژهای نزدیک به آن، مابقی طیف سبز)؛ ولتاژهای نزدیک 4.3 ولت نمایانگر طیف رنگ آبی و 0.8v نمایانگر طیف رنگ قرمز هست.

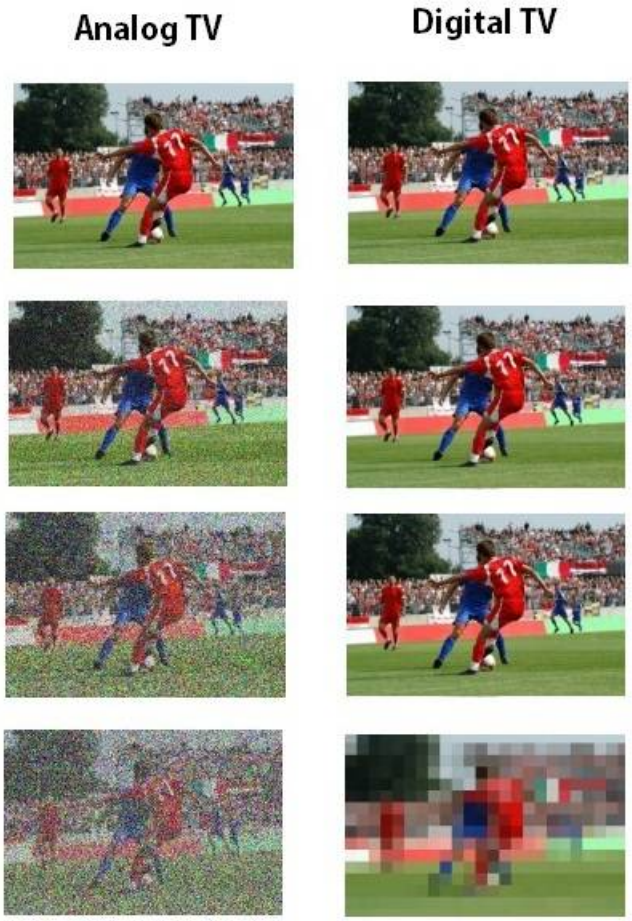
خب واضحه که این حالت بسیار سریعتر از حالت دیجیتال هست؛ اما در عین حال بسیار نسبت به نویز و تداخل ضعیفه و کافیه روی سیگنال اصلی، کمی نویز داشته باشیم یا مبدل آنالوگ به دیجیتال ما کمی خطا داشته باشه تا طیف رنگی بهم بریزه. اما در حالت دیجیتال، نویزی بسیار قوی لازم هست تا بتونه مثلا '1' رو به '0' تغییر بده یا بالعکس. عکس 5 نمونه دو تصویر آنالوگ (Broad band) و دیجیتال (Base band) ؛ همراه با میزان نویز القایی در خط رو نشون میده.

دوباره برگردیم سر وقت 10Base-T ؛ حرف T در اینجا نشون دهنده کابل مورد استفاده است. T یعنی کابل با سیم کشی از نوع زوج های بهم تابیده (Twisted pair) هست. برای فیبر نوری از حرف F استفاده میشه.

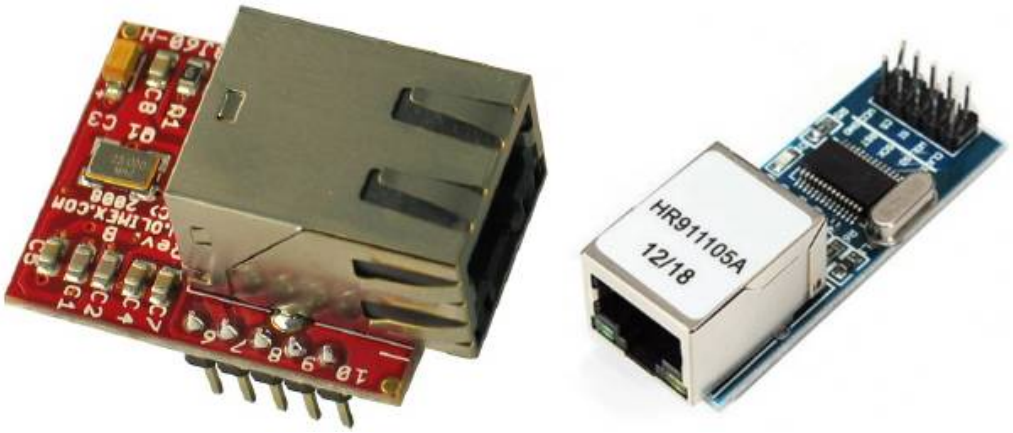
- در شبکه های اترنت اولیه، توپولوژی شبکه به صورت باس بوده و از کابل های کواکسیال (هم محور مثل سیم آنتن تلویزیون) استفاده میشده که به دلیل مشکلات و همینطور هزینه بالای اون در حال حاضر استفاده نمیشه. در حال حاضر شبکه به صورت ستاره پیاده سازی میشه. درون شبکه، هاست ها توسط سوییچ به هم وصلند و برای ارتباط با شبکه های دیگه از قطعه ای به نام روتر استفاده میشه.

برای ارتباط میکروکنترلر با ENC کافیه پین های پورت SPI میکرو به ENC وصل بشه. علاوه بر اون، به پین های $\overline{CS}$ و $\overline{RST}$ تراشه هم نیاز دارید. شما میتونید برای برد خودتون، PCB طراحی کنید یا حالت ساده تر اینه که از ماژول آماده ENC که به صورت های مختلف در بازار موجود هست؛ استفاده کنید. برای برد میکروکنترلر هم از بردهای آموزشی مختلف (مثل Blue pill) میتونید استفاده کنید. در شکل 6 تصویر دو ماژول تراشه ENC28J60 موجود در بازار رو مشاهده می کنید.

توجه داشته باشید که تراشه ENC با تغذیه 3.3v کار میکنه. به همین دلیل، ما برای راحتی کار از میکروهای STM32 خانواده F103 استفاده کردیم. اگر شما از میکروی 5 ولتی مثل AVR یا PIC استفاده میکنید؛ باید بدونید که پین های ورودی ENC قابلیت تحمل ورودی های 5 ولتی رو دارند (5v Tolerant). سیگنال MISO که خرجی تراشه ENC هست؛ رو هم میتونید مستقیم به میکرو وصل کنید یا اگر که میکروی شما 3.3V رو به عنوان سطح منطقی '1' قبول نمیکنه؛ از یک level shifter یا بافر استفاده کنید. (در دیتاشیت مثال داره).



شکل 5 تفاوت تصویر دیجیتال و آنالوگ



شکل 6 نمونه های ماژول آماده ENC28J60 به همراه سوکت و ملحقات

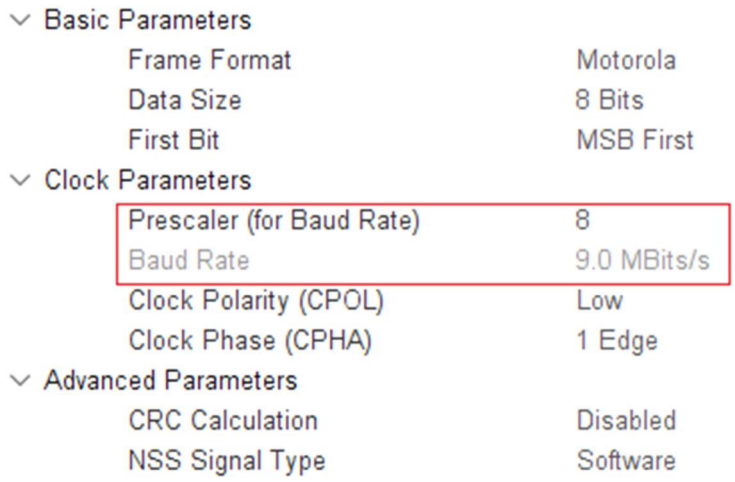
بطور خلاصه، به جز تغذیه و گراند ماژول، پنج پین دیگه رو هم به پین های میکرو متصل میکنیم. سه پین ارتباط SPI یعنی MOSI,MISO,SCK رو به یکی از پورت های SPI میکرو متصل می کنیم؛ علاوه بر این پین های $\overline{CS}$ و $\overline{RST}$ رو هم به دو پین میکرو متصل میکنیم و تنظیمات این دو پین رو به عنوان پورت خروجی انجام میدیم. توجه داشته باشید که هر دو ورودی $\overline{CS}$ و $\overline{RST}$ پایین فعال (Active Low) هستند؛ لذا هنگام انجام تنظیمات اولیه میکروکنترلر، مقدار اولیه این دو پین رو در حالت High یا همون '1' قرار بدید.

- به مابقی پین ها فعلا نیازی نیست. بعد از اتمام این دوره و برای انجام کارهای سریعتر و کاملتر میتونید از پین هایی مثل INT و WOL استفاده کنید. اولی برای راه اندازی وقفه (Interrupt) و دومی مخفف Wakeup On Lan برای استفاده از قابلیت sleep میکرو و ذخیره توان هست.

ایجاد پروژه جدید برای میکروکنترلر:

نرم افزار STM32CubeMX رو باز کنید. یک پروژه جدید ایجاد کنید. یک پورت SPI و دو پین خروجی برای $\overline{CS}$ و $\overline{RST}$ تعریف کنید. با توجه به استفاده از کریستال خارجی 8MHz برای میکروکنترلر؛ ما کلاک داخلی رو، روی 72MHz تنظیم کردیم. تنظیمات پورت SPI مثل شکل 7 باید باشه (داده 8 بیتی به فرمت MOTOROLA, MSB first,prescaler 8, CPOL=Low, CPHA=1Edge). توجه داشته باشید با اینکه پورت SPI چهار حالت برای تنظیم پلاریته و لبه کلاک داره؛ اما تراشه ENC فقط حالت اول آن، یعنی پلاریته Low و لبه اول، رو ساپورت (پشتیبانی) میکنه، پس تنظیم SPI فقط در این حالت باید انجام بشه. بیشترین سرعت کلاک پورت SPI که تراشه ENC قادر به دریافت هست، طبق گفته دیتاشیت 20MHz هست؛ در تنظیم مقدار prescaler به

این موضوع توجه داشته باشید و در ابتدا از انتخاب سرعت های بالا (جهت کاهش احتمال رخداد خطا) پرهیز کنید. پروژه رو ذخیره کنید و پروژه رو برای اجرا روی کامپایلر KEIL ایجاد کنید.



شکل 7 تنظیمات ارتباط SPI

- برای استفاده راحت تر در حین برنامه نویسی، ما دو زوج فایل به محیط برنامه اضافه میکنیم. یک زوج بصورت .c, .h برای راه اندازی تراشه و یک زوج .c, .h برای مابقی پروتکل ها.
- تراشه ENC قابلیت ارتباط DMA رو هم داره که استفاده نکردیم. محاسبه چک سام (check sum) رو هم میتونه به صورت سخت افزاری انجام بده که ما این مورد رو هم در نرم افزار انجام میدیم تا با فرآیند محاسبه اون آشنا بشیم.
- اصطلاحات داده (data) ، فریم (frame) ، استریم (stream) ، بسته (packet) فعلا به یک معنی به کار میروند؛ اما در ادامه متوجه میشیم که این کلمات در لایه های خاص استفاده میشن و برای درک بهتر موضوعات، تعریف شده اند.

خب بریم سراغ فریم اصلی اترنت. همونطور که چند خط بالاتر گفتیم؛ ما هر تراشه ای استفاده کنیم؛ باید قادر به ارسال و دریافت فریم های اصلی اترنت باشه. حالا برای اینکه بدونیم این فریم چیه و از چه چیزهایی تشکیل شده یه مقدمه کوتاه از بعضی اصول شبکه بگیم و کار رو ادامه بدیم.

فرض کنید شما میخواهید یک نامه برای کسی ارسال کنید. شرکت پست باید بدونه این نامه، به چه کسی و در کجا تحویل داده بشه؛ لذا شما به یک آدرس یکتا (unique) نیاز دارید که برای هر گیرنده خاص هست. بالطبع فرستنده هم، آدرسی همانند آدرس گیرنده داره. مثلا یک آدرس در حالت کلی اینطوریه: کشور X شهر یا استان

Y محله یا منطقه Z خیابان K کوچه M ساختمان با پلاک N و.... وقتی هم که ارسال در داخل یک شهر انجام میشه، دیگه نیاز به آدرس دهی کشور یا شهر نیست و مستقیم میریم سراغ محله!
به عنوان یک مثال دیگه فرض کنید تصمیم دارید به شخصی تلفن کنید، طبیعتا شخص مورد نظر دارای شماره تلفنی ست که در شبکه تلفن یکتاست (پس خودتون هم از یک شماره یکتا دارید زنگ میزنید).
اینو تا اینجا داشته باشید تا کمی هم از مدل های شبکه براتون بگم. واضحه که ایجاد شبکه های اولیه محصول یک نیاز بود. فرض کنید دانشگاهی می خواست کامپیوترهاش رو به هم متصل کنه تا راحت تر بتونه اطلاعات رو بین اون ها جابجا کنه یا به خاطر کمبود یکسری تجهیزات مثل پرینتر، نیاز بود که اشخاص از کامپیوترهای مختلف، بتونن فایلی رو برای چاپ روی پرینتر مذکور ارسال کنند (قبلا خود پرینتر جابجا می شد یا برای جابجایی فایل ها، نیاز بود حافظه هارد(Hard Disk) از کامپیوتر خارج بشه یا اطلاعات روی فلاپی دیسک ها ریخته بشه و سختی های خودش رو داشت. ضمنا این کارها در فواصل کم امکان پذیر بود). بعد از ایجاد شبکه ها این نیاز مطرح شد که مدل هایی به صورت استاندارد بوجود بیان که هر شبکه ای بر اساس اون ها طراحی بشه تا طراحی و توصیف شبکه راحت تر باشه. این بود که دو مدل برای شبکه؛ یکی با نام OSI(Open Systems Inteconnection) در 7 لایه و دیگری با نام TCP/IP stack یا Internet Protocol Suite در 5 لایه ایجاد شد. خواهیم دید که این دو مدل بسیار به هم شبیه هستند.

- یک ارتباط رو در ساده ترین حالت میشه به صورت یک مدل، فقط با دولایه هم تعریف کرد. لایه فیزیکی و لایه منطقی. لایه فیزیکی تفسیر سیگنال های دریافتی و تبدیل اونها به صفر و یک های متناظر هست و لایه منطقی هر تعریف دیگه در این ارتباط رو شامل میشه. مثلا یک ارتباط UART با استاندارد RS232 در لایه فیزیکی، تعریف میکنه که 0 یعنی ولتاژ بالاتر از 10v+ و 1 یعنی ولتاژ کمتر از 10v- و لایه منطقی اینگونه تعریف شده: حالت توقف (Idle) در سطح '1' هست. برای شروع ارتباط؛ یک بیت '0' ارسال میشه؛ بعد داده ها(با طول متفاوت) ارسال میشن؛ در صورت نیاز بیت توزان (Parity) و در نهایت یک یا دو بیت '1' به عنوان بیت پایان!

قصد نداریم به یکباره تمام این لایه ها رو توصیف کنیم و روال اینطوریه که تک تک این لایه ها رو، روی هم میچینیم تا هم مدلمون تکمیل بشه و هم مفاهیم و پروتکل ها رو یاد بگیریم.

لایه یک که پایین ترین لایه هست لایه فیزیکی نام داره، این لایه هیچ پردازشی انجام نمیده؛ جز اینکه بایت هایی که از لایه دوم دریافت کرده رو بصورت یک جریانی (Stream) از صفر ها و یک ها به سمت مقابل (گیرنده) بفرسته و یا از سمت مقابل یک استریم رو دریافت کنه و اون ها رو بایت به بایت تحویل لایه دوم بده. شکل 8 نمایش از این عملیات هست. همونطور که متوجه شدید در این لایه برای داده ها از کلمه ی "استریم" استفاده میشه که به جریانی از بیت ها گفته میشه.

شکل 8

- مجددا یادآوری میکنیم: هر برد اکترونیکی؛ برای ارتباط اترنتی، باید حداقل شامل این دولایه باشه.

لایه دوم، تعدادی بایت به لایه اول تحویل میده (اینکه این بایت ها چه اطلاعاتی تو خودشون دارند رو کمی جلوتر میگیم) و لایه اول یعنی لایه فیزیکی، این بایت ها رو به ترتیب دریافت از بیت کم ارزش یعنی LSb ارسال میکنه. لایه دوم اما بایت های یک فریم رو از پر ارزشترین بایت به لایه اول تحویل میده؛ یعنی به صورت MSB (بایت پرارزش اول) . لایه دوم کارش چیه؟ یکسری بایت از لایه سوم میگیره و باید اون ها رو ارسال کنه! اولین سوال اینه که به کجا ارسال کنه و آدرس مقصد چیه؟ پس مقصد نیاز به یک آدرس داره که در کل شبکه یکتا باشه. بالطبع خودش هم آدرسی از همین نوع داره. طبق استاندارد، در شبکه اترنت؛ این نوع آدرس رو بهش میگن MAC Address . مک آدرس از 6 بایت تشکیل شده و معمول اینه که بصورت هگزادسیمال نمایش داده میشه به این صورت:

40-AA-00-10-05-D4 یا 0A:55:4C:12:AF:B3

همونطور که می بینید برای جدسازی بخش های مختلف آدرس از : یا – استفاده میشه. توجه داشته باشید که 0 ها رو هم حتما بنویسید.

خب این آدرس چیه؟ عملا از دو بخش تشکیل شده، به سه بایت سمت چپ که بایت های پر ارزش هستند اصطلاحا OUI(Organization Unique Identifier) گفته میشه. از طرف سازمان IEEE به شرکت های تولید کننده چیپ های کنترلر اترنت، یک شماره سه بایتی اختصاص داده میشه که در ابتدای مک آدرس قرار میگیره؛ مثلا این شناسه برای شرکت میکروچیپ که تولید کننده ENC هست 00:04:A3 هست. نرم افزارهای شنود (sniffer) شبکه مثل wireshark از این شناسه برای تشخیص شرکت سازنده چیپ کنترلر شبکه استفاده میکنند. سه بایت بعدی، شماره سریال هر چیپ هست. این مجموعه 6 بایتی، بایست در شبکه ما یکتا باشه. این شش بایت، معمولا از طرف سازنده تراشه، درون چیپ نوشته میشن و هر چیپ شماره یا مک آدرس یکتایی داره. اگر هم مثل چیپ ENC برای مک آدرس؛ حافظه خالی گذاشتن و شما باید مقدار اون رو تعیین کنید؛ باید مراقب باشید که به هر چیپ یک مک آدرس یکتا (Unique) بدید!

- کم ارزشترین بیت از اولین بایت (در قسمت OUI) که اولین بیت ارسالی فریم هم هست، اگر '1' باشه؛ اصطلاحا این آدرس یک آدرس multicast هست. هنگامیکه آدرس multicast باشه؛ میتوان تعیین کرد که یک یا چند دستگاه در شبکه این فریم رو دریافت و پردازش کنند. به عنوان مثال؛ فریمی که برای استفاده از این حالت تعریف شده؛ فریم "Pause" هست با آدرس مقصد 01-80-c2-00-00-01 که باعث میشه برای لحظاتی دستگاه های موجود در شبکه؛ از ارسال فریم خودداری کنند (به شرطی که این قابلیت در اون ها تعریف شده باشه). این بیت وقتی '0' باشه؛ گفته میشه که آدرس مقصد؛ unicast هست؛ یعنی فقط دستگاهی که آدرس دهی شده؛ این فریم رو پردازش خواهد کرد. لذا آدرس خصوصی تمام دستگاه ها در شبکه (منجمله برد ما) باید در این بیت، شامل '0' باشد.
- ست بودن بیت دوم از پرارزشترین بایت هم نشوندهنده اینه که آدرس توسط شرکت سازنده درون تراشه نوشته شده در حالیکه '0' بودن این بیت به این معنیه که این آدرس توسط استفاده کننده نهایی تعیین شده (مثل وضعیت ما در تعیین آدرس برای ENC). عموما این تعریف برای بیت دوم مورد استفاده قرار نمیگیره و کاربران توجهی به ست بودن یا نبودن آن ندارند.
- یک مک آدرس خاص هم که تمام بیت های اون با '1' پر شده (یعنی به فرم FF:FF:FF:FF:FF:FF هست) رو میگیم مک آدرس عمومی (broadcast). هر گاه آدرس مقصد با این آدرس پر شده باشه؛ همه گیرنده ها در شبکه، این پیام رو دریافت و پردازش خواهند کرد.

خب دوباره برگردیم سراغ لایه دوم شبکه که در مدل OSI بهش Data Link Layer میگن. این لایه علاوه بر 12 بایت شامل مک آدرس ها (6 تا برای گیرنده و 6 تا برای فرستنده)، دو بایت هم برای شناسایی نوع داده هایی که داره ارسال میشه، به ابتدای فریم اضافه میکنه. همچنین برای بررسی صحت دریافت داده ها در سمت گیرنده؛ در انتهای فریم، چهار بایت برای بازبینی اطلاعات قرار میده. این چهار بایت یک عملیات CRC روی بایت های ارسال شده ست. لایه دوم تنها لایه ای هست که علاوه بر هدر (12 بایت مک آدرس ها و 2 بایت نوع فریم)، فوتری از نوع CRC هم به داده ها اضافه میکنه. به این بخش FCS (Frame Check Sequence) گفته میشه.

خب یکم بریم عقب تر، اگه یادتون باشه؛ گفتیم که برای ارتباط تنها دو روج سیم برای TX و RX استفاده شدن! و سیگنال کلاک نداریم! از طرفی، درون شبکه، تجهیزات مختلفی با سرعت های مختلف هستند. این ها چطور با هم سنکرون میشن؟

اینجاست که لایه دوم تعدادی بایت که بهشون میگن preamble هم در ابتدای frame و قبل از ارسال مک آدرس ها میفرسته.

- متوجه شدید که برای لایه اول از اصطلاح stream و برای لایه دوم از frame استفاده شد.

Preamble شامل 7 بایت با مقدار 0x55 و یک بایت هشتم با مقدار 0xD5 که به نام SFD (Start of Frame Delimiter) شناخته میشه. اگر این اعداد رو بصورت باینری بنویسید متوجه میشید که بصورت یک در میان ’0‘ و ’1‘ میشن.

0x55=01010101

0xD5=11010101

(در بعضی مراجع SFD رو هم جزو Preamble به حساب میارن و اونو 8 بایتی در نظر می گیرند) یادمون هست که هر بایت از کم ارزشترین بیت ارسال میشد. پس، ارتباط با ارسال یک بیت با مقدار ’1‘ شروع میشه (کم ارزشترین بیت از پرارزشترین بایت که 0x55 هست) و بعد به ترتیب صفر ها و یک ها ارسال میشن تا آخرین بیت از بایت هشتم که، به جای اینکه ’0‘ باشه دارای مقدار ’1‘ هست. تا اینجا کلاک دو سمت با هم سنکرون شده ن. هنگام ارسال آخرین بیت از بایت هشتم، طرف گیرنده، متوجه شروع ارسال مک آدرس ها میشه. بعد از ارسال فریم هم خط برای زمان مشخصی ساکت میشه که بهش IPG(InterPacket Gap) میگن. در حالت سکوت اصطلاحا خط در حالت Idle (توقف) هست و سیگنالی از نوع ’0‘ یا ’1‘ روی خط نیست. دو وضعیت ’0‘ و ’1‘ و همچنین Idle با توجه به اختلاف سطوح ولتاژ روی پین های TX+,TX- تعریف شده است (توجه داشته باشید که در سرعت 100Mb وضعیت Preamble کمی متفاوت هست نسبت به 10Mhz).

این نوع بسته بندی ؛ استاندارد Ethernet ii یا همون اترنت دو هست. بیاید این اطلاعات رو درون جدول ببینیم.

نام	Preamble	SFD	Destination MAC Address	Source MAC Address	Ether Type/Length	Payload (data)+pad(0x00)	FCS	IPG
تعداد بایت	7	1	6	6	2	46~1500	4	12
مقدار	0x55	0xD5	مک آدرس مقصد	مک آدرس مبدا	نوع/ طول پروتکل	داده دریافتی از لایه سوم	CRC	Idle

در مراجع؛ دو بخش Preamle و SFD رو به همراه بخش IPG، جزو بخش های اصلی فریم در نظر نمی گیرند. در دو بخش DA, SA مک آدرس گیرنده و فرستنده قرار دارد. پرارزش ترین بایت در هر مک آدرس، ابتدا ارسال

می شود. به عنوان مثال در صورتی که مک آدرس مقصد 0A:55:4C:12:AF:B3 باشد؛ ارسال با 0x0A آغاز و با 0xB3 خاتمه می یابد. مک آدرس مبدا نیز به همین منوال بعد از مک ادرس مقصد نوشته و ارسال خواهد شد. یادمون هم هست که گفتیم هر بایت از کم ارزشترین بیت ارسال میشه!

- به این نوع نوشتن، اصطلاحا فرم Big Endian یا فرمت اینترنتی گفته میشه؛ در حالی که در کامپیوترهای مبتنی بر پردازنده های 8086 (همین PC و لپتاپ هایی که در اختیار ماست) و اکثر میکروکنترلرها از فرم Little Endian استفاده می شود. بخواهیم ساده بگیم در فرمت Little Endian بایت با ارزش کمتر در خانه با آدرس کمتر نوشته می شود؛ ولی همانطور که می بینید، در فرمت اینترنتی بایت پرارزش در خانه های ابتدایی نوشته می شود. اگر با این دو فرمت آشنایی ندارید؛ به ضمیمه [8] مراجعه کنید.
- در اسناد موجود در نت گاهی اعداد واقع در شروع فریم رو به صورت بیتی از راست به چپ می نویسند (0x55 رو به صورت "10101010" و 0XD5 رو به صورت "10101011") و خواننده ممکنه فکر کنه که اعداد اصلی 0xAA و 0XAB هستند. حواستون باشه به این مورد!
- وضعیت ارسال بخش آغازین و انتهایی فریم، در سرعت های 100,1000Mb کمی متفاوت هست از سرعت 10Mb ولی از اونجاییکه این بخش رو همواره سخت افزار انجام میده؛ آنچنان اهمیتی از نظر ما نداره. برای دریافت اطلاعات بیشتر می تونید به نت مراجعه کنید.

بعد از مک آدرس ها دو بایت با عنوان Ether type/Length ارسال می شود. با استفاده از این دو بایت؛ گیرنده متوجه می شود که داده ی ارسالی، تحت چه پروتکلی (از لایه 3) ارسال شده یا شامل چه تعداد بایت هست.

سپس داده های اصلی قرار دارند. طبق استاندارد؛ بخش داده ها، حداقل باید دارای 46 بایت باشد. اگر تعداد داده های واقعی کمتر از آن باشد؛ باید در انتهای داده ها، تعدادی بایت با مقدار 0x00 قرار دهیم تا تعداد آن به 46 بایت برسد و به آن padding گویند. در انتهای پکت هم، 4 بایت برای بررسی صحت دریافت قرار دارند که به نام FCS(Frame Check Syquence) شناخته میشه و محاسبات آن با روش CRC (از ابتدای DA تا انتهای داده ها) انجام می شود. این محاسبات عموما توسط سخت افزار انجام میشه و ما ازش عبور می کنیم. تنها نکته ای که در این بخش، بهتره یادمون باشه؛ اینه که در هنگام ارسال CRC ؛ برعکس مابقی بایت ها؛ از پرارزشترین بیت یعنی بیت شماره 31 شروع و با ارسال بیت شماره 0 تمام می شود. در انتهای فریم هم به اندازه 12 بایت (96us در ارتباط 10Mb) روی خط سکوت برقرار می شود (خروجی در حالت Idle قرار میگیرد) که به آن IPG یا IFG گفته می شود(Inter Packet/Frame Gap). IPG و در سرعت 100Mb کمی متفاوت انجام می شود.

اگر بخش های Preamble,SFD,IPG رو در نظر نگیریم؛ تعداد بایت های کل فریم اترنت، حداقل 64 و حداکثر 1518 بایت هست. 12 بایت برای مک آدرس ها؛ 2 بایت برای Ether type/Length و 4 بایت برای CRC. مجموعا 18 بایت از کل فریم، سربار هست لذا حداقل داده ارسالی میشه 64-18=46

از سوی دیگر، حداکثر داده ارسالی هم میشه 1518-18=1500 که برای استفاده در عموم پروتکل های تعریف شده؛ کافیه. چنانچه بیش از این مقدار، نیاز باشه (مثلا برای ارسال یک فایل) داده های اصلی توسط پروتکل های مربوطه شکسته و خرد میشه و بعد ارسال میشن. همینجا چند اصطلاح دیگه رو معرفی کنیم :

"Octet" که در لغتنامه اینترنتی، نام دیگر "بایت" هست!

PDU(Protocol Data Unit) بطور کلی به داده هایی که توسط یک پروتکل ارسال میشه؛ گفته میشه. MTU(Maximum Transmission Unit) حداکثر داده ارسالی بر حسب بایت هست. در استاندارد بعضی از پروتکل ها، این الزام هست که گیرنده ها باید بتونن یک حداقلی از داده رو دریافت کنن. از طرفی حداکثر مقدار ارسالی هم توسط تنظیمات اولیه یا در مرحله مذاکره (Negotiation) یا هندشیک (دست دهی ، Handshake) ممکنه مورد توافق قرار بگیره. همچنین ماکزیمم اندازه داده های هر پیغام توسط فریم اصلی Ethernet ii هم محدود میشه(1500 بایت).

- فریم هایی هم داریم که 4 بایت اضافه دارند. این 4 بایت، بعد از مک آدرس ها و قبل از بخش Ether Type/Length قرار می گیرند. دو بایت اول از این چهار بایت، دارای مقدار ثابت 0x8100 است و به دو بایت بعدی تگ (tag) گفته میشه. لذا اندازه فریم ارسالی حداکثر 1522 بایت هست و حداقل 64 بایت. چون این 4 بایت، جزو هدر به حساب میان؛ حداکثر داده ارسالی همون 1500 و حداقل 42 بایت خواهد بود. به این استاندارد IEEE802.1Q گفته میشه و در ایجاد VLAN ها استفاده میشه. پشتیبانی از اینگونه فریم ها و یا هر گونه فریمی که اندازه بزرگتری دارد، بستگی به قابلیت های تراشه دارد. یک فریم خاص هم داریم به اسم magic packet که توسط شرکت AMD معرفی شده و برای بیرون آوردن یک دستگاه از حالت sleep مورد استفاده قرار میگیره. (همونطور که می بینید؛ اینجا به جای فریم از اصطلاح پکت استفاده شده!) در بخش داده های این فریم (یا پکت)؛ 6 بایت با مقدار 0xFF برای سنکرون شدن و 16 بار مک آدرس گیرنده (بدون اینکه فاصله ای بینشون باشه) قرار داده میشه. در بخش مک آدرس مقصد؛ هم میشه از آدرس عمومی (6 بایت شامل 0xFF) استفاده کرد و هم از مک آدرس دستگاه مورد نظر. و باز یک فریم خاص دیگه که به نام فریم کنترلی (control frame) شناخته میشه؛ در بخش Ether Type دارای عدد 0x8808 هست. در بخش داده ها 2 بایت برای نوع کنترل(opcode) و 2 بایت برای پارامترهای آن (در صورت وجود) به همراه 42 بایت 0x00 برای پدینگ داره. در حال

حاضر تنها یک فریم کنترلی به نام pause تعریف شده با opcode=0x0001 و DA=01-80-c2-00-00-01 که یک آدرس multicast هست و برای توقف موقتی ارسال داده در شبکه استفاده میشه. مقدار توقف در قسمت پارامتر مشخص میشه و هر عدد آن برابر با مدت زمان لازم برای ارسال 512 بیت هست (مثلا اگر در بخش پارامتر، عدد 2 قرار بدیم، به اندازه ارسال 1024 بیت، ارتباط متوقف میشه). برای لغو توقف هم باید دوباره همین فریم با پارامتر 0x0000 ارسال بشه. باز هم تکرار میکنم این اطلاعات صرفا جهت اطلاع هست و نیازی بهشون نداریم.

مهمترین بخش هر فریم، قسمت Ether Type/Length هست. اگر در این قسمت عددی کوچکتر یا مساوی 1500 معادل 0x05DC نوشته شده باشه؛ این عدد؛ اندازه بخش payload یا همون طول داده های ارسالی (Length) رو نشون میده. به این حالت میگن استاندارد IEEE 802.3 ؛ برای اطلاعات بیشتر در مورد نامگذاری و تاریخچه این استاندارد میتونید از لینک زیر استفاده کنید:

https://en.wikipedia.org/wiki/IEEE_802

اعداد 1501–1535 تعریف نشده اند و نباید استفاده بشوند. اما اگر در این قسمت، عددی بزرگتر/مساوی 1536 یا همان 0x0600 قرار داده شده باشه، این عدد نشانگر پروتکلیه که داده ها تحت اون ارسال شده ن. این حالت رو میگیم استاندارد Ethernet ii. چیزی که ما نیاز داریم و از ابتدا هم بارها بهش اشاره کردیم، در واقع این استاندارد هست. در اسناد موجود در اینترنت؛ در موارد زیادی منجمله در دیتاشیت ENC به جای Ethernetii از IEEE802.3 استفاده میشه؛ ولی شما باید متوجه تفاوت این دو حالت باشید. استاندارد مدنظر ما Ethernet ii هست و از IEEE802.3 موقعی میتونید استفاده کنید که بخواهید داده ها رو بدون استفاده از پروتکل های تعریف شده، ارسال کنید و یا خودتون پروتکل شخصی ایجاد کنید مثل magic packet.

- برای دریافت اطلاعات تکمیلی در مورد فریم اترنت به AN1120 “Ethernet Theory of Operation” از شرکت میکروچیپ مراجعه کنید.

تا اینجا چی یاد گرفتیم؟ اگر در قسمت Ethe type عددی کمتر از 1501 نوشته شده باشه؛ داریم از استاندارد IEEE802.3 استفاده میکنیم و در واقع اطلاعات این دو بایت Ether Length هست. این حالت برای ما کارایی نداره. مدنظر ما حالتیه که در قسمت Ether Type عددی بزرگتر از 1535 نوشته شده. این حالت همون استاندارد Ethernet ii (اترنت دو) هست. اعدادی که در این قسمت نوشته میشه از قبل تعریف شده ن و ما برای استفاده از هر پروتکلی مبتنی بر Ethernet ii باید در این قسمت اعداد مشخصی رو قرار بدیم. این اعداد توسط سازمانی بنام IANA(Internet Assigned Numbers Authority) اختصاص داده شده اند. جدول زیر تعدادی از این اعداد رو مشخص کرده.

Ether Type	Protocol
0x0800	**Internet Protocol Version4(Ipv4)**
0x0806	**Address Resolution Protocol(ARP)**
0x8035	Reverse Address Resolution Protocol(RARP)
0x86DD	Internet Protocol Version6(Ipv6)
0x8100	Tagged VLAN
0x88A4	EtherCAT Protocol

تعداد پروتکل های تعریف شده، بسیار بیشتر از این جدوله اما چیزی که برای ما اهمیت دارد؛ دو عدد 0x0800 و 0x0806 (که در جدول پررنگ تر نوشته شده ن) هست که با نام پروتکل های IPv4 و ARP شناخته میشوند. اگه یادتون باشه در ابتدا گفتیم که چنتا پروتکل؛ نیاز حتمی هست که یاد بگیریم تا اصول رو بفهمیم. سه پروتکل Ethernetii ؛ IP و ARP جزو اون ها هستند. پروتکل Ethernet ii که در لایه دوم قرار داره رو تعریف کردیم و دوتای دیگه که جزو لایه سه از مدل OSI هستند رو خواهیم گفت.

- IP یا همون پروتکل اینترنت، اصلیترین پروتکلیه که اینترنت روی اون بنا شده. اگه در جدول دقت کرده باشید دو ورژن برای IP داریم؛ Ipv4 و Ipv6. در سرتاسر این نوشتار ما هرجا نوشتیم IP منظورمون در واقع Ipv4 هست. در ادامه Ipv4 رو توضیح میدیم، چون همچنان مهمترین پروتکل مصرفی در اینترنت و مبحث شبکه هست. در موقع مناسب اگه فراموش نکنیم، علت بوجود اومدن Ipv6 و تفاوتش با Ipv4 رو خواهیم گفت.
- برای بررسی اعداد انحصاری و یا اختصاص یافته در پروتکل های مختلف توسط IANA ؛ به سند مرجع RFC 1700 مراجعه کنید. اگه بخوایم بگیم اسناد RFC شامل چه چیزهایی هستند، باید بگیم این اسناد اصلیترین مرجع تعریف استانداردها، پروتکل ها، الگوریتم ها و... در شبکه اینترنت هست. در ضمیمه [6] مفصل تر در مورد این اسناد صحبت میکنیم. در واقع بعدها اگر نیاز داشتید پروتکلی رو بطور کامل بررسی کنید یا پروتکل هایی که ما بهشون نرسیدیم، رو مطالعه کنید؛ نیاز هست به اسناد RFC مربوطه مراجعه کنید. RFC سرنام (acronym) اصطلاح Request For Comment هست. (خودتون رو زیاد درگیر این اصطلاحات و مخفف ها (abbreviation) نکنید وگرنه بیشتر گیج میشد. کم کم همه شون رو یاد میگیرید؛ بدون حفظ کردن)

خب برگردیم سراغ پروژه مون؛ تا اونجا رفتیم جلو که ارتباط SPI بین میکرو و ENC رو برقرار کردیم. پین های RST,CS تراشه ENC رو هم دادیم به دوتا از پین های میکرو که به صورت خروجی تعریف شده بودن. در نرم افزار STMCubeMX هم تنظیمات اولیه رو انجام دادیم و برای کامپایلر KEIL ازش خروجی گرفتیم. شما با هر

میکرو و کامپایلری میتونید کار کنید، فقط این کارها رو احتمالا باید خودتون هندل کنید و بعضی کدها رو هم بنا به نیاز تغییر بدید. برگردیم سر وقت دیتاشیت ENC28J60 :

اینجا بگیم که ما نمیخوایم دیتاشیت ENC رو به طور کامل ترجمه کنیم. فقط اون قسمت هایی که برامون لازمه رو توضیح میدیم و با میکرو راه میندازیم ، مابقی توانایی ها و قابلیت های ENC مثل خروجی INT برای وقفه؛ میمونه به عهده خودتون که ازش استفاده بکنید یا نه!

یه اعترافی هم اینجا بکنم؛ حوالی سال 90 نیاز داشتم در پروژه ای، داده هایی رو تحت پروتکل TCP برای کامپیوتر ارسال کنم. مداری که بستم و کدی که از نت برداشتم (برای میکروی STM32f107 که خودش کنترلر اترنت داره) جواب نداد و چون اطلاعاتم در مورد شبکه خیلی ناقص بود، عیب یابی هم نتونستم بکنم. مجبور شدم اون برد رو رها کنم و از ماژول آماده ایکه TCP/IP رو بدون درگیر شدن با مفاهیم شبکه به ارتباط سریال USART تبدیل میکرد؛ استفاده کنم. سال ها از این ماژول استفاده میکردم تا اینکه بالاخره دوباره خواستم برگردم و خودم یه تراشه کنترلر اترنت رو راه بندازم. از اونجاییکه میخواستم بعدها بتونم کد رو روی هر میکروی دیگه ای هم پیاده سازی کنم؛ تراشه ENC رو انتخاب کردم؛ طبیعتا کلی کد آماده برای انواع میکرو وجود داشت؛ ولی من دوست داشتم این بار اگه مشکی پیش اومد یا خواستم تغییراتی بدم ، بدونم دارم چکار میکنم. اینه که با جستجو بالاخره تونستم سایتی رو پیدا کنم که همینطور که من دارم الان توضیح میدم؛ مفاهیم رو توضیح داده بود؛ البته بسیار خلاصه تر و کم حجم تر از این که می بینید(در 5 صفحه اینترنتی). به علاوه از دو پروتکل اصلی UDP و TCP هم فقط UDP رو کار کرده بود، ولی طریقه حل مسیله و روش آموزشش، سر نخ (getting started) خوبی بود برای من. لذا با همون روش سعی دارم بطور کامل تر و البته مفصل تری این مسیر رو ادامه بدم و TCP, DHCP, HTTP رو هم با همین روش کار کنم. آدرس اون سایت هست microtechnics.ru کدهای مورد نیازتون رو البته تا پروتکل UDP (بدون DHCP,TCP, HTTP) میتونید از این سایت دانلود کنید؛ همینجا هم تجربه خودم رو بگم که بعد از دانلود و کامپایل برنامه؛ تعداد زیادی خطای نحوی (Syntax Error) داشت که باید اونها رو رفع کنید. مهمترینش تا اونجایی که یادمه؛ فاصله بین اشاره گر به ساختمان، یعنی عملگر <- بود که بین – و < یه فاصله افتاده بود؛ لذا تعداد زیادی خطا ایجاد می کرد. لینک دانلود بخش اصلی نرم افزار بدون TCP, DHCP,HTTP رو در انتهای لینک زیر می تونید پیدا کنید:

https://microtechnics.ru/stm32-i-ethernet-chast-5-transportnyj-uroven-protokol-udp/

خب کجا بودیم؟! ارتباط ENC رو برقرار کردیم. با کدویزارد هسته اولیه نرم افزار رو آماده کردیم و آماده ایم که ENC رو راه بندازیم و باهاش از طریق پورت شبکه به یه کامپیوتر وصل بشیم و اطلاعاتی رو ردوبدل کنیم. ابتدا دو فایل با نام های enc28j60.c و enc28j60.h رو به برنامه تون در محیط KEIL اضافه کنید (امیدوارم حداقل

با برنامه نویسی C آشنا باشید. اگر آشنا نیستید؛ این فایل رو ببندید و اول اون رو یاد بگیرید). در هدر فایل؛ اعلان ها (declaration) رو قرار میدیم و در فایل C، تعریف (definition) متغیرها و توابع رو.

طبق گفته دیتاشیت برای ارتباط SPI باید پین CS تراشه در منطق LOW قرار بگیره. اولین تابعی که مینویسیم تابعی برای کنترل پین CS تراشه ENC یه همچین چیزیه:

```
Static void SetCS(ENC28J60_CS_State state)
{
 HAL_Delay(1);
 HAL_GPIO_WritePin(ENC28J60_CS_PORT,ENC28J60_CS_PIN, (GPIO_PinState)state );
 HAL_Delay(1);
}
```

- استاتیک بودن تابع به کامپایلر میگه که از این تابع فقط در فایل خودش میتونه استفاده کنه (برای کنترل دسترسی به تابع از بیرون فایل و گزارش خطا)

و در فایل هدر (header) مشخص می کنیم پین های CS,RST کجاست. همچین enum مورد نظر برای ENC28J60_CS_State رو هم اعلان می کنیم.

```
#define  ENC28J60_CS_PORT  GPIOA
#define  ENC28J60_CS_PIN    GPIO_PIN_4

#define  ENC28J60_RESET_PORT     GPIOB
#define  ENC28J60_RESET_PIN       GPIO_PIN_0

typedef enum
{
 CS_LOW = 0,
 CS_HIGH = 1,
} ENC28J60_CS_State;
```

- تمام کدها بطور کامل در انتهای همین نوشتار اضافه میشه. همچنین از سایت اعلام شده هم میتونید این فایلها رو دانلود کنید. بطور جداگانه هم در سایت های ایرانی آپلود میشه که بتونید استفاده کنید. فعلا ذهنتون متوجه روش پیاده سازی و یادگیری مفاهیم باشه.

توابع زیر برای ارسال یا دریافت بایت یا بایت هایی از پورت SPI تعریف میشن.

```
Static void WriteBytes(uint8_t* data, uint16_t size)
{
 HAL_StatusTypeDef res = HAL_SPI_Transmit(&hspi1, data, size, ENC28J60_SPI_TIMEOUT);
}
```

```
/*-----------------------------------------------------------------------------*/
static void WriteByte(uint8_t data)
{
  HAL_StatusTypeDef res = HAL_SPI_Transmit(&hspi1, &data, 1, ENC28J60_SPI_TIMEOUT);
}
/*-----------------------------------------------------------------------------*/
static uint8_t ReadByte()
{
  uint8_t txData = 0x00;
  uint8_t rxData = 0x00;
 HAL_StatusTypeDef  res = HAL_SPI_TransmitReceive(&hspi1, &txData, &rxData, 1, ENC28J60_SPI_TIMEOUT);
  return rxData;
}
```

جهت کنترل ENC و برقراری ارتباط با آن مکانیزم ساده ای در نظر گرفته شده؛ بدین طریق که رجیستر(ثبات) های کنترلی در چهار بانک تقسیم شده اند. این چهار بانک شامل رجیسترهای کنترلی در سه گروه هستند که عبارتند از :

- رجیسترهای کنترل
- رجیسترهایی برای خواندن/نوشتن بافر اترنت
- رجیسترهای ارتباط با بخش PHY

نامگذاری رجیسترها (Register ، ثبات) هم شامل قاعده ساده ایست. ثباتهای کنترل اترنت با حرف E ؛ رجیسترهای مرتبط با بخش MAC با حروف MA و رجیسترهای مرتبط با ارتباط MII با حروف MI مشخص شده اند.

تصویر زیر نمایی از این رجیسترهاست.

TABLE 3-1: ENC28J60 CONTROL REGISTER MAP

Bank 0 Address	Name	Bank 1 Address	Name	Bank 2 Address	Name	Bank 3 Address	Name
00h	ERDPTL	00h	EHT0	00h	MACON1	00h	MAADR5
01h	ERDPTH	01h	EHT1	01h	Reserved	01h	MAADR6
02h	EWRPTL	02h	EHT2	02h	MACON3	02h	MAADR3
03h	EWRPTH	03h	EHT3	03h	MACON4	03h	MAADR4
04h	ETXSTL	04h	EHT4	04h	MABBIPG	04h	MAADR1
05h	ETXSTH	05h	EHT5	05h	—	05h	MAADR2
06h	ETXNDL	06h	EHT6	06h	MAIPGL	06h	EBSTSD
07h	ETXNDH	07h	EHT7	07h	MAIPGH	07h	EBSTCON
08h	ERXSTL	08h	EPMM0	08h	MACLCON1	08h	EBSTCSL
09h	ERXSTH	09h	EPMM1	09h	MACLCON2	09h	EBSTCSH
0Ah	ERXNDL	0Ah	EPMM2	0Ah	MAMXFLL	0Ah	MISTAT
0Bh	ERXNDH	0Bh	EPMM3	0Bh	MAMXFLH	0Bh	—
0Ch	ERXRDPTL	0Ch	EPMM4	0Ch	Reserved	0Ch	—
0Dh	ERXRDPTH	0Dh	EPMM5	0Dh	Reserved	0Dh	—
0Eh	ERXWRPTL	0Eh	EPMM6	0Eh	Reserved	0Eh	—
0Fh	ERXWRPTH	0Fh	EPMM7	0Fh	—	0Fh	—
10h	EDMASTL	10h	EPMCSL	10h	Reserved	10h	—
11h	EDMASTH	11h	EPMCSH	11h	Reserved	11h	—
12h	EDMANDL	12h	—	12h	MICMD	12h	EREVID
13h	EDMANDH	13h	—	13h	—	13h	—
14h	EDMADSTL	14h	EPMOL	14h	MIREGADR	14h	—
15h	EDMADSTH	15h	EPMOH	15h	Reserved	15h	ECOCON
16h	EDMACSL	16h	Reserved	16h	MIWRL	16h	Reserved
17h	EDMACSH	17h	Reserved	17h	MIWRH	17h	EFLOCON
18h	—	18h	ERXFCON	18h	MIRDL	18h	EPAUSL
19h	—	19h	EPKTCNT	19h	MIRDH	19h	EPAUSH
1Ah	Reserved	1Ah	Reserved	1Ah	Reserved	1Ah	Reserved
1Bh	EIE	1Bh	EIE	1Bh	EIE	1Bh	EIE
1Ch	EIR	1Ch	EIR	1Ch	EIR	1Ch	EIR
1Dh	ESTAT	1Dh	ESTAT	1Dh	ESTAT	1Dh	ESTAT
1Eh	ECON2	1Eh	ECON2	1Eh	ECON2	1Eh	ECON2
1Fh	ECON1	1Fh	ECON1	1Fh	ECON1	1Fh	ECON1

شکل 9 ثباتهای کنترلی در ENC28J60

بنا به گفته دیتاشیت، برای دسترسی به هر رجیستر، ابتدا باید بانک مورد نظر انتخاب شود. بیت های کنترلی انتخاب بانک، در ثبات ECON1 در آدرس 0x1F قرار گرفته اند. از طرفی چنانچه با دقت به شکل 9 نگاه کنیم متوجه می شویم که در انتهای تمام بانک ها، 5 ثبات با نام مشابه قرار دارند. در واقع این آدرس ها به یک ثبات ارجاع می دهند؛ درنتیجه در هنگام دسترسی به این رجیسترها، انتخاب یا تغییر بانک نیاز نیست؛ به عنوان مثال، چنانچه قصد ارتباط با EIR در آدرس 0x1C را داشته باشیم، نیاز نیست که بانک فعلی را تغییر دهیم و تنها کافیست اطلاعات مورد نیاز را از آدرس 0x1C بخوانیم یا بنویسیم.

برای ارتباط با ENC تنها 7 نوع دستورالعمل تعریف شده است. چهار دستور برای خواندن/نوشتن رجیسترهای کنترلی و حافظه بافر؛ دو دستور برای تغییرات بیتی و یک دستور برای ریست نرم افزاری (جهت ریست سخت افزاری نیز از پین RST استفاده خواهیم کرد).

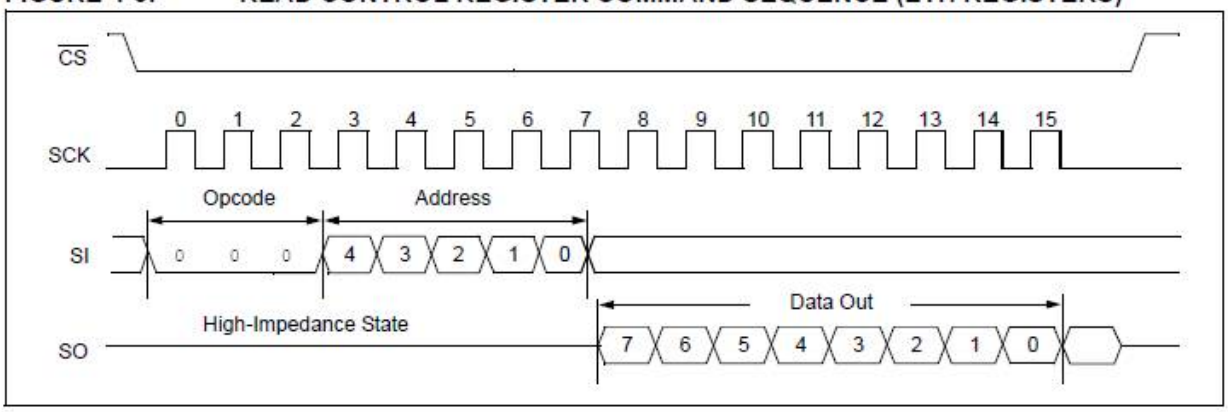
TABLE 4-1: SPI INSTRUCTION SET FOR THE ENC28J60

Instruction Name and Mnemonic	Byte 0								Byte 1 and Following							
	Opcode			Argument					Data							
Read Control Register (RCR)	0	0	0	a	a	a	a	a	N/A							
Read Buffer Memory (RBM)	0	0	1	1	1	0	1	0	N/A							
Write Control Register (WCR)	0	1	0	a	a	a	a	a	d	d	d	d	d	d	d	d
Write Buffer Memory (WBM)	0	1	1	1	1	0	1	0	d	d	d	d	d	d	d	d
Bit Field Set (BFS)	1	0	0	a	a	a	a	a	d	d	d	d	d	d	d	d
Bit Field Clear (BFC)	1	0	1	a	a	a	a	a	d	d	d	d	d	d	d	d
System Reset Command (Soft Reset) (SRC)	1	1	1	1	1	1	1	1	N/A							

Legend: a = control register address, d = data payload.

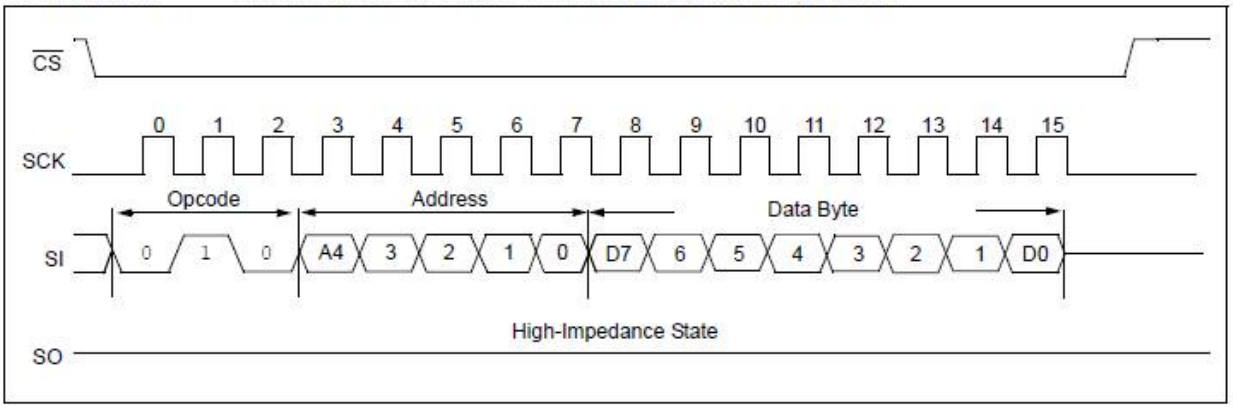
شکل 10 دستورات ارتباطی با ENC

FIGURE 4-3: READ CONTROL REGISTER COMMAND SEQUENCE (ETH REGISTERS)



شکل 11 خواندن ثباتهای کنترلی

FIGURE 4-5: WRITE CONTROL REGISTER COMMAND SEQUENCE



شکل 12 نوشتن در ثباتهای کنترلی

با توجه به نوع دستورها می بینیم که هر دستور از 3 بیت برای نوع دستور (Opcode) و 5 بیت که یا جهت آدرس دهی یا (با مقدار ثابت) جهت دستورات خاص؛ تشکیل شده. تنها تفاوت بین خواندن یا نوشتن در جهت بایت دوم است که یا ما برای ENC از پین MOSI ارسال خواهیم کرد یا از پین MISO از ENC خواهیم خواند.

برای اجرای عملیات خواندن/نوشتن ENC یک مدل ساده برای نرم افزارمون طراحی کردیم به این ترتیب که از یک متغیر یک بایتی برای مشخص کردن رجیستر مد نظرمون استفاده میکینم. ساختار این بایت در جدول زیر نشون داده شده :

7	6	5	4	3	2	1	0
نوع رجیستر	بانک		آدرس				
0= Ethernet 1=MAC,MII	00=Bank 0 01=Bank 1 10=Bank 2 11=Bank 3						

این مدل بصورت زیر در هدر فایل آمده است

```
typedef enum
{
  BANK_0,
```

```
 BANK_1,
 BANK_2,
 BANK_3,
} ENC28J60_RegBank;

typedef enum
{
 ETH_REG,
 MAC_MII_REG,
} ENC28J60_RegType;
```

- اگر به مقادیر داخل enum مقدار ندیم؛ به ترتیب از 0 به صورت صعودی شماره گذاری میشن.

