

[ENC28J60] Bài 1: Giới thiệu module ethernet enc28j60 – giao tiếp với ENC28J60

31 Tháng Ba, 2020 Đào Nguyễn Ethernet, IoT tutorial 2



Trong tutorial này, chúng ta sẽ làm quen với module ENC28J60 và giao tiếp với ENC28J60, module này có thể xử lý hầu hết các yêu cầu về giao thức mạng. Module kết nối trực tiếp tới hầu hết các vi điều khiển thông qua chuẩn giao tiếp SPI với tốc độ truyền/nhận lên đến 20MHz.

Theo dõi toàn bộ tutorial giao tiếp với ENC28J60 [tại đây](#)

Chuẩn bị cho chuỗi bài học giao tiếp với ENC28J60

Sau một hồi đắn đo, mình đã quyết định thực hiện toàn bộ tutorial giao tiếp với ENC28J60 trên công cụ mô phỏng Proteus do phần này hỗ trợ mô phỏng ic ENC28J60

Còn về vi điều khiển và trình dịch thì mình chọn làm việc trên chip Atmega328, trình dịch Codevision AVR bởi :

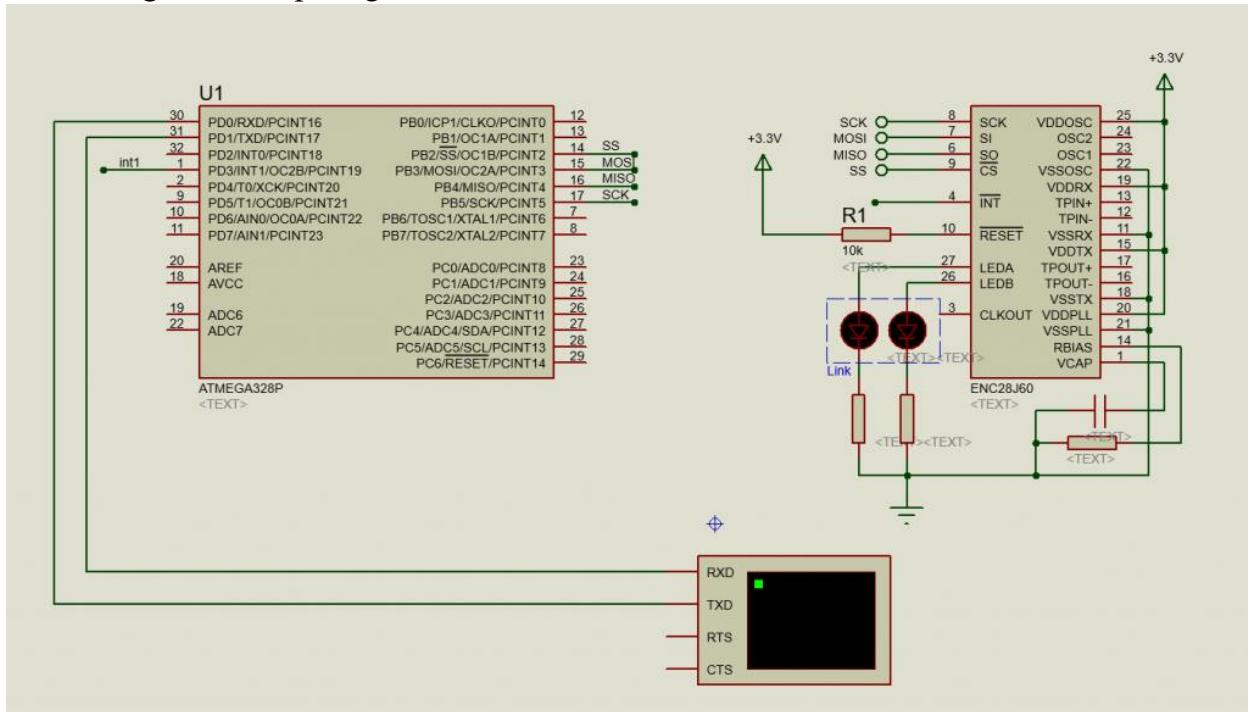
- So với các dòng 8051, PIC, thì dòng AVR có tốc độ nhanh hơn, bộ nhớ cao hơn
- Với STM8 thì chip này chưa hỗ trợ mô phỏng trên proteus
- Với STM32 thì phải cài các phiên bản cao hơn mới hỗ trợ mô phỏng, tuy nhiên mô phỏng chip stm32 thì trình mô phỏng chạy rất chậm, hay đơ

- Trình dịch AVR hỗ trợ tạo project sinh code rất tiện lợi, nó cũng hỗ trợ nhiều thư viện giúp ta có nhiều thời gian tập trung làm việc với chip ENC28J60 nhiều hơn mà không phải lò mò đi code từng thanh ghi trong chip avr
- Khi mô phỏng chúng ta debug SPI, UART rất tiện lợi

Do vậy, các bạn chuẩn bị các công cụ sau để theo dõi khóa học này nhé !