بعلاوه ثابت هایی که با استفاده از #define تعریف شده اند.

```
#define ENC28J60_REG_BANK_OFFSET        5
#define ENC28J60_REG_TYPE_OFFSET        7
#define ENC28J60_REG_BANK_MASK          0x60
#define ENC28J60_REG_TYPE_MASK          0x80
#define ENC28J60_REG_ADDR_MASK          0x1F
#define ENC28J60_BANK_0_BITS    (BANK_0 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_BANK_1_BITS    (BANK_1 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_BANK_2_BITS    (BANK_2 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_BANK_3_BITS    (BANK_3 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_BANK_COMMON_BITS    (BANK_0 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_COMMON_REGS_ADDR           0x1B
#define ENC28J60_ETH_REG_BIT    (ETH_REG << ENC28J60_REG_TYPE_OFFSET)
#define ENC28J60_MAC_MII_REG_BIT        (MAC_MII_REG << ENC28J60_REG_TYPE_OFFSET)
```

در واقع مدل ما در هنگام دسترسی به یک ثبات یه همچین چیزی هست:

Register = Type | Bank | Address

از طرفی نیاز داریم با تعدادی از بیت ها در بعضی از رجیسترها، به صورت مستقیم ارتباط داشته باشیم که اون ها رو هم به طور ثابت در برنامه تعریف میکنیم؛ مثل :

```
#define  ECON1_TXRST_BIT   (1 << 7)
#define  ECON1_RXRST_BIT   (1 << 6)
#define  ECON1_DMAST_BIT   (1 << 5)
#define  ECON1_CSUMEN_BIT  (1 << 4)
#define  ECON1_TXRTS_BIT   (1 << 3)
#define  ECON1_RXEN_BIT    (1 << 2)
#define  ECON1_BSEL1_BIT   (1 << 1)
```

```
#define  ECON1_BSEL0_BIT   (1 << 0)
```

خب بعد از انجام تعاریف (Define) حالا آماده ایم تعدادی تابع برای بررسی نوع رجیسترها، همچنین مشخص شدن بانک و آدرسشون بنویسیم. قبلتر گفتیم که هرگاه بخواهیم به یک ثبات دسترسی داشته باشیم(به جز بایت های مشترک در انتهای بانک ها)؛ اول باید مطمین شیم که در بانک مورد نظر هستیم یا نه!

```
Static  ENC28J60_RegType   getRegType(uint8_t reg)
{
 ENC28J60_RegType  type  =  (ENC28J60_RegType)((reg  &  ENC28J60_REG_TYPE_MASK)  >>
 ENC28J60_REG_TYPE_OFFSET);
 return type;
}
/*---------------------------------------------------------------------------*/
static  ENC28J60_RegBank   getRegBank(uint8_t reg)
{
 ENC28J60_RegBank  bank  =  (ENC28J60_RegBank)((reg  &  ENC28J60_REG_BANK_MASK)  >>
 ENC28J60_REG_BANK_OFFSET);
 return bank;
}
/*---------------------------------------------------------------------------*/
static  uint8_t   getRegAddr(uint8_t reg)
{
 uint8_t addr = (reg & ENC28J60_REG_ADDR_MASK);
 return addr;
}
```

در توابعی که در ادامه پیاده سازی شدن (Implementation) و هنگام دسترسی به هر رجیستری، ابتدا بررسی میکنیم که آیا در بانک مدنظر قرار داریم یا نه؟ و اگر نیاز بود بانک رو تغییر میدیم. اگه یادتون باشه؛ گفتیم بیت های انتخاب بانک در ثبات ECON1 قرار دارند (که خود این ثبات هم در بخش انتهایی و مشترک تمام بانکها هست؛ لذا برای دسترسی به این ثبات، دیگه لازم نیست بانک رو تغییر بدیم).

REGISTER 3-1: ECON1: ETHERNET CONTROL REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TXRST	RXRST	DMAST	CSUMEN	TXRTS	RXEN	BSEL1	BSEL0
bit 7							bit 0

از اونجاییکه ما به دفعات نیاز داریم بدونیم که در کدام بانک هستیم؛ به جای اینکه هر بار محتویات ECON1 رو بخونیم؛ بانک فعلی رو در یک متغیر ذخیره میکنیم. از شکل بالا هم متوجه میشیم که مقدار اولیه بانک بعد از ریست، روی بانک صفر هست، لذا مقدار اولیه متغیرمون رو هم روی Bank0 تنظیم کردیم.

```
Static  ENC28J60_RegBank   curBank = BANK_0;
```

حالا یه تابع مینویسیم که بانک یک رجیستر رو چک کنه؛ اگر بانک این رجیستر متفاوت از بانک فعلی بود؛ بانک رو تغییر میدیم :

```
static void CheckBank(uint8_t reg)
{
  uint8_t regAddr = getRegAddr(reg);
  if (regAddr < ENC28J60_COMMON_REGS_ADDR)
  {
    ENC28J60_RegBank regBank = getRegBank(reg);
    if (curBank != regBank)
    {
      uint8_t econ1Addr = getRegAddr(ECON1);
       // Clear bank bits
      SetCS(CS_LOW);
      WriteCommand(BIT_FIELD_CLEAR, econ1Addr);
      WriteByte(ECON1_BSEL1_BIT | ECON1_BSEL0_BIT);
      SetCS(CS_HIGH);
       // Set bank bits
      SetCS(CS_LOW);
      WriteCommand(BIT_FIELD_SET, econ1Addr);
      WriteByte(regBank);
      SetCS(CS_HIGH);
      curBank = regBank;
    }}}
```

این تابع مقدار یک رجیستر رو به فرمتی که تعریف کردیم دریافت میکنه؛ اول چک میکنه که این رجیستر از رجیسترهای مشترک نباشه، سپس اگر بانک فعلی از بانکی که رجیستر مدنظرمون توش هست؛ متفاوت بود با دستورات بیتی بانک رو به بانک مدنظر تغییر میده و در نهایت مقدار متغیر CurBank هم بروز میشه.

همونطور که در کد تابع ChekBank میبینید از توابعی استفاده شده که هنوز نمیدونیم چی هستن.

حالا بریم سر وقت دستورالعمل ها؛ یادمون هست که کلا 7 نوع دستور بیشتر نداشتیم. مجددا طبق روال برنامه مون؛ اونها رو با یک enum و یک آرایه تعریف میکنیم:

```
typedef enum
{
  READ_CONTROL_REG,
  READ_BUFFER_MEM,
  WRITE_CONTROL_REG,
  WRITE_BUFFER_MEM,
  BIT_FIELD_SET,
  BIT_FIELD_CLEAR,
  SYSTEM_RESET,
  COMMANDS_NUM,
} ENC28J60_Command;

static uint8_t commandOpCodes[COMMANDS_NUM] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x07};
```

و تعریف تابعی برای ارسال یک دستور به ENC به همراه داده مطلوب:

```
static  void  WriteCommand(ENC28J60_Command   command, uint8_t   argData)
{
  uint8_t data = 0;
  data = (commandOpCodes[command] << ENC28J60_OP_CODE_OFFSET) | argData;
  WriteByte(data);
}
```

حالا بر اساس این تابع؛ توابع اصلی رو پیاده سازی میکنیم، بعنوان مثال تابع خواندن یک رجستر کنترلی:

```
static uint8_t ReadControlReg(uint8_t reg)
{
  uint8_t data = 0;
  ENC28J60_RegType regType = getRegType(reg);
  uint8_t regAddr = getRegAddr(reg);
  CheckBank(reg);
  SetCS(CS_LOW);
  WriteCommand(READ_CONTROL_REG, regAddr);

  if (regType == MAC_MII_REG)
  {
    ReadByte();
  }
  data = ReadByte();
  SetCS(CS_HIGH);
  return data;
}
```

همونطور که مشخص هست؛ ابتدا بانک رو چک (و تنظیم) میکنیم؛ بعددستور خواندن رجیستر صادر میشه و بعد رجیستر خونده میشه . البته از دیتاشیت میدونیم که اگر نیاز به خواندن یک رجیستر از نوع MAC/MII هست در ابتدای پروسه خواندن داده اصلی، باید یک بایت dumy (الکی) رو بخونیم! و بعد داده اصلی از ENC خارج میشه؛ تنها نکته این تابع همینه.

توابع اصلی به علاوه توابعی که برای دسترسی به رجیسترهای دوبایتی مثل EWRPRL و EWRPRH نیاز هست در ادامه اومدن. یک تابع هم برای ریست نرم افزاری ENC نوشتیم.

```
Static  void  BitFieldSet(uint8_t  reg, uint8_t  regData)
{
  uint8_t regAddr = getRegAddr(reg);
  CheckBank(reg);

  SetCS(CS_LOW);
  WriteCommand(BIT_FIELD_SET, regAddr);
  WriteByte(regData);
  SetCS(CS_HIGH);
}

/*-----------------------------------------------------------------------------*/
static  void  BitFieldClear(uint8_t  reg, uint8_t  regData)
{
  uint8_t regAddr = getRegAddr(reg);
  CheckBank(reg);
    SetCS(CS_LOW);
  WriteCommand(BIT_FIELD_CLEAR, regAddr);
  WriteByte(regData);
  SetCS(CS_HIGH);
}
/*-----------------------------------------------------------------------------*/
static  uint8_t  ReadControlReg(uint8_t  reg)
{
  uint8_t data = 0;
  ENC28J60_RegType regType = getRegType(reg);
  uint8_t regAddr = getRegAddr(reg);
  CheckBank(reg);

  SetCS(CS_LOW);
  WriteCommand(READ_CONTROL_REG, regAddr);
```

```
  if (regType == MAC_MII_REG)
  {
    ReadByte();
  }
  data = ReadByte();

  SetCS(CS_HIGH);
  return data;
}
/*-----------------------------------------------------------------------*/
static void WriteControlReg(uint8_t reg, uint8_t regData)
{
  uint8_t regAddr = getRegAddr(reg);
  CheckBank(reg);

  SetCS(CS_LOW);
  WriteCommand(WRITE_CONTROL_REG, regAddr);
  WriteByte(regData);
  SetCS(CS_HIGH);
}
/*-----------------------------------------------------------------------*/
static void WriteControlRegPair(uint8_t reg, uint16_t regData)
{
  WriteControlReg(reg, (uint8_t)regData);
  WriteControlReg(reg + 1, (uint8_t)(regData >> 8));
}
/*-----------------------------------------------------------------------*/
static void WriteBufferMem(uint8_t *data, uint16_t size)
{
  SetCS(CS_LOW);
  WriteCommand(WRITE_BUFFER_MEM, ENC28J60_BUF_COMMAND_ARG);
  WriteBytes(data, size);
  SetCS(CS_HIGH);
}
/*-----------------------------------------------------------------------*/
static void ReadBufferMem(uint8_t *data, uint16_t size)
{
  SetCS(CS_LOW);
  WriteCommand(READ_BUFFER_MEM, ENC28J60_BUF_COMMAND_ARG);
```

```
  for (uint16_t I = 0; I < size; i++)
  {
    *data = ReadByte();
    data++;
  }

  SetCS(CS_HIGH);
}
/*----------------------------------------------------------------------------*/
static  void  SystemReset()
{
  SetCS(CS_LOW);
  WriteCommand(SYSTEM_RESET, ENC28J60_RESET_COMMAND_ARG);
  SetCS(CS_HIGH);

  curBank = BANK_0;
  HAL_Delay(100);
}
```

یک نکته دیگه هم اینکه به رجیسترهای بخش PHY دسترسی مستقیم (مثل بقیه رجیسترهایی که تا الان گفتیم) نداریم و طبق گفته دیتاشیت، باید یک روال طی بشه. برای نوشتن در این رجیسترها، باید آدرس رو در رجیستر MIREGADR بنویسم؛ در ادامه داده رو به صورت دوبایتی در ثبات های MIWRL و MIWRH می نویسم و صبر میکنیم تا نوشتن داده تموم بشه (با بررسی بیت MISTAT_BUSY_BIT از ثبات MISTAT)

خب پایه کار ارتباط با ENC گذاشته شده. حالا فقط نیاز هست ENC رو در ابتدای کار با مقادیر مناسب راه اندازی (Initialize) کنیم :

```
void ENC28J60_Init(void)
{
  HAL_Delay(500);
  HAL_GPIO_WritePin(ENC28J60_RESET_PORT, ENC28J60_RESET_PIN, GPIO_PIN_RESET);
  HAL_Delay(50);
  HAL_GPIO_WritePin(ENC28J60_RESET_PORT, ENC28J60_RESET_PIN, GPIO_PIN_SET);
  HAL_Delay(500);
  SystemReset();//software reset
  HAL_Delay(500);

  // Rx/Tx buffers
```

```
  WriteControlRegPair(ERXSTL, ENC28J60_RX_BUF_START);
  WriteControlRegPair(ERXNDL, ENC28J60_RX_BUF_END);
  WriteControlRegPair(ERDPTL, ENC28J60_RX_BUF_START);

  // MAC address
  WriteControlReg(MAADR1, macAddr[0]);
  WriteControlReg(MAADR2, macAddr[1]);
  WriteControlReg(MAADR3, macAddr[2]);
  WriteControlReg(MAADR4, macAddr[3]);
  WriteControlReg(MAADR5, macAddr[4]);
  WriteControlReg(MAADR6, macAddr[5]);

  WriteControlReg(MACON1, MACON1_TXPAUS_BIT | MACON1_RXPAUS_BIT |
MACON1_MARXEN_BIT);

  WriteControlReg(MACON3, MACON3_PADCFG0_BIT | MACON3_TXCRCEN_BIT |
MACON3_FRMLNEN_BIT);

  WriteControlRegPair(MAIPGL, ENC28J60_NBB_PACKET_GAP);
  WriteControlReg(MABBIPG, ENC28J60_BB_PACKET_GAP);

  WriteControlRegPair(MAMXFLL, ENC28J60_FRAME_DATA_MAX);

  // PHY resisters
  WritePhyReg(PHCON2, PHCON2_HDLDIS_BIT);

  ENC28J60_StartReceiving();
 }
```

حالا ENC آماده ست که فریم های اصلی اترنت رو دریافت یا ارسال کنه.

قبل از هر چیز بریم ببینیم، در یک فریم اترنت چی اطلاعاتی قرار داشت. قبلا هم گفتیم که یک فریم اترنت در استاندارد Ethernet ii شامل 7 بایت (octet) ثابت با مقدار 0x55 ؛ یک بایت ثابت با مقدار 0xD5 به عنوان شروع فریم یا SFD (این دو قسمت، جزو بخش اصلی فریم نیستند). سپس 12 بایت شامل مک آدرس مقصد و منبع (گیرنده و فرستنده)؛ دوبایت به عنوان نوع فریم که شامل عددی بزرگتر از 0x0600 هستند و ما با دوتاش کار خواهیم داشت: 0x0800 برای پروتکل Ipv4 و 0x0806 برای پروتکل ARP.

بعد از اون داده های اصلی قرار می گیرند. در ادامه 4 بایت CRC بعنوان پایان فریم یا FSC. همچنین به فاصله 12 بایت روی خط با عنوان IPG فاصله یا سکوت برقرار میشه. IPG هم جزو بخش های اصلی فریم به حساب نمیاد.

اگه خاطرتون باشه گفتیم در استاندارد IEEE 802.3 در دو بایت Ethe Type ، اندازه داده مشخص میشه. لذا ما میتونیم بفهمیم تعداد داده ها چنتاست؛ CRC کجا قرار داره و انتهای فریم کجاست. اما در استاندارد Ethernet ii که ما باهاش سروکار داریم، تعداد داده ها مشخص نیست، پس از کجا متوجه میشیم (یا بهتره بگیم ENC متوجه میشه) که انتهای فریم کجاست؟ جواب اینه : در ارتباط 10Mb از محل یا وقوع 12 بایت توقف (IPG) و در ارتباط 100Mb از وقوع سیگنالی که به نام /T/R/ شناخته میشه. در واقع ENC؛ داده ها رو میخونه تا برسه به IPG؛ سپس این داده ها رو بافر میکنه. البته شروطی هم چک میشه که عموما ما براش تعیین می کنیم؛ مثل بررسی CRC و آدرس صحیح (آیا فریم برای ما ارسال شده یا نه ؟). داده های اضافی مثل preamble,SFD و CRC جدا میشه و قسمت اصلی فریم؛ شامل آدرس های مقصد و منبع، نوع فریم؛ payload و CRC بافر میشه(درون حافظه ذخیره میشه) تا بعد توسط میکروکنترلر اون ها رو بخونیم و پردازش کنیم.

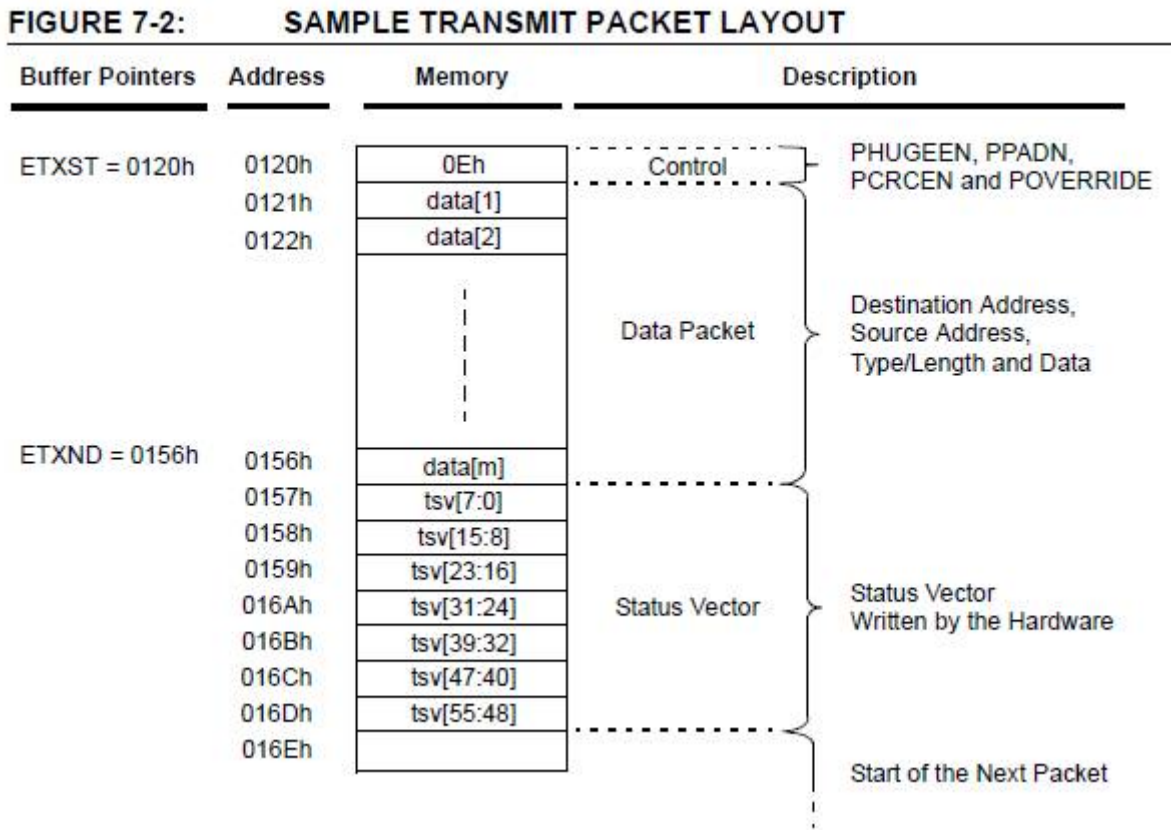
ENC دارای یه حافظه به اندازه 8KB هست که ازش برای دریافت و ارسال فریم ها استفاده میکنه. در واقع این حافظه بین دو بخش ارسال و دریافت تقسیم میشه. ما تعیین میکنیم که چه مقدار حافظه برای بخش ارسال و چه مقدار برای دریافت، قرار داده بشه. به این ترتیب که ابتدا و انتهای حافظه دریافت رو مشخص میکنیم. هرچی باقی بمونه؛ بعنوان حافظه ارسال استفاده میشه.

از اونجایی که حداکثر اندازه فریم ارسالی 1518 بایت هست؛ از طرفی نیاز هست برای یه مقدار اطلاعات اضافه در ابتدا و انتهای فریم ها؛ فضا وجود داشته باشه (دیتاشیت رو ببینید) ما 1536=0x600 بایت برای حافظه ارسال در نظر میگیریم. لذا بقیه حافظه برای دریافت اختصاص داده میشه که مقدار مناسبی هست و چندین فریم، قبل از پردازش توسط ما (میکروکنترلر) میتونه در این حافظه ذخیره بشه. این حافظه به صورت یک صف چرخشی (circular queue) مصرف میشه. بخوام ساده بگم؛ پکت ها به ترتیب دریافت؛ داخل حافظه (البته با تمهیداتی که داخل دیتاشیت گفته شده) ذخیره میشوند. اگر به انتهای حافظه برسیم؛ برمیگردیم از ابتدای حافظه شروع میکنیم. فقط در برنامه مون (داخل میکروکنترلر) باید حواسمون باشه که تعداد فریم های رسیده؛ اونقدر زیاد نشه که حافظه ای برای دریافت فریم های جدید باقی نمونه. اگر نمیدونید صف چرخشی چیه به کتاب های "ساختمان داده ها" رجوع کنید.

- یادم نیست که این نکته رو گفتم یا نه! بهرحال؛ چنانچه اندازه فریم کمتر از 64 بایت باشه؛ طبق استاندارد، باید انتهای داده ها رو به مقدار 0 پر کنیم تا اندازه فریم به 64 بایت برسه. به این کار padding میگن.

هر چند padding طبق گفته استاندارد، الزامی هست؛ اما در عمل پکت هایی دیدم که padding توش انجام نشده و در واقع استاندارد رعایت نشده. به عنوان مثال، در پکت های هندشیک ارسالی از طرف نرم افزار LabVIEW که اگه یادم موند در موقع خودش نشون میدم. این فریم ها رو میشه ارسال کرد اما ممکنه توسط اجزای میانی شبکه مثل روترها دور ریخته بشه(drop). یکی از مواردی که باید برای ENC تنظیم بکنیم، همین padding هست.

بعد از تعیین تکلیف وضعیت حافظه بافر؛ بریم سر وقت ارسال فریم. هنگام ارسال فریم؛ یک بایت کنترلی در ابتدای فریم و هفت بایت تعیین وضعیت در انتهای فریم، اضافه میشه؛ بعلاوه برای CRC هم با وجود اینکه توسط ENC محاسبه میشه، باید 4 بایت فضا در نظر بگیریم. یک فریم ارسالی ظاهری شبیه به شکل 13 خواهد داشت.



شکل 13 ساختار بافر ارسال

تعریف بایت کنترلی رو هم که از دیتاشیت ENC برداشتیم در شکل 14میبینید:

FIGURE 7-1: FORMAT FOR PER PACKET CONTROL BYTES

—	—	—	—	PHUGEEN	PPADEN	PCRCEN	POVERRIDE
bit 7							bit 0

bit 7-4 **Unused**

bit 3 **PHUGEEN:** Per Packet Huge Frame Enable bit

When POVERRIDE = 1:

1 = The packet will be transmitted in whole

0 = The MAC will transmit up to the number of bytes specified by MAMXFL. If the packet is larger than MAMXFL, it will be aborted after MAMXFL is reached.

When POVERRIDE = 0:

This bit is ignored.

bit 2 **PPADEN:** Per Packet Padding Enable bit

When POVERRIDE = 1:

1 = The packet will be zero padded to 60 bytes if it is less than 60 bytes

0 = The packet will be transmitted without adding any padding bytes

When POVERRIDE = 0:

This bit is ignored.

bit 1 **PCRCEN:** Per Packet CRC Enable bit

When POVERRIDE = 1:

1 = A valid CRC will be calculated and attached to the frame

0 = No CRC will be appended. The last 4 bytes of the frame will be checked for validity as a CRC.

When POVERRIDE = 0:

This bit is ignored.

bit 0 **POVERRIDE:** Per Packet Override bit

1 = The values of PCRCEN, PPADEN and PHUGEEN will override the configuration defined by MACON3

0 = The values in MACON3 will be used to determine how the packet will be transmitted

شکل 14 بایت کنترلی ارسال

- مواقعی وجود دارد که ارسال فریم های بزرگتر از 1500 بایت داده، به یک ارتباط سریعتر، کمک میکنه. بعنوان مثال فرض کنید در حال ارتباط با یک دوربین با رزولوشن و FPS(Frame Per Second) بالایی هستید(که عموما با سرعت گیگابیت کار میکنند). در این حالت نیاز دارید که فریم هایی با اندازه بزرگ ارسال کنید؛ هرچند که خارج از محدوده استاندارد هست. در این حالت به فریم های بزرگتر از 1518 بایت اصطلاحا Huge یا Long گفته میشه. (جهت اطلاع) در گیگابیت اترنت هم به فریم های بزرگتر از 9000 بایت Jumbo میگن. به فریم های بزرگتر از 6000 بایت Giant میگن. همچنین در

حالت کلی به فریم های کوچکتر از 64 بایت هم Runt گفته میشه. خودتون رو درگیر این اصطلاحات نکنید، این اطلاعات فعلا فقط جهت اطلاع هست.

در مورد padding هم توضیح دادیم و اما بیت PCRCEN؛ اگر خودتون میخواید که CRC رو حساب کنید؛ کافیه اون رو در انتهای فریم اضافه کنید و به ENC بگید که اینکار توسط شما انجام شده و 4 بایت انتهایی در بخش داده ها؛ CRC هست. اما اگر قراره ENC این کار رو انجام بده؛ با این بیت به ENC اجازه یا دستور محاسبه رو میدیم، CRC محاسبه شده در 4 بایت انتهایی قرار میگیره و ما فقط باید به اندازه 4 بایت براش جا در نظر بگیریم (ما از این روش استفاده کردیم)

در بیت POVERRIDE هم میگیم که آیا از تنظیمات داخل رجیستر MACON3 استفاده بشه یا از بیت های معرفی شده که ما از این وضعیت استفاده میکنیم. کد مورد نظر داخل تابع Initialize بصورت زیر نوشته شده:

```
WriteControlReg(MACON3, MACON3_PADCFG0_BIT | MACON3_TXCRCEN_BIT | MACON3_FRMLNEN_BIT);
```

4 ثبات 16 بیتی با نام های زیر، برای تعیین ابتدا و انتهای بافرهای ارسال/دریافت در بخش رجیسترهای کنترلی ENC موجود هست:

- *ETXSTL / ETXSTH* – address of the beginning of the buffer for transmission
- *ETXNDL / ETXNDH* – address of the end of the buffer for transmission
- *ERXSTL / ERXSTH* – address of the beginning of the receive buffer
- *ERXNDL / ERXNDH* – address of the end of the receive buffer

دو ثبات آخر برای تعیین محدوده بافر دریافت استفاده میشن و مابقی حافظه بطور پیش فرض، محدوده بافر ارسال (تعریف شده در دو ثبات 16 بیتی اول) رو مشخص میکنند.

- توجه داشته باشید که ENC هنگام ارسال؛ مقادیر ثبات های 16 بیتی ETXST و ETXND رو جهت تداخل با محدوده بافر گیرنده چک نمیکنه و این وظیفه میکروکنترلر (ما) هست که مراقب این خطا باشه.

چون به اندازه تقریبی یک فریم در حافظه ارسالی فضا اختصاص دادیم؛ پس در هر لحظه از زمان؛ فقط یک فریم در حال ارسال خواهد بود. از طرفی قرار هست این برنامه روی میکروکنترلر پیاده سازی بشه و از اونجاییکه یکی از اصلیترین محدودیت ها در میکروکنترلرها؛ کم بودن میزان حافظه دیتا (RAM) در اونهاست؛ پس ما فقط یک بخش از حافظه RAM رو برای دریافت ، پردازش و درصورت نیاز ارسال پاسخ (با همون حافظه) درون میکروکنترلر اختصاص میدیم. در واقع صبر میکنیم تا یک فریم برسه؛ اون رو بررسی و پردازش میکنیم و چنانچه نیاز به پاسخ

داشته باشه؛ داده ها رو در همون قسمت از RAM میکروکنترلر مینویسیم و بعد ارسال میکنیم به ENC تا روی خط ارسال بشه.

برای ارسال یک فریم، این عملیات انجام میشه:

- با بررسی بیت TXRTS از ثبات ECON1 صبر میکنیم تا ارسال فریم قبلی به اتمام برسه.
- آدرس صفر (و یا هر مقدار دیگه ای که اطمینان داریم بعنوان آدرس شروع فریم ارسال کافیه) رو در ثبات EWRPT مینویسم.
- ابتدا بایت کنترلی (شکل 14) رو با مقدار 0x00 مینویسیم و در ادامه داده های خودمون رو به بافر منتقل میکنیم.
- مقادیر مناسب رو با توجه به آدرس شروع و اندازه فریم در ثباتهای ETXST و ETXND مینویسیم.
- برای شروع عملیات ارسال بیت ETXRTS از ثبات ECON1 رو ست میکنیم.

```
Void ENC28J60_TransmitFrame(uint8_t *data, uint16_t size)
{
 while((ReadControlReg(ECON1) & ECON1_TXRTS_BIT) != 0)
 {
  if((ReadControlReg(EIR) & EIR_TXERIF_BIT) != 0)
  {
   BitFieldSet(ECON1, ECON1_TXRST_BIT);
   BitFieldClear(ECON1, ECON1_TXRST_BIT);
  }
 }

 WriteControlRegPair(EWRPTL, ENC28J60_TX_BUF_START);

 uint8_t  controlByte = 0x00;
 WriteBufferMem(&controlByte, 1);
 WriteBufferMem(data, size);

 WriteControlRegPair(ETXSTL, ENC28J60_TX_BUF_START);
 WriteControlRegPair(ETXNDL, ENC28J60_TX_BUF_START + size);

 BitFieldSet(ECON1, ECON1_TXRTS_BIT);
}
```

این نکته هم قابل توجه هست که چنانچه هنگام ارسال پکت قبلی خطایی رخ داده باشه، سخت افزار نمیتونه بیت TXRTS رو پاک کنه؛ لذا ما همواره بیت TXERIF از ثبات EIR رو چک میکنیم تا اگر خطایی رخ داده باشه، طبق

گفته دیتاشیت، یکبار بیت TXRST رو ست و ریست کنیم تا قادر به ارسال پکت جدید باشیم. لازم به ذکره که با بررسی بایت های کنترلی در انتهای فریم قبل میشه به نوع خطا پی ببریم و روال تصحیح اون مثل ارسال دوباره فریم قبلی رو انجام بدیم. تفسیر مقادیر بایت های کنترلی TSV در انتهای فریم ارسال در شکل 15 نمایش داده شده است. دو بایت ابتدایی نشان دهنده تعداد بایت های ارسالی ست.

TABLE 7-1: TRANSMIT STATUS VECTORS

Bit	Field	Description
55-52	Zero	0
51	Transmit VLAN Tagged Frame	Frame's length/type field contained 8100h which is the VLAN protocol identifier.
50	Backpressure Applied	Carrier sense method backpressure was previously applied.
49	Transmit Pause Control Frame	The frame transmitted was a control frame with a valid pause opcode.
48	Transmit Control Frame	The frame transmitted was a control frame.
47-32	Total Bytes Transmitted on Wire	Total bytes transmitted on the wire for the current packet, including all bytes from collided attempts.
31	Transmit Underrun	Reserved. This bit will always be '0'.
30	Transmit Giant	Byte count for frame was greater than MAMXFL.
29	Transmit Late Collision	Collision occurred beyond the collision window (MACLCON2).
28	Transmit Excessive Collision	Packet was aborted after the number of collisions exceeded the retransmission maximum (MACLCON1).
27	Transmit Excessive Defer	Packet was deferred in excess of 24,287 bit times (2.4287 ms).
26	Transmit Packet Defer	Packet was deferred for at least one attempt but less than an excessive defer.
25	Transmit Broadcast	Packet's destination address was a Broadcast address.
24	Transmit Multicast	Packet's destination address was a Multicast address.
23	Transmit Done	Transmission of the packet was completed.
22	Transmit Length Out of Range	Indicates that frame type/length field was larger than 1500 bytes (type field).
21	Transmit Length Check Error	Indicates that frame length field value in the packet does not match the actual data byte length and is not a type field. MACON3.FRMLNEN must be set to get this error.
20	Transmit CRC Error	The attached CRC in the packet did not match the internally generated CRC.
19-16	Transmit Collision Count	Number of collisions the current packet incurred during transmission attempts. It applies to successfully transmitted packets and as such, will not show the possible maximum count of 16 collisions.
15-0	Transmit Byte Count	Total bytes in frame not counting collided bytes.

شکل 15 بایتهای کنترل وضعیت ارسال

و اما دریافت فریم :

ما قبلا محدوده بافر دریافت رو مشخص کرده ایم، همچنین باید بگیم که ENC قابلیت فیلتر فریم های دریافتی رو با استفاده از ثبات ERXFCON رو در اختیار ما قرار داده:

REGISTER 8-1: ERXFCON: ETHERNET RECEIVE FILTER CONTROL REGISTER

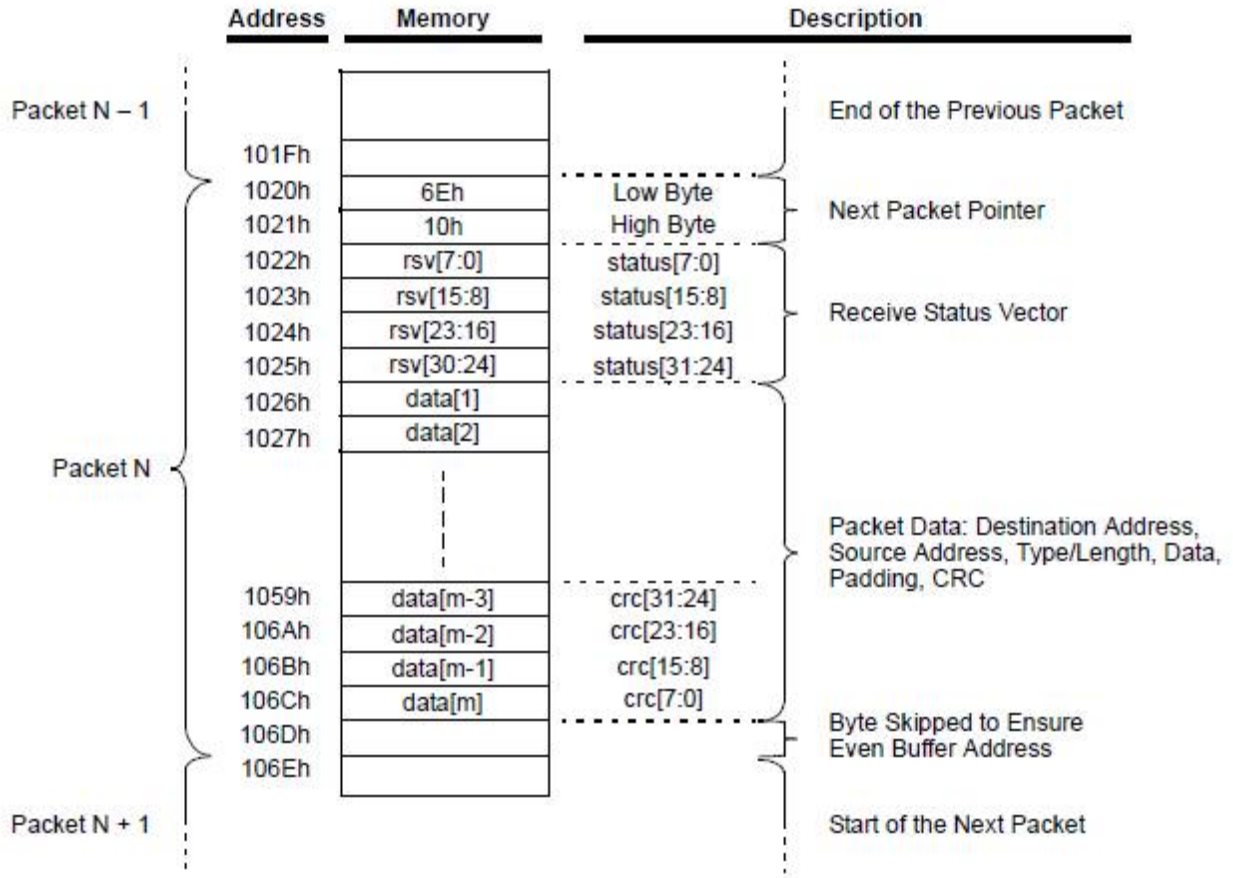
R/W-1	R/W-0	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1
UCEN	ANDOR	CRCEN	PMEN	MPEN	HTEN	MCEN	BCEN
bit 7							bit 0

این فیلترها قابل ترکیب منطقی با هم هستند. برای مطالعه کامل این فیلترها به دیتاشیت مراجعه کنید. بعد از تنظیمات اولیه (از جمله فیلترها) برای فعال سازی بخش دریافت کافیه بیت RXEN از ثبات ECON1 ست بشه که اون رو هم بصورت یه تابع پیاده سازی کردیم.

```
Void ENC28J60_StartReceiving()
{
  BitFieldSet(ECON1, ECON1_RXEN_BIT);
}
```

فریم های دریافتی به صورت شکل 16 در بافر دریافت قرار میگیرند. ابتدا دو بایت برای اشاره به آدرس شروع فریم دریافتی بعدی، 4 بایت برای بررسی وضعیت فریم، سپس داده های دریافتی فریم فعلی شامل مک آدرس ها و دو بایت نوع اترنت (ترکیب فریم استاندارد)؛ داده های اصلی و در انتها نیز 4 بایت CRC قرار می گیرند. چنانچه تعداد کل بایت ها فرد باشد؛ یک بایت در انتها خالی گذاشته میشود تا شروع فریم بعدی روی آدرس زوج باشد. اطلاعاتیکه درون 4 بایت وضعیت دریافت قرار می گیرند، به صورت شکل 17 تفسیر میشوند. همانطور که در شکل 17 مشخص شده است، دو بایت کم ارزش نشاندهنده تعداد بایت های دریافتی ست.

FIGURE 7-3: SAMPLE RECEIVE PACKET LAYOUT



شکل 16 نحوه ثبت فریم های دریافتی در بافر

TABLE 7-3: RECEIVE STATUS VECTORS

Bit	Field	Description
31	Zero	0
30	Receive VLAN Type Detected	Current frame was recognized as a VLAN tagged frame.
29	Receive Unknown Opcode	Current frame was recognized as a control frame but it contained an unknown opcode.
28	Receive Pause Control Frame	Current frame was recognized as a control frame containing a valid pause frame opcode and a valid destination address.
27	Receive Control Frame	Current frame was recognized as a control frame for having a valid type/length designating it as a control frame.
26	Dribble Nibble	Indicates that after the end of this packet, an additional 1 to 7 bits were received. The extra bits were thrown away.
25	Receive Broadcast Packet	Indicates packet received had a valid Broadcast address.
24	Receive Multicast Packet	Indicates packet received had a valid Multicast address.
23	Received Ok	Indicates that at the packet had a valid CRC and no symbol errors.
22	Length Out of Range	Indicates that frame type/length field was larger than 1500 bytes (type field).
21	Length Check Error	Indicates that frame length field value in the packet does not match the actual data byte length and specifies a valid length.
20	CRC Error	Indicates that frame CRC field value does not match the CRC calculated by the MAC.
19	Reserved	
18	Carrier Event Previously Seen	Indicates that at some time since the last receive, a carrier event was detected. The carrier event is not associated with this packet. A carrier event is activity on the receive channel that does not result in a packet receive attempt being made.
17	Reserved	
16	Long Event/Drop Event	Indicates a packet over 50,000 bit times occurred or that a packet was dropped since the last receive.
15-0	Received Byte Count	Indicates length of the received frame. This includes the destination address, source address, type/length, data, padding and CRC fields. This field is stored in little-endian format.

شکل 17 ثبات وضعیت فریم دریافتی

همونطور که قبلا گفتیم هسته برنامه ما (با فرض کمبود حافظه RAM) اینطور عمل میکنه که از همون حافظه اختصاص یافته برای دریافت فریم ها (درون میکروکنترلر و نه ENC) برای تهیه و ارسال پاسخ ارسال میشه؛ به همین دلیل هست که ما نوع ارتباط رو HalfDouplex در نظر گرفتیم. یک فریم رو دیافت و پردازش می کنیم؛ و چنانچه نیاز به پاسخ باشه؛ پاسخ رو در همون حافظه می چینیم و بعد به بافر ارسال ENC منتقل و دستور ارسال فریم رو صادر میکنیم.

- واضحه که این نوع کدنویسی درست نیست؛ زیرا موارد زیادی هست که بدون دریافت فریمی، نیاز هست که اطلاعاتی ارسال بشه، اما همونطور که گفتیم این نوشته برای این عرضه شده که ما با مفاهیم اصلی آشنا بشیم؛ طوریکه خودمون بتونیم یک سخت افزار (بهمراه نرم افزار) کامل برای استفاده در شبکه داشته باشیم.

بعد از دریافت هر فریم، اونها رو درون یک ساختار (structure) قرار میدیم. تعریف این استراکچر در زبان C همچین چیزیه:

```
typedef  struct  ENC28J60_Frame
{
 uint16_t  nextPtr;
 uint16_t  length;
 uint16_t  status;
 uint8_t  data[ENC28J60_FRAME_DATA_MAX];
 uint32_t  48hecksum;
} ENC28J60_Frame;
```

و تابع دریافت هم به صورت زیر هست:

```
uint16_t  ENC28J60_ReceiveFrame(ENC28J60_Frame*  frame)
{
 uint16_t dataSize = 0;
 uint8_t packetsNum = ReadControlReg(EPKTCNT);

 if (packetsNum > 0)
 {
  WriteControlRegPair(ERDPTL, curPtr);

  ReadBufferMem((uint8_t*)frame, ENC28J60_HEADER_SIZE);

  curPtr = frame->nextPtr;

  if ((frame->status & ENC28J60_FRAME_RX_OK_MASK) != 0)
  {
   dataSize = frame->length – ENC28J60_CRC_SIZE;

   if (dataSize > ENC28J60_FRAME_DATA_MAX)
   {
    dataSize = ENC28J60_FRAME_DATA_MAX;
   }

   ReadBufferMem((uint8_t*)&(frame->data[0]), dataSize);
   ReadBufferMem((uint8_t*)&(frame->48hecksum), ENC28J60_CRC_SIZE);
  }

  uint16_t nextPtr = frame->nextPtr – 1;
```

```
    if (nextPtr > ENC28J60_RX_BUF_END)
    {
      nextPtr = ENC28J60_RX_BUF_END;
    }

    WriteControlRegPair(ERXRDPTL, nextPtr);
    BitFieldSet(ECON2, ECON2_PKTDEC_BIT);
  }

  return dataSize;
}
```

ابتدا توسط رجیستر EPKTCNT بررسی می کنیم که آیا فریمی برای خواندن درون ENC هست یا نه؟ چنانچه شمارنده تعداد فریم ها، بزرگتر از صفر باشه؛ با استفاده از اشاره گر به پکت فعلی داده ها، کارهای زیر رو انجام میدیم:

در قدم اول 6 بایت شامل 2 بایت اشاره گر به فریم بعدی و 4 بایت وضعیت رو میخونیم. یادمون هم نمیره اشاره گر به فریم بعدی رو هم ذخیره کنیم که بتونیم فریم بعدی رو بخونیم. اگر فریم دریافت شده، بدون خطا باشه (بیت 23 از بایت های وضعیت رو ببینید) با توجه به تعداد بایت های دریافتی در این فریم (دو بایت اول در بایت های وضعیت)، بایت های اصلی رو میخونیم.

کار ما اینجا با ENC تموم میشه. این نکته بسیار مهم که قبلا هم اشاره کردیم؛ مجدد یادآوری میکنیم. شما هر مداری با هر نوع سخت افزار و هر کدی داشته باشید؛ در نهایت باید دو تابع داشته باشید که با اون ها بتونید فریم های موجود در شبکه رو بخونید و یا فریم های خودتون رو ارسال کنید، مابقی اجرای (Implementation) شبکه در اختیار نرم افزار میکروکنترلر شماست.

خب بریم سراغ مدل OSI یا حالت ساده تر اون مدل TCP/IP (ممکنه به این مدل Internet protocol Suite و یا TCP stack هم گفته بشه). پروتکل های شبکه، لایه روی لایه چیده شدن. در سمت فرستنده، اولین لایه ای که تولید کننده داده هست؛ در سمت گیرنده آخرین لایه ای خواهد بود که داده نهایی رو دریافت میکنه. بهمین دلیل بهش stack (به فارسی، پشته) میگن.

ما تا اینجا با دو لایه پایینی آشنا شدیم و با استفاده از ENC اون ها رو بوجود آوردیم.

پایین ترین لایه یا لایه اول رو به اسم لایه فیزیکال میشناسیم و لایه دوم که در اینجا فریم هایی به فرمت Ethenet ii رو ارسال/دریافت میکنه؛ در مدل OSI بهش میگن لایه ارتباط داده (Data Lick Layer) درحالیکه در مدل TCP/IP بهش فقط میگن لایه ارتباط (Link Layer) .

خب تا اینجا ما یک فریم دریافت کردیم ؛ یادمون هست که بعد از مک آدرس ها، دو بایت با اسم Ether Type وجود داشت که گفتیم اعداد داخل اون بزرگتر از 0x0600 خواهند بود و در جدولی، تعدادی از اعداد رزرو شده رو نشون دادیم و همچنین گفتیم که دو عدد 0x800 و 0x0806 برای ما حایز اهمیت هستند.

یک نکته که باعث سردرگمی خواننده میشه رو اینجا براتون باز کنیم. گفتیم که مک آدرس ها باید در شبکه منحصربفرد و یکتا باشند. در مراجع مختلف هم این جمله به همین نحو گفته میشه ولی درستش اینه که بگیم در یک شبکه داخلی یا محلی؛ باید مک آدرس ها یکتا باشه. متاسفانه ما، یک ساختار که چند هاست رو به هم متصل می کنند رو میگیم "شبکه"؛ ساختاری که در اون تعداد زیادی شبکه های کوچک با استفاده از روترها، به هم متصلند رو هم، باز میگیم "شبکه". برای عدم آشفتگی ذهنی خواننده بهتر اینه که به اولی بگیم شبکه محلی؛ شبکه داخلی یا حتا زیرشبکه. خب کمی سخت میشه چون اونطوری باید در تمام این نوشتار از "شبکه محلی" به جای شبکه استفاده کنیم. خواننده حواسش باید باشه که در قسمت اعظم این نوشتار منظور از شبکه؛ یک شبکه داخلی ست. اجازه بدید یه مثال بزنم. در سیستم پستی و هنگامی که نامه ها داخل یک شهر توزیع میشن؛ اسم محله باید یکتا باشه. اما هنگامی که سیستم پستی بین شهرهای مختلف انجام میشه؛ در این حالت شما میتونید دو محله در دو شهر مختلف با یک اسم داشته باشید. در این حالت اسم یکسان محلات باعث سردرگمی نخواهد شد چون قبل از اسم محله؛ با توجه به اسم شهر؛ نامه ها از هم تفکیک خواهند شد. در مبحث شبکه هم همینطوره؛ در یک شبکه داخلی یا شبکه محلی؛ مک آدرس باید یکتا باشه. اما در هنگام اتصال شبکه های مختلف به هم (مثل شبکه اینترنت)؛ میشود که دو دستگاه، مک آدرس یکسان داشته باشند؛ اما باید در دو شبکه داخلی مختلف قرار داشته باشند. در هنگام اتصال بین شبکه ها؛ ارتباط توسط روترها برقرار میشه و در این ارتباط چیزیکه مهم هست، آدرسی ست که بهش میگیم آدرس IP . در فریم های ارسالی برای شبکه های دیگه؛ روترها موقعی که یک فریم رو دریافت میکنند، به جای مک آدرس مبدا، مک آدرس خودشون رو قرار میدن؛ به جای مک آدرس مقصد هم با توجه به اینکه هاست مقصد در شبکه بعدی قرار داره یا در یک شبکه دیگه ای؛ فرق خواهد کرد. در حالت اول مک آدرس هاست مقصد و در حالت دوم، مک آدرس روتر بعدی قرار میگیره.

همونطور که گفتیم ممکنه دستگاه های مختلفی درون یک شبکه باشند که سازنده های مختلف (با OUI مختلفی) دارند. از طرفی مک آدرس ها، اصطلاحا Hard core هستند؛ یعنی از قبل درون حافظه تراشه ثبت میشن درنتیجه ما نمیتونیم فقط بر اساس اونها یک زیرشبکه بسازیم، یعنی شبکه مون رو به بخشهای مختلفی تقسیم بندی کنیم که هر زیرشبکه دارای محدوده آدرس خاصی باشند، به آدرس دهی پستی دقت کنید؛ اگه فقط در یک شهر بخواهید نامه بفرستید؛ کافیه فقط اسم منطقه یا محله بعلاوه خیابون رو بنویسید؛ اگر در یک کشور بخواهید نامه ارسال کنید؛ بایستی اسم شهر رو هم اضافه کنید؛این وضعیت در آدرس دهی شبکه تلفن هم هست، در تلفن داخل شهری نیاز به کد شهر نیست؛ اما بین شهرها، باید از کد شهر و در بین کشورها از کد کشور استفاده

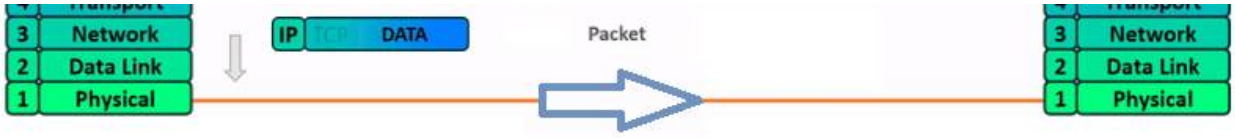
کنید. وجود زیرشبکه به دلایل مختلف نیازه؛ یکیش دلایل امنیتیه؛ که بتونیم دسترسی به شبکه رو محدود کنیم یا بتونیم قسمت های مختلف رو از هم تفکیک کنیم. مثلا در شبکه یک سازمان؛ بخش مالی از فنی جدا باشن و هر بخش تنها به داده های خودش دسترسی داشته باشه. اما خب تنها با اتکا به مک آدرس ها این امکان وجود نداره و درنتیجه نیاز هست که یک ساختار آدرس دهی جداگانه بوجود بیاد که اصطلاحا بهش میگن آدرس منطقی (در مقابل آدرس فیزیکی یا همون مک آدرس). اصلیترین پروتکل شبکه یعنی IP یا همون Internet Protocol از این روش آدرس دهی استفاده میکنه. در ارتباط بین شبکه ها به جای مک آدرس؛ از آدرس IP استفاده میشه. آدرس منطقی در این حالت یک نوع آدرس 4 بایتی است که بهش IP Address میگن. این آدرس معمولا به صورت دهدهی (decimal) نمایش داده میشه؛ به عنوان مثال 192.168.2.30

این آدرس دو بخش داره که اندازه هاشون بر حسب تعداد بایت، متفاوته (اما واضحه که جمعشون میشه 4) قسمت سمت چپ میشه آدرس زیرشبکه و مابقی آدرس، که سمت راست آدرس زیرشبکه قرار میگیرن ،میشه آدرس یا شماره host در اون زیرشبکه. در آدرسی که مثال زدیم اعداد 192.168.2 آدرس زیر شبکه و 30 شماره کامپیوتر در اون زیرشبکه است. اینجا این سوال ممکنه مطرح بشه که از کجا میدونیم کدام بخش های آدرس متعلق به زیر شبکه ن و کدام بخش ها متعلق به شماره host ؟ مفهومی بنام subnet mask داریم که اون هم مثل آدرس IP چهار بایتی هست. هر بیتی از subnet mask دارای مقدار '1' باشه؛ بیت متناظرش در آدرس IP جزو قسمت آدرس شبکه است و هر بیتی '0' باشه؛ بیت متناظرش در آدرس آیپی متعلق به آدرس host در زیرشبکه است. در آدرسی که مثال زدیم subnet mask برابره با 255.255.255.0 در نتیجه سه بایت سمت چپ؛ آدرس زیرشبکه ست و فقط بایت سمت راست؛ آدرس host در زیرشبکه رو مشخص میکنه. توجه داشته باشید که اگر subnet mask رو به صورت باینری بنویسیم؛ یک ها از سمت چپ شروع میشن و یک جایی تموم میشن و بین یک ها، صفری نخواهیم داشت(1 ها در سمت چپ و 0 ها در سمت راست قرار میگیرند و تنها در یک محل تغییر از 1 به 0 خواهیم داشت). گاهی هم subnet mask رو با یک عدد در جلوی آدرس IP ممکنه ببینید؛ مثلا آدرس به صورت 192.168.2.30/24 نمایش داده میشه به این معنی که 24 بیت اولیه آدرس، نشون دهنده آدرس زیرشبکه ست. در ضمیمه [10] در مورد کلاس های تخصیص آدرس IP و subnet mask بیشتر توضیح دادیم.

- Host (هاست یا به فارسی میزبان) به هر دستگاهی که در انتهای شبکه؛ داده نهایی رو ارسال یا دریافت میکنه، host میگن. لذا ابزارهایی مثل سوییچ ها (switch)، روترها (router یا مسیریاب)، هاب ها (hub) ، پل ها (bridge) ها؛ هاست به حساب نمیان. اما کامپیوترهای شخصی، گوشی های همراه، پرینترها و ... هاست هستند. بعلاوه به هر وسیله ای که داخل شبکه قرار میگیره و فریم ها به اون وارد یا

ازش خارج میشن هم، گره یا Node میگن. در ضمیمه [1] اطلاعات بیشتری در مورد اجزای شبکه در دسترس شما قرار میگیره.

همونطور که گفتیم در بخش نوع اترنت (دو بایت بعد از مک آدرس ها در فریم اترنت) عدد 0x0800 نشان دهنده اینه که فریم فعلی داده هایی رو در خودش داره که با فرمت پروتکل IP ارسال شده ن. اگه بخوایم روی مدل OSI یا TCP/IP توضیح بدیم نتیجه مثل شکل زیرهست. توجه داشته باشید که هر جا میگیم IP منظورمون Ipv4 هست.



در سمت فرستنده، اطلاعات از لایه 4 به لایه 3 که در مدل OSI به اون لایه شبکه (Network) میگن؛ تحویل داده میشه. لایه سوم با استفاده از پروتکل IP ، روی داده دریافتی از لایه چهارم، بسته بندی مخصوص به خودش رو انجام میده، این بسته بندی دو وظیفه اساسی داره. وظیفه اول که همیشه انجام میشه، طبق معمول، اضافه کردن بخشی بنام هدر (Header ؛ سرآمد یا سرآیند) به داده ها است؛ گاهی هم که حجم داده های دریافتی از لایه 4 بسیار زیاد هست و در یک فریم جا نمیشه؛ وظیفه تکه تکه کردن داده های لایه چهارم به قطعات کوچکتر و ارسال اون ها در چند بسته (Packet) هم اضافه میشه.

- لایه سوم در مدل TCP/IP بنام لایه اینترنت (Internet Layer) شناخته میشه.
- یکبار دیگه مرور کنیم: به جریان بیت هایی که لایه 1 ارسال میکرد؛ میگفتیم stream ؛ به داده هایی که لایه 2 به لایه 1 تحویل میداد؛ میگفتیم فریم (frame) و به داده هایی که لایه 3 به لایه 2 تحویل میده، پکت (packet یا بسته) گفته میشه.
- به فرآیند اضافه کردن هدر به داده های لایه بالاتر و تحویل داده جدید به لایه پایین تر ، کپسوله سازی (Encapsulation) گفته میشه. این در موقع ارسال هست. در گیرنده عکس این عمل انجام میشه ، بهش Decapsulation میگن.در گیرنده، هر لایه، داده ها رو از لایه پایین میگیره ، هدر خودش رو جدا میکنه و داده باقیمونده رو (بعلاوه تفسیر اون داده ها) به لایه بالاتر میده تا در نهایت داده اصلی به دست مصرف کننده نهایی برسه.

خب تا اینجا چی یاد گرفتیم ؛ اینکه لایه 3 به داده های دریافتی از لایه 4 یه هدر اضافه میکنه ؛ بعلاوه متوجه شدیم که در لایه 3 اصلیترین پروتکل شبکه اینترنت یا همون IP protocol قرار داره و اینکه این پروتکل از

آدرس دهی جداگانه ای (نسبت به آدرس فیزیکی یا همون مک آدرس) با نام آدرس منطقی یا IP Address استفاده میکنه.