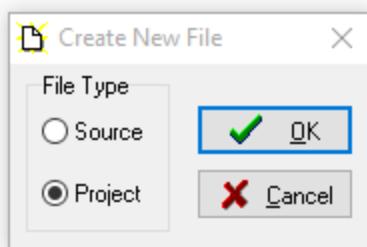
- Máy tính đã cài **Winpcap**
- Phần mềm mô phỏng Proteus 8.4
- Trình dịch CodeVision AVR

Vẽ chương trình mô phỏng



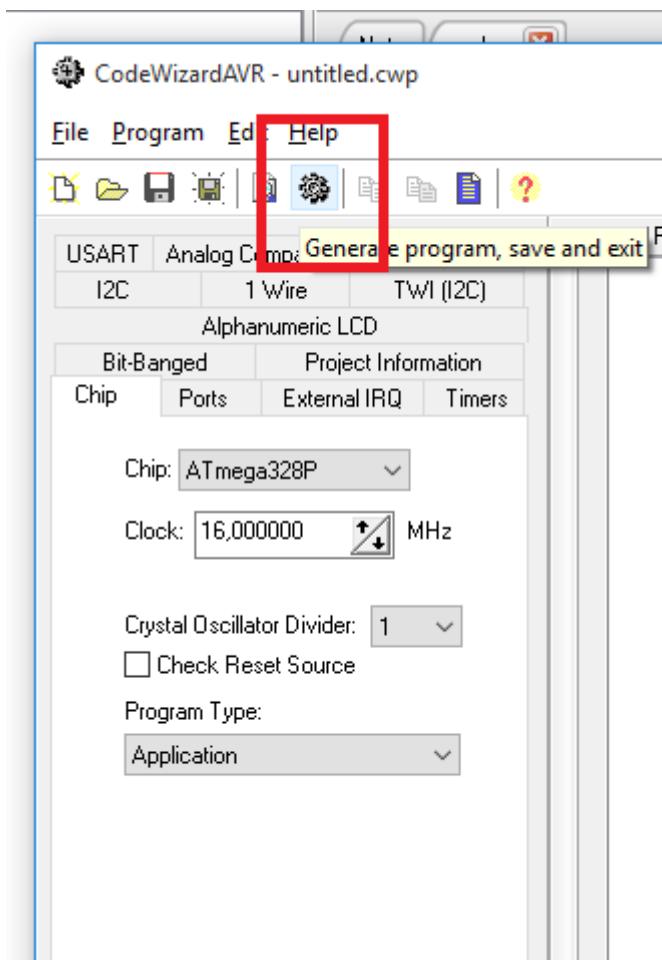
Khởi tạo project

Mở CodeVision AVR, vào **File** -> **New**, tích chọn **Project** -> **OK**



Chọn YES -> tích chọn **AT90,ATtiny, ATmega, FPLSLIC**

Trong cửa sổ **CodeWizardAVR**, các bạn chọn tab **Chip** và chọn chip ATmega328P, chọn thạch anh tốc độ **16,000000 Mhz** rồi tích vào biểu tượng răng cưa để tạo project



Phần mềm sẽ bắt ta lưu lại 1 số file, các bạn chọn vị trí lưu project với tên gì tùy các bạn ! Sau đó project sẽ được mở ra.

Xóa toàn bộ code phần mềm đã sinh cho các bạn rồi copy chõ code mình đã khởi tạo sẵn cho các bạn vào

```
1 #include <mega328p.h>
2 #include <delay.h>
3 #include <spi.h>
4 #include "uart.h"
5 void main(void)
```

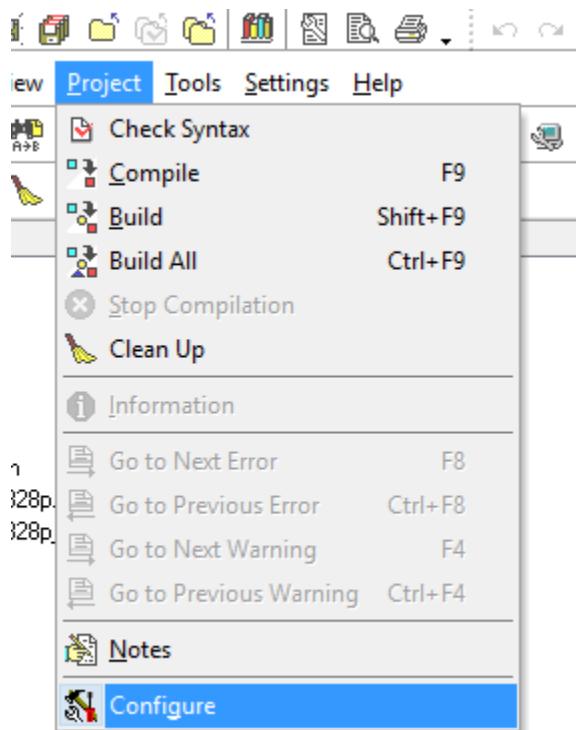
The image shows a code editor window with the following code listed:
1 #include <mega328p.h>
2 #include <delay.h>
3 #include <spi.h>
4 #include "uart.h"
5 void main(void)

```
6 {
7 #pragma optsize-
8 CLKPR=0x80;
9 CLKPR=0x00;
10#define _OPTIMIZE_SIZE_
11#pragma optsize+
12#endif
13
14PORTB=0xff;
15DDRB=0xff;
16DDRB.4=0;
17
18PORTC=0xff;
19DDRC=0xff;
20
21PORTD=0xff;
22DDRD=0xff;
23
24// SPI initialization
25// SPI Type: Master
26// SPI Clock Rate: 4000,000 kHz
27// SPI Clock Phase: Cycle Start
28// SPI Clock Polarity: Low
29// SPI Data Order: MSB First
30SPCR=0x50;
31SPSR=0x00;
32
33UART_init();
34#asm("sei")
35
36UART.putString("IOT47.com - ENC28J60 tutorial !!!\r\n");
37
38while (1)
```

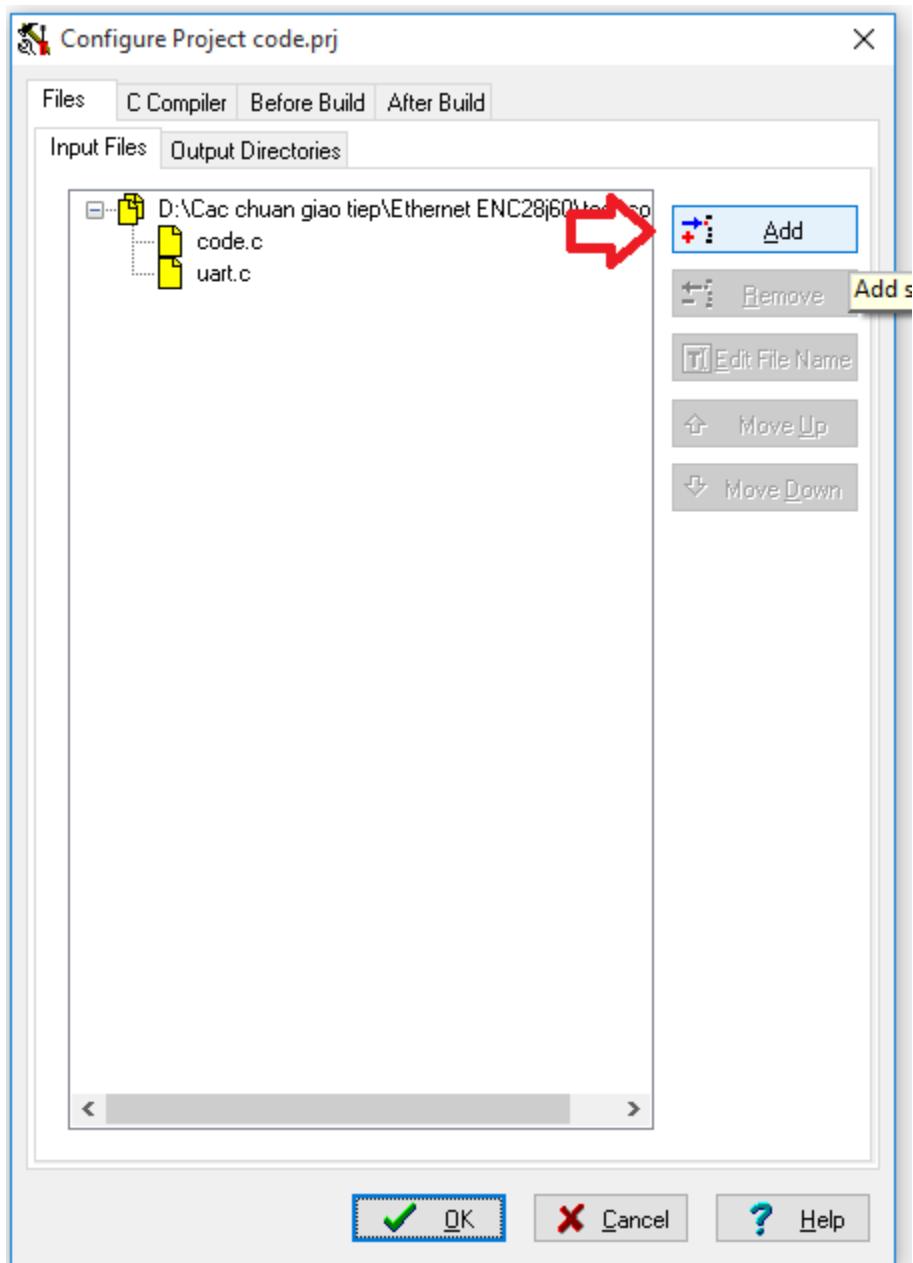
```
39  {
40
41
42  }
43}
```

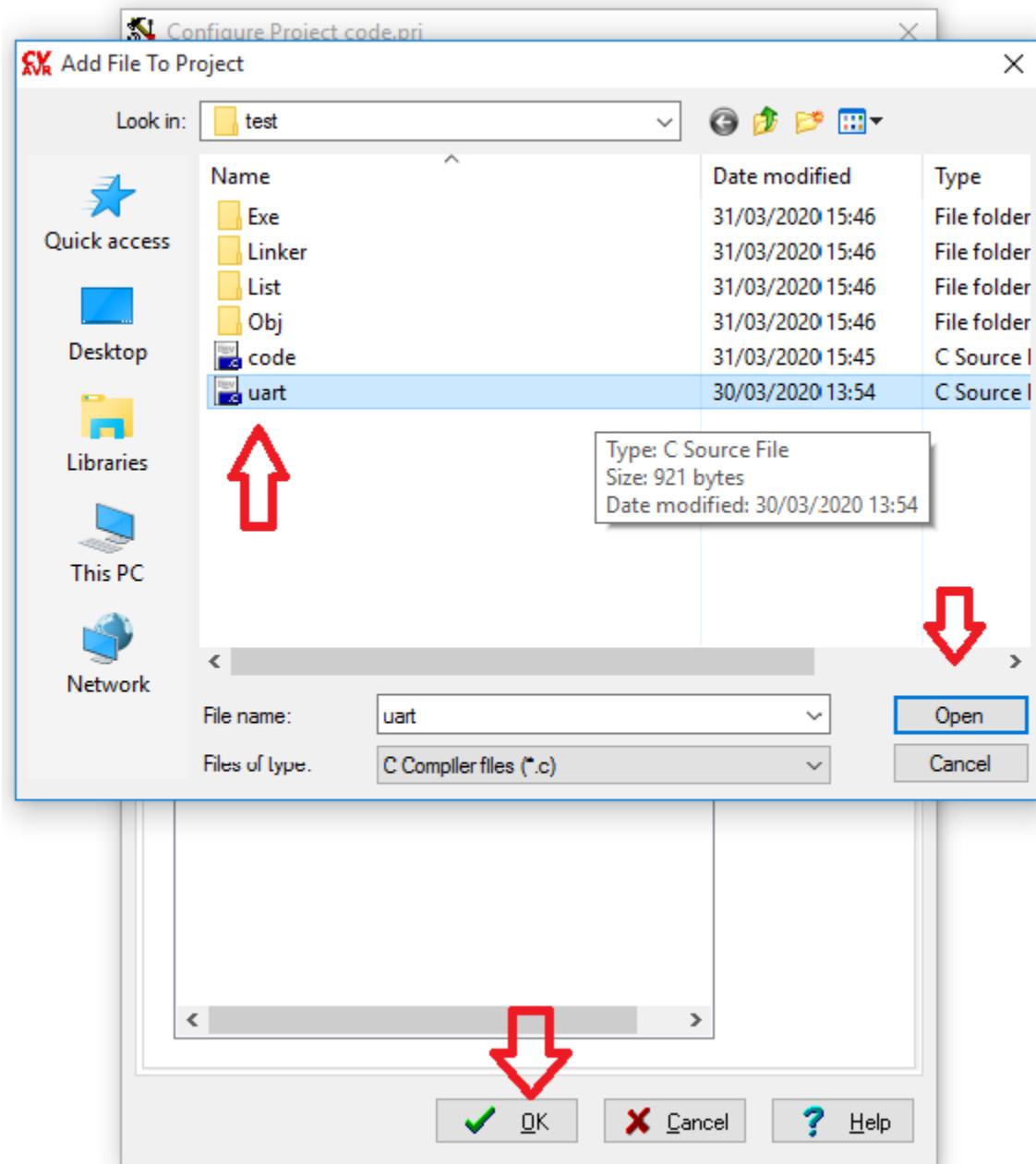
Ở đây mình có include thư viện <spi.h> <delay.h> đây là 2 thư viện phần mềm đã hỗ trợ sẵn cho chúng ta. Còn thư viện “uart.h” là do mình viết thêm, các bạn tải 2 file **uart.h** và **uart.c** về **tại đây** nhé !