آدرس IP ؛ یا از قبل (مثلا موقع کدنویسی میکروکنترلر) مشخص میشه که بهش میگیم حالت استاتیک و یا درون شبکه و با پروتکل های مخصوص مثل پروتکل DHCP (Dynamic Host Configuration Protocol) به host تعلق میگیره که به این حالت میگیم تخصیص دینامیک آدرس. در واقع در حالت دینامیک، هاست موقع اتصال به شبکه آدرس IP نداره؛ اما جزو اولین کارهاش، گرفتن آدرس IP هست. همینجا هم بگیم که پروتکل DHCP رو در موقع خودش توضیح میدیم. ما فعلا از حالت استاتیک استفاده میکنیم؛ چون هنوز چیزی از پروتکل DHCP نمیدونیم.

میدونیم که پروتکل IP از آدرس IP استفاده میکنه (البته هنوز با خود پروتکل IP درگیر نشدیم) اما یادمون هست که قبلا متوجه شدیم که لایه 2 از آدرس فیزیکی یا همون MAC Address استفاده میکرد، لذا اولین سوالی که باید برای ما پیش بیاد اینه که که یک هاست چطور میتونه بفهمه آدرس فیزیکی مرتبط با یک آدرس IP چیه؟ به عنوان مثال در همین برد خودمون؛ برد الکترونیکی قصد ارتباط با کامپیوتر رو داره؛ در این حالت نه آدرس آیپی و نه مک آدرس کامپیوتر رو نداریم! یک هاست، مک آدرس و IP آدرس خودش رو میدونه؛ فرض میکنیم آدرس IP طرف مقابل رو هم میدونه (اینکه چطور میدونه، الان برامون مهم نیست، به وقتش توضیح میدیم که معمولا با استفاده از سرویس DNS پیدا میشه یا بطور ثابت در برنامه اضافه شده!) اما لایه دوم باید مک آدرس طرف مقابل رو داشته باشه تا فریم رو ارسال کنه. اینجاست که لزوم داشتن فرآیندی، که قبل از هر کاری، بتونه مک آدرس طرف گیرنده رو بدست بیاره؛ احساس میشه. این وظیفه به عهده پروتکلی بنام ARP گذاشته شده.

- در عموم کتاب ها و منابع مربوط به اترنت؛ بعد از توضیح لایه دوم و فرمت فریم Ethernet ii به معرفی و اجرای پروتکل IP در لایه سوم (Ether Type=0x0800) پرداخته میشه؛ اما فکر میکنم قبل از اینکه پروتکل IP رو بررسی کنیم؛ بهتره لایه سوم رو با پروتکل ARP شروع کنیم. این نکته رو هم بگم که در بعضی منابع پروتکل ARP رو جزو لایه 2 به حساب آوردن، خیلی هم ایرادی نداره، چون از نظر عملکرد جزو لایه 2 حساب میشه، از نظر پیاده سازی جزو لایه 3 ؛ در ادامه متوجه منظورم خواهید شد.

پروتکل ARP :

پروتکل ARP (Address Resolution Protocol) با استفاده از آدرس IP مقصد یا گیرنده پیام؛ آدرس فیزیکی یا همون مک آدرس گیرنده رو بدست میاره تا بتونیم فریم های لایه 2 رو ارسال کنیم. فرض کنید شماره تلفن شخصی رو ندارید؛ چطور میتونید بهش تلفن کنید؟ ما ممکنه از سرویس هایی مثل 118 یا کتابچه های تلفن استفاده کنیم؛ اما در مبحث شبکه؛ سعی میشه با استفاده از قابلیت های خود شبکه، مشکلات حل بشه.

- در تعریف پروتکل های شبکه سعی شده اونها رو به گونه ای تعریف کنند که احتمالا در آینده و با تغییر ساختار شبکه؛ همچنان کارایی داشته باشند. به همین دلیل اونها رو تنها برای استفاده در یک شبکه خاص تعریف نکردن (تا حد امکان، عمومی تعریف شده ن).
- یه نکته کوچولو هم فقط جهت اطلاع بگیم، یه پروتکل دیگه ای در لایه سوم داریم که کارش عکس کار ARP هست، یعنی با آدرس فیزیکی ، آدرس منطقی رو بدست میاره. بهش میگن RARP یا همون Reverse ARP و در حال حاضر استفاده ای برای ما نداره.

کار پروتکل ARP چیه؟ هر وقت نیاز میشه یه بسته با استفاده از پروتکل IP رو بفرستیم، قبلش چک میکنیم که آیا مک آدرس گیرنده رو داریم یا نه؟ اگر مک آدرس رو نداشته باشیم، اول باید اون رو بدست بیاریم؛ پس هر هاست، نیاز داره جدولی داشته باشه به اسم جدول ARP که توش مشخص شده هر آیپی متعلق به کدام مک آدرس است. این عملیات در داخل میکروکنترلر پیاده سازی (Implementation) میشه، از طرفی چون ممکنه هر قطعه ای از شبکه خارج بشه (خاموش بشه؛ کابلش جدا بشه یا حتا هنگ و ریست شده باشه) معمول اینه که جدول ARP بطور مرتب چک و بروز میشه، به همین دلیل اگر با نرم افزاری مثل wireshark شبکه رو شنود کنید، میبینید که هر از گاهی بسته های ARP دوباره ارسال میشه.

- بسته های ARP مسیریابی نمیشوند؛ یعنی بسته های پروتکل ARP فقط درون شبکه داخلی انتشار می یابند و روترها این بسته را اصطلاحا روت نمی کنند. اگر این پرسش براتون بوجود اومده که اگر بخوایم با یک دستگاه در زیرشبکه دیگه ای ارتباط بگیریم؛ چطور مک آدرس اون رو به دست میاریم؛ جواب اینه که اصولا هر هاست در یک زیرشبکه از طریق یک روتر به شبکه های دیگه متصل هست؛ در این وضعیت، برای آدرس دهی، اطلاعات لایه سوم یعنی آدرس IP استفاده میشه. روتر بعد از دریافت یک بسته برای ارسال به یک شبکه دیگه؛ اطلاعات لایه دوم رو تغییر میده. هرگاه نیاز هست پیامی به شبکه دیگه ای ارسال بشه؛ هاست مبدا، این پیام رو به روتر میفرسته و در لایه دوم، مک آدرس روتر شبکه داخلی خودش یا همون gateway رو به عنوان مک آدرس مقصد اعلام میکنه. اما آدرس IP مقصد؛ آدرس هاست مورد نظر هست. بخش معرفی اجزا در ضمایم رو ببینید.

- تعداد زیادی نرم افزار که اصطلاحا به اونها sniffer میگن؛ برای شنود و بررسی داده های داخل شبکه وجود دارند که معروفترین اونها wireshark هست. در ضمیمه [5] در مورد استفاده از این نرم افزار یه توضیح ساده گذاشتیم. حتما این نرم افزار رو روی سیستمتون داشته باشید.

تا اینجا تراشه ENC28J60 رو راه انداختیم و دو تابع نوشتیم که توسط اونها، فریم های رسیده از شبکه رو دریافت و پردازش میکنیم یا فریم هایی رو روی شبکه ارسال می کنیم. شاکله برنامه رو هم گفتیم که میخوایم اینطور پیاده سازی کنیم که یه پکت رو میگیریم، پردازش میکنیم و اگه نیاز بود جوابش رو همونجا بوجود میاریم (داخل همون حافظه ای که داخل میکروکنترلر برای دریافت فریم ها در نظر گرفتیم).

وقتی یه فریم رو میگیریم، مک آدرس ها بالطبع سرجاشون هستند (در 12 بایت ابتدایی فریم) دو بایت بعدی در فریم بنام Ether type رو برمیداریم، در حالتی که این فریم متعلق به پروتکل ARP باشه، عدد داخل این دو بایت 0x0806 هست. پس بایت های داده ی داخل فریم متعلق به این پروتکل هستند. گفتیم هر پروتکلی، هدر مخصوص به خودش رو در ابتدای داده ها اضافه میکنه؛ پس در ابتدای بخش داده ها در یک فریم، باید بخش هدر مربوط به ARP رو تفکیک و بررسی کنیم. اگر بخواهیم ترکیب بایت های درون فریم در لایه دوم رو بصورت شکل نشون بدیم، یه همچین چیزی هست (از چپ به راست) :

6 octet Destination MAC Address	6 octet Source MAC Address	2 octet Ether Type 0x0806	Payload in frame		4 octet CRC
			28 octet ARP Header	18 octet Padding 0x00	

12 بایت ابتدایی مک آدرس ها هستند. 2 بایت با مقدار 0x0806 نوع پروتکل لایه سوم رو مشخص میکنه که در این حالت، ARP هست. در بخش داده های فریم (payload)؛ فقط بخش هدر در پروتکل ARP وجود داره که تعداد 28 بایت، فضا اشغال میکنه. داده ی دیگری توسط این پروتکل ارسال نمی شود و از آنجاییکه اندازه کل فریم 46 بایت (کمتر از 64 بایت برای حداقل طول فریم) است لذا در ادامه فریم؛ بسته به پیاده سازی لایه دوم؛ ممکن است شما 18 بایت با مقدار 0x00 برای padding داشته باشید. در انتهای فریم هم طبق معمول CRC خواهیم داشت. فرمت هدر، در پروتکل ARP در جدول زیر نشان داده شده است:

Octet Offset	0	1
0	HType (Hardware Type)	
2	PType (Protocol Type)	
4	HLen (Hardware Address Length)	PLen (Protocol Address Length)
6	OPER (Operation)	
8	SHA (Sender Hardware Address) first 2 Bytes	
10	SHA next 2 Bytes	
12	SHA Last 2 Bytes	
14	SPA (Sender Protocol Address) first 2 Bytes	
16	SPA last 2 Bytes	
18	THA (Target Hardware Address) first 2 Bytes	
20	THA next 2 Bytes	
22	THA Last 2 Bytes	
24	TPA (Target Protocol Address) first 2 Bytes	
26	TPA last 2 Bytes	

HTYPE : دو بایت با نام Htype، نوع سخت افزار رو مشخص میکنه. برای برد ما که از اترنت 10Mb استفاده میکنه؛ داخل این دو بایت عدد 0x0001 باید باشه.

- برای اینکه بدونید چه اعدادی در اینجا میتونن قرار بگیرن، سند RFC 1700 و ارجاعات اون رو ببینید. این RFC شامل اعداد رزرو شده در پروتکل های مختلف؛ برای قرار گیری در محل های مختلف رو ثبت کرده. این کار توسط سازمانی بنام IANA انجام میشه.
- بطور کلی در مورد شبکه اینترنت و استانداردهای اون، چند سازمان یا انجمن در اجرای کار دخیل هستند که ممکنه در طول این نوشتار بهشون برخورد کنیم، مثل :

IANA: Internet Assigned Numbers Authority

IETF: Internet Engineering Task Force

IEEE: Institute of Electrical and Electronics Engineers

DARPA: Defence Advanced Research Projects Agency

PTYPE : یا Protocol Type؛ شامل دو بایت؛ که اگه یادتون باشه برای پروتکل IP عدد 0x0800 بود. در واقع در این بخش نشون میدیم آدرس منطقی ما، آدرس مورد استفاده در پروتکل IP خواهد بود.

HLEN : در این قسمت که یک بایتی هست، تعداد بایت های آدرس فیزیکی یا سخت افزاری ما مشخص میشه و از اونجایی که مک آدرس ها، 6 بایتی هستند؛ در این جا عدد 0x06 نوشته میشه.

PLEN : در این بخش که یک بایت رو اشغال کرده؛ تعداد بایت های آدرس منطقی نوشته میشه، آدرس منطقی ما از نوع IP Address هست؛ پس در اینجا عدد 0x04 نوشته میشه.

OPER : نوع علیات رو مشخص میکنه. کلا دو نوع پیام روی ARP مورد استفاده ما هست، درخواست (Request) و پاسخ (Reply) که جهت درخواست عدد 0x01 و برای پاسخ عدد 0x02 استفاده میشه. سمتی که نیاز داره IP یه دستگاه دیگه رو پیدا کنه، درخواست (0x01) میفرسته و سمت مقابل پاسخ (0x02) میده.

SHA : یا آدرس سخت افزاری فرستنده. در این 6 بایت، مک آدرس فرستنده این پکت قرار میگیره.

SPA : در اینجا 4 بایت آدرس IP فرستنده نوشته میشه.

در ادامه و در بخش های THA و TPA هم مک آدرس و IP آدرس طرف مقابل نوشته میشه.

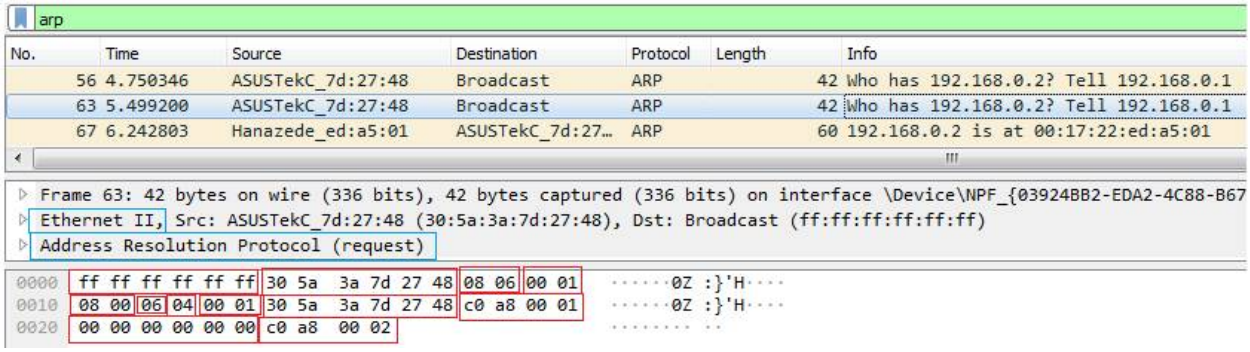
در ابتدا درخواستی از طرف درخواست کننده ارسال میشه. از اونجاییکه در این مرحله، هنوز مک آدرس طرف مقابل رو نمیدونه، پس به جای مک آدرس عدد FF:FF:FF:FF:FF:FF ارسال میشه. اگه یادتون باشه این آدرس رو بهش میگفتیم Broadcast Address در نتیجه تمامی دستگاه های متصل به شبکه داخلی، این پیام رو دریافت می کنند. در جواب این پیام(درخواست)؛ دستگاهی که IP آدرسش در پیام قبلی اومده؛ پیامی ارسال میکنه اما دیگه اینجا هم مک آدرس و هم IP آدرس طرف مقابل رو میدونه. مک آدرس خودش رو هم اضافه میکنه و نوع پیام رو در قسمت OPER عدد 0x02 میذاره و ارسال میکنه. در نتیجه این پیام دیگه broadcast نیست. به این فرآیند اصطلاحا Who has? میگن، درخواست کننده پیامش رو اینجور میفرسته:

Who has IP=aa-aa-aa-aa ? tell me IP=yy-yy-yy-yy at mac=zz:zz:zz:zz:zz:zz

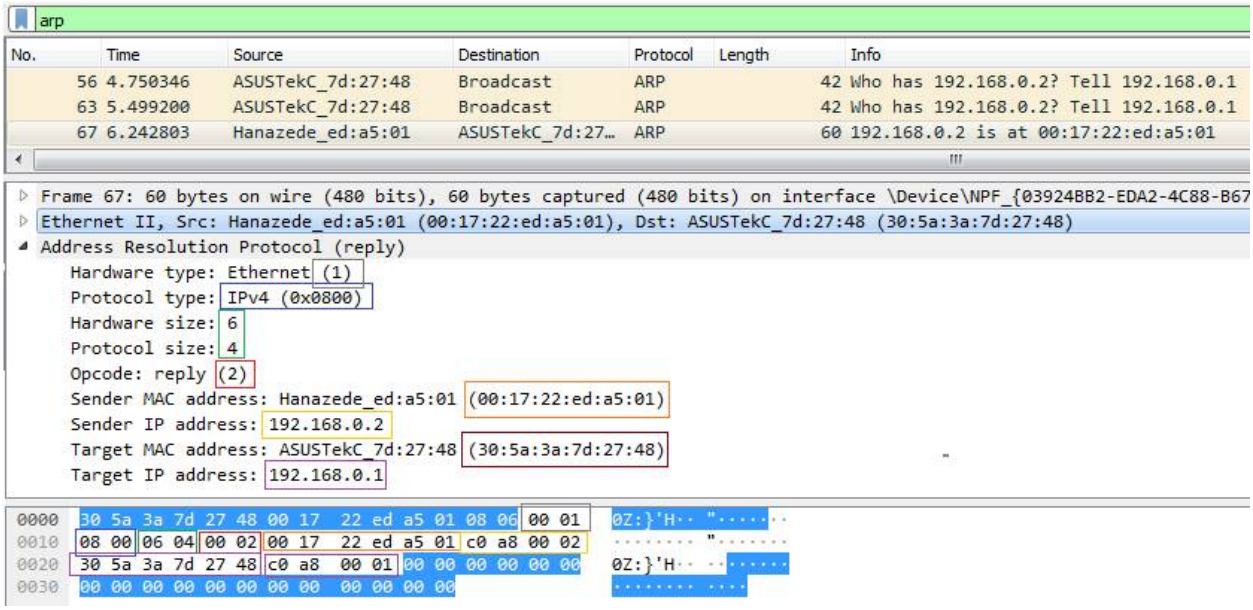
در جواب هم سمت مقابل میگه

I am at IP=xx-xx-xx-xx ,mac=bb:bb:bb:bb:bb:bb

در شکل های زیر درخواست و پاسخ ARP ثبت شده توسط نرم افزار wireshark رو می بینیم:



شکل 18 : درخواست ARP



شکل 19: پاسخ ARP

تصاویر مربوط به درخواست ARP از طرف کامپیوتر با سیستم عامل ویندوز و چیپ ASUS و پاسخ از طرف برد ما با تراشه ENC هست (بخش OUI مک آدرس رو مقدار 0x00,0x17,0x22 گذاشتیم؛ درنتیجه، در گزارش، wireshark سازنده چیپ رو شرکت دیگه ای به جز میکروچیپ تشخیص داده)؛

- صفرهای موجود در انتهای پیام پاسخ، ناشی از عمل padding هست ولی همونطور که در پیام درخواست میبینید، عمل padding از طرف ویندوز انجام نشده!
- در تصویر تعداد بایت های واقع در فریم درخواست؛ عدد 42 نوشته شده. اختلاف 4 تایی با عدد 46 بخاطر محاسبه نشدن بخش CRC هست.
- برای اطلاعات بیشتر در مورد پروتکل ARP میتونید به سند RFC 826 و بروزرسانی های اون مراجعه کنید.

اگه بخوایم نسبت این پروتکل و فریم Ethenet ii رو بصورت تصویری نشون بدیم؛ چیزی مثل جدول زیر میشه:

لایه 3 (Network Layer)	IP (packet)	ARP (packet)
لایه 2 (Data Link Layer)	Ethernet ii (frame)	
لایه 1 (Physical Layer)	'0' , '1' (Stream)	

بهتره بریم سر وقت نرم افزار میکروکنترلرمون تا کار رو ادامه بدیم و پاسخ به درخواستهای ARP رو توش پیاده سازی کنیم. اگه یادتون باشه ما یه ساختار به اسم ENC28J60_Frame داشتیم و یک تابع که توسط اون فریم های Ethernet ii رو دریافت و در متغیری از این ساختار (struct) ذخیره می کردیم.

از اوجایی که کار ما با تراشه ENC تموم شده و جهت اینکه احتمالا بعدا، براحتی بتونیم از تراشه دیگه ای استفاده بکنیم؛ برای اضافه کردن مابقی کدها (که پردازش پروتکل ها رو انجام خواهند داد) یک زوج فایل جدید بصورت .c, .h به برنامه اضافه میکنیم و اسمشون رو Ethernet.h , Ethernet.c میذاریم.

در فایل اصلی برنامه (main.c) و در تابع main؛ متغیری عمومی بصورت زیر تعریف میکنیم:

```
ENC28J60_Frame frame;
```

برای اینکه مجبور نباشید برگردید عقب، مجددن اعلان ساختار ENC28J60_Frame رو اینجا براتون میذاریم

```
typedef struct ENC28J60_Frame
{
  uint16_t nextPtr;
  uint16_t length;
  uint16_t status;
  uint8_t data[ENC28J60_FRAME_DATA_MAX];
  uint32_t 59hecksum;
} ENC28J60_Frame;
```

در فایل جدید تابعی به فرمت

```
void ETH_Process (ENC28J60_Frame* encFrame);
```

تعریف میکنیم، این تابع آدرس متغیری از نوع ساختار ENC28J60_Frame رو میگیره و داده های داخل اون رو پردازش میکنه. روال کار اینطوریه که در حلقه اصلی برنامه ((while(1)) ما بطور متوالی این تابع رو فراخوانی

میکنیم. این تابع با استفاده از تابعی که قبلا تعریف کردیم (ENC28J60_ReceiveFrame) فریم های دریافت شده رو پردازش میکنه و در صورت لزوم پاسخ میده (با استفاده از تابع ENC28J60_TransmitFrame) پس

داخل تابع ETH_Process اولین کاری که باید انجام بدیم اینه که ببینیم فریم جدیدی داریم یا نه:

```
uint16_t requestSize = ENC28J60_ReceiveFrame(encFrame);
```

اگر فریمی موجود باشه : `if (requestSize > 0)`

بخش داده از فریم جدید رو جدا کرده و نوع اون رو هم به ساختار جدیدی بنام ETH_Frame اصطلاحا typecast می کنیم:

```
ETH_Frame* ethFrame = (ETH_Frame*)encFrame->data;
uint16_t etherType = ntohs(ethFrame->etherType);
uint16_t ethDataLen = requestSize – sizeof(ETH_Frame);
```

همینجا اعلان ساختار ETH_Frame رو هم نشون بدیم:

```
typedef struct ETH_Frame
{
 uint8_t destMacAddr[MAC_ADDRESS_BYTES_NUM];
 uint8_t srcMacAddr[MAC_ADDRESS_BYTES_NUM];
 uint16_t etherType;
 uint8_t data[];
} ETH_Frame;
```

همونطور که می بینید؛ این ساختار به فرمت یک فریم Ethernet ii هست؛ یعنی دو آرایه 6 بایتی برای مک آدرس ها، 2 بایت برای Ether Type و یک آرایه با اندازه نامشخص از داده های فریم براش درنظر گرفتیم.

خب برگردیم سراغ کد قبلی، در خط اول قسمت encFrame->data رو اصطلاحا typecast کردیم و آدرس اون رو در اشاره گری بنام ethFrame ذخیره کردیم.

در خط بعدی نوع اترنت (etherType) رو هم از داخل ساختار کشیدیم بیرون و به صورت جداگانه در متغیری ذخیره کردیم. در خط سوم هم از اندازه تعداد بایت های فریم اصلی ، اندازه ساختار ETH_Frame رو کم کردیم ؛ به این ترتیب اندازه داده های درون فریم رو بدست آوردیم.

اگه دقت کرده باشید؛ در خط دوم از چیزی به اسم ntohs استفاده شده که ظاهرش مثل توابع C هست اما در واقع تنها یک define با آرگومان هست، قبل از پیاده سازیش باید بگیم چی هست و چی کاری میکنه!

- دو تفاوت بین روش ثبت و پردازش داده ها در شبکه و همچنین سیستم عامل ویندوز و بسیاری از کامپایلرهای میکروکنترلری هست که عدم توجه به اون ها؛ باعث باگ های نرم افزاری زیادی میشه. سعی کنید همیشه این دو تفاوت در ذهنتون باشه. اول اینکه در شبکه، اطلاعات به صورت Big Endian ردوبدل میشه؛ یعنی داده ی پرارزش ابتدا ارسال میشه (ضمیمه [8]) ،ولی این اطلاعات به ترتیب ورود، در حافظه ذخیره میشن درنتیجه داده های پرارزش در آدرس های کمتر حافظه (داخل آرایه ها) قرار می گیرند. نکته بعدی هم اینکه؛ بسیاری از محاسبات در شبکه به فرم مکمل 1 (one's complement) انجام میشه؛ در حالی که، در ویندوز و عموم میکرونترلرها محاسبات به فرم مکمل 2 (two's complement) انجام میشه. درباره مورد دوم به وقتش توضیح بیشتری میدیم.

تابع ntohs در واقع تفاوت اول رو حل و فصل میکنه. همونطور که گفتیم داده ها در شبکه بصورت Big Endian ارسال میشن، مثلا در ارسال دو بایت Ether Type که در پروتکل ARP برابر با 0x0806 بود؛ ابتدا بایت 0x08 ارسال میشه و بعد 0x06 ، خب ما این ها رو داخل یک آرایه ریختیم؛ از اونجایی که این اطلاعات به ترتیب ورود، درون این آرایه ذخیره شدن؛ لذا اگه بخوایم اونها رو پردازش کنیم، چیزی که میبینیم (میکروکنترلر میبینه) در واقع عدد 0x0608 هست.

برای همین میایم و با استفاده از ntohs جای بایت ها رو هنگام انتقال به متغیر etherType عوض می کنیم.

همونطور که گفتیم به جای تعریف تابعی برای اینکار؛ از define های آرگومان دار استفاده کردیم. یه نکته دیگه هم اینکه، دو تا define تعریف می کنیم؛ یکی برای داده های دو بایتی و دیگری برای چهار بایتی ها.

پس در ابتدای فایل Ethernet.c همچین چیزی اضافه میشه:

```
#define htons(val)          ((val << 8) & 0xFF00) | ((val >> 8) & 0xFF)
#define htonl(val)  ((val << 8) & 0xFF0000) | ((val >> 8) & 0xFF00) | ((val << 24) & 0xFF000000) |
((val >> 24) & 0xFF)

#define ntohs(val)     htons(val)
#define ntohl(val)      htonl(val)
```

h به جای host و n به جای network ، s برای short و l برای long بکار رفته. همونطور که دیده میشه کار ntohs و htons همینطور کار ntohl و htonl یکسانه و فقط به جهت تشخیص جهت ارتباط چهار تا شده ن وگرنه همون دوتای اول کفایت میکرد. کار این کد هم فقط جابجا کردن (swap) بایت های داخل متغیر هست.

خب برگردیم به ادامه تفسیر کد:

```
  // ARP protocol
  if (etherType == ETH_FRAME_TYPE_ARP)  //0x0806
  {
    responseSize = ARP_Process((ARP_Frame*)ethFrame->data, ethDataLen);
  }
```

نوع فریم رو چک میکنیم که اگر از نوع ARP باشه، جوابش رو تهیه کنیم. برای بار چندم یادآوری میکنیم که جواب ها در همون حافظه ای که برای اطلاعات دریافتی اختصاص دادیم، قرار میگیرند، لذا چنانچه نیاز به ارسال پاسخ باشه (مثل همینجا در ARP) کافیه جای مک آدرس ها رو داخل فریم اصلی عوض کنیم (در واقع جای فرستنده و گیرنده عوض میشه) و بقیه اطلاعات رو هم با توجه به پروتکل دریافتی تنظیم کنیم؛ مثلا اینجا باید در قسمت OPER عدد 0x02 رو بنویسیم.

- یه نکته ای هم اینجا بد نیست دوباره تذکر بدیم که در قسمت فیلترهای تراشه ENC ، حداقل باید دو قابلیت فعال شده باشه، دریافت پکت هایی که مک آدرس گیرنده ما توش هست بعلاوه پیام های broadcast شده با مک آدرس تمام 0xFF

و در انتها هم چنانچه جواب لازم باشه اون رو ارسال میکنیم

```
  if (responseSize > 0)
  {
    ETH_Response(ethFrame, responseSize);
  }
```

در نهایت تابع ETH_Process همچین چیزی میشه:

```
void  ETH_Process(ENC28J60_Frame*   encFrame)
{
 uint16_t  responseSize = 0;
 uint16_t  requestSize = ENC28J60_ReceiveFrame(encFrame);
 if (requestSize > 0)
 {
  ETH_Frame*   ethFrame = (ETH_Frame*)encFrame->data;
  uint16_t  etherType   = ntohs(ethFrame->etherType);
  uint16_t  ethDataLen = requestSize – sizeof(ETH_Frame);

  // ARP protocol
  if (etherType == ETH_FRAME_TYPE_ARP)  //0x0806
  {
    responseSize = ARP_Process((ARP_Frame*)ethFrame->data, ethDataLen);
  }
```

```
   if (responseSize > 0)          ////////////// Reply
   {
    ETH_Response(ethFrame, responseSize);
   }
  }
}
```

در کد بالا دو تابع، هنوز برای ما ناشناخته است. تابع ETH_Response؛ تنها جای مک آدرس گیرنده و فرستنده رو تغییر میده و فریم رو ارسال میکنه:

```
static void ETH_Response(ETH_Frame* ethFrame, uint16_t len)
{
 memcpy(ethFrame->destMacAddr, ethFrame->srcMacAddr, MAC_ADDRESS_BYTES_NUM);
 memcpy(ethFrame->srcMacAddr, macAddr, MAC_ADDRESS_BYTES_NUM);

 ENC28J60_TransmitFrame((uint8_t*)ethFrame, len + sizeof(ETH_Frame));
}
```

تابع memcpy از توابع داخلی KEIL و اصلی زبان C هست و محتویات یک قسمت از حافظه رو با اندازه معلوم به جای دیگه ای از حافظه کپی میکنه.

تابع ARP_Process هم در ادامه اومده و کار ساده ای داره. ابتدا بررسی میکنه که آدرس IP ما در بخش آدرس IP درخواست اومده باشه (یادمون هست که این پیام یک پیام عمومی هست و لذا تمام پیغام های ARP رو میکروکنترلر ما دریافت و پردازش خواهد کرد؛ اما تنها به اون هایی باید جواب بده که مقصدشون ما هستیم) و بعد برای تنظیم پیام پاسخ؛ در قسمت OPER عدد 0x0002 رو مینویسه و مک آدرس و IP آدرس رو داخل هدر ARP تنظیم میکنه!

```
uint16_t  ARP_Process(ARP_Frame* arpFrame, uint16_t frameLen)
{
 uint16_t newFrameLen = 0;
 if (memcmp(arpFrame->destIpAddr, ipAddr, IP_ADDRESS_BYTES_NUM) == 0)
 {
  if ((arpFrame->opCode) == ( ntohs(ARP_OP_CODE_REQUEST) ) ) //==0x0001
  {
   memcpy(arpFrame->destMacAddr, arpFrame->srcMacAddr, 6);
   memcpy(arpFrame->srcMacAddr, macAddr, 6);
   memcpy(arpFrame->destIpAddr, arpFrame->srcIpAddr, 4);
   memcpy(arpFrame->srcIpAddr, ipAddr, 4);
   arpFrame->opCode = htons(ARP_OP_CODE_RESPONSE);//0x0002
   newFrameLen = frameLen;
```

```
  }
 }
 return newFrameLen;
}
```

حلقه اصلی برنامه داخل فایل main.c هم این شکلی میشه:

```
 while (1)
 {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
       ETH_Process(&frame);
 }
```

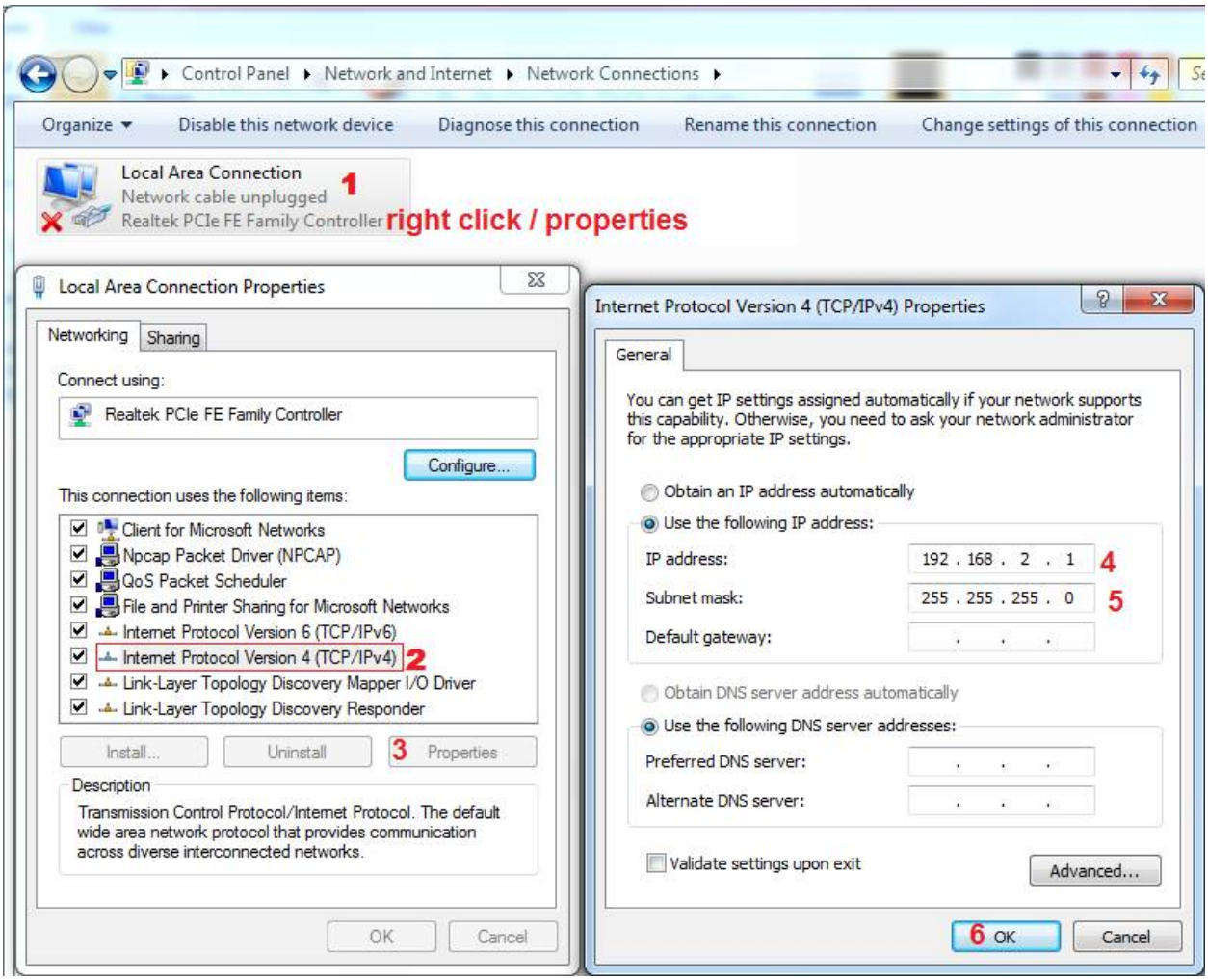
دو متغیر اصلی برنامه یعنی مک آدرس و IP آدرس خودمون رو هم یادمون نره که در ابتدای فایل enc28j60.c تعریف کنیم:

```
uint8_t  macAddr[MAC_ADDRESS_BYTES_NUM] = {0x00, 0x17, 0x22, 0xED, 0xA5, 0x01};
uint8_t  ipAddr[IP_ADDRESS_BYTES_NUM] = {192, 168, 0, 22};
```

فرض میکنیم که کد بدون خطا، کامپایل و پروگرم شده؛ حالا چجوری میشه برنامه رو تست کنیم؟

اولین کار اینه که محدوده (range) آیپی کامپیوتر و بردمون رو یکی کنیم؛ بطوریکه این دو در یک زیر شبکه قرار بگیرند، شما هر آیپی برای بردتون در نظر گرفتید، آیپی کامپیوتر رو هم در همون محدوده قرار بدید و فقط عدد اول (از سمت راست) رو تغییر بدید؛ مثلا اگر آیپی دستگاه 192.168.0.22 هست، شما میتونید آیپی کامپیوترتون رو 192.168.0.1 بذارید یا هر آیپی دیگه ای به فرم بالا بجز اینکه عدد آخرش 22 یا صفر باشه.

- جهت تغییر IP کامپیوتر هم وارد Control Panel\Network and Internet\Network and Sharing Center بشید (یا روی آیکنش در تسک بار ویندوز کلیک راست کنید و بازش کنید) وگزینه Change Adapter setting رو انتخاب کنید؛ روی کارت شبکه کلیک راست کنید و گزینه Properties رو انتخاب کنید، در پنجره باز شده، گزینه Ipv4 رو انتخاب کنید و دکمه Properties رو بزنید؛ در پنجره باز شده گزینه Use the following IP Address رو انتخاب کنید و IP خودتون رو وارد کنید. بعد هم فقط یه کلیک در قسمت subnet mask بکنید و در نهایت کلید OK رو بزنید و تمام.



شکل 20 نحوه تنظیم آدرس IP در ویندوز

پنجره cmd.exe یا همون کامند پرامپت ویندوز رو باز کنید. برای این کار در قسمت استارت ویندوز، تایپ کنید cmd و داخل پنجره باز شده تایپ کنید :

```
Ping  192.168.0.22
```

و Enter رو بزنید. همزمان میتوانید با استفاده از قابلیت دیباگ میکروکنترلرتون (اگه داشته باشه) و یا نرم افزار wireshark پیام ها رو رصد کنید.

- دستور یا عملیات ping رو به موقع توضیح میدیم. همینقدر بدونید که اگه کامپیوترتون مک آدرس برد شما رو نداشته باشه؛ قبل از عملیات ping؛ از پروتکل ARP استفاده میکنه تا مک آدرس ENC رو بدست بیاره. از اونجایی که هنوز پردازش دستور پینگ رو انجام نمیدیم؛ انجام این دستور با خطا مواجه میشه ولی ما فعلا فقط نیاز به ارسال فریم ARP داریم تا بردمون رو تست کنیم که با این روش انجام دادیم.

- بعد از ARP میریم سراغ IP ؛ بعدش میریم سمت ICMP و در اون نقطه از ping بیشتر صحبت خواهیم کرد.

قبل از اینکه بریم سر وقت پروتکل بعدی یعنی IP یا همون Internet Protocol با چنتا از قابلیت های تراشه ENC آشنا بشیم و همینطور چند اصطلاح جدید یاد بگیریم. اگه توجه کرده باشید روال کارمون تا اینجا اینطور بود که مفاهیم رو کم کم ارایه میدیم تا خواننده خیلی آشفته نشه در مقابل حجم اطلاعات وارده!

در دیتاشیت تراشه ENC چند قابلیت نوشته شده که شاید نیاز به توضیح داشته باشند. اولین قابلیت، تشخیص خودکار قطبیت (AutoPolarity detection) هست. در یک ارتباط دیفرانسیلی، اگه جای پین های مثبت و منفی رو اشتباه متصل کنیم؛ طبیعیه که ارتباط برقرار نشه ولی AutoPolarioty این قابلیت رو داره که حتا در صورت اتصال اشتباه پین های مثبت و منفی -TX,+TX به گیرنده هم، بیت ها رو درست بخونه. در ابتدای برقراری یک ارتباط؛ تراشه منتظر الگوی خاصی از بیت ها هست. چنانچه سطح بیت ها در این روال متقارن باشه؛ تراشه متوجه میشه که پین ها اشتباه متصل شده ن و در داخل تراشه، جای پین ها تعویض میشه.

دومین قابلیت Auto MDI-X هست. از اسم هم مشخصه که مربوط به بخش MDI هست، حالا چی هست؟ در یک ارتباط بایست بخش TX فرستنده به RX گیرنده متصل بشه. در تراشه های اولیه، هر دو دستگاه دارای بخش MDI بودند؛ لذا لازم بود که حتما از کابل های معروف به کابل کراس (cross) یا متقاطع استفاده بشه که جای پینهای TX و RX روی کابل تغییر میکرد. بعدها، تراشه هایی بوجود اومد که این تقاطع، داخل خود تراشه انجام میشد که بهش MDI-X میگفتن (اون X آخر، علامت تقاطع هست) لذا نیاز نبود که از کابل کراس استفاده بشه و اتصالات روی کابل یک به یک بود، به این کابل ها، کابل مستقیم یا straghit میگفتن. پس برای اتصال قطعات قدیمی و جدید در شبکه؛ 4 حالت برای کابل داشتیم:

	MDI	MDI-X
MDI	Cross	Straight
MDI-X	Straight	Cross

تراشه های با قابلیت Auto MDI-X با اجرای یک فرآیند، متوجه میشن که آیا ارتباط TX به Rx درست هست یا نه! و اگر این ارتباط به دستی برقرار نشده باشه؛ درون تراشه، ارتباط رو به شکل درست برقرار میکنن. در نتیجه در این حالت، از هر دو نوع کابل میتونید استفاده کنید. پیشنهاد میکنم فرآیند بررسی و تغییر اتصال در حالت Auto MDI-X رو در نت جستجو و مطالعه کنید، روش ساده اما جالبی برای انجام این کار داره!

یک اصطلاح و یک نکته دیگه هم Auto Negotiation یا مذاکره خودکار هست. هنگام روشن شدن یا هنگام اتصال کابل به سوکت؛ دو طرف کابل، برای بررسی قابلیت های همدیگه مثل سرعت، نوع ارتباط (نیم دوطرفه یا تمام دوطرفه) و ... یک فرآیندی رو طی می کنند تا حالت بهینه رو تنظیم کنند. به این فرآیند مذاکره خودکار گفته می شود. این تنظیمات رو می توان به صورت دستی هم انجام داد اما پیشنهاد نمی شود.

از طرفی ممکنه با اصطلاحی به نام Jabber Condition یا Jabber detection هم برخورد کنید. اگر فریم دریافتی، داده ای بیش از مقدار تعریف شده (1500 در اترنت پایه؛ 1504در Tag frame یا 1982 در Envelope frame) داشته باشه؛ اصطلاحا شرایط Jabber اتفاق افتاده و بررسی اون جزو وظایف قسمت PHY هست. از اونجاییکه این وضعیت ممکنه در حالت هنگ فرستنده اتفاق افتاده باشه (لذا باعث سرریز حافظه دریافت بشه)؛ میشه تراشه PHY رو طوری تنظیم کنیم که این فریم ها رو دریافت نکنه.

یک نکته کاربردی هم اینجا بگیم و بحث رو تموم کنیم. در حین ارتباط و به طور مرتب؛ برقراری ارتباط (متصل بودن کابل و روشن بودن سمت دیگر) با ارسال پالس های همزمانی که به نام LTP(Link Test Pulse) یا NLP(Normal Link Pulse) شناخته می شوند؛ چک خواهد شد. در سرعت 10Mb این پالس ها در حالت توقف (Idle) و تقریبا در هر 16ms ارسال می شوند. در سرعت 100Mb نوع بررسی برقراری ارتباط؛ همچنین نوع کدینگ preamble و داده ها متفاوت هست. خوشبختانه این امور توسط سخت افزار PHY انجام و از طریق ثبات های وضعیت داخلی به اطلاع کاربر رسانده می شود. ما نیاز ضروری به دانستن آنها نداریم ولی برای اطلاع بیشتر میتونید به سند کاربردی AN1120 از شرکت میکروچیپ یا اسناد شرکت های دیگه در این مورد مراجعه کنید.

پروتکل IP :

قبل از معرفی پروتکل IP؛ اجازه بدید موقعیت اون رو در لایه بندی شبکه، یک بار دیگه یادآوری کنیم. همچنین دوباره بگیم که منظورمون از IP؛ ورژن چهار اون یعنی IPv4 هست.

لایه 3 (Network Layer)	IP (packet)	ARP (packet)
لایه 2 (Data Link Layer)	Ethernet ii (frame)	
لایه 1 (Physical Layer)	‘0’ , ‘1’ (Stream)	

همونطور که در جدول دیده میشه؛ پروتکل IP در لایه سوم موسوم به لایه شبکه (Network) در مدل OSI یا لایه اینترنت در مدل TCP/IP قرار داره، این پروتکل از یک نوع آدرس دهی منطقی به اندازه 4 بایت استفاده میکنه. با چهار بایت میشه حدود 4.3 میلیارد host در شبکه داشته باشیم. پیشرفت و گسترش هر روزه اینترنت و نیاز به فضای آدرس دهی بزرگتر، باعث بوجود اومدن ورژن جدیدتری از IP شد که به جای 4 بایت از 8 کلمه دوبایتی (128 بیت) آدرس استفاده میکنه. این دو با نام های IPv4 و IPv6 از هم تشخیص داده میشن. این پروتکل ها، جدا از بحث آدرس دهی، تفاوت های دیگه ای هم با هم دارند؛ در نتیجه در لایه های بالاتر، پروتکل هایی مبتنی بر IPv6 تعریف شده ن که بسیاری از اون ها نمونه مشابهی در ورژن 4 دارند. در حال حاضر، مشکل کمبود آدرس در IPv4 رو به طریقی حل کرده اند (NAT رو در ضمایم ببینید) و IPv4 همچنان گسترده ترین پروتکل در لایه سوم هست. اینجا قصد داریم IPv4 رو بررسی و راه اندازی کنیم.

از اونجاییکه پروتکل IP در لایه سوم و در کنار ARP کار میکنه؛ در نتیجه براحتی میتونیم کد مورد نیاز رو در برنامه میکروکنترلر اضافه کنیم. یادمون هست که گفتیم IPv4 در بخش Ether Type از یک فریم استاندارد Ethernet ii عدد 0x0800 رو حمل میکنه. پس یک if دیگه برای پردازش پکت های IP به تابع ETH_Process اضافه می کنیم. اسم تابع رو؛ همچنان که روالمون هست IP_Process میذاریم.

پس تابع ETH_Process اینجوری تغییر میکنه:

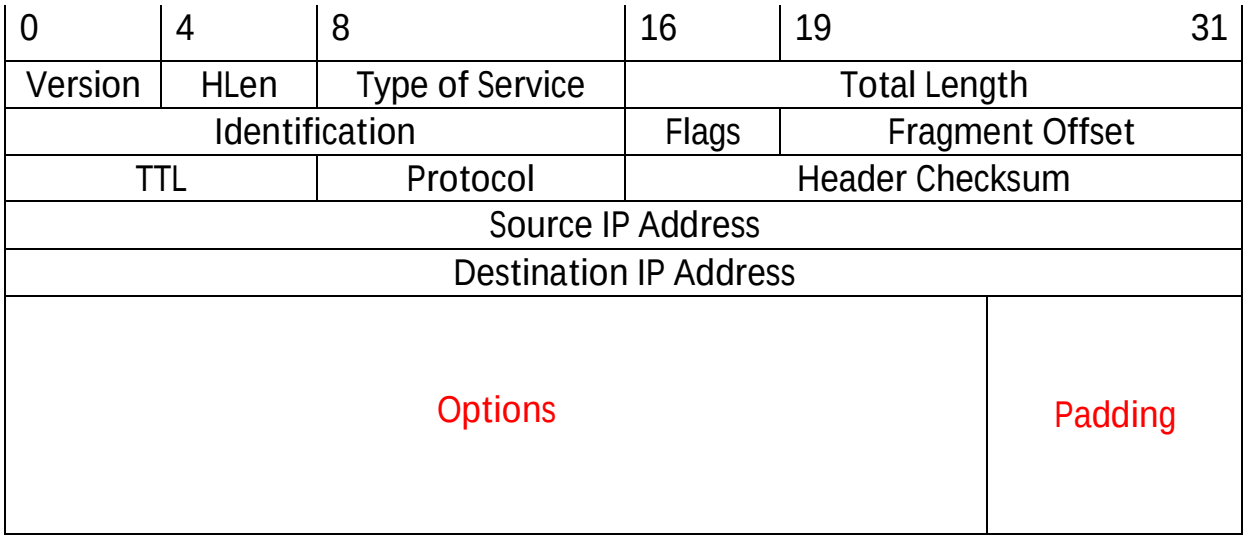
```
void ETH_Process(ENC28J60_Frame* encFrame)
{
  uint16_t  responseSize = 0;
  uint16_t  requestSize = ENC28J60_ReceiveFrame(encFrame);

  if (requestSize > 0)
```

```
 {
  ETH_Frame* ethFrame = (ETH_Frame*)encFrame->data;
  uint16_t  etherType = ntohs(ethFrame->etherType);
  uint16_t  ethDataLen = requestSize - sizeof(ETH_Frame);

  // ARP protocol
  if (etherType == ETH_FRAME_TYPE_ARP) // ==0x0806
  {
   responseSize = ARP_Process((ARP_Frame*)ethFrame->data, ethDataLen);
  }

  // IP protocol
  if (etherType == ETH_FRAME_TYPE_IP) // ==0x0800
  {
   responseSize = IP_Process((IP_Frame*)ethFrame->data, ethDataLen);
  }

  if (responseSize > 0)
  {
   ETH_Response(ethFrame, responseSize);
  }
 }
}
```

الان کافیه که تابع IP_Process رو پیاده سازی کنیم، قبلش باید بدونیم یک پکت IP چه شکلی هست. بالطبع این پروتکل هم هدر (Header) مخصوص به خودش رو به داده های دریافتی از لایه چهارم اضافه میکنه و اونو به لایه دو میده؛ پس مثل ARP باید در ابتدای بخش داده ها، ابتدا هدر پکت IP رو بررسی کنیم.

- توجه داشته باشید که پروتکل ARP چیزی رو از لایه چهارم یا بالاتر نمی گرفت؛ بلکه خودش مبدا تولید پکت بود ولی پروتکل IP داده هایی رو از لایه بالاتر میگیره؛ بهش هدر خودش رو اضافه میکنه (و اگه حجم داده ها زیاد باشه، ممکنه داده ها رو خرد هم بکنه که در حال حاضر بهش کاری نداریم) و میده به لایه دوم تا با فرمت فریم Ethernet ii ارسال بشه.

فرمت هدر پروتکل IP رو در جدول زیر می بینید، اعداد، نشون دهنده شماره بیت شروع هر بخش هست.



Version : در این بخش که 4 بیت اشغال کرده؛ عدد 4 بعنوان ورژن IPv4 قرار میگیره. بصورت بیتی "0100"

HLen : یا Header Length که گاهی بهش IHLen=Internet Header Length هم گفته میشه. این بخش نیز 4 بیتی هست و طول هدر پروتکل IP رو نشون میده. از اونجاییکه ممکنه بخش Option در هدر وجود نداشته باشه، لذا نیاز بوده که به طریقی طول هدر مشخص باشه. عدد واقع در HLen رو باید در 4 ضرب کنیم تا طول واقعی هدر رو بدست بیاریم؛ زیرا این عدد نشاندهنده طول هدر برحسب DWORD یا کلمات 4 بایتی هست. پس در این محل، حداقل عدد 5 (به صورت بیتی "0101") و حداکثر عدد 15 (به صورت بیتی "1111") قرار میگیره که نشون میده بخش هدر در پروتکل IP حداقل 20 بایت (بدون Option) و حداکثر 60 بایت (40 بایت برای بخش Option) میتونه باشه. همچنین متوجه شدیم که بخش Option ، متغیر و بین صفر تا حداکثر 40 بایت خواهد بود. دو بخش Ver و HLen با هم، یک بایت رو در هدر تشکیل میدن.

TOS : یا type of service ، به اندازه یک بایت؛ به صورت بیتی تفسیر میشه و پکت ها رو به کلاس های مختلف سرویس دهی تقسیم میکنه. ما فعلا در این بخش 0x00 قرار میدیم یعنی از سرویس خاصی استفاده نمی کنیم. در اسناد RFC که آخرین تغییرات پروتکل آیپی رو دارند، این قسمت خودش از دو بخش تشکیل شده. 6 بیت پرارزش با نام DSCP (Differentiated Services Code Point) و 2 بیت کم ارزش با نام ECN (Explicit Congestion Notification).

TLen : یا Total Length طول کل پکت IP شامل "هدر + داده" رو بر حسب بایت نشون داده میشه (و نه DWORD). از اونجایی که برای این بخش دو بایت در نظر گرفته شده؛ پس حداکثر 65535 بایت رو میشه در یک پکت IP ارسال کرد. مطابق استاندارد؛ گیرنده پکت IP، حدقل، باید قابلیت دریافت 576 بایت رو داشته باشه

و فرستنده، تنها در صورتی باید پکت های بزرگتر از 576 بایت رو ارسال کنه که مطمین باشه گیرنده میتونه این پکت رو دریافت کنه.

- یادتونه که گفتیم حداکثر تعداد بایت ارسالی برای فریم Ethernet ii مقدار 1500 هست و اگر این سوال براتون پیش اومده که چطور میشه بیش از این مقدار رو ارسال کرد؟!؛ باید بگیم که استاندارد IP فارغ از سخت افزار و پروتکل لایه دوم، تعریف شده و ما همچنان در فریم های ارسالی، حداکثر 1500 بایت داده خواهیم داشت. در واقع پروتکل IP مستقل از پروتکل لایه 2 طوری تعریف شده که بتونه حداکثر 65535 بایت ارسال کنه و اگر ما در لایه دوم از پروتکلی استفاده کنیم که بتواند تعداد بایت بیشتری ارسال کند؛ پروتکل IP تا 65535 بایت داده، میتونه بهش تحویل بده.

ID : یا IDentification شناسه پکت. چنانچه داده دریافتی از لایه چهارم خیلی بزرگ باشه و پروتکل IP در سمت فرستنده تصمیم بگیره اون داده ها رو به قطعات کوچکتری تقسیم (Fragmentation) و سپس ارسال کنه؛ از این شناسه به همراه بخش Fragment Offset برای شماره گذاری و شناسایی ترتیب پکت های ارسالی استفاده میشه. ما از این قابلیت استفاده نمی کنیم. در نتیجه درون اون میشه صفر یا هر عدد دیگه ای گذاشت.

Flags : یا پرچم ؛ این بخش تنها شامل 3 بیت هست، بیت پر ارزش (اولین بیت از سمت چپ) همواره '0' هست. بیت دوم و سوم بترتیب اسمشون DF و MF هست

DF مخفف Don't Fragment هست. فرستنده این بیت، با یک کردن اون، به اجزای میانی شبکه مثل روتر ها اعلام میکنه که اجازه خرد کردن (Fragmentation) داده های ارسالی رو ندارند لذا اگر روتری این بسته رو دریافت کنه و نتونه بدون خرد کردن اون رو به مقصد برسونه، پکت رو کنار میذاره (discard it!)

MF : مخفف More Fragment؛ هنگامی که داده های اصلی خرد میشن، این بیت '1' میشه تا به گیرنده بفهمونه که این بسته هنوز تموم نشده و قسمت های (Fragment) بیشتری در راهند. هنگام ارسال آخرین قسمت، این بیت '0' میشه.

در یک ارتباط عادی از طریق IP ما به این پرچم ها نیاز نداریم و اونها رو با 0 پر می کنیم.

Fragment Offset : چنانچه بسته ای خرد بشه؛ این قسمت نشوندهنده آفست بخش ارسالی از ابتدای داده اصلی است. این بخش، در صورت استفاده، با بخش ID استفاده میشه. از اونجایی که بخش داده ی اصلی ، حداکثر اندازه ای برابر با 65535 داره (16 بیت) و این بخش تنها 13 بیت رو شامل میشه؛ عدد واقع در این بخش رو باید در 8 ضرب کنیم. همینطور بیاد داشته باشید که آفست اولین قسمت (Fragment) ارسالی صفر خواهد بود.

اجازه بدید مثالی بزنیم، فرض کنید یک داده 1200 بایتی به قسمتهای 576 بایتی تقسیم شده تا ارسال بشه. یک ID ثابت و تصادفی برای این قسمت ها در نظر گرفته میشه، مثلا 1001 ، حالا اولین قسمت با ID=1001 و Offset=0 و MF='1' ارسال میشه ، دومین قسمت، مجدد با ID=1001 اما با offset=72 ارسال خواهد شد(576/8=72) و MF='1' ؛ در انتها و هنگام ارسال آخرین بخش ID=1001 و MF='0' و offset=144 خواهد بود. باز یادآوری کنیم که ما از این قابلیت استفاده ای نخواهیم کرد و اطلاعات فوق در اینجا فقط جهت اطلاع ثبت شده.

TTL : یا Time To Live که میشه "زمان زنده بودن" ترجمه ش کرد. با اینکه بنظر میرسه جنس این قسمت، از نوع زمان باشه؛ اما در واقع این بخش، حداکثر تعداد مجاز دستگاه های واسط (روتر یا مسیریاب) در بین مسیر از فرستنده تا گیرنده رو مشخص میکنه. این بخش برای این در نظر گرفته شده که اگر پکتی به هر دلیلی به مقصد نرسید؛ تا ابد درون شبکه باقی نمونه. فرستنده، TTL رو با یک عدد مطلوب مثلا 128 یا 69 پر میکنه. هنگام ارسال و جابجایی بین زیرشبکه ها، هر وقت این پکت وارد روتری بشه؛ روتر از عدد TTL یکی کم میکنه و اگر نتیجه صفر باشه، این پکت رو دور میندازه (discard)! ولی یه پیغام خطا (با استفاده از پروتکل ICMP) به فرستنده برمیگردونه.

- عملیات trace route که همراه با عملیات ping معرفی خواهند شد! از این آیتم استفاده میکنه تا نشون بده یک پکت در راه رسیدن به مقصد از کدام روترها عبور کرده. ضمیمه [4] رو ببینید!

Protocol : این آیتم یک بایتی، نشوندهنده پروتکلی از لایه چهارمه که داره از این پکت استفاده میکنه (پروتکلی در لایه 4 که داده ها متعلق به اون پروتکل هستند). سه تا از پروتکل های لایه چهارم که ما قصد داریم در این نوشتار ازشون استفاده کنیم؛ شماره های شناساییشون عبارتست از:

ICMP = 0x01
UDP = 0x11
TCP = 0x06

CheckSum : دیدیم که در فریم اترنت، برای بررسی صحت ارسال و دریافت داده ها، در انتهای فریم، بخشی بنام CRC اضافه میشد. علاوه بر این، اکثر پروتکل های لایه های بالاتر مثل IP,ICMP,UDP و TCP بطور جداگانه، درون هدرشون چند بایت برای بررسی مجدد سلامت اطلاعات، قرار میدن که معمولا محسباتشون بر اساس فرآیند چک سام هست. از اونجاییکه این محاسبات مثل هم هست (فقط داده هایی که روشون عملیات چک سام انجام میشه متفاوته) لذا در همین جا و بعد از معرفی بخش هدر (Header)، توضیح نسبتا کاملی از روش انجام محاسبه چک سام ارایه میدیم. در پروتکل IP محاسبه چک سام فقط روی هدر انجام میشه در حالیکه

در بعضی دیگه ممکنه علاوه بر هدر؛ داده ها payload و یا بخش های دیگه ای؛ در هنگام محاسبه چک سام استفاده بشوند.

Source & Destination IP : در این دو بخش به ترتیب آدرس IP فرستنده و گیرنده قرار میگیره.

Options+Padding : بطور معمول از بخش Option برای ارسال گزینه های اختیاری یا تنظیمی استفاده میشه. ما از این بخش استفاده ای نمی کنیم، در نتیجه اندازه هدر IP همواره برای ما 20 بایت خواهد بود و بخش HLen با عدد 5 پر میشه (همونطور که گفتیم Hlen بر اساس داده های 4 بایتی محاسبه شده؛ پس عدد داخل HLen باید در 4 ضرب بشه تا طول هدر بر حسب بایت بدست بیاد). از طرفی، چنانچه از بخش Option استفاده بشه و مجموع بایت های گزینه های استفاده شده، مضربی از 4 نباشند؛ در انتهای بخش Option عملیات padding انجام میشه؛ یعنی با صفر پر میشه تا کل بایت های بخش Option مضربی از 4 باشند.

- استاندارد IP در RFC 791 و بروزرسانی (update) های اون اومده.

محاسبه چک سام :

بطور کلی فرآیند چک سام اینطوری انجام میشه: در بخشی که میخوان چک سام رو حساب کنن؛ تمام داده ها به ترتیب خاصی با هم جمع میشن و در نهایت نتیجه این جمع با یک روال تعریف شده، در محل چک سام نوشته میشه.

- **گفتیم که در پروتکل IP چک سام روی هدر انجام میشه. از اونجاییکه بخش (field) چک سام، خودش داخل هدر هست؛ قبل از انجام چک سام، ابتدا داخل این بخش، عدد صفر نوشته میشه؛ بعد محاسبات انجام میشه و در آخر، نتیجه در همین قسمت نوشته میشود.**

برای تعریف چک سام اینترنتی، در اسناد اینطور گفته شده:

The checksum field is the 16 bit one’s complement of the one’s complement sum of all 16 bit words.

یعنی "چک سام نتیجه 16 بیتی و به فرم مکمل یک (one’s complement) جمع همه داده ها (یی که قراره در محاسبات چک سام وجود داشته باشند) به صورت 16 بیتی"

توضیح ساده ترش اینه: بایت هایی که باید چک سامشون محاسبه بشه رو به صورت 16 بیتی (دو بایتی) جدا می کنیم، این دو بایتی ها رو به فرم مکمل یک جمع میکنیم؛ نتیجه جمع رو مکمل یک می گیریم و در فیلد

checksum می نویسیم. به عنوان مثال اگر نتیجه جمع شد 11+ ؛ در بخش مربوطه 11- مینویسم (اعداد به فرم مکمل یک خواهند بود).

حالا چند تا نکته :

- فرم مکمل یک، خیلی ساده ست؛ در اعداد منفی، تمام بیت ها رو NOT میکنیم! مثلا عدد 7+ به صورت 16 بیتی؛ میشه 0000,0000,0000,0111 و عدد 7- به فرم مکمل یک میشه 1111,1111,1111,1000

- در فرم مکمل یک؛ ما دو تا عدد به عنوان صفر داریم ، اگر به صورت 16 بیتی در نظر بگیریم، این دو تا صفر با 0x0000 و 0xFFFF نشون داده میشن، برای تمایز این دو، به اولی +0 و به دومی -0 میگن.

- در محاسبه جمع اعداد به فرم مکمل یک ، اگر رقم نقلی (carry) تولید بشه، برای درست بودن جواب، باید مقدار رقم نقلی رو به عدد حاصل اضافه کنیم. بعنوان مثال 5-7+ رو اگه محاسباتمون 4 بیتی باشه:

```
    0111    (+7)
+   1010    (-5)          //+5==0101 => -5== 1010
-------------
1   0001
```

همونطور که میبینیم اینجا عدد 1 رو به عنوان رقم نقلی داریم؛ لذا با نتیجه اصلی که "0001" هست جمع میشه که میشه 0010 یا 2 که این، نتیجه ی درست محاسبه هست.

- محاسبات به فرم مکمل یک، یکسری خصوصیات جالب هم داره که اونها رو در شکل زیر و با سه روش نشون دادیم. فرض کنید داده هایی که باید روشون چک سام انجام بشه از چپ به راست به فرمت شانزده شانزدهی (Hexadecimal) عبارتند از :

00,01,f2,03,f4,f5,f6,f7

	بایت به بایت		دو بایتی با ترتیب نرمال	دو بایتی با ترتیب معکوس
Byte 0,1	00	01	0001	0100
Byte 2,3	F2	03	F203	03F2
Byte 4,5	F4	F5	F4F5	F5F4
Byte 6,7	F6	F7	F6F7	F7F6
جمع اول	2DC	1F0	2DDF0	1F2DC
نقلی	1	2	2	1
جمع دوم	DD	F2	DDF2	F2DD
جابجایی نهایی	DDF2		DDF2	DDF2

حالت اول اینه که بایت های فرد با هم و بایت های زوج با هم جمع میشن. حالت دوم اینه که بایت ها به فرم Big Endian با هم جمع شدن(حالت نرمال شبکه) و حالت سوم اینه که ابتدا بایت های درون داده 16 بیتی، جابجا (swap) شدن (Little Endian).

اگر محاسبات چک سام در نرم افزار (مثلا در نرم افزار میکروکنترلر) پیاده سازی بشه؛ سرعت انجام اون بسیار اهمیت داره. در سند RFC 1071 و آپدیت های اون، بعضی الگوریتم های محاسبه چک سام با توجه به خصوصیات محاسبه به فرم مکمل یک پرداخته و بررسی شده.

یادآوری کنیم که چک سام در پروتکل IP، تنها روی بایت های واقع در هدر انجام میشه.

- در سیستم عامل ویندوز و همچنین در کدنویسی میکروکنترلرها ، محاسبات به فرمت مکمل دو (two's complement) انجام میشن، عدم توجه به این نکته، عامل بسیاری از باگ های نرم افزاری ست.

قبل از اینکه برگردیم سراغ پروتکل IP، نیاز هست که یه تابع بنویسیم برای محاسبه چک سام. از اونجایی که داده هایی که توسط لایه دوم دریافت شده ن، درون یک آرایه به فرمت uint8_t ذخیره شدن، تابعی که مینویسیم دو آرگومان داره؛ اولی یک اشاره گر به ابتدای داده ها و دومی، طول داده ها رو میگیره و یک نتیجه 16 بیتی که همون چک سام هست، بما برمی گردونه.

```
uint16_t  IP_CalcCheckSum( uint8_t*  data, uint16_t  len)
{
 uint32_t res = 0;
 uint16_t * ptr = ( uint16_t* ) data;
```

```
 while ( len > 1 )
 {
  res += *ptr;
  ptr ++ ;
  len -= 2 ;
 }
 if ( len > 0 )
 {
  res += * ( uint8_t* ) ptr;
 }
 while ( res > 0xffff )
 {
  res = ( res >> 16 ) + ( res & 0xFFFF ) ;
 }
 return ~ (( uint16_t ) res ) ;   // or XOR with 0xFFFF
}
```

چون در عملیات جمع، ممکنه رقم نقلی (carry) داشته باشه؛ برای ذخیره نتیجه، یک متغیر 32 بیتی در نظر گرفته شده. در ابتدای تابع، بایت ها رو دوتا دوتا با هم جمع می کنیم، اگر تعداد بایت ها فرد باشه؛ در نهایت یک بایت اضافه میماند، اون رو هم جمع میکنیم. از اونجاییکه عملیات جمع ممکنه رقم نقلی داشته باشه و نتیجه بیشتر از 16 بیت بشه، مقدار نقلی رو با نتیجه اصلی، جمع میکنیم (برای فهم ساده تر الگوریتم؛ کد به صورت بالا نوشته شده). در نهایت دو بایت نتیجه جمع، به صورت بیتی NOT شده ن (تا مکمل یک جواب جمع بدست بیاد) و اینو به عنوان نتیجه نهایی چک سام برمیگردونیم.

- باز هم یادآوری میکنیم که این کدها در میکروکنترلر، به فرم مکمل دو انجام میشن و اگر تفاوتی در کد و الگوریتم گفته شده میبینید به این علت هست. خودتون میتونید به صورت دستی، محاسبات رو انجام بدید تا از صحت کد مطمین بشید.
- یادمون نره که قبل از فراخوانی تابع چک سام؛ فیلد چک سام رو صفر کنیم.

در سمت گیرنده، طبق الزام استاندارد، بایستی چک سام بررسی بشه (فرستنده هم ملزم به ارسال اون هست) در حالیکه در بسیاری از کدهای داخل نت، چک سام محاسبه یا بررسی نشده! در اینجا (یعنی سمت گیرنده) دو کار میتونیم انجام بدیم، راه ساده تر اینه که اینجا هم چک سام رو حساب کنیم و ببینیم که آیا یکی هستند یا نه!

راه دوم که بنظر سریعتر هست؛ اینه که در سمت گیرنده فیلد چک سام رو صفر نکنیم و محاسبات رو انجام بدیم، در نتیجه جواب نهایی، همواره باید صفر باشه. چرا؟ یادمونه که ما در آخرین مرحله از محاسبه چک سام؛ مکمل یک چک سام رو ثبت می کردیم؛ مثلا اگر چک سام میشد 6+ ؛ ما مقدار 6- رو ذخیره می کردیم ، در نتیجه در

سمت گیرنده، اگر بدون صفر کردن چک سام، محاسبات رو در سمت گیرنده انجام بدیم، باید به نتیجه صفر برسیم.

نکاتی در مورد بخش Option و فرمت TLV :

هرچند از بخش option در پروتکل IP فعلا استفاده ای نمی کنیم اما بعضی موارد رو خوب هست که اینجا توضیح بدیم قبل از اینکه از این بخش بگذریم؛ چون این بخش هم مثل چک سام در بسیاری از پروتکل ها وجود داره. در پروتکل هایی که در هدر، بخش Option (گزینه) با طول نامشخص دارند، گزینه ها به دو صورت ارسال میشن:

- گزینه هایی که فقط بودن یا نبودنشان حایز اهمیت هست و مقدار تنظیم شونده ندارند. این گزینه ها تنها یک بایت رو اشغال می کنند.
- گزینه هایی که دارای مقدار تنظیم شونده هستند و چند بایت را اشغال می کنند. در این حالت، اولین بایت نوع گزینه (Type) رو مشخص میکنه؛ دومین بایت اندازه (Length) گزینه رو مشخص میکنه و مقادیر (Value) گزینه از بایت سوم به بعد قرار میگیرند.

در پروتکل IP دو گزینه داریم که فقط بودن یا نبودنشون مهمه و یک بایتی هستند:

گزینه	مقدار داخل بایت
End of Option List = انتهای لیست گزینه ها	0x00
No Operation = بدون عملیات	0x01

گزینه "انتهای لیست گزینه ها " معمولا در آخر گزینه ها قرار میگیره و همون padding هست. چنانچه بخش Option دقیقا مضربی از 4 باشه، نیازی به استفاده از این گزینه نیست.

گزینه "بدون عملیات" عموما در بین گزینه ها استفاده میشه؛ برای اینکه شروع گزینه بعدی دقیقا روی آدرس با مضرب 4 بیفته (برای راحتی پردازش). توجه داشته باشید که هر دوی این گزینه ها اختیاریست و نوع استفاده از اون به نظر برنامه نویس بستگی داره.

حالت دیگه برای Option اینه که همونطور که گفتیم چند بایتی و به فرم زیر باشن. بایت اول نوع (Type or Kind) گزینه، بایت دوم طول بایت های اشغال شده توسط گزینه (Length) و بایت های سوم به بعد اطلاعات

یا مقادیر گزینه(Value). در بعضی پروتکل ها، در بایت دوم یعنی "طول بایت های گزینه" دو بایت نوع و طول هم به حساب میان؛ لذا در این حالت، کمترین مقدار اندازه گزینه، عدد 2 خواهد بود؛ بدین معنی که گزینه فاقد داده یا مقدار هست!

بعنوان مثال فرض کنید گزینه ی نوع 7 ، دارای سه بایت داده با ارقام 0x0A,0x0B,0x0C هست؛ لذا در فرم اول این گزینه، اینطوری پیاده سازی میشه 0x07,0x03,0X0A,0X0B,0X0C و در حالت دوم به صورت 0X07,0X05,0X0A,0X0B,0X0C پیاده سازی خواهد شد. به این فرمت اصطلاحا TLV گفته میشه که سرنام کلمات Type,Length,Value هست و پروتکل IP از فرم دوم اون استفاده میکنه (یعنی در بخش Length تعداد بایت های نوع و طول هم لحاظ میشه).

همونطور که گفتیم (بسته به تعریف استاندارد پروتکل) ممکنه گزینه هایی داشته باشیم که مقدار تنظیم شونده ندارند اما از فرمت TLV استفاده می کنند؛ مثلا گزینه شماره 2 در فرم اول به صورت 0X02,0X00 و در حالت دوم بصورت 0X02,0X02 ثبت خواهد شد.

مجدد یادآوری میکنیم؛ چنانچه تعداد کل بایت های یک گزینه مضرب 4 نباشه، معمولا در انتهای این گزینه، از گزینه شماره 1 یا همون No operation استفاده میشه تا هم بین گزینه ها فاصله بندازه و هم گزینه بعدی در آدرسی با مضرب 4 شروع بشه.

و در انتهای گزینه ها هم چنانچه جای خالی باقی مونده باشه (بر اساس مقدار HLen)؛ گزینه شماره 0 یا همون End of Option List میتونه استفاده بشه.

مثال: فرض کنید ما دو گزینه با شماره های 8 و 9 داریم که هر کدام فقط یک بایت مقدار (Value) با مقدار 0x0A دارند. در پروتکل IP این دو گزینه به ترتیب بصورت 0x08,0x03,0x0A و 0x09,0x03,0x0A خواهد بود؛ در نتیجه قسمت Option میتونه همچین شکلی داشته باشه

0x08,0x03,0x0A,0x01,0x09,0x03,0x0A,0x00

از سمت چپ سه بایت اول مربوط به گزینه شماره 8 هستند، از اونجایی که این گزینه، 3 بایت داره؛ برای اینکه گزینه بعدی در آدرس مضرب 4 قرار بگیره یه گزینه 0x01 یا همون No Operation در بایت چهارم قرار گرفته، 3 بایت بعدی گزینه شماره 9 هست و از اونجایی که گزینه دیگه ای نداریم؛ همچنین یک بایت باید اضافه کنیم تا کل بایت های قسمت Option مضربی از 4 باشه؛ در انتهای بایت ها یه بایت 0x00 که همون End of Option List یا padding هست، اضافه شده. اینم فراموش نکنیم که در این وضعیت مقدار HLen عدد 7 خواهد بود (5 برای بخش اصلی هدر بعلاوه 2 که از تقسیم 8 بایت گزینه ها به 4 بدست میاد).

یا اینکه اینطوری باشه:

0x08,0x03,0x0A,0x09,0x03,0x0A,0x00,0x00

همونطور که می بینید؛ دیگه بین گزینه ها از No Operation استفاده نشده؛ در عوض مجبور شدیم در آخر بایت ها، دو بایت 0x00 برای padding داشته باشیم.

- ترتیب قرار گرفتن گزینه ها در بخش Option اهمیتی ندارد. در مثال بالا میتونستیم اول گزینه 9 رو قرار بدیم و بعد گزینه 8 رو.
- برنامه نویس در اجرای کد(Implementation) بایست کد رو طوری بنویسه که بتونه هر نوعی از ارسال بخش Option ها رو بخونه و درک کنه.
- چنانچه گزینه ای برای گیرنده ناشناخته باشه؛ عموما دور ریخته میشه و در عوض پیغام خطایی (معمولا با پروتکل ICMP) به فرستنده ارسال میشه.
- در بعضی پروتکل ها، استاندارد میگه که در نرم افزار گیرنده؛ دریافت و پردازش تعدادی از گزینه ها الزامیه که این الزامات در فایل های استاندارد RFC مربوط به هر پروتکلی اومده. پردازش مابقی گزینه ها اختیاریست. در پروتکل IP، پردازش دو گزینه 0x00 و 0x01 الزامیست.

برگردیم سراغ کدنویسی. تا اینجا پیش رفتیم که داخل تایع ETH_Process به صورت زیر تست کردیم ببینیم آیا فریم دریافتی شامل پروتکل IP هست یا نه:

```
// IP protocol
if (etherType == ETH_FRAME_TYPE_IP) // ==0x0800
{
  responseSize = IP_Process((IP_Frame*)ethFrame->data, ethDataLen);
}
```

مثل وضعیتی که در برخورد با پروتکل ARP داشتیم در اینجا نیز برای تفکیک بخش های مختلف هدر و داده در پروتکل IP یک ساختار (struct) به صورت زیر تعریف می کنیم:

```
typedef  struct  IP_Frame
{
 uint8_t  verHeaderLen;
 uint8_t  diffServices;
 uint16_ t  len;
 uint16_t  fragId;
 uint16_t  fragOffset;
```

```
 uint8_t   timeToLive;
 uint8_t   protocol;
 uint16_t   checkSum;
 uint8_t   srcIpAddr [ IP_ADDRESS_BYTES_NUM ] ;
 uint8_t   destIpAddr [ IP_ADDRESS_BYTES_NUM ] ;
 uint8_t   data [] ;
} IP_Frame ;
```

اگر خواسته باشید در اجرای پروتکل IP بخش Options رو هم در نظر بگیرید؛ از اونجاییکه طول این بخش نامعین هست؛ لذا اون رو هم داخل آرایه data ببینید. طبیعیه که در این حالت، با توجه به عدد واقع در بخش Hlen اندازه بخش هدر مشخص میشه، اگر این عدد بزرگتر از 20 بایت باشه (HLen=5) در نتیجه بخش ابتدایی data مربوط به قسمت Options هست.

و اما پیاده سازی تابع IP_Process:

از اونجایی که به جهت سادگی در ارایه مفهوم از پردازش بخش Option صرفنظر کردیم و همینطور از قابلیت Fragmentation در این پروتکل استفاده نمی کنیم؛ لذا تعریف تابع IP_Process() هم بصورت زیر انجام شده:

```
uint16_t  IP_Process ( IP_Frame*  ipFrame, uint16_t  frameLen )
{
 uint16_t newFrameLen = 0;

 if (memcmp(ipFrame->destIpAddr, ipAddr, IP_ADDRESS_BYTES_NUM) == 0)
 {
  uint16_t  rxCheckSum = ipFrame->checkSum;
  ipFrame->checkSum = 0;
  uint16_t  calcCheckSum = IP_CalcCheckSum((uint8_t*)ipFrame, sizeof(IP_Frame));

  if (rxCheckSum == calcCheckSum)
  {
   uint16_t  dataLen = ntohs(ipFrame->len) - ((ipFrame->verHeaderLen & 0x0F)*4);
   uint16_t  newDataLen = 0;

   if (ipFrame->protocol == IP_FRAME_PROTOCOL_ICMP)  //=0x01
   {
       //do something for ICMP
   }
   else if (ipFrame->protocol == IP_FRAME_PROTOCOL_UDP)       //=0x11
   {
```

```
        //do something for UDP
    }
    else if (ipFrame->protocol == IP_FRAME_PROTOCOL_TCP)        //=0x06
    {
        //do something for TCP
    }

    if(newDataLen)
        newFrameLen = newDataLen + sizeof(IP_Frame);
    else
        newFrameLen=0;
    if(newFrameLen)
    {
      ipFrame->len = htons(newFrameLen);
      ipFrame->fragId = 0;
      ipFrame->fragOffset = 0;

    //swap IP Addresses for reply
     memcpy(ipFrame->destIpAddr, ipFrame->srcIpAddr, IP_ADDRESS_BYTES_NUM);
     memcpy(ipFrame->srcIpAddr, ipAddr, IP_ADDRESS_BYTES_NUM);

    //calculate new checksum
     ipFrame->checkSum = IP_CalcCheckSum((uint8_t*)ipFrame, sizeof(IP_Frame));
    }
   }
  }
  return newFrameLen;
}
```

در گام اول با بررسی آدرس آیپی موجود در پیام؛ بررسی میکنیم که آیا پیام فعلی برای ما ارسال شده یا نه؟ قدم بعدی، محاسبه چک سام و بررسی تساوی چک سام دریافتی و چک سام محاسبه شده است. اگر هر دو بخش رو رد کردیم؛ حالا باید بررسی کنیم کدام پروتکل در لایه چهارم این پیام رو فرستاده؛ سه پروتکل ICMP,UDP و TCP برای ما اهمیت دارند؛ پس بخش پروتکل از هدر IP رو بررسی میکنیم و عملیات مناسب رو برای پردازش این پروتکل ها انجام خواهیم داد. از اونجاییکه این پروتکل ها رو هنوز معرفی نکردیم؛ تنها هسته مناسب برای پیاده سازی اونها رو در کد میبینیم. اینجا دوباره شماره پروتکل های لایه چهارم رو نوشتیم:

ICMP = 0x01

UDP = 0x11

TCP = 0x06

در انتهای تابع هم اگر پکت دریافتی، نیاز به ارسال پاسخ داشته باشه (با بررسی اندازه پاسخ) جای آدرس های IP فرستنده و گیرنده رو تغییر میدیم؛ بیت های متعلق به Fragmentation رو پاک میکنیم و چک سام جدید رو محاسبه میکنیم تا چینش هدر IP انجام شده باشه و در نهایت میزان داده های پاسخ رو برمیگردونیم. اگر این عدد صفر نباشه به این معنی هست که پیام IP دارای پاسخ هست؛ لذا لایه دوم هم که از قبل کدهاش رو نوشتیم، پیغام رو ارسال خواهد کرد.

اگه بخوایم وضعیت این پروتکل ها رو در مدل OSI یا TCP/IP نشون بدیم مثل جدول زیر باید باشه:

لایه 4 (Transport Layer)	TCP(segment)	UDP(segment)	ICMP	
لایه 3 (Network Layer)	IP (packet)			ARP
لایه 2 (Data Link Layer)	Ethernet ii (frame)			
لایه 1 (Physical Layer)	‘0’ , ‘1’ (Stream)			

- تعداد پروتکل های موجود در هر لایه بسیار بیشتر از این جدول هست و ما مهمترین و اصلیترین ها رو معرفی کردیم.
- به جریان بیت های ارسالی لایه یک، stream میگفتیم؛ داده های ارسالی در لایه دوم رو frame نامگذاری کردیم؛ داده لایه سوم رو packet و داده ارسالی لایه چهارم رو هم segment میگن. لایه چهارم در مدل OSI بنام لایه انتقال (Transport یا Transmission) شناخته میشه و به وقتش به بررسی اونها خواهیم پرداخت.

اولین لایه از بالا که قادر به ارسال داده خام هست، لایه چهارمه. مثلا فرض کنید بردی طراحی کردید که قراره اطلاعات یک تعداد سنسور دما که شماره گذاری شده ن رو به یک کامپیوتر منتقل کنه. در این حالت میتونید از دو پروتکل UDP یا TCP برای ارسال اطلاعات متنی، مثل رشته زیر استفاده کنید.

“T1=10.1,T2=12.3,T3=14.1”

پروتکل ICMP(Intenet Control Message Protocol) همونطور که از اسمش پیداست؛ جهت ارسال/دریافت پیام های کنترلی شبکه استفاده میشه(مثل موارد خطا). دو عملیات ping و trace route در شبکه با استفاده از قابلیت های پروتکل ICMP پیاده سازی میشن که بهشون اشاره خواهیم کرد و در ادامه ICMP اولین پروتکلی هست که بعد از IP میریم سراغش. پروتکل ICMP برای این بوجود اومده که اگر یک ارتباط IP دچار مشکل شد؛ بشه به فرستنده یا ادمین شبکه اطلاع داد.

اما برای ارسال داده خام از دو پروتکل UDP(User Datagram Protocol) یا TCP(Transmission Control Protocol) استفاده میشه. هر گاه در تبادل اطلاعات زمان بر صحت و سلامت ارسال، ارجحیت داشته باشه از UDP و هرگاه صحت و سلامت تبادل اطلاعات بر زمان ارجح باشه از TCP استفاده میشه. در UDP، داده ها، فقط ارسال میشن؛ بدون اینکه بررسی بشه آیا بطور صحیح به مقصد رسیدن یا نه؟ (اصلا رسیدن یا نه) اما در TCP، صحت رسیدن اطلاعات به مقصد بررسی میشه و چنانچه خطایی رخ بده با استفاده از قابلیت هایی مثل بازارسال (retransmission)؛ داده ها رو مجددا ارسال می کنند. با اینکه هر دو پروتکل به صورت کلاینت/سرور (مشتری/خدمات دهنده) در نظر گرفته میشه؛ اما هر دو سمت شبکه، میتونن داده هایی ارسال یا از طرف مقابل دریافت کنند.

فرض کنید شما در حال استفاده از یک دوربین با رزولوشن بالا هستید و سرعت نمونه برداری و ارسال فریم های تصویر هم بالاست. در این حالت بهتره از UDP استفاده بشه زیرا گم شدن یک یا دو فریم از تصویر اهمیتی نداره. بعنوان یک مثال دیگه از این پروتکل، میشه به ارسال اطلاعات دما یا حتا سیگنال دیجیتال یک میکروفون اشاره کرد زیرا در هر دو حالت، اگر یک یا چند نمونه از بین بروند و به مقصد نرسند؛ اهمیتی نداره؛ اما وقتی که شما در حال ارسال اطلاعات یک دیتابیس (database یا پایگاه داده) هستید یا اینکه دارید اطلاعات یک صفحه وب رو ارسال میکنید؛ در این حالت، صحت ارسال و دریافت بسیار مهم هست و درنتیجه، در این جا از پروتکل TCP استفاده میشه.

- با بررسی وضعیت پیغام های موجود روی خط با wireshark خواهیم دید که در هر لحظه پیغام های زیادی روی خط ردوبدل میشن که مقصد تعداد زیادی از اونها برد ما نیست؛ در یک شبکه واقعی، تعداد زیادی از این پیام ها توسط تجهیزات میانی مثل روترها و سوییچ ها فیلتر میشن. اما در وضعیت فعلی که ما با یک کابل بطور مستقیم به PC متصل هستیم؛ برای اینکه تعداد پیام های نامرتبط رو کاهش بدیم؛ پیشنهاد میشه که فیلترهای ENC رو فعال کنیم تا از دریافت تمامی پیغام ها و پردازش اضافه در میکروکنترلر جلوگیری بشه.

پروتکل ICMP :

پروتکل ICMP پروتکلی ست برای مدیریت، کنترل و همچنین گزارش خطاهایی که احتمالا در یک ارتباط بوجود خواهند آمد. در نتیجه این پروتکل، کاربردهای زیادی داره. یکی از پرکاربردترین وظایف، بررسی برقراری ارتباط با یک هاست دیگه در شبکه است. به این عملیات اصطلاحا پینگ (ping) گفته میشه. در عملیات پینگ، داده هایی (عموما با طول و مقدار ثابت که وابسته به پیاده سازی است) برای هاست موردنظر ارسال میشه و انتظار میره همان داده ها، به ما برگردانده (echo) بشه. در این فرآیند، اطلاعاتی مثل زمان تقریبی دسترسی به هاست موردنظر، محاسبه و گزارش می شود. ما پروتکل ICMP رو فقط جهت پردازش عملیات ping؛ اون هم تنها در حالت پاسخ دهی پیاده سازی میکنیم. برای همین، برد ما قادر به ارسال دستور پینگ نیست و فقط میتونه به درخواست پینگ جواب بده. بیشتر از این هم در حال حاضر نیاز نیست. یکی دیگر از دستوراتی که از ICMP استفاده میکنه تا اطلاعات روترهای بین دو هاست رو بدست بیاره؛ Treace Route هست که در ضمیمه ها معرفی خواهیم کرد.

همچنین در صفحات قبلی این نوشتار گفتیم که بعضی مواقع در صورت رخداد خطا یا بوجود آمدن وضعیت خاصی در شبکه، پیغامی برای فرستنده یا ادمین (مدیر ، Admin) شبکه ارسال میشه؛ (مثلا هنگامی که یک روتر با TTL=0 مواجه میشه)؛ گفتیم که عموما بسته دور ریخته میشه؛ ولی به فرستنده اطلاع داده میشه. اینگونه گزارشات، توسط پروتکل ICMP انجام میشه. دو وضعیت دیگه وقتیه که هنگام مسیریابی توسط روترها، مسیری برای رسیدن به زیرشبکه مقصد نیست یا اینکه پکت به زیرشبکه مقصد رسیده؛ اما هاست مورد نظر در اون زیرشبکه وجود نداره. در این دو وضعیت هم پیغام هایی از طرف روتر نهایی به فرستنده ارسال میشه با نام های Destination network unreachable و Destination host unreachable که ارسال این دو پیغام هم توسط ICMP انجام میشه. وقتی هم که پیغام به دلیل TTL=0 دور ریخته میشه؛ پیغامی با کد Time exceeded به فرستنده برمیگرده. جهت مشاهده مابقی پیغام ها به سند RFC 792 رجوع کنید. یادتون باشه که ICMP قابلیت پیاده سازی این پیام ها رو دراختیار میذاره، ولی کد مربوطه، باید در پردازنده پیاده سازی (Implementatio) شده باشه. به عنوان مثال اگر در میکروکنترلرتون عملیات ping رو ننوشته باشید، بالطبع درخواست یا پاسخ پینگ رو نمیتونید اجرا کنید و برد شما به پینگ جواب نخواهد داد.

ابتدا ببینیم هدر (Header) در پروتکل ICMP چه شکلی هست :

Octet Offset	0	1	2	3
0	Type	Code (sybtype)	CheckSum	
4	Rest of Header			

Type & Code : دو بایت برای نوع (type) و زیرنوع (subtype) یا همان کد در پیام کنترلی

Checksum : چک سام در پروتکل ICMP. محاسبه چک سام برای پروتکل IP رو کمی قبل تر گفتیم، چک سام در ICMP هم مثل IP هست؛ با این تفاوت که در اینجا چک سام روی هدر و تمام داده ها انجام میشه. در نتیجه از همون تابعی که برای محاسبه چک سام IP استفاده کردیم؛ می توان اینجا هم استفاده کرد. وقتی یک پیام به فرمت ICMP ارسال میشه؛ در واقع دارای دو چک سام هست؛ یکی در لایه سوم برای IP و دیگری در لایه چهارم برای ICMP.

Rest of Header : شامل 4 بایت که بر اساس بخش های نوع و زیرنوع تفسیر می شوند.

بعد از بخش هدر ، داده های این پروتکل قرار می گیرند. البته تعداد و مقدار این داده ها، وابسته به عملیاتی ست که در حال استفاده از ICMP است؛ پس ممکنه این پروتکل داده ای نداشته باشه یا دارای داده هایی با مقادیر خاص باشه.

عملیات ping :

یک روش ساده برای اطلاع از اینکه ارتباط بین دو هاست به درستی برقرار هست؛ استفاده از عملیات (یا دستور) ping هست. این پیغام به صورت درخواست/پاسخ (Request/Reply) پیاده سازی میشه. فرستنده از گیرنده درخواست میکنه، اطلاعاتی که برای گیرنده ارسال میشه رو به فرستنده برگردونه (echo). این عملیات با استفاده از پروتکل ICMP اجرا میشه.

اینکه نامگذاری ping از کجا اومده؛ در بعضی جاها منجمله دانشنامه ویکیپدیا گفته شده که این نام از عملیات بازتاب امواج سونار در زیردریایی ها یا اژدرها گرفته شده (نام echo نشون دهنده این وضعیت هست). توصیف دیگه ای که دیدم و به نظرم جهت آشنایی با مفهوم پینگ به ذهن آشناتر هست، گفته بود این اسم از ورزش پینگ پنگ (نام قدیمی ورزش تنیس روی میز که بنظر اصطلاحی به زبان چینی است) گرفته شده، همونطور که

در این ورزش مشخصه؛ توپ به دفعات بین دو بازیکن رفت و برگشت داره! در عملیات پینگ در ویندوز هم، چهار بار؛ و هربار تعداد 32 بایت با مقدار ثابت؛ به مقصد فرستاده میشه و انتظار میره که همونا برگرده. البته این فرآیند با استفاده از تنظیمات دستور پینگ، قابل تغییر هست و در سیستم عامل های دیگه مثل لینوکس هم ممکنه بطور دیگه ای پیاده سازی شده باشه.

در عملیات پینگ، بخش 4 بایتی rest of header به دو بخش دو بایتی با نام های ID,Sequence Number تقسیم میشه.

از طرفی بخش Type برای فرستنده (درخواست کننده) برابر با 0x08 و برای گیرنده (پاسخ دهنده) عدد 0x00 هست که به صورت زیر در کد میاد :

```
#define  ICMP_FRAME_TYPE_ECHO_REQUEST  0x08
#define  ICMP_FRAME_TYPE_ECHO_REPLY    0x00
```

بخش Code در هر دو حالت، 0x00 است.

بریم سر وقت کدنویسی؛ ابتدا ساختار مناسبی برای هدر ICMP تعریف میکنیم:

```
typedef  struct  ICMP_EchoFrame
{
 uint8_t  type;
 uint8_t  code;
 uint16_t  checkSum;
 uint16_t  id;
 uint16_t  seqNum;
 uint8_t  data [];
} ICMP_EchoFrame;
```

در تابع پردازشگر پکت های IP (یعنی در تابع IP_Process) ابتدا بررسی می کنیم که آیا محتویات پکت دریافتی متعلق به پروتکل ICMP هست؟ و اگر جواب مثبت بود؛ تابعی که برای پروتکل ICMP در نظر گرفتیم (و فعلا فقط درخواستهای پینگ رو جواب میده) رو فراخوانی میکنیم:

```
if (ipFrame->protocol == IP_FRAME_PROTOCOL_ICMP)      //==0x01
  {
    newDataLen = ICMP_Process((ICMP_EchoFrame*)ipFrame->data, dataLen);
  }
```

گفتیم که در عملیات پینگ؛ داده هایی که از طرف درخواست کننده ارسال میشن، عینا برگشت داده میشن(echo) فقط در این بخش مقدار type از 0x08 به 0x00 تغییر میکنه و از اونجاییکه داده های بخش مربوط به چک سام تغییر کرده ؛ بالطبع در پاسخ؛ مقدار چک سام تغییر می کنه و باید دوباره محاسبه بشه.

- لزومی نداره در هنگام محاسبه چک سام برای پکت پاسخ؛ تمامی پروسه جمع کردن ها و مکمل گیری دوباره انجام بشه و فقط کافیه یه جمع و تفریق ساده انجام بشه! اگه گفتید چرا؟ جهت اطلاع، روترها هم که ناچار به تغییر بسته IP هستند (مثل کاهش TTL) بطور معمول از این روش برای تغییر چک سام ها استفاده می کنند و دوباره چک سام رو از ابتدا حساب نمی کنند. یه کم جلوتر علت رو میگیم.

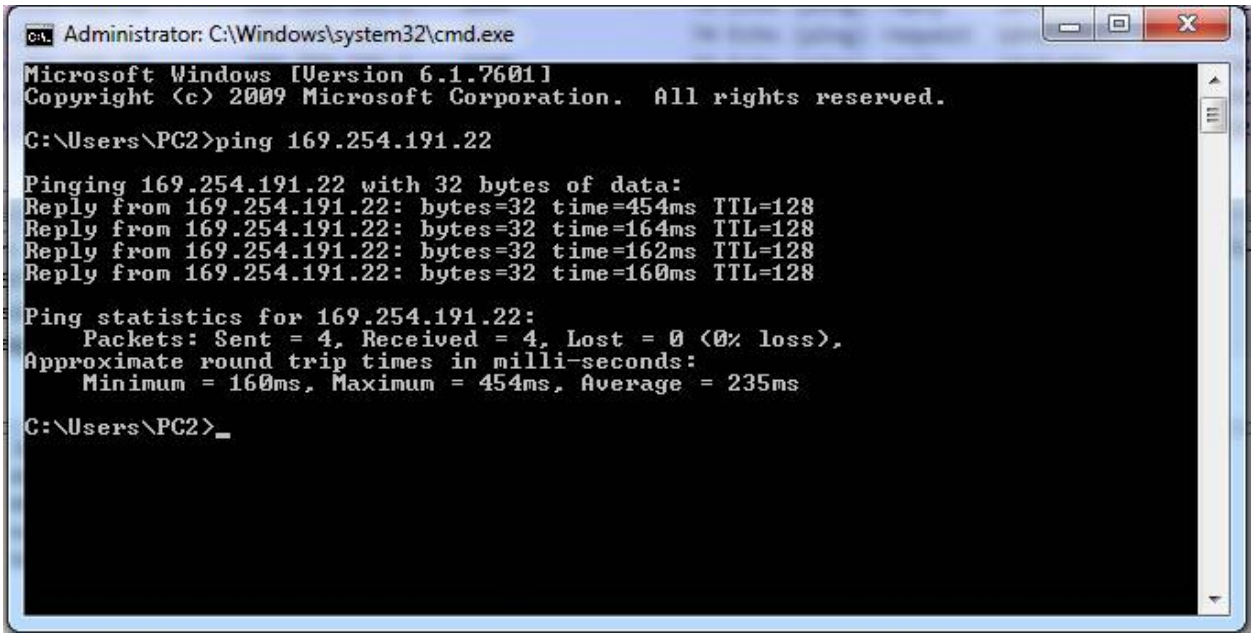
بریم ببینیم تابع ICMP_Process چطور پیاده سازی شده :

```
uint16_t  ICMP_Process ( ICMP_EchoFrame* icmpFrame, uint16_t frameLen )
{
 uint16_t  newFrameLen = 0;

 uint16_t  rxCheckSum = icmpFrame-> checkSum;
 icmpFrame->checkSum = 0;
 uint16_t calcCheckSum = IP_CalcCheckSum (( uint8_t* ) icmpFrame, frameLen ) ;

 if ( rxCheckSum == calcCheckSum )
 {
  if ( icmpFrame->type == ICMP_FRAME_TYPE_ECHO_REQUEST )   //=0x08
  {
   icmpFrame->type = ICMP_FRAME_TYPE_ECHO_REPLY;   //=0x00
   icmpFrame->checkSum = IP_CalcCheckSum (( uint8_t * ) icmpFrame, frameLen ) ;
   newFrameLen = frameLen;
  }
 }
 return   newFrameLen;
}
```

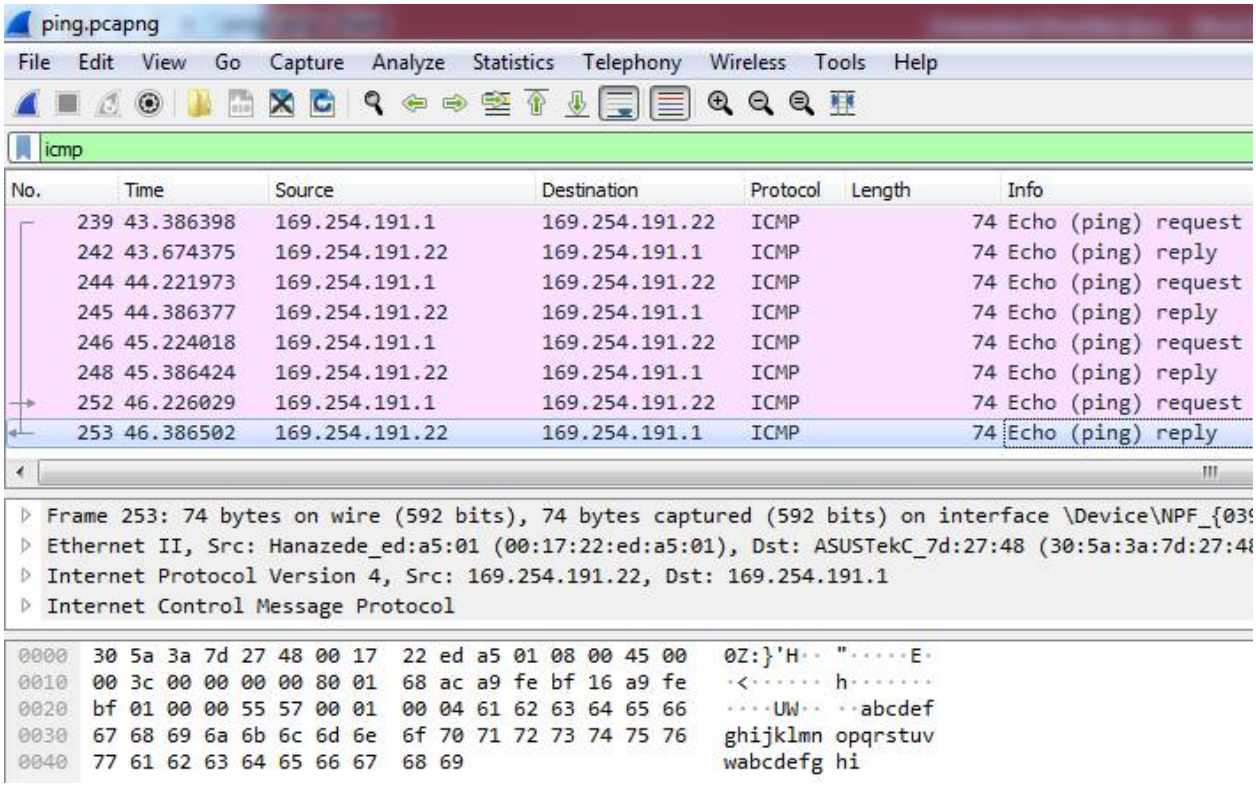
طبق معمول صحت چک سام بررسی شده، و بعد بررسی کردیم که آیا در قسمت type در هدر، درخواست پینگ داریم؟ (بالطبع اگه بخوایم درخواست های دیگه رو بررسی کنیم؛ باید زیر این if اضافه کنیم) اگر نتیجه مثبت باشه ؛ فقط بخش نوع فریم فعلی رو تغییر میدیم؛ چک سام جدید رو در جای خودش می نویسیم و از تابع خارج میشیم؛ و همونطور که از قبل پیاده سازی شده، لایه سوم این نتیجه رو میگیره ، IP فرستنده گیرنده رو تغییر میده، چک سام خودش رو بوجود میاره و اونو به لایه دو میده؛ در این لایه هم جای مک آدرس ها تغییر میکنه و فریم ارسال میشه.

- عملیات پینگ در ویندوز، به سادگی با دستور ping در کامند پرامپت ویندوز اجرا میشه و فقط کافیه IP هاست مورد نظرمون رو بهش بدیم (برای باز کردن کامند پرامپت در منوی استارت ویندوز تایپ کنید cmd) البته این دستور، آرگومان های بیشتری رو میگیره که میتونید بطور کامل کنترلش کنید ولی در حالت پیش فرض، دستور پینگ در ویندوز؛ چهار پیغام شامل 32 کاراکتر اسکی به مقصد ارسال میکنه و انتظار داره همون ها رو در جواب ببینه. بعد از اتمام پروسه پینگ، اطلاعات آماری مثل زمان ارسال و دریافت پاسخ و حداقل/حداکثر/میانگین پاسخ گویی و تعداد روترهای در مسیر رو بما نشون میده (با محاسبه تعداد کاهش در TTL).

در دو شکل زیر نتیجه درخواست و پاسخ پینگ برد ما که توسط ویندوز فرستاده شده؛ رو می بینیم:

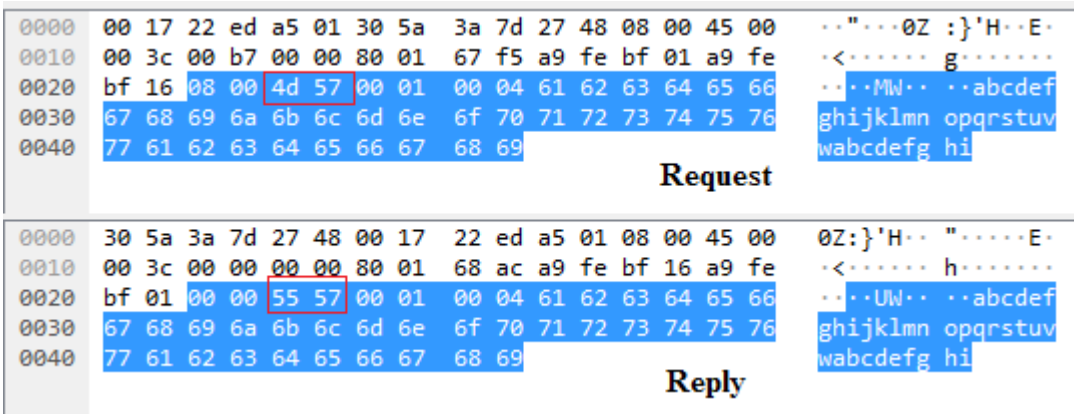


شکل 21 : اجرای دستور پینگ در ویندوز



شکل 22 : شنود پروتکل ICMP با نرم افزار wireshark

همونطور که در تصویر معلومه؛ با یکبار اجرای دستور پینگ، چهار مرتبه درخواست پینگ ارسال و پاسخ دریافت شده است. اطلاعات موجود در آخرین پاسخ شکل بالا (انتخاب شده با رنگ آبی) بصورت در شکل زیر ثبت شده :



همونطور که در شکل دیده میشه؛ تنها بخش type از مقدار 0x08 به 0x00 تغییر کرده. لذا چک سام هم بناچار تغییر داشته و این تغییر تنها در بایت پرارزش چک سام هست (در واقع از چک سام اولیه 0x0800 یا از بایت پرارزش 0x08 تا کم شده!). این جواب سوالی ست که کمی قبل تر پرسیدیم. محل قرارگیری بایت type در بخش پرارزش کلمه 16 بیتی (word) بوده؛ از type هشت تا کم شده که نتیجه این کاهش، شده تغییر 0x55 در بایت پرارزش چک سام به 0x47 ؛ پس ساده تر اینه که، هنگام محاسبه چک سام، به جای محاسبه تمام چک سام؛ فقط مقدار بایت پرارزش رو 8 تا؛ (یا از کل چک سام، 0x0800 تا) کم کنیم.

پروتکل UDP :

شاید انتظار داشتید که در این مرحله با پروتکل TCP شروع کنیم اما از اونجایی که پیاده سازی پروتکل UDP بسیار ساده تر از TCP هست؛ بعلاوه این دو پروتکل مفاهیم مشترک زیادی دارند؛ تصمیم داریم با UDP کار رو جلو ببریم.

همونطور که دیدیم؛ پروتکل ICMP هم مثل پروتکل ARP دریافت کننده داده ای از لایه های بالاتر نبود ولی دو پروتکل UDP و TCP عموما بسته هایی از لایه های بالاتر دریافت کرده، هدر خودشون رو اضافه میکنند؛ سپس این داده ها رو که دیگه اینجا بهش میگیم سگمنت (segment) تحویل لایه سه میدن. در هنگام دریافت هم، از لایه سه سگمنت رو میگیرند؛ هدر خودشون رو برمیدارن ؛ داده رو تفسیر میکنند و بعد میدن به لایه چهارم و...

لایه های 5 و بالاتر یا L5+ or L5~7	داده اصلی Main data
لایه 4 (Transport Layer)	TCP or UDP segments
لایه 3 (Network Layer)	IP Packets
لایه 2 (Data Link Layer)	Ethernet ii frames
لایه 1 (Physical Layer)	‘0’ , ‘1’ Stream

یادمون هست که در لایه دو ما آدرس های فیزیکی یا همون MAC Address ها رو داشتیم و بعد دلیل بوجود اومدن آدرس های منطقی یا همون IP Address رو گفتیم که در لایه سه مورد استفاده قرار می گرفتند. خب تا اینجا ما تونستیم ارتباط هاست به هاست رو در شبکه داشته باشیم. اما اگر در داخل یک هاست، سرویس ها یا نرم افزارهای مختلفی در حال کار باشند؛ چطور باید مشخص کنیم که هر بسته باید به کدام سرویس تحویل بشود؟ اینجاست که یک نوع آدرس دهی جدید بنام شماره port رو داریم. شماره پورت ها مثل شماره های داخلی تلفن بر روی یک شماره اصلی هستند و عملا آدرس دهی سرویس به سرویس رو پوشش میدن. توجه داشته باشید که این شماره پورت رو با پورت های سخت افزاری مثل SPI یا UART اشتباه نگیرید.

شماره پورت یه عدد 16 بیتی ست؛ درنتیجه میشه باهاش 65536 سرویس مجزا رو روی یک هاست داشته باشیم. البته تعدادی از این پورت ها توسط IANA برای پروتکل های لایه های بالاتر درنظر گرفته شده ن و اختصاصی هستند. تعدادی دیگه هم رزرو شده ن؛ اما قسمت عمده پورت ها در اختیار کاربر هست. از اونجاییکه پورت های

اختصاصی عموما در شماره های کوچکتر از 1023 قرار دارند؛ پیشنهاد ما اینه که برای سخت افزارتون از پورت های با شماره بزرگتر از 1023 و کوچکتر از 49151 استفاده کنید. البته این فقط یک پیشنهاده و الزامی نیست. در جدول زیر تعدادی از پورت های اختصاصی برای پروتکل های لایه های بالاتر رو مشاهده میکنید:

پورت	پروتکل
7	Echo protocol
11	Day Time protocol
15	Netstat service
20,21	FTP (File Transfer Protocol)
22	SSH (Secure SHell)
23	Telnet Protocol (Telecomunication network)
25	SMTP (Simple Mail Transfer Protocol)
37	TIME protocol
53	DNS (Domain Name Service)
67,68	DHCP (Dynamic Host Configuration Protocol) And BOOTP (Boostarp Protocol)
69	TFTP (Trivial File Transfer Protocol)
80	HTTP (HyperText Transfer Protocol)
115	SFTP (Simple File Transfer Protocol)
161	SNMP (Simple Network Management Protocol)

همونطور که دیده میشه؛ ممکنه برای یک عملیات خاص مثل انتقال فایل؛ پروتکل های مختلفی بر روی پورت های مختلف داشته باشیم (FTP, TFTP) و یا یک هنگام بروزرسانی یک پروتکل قدیمی از همان شماره پورت برای پروتکل جدیدتر استفاده شده باشد؛ مثل پورت های 67,68 که در ابتدا برای سرویس های درخواست/پاسخ BOOTP در نظر گرفته شده بودند؛ اما با ظهور پروتکل DHCP که برای اعمال تنظیمات یک هاست توسط یک سرور بکار می رود؛ مجددا از همین پورت ها استفاده شده است. تذکر بدیم که در ادامه این سند، پروتکل DHCP و یکی از اصلیترین کارکردهای آن یعنی تخصیص IP Address به هاست های کلاینت را خواهیم داشت.

برگردیم سراغ UDP . پروتکل UDP (Use Datagram Protocol) برای ارسال و دریافت داده هایی استفاده میشوند که بین دو مقوله "زمان" یا "سلامت" انتقال داده؛ برای زمان اهمیت بیشتری قایل هستند. این پروتکل اصطلاحا یک پروتکل connectionless یا بدون اتصال است، بدین معنی که اولا فرستنده و گیرنده قبل از شروع انتقال داده، با هم مذاکره یا هماهنگی خاصی ندارند! بعلاوه فرستنده، بعد از ارسال پیام؛ بررسی نمیکند که

آیا داده ها به سلامت رسیده اند یا خیر! البته این موضوع، رفتار تعریف شده در استاندارد این پروتکل است ولی اگر برنامه نویسی هر دو سمت فرستنده و گیرنده دست شما باشه ؛ خودتون میتونید این بررسی رو انجام بدید.

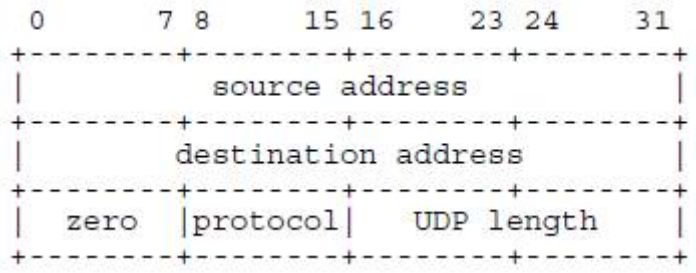
بخش هدر در یک سگمنت UDP بصورت زیر هست :

Octet Offset	0	1	2	3
0	Source Port		Destination Port	
4	Length		CheckSum	

Source & Destination Port : دو عدد 16 بیتی که شماره پورت فرستنده و گیرنده رو مشخص میکنه. به جز در پروتکل های استاندارد؛ هر شماره پورتی، میتونه استفاده بشه؛ مشروط به اینکه این شماره پورت، توسط سرویس دیگه ای در حال استفاده نباشه و به اصطلاح آزاد باشه. در ارتباط های یک طرفه (یک سمت فقط ارسال داره؛ سمت دیگه فقط دریافت میکنه)؛ پورت مبدا کاربردی نداره؛ اما بهرحال باید شماره ای به عنوان شماره پورت مبدا قرار داده بشه.

Length : طول تمام بایت های واقع در سگمنت رو مشخص میکنه؛ به عبارتی طول هدر بعلاوه طول داده؛ پس کمترین مقدار Length عدد 8 هست.

Checksum : چک سام روی هدر؛ payload یا همان داده ها و یک بخش خاص از پکت IP که این سگمنت داخلش هست؛ انجام میشه. این کار به دلایل امنیتی و برای اطمینان بیشتر از اینکه احیانا سگمنت ها؛ بین پکت های مختلف (عموما در روترها) جابجا نشده باشند؛ انجام میشه. اگر قرار بود چک سام تنها روی بخش هدر و داده ها انجام بشه، اگر سگمنت های UDP بین دو پکت مختلف از پروتکل IP جابجا بشوند؛ قابل تشخیص نخواهد بود. این بخش خاص که در محاسبه چک سام اضافه شده، بنام شبه هدر (psudo header) شناخته میشه و ترکیبش مثل شکل زیر هست :



همونطور که در شکل میبینید؛ شبه هدر شامل آدرس IP فرستنده و گیرنده بعلاوه شماره پروتکل UDP (عدد 17 یا همون 0x11) و طول سگمنت UDP هست. قسمت zero هم صفر هست (برای اینکه اندازه بخش protocol؛ 16 بیتی باشه).