Sau khi tải xong 2 file thư viện uart về thì các bạn bỏ nó vào cùng chỗ lưu project. Sau đó vào **Project –> Configure**

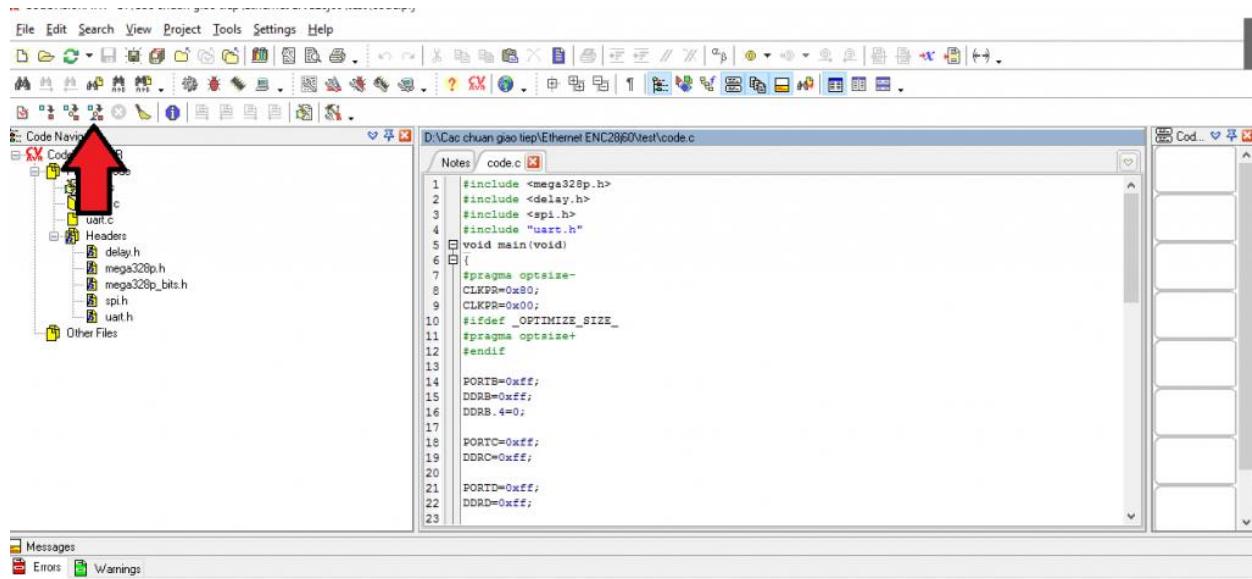


Click **ADD** và tiến hành Add file uart.c vào nhé !