پس قبل از انجام چک سام، ابتدا بخش چک سام در UDP رو صفر میکنیم؛ سپس IP آدرس های فرستنده و گیرنده رو بصورت 16 بیتی جمع می کنیم. عدد حاصل رو ابتدا با 0x0011 (شماره پروتکلUDP و بخش Zero) و بعد هم با اندازه طول سگمنت UDP جمع میکنیم. حالا این نتیجه رو به عنوان مقدار اولیه در تابعی که برای چک سام نوشتیم، استفاده میکنیم.

- اولین استاندارد از پروتکل UDP در سند RFC 768 اومده. برای اطلاعات بیشتر به این سند و آپدیت های آن مراجعه کنید.
- در پروتکل TCP هم خواهیم دید که محاسبه چک سام؛ شامل بخش شبه هدری با همین فرمت است. تنها تفاوت در اینه که شماره پروتکل در اونجا عدد 0x06 هست. شما میتونید در نت، کدهای بهینه ای که چک سام رو برای هر چهار پروتکل IP,ICMP,UDP و TCP انجام میده؛ پیدا کنید.

حالا نوبت اینه که تابع IP_Process() رو طوری تغییر بدیم که بتونه سگمنت های UDP رو هم دریافت کنه. از اونجاییکه ما داریم با مفاهیم اولیه و پروتکل های اصلی آشنا میشیم؛ اینجا کد رو طوری مینویسیم که بعد از دریافت هر سگمنت UDP ، به تک تک بایت های دریافتی، یک عدد اضافه کنه و به فرستنده برگردونه. قاعدتا شما میتونید از UDP برای ارسال هر نوع داده یا پردازشی استفاده بکنید. به عنوان مثال ارسال داده های تعدادی سنسور از یک برد الکترونیکی به کامپیوتر؛ پردازش داده ها در کامپیوتر و سپس ارسال دستور از کامپیوتر به برد الکترونیکی جهت راه اندازی تعداد عملگر مثل رله؛ شیر برقی و...

- همونطور که میدونید هسته اصلی برنامه ما طوری نوشته شده که ابتدا یک فریم رو دریافت میکنه و در صورت لزوم، بهش جواب میده (مثل عملیات پینگ). اما اگر میخواهید بدون انتظار برای دریافت یک سگمنت UDP؛ بتونید یک سگمنت UDP ایجاد و ارسال کنید؛ باید نحوه پیاده سازی کد رو تغییر بدید.

تابع IP_Process() اینگونه تغییر خواهد کرد:

```
if (ipFrame->protocol == IP_FRAME_PROTOCOL_ICMP)      //==0x01
    {
     newDataLen = ICMP_Process((ICMP_EchoFrame*)ipFrame->data, dataLen);
    }

else if (ipFrame->protocol == IP_FRAME_PROTOCOL_UDP)          //==0x11
    {
     newDataLen = UDP_Process((UDP_Frame*)ipFrame->data, dataLen);
    }
```

ساختار UDP_Frame مثل ساختارهای قبلی، طوری تعریف شده که بتونیم هدر و داده های سگمنت رو از هم جدا کنیم. همچنین یک پورت برای ارتباط UDP معرفی میکنیم:

```
#define  UDP_DEMO_PORT  2020

typedef  struct  UDP_Frame
{
 uint16_t  srcPort;
 uint16_t  destPort;
 uint16_t  len;
 uint16_t  checkSum;
 uint8_t  data[];
} UDP_Frame;
```

و تابع UDP_Process() هم اینگونه تعریف میشه :

```
uint16_t    UDP_Process(UDP_Frame*  udpFrame, uint16_t  frameLen)
{

 uint16_t  destPort = ntohs(udpFrame->destPort);
 uint16_t  len = ntohs(udpFrame->len);
 uint16_t  dataLen = len - sizeof(UDP_Frame);

 if (destPort == UDP_DEMO_PORT)
 {
  for(uint16_t  i = 0 ; i < dataLen - 1; i++)
  {
   udpFrame->data[i]++; // به بایت های دریافتی یک عدد اضافه میکنیم
  }
 }
```

```
  uint16_t   swapPort = udpFrame->destPort;
  udpFrame->destPort = udpFrame->srcPort;
  udpFrame->srcPort = swapPort;

  udpFrame->checkSum = UDP_checkSum();  // calculate new checksum for UDP
  return   len;
}
```

همونطور که در کد دیده میشه؛ بعد از دریافت یک سگمنت ابتدا بررسی میشه که آیا پورتی که ما در انتظارش هستیم آدرس دهی شده یا نه؟ (مقدار چک سام رو بررسی نکردیم! اما بهتره که بررسی بشه؛ استاندارد هم تاکید میکنه که انجام بشه) سپس به مقدار بایت های دریافتی، یک عدد اضافه کردیم (با عملگر ++) بعد جای پورت فرستنده/گیرنده رو در درون سگمنت تغییر دادیم؛ چک سام جدید رو حساب کردیم و سگمنت خودمون رو برگردوندیم به لایه IP تا ارسال کنه.

برای تست پروتکل UDP میتونید خودتون برنامه ی ساده ای بنویسید یا از برنامه های آماده و رایگانی مثل echotool استفاده کنید. فایل اجرایی این نرم افزار رو می تونید از https://github.com/PavelBansky/EchoTool/tree/master دانلود کنید (این نرم افزار در command prompt ویندوز اجرا میشه). همچنین بسیاری از شرکت های تولید کننده ماژول های اترنت مثل USR نیز نرم افزارهایی برای ارتباط در اختیار کاربران قرار می دهند. نرم افزار تست این شرکت رو از آدرس https://www.pusr.com/support/download/PC-Test-Software-USR-TCP232-Test-V1-3.html دانلود کنید.

کار ما با UDP در اینجا تموم میشه. قبل از اینکه بخواهیم پروتکل TCP رو توضیح بدیم، ترجیحمون اینه که پروتکل DHCP رو معرفی کنیم و کمی هم از DNS بگیم. علت اینکار اینه که پروتکل TCP پروتکل پیچیده ایه (قورباغه رو الان قورت نمیدیم!) از طرفی DHCP و DNS روی UDP کار می کنند.

DNS چیست؟

در لایه های بالاتر از لایه چهارم؛ پروتکل های بسیار زیادی تعریف شده اند که تعدادی بر اساس UDP کار می کنند (یعنی روی UDP پیاده سازی شده ن). از میان این پروتکل ها؛ دو تاشون خیلی معروف هستند و کاربرد زیادی دارند. اولی به نام DNS (Domain Name Service) شناخته میشه و دومی به نام DHCP (Dynamic Host Control Protocol)

تا اینجا دیدیم که هنگام برقراری یک ارتباط، دونستن آدرس IP مقصد ضروری هست و فرض کردیم که این آدرس رو داریم. در ابتدای پیدایش شبکه ها؛ چون تعداد IP ها در یک شبکه داخلی کم بود؛ این وظیفه ادمین ها بود که IP ها رو به صورت دستی تنظیم کنند. با گسترش یک شبکه و در نتیجه گسترش فضای آدرس دهی و علاوه بر اون استفاده عمومی از شبکه ها توسط افراد غیرمتخصص؛ داشتن پروتکلی که بشه IP یک هاست رو به طور خودکار تنظیم کرد یا بدست آورد؛ ضروری به نظر می رسید. علاوه بر این، کار با اعداد (در آدرس IP) معمولا سخت تر و نامفهوم تر از به کار بردن اسامی هست. هدف این بود که به جای استفاده از اعداد برای آدرس دهی؛ از اسم یا متن استفاده بشود. واضحه که برای کاربر، استفاده از آدرس یا اسمی مثل google.com بسیار راحت تر و مفهوم تر هست نسبت به آدرسی به شکل14.151.6.17؛ حفظ کردنش هم ساده تره به علاوه تعداد آدرس ها که زیاد بشه؛ استفاده از روش آدرس دهی عددی برای کاربر، بسیار مشکل میشه.

در دهه های قبل از اختراع کامپیوتر و شبکه، در موارد مشابه، مثلا در سیستم تلفن ثابت، راهکارهایی مثل چاپ کتابچه شماره تلفن ها یا استفاده از مرجع شماره تلفن ها (مثل شماره 118 در ایران) کار رو راه میانداخت. اما در دنیای کامپیوترها، این وظیفه هم به عهده خود کامپیوتر و پروتکل ها گذاشته شده. یک سیستمی (کامپیوتر، دیتابیس، سرور؛ سیستم خلاصه!) به اسم DNS سرور وجود داره که شما بهش اسم سایت رو میدید؛ سیستم موردنظر، آدرس IP متعلق به اون اسم رو بهتون میده. به عنوان مثال وقتی در جستجوگر اینترنت (Browser) مینویسید google.com ابتدا یک پیغام برای سرور DNS در شبکه ارسال میشه، اگر آدرس IP این سایت در جداول این سرور باشه؛ بهتون میده وگرنه این پیغام رو به یه DNS سرور دیگه میده تا اون بهتون جواب بده. بهرحال در نهایت شما IP اون سایت رو با این سرویس به دست خواهید آورد. ما بیشتر از این DNS رو توضیح نمیدیم و شما برای اطلاعات بیشتر میتونید به سند RFC 882 و بروزرسانی های اون مراجعه کنید.

DHCP یا Dunamic Host Configuration Protocol :

و اما DHCP. پروتکل دیگه ای که بر روی UDP کار میکنه؛ DHCP هست که وظیفه ش، دریافت و اعمال تنظیمات یک کلاینت هست. این تنظیمات میتونه شامل خیلی موارد باشه؛ از دریافت آدرس IP تا دریافت فایل بوت (boot) کلاینت، آدرس سرور DNS ، مقدار subnet mask ، آدرس gateway و ...

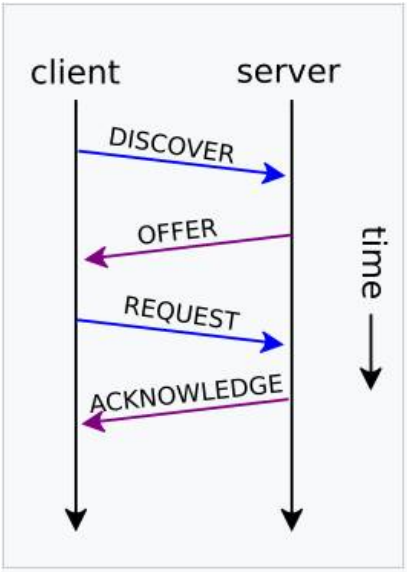
جهت برآورده کردن این منظور؛ در ابتدا از پروتکلی بنام (BOOTP (BOOT Protocol استفاده میشد اما در نهایت BOOTP با DHCP جایگزین شد. یکی از اصلیترین دلایل تعریف استانداردهای BOOTP و DHCP این موضوع بود که تخصیص آدرس IP به کلاینت ها، کاملا خودکار (اما با اعمال نظر ادمین شبکه) باشه تا احیانا تداخل آدرس (دو یا چند سیستم با یک آیپی) در شبکه نداشته باشیم. تا اینجای کار، ما (برنامه نویس) آدرس IP رو داخل کد به صورت ثابت نوشته بودیم؛ اما واضحه که این روش، روش درستی برای این کار نیست. فرض کنید در یک شبکه صدها سیستم وجود داشته باشه؛ آیا میخواهیم به صورت دستی، تک تک اون ها رو آدرس دهی کنیم؟ به علاوه، از امکانات یک شبکه، ممکنه تعداد بسیار زیادی سیستم، بخواهند استفاده کنند و ما با محدودیت آدرس مواجه باشیم. لذا در DHCP، تمهید لازم برای اختصاص موقت یک آدرس IP به یک هاست هم دیده شده. اگر قرار بود هر سیستمی یک آیپی دایمی داشته باشه؛ به سرعت با کمبود آدرس مواجه می شدیم. وقتی یک ارتباط قطع میشه یا مهلت اجاره آدرس تموم میشه؛ سرور DHCP، از اون آیپی برای دستگاه های دیگه استفاده خواهد کرد. در یک ارتباط اینترنتی نیز اگر دقت کرده باشید؛ احتمالا متوجه این موضوع شده اید که گاهی در حین ارتباط، آدرس IP شما تغییر میکنه. به این مفهوم اصطلاحا lease Time یا زمان اجاره گفته میشه. قبل از سرآمدن مهلت اجاره؛ هاست میهمان (کلاینت) باید تقاضای تمدید مهلت رو بکنه که ممکنه پذیرفته نشه؛ یا اگر پذیرفته بشه؛ یک آدرس جدید به هاست اختصاص داده بشه.

یک سوال؛ الان میدونیم که DHCP از UDP استفاده میکنه. خود UDP هم بر روی پروتکل IP بنا شده ؛ این سیستم که هنوز IP نداره، پس چطوری کار میکنه ؟! (سیستمی که هنوز آدرس IP نداره؛ از پروتکل IP استفاده میکنه تا آدرس IP خودشو بدست بیاره!)

در پروتکل IP هم مثل استاندارد Ethernet ii ؛ یک آدرس خاص داریم که تمام قسمت های اون با بیت های '1' پر شده؛ پس مقدار این IP خاص هست 255.255.255.255 به این آدرس اصطلاحا آدرس عمومی در لایه سوم گفته میشه. در پیاده سازی پروتکل IP و برای برقراری یک ارتباط DHCP ؛ سخت افزار شما (میکروکنترلر) ؛ باید بتونه بسته هایی با آدرس عمومی رو دریافت و پردازش کنه تا بتونه با استفاده از این پروتکل، آدرس IP خودش رو از DHCP server بگیره.

فرآیند یک ارتباط مبتنی بر DHCP و اختصاص آدرس IP و تنظیمات دیگه، در چهار مرحله اجرا میشه:

- اکتشاف Discovery (از طرف کلاینت)
- پیشنهاد Offer (از طرف سرور)
- درخواست Request (از طرف کلاینت)
- پذیرش Acknowledge یا عدم پذیرش NACK (از طرف سرور)



به این چهار مرحله اصطلاحا DORA گفته میشه که سرنام چهار مرحله فوق هست. این فرآیند با یک سوال از طرف کلاینت شروع میشه:

"کسی هست بتونه بمن IP بده؟ ترجیح من آیپی فلان هست!" ترجیح یک آدرس خاص؛ مال وقتیه که دستگاه از قبل در شبکه بوده و مدت زمان اجاره ش (Lease Time) تموم شده.

در جواب DHCP server میگه که "فلان آیپی رو دارم، میخوای؟" البته قبلش ممکنه سرور با یک عملیات خاص مثلا ping یا ARP بررسی کنه که در شبکه دستگاهی با آیپی پیشنهادی وجود نداشته باشه! کلاینت هم ممکنه قبل از درخواست خودش، همین کار رو بکنه.

یک نکته مهم هم اینکه در یک شبکه ممکنه چند DHCP server داشته باشیم و حتا ممکنه این سرورها با هم، همپوشانی آدرس داشته باشند؛ یعنی بخشی از محدوده آدرس دهی آنها یکسان باشه. در نتیجه، کلاینت ممکنه بعد از درخواست اولیه، چندین پیشنهاد دریافت کنه!

در مرحله سوم؛ کلاینت باید از بین جواب های رسیده از سرورهای احتمالی، یکی رو انتخاب کنه و به اون سرور جواب بده که "من فلان آیپی رو برمیدارم" (مابقی سرورها بدون جواب بمونن، داستان رو فراموش میکنند)

در مرحله آخر (چهارم) هم سرور یک پیغام به کلاینت میفرسته که بگه درخواست پذیرفته یا رد شده؛ هر جای این مراحل، عملیات دچار خطا بشه؛ باید از ابتدا شروع بشه. در نهایت هم، معمولا سرور، آدرس اختصاص یافته رو در جدولی ذخیره میکنه تا از اختصاص مجدد اون به یه سیستم دیگه جلوگیری کنه. کلاینت هم میتونه برای استفاده های بعدی آدرس سرور رو نگه داره.

در ابتدای فرآیند؛ کلاینت فقط مک آدرس خودش رو داره. خودش که آدرس آیپی نداره، هیچ! مک آدرس و آیپی آدرس سرور رو هم نداره پس پیغام مرحله اول یک پیغام عمومی (هم آیپی آدرس در لایه سوم و هم مک آدرس در لایه دوم) هست، اما در مراحل بعدی که کلاینت و سرور؛ آیپی و مک آدرس های هم رو دارند، میتونند پیغام ها رو بصورت خصوصی ارسال کنند. البته در حالت خاصی که کلاینت از قبل، آدرسی رو اجاره کرده باشه؛ داستان فرق میکنه و چون اون ها همدیگه رو میشناسند، میتونن از ابتدا پیغام خصوصی (unicast) بفرستن.

- تا الان مقدار آدرس IP رو بطور مستقیم در کد تعریف کرده بودیم، به این وضعیت اصطلاحا Static IP میگن. با استفاده از پروتکل DHCP قادریم به سه گونه یا روش، مقدار آدرس IP کلاینت رو تعیین کنیم. در اولین حالت بنام Automatic Allocation یک آدرس دایمی به کلاینت داده میشه. در حالت دوم، سرور یک آدرس IP موقتی (برای مدت زمان مشخص) به کلاینت اختصاص میده. در این حالت، کلاینت میتونه قبل از سر اومدن زمان اجاره؛ این آدرس رو رها (Release) کنه یا اینکه درخواست تمدید بکنه. این حالت رو Dynamic Allocation میگن. در حالت سوم به نام Manual Allocation؛ اختصاص آدرس بطور مستقیم توسط ادمین شبکه تعیین میشه. این حالت رو با حالت استاتیک نباید اشتباه گرفت. در این حالت، درخواست آدرس IP همچنان توسط پروتکل DHCP انجام میشه؛ اما آدرس، بطور خودکار توسط سرور اختصاص داده نمیشه؛ بلکه ادمین خودش آدرس رو مشخص میکنه. در دو حالت دیگه هم، بازه ی آدرسی (range) که توسط سرور میتونه اختصاص پیدا کنه؛ عموما توسط ادمین مشخص میشه. مثلا ادمین مشخص میکنه که آدرس اختصاص یافته از 192,168,1,5 تا 192,168,1,150 انتخاب و پیشنهاد بشه. به این بازه در زبان شبکه، استخر (pool) آدرس گفته میشه. در شبکه جهانی اینترنت، از نوع دوم یعنی Dynamic Allocation یا اختصاص پویا استفاده میشه.

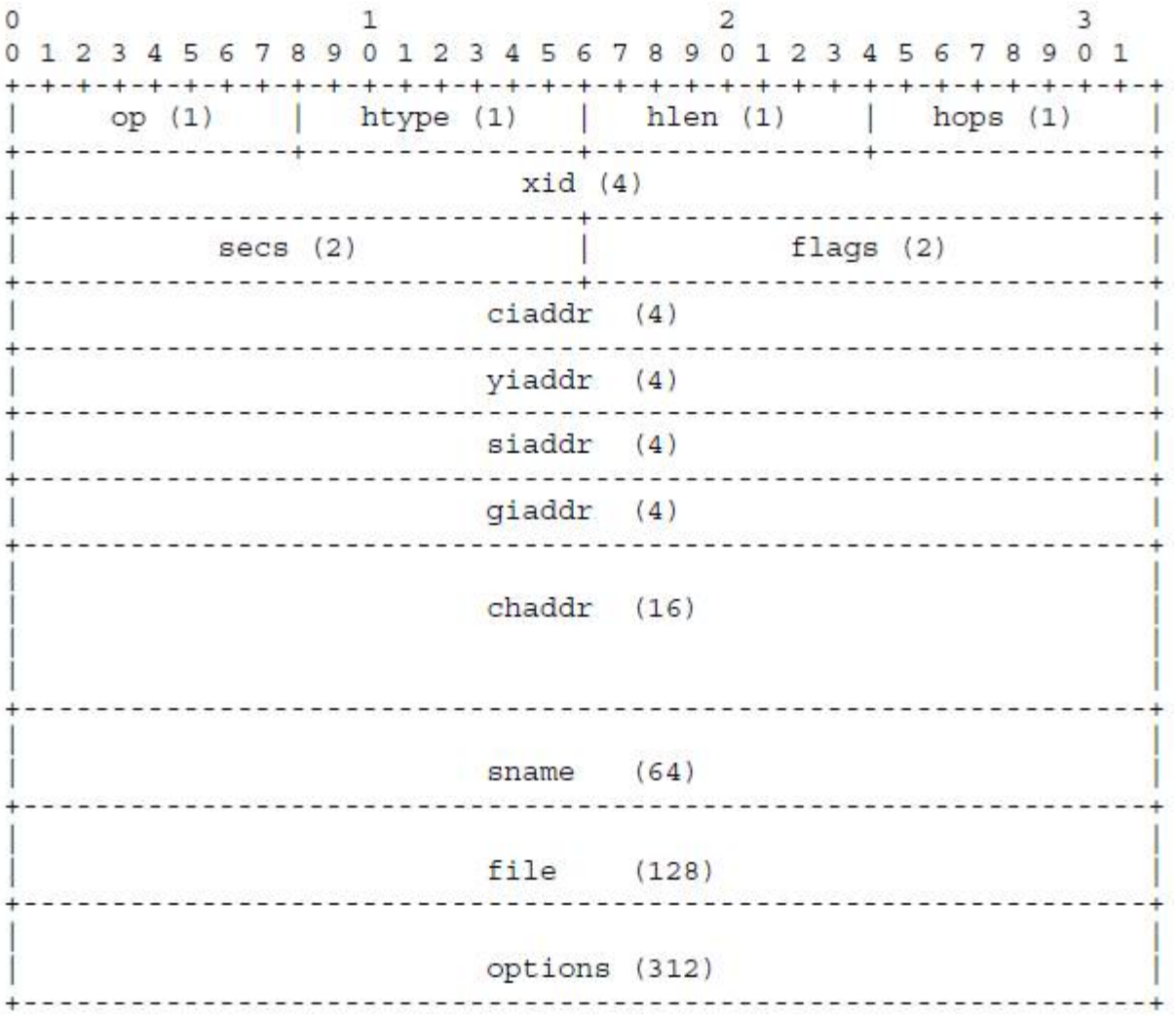
از آنجاییکه پروتکل DHCP در لایه چهارم کار میکنه؛ دو پورت، 68 برای کلاینت و 67 برای سرور، جهت استفاده در این پروتکل اختصاص یافته اند.

اطلاعات کاملتری از پیاده سازی ؛ روش ها ، مفاهیم و الزامات پروتکل DHCP رو میتونید در سند RFC 1541 و ارجاعات اون پیدا کنید.

و اما پیاده سازی در میکروکنترلر. ابتدا بریم هدر DHCP رو بررسی کنیم.

فرمت هدر DHCP :

در شکل زیر فرمت یک پیام DHCP رو مشاهده میکنید، توجه داشته باشید که فرمت هدر در پروتکل های BOOTP, DHCP یکسان هست. اعداد داخل پرانتز تعداد بایت های هر بخش رو مشخص کرده:



OP : نوع عملیات (Operation) رو مشخص میکنه. در پیام های ارسالی از کلاینت، این بخش دارای مقدار 0x01 به معنای "درخواست" و در بازگشت دارای مقدار 0x02 به معنای "پاسخ" خواهد بود. همچنین در بخش Options نیز گزینه ای برای مشخص کردن مراحل چهارگانه ای که گفته شد؛ وجود دارد.

HType : Hardware Type یا نوع سخت افزار؛ برای اترنت 10Mb عدد 0x01 است.

HLen : Hardware Address Length یا طول آدرس سخت افزاری که برای مک آدرس عدد 0x06 است.

Hops : تعداد hop ها یا همان روترهای بین کلاینت و سرور را مشخص میکند و یادگار پروتکل BOOTP است. این بخش در DHCP همواره 0x00 است.

XID : یا شماره شناسایی کلاینت؛ شامل یک عدد تصادفی 4 بایتی است که در مرحله Discovery توسط کلاینت انتخاب و ارسال میشود. بعد از آن سرور و خود کلاینت از این عدد برای شناسایی همدیگه استفاده میکنند. یادمون هست که گفتیم، ممکنه چندین سرور پاسخ بدهند یا همزمان چندین کلاینت در حال درخواست باشند.

SECs : در این بخش که از سمت کلاینت نوشته میشود، ثانیه هایی که از شروع بوت کلاینت گذشته است، قرار میگیرد. از این بخش استفاده ای نمیکنیم و عدد 0x0000 را قرار میدهیم.

Flags : تنها یک بیت (پر ارزشترین بیت) از این بخش 16 بیتی، تعریف و استفاده شده است. مابقی بیت ها رزرو شده و همواره صفر هستند. پرارزشترین بیت، به نام بیت B یا همان Broadcast تنها توسط کلاینت، در دو پیام discovery یا Request میتواند ست بشود که به سرور این پیغام را میدهد که پاسخ پیام های کلاینت را به صورت Broadcast یا همگانی (مک آدرس و آیپی آدرس با بیت های '1') ارسال کند و بدین معنی است که در حال حاضر کلاینت نمیتواند پیام های خصوصی (unicast) را پردازش کند. فلسفه وجود این بیت هم بنظر واضحه دیگه؛ کلاینت هنوز آدرس نداره،پس از سرور میخواد که فعلا پاسخ پیام ها رو به صورت Broadcast برگردونه. اما خود کلاینت از اونجاییکه مک آدرس و آیپی آدرس سرور رو، در مرحله Offer دریافت کرده؛ از اون ها استفاده میکنه و جواب خودش در مرحله سوم (Request) رو بصورت unicast ارسال میکنه. نتیجه اینکه بخش Flags از طرف کلاینت با مقدار 0x8000 و از طرف سرور با مقدار 0x0000 ارسال میشود.

CIAddr : یا Client Ip Address شامل 4 بایت برای آدرس آیپی کلاینت. این آدرس در واقع آدرس فعلی(قبلی) کلاینت هست و نه آدرس آیپی جدید. در مرحله اول و دوم مقدار این بخش 0x00000000 خواهد بود (مگر اینکه کلاینت از قبل در شبکه بوده و آدرسی داشته است). در مرحله سوم و چهارم، در این بخش مقدار آیپی اختصاص یافته قرار میگیره. توجه داشته باشد چنانچه بیت B از بخش Flags در مرحله یک توسط کلاینت ست نشده باشه؛ سرور میتونه در مرحله دوم و در هدر پروتکل IP خودش از این آدرس استفاده کنه و پیغام رو

unicast بفرسته. شاید گیج کننده باشه. بهرحال ممکنه کلاینت از قبل تو شبکه بوده و آدرس داشته و مدت زمان اجاره ش تموم شده و الان درخواست آیپی جدید یا تمدید آدرس فعلی رو داره؛ از طرفی پیام ها خودشون روی پروتکل IP سوار هستند؛ لذا سرور (اگر بیت B ست نشود) میتونه پیغام رو مستقیم به آیپی فعلی بفرسته. ما حالت ساده رو در نظر میگیرم و فرض میکنیم در ابتدای کار، بردمون میخواد آیپی بگیره؛ لذا این بخش رو با 0x00 پر میکنیم و بیت B رو هم ست میکنیم، خلاص! به جاش باید آدرس آیپی عمومی 255.255.255.255 رو هم تو برنامه میکرومون پردازش کنیم.

YIAddr : Your(client) Ip Address در مرحله دوم و چهارم، سرور در این قسمت آدرس آیپی پیشنهادی رو مینویسه. در مرحله اول و سوم این قسمت توسط کلاینت با 0x00 پر میشه.

SIAddr : Server Ip Address در مرحله اول این قسمت با 0x00 پر میشه؛ اما در مراحل دو تا چهار در این قسمت، آدرس آیپی سرور نوشته میشه (در مرحله سوم کلاینت ممکنه چند پیشنهاد دریافت کنه و آدرس سرور انتخاب شده رو اینجا مینویسه؛ توجه داشته باشید که همزمان در لایه سوم هم آدرس آیپی گیرنده همین هست).

GIAddr : aGent Ip Addr یا Gateway Ip Addr ؛ آدرس عامل (agnet، نماینده) باقیمانده از پروتکل BOOTP و در تمام مراحل با 0x00 پر میشه. توجه داشته باشید که برای سازگاری با پروتکل BOOTP؛ هدر تغییر خاصی نکرده تا سرورهای DHCP قادر به راه اندازی دستگاه هایی که هنوز با BOOTP کار میکنند؛ باشند.

CHAddr : Client Hardware Address در تمامی مراحل چهارگانه، در 6 بایت ابتدایی این قسمت مک آدرس کلاینت نوشته میشه؛ مابقی بایت ها (Octet) با 0x00 پر میشن. توجه داشته باشید که این یه پروتکل عمومیه که ممکنه در شبکه هایی با آدرس های سخت افزاری با تعداد بایت متفاوت استفاده بشه لذا 16 بایت برای این قسمت در نظر گرفته شده.

SName یا Server Host Name یک رشته خاتمه یافته با null (رشته های متنی به فرمت زبان C) در اینجا ممکنه اسم هاست سرور نوشته بشه؛ اما عموما این قسمت و قسمت بعدی (با هم 192 بایت) با 0x00 پر میشن مگر اینکه در قسمت Options جا کم بیاد و نیاز به بایت های اضافی باشه که از این فضا استفاده میشه (در جای خودش میگیم)

File : در این قسمت آدرس و نام فایل بوت کلاینت قرار میگیره (نام فایل و نه خود فایل!). نام فایل ها هم رشته های خاتمه یافته با null هستند. ممکنه کلاینت هایی از این پروتکل استفاده بکنند که علاوه بر تقاضای آدرس آیپی و تنظیمات دیگه، نیاز به دانلود فایلی برای بوت هم داشته باشند. این قسمت هم معمولا با 0x00 پر میشه.

Options : گزینه های اختیاری که ممکنه در این ارتباط مورد نیاز باشند؛ در این قسمت و به فرم TLV نوع دوم ثبت میشن (یک بایت برای نوع گزینه، یک بایت برای تعداد داده های گزینه بغیر از دو بایت نوع و طول؛ و در ادامه داده های گزینه قرار می گیرند). اگر تعداد 312 بایت برای کل گزینه ها کافی نباشه؛ کاربر میتونه از قسمتهای sname و file هم استفاده بکنه. بدین ترتیب که از ابتدای بخش گزینه ها شروع میکنه به نوشتن. وقتی این قسمت پر شد از ابتدای sname مابقی گزینه ها رو مینویسه میاد پایین. از اونجاییکه ما فقط نیاز به داشتن IP داریم، از این قابلیت استفاده ای نمیکنیم؛ ولی یک پیاده سازی کامل از این پروتکل، باید تمامی این موارد رو رعایت بکنه. تعداد آپشن های تعریف شده برای این پروتکل فهرست بلندی داره که چنتاش رو در ادامه بررسی می کنیم. جهت بررسی فهرست کامل گزینه ها به RFC مربوطه یا نت مراجعه کنید. در صورتیکه تعداد بایت مورد نیاز، کمتر از 312 بایت باشه ، انتهای اون رو با آپشن 0xFF به عنوان انتهای بخش گزینه ها پر می کنیم ؛ همچنین بین گزینه ها میتوان با آپشن 0x00 عملیات pad رو انجام داد. توجه داشته باشید که در پروتکل DHCP گزینه 0xFF بعنوان end of option list است و نه 0x00. معمول هم اینه که افزودن 0x00 طوری صورت میگیره که گزینه ها در مرزهایی با مضرب 4 قرار بگیرند تا پردازش آنها در سمت گیرنده راحت تر باشه.

magic cookie چیست؟

گفتیم که فرمت هدر در پروتکل های BOOTP و DHCP یکسان هست. پس چگونه این دو از هم تشخیص داده میشن؟ سرور DHCP و همینطور نرم افزارهای شنود مثل wireshark) با استفاده از مفهوم magic cookie تشخیص میدن که پیام رسیده از نوع BOOTP هست یا DHCP. بدین صورت که اگر 4 بایت ابتدایی در قسمت Options با اعداد ثابت 99,130,83,99 یا 0x63825363 پر شده باشن؛ بدین معنی است که پیام ارسالی مبتنی بر DHCP است وگرنه داریم از BOOTP استفاده میکنیم. گزینه های مورد نیازمون هم بعد از 4 بایت magic cookie نوشته میشن.

معرفی تعدادی از آپشن های پروتکل DHCP :

مهمترین مورد در بخش گزینه ها؛ گزینه شما 53 با نام "Message Type" یا همان نوع پیام است. این گزینه فقط یک بایت داده داره در نتیجه 3 بایت اشغال میکنه و بصورت 0x35,0x01,0x-- استفاده میشه که 0x-- نوع پیام رو مشخص میکنه و شامل موارد زیر هست:

0x01 : DHCP Discover

0x02 : DHCP Offer

0x03 : DHCP Request

0x05 : DHCP ACK

0x06 : DHCP NACK

موارد دیگه ای هم برای نوع پیام هست که اینجا اعلام نشد(RFC 2132). در مرحله Discovery کلاینت این گزینه رو به صورت 0x35,0x01,0x01 میفرسته؛ در مرحله دوم سرور 0x035,0x01,0x02 و ...

گزینه مهم دیگه، گزینه شماره 50 با نام Requested IP Address هست. این گزینه 4 بایت داده برای IP مورد تقاضا (ترجیحی) هست و در نتیجه 6 بایت رو اشغال میکنه که دو بایت ابتدایی اون 0x32,0x04 و 4 بایت بعدی آدرس IP موردتقاضا رو نشون میده.

از گزینه های معروف دیگه میشه به گزینه 23 جهت مشخص نمودن مقدار پیش فرض TTL در یک ارتباط مبتنی بر IP ؛ گزینه شماره 6 برای تعیین سرور DNS ؛ گزینه شماره 51 جهت تعیین زمان اجاره و یا گزینه 52 برای وقتی که مقدار بایت های بخش آپشن، جهت قرار دادن گزینه های ارسالی کافی نباشه. با این گزینه، گیرنده متوجه میشه که بخش های SName و File نیز دربرگیرنده بخشی از گزینه ها هستند. برای اطلاعات بیشتر در مورد گزینه های پروتکل DHCP به سند RFC 2132 مراجعه کنید.

یه مرور کوچولو بکنیم. ابتدا کلاینت روشن میشه (و برای سادگی، فرض میکنیم که کابل شبکه هم متصله و همه چی گل و بلبله) کلاینت که فقط مک آدرس خودش رو میدونه؛ یک پیغام عمومی در لایه دو و سه شبکه (مک آدرس مقصد FF:FF:FF:FF:FF:FF ؛ آیپی آدرس مقصد 255.255.255.255؛ آدرس آیپی مبدا 0.0.0.0) با پورت مبدا 68 و پورت مقصد 67 روی شبکه میفرسته. بیت B هم ست میشه. در جواب، سرور که الان مک آدرس کلاینت رو داره؛ یک پیغام خصوصی در لایه دوم (با مک آدرس گیرنده یعنی کلاینت) و عمومی در لایه سوم (آیپی 255,255,255,255) با پورت مبدا 67 و مقصد 68 میفرسته. در مرحله سوم، کلاینت از بین پاسخ های دریافتی یکی رو انتخاب میکنه (در استاندارد، مواردی مثل زمان انتظار برای دریافت پیشنهاد و ... ذکر شده) و حالا؛ که هم مک آدرس و هم آیپی آدرس سرور رو داره؛ یک پیغام خصوصی به سرور میفرسته و در نهایت، در مرحله چهارم، سرور با تایید این عملیات به صورت خصوصی یا عمومی (با توجه به بیت B در مرحله سوم) کار رو تموم میکنه.

- از اونجاییکه بسیاری از ویندوزها، به صورت پیش فرض نمیتونن بصورت یک DHCP server عمل کنند؛ ما از یک نرم افزار جانبی برای ایجاد قابلیت DHCP server بر روی ویندوز استفاده کردیم. این نرم افزار

به نام DHCPsrv رو از dhcpserver.de/cms/download دانلود کنید (ورژن رایگانش 2.5.2.3)؛ سپس با توجه به راهنمایی های عنوان شده در فایل Readme.txt این قابلیت رو فعال کنید.

خب بریم سراغ کد؛ ما یه نرم افزار دانلود کردیم تا بتونیم ویندوز رو به صورت یک DHCP server فعال کنیم. در ابتدای برنامه؛ بعد از پیکربندی تراشه ENC (و احتمالا در یک حلقه بی نهایت) ما باید فرآیند دریافت آدرس IP رو باید انجام بدیم تا بعد از دریافت آدرس IP بتوانیم عملکرد عادی برد رو داشته باشیم. کدی که در میکروکنترلر نوشته شده؛ اینطوری طراحی شده که پس از دریافت یک پیام روی UDP؛ در صورت نیاز، بهش جواب میده. پیام مرحله 3 رو که در جواب پیام مرحله 2 میتونیم بفرستیم. پیام مرحله چهار هم که نیاز به جواب نداره. میمونه پیام مرحله اول که با این روش قابل ارسال نیست. پس ما باید یک فریم حامل پیغامی مبتنی بر DHCP/UDP/IP/Ethernt ii ایجاد و توسط تابع ENC28J60_TransmitFrame ارسال کنیم تا DHCP server رو در جریان خواسته خودمون قرار بدیم. طبیعیه که باید یک روتین مناسب برای این عملیات نوشته بشه. اما جهت درک ساده تر مفهوم ارسال درخواست DHCP Dicovery؛ این قسمت از کد رو به صورت بسیار ساده ای پیاده سازی کردیم. بدین ترتیب که یک فریم Ethernet ii رو طوری چیدمان کردیم که پروتکل های DHCP,UDP و IP رو شامل باشه. این کد بصورت زیر در فایل main.c و بعد از راه اندازی اولیه ENC توسط تابع ENC28j60_init پیاده سازی شده:

```
ENC28J60_Init();
                                                    //ایجاد هدر لایه دوم//
                //در این قسمت مک آدرس مقصد رو عمومی اعلام کردیم و مک آدرس خودمون رو هم بعنوان مبدا نوشتیم//
//Destination MAC Address ff:ff:ff:ff:ff:ff
frame.data[0]=0xff;frame.data[1]=0xff;frame.data[2]=0xff;frame.data[3]=0xff;frame.data[4]=0x
ff;frame.data[5]=0xff;
//Source MAC Address, for example 00:17:22:ed:a5:01
// or copy from enc28j60.c file
frame.data[6]=0x00;frame.data[7]=0x17;frame.data[8]=0x22;frame.data[9]=0xED;
frame.data[10]=0xA5;frame.data[11]=0x01;
                                                    //پروتکل مد نظرمون در لایه 3 رو آیپی معرفی کردیم//
frame.data[12]=0x08; frame.data[13]=0x00;  //IP protocol=0x0800
                                                    //حالا هدر پروتکل آیپی رو ست کردیم//
// IP Header
frame.data[14]=0x45; //VER=4, HLen=5
frame.data[15]=0x00; //TOS=0x00
frame.data[16]=0x01;  frame.data[17]=0x1A;  // total length of IP packet=UDP+20=282 bytes
frame.data[18]=0x00; frame.data[19]=0x00;  // ID=0x0000
```

```
frame.data[20]=0x40; frame.data[21]=0x00;  // Flag=0b010,Offset=0x00;
frame.data[22]=0x40; // TTL=64
frame.data[23]=0x11; // UDP =0x11 پروتکل
```

//مقدار چک سام آیپی رو قرار میدیم؛ پایین کد نوشتیم که چطور حساب کردیم//

```
// CheckSum for IP packet
frame.data[24]=0x39;
frame.data[25]=0xd4;
```

//آدرس IP خودمون رو هنوز نداریم و جاش صفر میذاریم، آدرس آیپی مقصد رو هم نمیدونیم و عمومی ارسال میکنیم//

```
frame.data[26]=0x00;frame.data[27]=0x00;frame.data[28]=0x00;frame.data[29]=0x00; //SA
frame.data[30]=255;frame.data[31]=255;frame.data[32]=255;frame.data[33]=255;      //DA
```

//حالا هدر پروتکل UDP رو میچینیم//

```
///// UDP Header
frame.data[34]=0x00; frame.data[35]=0x44; //Source port=68
frame.data[36]=0x00; frame.data[37]=0x43; //Destination port=67
```

//اینجا مقدار طول سگمنت رو نوشتیم!//

```
frame.data[38]=0x01; frame.data[39]=0x06;  // UDP data length+8=254+8=262=0x0106
```

//مقدار چک سام UDP رو هم اینجا مینویسم. طریقه محاسبه چک سام در پایین کد گفته شده//

```
// CheckSum for UDP segment
frame.data[40]=0x48; frame.data[41]=0xD5;
```

//حالا هدر پروتکل DHCP //

```
// DHCP Header
frame.data[42]=0x01; //operation Request=1
frame.data[43]=0x01; //HType ethernet=1
frame.data[44]=0x06; //MAC Address Length=6
frame.data[45]=0x00; //hops=0
//XID = 0xAABBCCDD
frame.data[46]=0xAA;frame.data[47]=0xBB;frame.data[48]=0xCC;frame.data[49]=0xDD;
frame.data[50]=0x00;frame.data[51]=0x00; //SECs=0
frame.data[52]=0x80;frame.data[53]=0x00; //B bit=1
frame.data[54]=0x00;frame.data[55]=0x00;frame.data[56]=0x00;frame.data[57]=0x00;//ciaddr=0
frame.data[58]=0x00;frame.data[59]=0x00;frame.data[60]=0x00;frame.data[61]=0x00;//yiaddr=0
frame.data[62]=0x00;frame.data[63]=0x00;frame.data[64]=0x00;frame.data[65]=0x00;//siaddr=0
frame.data[66]=0x00;frame.data[67]=0x00;frame.data[68]=0x00;frame.data[69]=0x00;//giaddr=0
///MAC Address 16bytes(octet)
frame.data[70]=0x00;frame.data[71]=0x17;frame.data[72]=0x22;frame.data[73]=0xED;//chaddr
frame.data[74]=0xA5;frame.data[75]=0x01;frame.data[76]=0x00;frame.data[77]=0x00;//chaddr
frame.data[78]=0x00;frame.data[79]=0x00;frame.data[80]=0x00;frame.data[81]=0x00;//chaddr
frame.data[82]=0x00;frame.data[83]=0x00;frame.data[84]=0x00;frame.data[85]=0x00;//chaddr
```

// 192 بایت 0x00 برای بخش های sname,file //

```
//192 bytes(sname,file) =0x00
for(int k=0;k<192;k++)
 {frame.data[86+k]=0x00;}
```

//تنظیم مقدار magic cookie //

```
frame.data[278]=0x63; frame.data[279]=0x82; frame.data[280]=0x53; frame.data[281]=0x63
```

//تنها گزینه ارسالی؛ گزینه شماره 53 با مقدار 0x01 به معنای پیام دیسکاوری//

```
frame.data[282]=0x35;frame.data[283]=0x01;frame.data[284]=0x01;
```

// پدینگ و بستن انتهای پیام//

```
for(int k=285;k<295;k++)
{frame.data[k]=0x00;}
frame.data[295]=0xff; // end of option list
```

// و در نهایت ارسال پیام//

```
ENC28J60_TransmitFrame(&frame.data[0],296);
```

- آپشن 0xFF به معنای انتهای گزینه ها رو میتوان درست بعد از آخرین پیام قرار داد و نه انتهای بخش گزینه ها؛ همچنین در بین پیام ها یا در انتهای بخش گزینه ها، میتوان برای قرار دادن گزینه ها در مرزهای مضرب چهار، از پدینگ یعنی گزینه 0x00 استفاده کرد.
- برای محاسبه چک سام در پروتکل های IP, UDP قاعدتا باید تابع مناسبی بنویسیم. اما اینجا اونها رو مستقیم نوشتیم. برای محاسبه چک سام هم از یک حقه ساده استفاده کردیم. در نرم افزار wireshark محاسبه و بررسی مقدار چک سام رو فعال کردیم (Edit/Preferences/Advance). یکبار پیام رو ارسال کردیم تا ببینیم مقدار درست چک سام، چی باید باشه. مقدار مناسب رو در کد قرار دادیم و دوباره کامپایل و پروگرم کردیم! یادآوری کنیم؛ ممکنه بعضی از پیاده سازی های این دو پروتکل؛ درستی چک سام رو بررسی نکنند. اما نرم افزاری که بعنوان DHCP Server دانلود کردیم ، بررسی میکنه!
- یادمون هم نره که مقدار آیپی رو در فایل enc28j60.c به آدرس عمومی تنظیم کنیم:

```
uint8_t  ipAddr[IP_ADDRESS_BYTES_NUM] = {255, 255, 255, 255};
```

برای پیغام مرحله سوم (Request) در پاسخ Offer هم از کدی که تا الان داشتیم؛ استفاده کردیم. چون آیپی برد ما در حال حاضر 255,255,255,255 هست؛ پس کدی که الان در اختیار داریم؛ پکت های عمومی آیپی رو دریافت میکنه و چون پیغام روی UDP ارسال میشه، کافیه در تابع UDP_Process تغییرات لازم رو اعمال کنیم. اولین قدم اینه که بررسی کنیم آیا روی پورت 68 چیزی اومده یا نه؟

```
else if(destPort == UDP_DHCP_PORT)          //DHCP client=68
```

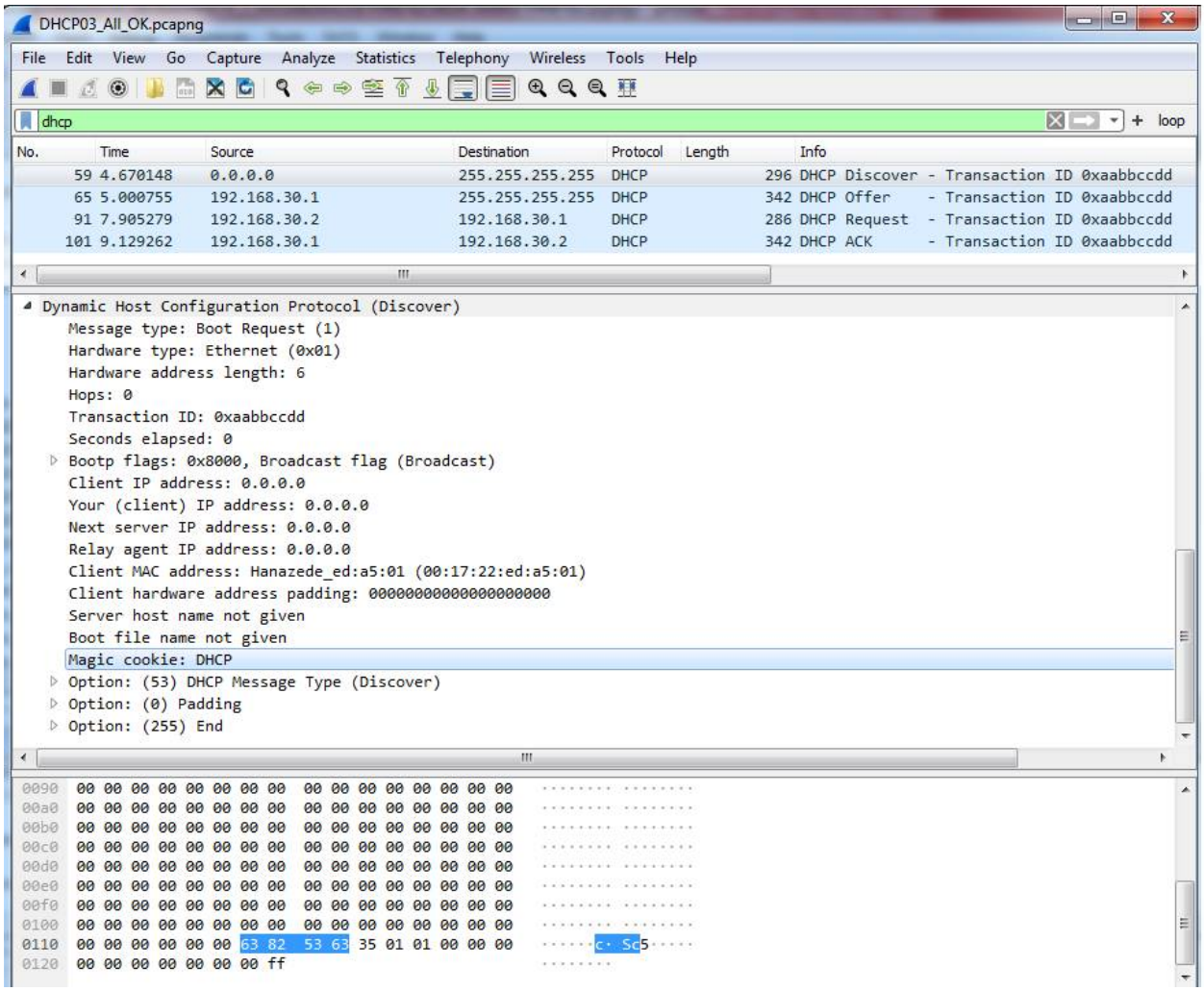
گام بعدی اینه که ببینیم آیا پیغام فعلی پیغام Offer یا ACK هست. برای این کار در ابتدای بخش گزینه ها، اول باید مجیک کوکی رو پیدا کنیم و بعد اگر مقدار موجود در گزینه 53 ، عدد 2 بود (یعنی پیغام Offer) باید پاسخ پیام که همون پیغام Request هست رو بفرستیم. مقدار آیپی واقعی رو هم از بخش yiaddr بدست میاریم و به عنوان آدرس IP خودمون ذخیره میکنیم. البته بهتر بود بعد از دریافت Ack انجام بشه.

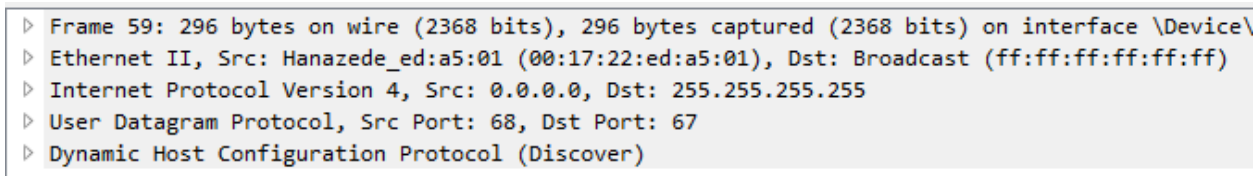
```
ipAddr[0]=udpFrame->data[16]; ipAddr[1]=udpFrame->data[17];
ipAddr[2]=udpFrame->data[18]; ipAddr[3]=udpFrame->data[19];
```

- پیغام ACK نیاز به جواب نداره.
- در پاسخ به پیغام Offer میتونید بیت B رو ریست کنید تا پیغام ACK بصورت خصوصی برای شما ارسال بشه.

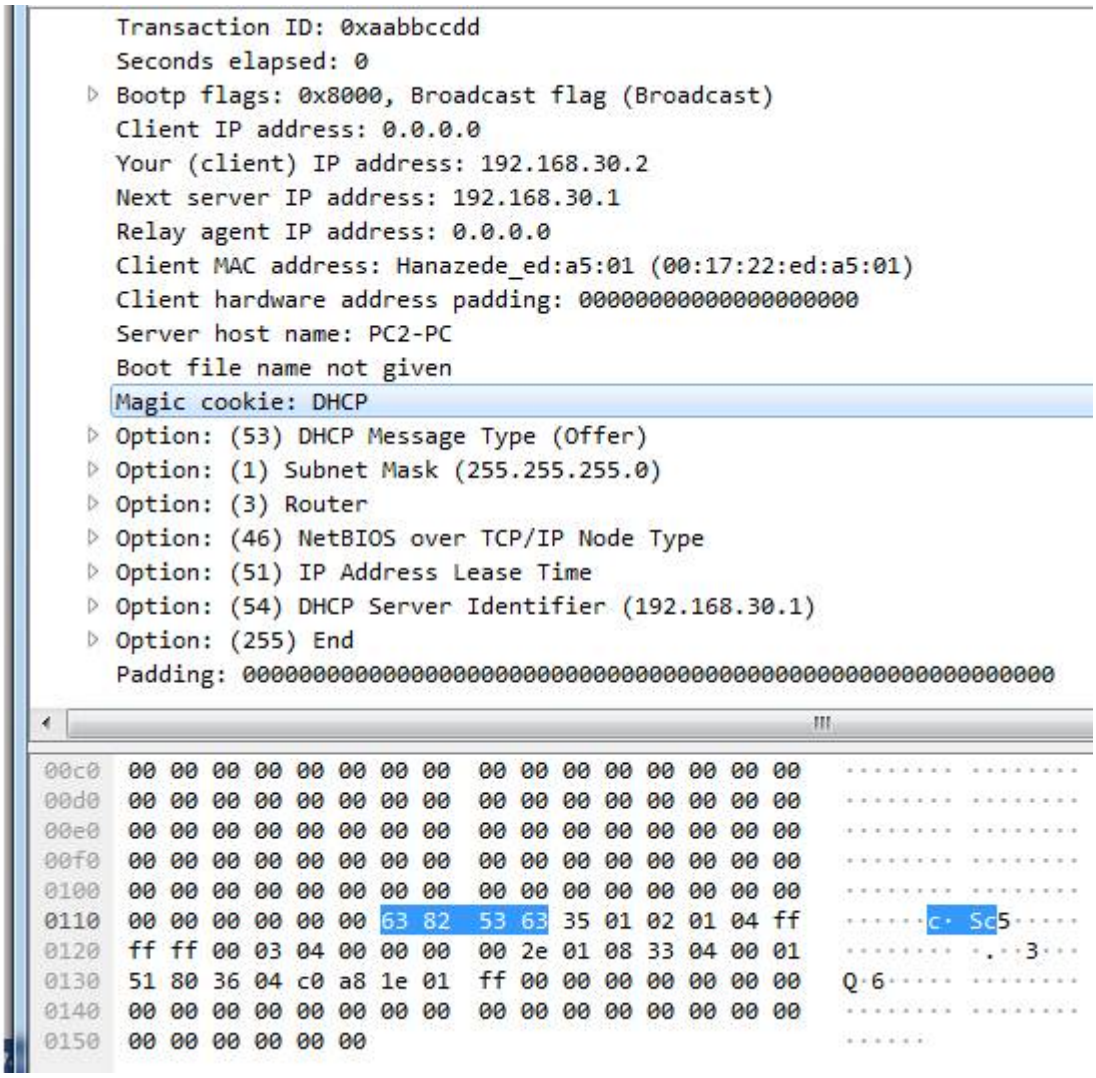
برای تست؛ آدرس IP کامپیوتر رو 192,168,30,1 قرار دادیم و محدوده مجاز برای پیشنهاد آدرس رو هم 192,168,30,2 تا 192,168,30,255 ست کردیم. بریم ببینیم wireshark این پیغام ها رو چطور ثبت کرده:

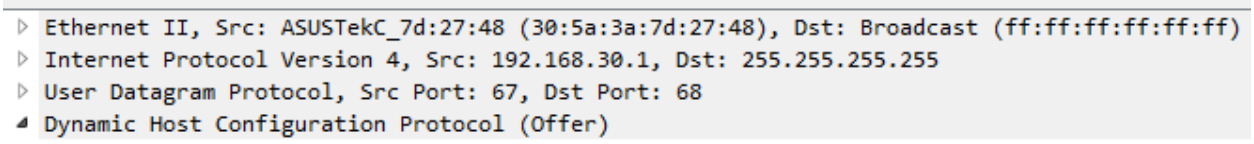
پیغام Discover فرستاده شده توسط کلاینت (میکروکنترلر):





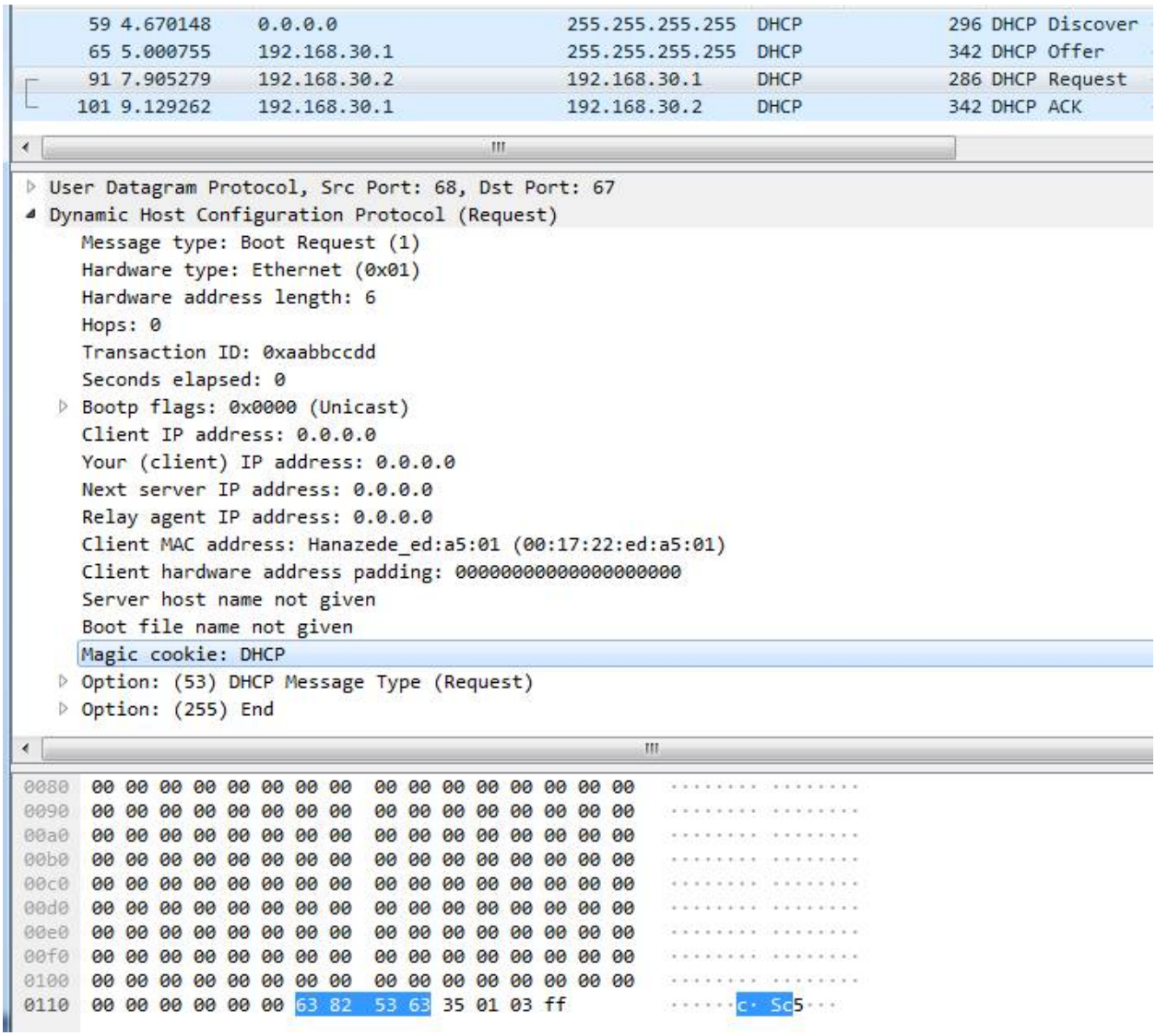
پیغام Offer ارسال شده توسط سرور:

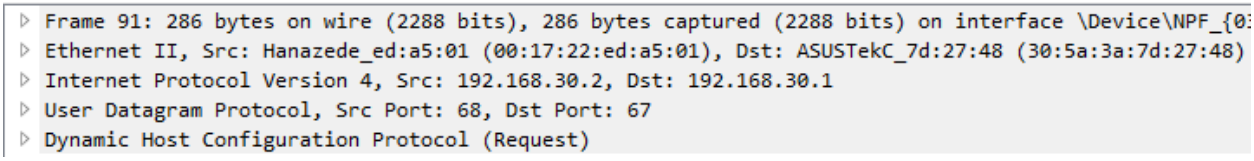




- همانطور که دیده می شود علاوه بر مقدار آدرس آیپی؛ تعدادی گزینه دیگه هم مثل subnet mask ؛ Lease Time و ... توسط سرور فرستاده شده.