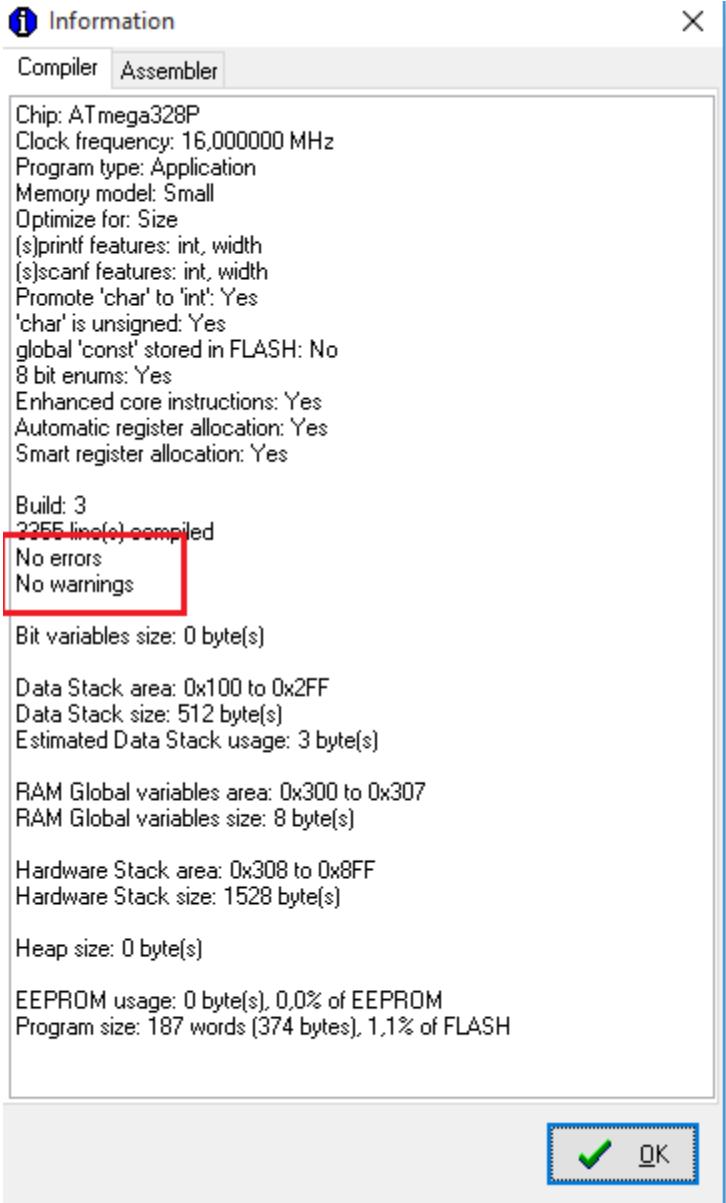




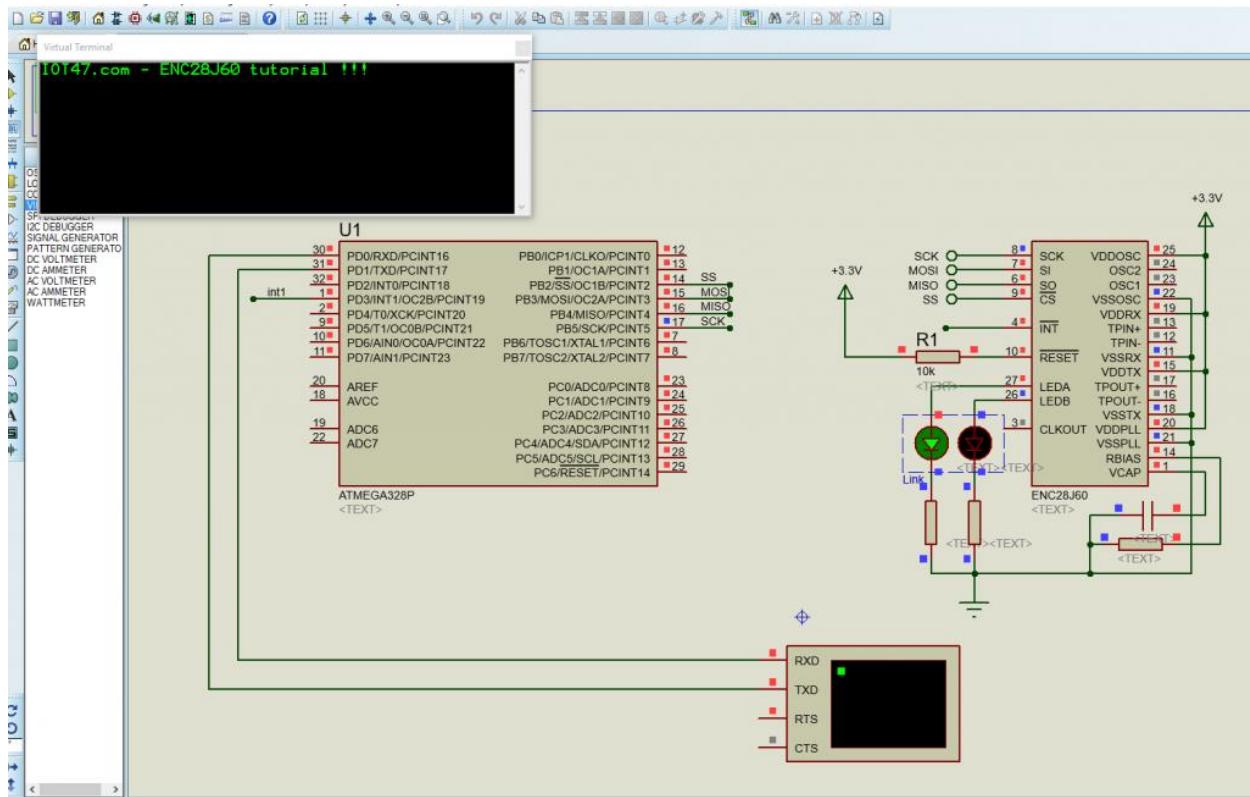
Như vậy là thư viện uart đã được thêm vào project. Các bạn ấn vào icon Build để biên dịch chương trình



Sau khi biên dịch xong nhận được thông báo không lỗi , không cảnh báo là OK !



File hex sẽ được sinh ra trong thư mục exe, chạy thử code và được như này là OK !



Kết luận

Như vậy, chúng ta đã hoàn thành các bước chuẩn bị cho hành trình phá đảo module enc28j60, hành trình phía trước sẽ rất vất vả và dài hơi, các bạn hãy chú ý theo dõi từng bài một nhé !

Các bạn có thể tải project mình đã chuẩn bị sẵn [tại đây](#) !

[ENC28J60] Bài 2: Khởi tạo module enc28j60 – giao tiếp ENC28J60

1 Tháng Tư, 2020 Đào Nguyễn Ethernet, IoT tutorial 11



Theo dõi toàn bộ tutorial hướng dẫn giao tiếp với enc28j60 [tại đây](#)

Hôm nay chúng ta sẽ bước đầu làm quen với các hàm khởi tạo, thiết lập module ENC28J60 để giao tiếp với enc28j60

Do khối lượng chương trình của chúng ta khá là đồ sộ nên việc viết hết code vào trong 1 file main không phải là ý kiến hay. Do vậy, ngay từ bây giờ mình sẽ chia file ra thành các phần nhỏ hơn để tiện cho việc viết code và chỉnh sửa, nâng cấp !

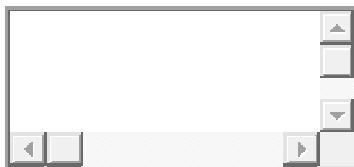
Mình sẽ tạo 2 file tên là `enc28j60.h` và `enc28j60.c`

File `enc28j60.h` sẽ là nơi khai báo các thư viện, các định nghĩa (`#define`) và khai báo nguyên mẫu hàm. Còn file `enc28j60.c` sẽ chứa nội dung của code

Các bạn vào **File -> New**, tích vào **source** để tạo file và lưu cùng cấp với chỗ để file `uart.c` `uart.h` luôn nhé !

Tạo xong thì add file `enc28j60.c` vào project bằng cách vào **Project -> Configure** rồi làm như cách mà các bạn đã add file `uart.c` ở [bài 1](#) ấy !

Ở file [enc28j60.h](#) mình sẽ thêm đoạn chương trình khai báo các thư viện cho file này !



```
1 #ifndef ENC28J60_H_
2 #define ENC28J60_H_
3 //-----
4 //Include cac thu vien can thiet
5 #include <mega328p.h>
6 #include <string.h>
7 #include <stdlib.h>
8 #include <stdint.h>
9 #include <stdio.h>
10#include <spi.h>
11#include "uart.h"
12//-----
13#define CS_GPIO_PORT PORTB
14#define CS_PIN      PORTB.2
15#define SS_SELECT() CS_PIN=0
16#define SS_DESELECT() CS_PIN=1
17//-----
18//khai bao nguyen mau ham
19
20//-----
21#endif /* ENC28J60_H_ */
```

Do mình kết nối chân CS của ENC29J60 với chân B2 nên mình khai báo ở trong file này luôn !

Lệnh [SS_SELECT\(\)](#) có tác dụng đưa chân B2 xuống 0 để enable giao tiếp SPI, còn [SS_DESELECT\(\)](#) kéo chân B2 lên 1 để kết thúc quá trình giao tiếp (nhường đường SPI cho thiết bị khác)

Tiếp đến, mình mở file [enc28j60.c](#) để bắt đầu viết các hàm giao tiếp với chip ENC28J60. Chúng ta sẽ bắt đầu với hàm [enc28j60_ini](#) để khởi tạo chip. Mình cũng viết sẵn thêm hàm [ENC28J60_error](#) có nhiệm vụ in thông báo lỗi ra màn uart debug nếu việc khởi tạo thất bại !



```
1 #include "enc28j60.h"
2 //-----
3 void ENC29J60_ini(void)
4 {
5
6 }
7 static void ENC28J60_error (void)
8 {
9     UART.putString("Khoi tao module ENC28J60 that bai !");
10}
11//-----
12//End file
```

Các bạn nhớ khai báo nguyên mẫu hàm của các hàm trong file .c vào file .h nhé !

Viết chương trình khởi tạo chip enc28j60

Hoạt động của chíp ENC28J60 phụ thuộc hoàn toàn vào các lệnh mà vi điều khiển đưa vào các thanh ghi điều khiển thông qua giao tiếp SPI. Các lệnh này có thể là 1 hoặc nhiều byte, mình sẽ gọi nó là các command. Mỗi command sẽ có 1 địa chỉ riêng và 1 value chứa các bit setup cho nó !

Cấu trúc địa chỉ của các command gồm: 3 bit Opcode + 5 bit Argument (tức 5bit địa chỉ) theo sau đó là 1 hoặc nhiều byte dữ liệu

Các command này có thể chia thành 7 nhóm theo bảng sau:

TABLE 4-1: SPI™ INSTRUCTION SET FOR THE ENC28J60

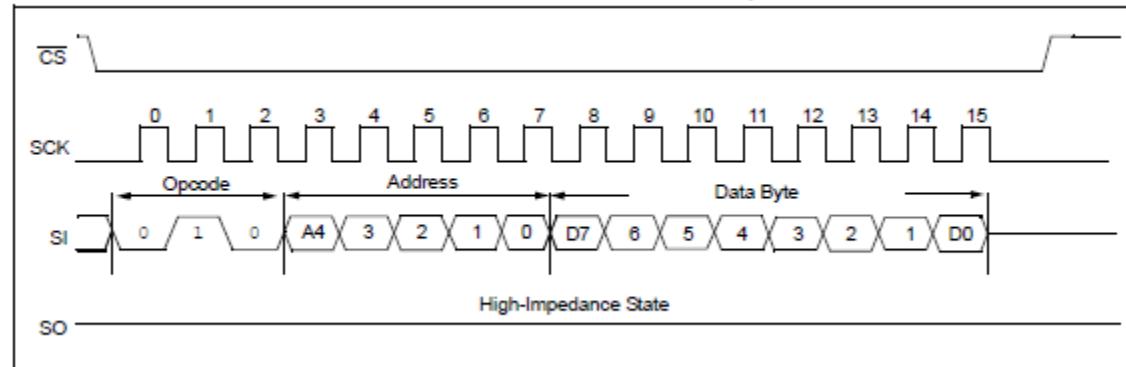
Instruction Name and Mnemonic	Byte 0					Byte 1 and Following							
	Opcode	Argument				Data							
Read Control Register (RCR)	0 0 0	a a a a a	N/A										
Read Buffer Memory (RBM)	0 0 1	1 1 0 1 0	N/A										
Write Control Register (WCR)	0 1 0	a a a a a	d d d d d d d d										
Write Buffer Memory (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 Command (Soft Reset) (SC)	1 1 1	1 1 1 1 1	N/A										

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

Như vậy 3 bit Opcode quyết định lệnh bạn gửi tới thuộc nhóm lệnh nào.