پیغام Request که میکروکنترلر در پاسخ به Offer به سرور برگردونده:





و در نهایت پیغام Ack توسط سرور:

```
▷ User Datagram Protocol, Src Port: 67, Dst Port: 68
◢ Dynamic Host Configuration Protocol (ACK)
    Message type: Boot Reply (2)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0xaabbccdd
    Seconds elapsed: 0
  ▷ Bootp flags: 0x0000 (Unicast)
    Client IP address: 0.0.0.0
    Your (client) IP address: 192.168.30.2
    Next server IP address: 192.168.30.1
    Relay agent IP address: 0.0.0.0
    Client MAC address: Hanazede_ed:a5:01 (00:17:22:ed:a5:01)
    Client hardware address padding: 00000000000000000000
    Server host name: PC2-PC
    Boot file name not given
    Magic cookie: DHCP
  ▷ Option: (53) DHCP Message Type (ACK)
  ▷ Option: (1) Subnet Mask (255.255.255.0)
```

```
00c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0110  00 00 00 00 00 00 63 82  53 63 35 01 05 01 04 ff   ......c· Sc5·····
0120  ff ff 00 03 04 00 00 00  00 2e 01 08 33 04 00 01   ........ ....3···
0130  51 80 36 04 c0 a8 1e 01  ff 00 00 00 00 00 00 00   Q·6····· ........
0140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0150  00 00 00 00 00 00                                 ......
```

```
▷ Frame 101: 342 bytes on wire (2736 bits), 342 bytes captured (2736 bits) on interface \Device\NPF_{
▷ Ethernet II, Src: ASUSTekC_7d:27:48 (30:5a:3a:7d:27:48), Dst: Hanazede_ed:a5:01 (00:17:22:ed:a5:01)
▷ Internet Protocol Version 4, Src: 192.168.30.1, Dst: 192.168.30.2
▷ User Datagram Protocol, Src Port: 67, Dst Port: 68
▷ Dynamic Host Configuration Protocol (ACK)
```

بعد از عملیات هم اگر پنجره برنامه DHCP sever رو باز کنید (کلیک راست روی آیکن نرم افزار در تسک بار ویندوز و انتخاب open status) باید همچین چیزی ببینید :

بعد از اتمام پروسه تخصیص آدرس IP؛ با دستور پینگ؛ تنظیم شدن آدرس آیپی رو چک کنید.

پروتکل TCP :

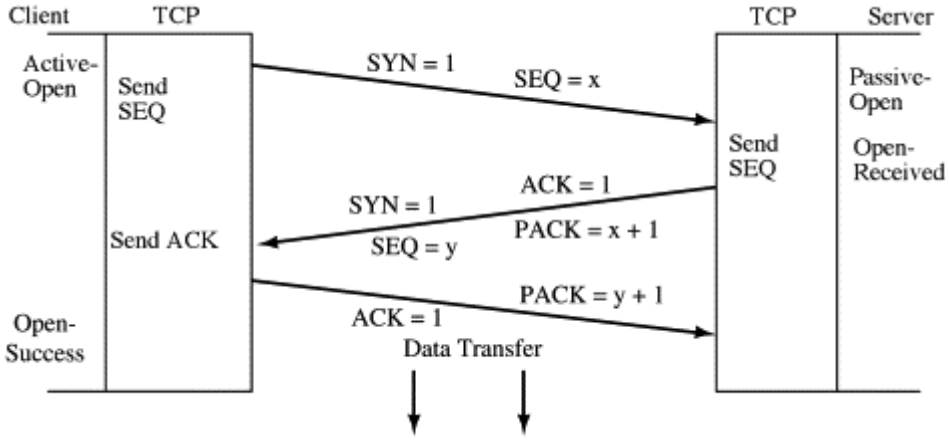
و اما رسیدیم به، احتمالا مهمترین و معروفترین پروتکل شبکه یعنی TCP. در بخش معرفی UDP گفتیم که UDP یک ارتباط بدون اتصال (ConnectionLess) هست؛ بدین معنی که دو طرف یک ارتباط، قبل از ارسال داده، گفتگو یا مذاکره ای با هم ندارند. TCP در مقابل این وضعیت قرار میگیره و به عبارتی اتصال گرا (Connection Oriented) است. تعریف اتصال گرایی TCP اینه که، قبل از ارسال داده و برای شروع این ارتباط؛ دو طرف مذاکره ی کوچک اما مهمی انجام میدن و در حین تبادل داده نیز، وضعیت ارتباط بطور مرتب چک خواهد شد.

یک ارتباط همواره با درخواست از طرف کلاینت آغاز میشه. توجه داشته باشید که اصطلاحات کلاینت و سرور در تعریف استاندارد TCP نیامده (در UDP هم نیامده)؛ و هر دو طرف بدون توجه به کلاینت یا سرور بودن، توان ارسال و دریافت داده رو دارند. اما برای تفهیم بهتر، در متون مرتبط، از مفهوم کلاینت/سرور استفاده شده؛ ما نیز از این کلمات استفاده خواهیم کرد.

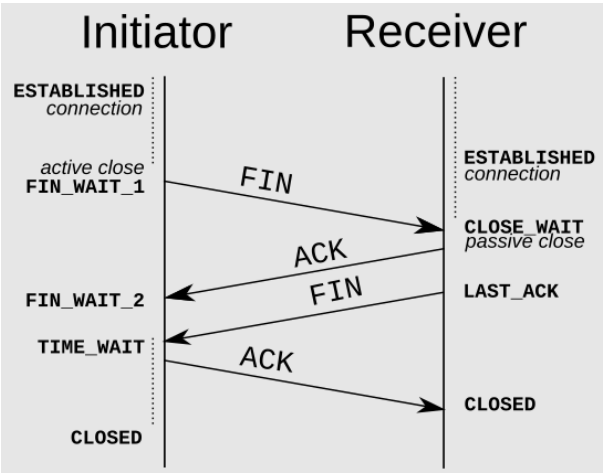
قبل از اینکه هر یک از دو طرف، بتوانند داده ای ارسال کنند؛ در یک پروسه ی 4 مرحله ای، یک ارتباط، ایجاد و پایدار (Stable) می شود. ابتدا کلاینت؛ درخواست برقراری ارتباط را به همراه تنظیماتی که در این ارتباط نیازمند آن است؛ به سرور میفرستد. در مرحله دوم، سرور این درخواست را تایید (Acknowledge) میکند. در گام سوم مجددا سرور، تنظیمات مورد نیاز خود را به اطلاع کلاینت میرساند و در مرحله آخر (چهارم) پیغام مرحله 3 توسط کلاینت تایید (Ack) می شود. در این نقطه، اصطلاحا ارتباط پایدار (Stable) شده است. از این نقطه به بعد، هر دو دستگاه می توانند به طرف مقابل، داده ای ارسال کنند. هر سگمنتی از داده که ارسال می شود؛ دارای شماره شناسایی است و طرف مقابل بایستی دریافت سگمنت را با ارسال پیام ACK با فرمت تعریف شده؛ اطلاع دهد. چنانچه داده ها به ترتیبی غیر از ترتیب ارسال به گیرنده برسد(به دلیل وجود روترها در میانه راه)؛ گیرنده با استفاده از شماره های شناسایی؛ می تواند داده ها رو به نحو صحیح در محل خود گذاشته و استفاده کند (بعنوان مثال هنگام ارسال یک فایل بزرگ).

در انتهای کار و برای بستن این ارتباط نیز، پروسه ای 4 مرحله ای انجام می شود. سمتی که دیگر داده ای برای ارسال ندارد(کلاینت یا سرور)؛ پیغام درخواست اتمام ارتباط را می فرستد و منتظر دریافت ACK از طرف مقابل می شود. در این زمان، این سمت اصطلاحا در وضعیت close wait قرار میگیرد. این انتظار به این دلیل است که سمت دیگر، شاید هنوز داده ای برای ارسال داشته باشد. سمت دیگر نیز بعد از اینکه تمام داده های خود را ارسال کرد؛ پیغام اتمام خود را بایستی ارسال کرده و منتظر دریافت ACK از سمت مقابل بماند. به این حالت هم اصطلاحا Fin wait گویند.

از آنجایی که در آغاز ارتباط؛ مراحل 2 و 3 از سمت سرور، معمولا در یک پیغام انجام می گیرد؛ چهار مرحله در 3 پیغام خلاصه می شود. به این مراحل، اصطلاحا "دست دهی سه مرحله ای" (3 phase Handshake) یا "فاز دست دهی سه گانه" گفته می شود. نام دیگر آن، مرحله یا فاز SYN است. اتمام یک ارتباط، اما؛ بطور معمول در همان 4 مرحله انجام می پذیرد.



شکل 23 برقراری ارتباط TCP



شکل 24 اتمام یک ارتباط TCP

در حالت عادی و قبل از شروع یک ارتباط؛ اصطلاحا سرور در حالت شنود (Listen) بر روی پورت مورد نظر قرار دارد. به این وضعیت Passive Open نیز گفته می شود. برای شروع یک ارتباط، کلاینت اولین سگمنت با نام SYN را ارسال می کند. به این عمل، اصطلاحا Active Open گفته می شود.

احتمالا مهمترین اطلاعاتی که هر دو سمت در یک ارتباط استفاده خواهند کرد؛ شماره ترتیب (Sequence Number) و شماره تایید (Acknoledge Number) است. اجازه بدید با یک مثال موضوع رو روشن کنیم. فرض کنید در یک ارتباط بین دو هاست؛ که ما آنها را A و B نامگذاری کردیم، هاست A قصد ارسال یک فایل 5000 بایتی را دارد. چون حجم فایل بزرگ است؛ فرستنده تصمیم میگیرد این فایل را به قسمت های 1000 بایتی تقسیم و ارسال کند. در ابتدا سیستم A **"شماره ترتیب"** خودش رو 1 اعلام میکنه و 1000 بایت اول رو به سیستم B ارسال میکنه. شماره ترتیب در این مرحله به این معنی است که بایت هایی که در این سگمنت هست؛ از بایت شماره 1 شروع میشوند، بالطبع چون 1000 بایت ارسال شده؛ بایت آخر، بایت هزارم هست. سیستم B بعد از دریافت این سگمنت، یک جواب تایید به A ارسال میکند که درون آن اصطلاحا **"شماره تایید"** خودش رو، 1001 اعلام میکنه، به این معنی که B به A اعلام میکنه؛ داده های تا قبل از بایت 1001 به درستی توسط B دریافت شده اند و B منتظر دریافت بایت هزار و یکم به بعد است. در گام بعدی؛ 1000 بایت دوم با seq num=1001 ارسال می شود؛ یعنی شروع داده ها در این سگمنت از بایت هزار و یکم هست. گیرنده با دریافت این سگمنت به فرستنده پیغامی میفرستد با Ack Num=2001 یعنی بایت ها تا بایت 2000 به درستی دریافت و منتظر دریافت بایت های 2001 به ببعد هست و الی آخر...

توجه داشته باشید؛ از آنجایی که هر دو سمت قادر به ارسال داده هستند؛ لذا هر دو طرف، همزمان دارای Seq Num و Ack Num متعلق به خود هستند. Seq Num برای داده های ارسالی و Ack Num برای داده های دریافتی، مورد استفاده قرار میگیرند.

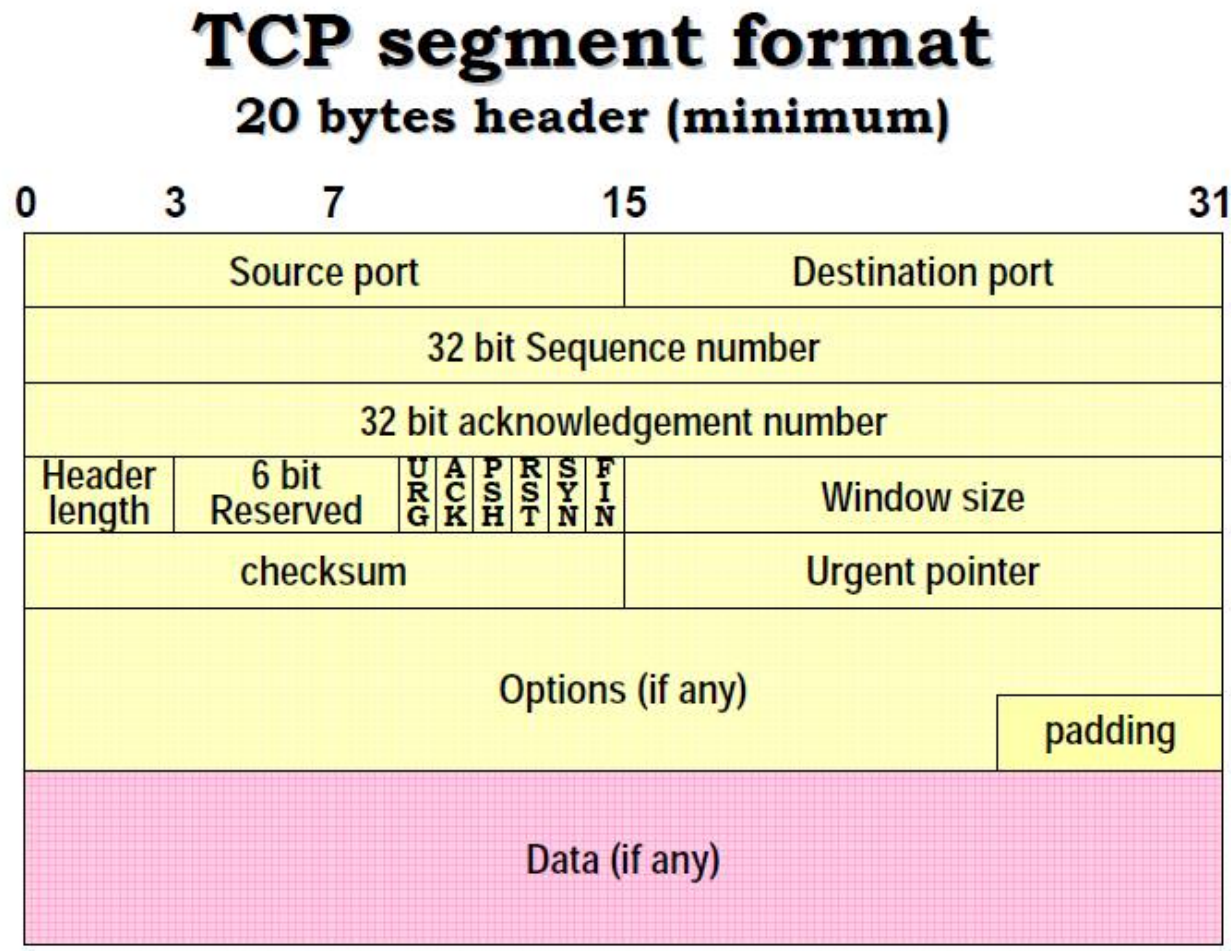
احتمالا اگر سوالاتی دارید مثل "اگر سگمنتی گم بشه؛ چی؟" یا "اگر سگمنت ها با ترتیبی غیر از ترتیب ارسال؛ به گیرنده برسند؛ چی؟" باید بگیم در پروتکل TCP تمهیداتی برای این موارد هم دیده شده که ما سعی میکنیم بعضی از اونها رو اینجا توضیح بدیم ولی برای اطلاع بیشتر میتونید به سند RFC 793 و بروزرسانی های اون مراجعه کنید.

نکته بسیار مهم هم اینکه؛ به دلیل رعایت موارد امنیتی و پیش بینی پذیر نبودن شماره ترتیب ها (Seq Num) و جلوگیری از بعضی حملات هکری، Seq Num از صفر شروع نمی شود و جهت این کار از یک عدد تصادفی استفاده میشود. پس Seq Num درای آفست هست.

- پروتکل TCP پروتکل پیچیده ایست، به همین دلیل اکثر قریب به اتفاق کدهای پیاده سازی این پروتکل برای میکروکنترلرها ، تنها بخش های اصلی و ضروری این پروتکل رو پیاده سازی کردند.
- پروتکل TCP در لایه چهارم مدل شبکه تعریف شده؛ لذا هر دو طرف ارتباط، از شماره پورت برای آدرس دهی در لایه چهارم استفاده میکنند. یادآوری میشه که شماره پورت یک عدد دو بایتی ست که بعضی از

آنها به پروتکل های لایه های بالاتر اختصاص یافته (مثل 67 و 68 برای DHCP) یا اینکه رزرو شده اند و نبایست در یک ارتباط TCP شخصی از آنها استفاده کرد.

طبق معمول، بریم سروقت هدر TCP :



Source & Dstination Ports : در این دو بخش که هر کدام دو بایت نیاز دارند؛ شماره پورت مبدا و مقصد اعلام می گردد.

Seq Num & Ack Num : همونطور که گفتیم؛ در این بخش های چهاربایتی، شماره ترتیب و تایید هر طرف اعلام میشه. در شروع یک ارتباط (هندشیک سه مرحله ای) مقدار اولیه (مقدار تصادفی که نشوندهنده آفست هست) در Seq Num قرار میگیره و بیت SYN که در بخش Flags قرار دارد؛ یک می شود. به عنوان مثال، کلاینت در آغاز هندشیک با SYN=1 مقدار SEQ NUM=12345 رو اعلام میکنه؛ یعنی اینکه عدد صفر از سمت کلاینت 12345هست؛ پس در مراحل انتقال داده، اولین Seq Num برابر با 12346 خواهد بود. این مقدار

آفست از طرف سرور و در هنگام ارسال Ack Num لحاظ می شود. سرور نیز در پاسخ (هنگام فاز دوم از هندشیک) ، همین کار را انجام می دهد؛ یعنی یک عدد تصادفی بعنوان مقدار آفست شماره ترتیب خودش اعلام میکنه و بیت SYN رو یک میکنه. علاوه بر این سرور در پیغام خودش بیت ACK که در بخش Flags قرار داره رو هم ست میکنه که دریافت پیام مرحله اول رو اطلاع بده.

Header Length : یا طول هدر؛ یک بخش 4 بیتی هست و چون اندازه هدر برحسب کلمات چهاربایتی (DWORD) محاسبه شده، عدد داخل آن باید در 4 ضرب بشه تا اندازه هدر بر حسب بایت بدست بیاید. حداقل مقدار در این بیت ها "0101" معادل 5 خواهد بود که یعنی هدر 20=4*5 بایت هست. ماکزیمم عدد هم 15 هست که یعنی هدر حداکثر 60 بایت خواهد بود. توجه داشته باشید که بخش اصلی هدر 20 بایتی هست اما یک بخش با نام Options در هدر قرار دارد که طول آن بین 0 (بدون Option) تا 40 بایت (حداکثر Option) تغییر می کند.

Flags: 12 بیت باقیمانده در کنار Heade Length عموما بنام پرچم (Flags) شناخته می شود و بصورت بیتی تفسیر می شوند. در استانداردهای گوناگون از 6 بیت تا نهایتا 9 بیت کم ارزش آن استفاده شده و مابقی بیت های پر ارزش، صفر (Zero) بوده و استفاده نمی شوند. بهمین دلیل ممکن است در نت، مواردی پیدا کنید که این بخش پرچم، بیت های بیشتری نسبت به تصویر قبلی داشته باشد؛ اما در تمامی آنها 6 بیت کم ارزش، بطور حتم وجود دارند که الزام استاندارد اولیه TCP هست. این بیت ها رو بطور جداگانه مورد بررسی قرار بدیم، ببینیم چی هستند :

FIN : یا پایان (Final)؛ هر گاه یک سمت ارتباط؛ داده دیگری برای ارسال نداشته باشه؛ این بیت رو یک میکنه. توجه داشته باشید که در اینجا ارتباط پایان نمیگیرد؛ چون ممکن است طرف مقابل همچنان داده هایی برای ارسال داشته باشد. همونطور که گفتیم برای قطع ارتباط بعد از اتمام ارسال داده از هر دو طرف؛ یک هندشیک 4 مرحله ای هم برای پایان دادن به ارتباط (Close) خواهیم داشت. ست شدن این بیت، نشوندهنده آغاز پروسه اتمام (close) در یک ارتباط است.

SYN : یا Synchron. هر گاه این بیت یک باشد؛ یعنی مقدار موجود در بخش Sequence Number مقدار اولیه (صفر یا آفست) را نشون میده! توجه داشته باشید که این بیت، تنها در مرحله یک و دو از هندشیک اولیه استفاده میشه و در این دو مرحله، هر طرف مقدار آفست خودش رو اعلام میکنه.

RST : یا Reset. از این بیت در هندشیک 4 مرحله ای در پایان ارتباط (Closing) استفاده میشه. هر گاه این بیت یک بشه؛ فرستنده ی این بیت به طرف مقابل اعلام میکنه که ارتباط رو خاتمه یافته فرض میکنه و دیگه به پیغام های طرف مقابل جواب نمیده.

PSH : مخفف Push است. گفتیم که در حالت عادی، هر سمت یک مقدار داده رو در سگمنت های مختلف ارسال میکنه (مثلا یک فایل بزرگ رو). پس در حالت عادی، گیرنده باید صبر کنه تا تمام داده ها رو بگیره و بعد از اون ها استفاده بکنه. ست کردن این بیت، به طرف گیرنده میفهمونه که دیگه منتظر سگمنت بعدی نمونه و داده هایی که تا اینجا ارسال شده رو استفاده کنه.

Ack : بیت تایید (Acknowledge). در استاندارد گفته شده، ست بودن این بیت در یک سگمنت، یعنی اینکه Ack Num واقع در سگمنت معتبر هست. بطور کلی ست کردن این بیت، به طرف دیگه میفهمونه اطلاعاتی دریافت شده؛ همین! و در واقع، جز در مرحله یک از هند شیک ابتدایی، در تمام سگمنت های ارسالی بعدی، این بیت یک خواهد بود؛ به این معنی که پیام قبلی دریافت شده است!

URG : مخفف Urgent یا اضطراری؛ این بیت همراه با بخش Urgent Pointer استفاده میشه. هر گاه این بیت یک بشه؛ یعنی بخش Urgent Pointer حاوی اطلاعات با اهمیت هست. هر گاه در حین یک ارتباط نرمال؛ نیاز باشه که داده هایی بصورت ضروری و بدون صبر کردن برای دریافت مابقی بایت ها؛ مورد پردازش قرار بگیره؛ از این بخش ها استفاده میشه. بذارید یه مثال بزنیم. فرض کنید مدار الکترونیکی شما یه برد کنترلی هست شامل یک سنسور دمای حساس بعلاوه چند سنسور و عملگر (رله، ولو Valve، لامپ) دیگه و برد در هر سگمنت، اطلاعات یک بخشی از سنسورها رو میفرسته و احتمالا از سمت مقابل دستوراتی میگیره که کدام خروجی ها رو فعال کنه. حالا سنسور دمای اصلی؛ یک دمای غیرعادی نشون میده، برد با یک کردن بیت URG و ارسال مقدار سنسور دما به طرف مقابل میفهمونه که یک حالت ضروری پیش اومده که نیاز به پردازش فوری داره. در نتیجه گیرنده این سگمنت رو بطور ویژه ای پردازش میکنه و منتظر دریافت اطلاعات مابقی سنسورها نمی مونه.

Window Size : با استفاده از این بخش، هر طرف به طرف مقابل میگه که چقدر حافظه برای دریافت داده داره. بهتره که باز هم با یک مثال موضوع رو روشن تر کنیم. فرض کنید فرستنده در حال ارسال یک فایل بسیار حجیم هست. گیرنده برای دریافت؛ یک حافظه 10 کیلوبایتی در RAM اختصاص داده؛ ابتدا مقدار Window Size رو همون 10KB اعلام میکنه و هر بار با دریافت مقداری داده؛ مقدار window size رو کاهش میده و داده ها رو در رم خودش مینویسه؛ وقتی دیگه حافظه ش پر شد، window size رو صفر اعلام میکنه تا فرستنده کمی صبر کنه. بعد داده ها رو از رم به درون فایلی در hard disk منتقل میکنه و حالا دوباره به فرستنده مقدار window size رو 10KB اعلام میکنه تا فرستنده دوباره شروع به ارسال داده بکنه. توجه داشته باشید که تعداد زیادی از این بخش ها در واقع تمهیدات لازم برای یک ارتباط مطمین هستند و لزومی نداره کاربر از تمام اونها استفاده بکنه. ما هم از اونجایی که همواره در حال پردازش هستیم از این بخش و بعضی قابلیت های دیگه TCP

استفاده نخواهیم کرد مگر اینکه سرعت ارسال بالا باشه و میکروکنترلر عقب بمونه. در این حالت میتونید این بخش رو صفر اعلام کنید تا طرف مقابل کمی صبر کنه.

Check Sum : بخش چک سام. در اینجا هم مثل پروتکل UDP شامل هدر، داده ها و بخش شبه هدر (psudoHeader) هست. جهت یادآوری بخش شبه هدر از آدرس های IP فرستنده/گیرنده، Zero+TCP protocol=0x0006 و اندازه طول سگمنت ارسالی هست. توجه داشته باشید که بدلیل بزرگ بودن بخش محاسباتی و زمانبر بودن انجام این محاسبات؛ نیاز هست که یه کد سریع برای این کار نوشته بشه و یا در صورت امکان در سخت افزار انجام بشه.

Urgent Pointer : یا اشاره گر به بخش ضروری. چنانچه نیاز باشه که داده ها بصورت ضروری (با اولویت بالا و بدون انتظار برای رسیدن مابقی اطلاعات) پردازش بشن، بیت URG که بالاتر معرفی کردیم، ست میشه؛ داده های ضروری در ابتدای بخش داده ها نوشته و ارسال میشن و در این قسمت، آدرس آخرین بایت بخش Urgnet قرار میگیره تا گیرنده متوجه بشه که داد های ضروری از ابتدا تا کجای بخش داده ها هستند. بقیه داده هم که داده عادی تلقی خواهند شد؛ بعد از بخش urgent نوشته میشن. برای مثال فرض کنید سگمنت فعلی؛ بخش option نداره و داده ها از بایت بیست و یکم به بعد در یک سگمنت نوشته شده اند. حالا فرض کنید تعداد 10 بایت، داده ضروری داریم. بیت URG ست میشه و در این قسمت عدد 10 نوشته میشه به این معنی که از بایت 21 تا 30 شامل داده های ضروری هست و داده های عادی از بایت سی و یکم به بعد نوشته شده اند.

Options : یا بخش گزینه ها. این بخش چنانچه وجود داشته باشد؛ باید اندازه آن مضربی از 4 باشه؛ چون طول آن برحسب کلمات 4بایتی (DWORD) محاسبه و اندازه آن به بخش Header Length اضافه میشه. در صورتی که اندازه بخش Option مضربی از 4 نباشد، در انتهای آن بایت هایی با مقدار 0x00 اضافه میشود (padding).

در استاندارد TCP تعداد زیادی گزینه (Option) معرفی شده است؛ همچنین گزینه هایی در حال بهینه سازی و استفاده خصوصی هستند که ممکنه در آینده به استاندارد اضافه بشه. اما یک هاست، طبق الزام استاندارد، حداقل باید تعداد خاصی از آنها را پشتیبانی (support) نماید.

همانطور که در بخش UDP و مبحث TLV گفتیم؛ بطور کلی گزینه ها دو حالت دارند:

- یا یک بایتی هستند که تنها شامل دو گزینه نوع End of Option List=0x00 و No Operation=0x01 هست. توجه داشته باشید که در پروتکل TCP انتهای گزینه ها با 0x00 و در پروتکل DHCP انتهای گزینه ها با 0xFF مشخص میشد.

❖ یا با فرمت TLV نوع دوم هستند؛ یعنی دارای یک بایت برای نوع گزینه؛یک بایت برای اندازه گزینه به همراه تعداد مشخصی بایت، برای تنظیمات یا مقادیر گزینه هستند. در بخش اندازه، دو بایت نوع و طول هم در نظر گرفته می شوند؛ لذا در بخش اندازه، حداقل عدد 2 (برای حالت بدون مقدار) قرار می گیرد.

در ادامه تعدادی از مهمترین گزینه ها رو بررسی می کنیم.

MSS= Maximum Segment Size.

Type=0x02, Length=0x04, Value=2Bytes

آپشن نوع 2 ؛ چهار بایتی هست، از این آپشن برای اینکه به طرف مقابل بفهمانیم در هر سگمنت حداکثر چه تعداد بایت میتواند ارسال کند، استفاده میشود. **این آپشن فقط در دو مرحله ابتدایی هندشیک اولیه (مراحل SYN) میتواند وجود داشته باشد**. از آنجایی که برای مقدار MSS دو بایت در نظر گرفته شده؛ حداکثر سایز داده ارسالی 65535 بایت هست.

- طبق استاندارد، گزینه های 1,0 و 2 باید توسط هر اجرای (Implementation) پروتکل TCP لحاظ شود. یعنی کد میکروکنترلرتون هرچقدر ساده باشه، باید این سه گزینه رو بتونه دریافت و پردازش کنه.
- این گزینه رو با wondow size اشتباه نگیرید. Window size میزان کل حافظه در دسترس برای دریافت سگمنت های متوالی رو نشون میده. در حالی که گزینه شماره 2 حداکثر تعداد بایت های واقع در یک سگمنت رو اعلام میکنه.

WS= Window Scale.

Type=0x03, Length=0x03, Value=1Byte

در هدر بخشی با نام Window Size وجود داشت که هر طرف به طرف مقابل میفهماند که چقدر حافظه دیگر برای دریافت اطلاعات دارد. این بخش دوبایتی بود؛ لذا حداکثر عدد داخل آن 65535 میتواند باشد. چنانچه هاستی، دارای مقدار بزرگتری حافظه باشد؛ از گزینه WS برای بزرگتر کردن مقدار window size میتونه استفاده کنه. همانطور که در تعریف این گزینه دیده می شود؛ این گزینه یک بایتی ست و عدد داخل آن بصورت توانی از 2 تفسیر می شود. فرض کنید ws=4 و window size= 1024 باشد، اندازه اصلی window size برابر با 16384 بایت خواهد بود : $1024 \times 2^4 = 1024 \times 16 = 16384$

SACK Permit= Selective Acknowledge Permit.

Type=0x04,Length=0x02 without value.

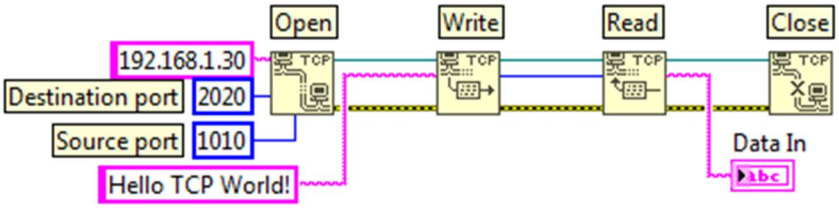
SACK = Selective Acknowledge.

Type=0x05,Length=10,18,26 or 34

این دو گزینه معمولا با هم استفاده می شوند. یادتون باشه گفتیم سگمنت ها همراه با Seq Num ارسال میشن. فرض کنید، در حال ارسال یک فایل 10 کیلو بایتی در سگمنت های 1 کیلوبایتی هستیم. سگمنتهای اول و دوم و چهارم به بعد بدرستی در گیرنده دریافت شده اند؛ اما سگمنت سوم به دلایلی به مقصد نرسیده است. در استاندارد اولیه که این گزینه وجود نداشت؛ گیرنده، در قسمت Ack Num تنها میتوانست دریافت داده ها تا انتهای سگمنت دوم را تایید کند و فرستنده مجبور بود که سگمنتهای سوم تا دهم رو مجدد ارسال کنه (retransmission) اما در سال 1996 و در سند RFC 2018 این گزینه معرفی و مورد استفاده قرار گرفت. هدف، حل این ایراد بود. **طرفی که قادر به پذیرش SACK هست؛ در مرحله هندشیک اولیه (مرحله SYN) آنرا با استفاده از گزینه SACK Permit=0x04 اعلام میکند** و طرف مقابل چنانچه بخواهد از این قابلیت استفاده کند؛ حداکثر میتواند چهار SACK داشته باشد. در مثال بالا یک SACK برای سگمنت های اول و دوم و یک SACK برای سگمنت های چهارم تا دهم استفاده خواهد شد (دو SACK). در نتیجه فرستنده داده، متوجه فقدان SACK برای سگمنت سوم شده و تنها آن را بازارسال خواهد کرد. همانطور که دیده می شود(بخش اندازه رو ببینید) برای هر SACK، هشت بایت در نظر گرفته شده است. چهار بایت برای شروع و چهار بایت برای اعلام پایان هر SACK. در واقع با استفاده از این قابلیت، دیگه نیاز نیست برای هر سگمنت یک ACK جداگانه ارسال بشه. برای اطلاع از مابقی گزینه ها به اسناد RFC مراجعه و یا در نت جستجو کنید.

و اما اجرای کد TCP :

با توجه به پیکربندی نرم افزار میکروکنترلر و به جهت سادگی (چون پیغام اول رو کلاینت میفرسته) حالت TCP server رو در میکرو پیاده سازی می کنیم. کلاینت (PC) یک ارتباط رو در پورت مورد نظر رو باز میکنه (open)؛ تعدادی داده رو ارسال (TCP send) و سعی میکنه به همون تعداد داده بخونه و بعد ارتباط رو میبنده (close). برای اجرای این کد در محیط LabVIEW قطعه کد زیر را استفاده کرده ایم:



در میکروکنترلر به درخواست ارتباط جواب مثبت میدیم و داده های ارسالی از طرف کامپیوتر رو به خودش برمیگردونیم(echo). اولین پیام دریافتی؛ اولین سگمنت از هندشیک شروع خواهد بود. کامپیوتر به عنوان کلاینت، درخواستی برای شروع ارتباط به میکروکنترلر ارسال میکنه، بالطبع اولین قدم اینه که در تابع IP_Process بررسی کنیم که آیا پکت فعلی (در پروتکل IP) یک پکت شامل سگمنت TCP هست یا نه؟ کد تابع IP_Process بصورت زیر تغییر میکنه:

```
if (ipFrame->protocol == IP_FRAME_PROTOCOL_ICMP) //==0x01
  {
   newDataLen = ICMP_Process((ICMP_EchoFrame*)ipFrame->data, dataLen);
  }
else if (ipFrame->protocol == IP_FRAME_PROTOCOL_UDP) // ==0x11
  {
   newDataLen = UDP_Process((UDP_Frame*)ipFrame->data, dataLen);
  }
else if (ipFrame->protocol == IP_FRAME_PROTOCOL_TCP) // ==0x06
  {
   newDataLen = TCP_Process((TCP_Frame*)ipFrame->data, dataLen,ipFrame);
  }
```

گام بعدی، تعریف ثوابتی مثل شماره پورت و بیت های بخش flags است؛ آفست Seq num رو هم مقدار ثابت 0x1A درنظر گرفتیم؛ هرچند بهتره این مقدار تصادفی انتخاب بشه. در بعضی مراجع مقدار آفست رو init seq num هم گفته اند. همچنین ساختاری برای پردازش هدر TCP تعریف میکنیم و یک define برای بدست آوردن طول قسمت Options در صورت وجود.

```
#define  TCP_DEMO_PORT   2020
#define  TCP_MYSEQ       0x1A
#define  OPTION_LENGTH(x)  ( ( (x >> 12) & 0x000f) * 4 - 20 )

#define  TCP_FIN_FLAG  (1<<0)
#define  TCP_SYN_FLAG  (1<<1)
#define  TCP_RST_FLAG  (1<<2)
```

```
#define   TCP_PSH_FLAG   (1<<3)
#define   TCP_ACK_FLAG   (1<<4)
#define   TCP_URG_FLAG   (1<<5)

//tcp header length 20~60 (option 0~40)bytes
typedef struct TCP_Frame
{
 uint16_t   srcPort;
 uint16_t   destPort;
 uint32_t   seqnum;
 uint32_t   acknum;
 uint16_t   flags;
 uint16_t   winsize;
 uint16_t   chksum;
 uint16_t   urgptr;
 uint8_t   data[]; // options+data
} TCP_Frame;
```

مطابق روالمون تا اینجا، تابعی بنام TCP_Process مینویسیم که کدهای این پروتکل رو اجرا کنه. در این تابع؛ ابتدا دو مورد رو بررسی میکنیم. آیا پیغام، روی پورت مدنظر ما ارسال شده است؟ و آیا چک سام صحیح است؟

```
if ( (ntohs(tcpFrame->destPort)) == TCP_DEMO_PORT)
 {
        uint16_t  calcCheckSum = TCP_CalcCheckSum (( uint8_t* ) tcpFrame, frameLen ) ;
         if ( rxCheckSum == calcCheckSum )
         {

         }

 }
```

اگر این موارد صحیح باشد، وارد قسمت اصلی پردازش میشویم. بخش Flags رو در یک متغیر ذخیره می کنیم :

```
flag = htons(tcpFrame->flags);
```

پکت های دریافتی سه حالت خواهند داشت:

- یا پیغام SYN دریافت و در پاسخ seq num ارسال میشه. مقدار window size هم 8192 اعلام شده که برای ما فعلا بی اهمیت هست. در بخش Option هم، تنها گزینه MSS با مقدار 1460 فرستاده

شده (20 بایت برای هدر TCP؛ 20 بایت برای هدر IP و 18 بایت سربار در فریم Ethernet ii لذا از 1518 بایت 1460 باقی می ماند).

```
if( flag & TCP_SYN_FLAG)//it is SYN packet => create SYNACK packet
{
 temp=0x0000;
 tcpFrame->flags = htons((temp | TCP_ACK_FLAG | TCP_SYN_FLAG | (((uint16_t)6)<<12)));
myack=(ntohl(tcpFrame->seqnum)) +1;
tcpFrame->acknum= htonl(myack);
tcpFrame->seqnum = htonl(TCP_MYSEQ);
myseq=TCP_MYSEQ+1;
tcpFrame->winsize = htons(8192);
tcpFrame->urgptr = 0;

//only MSS Option(0x02) . for this example MSS=1460=0x05B4
tcpFrame->data[0]=0x02;
tcpFrame->data[1]=0x04;
tcpFrame->data[2]=0x05;
tcpFrame->data[3]=0xB4;
newFrameLen = 24;
}
```

- یا داده های اصلی دریافت شده؛ که عین آن به کامپیوتر برگشت داده می شود(echo). مقدار win Size همچنان ثابت هست؛ چون ما هر سگمنتی رو در لحظه پردازش میکنیم(اگر با wireshark سگمنت ها رو رصد کنید متوجه میشید که بیت push هم از طرف فرستنده همواره ست شده است). کد برای فهم آسانتر در ساده ترین حالت نوشته شده است:

```
else if(flag & TCP_PSH_FLAG)
{
 temp=0x0000;
tcpFrame->flags = htons((temp | TCP_ACK_FLAG | TCP_PSH_FLAG | (((uint16_t)5)<<12)));
myack=(ntohl(tcpFrame->seqnum)) +(frameLen-20);
tcpFrame->acknum= htonl(myack);
tcpFrame->seqnum = htonl(myseq);
myseq+=(frameLen-20);
tcpFrame->winsize = htons(8192);
newFrameLen = frameLen;
}
```

- یا کامپیوتر با ارسال پرچم FIN درخواست پایان (با اجرای تابع Close TCP در کد داخل کامپیوتر) ارتباط را داده است؛ لذا ما هم با ست کردن بیت های FIN, RST ارتباط را میبندیم (Close). در این حالت ارتباط از سمت ما، تنها با یک پیام بسته می شود :

```
else if(flag & TCP_FIN_FLAG)
{
 temp=0x0000;
 tcpFrame->flags = htons((temp | TCP_ACK_FLAG | TCP_FIN_FLAG | TCP_RST_FLAG |
(((uint16_t)5)<<12)));
 myack=(ntohl(tcpFrame->seqnum)) +(frameLen-20);
 tcpFrame->acknum= htonl(myack);
 tcpFrame->seqnum = htonl(myseq);
 myseq+=(frameLen-20);
 newFrameLen = frameLen;
}
```

- توجه داشته باشید که با توجه به نوع پیاده سازی؛ سگمنت دریافتی در مرحله سوم هندشیک؛ پاسخی ندارد!
- در تمام سگمنت های ارسالی از سمت سرور (میکروکنترلر) بیت Ack یک خواهد بود، در سمت کلاینت نیز، تنها در سگمنت اول این بیت صفر است و در مابقی سگمنت ها این بیت یک می شود.

در اینجا به پایان این نوشتار رسیدیم؛ اگر وقتی باشه و حوصله ای؛ احتمالا پروتکل HTTP هم به متن اضافه خواهد شد. اما اگر شما تونستید تا اینجا خودتون رو برسونید؛ براحتی میتونید پروتکل های دیگه رو هم خودتون اجرا کنید. در ادامه به ضمیمه ها خواهیم پرداخت و در بخش نهایی نیز تمامی کدها قرار داده خواهند شد. موفق باشید.

ضمیمه 1 : معرفی اجزای یک شبکه واقعی:

در ارتباط اترنتی که پیاده سازی کردیم؛ تنها دو قطعه موجود بود که بطور مستقیم توسط کابل شبکه به هم متصل بودند. اما یک شبکه واقعی دارای تعداد بسیار زیادی دستگاه و تجهیزات ارتباطی در شبکه است. سعی میکنیم در ادامه این تجهیزات را معرفی و عملکرد آنها را تعریف کنیم. در شبکه های ابتدایی مبتنی بر اترنت ، ارتباط با استفاده از کابل های هم محور (کواکسیال ؛ مثل کابل آنتن تلویزیون) برقرار میشد. ارتباط به صورت یک باس (Bus) بوجود می آمد و تمام دستگاه ها؛ هم زمان میتوانستند از باس بخوانند یا هنگام آزاد بودن خط؛ اطلاعات خود را ارسال کنند. یک مشکل در این ارتباط؛ این بود که چنانچه کابل از جایی قطع می شد؛ ارتباط دو طرف بریدگی قطع میشد. از طرفی به دلیل استفاده از یک خط؛ وضعیت تصادم (Collision) بسیار پیش می آمد. تصادم به حالتی میگویند که دو یا چند دستگاه، بطور همزمان، قصد ارسال داده های خود را داشته باشند، فرض کنید افرادی دور یک میز در جلسه ای، مشغول صحبت هستند. برای فهم درست صحبت ها، در هر لحظه تنها یک نفر میتواند صحبت کند. افرادی که قصد صحبت داشته باشند؛ باید منتظر بمانند تا فرد در حال صحبت، ساکت شود. وقتی گوینده، حرفاش تموم بشه؛ ممکنه دو یا چند فرد، قصد صحبت داشته باشند؛ درنتیجه در گفتگو اختلال ایجاد میشه و حرفها نامفهوم خواهد بود! هر چه تعداد نفرات بیشتر باشد؛ احتمال تصادم بیشتر خواهد بود. لذا برای پرهیز از تصادم؛ الگوریتمی بنام CSMA/CD = Carrier Sense-Multiple Access/Collision Detection بوجود آمد. اگه بخواهیم این اسم گذاری رو تحلیل کنیم، اینطور باید بگیم: قابلیت احساس سیگنال حامل (شنود خط و انتظار برای برقراری سکوت) با قابلیت دستیابی چند نفر همزمان و تشخیص تصادم! این الگوریتم با توزیع تصادفی فرآیند "صبر و تلاش" سعی در حل مشکل داشت. بدین صورت که هر فرد برای آغاز صحبت، باید منتظر میماند تا خط ساکت شود و سپس شروع به صحبت نماید. اما از آنجاییکه همزمان خود نیز در حال شنیدن است؛ چنانچه متوجه تصادم شود؛ برای مدت زمان محدودی که به صورت تصادفی مشخص شده است، منتظر می ماند و سپس دوباره اقدام میکند. اگر باز هم تصادم پیش بیاید؛ مدت زمان انتظار، بصورت هندسی افزایش میابد تا بالاخره بتواند داده خود را ارسال کند و یا بعد از مقدار تکرار مشخصی از ارسال فریم خودداری کرده و گزارش خطا می دهد. از آنجاییکه مقدار انتظار، تصادفی ست؛ معمولا یکی از افراد موفق به بیان جملات خود میشود و در نتیجه شرایط برای صحبت کردن دیگرانی که منتظر صحبت بودند؛ فراهم خواهد شد. به سادگی میتوان فهمید که هرچند این روش مشکل را حل می نماید؛ اما در نهایت، سرعت نهایی انتقال داده در خط، بسیار پایین خواهد بود. لذا برای حل کامل این مشکل، قطعاتی بنام سوییچ اختراع شد. همچنین متذکر شویم که این الگوریتم تنها در سرعت 10Mb, Half Douplex مورد استفاده قرار می گیرد.

Switch :

در شبکه های فعلی، دستگاه های واقع در یک زیرشبکه (زیرشبکه بخش مشخصی از یک شبکه است که دارای آدرس مشخص است، طوریکه قسمت پر ارزش IP آدرس در یک زیرشبکه، ثابت است) به طور مستقیم به شبکه های دیگر متصل نیستند؛ بلکه به واسطه روتر ها به شبکه های دیگر متصل خواهند بود. ارتباط در درون شبکه نیز توسط سوییچ ها صورت می گیرد. در واقع تمامی دستگاه های درون شبکه فقط به سوییچ متصل میشوند و از طریق آن با دستگاه های دیگر در زیرشبکه خود، ارتباط دارند. سوییچ دستگاهی ست که تعدادی پورت (معمولا با ضرایب 24 مثل 24و48و96) دارد. در اینجا منظور از پورت؛ پورت های اترنت روی سوییچ هستند، با پورت در پروتکل های لایه 4 اشتباه نشود! هر سوییچ، درون خود جدولی بنام switch table دارد که ارتباط مک آدرس ها و پورت های روی سوییچ را مشخص میکند (هر مک آدرس به کدام پورت سوییچ متصل است). در ابتدا این جدول خالی است. هر دستگاهی که پیغامی ارسال میکند؛ ابتدا این پیام وارد سوییچ می شود و در اینجا مک آدرس آن و شماره پورتی که که از طریق آن به سوییچ متصل است؛ در جدول ذخیره میشود. از آنجایی که در مراحل ابتدایی، هنوز جدول کامل نیست؛ سوییچ نمی داند که این پیام را به کدام مقصد ارسال کند؛ پس پیغام را به تمامی پورت های دیگر ارسال می کند. هر پورتی که جواب پیغام را بدهد؛ مک آدرس خود را هم به سوییچ اعلام کرده است. لذا بعد از مدت زمان کمی، جدول تکمیل می شود. متوجه شدیم که سوییچ ها، مک آدرس را ذخیره می کنند پس در لایه دوم از مدل OSI کار میکنند.

- ارتباط در زیرشبکه؛ وقتی که سوییچ ها حضور دارند؛ اصطلاحا به صورت ستاره است (در قیاس با حالت اصلی که به صورت باس بود)
- از اونجایی که در زیرشبکه های دارای سوییچ، هیچ دو هاستی با هم مرتبط نیستند؛ لذا مشکل تصادم بوجود نخواهد آمد و دیگر نیاز به استفاده از الگوریتم CSMA/CD نیست و ارتباط میتواند Full Douplex باشد.

HUB :

هاب نیز مثل سوییچ؛ وسیله ای ست برای ارتباط دهی چند هاست. برخلاف سوییچ ها، هاب ها پیغام های روی یک پورت را عینا به پورت های دیگر ارسال میکنند و هیچ پردازشی انجام نمیدهند. بطور کلی میتوان گفت که سوییچ ها، هوشمند اما هاب ها، غیرهوشمند رفتار می کنند؛ لذا درون خود، هیچ جدول ذخیره شده ای ندارند. احتمالا قبلا با هاب های پورت (port یا درگاه) USB آشنا بوده اید. فرض کنید کامپیوتر شما، تنها یک پورت

USB دارد. ولی شما نیاز دارید که همزمان از کیبرد، ماوس و حافظه فلش که با پورت USB کار میکنند؛ استفاده کنید، در این حالت شما نیاز به یک هاب دارید. عموما هاب ها یک پورت خاص بنام uplink دارند و اطلاعات دریافتی از پورت های دیگر را به uplink منتقل کرده یا از uplink گرفته و به تمام پورت های دیگر منتقل میکنند. هاب در شبکه نیز چنین فعالیتی دارد. هاب ها به نسبت سوییچ ها؛ تجهیزات قدیمی تری هستند و در شبکه های جدید استفاده نمی شوند. هاب ها به دو نوع پسیو (غیرفعال) و اکتیو (فعال) هم تقسیم بندی میشوند. نوع پسیو تنها مثل یک کانکتور عمل میکنه یعنی یک سیگنال رو به شاخه های دیگه منتقل میکنه اما هاب های فعال همانند یک ریپیتر چند کاناله کار میکنند.

Repeater :

یا تکرارکننده، یک اصطلاح عمومی ست در تبادل سیگنال. در فواصل زیاد معمولا سیگنال اصلی توسط نویز خراب میشه. لذا اگر فاصله زیادتر از حد استاندارد باشه؛ نیاز هست در میانه راه، سیگنال اصلی تقویت و باز ارسال (retransmit) بشه. این وظیفه ریپیتر ها هست. از جاهایی که ریپیتر ها، زیاد استفاده میشن؛ تقویت امواج موبایل و پخش دوباره اون در فضاهایی است که امواج موبایل ضعیفی دارند. یک گیرنده، امواج رو در موقعیت مناسب (مثلا بالای یک برج مسکونی) دریافت و اون رو در طبقات زیرزمین مجددا منتشر میکنه. ریپیترها در شبکه نیز وظیفه ای جز تقویت و ارسال دوباره سیگنال ها ندارند.