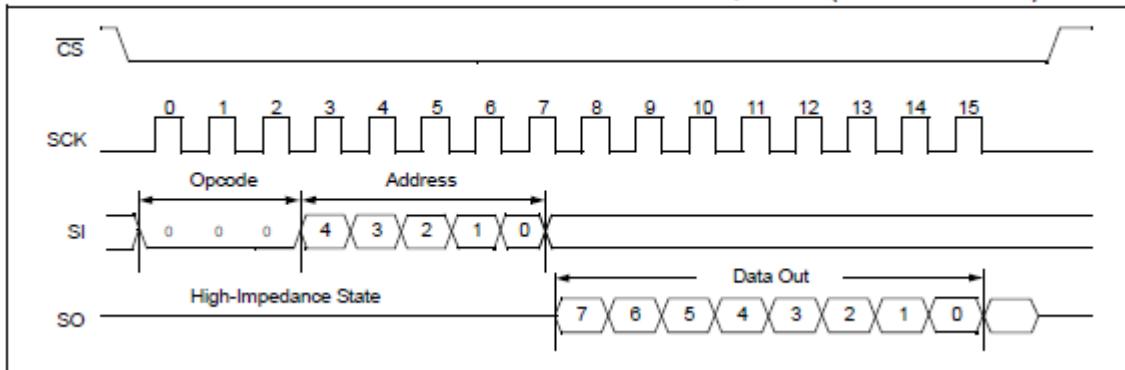
Hãy xem cách 1 command được gửi tới chip

FIGURE 4-5: WRITE CONTROL REGISTER COMMAND SEQUENCE



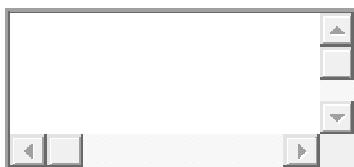
Việc đọc dữ liệu trong thanh ghi cũng tương tự

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



Viết chương trình đọc ghi cơ bản

Trước tiên mình sẽ viết 2 hàm phục vụ cho giao tiếp SPI ở file `enc28j60.c` đã



```

1 void SPI_SendByte(uint8_t _byte)
2 {
3     spi(_byte);
4 }
5 //-----
6 uint8_t SPI_ReceiveByte(void)
7 {
8     uint8_t _byte = spi(0xFF);
9     return _byte;
10}

```

Hàm `spi` là hàm trong thư viện `<spi.h>` mà phần mềm `CodeVision AVR` đã hỗ trợ chúng ta, chỉ việc lấy sai thui !

Hàm `SPI_SendByte` có nhiệm vụ gửi 1 byte data ra cổng SPI, còn hàm `SPI_ReceiveByte` dùng để đọc 1 byte data từ cổng SPI

Ở table 4.1, các bạn có thể thấy có tới 3 nhóm command có Opcode + Argument (5 bit Address) cố định, đó là RBM=0x3A dùng để đọc Buffer, WBM=0x7A dùng để ghi Buffer và SC=0xFF dùng reset chip

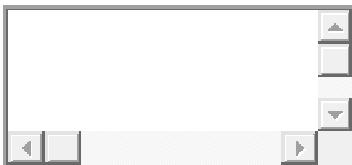
Tại file `enc28j60.c` mình sẽ viết 1 hàm để ghi 1 command = Opcode + Argument (nhớ thêm nguyên mẫu hàm vào file .h)



```
1void ENC28J60_write_command(uint8_t op,uint8_t adres, uint8_t data)
2{
3 SS_SELECT();
4 SPI_SendByte(op|(adres&0x1F));
5 SPI_SendByte(data);
6 SS_DESELECT();
7}
```

Nhu đã nói ở trên, 1 byte command gửi tới được ghép với 3 bit Opcode và 5 bit Address, do vậy đoạn lệnh `op|(adres&0x1F)` sẽ có nhiệm vụ ghép `op` và `adres` vào thành 1 byte, tuy nhiên, trước khi ghép bằng lệnh OR. Mình sẽ cho `adres&0x1F` để xóa 3bit cao nhất về 0 đã (dành chỗ cho 3 bit Opcode) `0x1F = 0b0001.1111`

Hàm ghi đã xong, giờ viết tiếp hàm đọc (nhớ thêm nguyên mẫu hàm vào file .h)



```
1static uint8_t ENC28J60_read_command(uint8_t op,uint8_t adres)
2{
3 uint8_t result;
4 SS_SELECT();
5 SPI_SendByte(op|(adres&0x1F));
6 result=SPI_ReceiveByte();
7 SS_DESELECT();
8 return result;
9}
```

Thiết lập bank

Tuy nhiên, hay nhìn vào bản đồ của các thanh ghi điều khiển

TABLE 3-1: ENC28J60 CONTROL REGISTER MAP

Bank 0		Bank 1		Bank 2		Bank 3	
Address	Name	Address	Name	Address	Name	Address	Name
00h	ERDPTL	00h	EHT0	00h	MACON1	00h	MAADR1
01h	ERDPHT	01h	EHT1	01h	MACON2	01h	MAADR0
02h	EWRPTL	02h	EHT2	02h	MACON3	02h	MAADR3
03h	EWRPTH	03h	EHT3	03h	MACON4	03h	MAADR2
04h	ETXSTL	04h	EHT4	04h	MABBPG	04h	MAADR5
05h	ETXSTH	05h	EHT5	05h	—	05h	MAADR4
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	MAMXFLLH	0Bh	—
0Ch	ERXRDP TL	0Ch	EPMM4	0Ch	Reserved	0Ch	—
0Dh	ERXRDP TH	0Dh	EPMM5	0Dh	MAPHSUP	0Dh	—
0Eh	ERXWRPTL	0Eh	EPMM6	0Eh	Reserved	0Eh	—
0Fh	ERXWRPTH	0Fh	EPMM7	0Fh	—	0Fh	—
10h	EDMASTL	10h	EPMC SL	10h	Reserved	10h	—
11h	EDMASTH	11h	EPMC SH	11h	MICON	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	EWOLIE	16h	MIWRL	16h	Reserved
17h	EDMACSH	17h	EWOLIR	17h	MIWRH	17h	EFLOCON
18h	—	18h	ERXFCON	18h	MIRDL	18h	EPAUSL
19h	—	19h	EPKT CNT	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