Router :

روتر یا مسیریاب برای ارتباط دو زیرشبکه با هم استفاده می شوند. در واقع بدون روتر، ارتباط ما با شبکه هایی با وسعت جهانی برقرار نمیشد. فرض کنید نیاز دارید یک زیرشبکه با آدرس 192.168.1.X را به زیرشبکه ی 192.168.7.X متصل نمایید؛ در این حالت از روتر استفاده میشود. روترها دو یا چند شبکه رو بهم اتصال میدن و در لبه یا مرز زیرشبکه ها قرار میگیرند. وقتی که یک هاست اطلاعاتی را به سوییچ می فرستد؛ سوییچ بررسی میکند که مقصد پیام در زیرشبکه خودش قرار دارد یا نه؟ اگر مقصد در همان زیر شبکه باشد، پیام تحویل مقصد میشود ولی چنانچه مقصد پیام در شبکه دیگری باشد (مثلا موقعی که شما میخواید به سایت google وصل بشید) سوییچ این پیغام رو به gateway تحویل میده. gateway نام دیگر روتری ست که زیرشبکه شما رو به شبکه های دیگه وصل میکنه. روتر اگر بدونه پیغام رو باید به کدوم هاست در زیرشبکه دوم (در سمت دیگرش) تحویل بده؛ (با استفاده از سوییچ واقع در زیرشبکه دوم) این کار رو میکنه و گرنه پیغام رو به یک روتر دیگه که

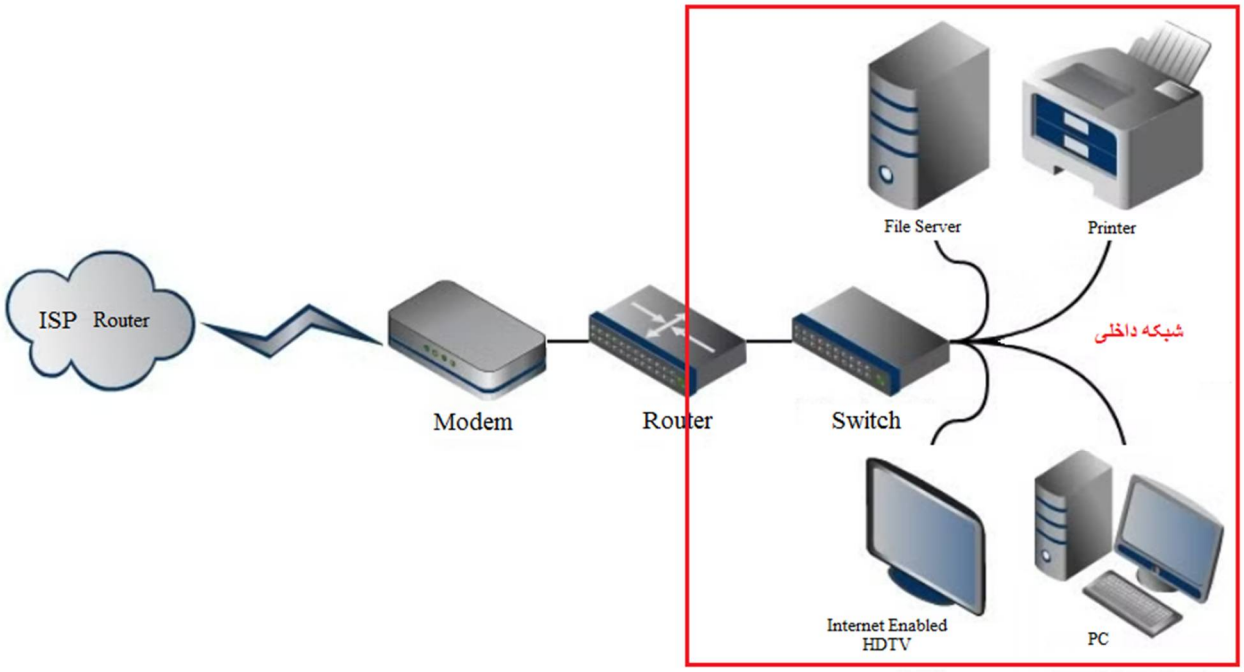
براش تعریف شده؛ تحویل میده. روترها دارای دو یا چند پورت هستند که هر کدام آدرسی از یک زیرشبکه را دارند. در ادامه مثال بالا؛ روتر ما، احتمالا در یک پورت، آدرس 192.168.1.5 و در دیگری، آدرس 192.168.7.3 را دارا خواهد بود. روترها در لایه سوم کار می کنند؛ لذا روترها نیز جدولی بنام route table در خود دارند که آدرس IP تجهیزات متصل به خود را نگه میدارد. همچنین روترها دارای جدولی بنام ARP table هستند که برای دخیره اطلاعات لایه دوم استفاده میشود. اگر در خاطر داشته باشید در پکت IP بخشی بنام TTL وجود داشت و گفتیم که هرگاه روتری یک پیغام را دریافت کند، از مقدار TTL یک واحد کم میکند (جهت یادآوری؛ مجبور میشه چک سام رو هم دوباره حساب کنه). اگر مقدار TTL صفر شود، روتر پیغام را رها کرده (بدلیل اینکه پیغامی تا ابد در شبکه نچرخد!) و پیغام خطایی با استفاده از پروتکل ICMP به فرستنده برمیگردونه. به عملیاتی که روتر انجام میده، اصطلاحا hop کردن میگن. لذا دو اصطلاح hop و router معادل هم هستند. اولی نام عمل و دومی دستگاه اجرا کننده عمل هاپ هست. به جهت کاهش ترافیک یا بررسی سرعت ارسال؛ روتر های چندپورت، ممکن است بسته های دریافتی رو از طرق مختلف به مقصد برسونند و لزوما مسیر یک پکت از مبدا تا مقصد ثابت نیست.

- سوییچ هایی بنام سوییچ لایه 3 تولید و عرضه شده اند که عملا کار سوییچ و روتینگ رو با هم انجام میدن.

Bridge:

یا پل، وسیله ایست برای ارتباط دو بخش جدا از هم در یک زیرشبکه؛ پس این قطعات در لایه دوم فعالیت دارند و عموما در شبکه هاییکه انواع گوناگون ارتباط (مثلا یک مودم که هم ارتباط اترنتی داره و هم وای فای) دارند، نیاز هست؛ یا در تقسیم بندی یک شبکه به چند زیرشبکه که از نظر فیزیکی جدا از هم اما از نظر آدرس فیزیکی در یک زیرشبکه قرار دارند؛ مورد استفاده قرار میگیرد. پل ها اصولا برای ارتباط شاخه ها یا بخش های مختلف یک زیرشبکه به هم استفاده میشن در قیاس با روترها که برای ارتباط دو زیر شبکه مختلف به هم استفاده میشدن. به عنوان مثال، تجهیزاتی که بعنوان مودم ADSL در منازل استفاده میشن؛ دارای پل هم هستند؛ چون هم ارتباط اترنتی، و هم وای فای دارند. همچنین این مودم ها روتر نیز هستند و زیرشبکه خانگی شما رو به یک روتر در ISP شما متصل میکنند. سوییچ هم هستند، چون شبکه داخلی منزل شما رو مدیریت میکنند. وظیفه اصلیشون هم که مودم بودنه؛ یعنی تبدیل سیگنال های دیجیتال (’1‘ و ’0‘) به سیگنال های آنالوگی که روی خط تلفن ارسال بشن.

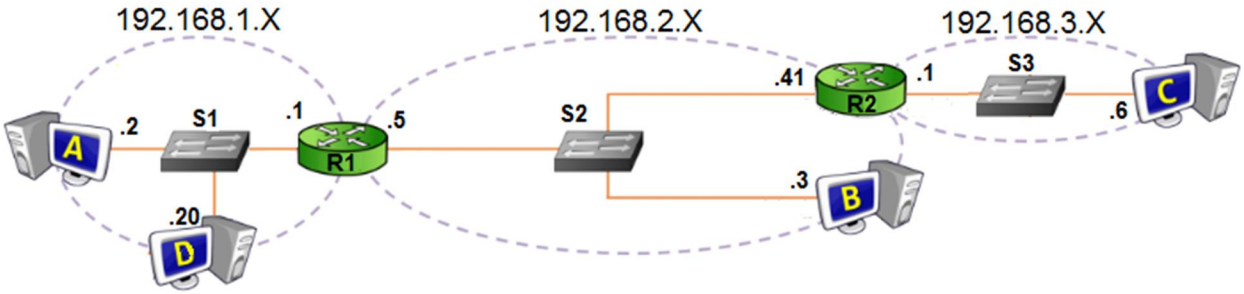
یک زیر شبکه واقعی، عموما به صورت شکل زیر پیاده سازی میشه:



همه هاست ها، درون زیرشبکه خودشون، تنها از طریق سوییچ؛ با هم ارتباط دارند و برای ارتباط با شبکه های دیگه از روتر استفاده می کنند که اون هم به طور معمول، حداقل با یک روتر دیگه، ارتباط داره. توجه داشته باشید که سوییچ ها، فریم ها رو دستکاری نمی کنند؛ هرچند اطلاعات لایه دو رو برای مصرف خودشون در داخل جدولی به نام switch table ذخیره می کنند. اما روترها نیاز دارند که فریم ها و پکت ها رو دستکاری کنند تا مسیر پکت رو به مقصد، مشخص کنند. همچنین اطلاعات لایه 3 و بعضی اطلاعات لایه 2 رو در دو جدول به نام های route table,ARP table نگه داری می کنند. route table آدرس شبکه های موجود در روتر رو نگه میداره؛ یعنی شبکه هایی که روتر در مرز اون ها قرار داره و قراره این شبکه ها رو بهم متصل بکنه. ARP table هم اطلاعات هاست های در دسترس رو نگه میداره؛ یعنی هاست هایی که روتر با اونها تشکیل یک شبکه رو دادند و در یک زیرمجموعه از آدرس قرار دارند. تفاوت دیگه بین سوییچ و روتر در اینه که اصولا سوییچ عضوی از شبکه نیست، یعنی دارای آدرس (نه مک و نه آیپی) نیست؛ درحالی که روتر عضوی از شبکه است و هم مک آدرس و هم آدرس آیپی داره.

- یکی از مواردی که هر هاست باید بدونه؛ علاوه بر آدرس (مک و آیپی) های خودش و subnet mask آدرس آیپی default gateway یا همون روتر در زیرشبکه ست. مک آدرس روتر هم میتونه با ARP بدست بیاد.

اجازه بدید سفر پکت از یک هاست به هاست مقصد رو با یک مثال روشن کنیم. شکل زیر یک شبکه ساده اما واقعی رو نشون میده که در اون، سه زیرشبکه از طریق دو روتر به هم متصل شده اند. توجه داشته باشید که یک روتر می تونه بیش از دو شبکه رو بهم وصل کنه.



در ساده ترین حالت، فرض کنید هاست A بسته ای به مقصد D در زیرشبکه خودش ارسال میکنه. از اونجاییکه در این حالت؛ هاست A نیاز به مک آدرس D داره؛ لذا قبل از ارسال پکت و احتمالا با استفاده از پروتکل ARP، مک آدرس D (همچنین مک آدرس روتر R1) رو بدست آورده است. این بسته پس از خروج از A به سوییچ S1 میرسه؛ اگر سوییچ بداند مقصد کجاست (با بررسی مک آدرس مقصد در لایه دوم)؛ بسته رو تحویل مقصد میده و گرنه اون رو به تمام پورت های خودش (به جز پورت فرستنده) میفرسته. در هر دو حالت فریم به D خواهد رسید. در این ارتباط، روترها درگیر نمی شوند.

حالا فرض کنید A قصد ارسال پیامی به B دارد. در پیام هایی که متعلق به زیرشبکه فعلی نیست؛ هاست نیاز به دونستن مک آدرس مقصد نداره (و به همین دلیل هست که پیام های ARP از زیرشبکه خارج یا اصطلاحا مسیریابی نمی شوند). در این حالت، مک آدرس مقصد در فریم ارسالی، مک آدرس روتر (R1 (gateway باید باشه تا سوییچ S1 فریم رو به روتر R1 بفرسته. توجه داشته باشید که روتر R1 در سمت چپ خودش آدرس آیپی 192.168.1.1 و در سمت راست آدرس 192.168.2.5 رو داره. همچنین در هر کدوم از دو شبکه هم؛ مک آدرس خودش رو داره که برای شلوغ نشدن، در تصویر نوشته نشده اند. روتر R1 بعد از دریافت این فریم؛ با بررسی آیپی مقصد، متوجه میشه که این پکت برای زیرشبکه سمت راست خودش هست. اگر بدونه B کجاست؛ یک فریم

جدید میسازه که با مک آدرس خودش و هاست B پر شده، ولی داده های پکت IP و همینطور آدرس های IP در فریم اولیه تغییر نکرده اند. یکسری کارهای دیگه مثل کاهش TTL و محاسبه مجدد چک سام رو هم انجام میده و درنهایت، این فریم جدید رو از طریق S2 برای B ارسال میکنه. اما اگر ندونه B چه آدرسی در شبکه سمت راست داره؛ یک پیغام عمومی در زیرشبکه میفرسته یا سعی میکنه قبل از ارسال؛ مک آدرس B رو بدست بیاره. توجه داشته باشید که روترها و همچنین سوییچ ها در ابتدای شروع به کار شبکه؛ فرایند یادگیری رو خواهند داشت که شامل تعداد زیادی روش یادگیری هست. در این فرآیند یادگیری که گاهی اوقات به طور مستقیم و توسط ادمین شبکه مشخص میشه؛ اطلاعاتی که روتر به اون ها نیاز داره، بدست میاد. در مثال فعلی، روتر، مک آدرس B رو از قبل داره و پیام به مقصد میرسه.

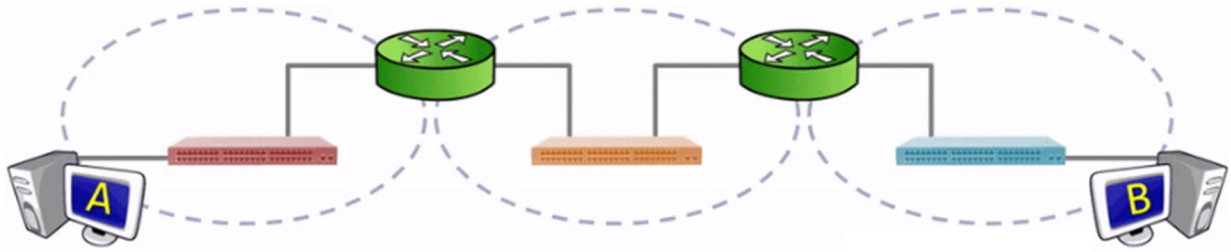
در حالت سوم هاست A تصمیم داره پکتی برای هاست C ارسال کنه. تا رسیدن پیام به R1 مثل وضعیت قبل هست اما اینجا R1 با بررسی آیپی آدرس مقصد، متوجه میشه که این پیام متعلق به زیرشبکه دیگریست. در این وضعیت هم اگر بدونه شبکه 192.168.3.X در سمت روتر R2 هست؛ پیام رو به R2 ارسال میکنه (مک آدرس خودش و مک آدرس R2 در شبکه 192.168.2.X استفاده میشه در فریم لایه دوم) و گرنه با استفاده از اطلاعات لایه سوم از فریم اصلی؛ فریم جدیدی درست میکنه و به روتر پیش فرض خودش (که توسط ادمین شبکه تنظیم شده) ارسال میکنه. طی فرآیند یادگیری، مسیرها بطور بهینه ای بدست می آیند و لذا در حالت عادی، مسیر رسیدن به یک شبکه در هرجای دنیا ثابت هست(دستور trace route رو ببینید)؛ اما همچنان روترها میتوانند برای ارسال یک پکت، از میان مسیرهای موجود، یکی را انتخاب کنند و پکت رو از مسیر دیگه ای ارسال کنند. (مثلا برای جلوگیری از ترافیک در یک سمت) به همین دلیل گفته میشود که پکت ها از مسیرهای گوناگون به مقصد می رسند.

دستور tracert :

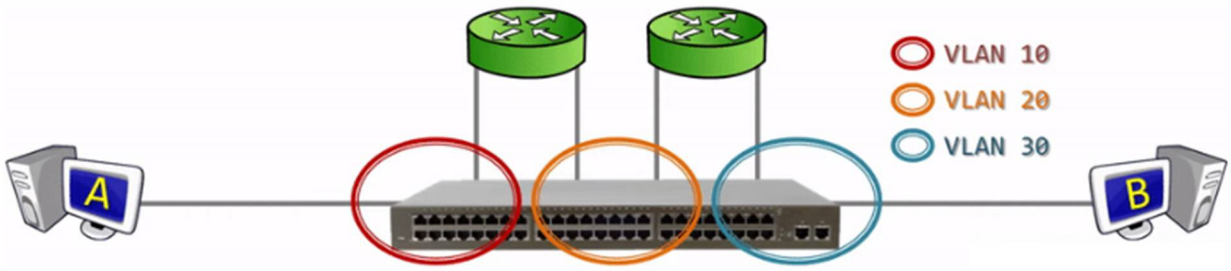
عملیات trace route به همراه ping دو عملیات یا دستور مهم هستند که با استفاده از پروتکل ICMP اطلاعاتی از وضعیت شبکه به ما میدن. پینگ رو که قبلا معرفی و پاسخ پینگ رو پیاده سازی کردیم. عملیات trace route هم اینطور عمل میکنه که ابتدا مقدار TTL رو 1 قرار میده و بسته ای برای هاست مقصد، میفرسته. بالطبع اولین روتر در مسیر باعث میشه TTL صفر بشه و پیغامی به فرستنده ارسال بشه به این ترتیب آدرس IP اولین روتر که همون gateway هست درمیاد. در مرحله بعد، TTL برابر با 2 قرار داده میشه؛ این بار آدرس دومین روتر در مسیر بدست میاد و .... شما با استفاده از این عملیات میتونید مسیرهایی که پیغام شما برای رسیدن به مثلا یک سایت خاص طی میکنه رو بدست بیارید.

ضمیمه 2 : معرفی VLAN و استاندارد IEEE 802.1Q

VLAN سرنام Virtual LAN یا شبکه مجازی است. یک شبکه ساده متشکل از سه زیر شبکه ؛ با تعداد هاست های کم، مثل شکل زیر رو در نظر بگیرید:



گفتیم که سوییچ ها عموما دارای تعداد زیادی پورت سخت افزاری هستند. در طرح بالا تعداد زیادی از پورت های هر سه سوییچ بلا استفاده مانده اند. اگر بتوانیم پورت های سخت افزاری بلااستفاده در سوییچ ها رو برای اتصال زیر شبکه های مختلف استفاده کنیم؛ تا حد زیادی در هزینه های برپایی شبکه صرفه جویی کرده ایم. برای رفع این مشکل، مفهومی بنام VLAN بوجود اومده. بدین طریق که هرکدام از پورت های سوییچ، برای کار با یک زیرشبکه خاص پیکربندی می شوند (توسط تنظیم کننده سوییچ یا ادمین شبکه). شکل زیر رو ببینید:



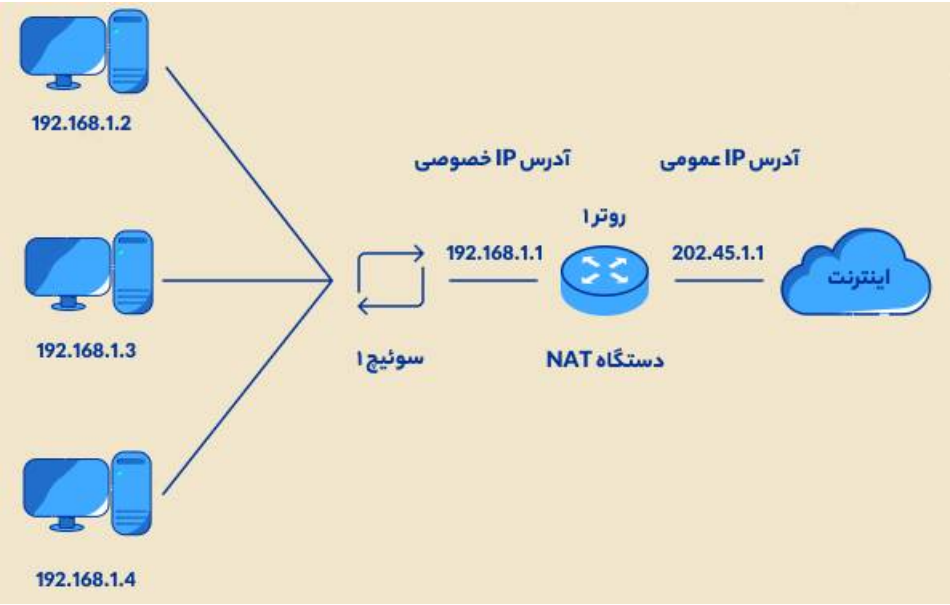
همونطور که دیده میشه با استفاده از مفهوم VLAN میتوان تعداد سوییچ های مورد نیاز رو کاهش داد. تنظیم کننده سوییچ (ادمین یا نصاب شبکه)، تعدادی از پورت ها رو برای کار با یک زیرشبکه خاص(با آدرس خاص) تنظیم میکنه (البته سوییچ باید این قابلیت رو داشته باشه). به پورت هایی که هاست ها رو به سوییچ وصل میکنند اصطلاحا Access Port و به پورت هایی که به تجهیزات دیگه مثل روتر ها متصل هستند؛ اصطلاحا Trunk Port گفته میشه. پورت هایی که به زیرشبکه خاصی اختصاص داده نشوند به VLAN پیش فرض اختصاص خواهند یافت. در نتیجه در اینجا یک سوییچ به صورت سه سوییچ مجزا عمل میکنه و اگر در یک زیرشبکه پیغامی برای زیرشبکه دیگری ارسال بشه؛ سوییچ ، پیغام رو به پورت تعریف شده بعنوان gateway اون زیرشبکه تحویل میده. به عنوان مثال،اگر در شکل بالا، هاست A در زیرشبکه 10 قصد ارسال پیام به B درزیرشبکه 30 رو داشته باشه؛ ابتدا پیغام به سوییچ تحویل میشه، از اونجاییکه مقصد این پیام زیرشبکه دیگری ست؛ این پیام به روتر

سمت چپ تحویل میشه، روتر مجددا به سوییچ برمیگردونه (اینبار به زیرشبکه 20 به رنگ نارنجی) ؛ از اونجاییکه این زیرشبکه نیز ، مقصد نهایی داده نیست؛ لذا سوییچ در زیرشبکه 20 پیام رو به روتر سمت راست تحویل میده؛ برای دومین بار، پیام به سوییچ برمیگرده و در نهایت تحویل هاست B میشه. شماره زیرشبکه با مفهومی بنام تگ (tag) شناخته میشه.

اگر از بخش های اولیه این متن و معرفی ساختار فریم Ethernet ii چیزی در خاطرتون مونده باشه؛ اونجا گفتیم که یک فریم به فرمت Ethernet ii یا IEEE802.3 حداقل 64 و حداکثر دارای 1518 بایت هست و همونجا اشاره کردیم استانداردهایی وجود دارند که در آن ها؛ بعد از 12 بایت شامل مک آدرس ها و قبل از دو بایت Ether Type؛ چهار بایت اضافه خواهیم داشت لذا حداقل پیام در این حالت 68 بایت و حداکثر 1522 بایت خواهد بود. استانداردی که این حالت براش تعریف شده IEEE 802.1Q نام دارد و در VALN ها استفاده میشه. دو بایت ابتدایی در تگ ها بنام TPID (Tag Protocol ID) همواره دارای عدد 0x8100 بوده و دو بایت بعدی بنام TCI (Tag Control Information) تگ پیام هست که نشون میده پیام فعلی به کدام VLAN تعلق داره. بخش TCI خودش از سه زیربخش PCP(3bit),DEI(1bit),VID(12bit) تشکیل شده. بخش VID شماره تگ رو مشخص میکنه. برای اطلاعات بیشتر به نت مراجعه کنید.

ضمیمه 3 : معرفی مفهوم NAT

Network Addres Translation یا "ترجمه آدرس شبکه" روشی است برای ارتباط یک زیرشبکه با شبکه جهانی اینترنت؛ یا تبدیل و تغییر (Mapping) یک آدرس به آدرس یک شبکه خارجی. احتمالا متوجه شده اید که در شبکه خانگی شما؛ تجهیزات دارای آیپی های کلاس C هستند مثلا 192.168.1.X ؛ خونه همسایه و دوست و آشنا و در دفتر کار شما نیز وضع همینه. اما مگه قرار نبود در تمام شبکه، آیپی ها یکتا (Uniqe) باشند! اینطوری که آیپی های مشترک زیاد داریم! پس چطور تجهیزات به اینترنت دسترسی دارند. این وظیفه پروتکلیه بنام NAT که توسط سخت افزار انجام میشه (مودم) و کاربر عموما با اون درگیر نمیشه. مودم شما به یک روتر در ISP شما متصل هست و پیغام هایی که از طرف تجهیزات شما (کامپیوتر یا گوشی های تلفن همراه که از طریق وای فای به مودم متصل هستند) به مودم ارسال میشه. با استفاده از NAT تغییر آدرس داده میشن به محدوده آدرسی که ISP در اختیار شما گذاشته (در NAT تعدادی آدرس در شبکه داخلی به یک آدرس در شبکه عمومی ترجمه یا لینک میشه. مودم در حافظه خودش، اطلاعاتی از پیام های ارسالی رو ذخیره میکنه که بدونه یک پیغام به یک IP خاص (در شبکه خارجی) از طرف کدام وسیله آمده و به شبکه اینترنت ارسال شده تا در برگشت؛ پاسخ ها رو به همون دستگاه تحویل بده. مودم ها برای این کار از اطلاعات IP header و شماره پورت (در لایه چهارم) استفاده می کنند. همونطور که می دونید در پروتکل IPV4 تعداد 4 بایت برای آدرس دهی موجود هست. خیلی زود (در اواسط دهه 90 میلادی) متوجه شده ن که این تعداد آدرس، احتمالا برای شبکه ای به وسعت اینترنت کم هست. لذا قبل از تعریف پروتکل IPV6 و به عنوان یک راه حل کوتاه مدت، مفهوم NAT بوجود اومد. اما از اونجایی که این روش، کارایی خودشو نشون داده؛ همچنان در حال استفاده ست و بنظر میرسه در آینده نزدیک نیز کنار گذاشته نشود. برای کسب اطلاعات بیشتر در مورد NAT به سند RFC 2663 مراجعه کنید.



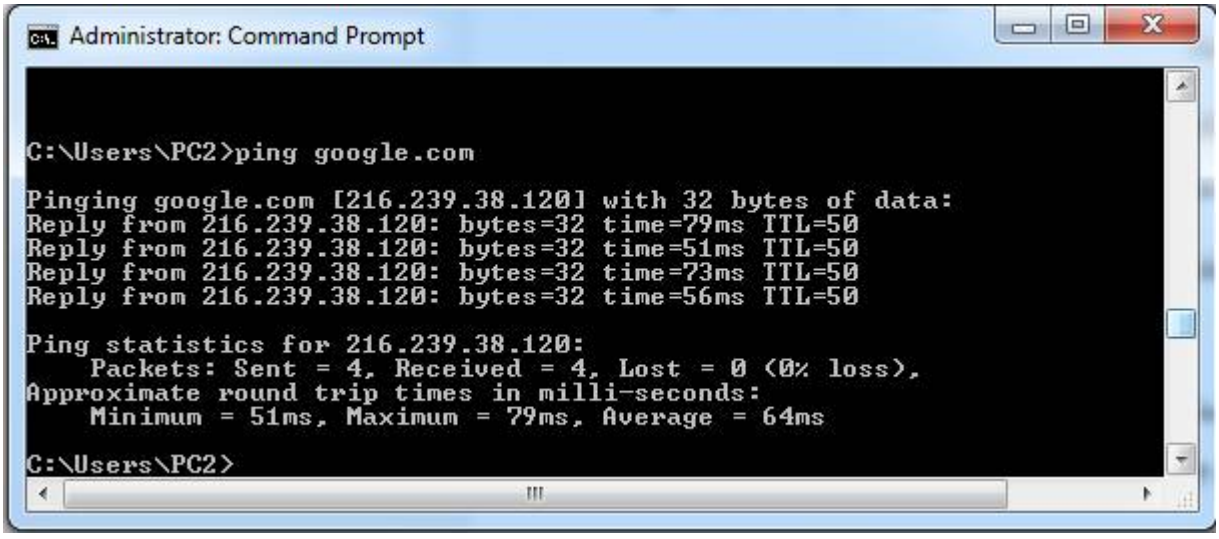
ضمیمه 4 : آشنایی با دستورات پرکاربرد در مدیریت شبکه

دستور ipconfig : کامند پرامپت را باز کنید (در منوی استارت کلمه cmd.exe روجستجو کنید یا از بخش accessories روی command prompt کلیک کنید).

در خط فرمان تایپ کنید ipconfig و اینتر را بزنید. اطلاعات IP کارت های شبکه متصل به سیستم نمایش داده خواهد شد. علاوه بر این میتوانید با استفاده از این دستور تنظیماتی را نیز انجام دهید.

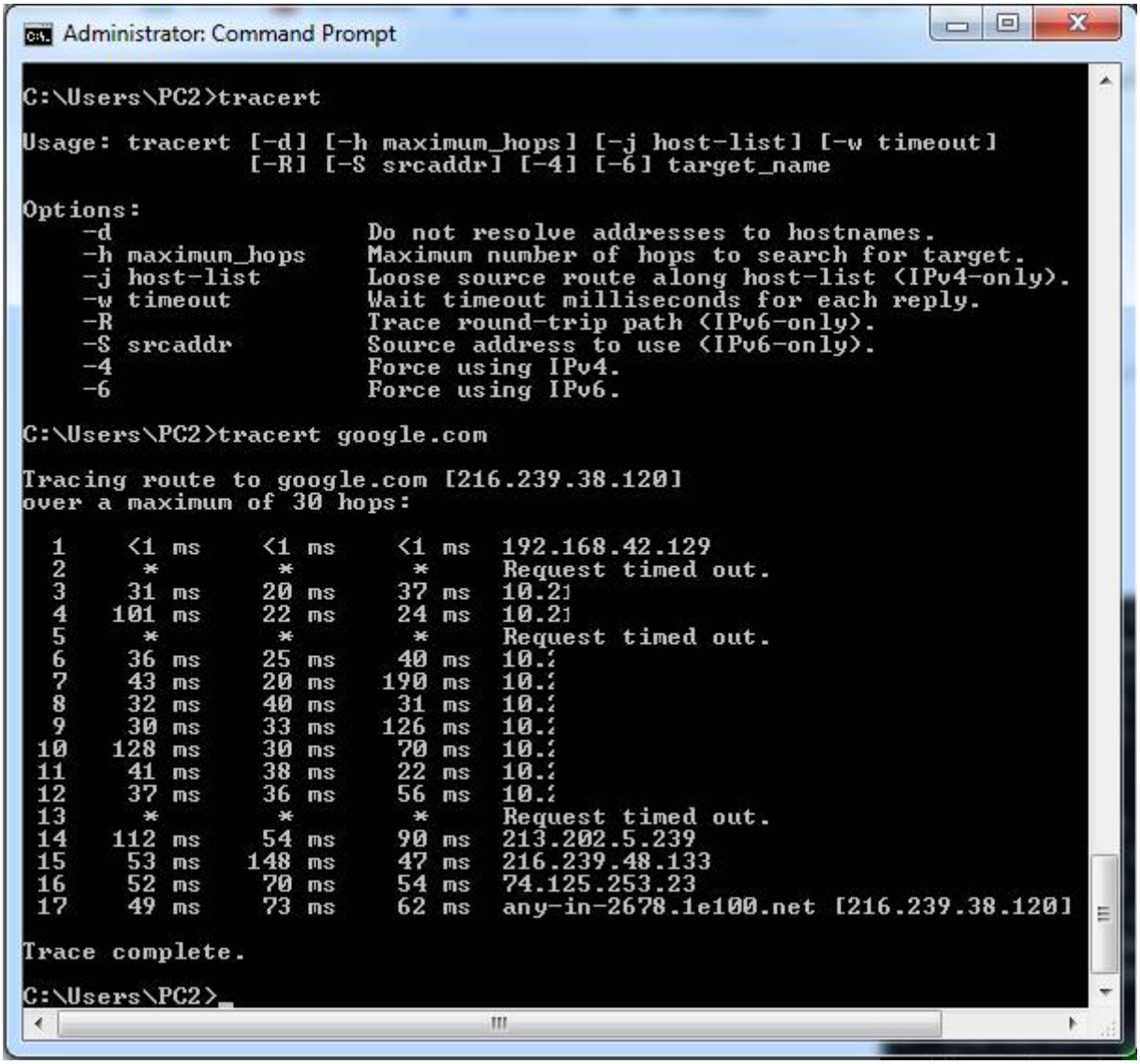
این دستور آرگومان هایی را هم می پذیرد؛ با استفاده از دستور ipconfig /all اطلاعات آرگومان های این دستور بطور کامل در اختیار کاربر قرار خواهد گرفت.

دستور ping : پینگ عملیاتی ست مبتنی بر پروتکل ICMP جهت بررسی در دسترس بودن یک هاست. دستور ping google.com رو ببینیم چه اطلاعاتی بما میده:



توجه داشته باشید در این حالت؛ ابتدا با استفاده از سرویس DNS آدرس IP گوگل بدست آمده (مگر اینکه از قبل این اطلاعات در حافظه سیستم باشد) و سپس عملیات پینگ انجام میشود.

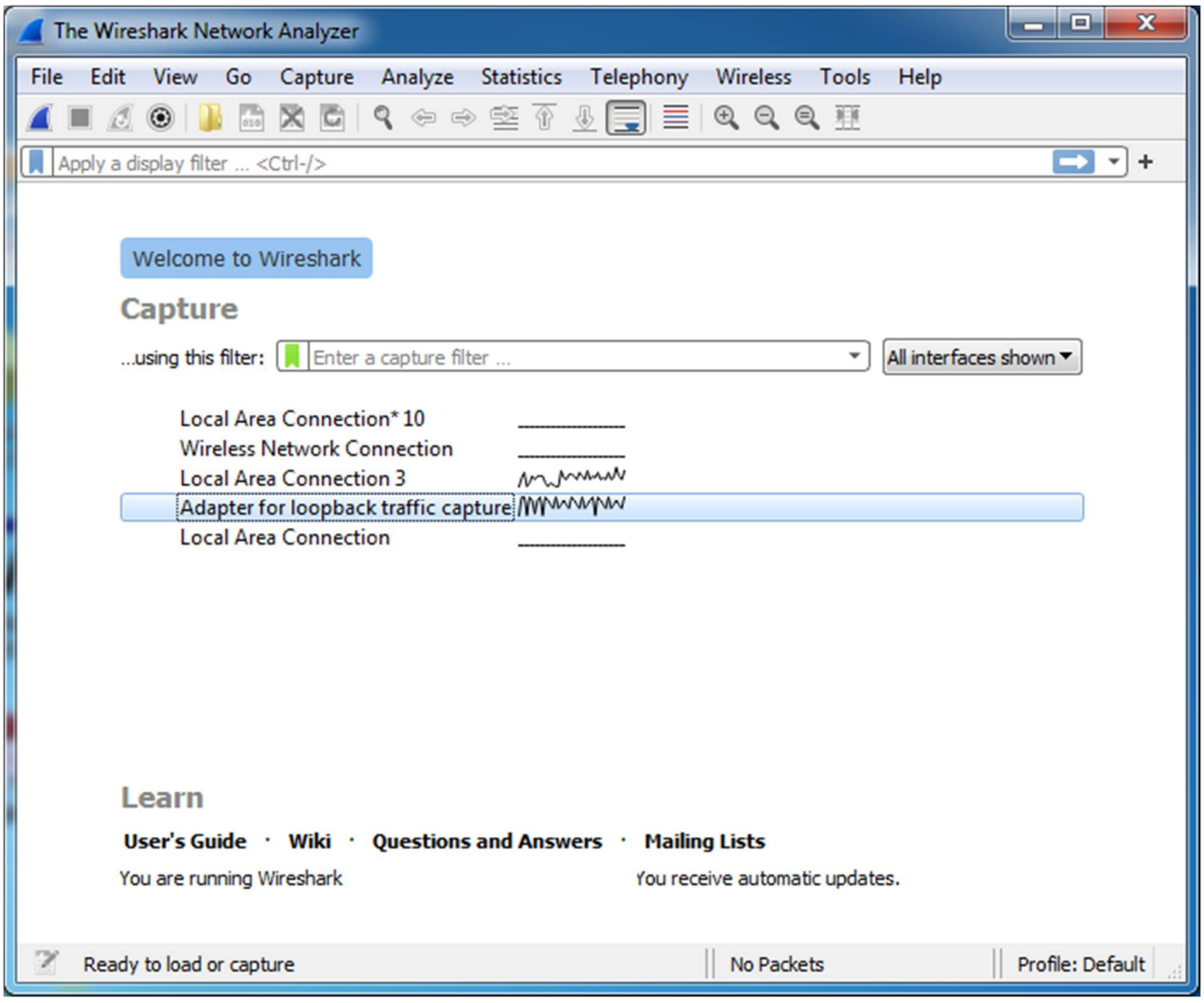
عملیات traceroute : عملیات traceroute با استفاده از پروتکل ICMP اطلاعاتی از مسیر دسترسی به هاست مورد نظر را بدست می دهد. در ویندوز این عملیات با دستور tracert در خط فرمان (command prompt) انجام میشه.



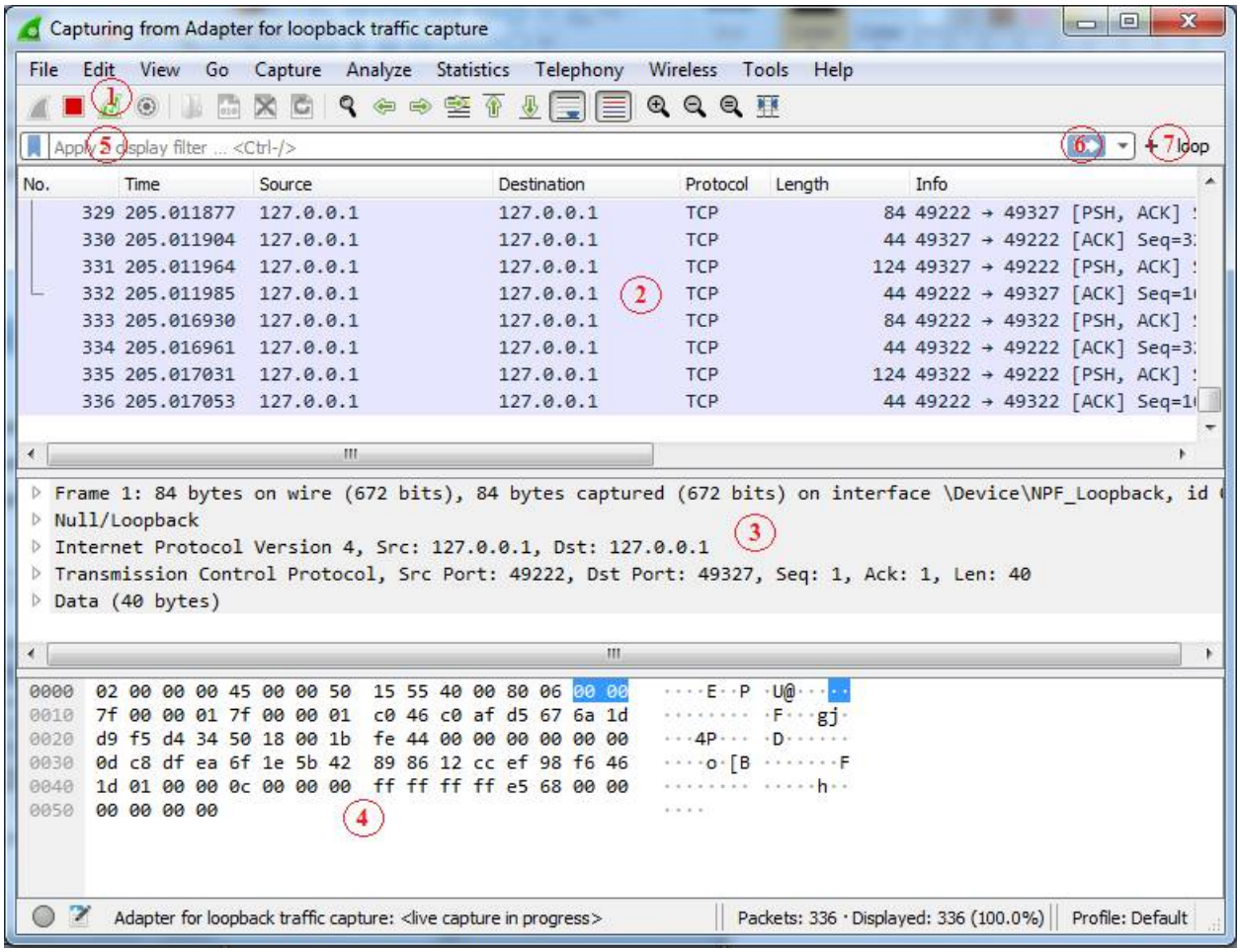
در قسمت بالایی تصویر، دستور tracert بدون هیچ آرگومانی و در دستور پایینی tracert google.com اجرا شده است. در گزارش این عملیات، روترهای واسط بین دستگاه شما و سرور اصلی نمایش داده می شود. توجه داشته باشید که این مسیر عملا بدلیل استفاده از default gateway ها جز در موارد خاص؛ تقریبن همواره ثابت است.

ضمیمه 5 : معرفی نرم افزار wireshark

در ارتباط با مبحث شبکه، یکی از بهترین نرم افزارهای شنود (sniffer) و بررسی پکت های ارسالی یا دریافتی، نرم افزار wireshark هست. این نرم افزار طبیعتا قابلیت های بسیار زیادی داره؛ اما برای شروع، ما سعی میکنیم یه توضیح ابتدایی و ساده بدیم. بعد از اجرای نرم افزار در قسمت Capture لیست تجهیزات و ارتباطات مبتنی بر شبکه رو می بینید؛ رابط مورد نظر رو انتخاب و روش دابل کلیک کنید.



در پنجره اصلی برنامه؛ اطلاعات روی خط، نمایش داده میشه. طبق معمول موارد زیادی از طریق منوها و ابزار این نرم افزار در اختیار کاربر قرار داره که بعضی رو در ادامه معرفی کردیم:



1- دکمه هایی برای استارت/استپ عملکرد شنود

2- نمایش تمامی پکت های دریافتی؛ اگر در بخش 5 فیلتری اعمال شده باشد؛ فقط پکت های مطلوب، نمایش داده می شود؛ اما هنگام ذخیره (Save) میتوان تمامی پکت ها را ذخیره کرد.

3- با کلیک بر روی یک پکت در بخش 2 ، مشخصات آن در این قسمت نمایش داده می شود. پروتکل های لایه های مختلف، طبقه بندی می شوند و همانطور که در تصویر مشخص است؛ ابتدا کل frame سپس لایه IP بعد TCP به همراه داده ارسالی در اینجا نمایش داده شده است.

4- با انتخاب یک لایه در بخش 3 قسمت های مختلف لایه انتخاب شده، در اینجا نمایش داده می شود.

5- از مهمترین قابلیت های نرم افزار wireshark ؛ توانایی ایجاد فیلترهای دلخواه است. این فیلترها با سبک نوشتاری(syntax) شبیه به زبان های برنامه نویسی پیاده سازی می شوند؛ به عنوان مثال فیلتر :

ARP or TCP.Port==1010

تنها "پکت های پروتکل ARP یا پکت های TCP که پورت آنها (مبدا یا مقصد) 1010 است"، را نمایش خواهد داد. بعد از نوشتن یا انتخاب یک فیلتر از پیش نوشته شده؛ برای اعمال فیلتر، دکمه 6 را فشار دهید.

7-جهت تعریف و ذخیره یک فیلتر، از این قسمت استفاده کنید.

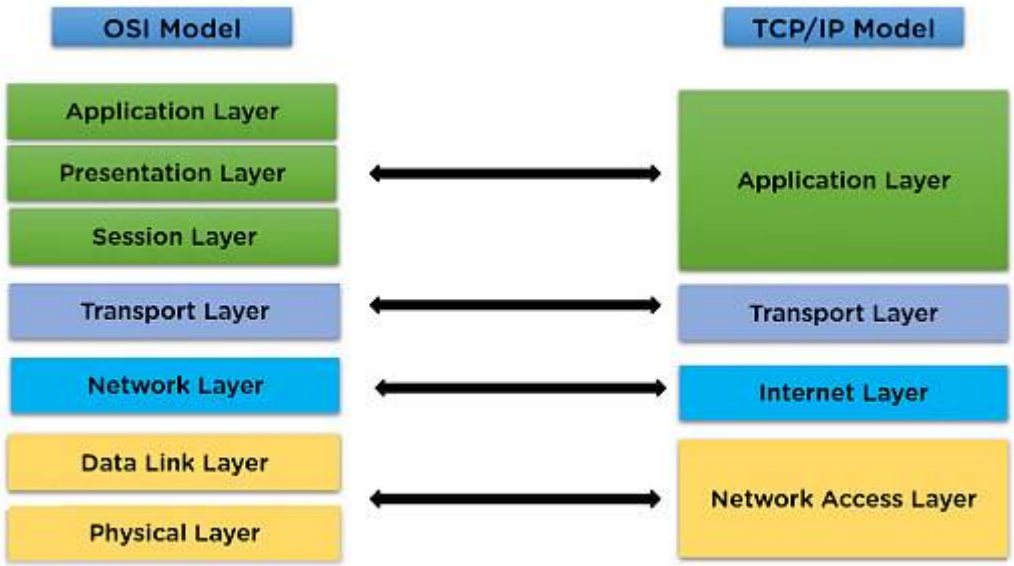
- فریم های دریافتی را میتوان با استفاده از گزینه File/Save ذخیره نمود.
- برای تمایز بهتر اطلاعات، میتوان رنگ بندی نمایش رو هم، سفارشی سازی کنید.

ضمیمه 6 : اسناد RFC

اسناد RFC برای بیان استانداردها ؛ الگوریتم ها یا اصلاح مفاهیم و پروتکل های فعلی در شبکه جهانی اینترنت بوجود اومده. با اینکه متولی این کار سازمان IETF هست؛ اما این اسناد توسط اشخاص، گروه ها و سازمان های مختلفی نوشته شده (مثلا توسط افرادی که برای اولین بار یک پروتکل رو معرفی کردند). این اسناد در سه نوع STD(STanDard), FYI(For Your Information), BCP(Best Current Practice) معرفی میشن. اسناد STD، استانداردها رو در خودشون دارند مثل RFC 793 برای معرفی اولین استاندارد TCP؛ اسناد FYI برای اطلاع بیشتر در مورد یک موضوع خاص و BCP ها دربرگیرنده نتایج تست انواع الگوریتم ها و بهترین وضعیت اونهاست(بعنوان مثال الگوریتم های محاسبه چک سام). این اسناد شماره گذاری میشن و هر بار که در یک موضوع خاص، یک RFC جدید ثبت میشه؛ اسناد قبلی وابسته به اون، ویرایش میشن تا خواننده بتونه دنباله اسناد مرتبط رو داشته باشه. برای دانلود و مطالعه این اسناد میتوانید از دو سایت rfc-editor.org و datatracker.ietf.org استفاده کنید.

ضمیمه 7 : مدل های OSI و TCP/IP

در مسیر پیشرفت علوم مهندسی، ابتدا یک نیاز مطرح شده. سپس برای رفع اون نیاز؛ روش ها یا مفاهیم جدید محقق شده ن و در ادامه احتمالا سعی بر استاندارد سازی این فرآیندها صورت گرفته است. مبحث شبکه نیز از این قاعده پیروی میکنه. در نتیجه ابتدا نیاز به شبکه سازی بوجود اومد. در ادامه روش های مختلفی برای ایجاد یک شبکه طراحی و ساخته شد و در نهایت این روش ها استاندارد شده ن. مدل های OSI و TCP/IP مهمترین مدلهای طراحی و لایه بندی یک شبکه هستند؛ با این هدف که در یک ارتباط، وظیفه هر بخش مشخص باشه. با این رویکرد؛ هم طراحی مفاهیم جدید مبتنی بر شبکه و هم تعمیر/نگهداری شبکه ساده تر میشه. دو مدل، بسیار شبیه بهم طراحی شده ن و تفاوت اصلیشون اینه که در لایه های بالاتر از لایه چهارم ، در مدل TCP/IP تنها یک لایه تحت عنوان لایه برنامه (Application) تعریف شده؛ اما در مدل OSI ، این بخش، خودش از سه لایه، تشکیل شده و سعی شده هر لایه تعریف و وظایف مشخصی داشته باشه. هرچند می بینیم که گاها یک عملیات در دو یا چند لایه داره کار میکنه و لایه ها با هم همپوشانی دارند. ما قصد نداریم مطالبی که عموما در کتاب های درسی در مورد این دو مدل گفته میشه رو اینجا تکرار کنیم. در طول این نوشتار تقریبا با همه لایه ها آشنا شدیم. اینجا یکبار دیگه و بطور خلاصه هر لایه رو معرفی میکنیم. در شکل زیر لایه بندی و تفاوت دو مدل رو می بینید.



- در بعضی مراجع؛ لایه اول در مدل TCP/IP رو؛ همانند مدل OSI در دو لایه Physical وLink نشان داده اند و در نتیجه مدل TCP/IP رو به صورت 5 لایه تصویر کرده اند. همچنین بگیم که گاهی به آن Internet Protocol Suite یا TCP/ip Stack هم گفته میشه.

لایه 1: لایه یک با نام لایه فیزیکی در هر ارتباطی وجود داره. شما باید بدونید که سیگنال ها (دیجیتال یا آنالوگ) به چه طریقی به مقصد میرسند؛ ارتباط با سیم یا بدون سیم هست؛ مدولاسیون از چه نوعی هست؛ سیگنال ها تکی یا دیفرانسیلی هستند؛ باودریت چقدره؛ آیا از فیبر نوری استفاده میشه ، شایدم از دود استفاده بشه!

نکته: در یک ارتباط، حتا در غیر از حالت شبکه ؛ مثل وقتی که فقط دو سخت افزار پیام هایی بین خودشون رد و بدل میکنند، یک سنسور و یک میکروکنترلر مثلا؛ یا ارتباط سریال بین میکروکنترلر و کامپیوتر؛ بخش سخت افزاری ارتباط در لایه یک گنجونده میشه و بخش نرم افزاری ارتباط هم تنها در یک لایه با عنوان لایه منطقی ایجاد میشه (مثلا در ارتباط RS232 و مفاهیمی مثل تعداد بیت ها، آغاز و پایان یک ارتباط و ....)

لایه 2 : الان فرض بر اینه که در لایه قبل؛ یک سیگنال دیجیتال رو به سمت مقابل فرستادیم. لایه دوم، اولین لایه ایه که مفاهیم نرم افزاری و منطقی وارد میشن؛ مثلا ابتدا و انتهای یک استریم (مجموعه ای از بیت ها) کجاست؟ چند بیت فرستاده میشه؟ در شبکه این بیت ها رو کدام گره فرستاده و کدام گره باید از اون ها استفاده کنه؟ لذا اینجا ما نیاز به یک فرمت آدرس دهی داریم! این آدرس رو عموما آدرس سخت افزاری میگن و واضحه که هر آدرس برای ارسال در این لایه؛ باید یکتا باشه! پروتکل های Ethernet ii و IEEE802.3 از اصلیترین پروتکل های این لایه هستند که در این نوشتار توضیح داده شده ن.

لایه 3 : وجود بعضی الزامات همچون موارد امنیتی یا نیاز به جداسازی زیرشبکه ها از یکدیگر(مثلا برای مسیریابی)؛ باعث شده به یک فرمت آدرس دهی دیگه؛ احساس نیاز بشه که قابلیت اعمال تغییر از سوی ادمین رو هم داشته باشه. به این آدرس؛ آدرس منطقی ؛ اینترنتی و یا در قیاس با آدرس سخت افزاری؛ آدرس نرم افزاری گفته میشه. این نحوه آدرس دهی در این لایه پیاده سازی میشه. همچنین گزارشات مربوط به خطا عموما در این لایه اجرا میشه. عملیات دیگه ای که در این لایه انجام میشه، تکه تکه کردن داده های بزرگ برای ارسال هست. مهمترین پروتکل های این لایه ARP , IPv4, IPv6 هستند که در این نوشته؛ دوتای اول رو توصیف کردیم. در بعضی مراجع ARP رو جزو لایه دوم به حساب می آورند.

لایه 4 : در لایه یک تا سه؛ ارتباط هاست (host) به هاست برقرار شده؛ اما برای ارتباط سرویس با سرویس، مجددا نیاز به یک آدرس دهی داخلی (درون یک هاست) داریم. این نوع آدرس رو اصطلاحا آدرس یا شماره پورت میگن و در این لایه قرار داره. فرض کنید در یک کامپیوتر، همزمان چند برنامه از شبکه استفاده میک نند. طبیعیه که آدرس فیزیکی و منطقی همه این ها یکسانه. لذا از شماره پورت ها، جهت تعیین مقصد هر بسته درون یک هاست استفاده میشه. از وظایف دیگه این طبقه مدیریت ترافیک و جلوگیری از موارد ازدحام (Congestion) هست. همچنین درخواست ارسال دوباره داده ها در این لایه انجام میشه. سه پروتکل UDP, TCP, ICMP اصلیترین پروتکل های این لایه هستند که یک به یک توضیح داده شده ن. پروتکل های UDP,TCP جهت

ارسال/دریافت داده و پروتکل ICMP جهت عیب یابی،گزارش خطا و بررسی سلامت رفتار شبکه استفاده میشن. در بعضی مراجع ICMP رو در لایه سوم بررسی میکنند!

لایه های 5 تا 7 : که گاهی به اون ها لایه +5 هم گفته میشه؛ در ارتباط مستقیم با داده های نهایی هستند که توسط کاربر یا نرم افزارها و سرویس ها استفاده میشن. حتا در تعاریف کتابی نیز؛ قسمت زیادی از وظایفی که برای این لایه ها تعریف شده ن، با هم همپوشانی دارند. انواع پروتکل ها برای مواردی چون ارسال ایمیل (مثل SMTP)، ارسال فایل ها (مثل FTP) و یا نمایش صفحات وب (مثل HTTP یا HTTPS) در این لایه ها انجام میگیرد.

ضمیمه 8 : Big Endian vs Little Endian

هنگامی که ما در حال نوشتن اعداد بر روی کاغذ هستیم؛ بطور معمول نوشتن اعداد را از سمت چپ با نوشتن پرارزشترین رقم شروع می کنیم. بعنوان مثال عدد هزار و سیصد و چهل و هفت بصورت 1347 نوشته می شود. این یک قرارداد است. براحتی میشد این قرار داد طور دیگری باشد؛ مثلا ما از سمت راست اعداد را بنویسیم که همان عدد بصورت 7431 نوشته می شد. از طرفی هنگام نوشتن اعداد با ارقام زیاد، تنها محدودیت ما عرض کاغذ زیر دست ماست و عملا هر عددی، با هر تعداد رقم را، میتوانیم بنویسیم. ولی در محاسبات کامپیوتری؛ اولا باید تعداد بایت هاییکه یک عدد اشغال میکند (و در نتیجه گستره اعداد)، از قبل مشخص باشد؛ ثانیا باید بدانیم (کامپایلر بداند) که تفسیر صفرها و یک های درون این بایت ها، چگونه باید باشد. آیا این یک عدد اعشاری است یا صحیح؛ علامت دار است یا بدون علامت؛ یا اینکه تنها نمایشگر یک کاراکتر اسکی ست.

اعداد برای ذخیره شدن در حافظه؛ نیاز به اشغال یک یا چند بایت دارند؛ معمول اینست که تعداد1؛ 2 ؛ 4 و یا 8 بایت، جهت این کار اختصاص می یابد (برای اعدادی که تعداد بایت بیشتری نیاز دارند؛ مثلا موجودی حساب یک نفر در بانک؛ از روش های دیگری مثل کلاس ها استفاده می شود). تعداد و تفسیر بایت ها در بسیاری از زبان های برنامه نویسی؛ توسط برنامه نویس با اعلان هایی مثل float یا unsigned int مشخص می شود (مابقی کامپایلرها مثل basic خود تشخیص میدهند چه تعداد بایت اختصاص دهند؛ ولی نتیجه کار فرقی نمیکند)

هنگام ذخیره سازی این بایت ها در حافظه RAM ؛ باید از قبل مشخص باشد که کدام بایت ها در کدام خانه های حافظه قرار میگیرند و اشاره گر (Pointer) به داده مربوطه، به کدام بایت اشاره دارد و چه آدرسی را برمی گرداند.

این عملیات در کامپیوترهای مبتنی بر معماری 8086 و سیستم عامل ویندوز به این ترتیب انجام می شود که بایت کم ارزش تر، در خانه حافظه با آدرس کوچکتر قرار می گیرد. به این حالت Little Endian می گویند. مثال زیر طریقه ثبت دو عدد بدون علامت 16 بیتی با محتوای 0xAA55 و 0x2344 را در حالت Little Endian را نمایش می دهد. اشاره گر به داده اول مقدار(آدرس) 1001 و اشاره گر به داده دوم هم مقدار 997 را برمیگرداند. در حالت Big Endian، بایت پر ارزش در خانه با آدرس کوچکتر ذخیره می شود ولی همچنان اشاره گرها اولین آدرس اشغال شده (آدرس کوچکتر را بر می گردانند) در واقع تفاوت این دو حالت، در جابجایی محل ذخیره بایت های پرارزش و کم ارزش است. در داده های 4 یا 8 بایتی نیز همین وضعیت برقرار است؛ مثل اینکه بایت ها بصورت آینه ای قرینه شده اند و بایت پرارزش در آدرس کوچکتر قرار میگیرد و توسط اشاره گر، اولین آدرس (کمترین آدرس اشغال شده) بر می گردد . در سیستم عامل ویندوز و البته عموم کامپایلرهای میکروکنترلرها؛ فرمت ذخیره Little Endian استفاده می شود؛ اما در تعریف پروتکل های اینترنتی، بنا به دلایلی از فرمت Big Endian استفاده شده است؛ بدین معنی که اگر بایت های دریافتی یا ارسالی را در یک آرایه از

بایت ها ذخیره کنیم، بایت های پرارزش تر در خانه های با آدرس کوچکتر آرایه قرار می گیرند. این نحوه قرارگیری هنگام پردازش اطلاعات باید مورد توجه واقع شود وگرنه خطاهای زیادی خواهیم داشت.

آدرس	مقدار	
1004	؟	متغیرهای دیگر
1003	؟	متغیرهای دیگر
1102	0xAA	متغیر از نوع Uint16_t
1001	0x55	
1000	؟	متغیرهای دیگر
999	؟	متغیرهای دیگر
998	0x23	متغیر از نوع Uint16_t
997	0x44	
996	؟	متغیرهای دیگر
995	؟	متغیرهای دیگر
994	؟	متغیرهای دیگر

جدول : فرمت ذخیره Little Endian

ضمیمه 9 : انواع کابلینگ در اترنت

در بخش فیزیکال یک ارتباط اترنت، از کاکتورهای RJ45 استفاده می شود. RJ مخفف Registred Jack است. به این کانکتور اصطلاحا 8P8C یا 8Position,8Connection هم گفته می شود؛ بدین معنی که در این کانکتور 8 پین وجود داره و از هر 8 تای آن می توان استفاده کرد. برای سیم بندی، از زوج سیم های به هم تابیده (Twisted Pair) استفاده شده. همونطور که میدونیم؛ سیگنال ها بصورت دیفرانسیلی ارسال میشن. از خصوصیات سیم کشی تابیده؛ خنثا کردن نویز مشترک روی هر دو سیم هست. رنگ بندی سیم ها نیز در دو استاندارد TIA568A, TIA568B تعریف شده ن. این دو استاندارد در شکل زیر نشان داده شده است.

Pin	T568A pair	T568A color	T568B pair	T568B color	10BASE-T/100BASE-TX signal	1000BASE-T/10GBASE-T signal	Wire	Diagram
1	3	white/green stripe	2	white/orange stripe	TD+	DA+	tip	
2	3	green solid	2	orange solid	TD−	DA−	ring	
3	2	white/orange stripe	3	white/green stripe	RD+	DB+	tip	
4	1	blue solid	1	blue solid	not used	DC+	ring	
5	1	white/blue stripe	1	white/blue stripe	not used	DC−	tip	
6	2	orange solid	3	green solid	RD−	DB−	ring	
7	4	white/brown stripe	4	white/brown stripe	not used	DD+	tip	
8	4	brown solid	4	brown solid	not used	DD−	ring	

Pin numbering on plug face. Connected pins on plug and jack have the same number.

همچنین کابل های اترنت را؛ از جهت نوع سیم بندی در دو سمت؛ میتوان به دو دسته تقسیم کرد. کابل مستقیم (Straghit یا یک به یک) و کابل متقاطع یا ضربدری (Cross) تفاوت این دو نوع در اینه که در کابل مستقیم، هر پین کانکتور در یک سمت، به پین مشابه در کانکتور سمت دیگر رفته است. در حالی که در کابل کراس،

سیگنال های TX در یک سمت، به سیگنال های RX در سمت دیگر متصل شده اند و بالعکس. قبلا گفته ایم که با توجه به دو قابلیت Auto Polarity و Auto MDIX انتخاب هر دو کابل میسر است.

خود کابل ها نیز با توجه به نوع پوشش و سطح محافظت از آنها؛ به 7 بخش یا نوع (Category) تقسیم شده و به کابل های CAT معروف هستند. کابل هایی که پوشش محافظتی ندارند، به نام UTP(Unshielded Twisted Pair) ؛ کابل های که پوشش محافظتی دارند به نام STP(Shielded Twisted Pair) و کابل هایی که دارای فویل محافظتی هستند؛ بنام FTP(Foil Twisted Pair) شناخته می شوند. در CAT های بالاتر، علاوه بر اینکه هر زوج سیم، روکش محافظ مخصوص دارد، تمام کابل نیز دارای ورقه فلزی محافظت از نویز باشد. نوع لایه محافظ نیز در کیفیت یک ارتباط موثر خواهد بود، لذا ممکن است در یک کابل از یک لایه ورقه آلومینیومی برای محافظت استفاده شده باشد و در دیگری از یک توری فلزی پیچیده شده. هر چه سرعت ارتباط بالاتر باشد یا ارتباط در فاصله بیشتری استفاده گردد؛ ناچار به استفاده از کابل های با CAT بالاتر خواهیم بود. در ارتباط 10Mbps ؛ بایستی حداقل کابل نوع CAT3 انتخاب گردد؛ در حالی که مثلا در ارتباط با یک دوربین صنعتی، نوع کابل باید حداقل از نوع CAT5e (Enhanced CAT5) یا CAT6 انتخاب گردد. همچنین سعی کنید از کابل های کمتر از یک متر استفاده نکنید و در مسافت های زیاد، از کابل های با کیفیت تر استفاده کنید. در شکل زیر تفاوت بعضی از این کابل ها را مشاهده می کنید.

 UTP CAT5E LAN CABLE	 FTP CAT5E LAN CABLE	 SFTP CAT5E LAN CABLE	 UTP CAT5E OUTDOOR CABLE
 UTP CAT6 LAN CABLE	 FTP CAT6 LAN CABLE	 SFTP CAT6 LAN CABLE	 UTP CAT6 OUTDOOR CABLE
 SFTP CAT5E OUTDOOR CABLE	 SFTP CAT6 OUTDOOR CABLE	 U/FTP CAT6A LAN CABLE	 SFTP CAT7 LAN CABLE

ضمیمه 10 : کلاس های آدرس IP

همونطور که میدونید؛ آدرس IP یک آدرس منطقی به اندازه 4 بایت هست. با این تعداد بایت میشه تعداد 2 به توان 32 هاست رو آدرس دهی کرد؛ یعنی تقریبا 4.3 میلیارد هاست. این 4 بایت به دو بخش شماره ی شبکه و شماره ی هاست در شبکه تقسیم میشه و شامل کلاس های 5 گانه ای هست. بسته به اینکه پرارزشترین بیت های هر کلاس دارای چه مقداری باشند (و در نتیجه آدرس هر شبکه با چه عددی شروع بشه) کلاس ها از هم متمایز میشن. جدول زیر رو ببینید:

کلاس	بیت پرارزش اولین بایت	از	تا
A	0	0.0.0.0	126.255.255.255
B	10	128.0.0.0	191.255.255.255
C	110	192.0.0.0	223.255.255.255
D (Multicast)	1110	224.0.0.0	239.255.255.255
E (Exprimental)	1111	240.0.0.0	255.255.255.255

همونطور که میبینید بعضی از آدرس ها در جدول نیستند و رزور شده ن. بعنوان مثال آدرس 127.0.0.1 بعنوان آدرس local host شناخته میشه و آدرس همون کامپیوتر یا هاستیه که در حال ارسال پکت هست. از این آدرس برای ارتباط بین دو سرویس در یک هاست استفاده میشه؛ مثلا وقتی دو نرم افزار روی یک کامپیوتر با هم ارتباط تحت شبکه می گیرند. اصطلاحا به این آدرس، لوپ بک (loopback) هم گفته میشه. اینگونه پیام ها روی کارت شبکه نمیرن و مستقیمن از بافر ارسال به بافر دریافت منتقل میشن. فریم بندی این پیام ها رو در wireshark ببینید!

از طرفی؛ معمول اینه که اولین آدرس در شبکه یعنی آدرس 0 رو به عنوان آدرس خود شبکه رزرو میکنند و برای آدرس دهی به یک هاست استفاده نمیشه و شما مثلا آدرس 192.168.1.0 رو در جایی به عنوان آدرس هاست نمی بینید. آدرس خاص دیگه در یک شبکه؛ آدرس تمام '1' در یک شبکه است. این آدرس هم به عنوان آدرس عمومی در اون شبکه درنظر گرفته میشه؛ به عنوان مثال؛ در شبکه بالا، آدرس 192.168.1.255 آدرس عمومی در شبکه ست. تمام هاست های موجود در یک شبکه؛ پیام های آدرس عمومی رو دریافت خواهند کرد. پس برای آدرس دهی مستقیم به یک هاست؛ از اعداد بین آدرس شبکه و ادرس عمومی استفاده میشه.

گفتیم که قسمت سمت چپ (بایت های پرارزش) آدرس شبکه رو مشخص می کنند و مابقی بایت ها، آدرس host در شبکه رو مشخص میکنند؛ پس در آدرس دهی کلاس A ما میتوانیم بیش از 16 میلیون هاست رو در یک شبکه آدرس دهی بکنیم. در کلاس B این تعداد به 65535 هاست کاهش پیدا می کنه و در نهایت در کلاس

C ما حداکثر 255 هاست در شبکه خواهیم داشت. بسته به اینکه در شبکه مون چه تعداد هاست نیاز داریم؛ باید از یکی از این کلاس ها استفاده کنیم و اگر تعدادی از آدرس ها بدون استفاده باشن؛ میشه اون محدوده رو در اختیار دیگران گذاشت. با استفاده از مفهوم زیرشبکه؛ یک شبکه به چند بخش تقسیم میشه که هر بخش تعداد کافی آدرس برای شبکه مورد نیاز رو داشته باشه. اینکه کدام بخش از آدرس، به عنوان آدرس شبکه و کدام بخش به عنوان آدرس هاست استفاده میشه؛ با مفهومی بنام subnet mask مشخص میشه. Subnet mask هم عددی 4 بایتی ست و معمول اینه که بیشتر از اعداد 255 یا 0 در اون استفاده میشه؛ پس subnet mask برای شبکه های کلاس A عدد 255.0.0.0 خواهد بود؛ برای کلاس B عدد 255.255.0.0 و برای کلاس C عدد 255.255.255.0

- بعضی مواقع subnet mask رو همراه با آدرس آیپی مشخص می کنند؛ به عنوان مثال آدرس هاست به صورت 192.168.1.10/24 نوشته میشه؛ بدین معنی که آدرس هاست ما 192.168.1.10 هست و 24 بیت ابتدایی از این آدرس؛ آدرس شبکه هست.

در صورتی که بخواهیم یک شبکه رو به چند زیرشبکه تقسیم کنیم (برای بهبود سرعت مسیریابی؛ ایجاد امنیت و یا جداسازی دو زیربخش مثلا زیرشبکه بخش مالی و زیرشبکه بخش فنی در یک شرکت) میشه تعداد بیشتری بیت رو به زیرشبکه اختصاص بدیم. فرض کنید در شبکه ای با آدرس 192.168.1.X تعداد تقریبی 250 هاست رو میخوایم به دو گروه مساوی در دو زیرشبکه تقسیم کنیم. در این وضعیت بیت بیست و پنجم از آدرس رو هم به subnet mask اختصاص می دهیم؛ لذا فرم آدرسمون میشه 192.168.1.x/25. آدرس زیرشبکه اول رو 192.168.1.0/25 و دومی رو 192.168.1.128/25 معرفی می کنیم. مجددا در هر زیرشبکه؛ آدرس اول؛ آدرس شبکه است و آدرس آخر؛ آدرس عمومی در زیرشبکه. لذا در مثال بالا آدرس 192.168.1.0 آدرس زیرشبکه اول و آدرس 192.168.1.127 آدرس عمومی این زیرشبکه خواهد بود. در زیر شبکه دوم؛ آدرس شبکه 192.168.1.128 و آدرس عمومی آن 192.168.1.255. نوشتن بایت آخر در این آدرس ها بصورت بیتی؛ در فهم بهتر موضوع کمک کننده ست.

0 ="00000000"

127="01111111"

128="10000000"

255="11111111"

محتویات فایل ENC28J60.h

```
#ifndef ENC28J60_H
#define ENC28J60_H
#include "stm32f1xx_hal.h"
#define ENC28J60_CS_PORT        GPIOA
#define ENC28J60_CS_PIN    GPIO_PIN_4
#define ENC28J60_RESET_PORT      GPIOB
#define ENC28J60_RESET_PIN GPIO_PIN_0
#define ENC28J60_SPI_TIMEOUT     10
#define ENC28J60_OP_CODE_OFFSET   5
#define ENC28J60_REG_BANK_OFFSET  5
#define ENC28J60_REG_TYPE_OFFSET  7
#define ENC28J60_TX_BUF_START     0x0000
#define ENC28J60_RX_BUF_START     0x0600
#define ENC28J60_RX_BUF_END       0x1FFF
#define ENC28J60_FRAME_RX_OK_MASK 0x80
#define ENC28J60_REG_BANK_MASK    0x60
#define ENC28J60_REG_TYPE_MASK    0x80
#define ENC28J60_REG_ADDR_MASK    0x1F
#define ENC28J60_BUF_COMMAND_ARG  0x1A
#define ENC28J60_RESET_COMMAND_ARG        0x1F
#define ENC28J60_FRAME_DATA_MAX   1024
#define ENC28J60_BB_PACKET_GAP    0x15
#define ENC28J60_NBB_PACKET_GAP   0x0C12
#define ENC28J60_BANK_0_BITS      (BANK_0 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_BANK_1_BITS      (BANK_1 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_BANK_2_BITS      (BANK_2 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_BANK_3_BITS      (BANK_3 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_BANK_COMMON_BITS (BANK_0 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_COMMON_REGS_ADDR 0x1B
#define ENC28J60_ETH_REG_BIT      (ETH_REG << ENC28J60_REG_TYPE_OFFSET)
#define ENC28J60_MAC_MII_REG_BIT  (MAC_MII_REG << ENC28J60_REG_TYPE_OFFSET)
// Common bank registers
#define EIE        (0x1B | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_COMMON_BITS)
#define EIR        (0x1C | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_COMMON_BITS)
#define ESTAT      (0x1D | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_COMMON_BITS)
#define ECON2      (0x1E | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_COMMON_BITS)
#define ECON1      (0x1F | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_COMMON_BITS)
#define EIR_PKTIF_BIT     (1 << 6)
#define EIR_DMAIF_BIT     (1 << 5)
```

```
#define EIR_LINKIF_BIT    (1 << 4)
#define EIR_TXIF_BIT      (1 << 3)
#define EIR_TXERIF_BIT    (1 << 1)
#define EIR_RXERIF_BIT    (1 << 0)
#define ECON2_AUTOINC_BIT  (1 << 7)
#define ECON2_PKTDEC_BIT   (1 << 6)
#define ECON2_PWRSV_BIT    (1 << 5)
#define ECON2_VRPS_BIT     (1 << 3)
#define ECON1_TXRST_BIT    (1 << 7)
#define ECON1_RXRST_BIT    (1 << 6)
#define ECON1_DMAST_BIT    (1 << 5)
#define ECON1_CSUMEN_BIT   (1 << 4)
#define ECON1_TXRTS_BIT    (1 << 3)
#define ECON1_RXEN_BIT     (1 << 2)
#define ECON1_BSEL1_BIT    (1 << 1)
#define ECON1_BSEL0_BIT    (1 << 0)
// Bank 0 registers
#define ERDPTL      (0x00 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERDPTH      (0x01 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EWRPTL      (0x02 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EWRPTH      (0x03 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ETXSTL      (0x04 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ETXSTH      (0x05 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ETXNDL      (0x06 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ETXNDH      (0x07 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXSTL      (0x08 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXSTH      (0x09 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXNDL      (0x0A | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXNDH      (0x0B | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXRDPTL    (0x0C | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXRDPTH    (0x0D | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXWRPTL    (0x0E | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXWRPTH    (0x0F | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMASTL     (0x10 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMASTH     (0x11 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMANDL     (0x12 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMANDH     (0x13 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMADSTL    (0x14 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMADSTH    (0x15 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMACSL     (0x16 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMACSH     (0x17 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
```

```
// Bank 1 registers
#define EHT0       (0x00 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EHT1       (0x01 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EHT2       (0x02 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EHT3       (0x03 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EHT4       (0x04 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EHT5       (0x05 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EHT6       (0x06 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EHT7       (0x07 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM0      (0x08 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM1      (0x09 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM2      (0x0A | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM3      (0x0B | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM4      (0x0C | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM5      (0x0D | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM6      (0x0E | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM7      (0x0F | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMCSL     (0x10 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMCSH     (0x11 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMOL      (0x14 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMOH      (0x15 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define ERXFCON    (0x18 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define ERXFCON_UCEN_BIT   (1 << 7)
#define ERXFCON_ANDOR_BIT  (1 << 6)
#define ERXFCON_CRCEN_BIT  (1 << 5)
#define ERXFCON_PMEN_BIT   (1 << 4)
#define ERXFCON_MPEN_BIT   (1 << 3)
#define ERXFCON_HTEN_BIT   (1 << 2)
#define ERXFCON_MCEN_BIT   (1 << 1)
#define ERXFCON_BCEN_BIT   (1 << 0)
#define EPKTCNT    (0x19 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
// Bank 2 registers
#define MACON1     (0x00 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MACON1_TXPAUS_BIT  (1 << 3)
#define MACON1_RXPAUS_BIT  (1 << 2)
#define MACON1_PASSALL_BIT (1 << 1)
#define MACON1_MARXEN_BIT  (1 << 0)
#define MACON3     (0x02 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MACON3_PADCFG2_BIT (1 << 7)
#define MACON3_PADCFG1_BIT (1 << 6)
#define MACON3_PADCFG0_BIT (1 << 5)
```

```
#define MACON3_TXCRCEN_BIT (1 << 4)
#define MACON3_PHDRLEN_BIT (1 << 3)
#define MACON3_HFRMEN_BIT  (1 << 2)
#define MACON3_FRMLNEN_BIT (1 << 1)
#define MACON3_FULDPX_BIT  (1 << 0)
#define MACON4     (0x03 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MABBIPG    (0x04 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MAIPGL     (0x06 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MAIPGH     (0x07 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MACLCON1   (0x08 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MACLCON2   (0x09 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MAMXFLL    (0x0A | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MAMXFLH    (0x0B | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MICMD      (0x12 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MICMD_MIIRD_BIT   (1 << 0)
#define MIREGADR   (0x14 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MIWRL      (0x16 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MIWRH      (0x17 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MIRDL      (0x18 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MIRDH      (0x19 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
// Bank 3 registers
#define MAADR5     (0x00 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_3_BITS)
#define MAADR6     (0x01 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_3_BITS)
#define MAADR3     (0x02 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_3_BITS)
#define MAADR4     (0x03 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_3_BITS)
#define MAADR1     (0x04 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_3_BITS)
#define MAADR2     (0x05 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_3_BITS)
#define EBSTSD     (0x06 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define EBSTCON    (0x07 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define EBSTCSL    (0x08 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define EBSTCSH    (0x09 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define MISTAT     (0x0A | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_3_BITS)
#define MISTAT_BUSY_BIT   (1 << 0)
#define EREVID     (0x12 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define ECOCON     (0x15 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define EFLOCON    (0x17 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define EPAUSL     (0x18 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define EPAUSH     (0x19 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
// PHY registers
#define PHCON1     (0x00)
#define PHSTAT1    (0x01)
```

```
#define PHID1       (0x02)
#define PHID2       (0x03)
#define PHCON2      (0x10)
#define PHCON2_FRCLNK_BIT  (1 << 14)
#define PHCON2_TXIS_BIT    (1 << 13)
#define PHCON2_JABBER_BIT  (1 << 10)
#define PHCON2_HDLDIS_BIT  (1 << 8)
#define PHSTAT2  (0x11)
#define PHIE    (0x12)
#define PHIR    (0x13)
#define PHLCON  (0x14)
#define PHLCON_LACFG3_BIT  (1 << 11)
#define PHLCON_LACFG2_BIT  (1 << 10)
#define PHLCON_LACFG1_BIT  (1 << 9)
#define PHLCON_LACFG0_BIT  (1 << 8)
#define PHLCON_LBCFG3_BIT  (1 << 7)
#define PHLCON_LBCFG2_BIT  (1 << 6)
#define PHLCON_LBCFG1_BIT  (1 << 5)
#define PHLCON_LBCFG0_BIT  (1 << 4)
#define PHLCON_LFRQ1_BIT   (1 << 3)
#define PHLCON_LFRQ0_BIT   (1 << 2)
#define PHLCON_STRCH_BIT   (1 << 1)
#define MISTAT_BUSY_BIT    (1 << 0)
#define ENC28J60_HEADER_SIZE      6
#define ENC28J60_CRC_SIZE  4

typedef enum
{
  READ_CONTROL_REG,
  READ_BUFFER_MEM,
  WRITE_CONTROL_REG,
  WRITE_BUFFER_MEM,
  BIT_FIELD_SET,
  BIT_FIELD_CLEAR,
  SYSTEM_RESET,
  COMMANDS_NUM,
} ENC28J60_Command;

typedef enum
{
  CS_LOW = 0,
```

```
  CS_HIGH = 1,
} ENC28J60_CS_State;

typedef enum
{
  BANK_0,
  BANK_1,
  BANK_2,
  BANK_3,
} ENC28J60_RegBank;

typedef enum
{
  ETH_REG,
  MAC_MII_REG,
} ENC28J60_RegType;

typedef struct ENC28J60_Frame
{
  uint16_t nextPtr;
  uint16_t length;
  uint16_t status;
  uint8_t data[ENC28J60_FRAME_DATA_MAX];
  uint32_t checkSum;
} ENC28J60_Frame;

extern void ENC28J60_Init(void);
extern uint16_t ENC28J60_ReceiveFrame(ENC28J60_Frame* frame);
extern void ENC28J60_TransmitFrame(uint8_t *data, uint16_t size);

#endif // #ifndef ENC28J60_H
```

محتویات فایل ENC28J60.C

```
#include "enc28j60.h"
#include "common.h"
uint8_t macAddr[MAC_ADDRESS_BYTES_NUM] = {0x00, 0x17, 0x22, 0xED, 0xA5, 0x01};
uint8_t ipAddr[IP_ADDRESS_BYTES_NUM] = {255, 255, 255, 255};
static uint8_t commandOpCodes[COMMANDS_NUM] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05,
0x07};
static ENC28J60_RegBank curBank = BANK_0;
static uint16_t curPtr = ENC28J60_RX_BUF_START;
extern SPI_HandleTypeDef hspi1;

static void SetCS(ENC28J60_CS_State state)
{
    HAL_Delay(1);
  HAL_GPIO_WritePin(ENC28J60_CS_PORT, ENC28J60_CS_PIN, (GPIO_PinState)state);
    HAL_Delay(1);
}

/*---------------------------------------------------------------------------*/
static void WriteBytes(uint8_t* data, uint16_t size)
{
  HAL_StatusTypeDef res = HAL_SPI_Transmit(&hspi1, data, size, ENC28J60_SPI_TIMEOUT);
}
/*---------------------------------------------------------------------------*/
static void WriteByte(uint8_t data)
{
  HAL_StatusTypeDef res = HAL_SPI_Transmit(&hspi1, &data, 1, ENC28J60_SPI_TIMEOUT);
}
/*---------------------------------------------------------------------------*/
static uint8_t ReadByte()
{
  uint8_t txData = 0x00;
  uint8_t rxData = 0x00;
  HAL_StatusTypeDef res = HAL_SPI_TransmitReceive(&hspi1, &txData, &rxData, 1,
ENC28J60_SPI_TIMEOUT);
  return rxData;
}
/*---------------------------------------------------------------------------*/
static ENC28J60_RegType getRegType(uint8_t reg)
{
```

```
 ENC28J60_RegType type = (ENC28J60_RegType)((reg & ENC28J60_REG_TYPE_MASK) >>
ENC28J60_REG_TYPE_OFFSET);
 return type;
}
/*---------------------------------------------------------------------------*/
static ENC28J60_RegBank getRegBank(uint8_t reg)
{
 ENC28J60_RegBank bank = (ENC28J60_RegBank)((reg & ENC28J60_REG_BANK_MASK) >>
ENC28J60_REG_BANK_OFFSET);
 return bank;
}
/*---------------------------------------------------------------------------*/
static uint8_t getRegAddr(uint8_t reg)
{
 uint8_t addr = (reg & ENC28J60_REG_ADDR_MASK);
 return addr;
}
/*---------------------------------------------------------------------------*/
static void WriteCommand(ENC28J60_Command command, uint8_t argData)
{
 uint8_t data = 0;
 data = (commandOpCodes[command] << ENC28J60_OP_CODE_OFFSET) | argData;
 WriteByte(data);
}
/*---------------------------------------------------------------------------*/
static void CheckBank(uint8_t reg)
{
 uint8_t regAddr = getRegAddr(reg);
 if (regAddr < ENC28J60_COMMON_REGS_ADDR)
 {
  ENC28J60_RegBank regBank = getRegBank(reg);
  if (curBank != regBank)
  {
   uint8_t econ1Addr = getRegAddr(ECON1);

   // Clear bank bits
   SetCS(CS_LOW);
   WriteCommand(BIT_FIELD_CLEAR, econ1Addr);
   WriteByte(ECON1_BSEL1_BIT | ECON1_BSEL0_BIT);
   SetCS(CS_HIGH);
```

```
      // Set bank bits
      SetCS(CS_LOW);
      WriteCommand(BIT_FIELD_SET, econ1Addr);
      WriteByte(regBank);
      SetCS(CS_HIGH);
      curBank = regBank;
    }
  }
}
/*-----------------------------------------------------------------------*/
static void BitFieldSet(uint8_t reg, uint8_t regData)
{
  uint8_t regAddr = getRegAddr(reg);
  CheckBank(reg);

  SetCS(CS_LOW);
  WriteCommand(BIT_FIELD_SET, regAddr);
  WriteByte(regData);
  SetCS(CS_HIGH);
}
/*-----------------------------------------------------------------------*/
static void BitFieldClear(uint8_t reg, uint8_t regData)
{
  uint8_t regAddr = getRegAddr(reg);
  CheckBank(reg);

  SetCS(CS_LOW);
  WriteCommand(BIT_FIELD_CLEAR, regAddr);
  WriteByte(regData);
  SetCS(CS_HIGH);
}
/*-----------------------------------------------------------------------*/
static uint8_t ReadControlReg(uint8_t reg)
{
  uint8_t data = 0;
  ENC28J60_RegType regType = getRegType(reg);
  uint8_t regAddr = getRegAddr(reg);
  CheckBank(reg);

  SetCS(CS_LOW);
  WriteCommand(READ_CONTROL_REG, regAddr);
```

```
  if (regType == MAC_MII_REG)
  {
    ReadByte();
  }
  data = ReadByte();

  SetCS(CS_HIGH);
  return data;
}
/*-----------------------------------------------------------------------------*/
static void WriteControlReg(uint8_t reg, uint8_t regData)
{
  uint8_t regAddr = getRegAddr(reg);
  CheckBank(reg);
  SetCS(CS_LOW);
  WriteCommand(WRITE_CONTROL_REG, regAddr);
  WriteByte(regData);
  SetCS(CS_HIGH);
}
/*-----------------------------------------------------------------------------*/
static void WriteControlRegPair(uint8_t reg, uint16_t regData)
{
  WriteControlReg(reg, (uint8_t)regData);
  WriteControlReg(reg + 1, (uint8_t)(regData >> 8));
}
/*-----------------------------------------------------------------------------*/
//static uint16_t ReadControlRegPair(uint8_t reg)
//{
//  uint16_t data = 0;
//  data = (uint16_t)ReadControlReg(reg) | ((uint16_t)ReadControlReg(reg + 1) << 8);
//  return data;
//}
/*-----------------------------------------------------------------------------*/
static void WriteBufferMem(uint8_t *data, uint16_t size)
{
  SetCS(CS_LOW);
  WriteCommand(WRITE_BUFFER_MEM, ENC28J60_BUF_COMMAND_ARG);
  WriteBytes(data, size);
  SetCS(CS_HIGH);
}
```

```
/*-------------------------------------------------------------------------*/
static void ReadBufferMem(uint8_t *data, uint16_t size)
{
  SetCS(CS_LOW);
  WriteCommand(READ_BUFFER_MEM, ENC28J60_BUF_COMMAND_ARG);

  for (uint16_t i = 0; i < size; i++)
  {
    *data = ReadByte();
    data++;
  }

  SetCS(CS_HIGH);
}
/*-------------------------------------------------------------------------*/
static void SystemReset()
{
  SetCS(CS_LOW);
  WriteCommand(SYSTEM_RESET, ENC28J60_RESET_COMMAND_ARG);
  SetCS(CS_HIGH);

  curBank = BANK_0;
  HAL_Delay(100);
}
/*-------------------------------------------------------------------------*/
//static uint16_t ReadPhyReg(uint8_t reg)
//{
//  uint16_t data = 0;
//  uint8_t regAddr = getRegAddr(reg);
//
//  WriteControlReg(MIREGADR, regAddr);
//  BitFieldSet(MICMD, MICMD_MIIRD_BIT);
//
//  while((ReadControlReg(MISTAT) & MISTAT_BUSY_BIT) != 0);
//
//  BitFieldClear(MICMD, MICMD_MIIRD_BIT);
//  data = ReadControlRegPair(MIRDL);
//
//  return data;
//}
```

```
/*---------------------------------------------------------------------------*/
void ENC28J60_StartReceiving()
{
  BitFieldSet(ECON1, ECON1_RXEN_BIT);
}
/*---------------------------------------------------------------------------*/
static void WritePhyReg(uint8_t reg, uint16_t regData)
{
  uint8_t regAddr = getRegAddr(reg);

  WriteControlReg(MIREGADR, regAddr);
  WriteControlRegPair(MIWRL, regData);

  while((ReadControlReg(MISTAT) & MISTAT_BUSY_BIT) != 0);
}
/*---------------------------------------------------------------------------*/
void ENC28J60_Init(void)
{
      HAL_Delay(500);
  HAL_GPIO_WritePin(ENC28J60_RESET_PORT, ENC28J60_RESET_PIN, GPIO_PIN_RESET);
  HAL_Delay(50);
  HAL_GPIO_WritePin(ENC28J60_RESET_PORT, ENC28J60_RESET_PIN, GPIO_PIN_SET);
  HAL_Delay(500);
  SystemReset();//software reset
  HAL_Delay(500);
  // Rx/Tx buffers
  WriteControlRegPair(ERXSTL, ENC28J60_RX_BUF_START);
  WriteControlRegPair(ERXNDL, ENC28J60_RX_BUF_END);
  WriteControlRegPair(ERDPTL, ENC28J60_RX_BUF_START);
  // MAC address
  WriteControlReg(MAADR1, macAddr[0]);
  WriteControlReg(MAADR2, macAddr[1]);
  WriteControlReg(MAADR3, macAddr[2]);
  WriteControlReg(MAADR4, macAddr[3]);
  WriteControlReg(MAADR5, macAddr[4]);
  WriteControlReg(MAADR6, macAddr[5]);
  WriteControlReg(MACON1,      MACON1_TXPAUS_BIT      |      MACON1_RXPAUS_BIT      |
MACON1_MARXEN_BIT);
```

```
  WriteControlReg(MACON3, MACON3_PADCFG0_BIT | MACON3_TXCRCEN_BIT |
MACON3_FRMLNEN_BIT);
  WriteControlRegPair(MAIPGL, ENC28J60_NBB_PACKET_GAP);
  WriteControlReg(MABBIPG, ENC28J60_BB_PACKET_GAP);
  WriteControlRegPair(MAMXFLL, ENC28J60_FRAME_DATA_MAX);
    // PHY resisters
  WritePhyReg(PHCON2, PHCON2_HDLDIS_BIT);
  ENC28J60_StartReceiving();
}
/*-----------------------------------------------------------------------*/
uint16_t ENC28J60_ReceiveFrame(ENC28J60_Frame* frame)
{
  uint16_t dataSize = 0;
  uint8_t packetsNum = ReadControlReg(EPKTCNT);
  if (packetsNum > 0)
  {
    WriteControlRegPair(ERDPTL, curPtr);
   ReadBufferMem((uint8_t*)frame, ENC28J60_HEADER_SIZE);
    curPtr = frame->nextPtr;
    if ((frame->status & ENC28J60_FRAME_RX_OK_MASK) != 0)
    {
      dataSize = frame->length - ENC28J60_CRC_SIZE;
      if (dataSize > ENC28J60_FRAME_DATA_MAX)
      {
        dataSize = ENC28J60_FRAME_DATA_MAX;
      }
     ReadBufferMem((uint8_t*)&(frame->data[0]), dataSize);
      ReadBufferMem((uint8_t*)&(frame->checkSum), ENC28J60_CRC_SIZE);
    }
    uint16_t nextPtr = frame->nextPtr - 1;
    if (nextPtr > ENC28J60_RX_BUF_END)
    {
      nextPtr = ENC28J60_RX_BUF_END;
    }
    WriteControlRegPair(ERXRDPTL, nextPtr);
    BitFieldSet(ECON2, ECON2_PKTDEC_BIT);
  }
  return dataSize;
}
```

```
/*---------------------------------------------------------------------------*/
void ENC28J60_TransmitFrame(uint8_t *data, uint16_t size)
{
  while((ReadControlReg(ECON1) & ECON1_TXRTS_BIT) != 0)
  {
    if((ReadControlReg(EIR) & EIR_TXERIF_BIT) != 0)
    {
      BitFieldSet(ECON1, ECON1_TXRST_BIT);
      BitFieldClear(ECON1, ECON1_TXRST_BIT);
    }
  }
 WriteControlRegPair(EWRPTL, ENC28J60_TX_BUF_START);
  uint8_t controlByte = 0x00;
  WriteBufferMem(&controlByte, 1);
  WriteBufferMem(data, size);
 WriteControlRegPair(ETXSTL, ENC28J60_TX_BUF_START);
  WriteControlRegPair(ETXNDL, ENC28J60_TX_BUF_START + size);
  BitFieldSet(ECON1, ECON1_TXRTS_BIT);
}
```

محتويات فايل common.h

```
#ifndef COMMON_H
#define COMMON_H

#define MAC_ADDRESS_BYTES_NUM                    6
#define IP_ADDRESS_BYTES_NUM                     4
#define htons(val)      ((val << 8) & 0xFF00) | ((val >> 8) & 0xFF)
#define htonl(val)    ((val << 8) & 0xFF0000) | ((val >> 8) & 0xFF00) | ((val << 24) & 0xFF000000)
| ((val >> 24) & 0xFF)

#define ntohs(val)         htons(val)
#define ntohl(val)         htonl(val)

extern uint8_t ipAddr[IP_ADDRESS_BYTES_NUM];
extern  uint8_t macAddr[MAC_ADDRESS_BYTES_NUM];

#endif // #ifndef COMMON_H
```

محتویات فایل ETHERNET.h

```
#ifndef ETHERNET_H
#define ETHERNET_H

#include "stm32f1xx_hal.h"
#include "..\\ENC28J60\\enc28j60.h"
#include "..\\ENC28J60\\common.h"


#define ETH_FRAME_TYPE_ARP   0x0806
#define ETH_FRAME_TYPE_IP                0x0800

typedef struct ETH_Frame
{
  uint8_t destMacAddr[MAC_ADDRESS_BYTES_NUM];
  uint8_t srcMacAddr[MAC_ADDRESS_BYTES_NUM];
  uint16_t etherType;
  uint8_t data[];
} ETH_Frame;

void ETH_Process(ENC28J60_Frame* encFrame);

#endif // #ifndef ETHERNET_H
```

محتویات فایل ETHERNET.C

```
#include <string.h>
#include "ethernet.h"
#include "..\\ARP\\arp.h"
#include "..\\IP\\ip.h"
static void ETH_Response(ETH_Frame* ethFrame, uint16_t len)
{
  memcpy(ethFrame->destMacAddr, ethFrame->srcMacAddr, MAC_ADDRESS_BYTES_NUM);
  memcpy(ethFrame->srcMacAddr, macAddr, MAC_ADDRESS_BYTES_NUM);

  ENC28J60_TransmitFrame((uint8_t*)ethFrame, len + sizeof(ETH_Frame));
}
/*----------------------------------------------------------------------------*/
void ETH_Process(ENC28J60_Frame* encFrame)
{
  uint16_t responseSize = 0;
  uint16_t requestSize = ENC28J60_ReceiveFrame(encFrame);
  if (requestSize > 0)
  {
    ETH_Frame* ethFrame = (ETH_Frame*)encFrame->data;
    uint16_t etherType = ntohs(ethFrame->etherType);
    uint16_t ethDataLen = requestSize - sizeof(ETH_Frame);
     // ARP protocol
    if (etherType == ETH_FRAME_TYPE_ARP)
    {
      responseSize = ARP_Process((ARP_Frame*)ethFrame->data, ethDataLen);
    }
     // IP protocol
    if (etherType == ETH_FRAME_TYPE_IP)
    {
      responseSize = IP_Process((IP_Frame*)ethFrame->data, ethDataLen);
    }
   if (responseSize > 0)
    {
      ETH_Response(ethFrame, responseSize);
    }
  }
}
```

محتویات فایل **ARP.h**

```
#ifndef ARP_H
#define ARP_H

#include "stm32f1xx_hal.h"
#include "..\\ENC28J60\\common.h"

#define ARP_OP_CODE_REQUEST         0x0001
#define ARP_OP_CODE_RESPONSE        0x0002

typedef struct ARP_Frame
{
  uint16_t hType;
  uint16_t pType;
  uint8_t hLen;
  uint8_t pLen;
  uint16_t opCode;
  uint8_t srcMacAddr[MAC_ADDRESS_BYTES_NUM];
  uint8_t srcIpAddr[IP_ADDRESS_BYTES_NUM];
  uint8_t destMacAddr[MAC_ADDRESS_BYTES_NUM];
  uint8_t destIpAddr[IP_ADDRESS_BYTES_NUM];
} ARP_Frame;

extern uint16_t ARP_Process(ARP_Frame* arpFrame, uint16_t frameLen);

#endif // #ifndef ARP_H
```

محتويات فايل ARP.C

```
#include "arp.h"
#include <string.h>

uint16_t ARP_Process(ARP_Frame* arpFrame, uint16_t frameLen)
{
  uint16_t newFrameLen = 0;

  if (memcmp(arpFrame->destIpAddr, ipAddr, IP_ADDRESS_BYTES_NUM) == 0)
  {
    if ((arpFrame->opCode) == ( ntohs(ARP_OP_CODE_REQUEST) ) )
    {
      memcpy(arpFrame->destMacAddr,arpFrame->srcMacAddr, AC_ADDRESS_BYTES_NUM);
      memcpy(arpFrame->srcMacAddr, macAddr, MAC_ADDRESS_BYTES_NUM);

      memcpy(arpFrame->destIpAddr, arpFrame->srcIpAddr, IP_ADDRESS_BYTES_NUM);
      memcpy(arpFrame->srcIpAddr, ipAddr, IP_ADDRESS_BYTES_NUM);

      arpFrame->opCode = htons(ARP_OP_CODE_RESPONSE);
      newFrameLen = frameLen;
    }
  }

  return newFrameLen;
}
```

محتویات فایل IP.h

```
#ifndef IP_H
#define IP_H

#include "stm32f1xx_hal.h"
#include "..\\Modules\\Ethernet\\ethernet.h"
#include "..\\ENC28J60\\common.h"

///icmp=0x01, udp=17, tcp=6
#define IP_FRAME_PROTOCOL_ICMP     0x01
#define IP_FRAME_PROTOCOL_UDP      0x11
#define IP_FRAME_PROTOCOL_TCP      0x06

extern ENC28J60_Frame frame;
extern uint8_t ipAddrglobal[IP_ADDRESS_BYTES_NUM];
typedef struct IP_Frame
{
  uint8_t verHeaderLen;
  uint8_t diffServices;
  uint16_t len;
  uint16_t fragId;
  uint16_t fragOffset;
  uint8_t timeToLive;
  uint8_t protocol;
  uint16_t checkSum;
  uint8_t srcIpAddr [ IP_ADDRESS_BYTES_NUM ] ;
  uint8_t destIpAddr [ IP_ADDRESS_BYTES_NUM ] ;
  uint8_t data [] ;
} IP_Frame ;

uint16_t IP_CalcCheckSum ( uint8_t* data, uint16_t len);
uint16_t IP_Process ( IP_Frame* ipFrame, uint16_t frameLen ) ;

#endif // #ifndef IP_H
```

محتویات فایل IP.c

```
#include "ip.h"
#include "..\\ICMP\\icmp.h"
#include "..\\TCP\\tcp.h"
#include "..\\UDP\\udp.h"
#include "..\\ETHERNET\\ethernet.h"
#include <string.h>

uint16_t IP_CalcCheckSum( uint8_t* data, uint16_t len)
{
 uint32_t res = 0;
 uint16_t * ptr = ( uint16_t* ) data;

 while ( len > 1 )
 {
  res += *ptr;
  ptr ++ ;
  len -= 2 ;
 }

 if ( len > 0 )
 {
  res += * ( uint8_t* ) ptr;
 }
 while ( res > 0xffff )
 {
  res = ( res >> 16 ) + ( res & 0xFFFF ) ;
 }

 return ~ (( uint16_t ) res ) ;
}
/*-------------------------------------------------------------------------*/
uint16_t IP_Process ( IP_Frame * ipFrame, uint16_t frameLen )
{
 uint16_t newFrameLen = 0;

 if (memcmp(ipFrame->destIpAddr, ipAddr, IP_ADDRESS_BYTES_NUM) == 0)
 {
  uint16_t rxCheckSum = ipFrame->checkSum;
  ipFrame->checkSum = 0;
```

```
  uint16_t calcCheckSum = IP_CalcCheckSum((uint8_t*)ipFrame, sizeof(IP_Frame));

  if (rxCheckSum == calcCheckSum)
  {
    //uint16_t dataLen = frameLen - sizeof(IP_Frame);
     uint16_t dataLen = ntohs(ipFrame->len) - ((ipFrame->verHeaderLen & 0x0F)*4);
    uint16_t newDataLen = 0;

    if (ipFrame->protocol == IP_FRAME_PROTOCOL_ICMP)
    {
      newDataLen = ICMP_Process((ICMP_EchoFrame*)ipFrame->data, dataLen);
    }
    else if (ipFrame->protocol == IP_FRAME_PROTOCOL_UDP)
    {
      newDataLen = UDP_Process((UDP_Frame*)ipFrame->data, dataLen);
    }
   else if (ipFrame->protocol == IP_FRAME_PROTOCOL_TCP)
    {
      newDataLen = TCP_Process((TCP_Frame*)ipFrame->data, dataLen,ipFrame);
    }
    //newFrameLen = newDataLen + sizeof(IP_Frame);
     if(newDataLen)
              newFrameLen = newDataLen + sizeof(IP_Frame);
     else
              newFrameLen=0;

    ipFrame->len = htons(newFrameLen);

    ipFrame->fragId = 0;
    ipFrame->fragOffset = 0;

    memcpy(ipFrame->destIpAddr, ipFrame->srcIpAddr, IP_ADDRESS_BYTES_NUM);
    memcpy(ipFrame->srcIpAddr, ipAddr, IP_ADDRESS_BYTES_NUM);

    ipFrame->checkSum = IP_CalcCheckSum((uint8_t*)ipFrame, sizeof(IP_Frame));
   }
  }

  return newFrameLen;
 }
```

محتویات فایل ICMP.h

```
#ifndef ICMP_H
#define ICMP_H

#include "stm32f1xx_hal.h"
#include "..\\ENC28J60\\common.h"

#define ICMP_FRAME_TYPE_ECHO_REQUEST 0x08
#define ICMP_FRAME_TYPE_ECHO_REPLY 0x00

typedef struct ICMP_EchoFrame
{
  uint8_t type;
  uint8_t code;
  uint16_t checkSum;
  uint16_t id;
  uint16_t seqNum;
  uint8_t data [];
} ICMP_EchoFrame;

uint16_t ICMP_Process ( ICMP_EchoFrame* icmpFrame, uint16_t frameLen ) ;

#endif // #ifndef ICMP_H
```

محتویات فایل ICMP.c

```
#include "icmp.h"
#include "..\\IP\\ip.h"

uint16_t ICMP_Process ( ICMP_EchoFrame* icmpFrame, uint16_t frameLen )
{
 uint16_t newFrameLen = 0;

 uint16_t rxCheckSum = icmpFrame-> checkSum;
 icmpFrame->checkSum = 0;
 uint16_t calcCheckSum = IP_CalcCheckSum (( uint8_t* ) icmpFrame, frameLen ) ;

 if ( rxCheckSum == calcCheckSum )
 {
  if ( icmpFrame->type == ICMP_FRAME_TYPE_ECHO_REQUEST )
  {
   icmpFrame->type = ICMP_FRAME_TYPE_ECHO_REPLY;
   icmpFrame->checkSum = IP_CalcCheckSum (( uint8_t * ) icmpFrame, frameLen ) ;
   newFrameLen = frameLen;
  }
 }

 return newFrameLen;
}
```

محتویات فایل UDP.h

```
#ifndef UDP_H
#define UDP_H


#include "stm32f1xx_hal.h"
#include "..\\ENC28J60\\common.h"

#define UDP_DEMO_PORT        33333
#define UDP_DHCP_PORT        68

typedef struct UDP_Frame
{
  uint16_t srcPort;
  uint16_t destPort;
  uint16_t len;
  uint16_t checkSum;
  uint8_t data[];
} UDP_Frame;

uint16_t UDP_Process(UDP_Frame* udpFrame, uint16_t frameLen);

#endif // #ifndef UDP_H
```

محتویات فایل UDP.c

```
#include "..\\UDP\\udp.h"
#include "..\\IP\\ip.h"

uint16_t UDP_Process(UDP_Frame* udpFrame, uint16_t frameLen)
{
  uint16_t newFrameLen = 0;

  uint16_t destPort = ntohs(udpFrame->destPort);
  uint16_t len = ntohs(udpFrame->len);
  uint16_t dataLen = len - sizeof(UDP_Frame);

  if (destPort == UDP_DEMO_PORT)
  {
    for(uint16_t i = 0 ; i < dataLen - 1; i++)
    {
      udpFrame->data[i]++;
    }
  }

  uint16_t swapPort = udpFrame->destPort;
  udpFrame->destPort = udpFrame->srcPort;
  udpFrame->srcPort = swapPort;

  udpFrame->checkSum = 0;
  newFrameLen = len;

  return newFrameLen;
}
```

محتویات فایل TCP.h

```
#ifndef TCP_H
#define TCP_H

#include "stm32f1xx_hal.h"
#include "..\\ENC28J60\\common.h"
#include "..\\IP\\ip.h"

#define TCP_DEMO_PORT          2020
#define TCP_MYSEQ              0x1a
#define MSS         1460
#define OPTION_LENGTH(x) ( ( (x >> 12) & 0x000f) * 4 - 20 )
#define TCP_FIN_FLAG (1<<0)
#define TCP_SYN_FLAG (1<<1)
#define TCP_RST_FLAG (1<<2)
#define TCP_PSH_FLAG (1<<3)
#define TCP_ACK_FLAG (1<<4)
#define TCP_URG_FLAG (1<<5)
#define TCP_ECE_FLAG (1<<6)
#define TCP_CWR_FLAG (1<<7)

//tcp header length 20~60 (option 0~40)byte
typedef struct TCP_Frame
{
  uint16_t srcPort;
  uint16_t destPort;
  uint32_t   seqnum;
  uint32_t   acknum;
  uint16_t flags;
  uint16_t winsize;
  uint16_t chksum;
  uint16_t urgptr;
  uint8_t data[];//options+data
} TCP_Frame;

uint16_t  TCP_CalcCheckSum(uint8_t* data, uint16_t len, IP_Frame* ipframe);
uint16_t TCP_Process(TCP_Frame* tcpFrame, uint16_t frameLen, IP_Frame* ipframe);

#endif // #ifndef TCP_H
```

محتويات فايل TCP.h

```
#include "..\\TCP\\tcp.h"
#include "..\\IP\\ip.h"
#include <string.h>
#include <stdlib.h>

uint16_t  TCP_CalcCheckSum(uint8_t* data, uint16_t len, IP_Frame* ipframe)
{
    uint32_t checksum = 0;
    checksum += ipframe->protocol;
    checksum += ((ipframe->destIpAddr[0]<<8) + ipframe->destIpAddr[1]);
    checksum += ((ipframe->destIpAddr[2]<<8) + ipframe->destIpAddr[3]);
    checksum += ((ipframe->srcIpAddr[0]<<8) + ipframe->srcIpAddr[1]);
    checksum += ((ipframe->srcIpAddr[2]<<8) + ipframe->srcIpAddr[3]);
    checksum += len;
    // build the sum of 16bit words
    while(len >1)
    {
     checksum += 0xFFFF & (*data<<8 | *(data+1));
     data += 2;
     len-=2;
    }
    // if there is a byte left then add it (padded with zero on right of it)
    if (len)
    {
     checksum += (0xFF & *data)<<8;
    }
    // now calculate the sum over the bytes in the sum
    // until the result is only 16bit long
    while (checksum>>16)
    {
     checksum = (checksum & 0xFFFF)+(checksum >> 16);
    }
    // build 1's complement:
    checksum = (~checksum) & 0xFFFF;
    //or  return( (unsigned  int) checksum ^ 0xFFFF);

    return( htons((unsigned  int)(checksum)) );
}
```

```
uint32_t        myseq=0;
uint32_t        myack=0;

uint16_t TCP_Process(TCP_Frame* tcpFrame, uint16_t frameLen, IP_Frame* ipframe)
{
  uint16_t newFrameLen = 0;
  uint16_t flag=0;
  uint16_t temp = 0;
  if ( (ntohs(tcpFrame->destPort)) == TCP_DEMO_PORT)
  {
     flag = htons(tcpFrame->flags);
      if( flag & TCP_SYN_FLAG)//it is SYN packet => create SYNACK packet
      {
       temp=0x0000;
      tcpFrame->flags = htons((temp | TCP_ACK_FLAG | TCP_SYN_FLAG | (((uint16_t)6)<<12)));
      myack=(ntohl(tcpFrame->seqnum)) +1;
      tcpFrame->acknum= htonl(myack);
      tcpFrame->seqnum = htonl(TCP_MYSEQ);
      myseq=TCP_MYSEQ+1;
      tcpFrame->winsize = htons(8192);
      tcpFrame->urgptr = 0;
      //only MSS=1460
      tcpFrame->data[0]=0x02;tcpFrame->data[1]=0x04;
      tcpFrame->data[2]=0x05;tcpFrame->data[3]=0xb4;
      newFrameLen = 24;
      }
      else if(flag & TCP_PSH_FLAG)
      {
        temp=0x0000;
        tcpFrame->flags = htons((temp | TCP_ACK_FLAG | TCP_PSH_FLAG | (((uint16_t)5)<<12)));
        myack=(ntohl(tcpFrame->seqnum)) +(frameLen-20);
        tcpFrame->acknum= htonl(myack);
        tcpFrame->seqnum = htonl(myseq);
        myseq+=(frameLen-20);
        tcpFrame->winsize = htons(8192);
        newFrameLen = frameLen;
      }
      else if(flag & TCP_FIN_FLAG)
      {
        temp=0x0000;
```

```
      tcpFrame->flags = htons((temp | TCP_ACK_FLAG | TCP_FIN_FLAG | TCP_RST_FLAG |
     (((uint16_t)5)<<12)));
      myack=(ntohl(tcpFrame->seqnum)) +(frameLen-20);
      tcpFrame->acknum= htonl(myack);
      tcpFrame->seqnum = htonl(myseq);
      myseq+=(frameLen-20);
      newFrameLen = frameLen;
     }
     else
     {
      newFrameLen = 0;
     }

      }//if == TCP_DEMO_PORT

  uint16_t swapPort = tcpFrame->destPort;
  tcpFrame->destPort = tcpFrame->srcPort;
  tcpFrame->srcPort = swapPort;
  tcpFrame->chksum = 0x0000;
  if(newFrameLen)
    tcpFrame->chksum = TCP_CalcCheckSum((uint8_t*)tcpFrame, newFrameLen, ipframe);

  return newFrameLen;
}
```

و بخش اصلی فایل main.c شامل تعریف متغیر frame و تابع main

```
ENC28J60_Frame frame;
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_SPI1_Init();
  /* USER CODE BEGIN 2 */

ENC28J60_Init();

// DHCP Discovery

//0x00, 0x17, 0x22, 0xED, 0xA5, 0x01
frame.data[0]=0xff;frame.data[1]=0xff;frame.data[2]=0xff;frame.data[3]=0xff;
frame.data[4]=0xff;frame.data[5]=0xff;
frame.data[6]=0x00;frame.data[7]=0x17;frame.data[8]=0x22;frame.data[9]=0xED;
frame.data[10]=0xA5;frame.data[11]=0x01;
frame.data[12]=0x08;frame.data[13]=0x00;//IP protocol
/////////IP
```

```
frame.data[14]=0x45;//VER=4, HLen=5
frame.data[15]=0x00;//TOS=0x00
frame.data[16]=0x01;frame.data[17]=0x1A; // total length of IP packet=267byte
frame.data[18]=0x00;frame.data[19]=0x00; // ID=0x0000
frame.data[20]=0x40;frame.data[21]=0x00; // FLAG=0b010,Offset=0x00;
frame.data[22]=0x40; // TTL=64
frame.data[23]=0x11; // UDP/IP
frame.data[24]=0x00;frame.data[25]=0x00; //Header Checksum?
frame.data[26]=0x00;frame.data[27]=0x00;frame.data[28]=0x00;frame.data[29]=0x00;//SA IP
frame.data[30]=255;frame.data[31]=255;frame.data[32]=255;frame.data[33]=255;//DA IP

//calc checksum after set Total length
frame.data[24]=0x39;//IP_CalcCheckSum(&frame.data[14],20)>>8;
frame.data[25]=0xd4;//IP_CalcCheckSum(&frame.data[14],20);

///// UDP Header
frame.data[34]=0x00;frame.data[35]=0x44;//Source port=68
frame.data[36]=0x00;frame.data[37]=0x43;//Source port=67
frame.data[38]=0x01;frame.data[39]=0x06;// UDP datalength+8=239+8=244=0x00F7
frame.data[40]=0x48; frame.data[41]=0xD5;

//////////DHCP Header
frame.data[42]=0x01; //operation Request=1
frame.data[43]=0x01; //HType ethernet=1
frame.data[44]=0x06; //MAC Address Length=6
frame.data[45]=0x00; //hops=0
frame.data[46]=0xAA;frame.data[47]=0xBB;frame.data[48]=0xCC;frame.data[49]=0xDD;//XID
frame.data[50]=0x00;frame.data[51]=0x00;//SECs=0
frame.data[52]=0x80;frame.data[53]=0x00;//B bit=1
frame.data[54]=0x00;frame.data[55]=0x00;frame.data[56]=0x00;frame.data[57]=0x00;//ciaddr
frame.data[58]=0x00;frame.data[59]=0x00;frame.data[60]=0x00;frame.data[61]=0x00;//yiaddr
frame.data[62]=0x00;frame.data[63]=0x00;frame.data[64]=0x00;frame.data[65]=0x00;//siaddr
frame.data[66]=0x00;frame.data[67]=0x00;frame.data[68]=0x00;frame.data[69]=0x00;//giaddr
///MAC Address 16byte(octet) chaddr
frame.data[70]=0x00;frame.data[71]=0x17;frame.data[72]=0x22;frame.data[73]=0xED;
frame.data[74]=0xA5;frame.data[75]=0x01;frame.data[76]=0x00;frame.data[77]=0x00;
frame.data[78]=0x00;frame.data[79]=0x00;frame.data[80]=0x00;frame.data[81]=0x00;
frame.data[82]=0x00;frame.data[83]=0x00;frame.data[84]=0x00;frame.data[85]=0x00;
//192 byte(sname,files)=0
for(int k=0;k<202;k++)
    {frame.data[86+k]=0x00;}
```

```
//magic coockie
frame.data[278]=0x63;frame.data[279]=0x82;frame.data[280]=0x53;frame.data[281]=0x63;
frame.data[282]=0x35;frame.data[283]=0x01;frame.data[284]=0x01;
for(int k=285;k<295;k++)
    {frame.data[k]=0x00;}
frame.data[295]=0xff;// end of option list

ENC28J60_TransmitFrame(&frame.data[0],296);
HAL_Delay(500);
 /* USER CODE END 2 */

 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {
  /* USER CODE END WHILE */

   /* USER CODE BEGIN 3 */
         ETH_Process(&frame);

 }
 /* USER CODE END 3 */
}
```