Cùng 1 địa chỉ có tới 4 loại command, để phân biệt các command có cùng địa chỉ, người ta thêm vào cái dở hơi gọi là bank, có 4 bank BANK0 BANK1 BANK2 BANK3. Trước thao tác đọc ghi command, chúng ta bắt buộc phải thông báo cho chip biết command này thuộc bank nào ! Thanh ghi chịu trách nhiệm lưu vị trí bank là ECON1 có địa chỉ là 0x1F. Cấu trúc của ECON1 như sau:

REGISTER 3-1: ECON1: ETHERNET CONTROL REGISTER 1

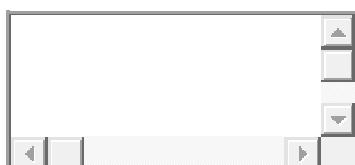
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TXRST	RXRST	DMAST	CSUMEN	TXRTS	RXEN	BSEL1	BSEL0
bit 7							bit 0

- bit 7 TXRST: Transmit Logic Reset bit
1 = Transmit logic is held in Reset
0 = Normal operation
- bit 6 RXRST: Receive Logic Reset bit
1 = Receive logic is held in Reset
0 = Normal operation
- bit 5 DMAST: DMA Start and Busy Status bit
1 = DMA copy or checksum operation is in progress
0 = DMA hardware is Idle
- bit 4 CSUMEN: DMA Checksum Enable bit
1 = DMA hardware calculates checksums
0 = DMA hardware copies buffer memory
- bit 3 TXRTS: Transmit Request To Send bit
1 = The transmit logic is attempting to transmit a packet
0 = The transmit logic is Idle
- bit 2 RXEN: Receive Enable bit
1 = Packets which pass the current filter configuration will be written into the receive buffer
0 = All packets received will be ignored
- bit 1-0 BSEL1:BSEL0: Bank Select bits
11 = SPI accesses registers in Bank 3
10 = SPI accesses registers in Bank 2
01 = SPI accesses registers in Bank 1
00 = SPI accesses registers in Bank 0

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared
		x = Bit is unknown

Như vậy chỉ có 2 bit thấp nhất BSEL1 và BSEL0 quyết định việc chọn bank. Mình sẽ định nghĩa thanh ghi ECON1 và vị trí trọng số của tất cả các bit trong thanh ghi này vào file .h luôn



```

1 // ENC28J60 ECON1 Register Bit Definitions
2 #define ECON1      0x1F
3 #define ECON1_TXRST 0x80
4 #define ECON1_RXRST 0x40
5 #define ECON1_DMAST 0x20
6 #define ECON1_CSUMEN 0x10
7 #define ECON1_TXRTS 0x08
8 #define ECON1_RXEN   0x04
9 #define ECON1_BSEL1  0x02
10#define ECON1_BSEL0  0x01

```

Thiết kế chương trình chọn bank

Do có tới 4 command có cùng 1 địa chỉ, ví dụ:
ERDPTL và EHT0 và MACON1 và MAADR1 đều có địa chỉ là 0x00

Bank 0	Bank 1	Bank 2	Bank 3
Address	Name	Address	Name
00h	ERDPTL	00h	EHT0
			00h
	MACON1		MAADR1

Bản thân chúng ta khi lập trình phải có thêm thông tin để phân biệt địa chỉ này là của bank nào, các bạn hãy để ý table 3-1 giải địa chỉ của các command chỉ bắt đầu từ 0x00 – > 0x1F là hết. Tức là chỉ cần 5bit đầu tiên (từ bit0 đến bit4) để lưu địa chỉ là đủ rồi, chúng ta sẽ sử dụng bit5 và bit6 để lưu bank của command đó.

Tức là với bank 0 mình sẽ OR với 0x00, với bank1 mình OR thêm 0x20, với bank2 thì OR thêm 0x40, với bank3 thì OR thêm 0x60

Ví dụ: ERDPTL có địa chỉ 0x00 thuộc bank 0 nên địa chỉ có nó sẽ là 0x00

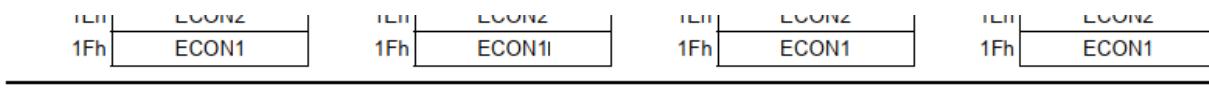
EHT0 thuộc bank 1 nên địa chỉ của nó sẽ là 0x00|0x20

MACON1 thuộc bank 2 nên địa chỉ của nó sẽ là 0x00|0x40

MAADR1 thuộc bank3 nên địa chỉ của nó là 0x00|0x60

Bây giờ khi ghi 1 command, mình sẽ tách ra để lấy 2 bit chứa bank tại vị trí bit6 và bit5, nhưng khi truyền ra cho ENC28J60 thì 2 bit này phải ở vị trí bit1 và bit0 nên mình sẽ dịch sang phải 5 phát cho nó đúng vị trí

Nhưng mà có tận 4 thanh ghi ECON1 trong 4 bank



Chúng ta sẽ sử dụng nhóm lệnh BFS (Bit Field Set) và BFC (Bit Field Clear) để thao tác 1 phát lên cả 4 thanh ghi ECON1 trên cả 4 bank.

Với Bit Field Set có Opcode là 0x80 dùng để set bit lên 1
và Bit Field Clear có Opcode là 0xA0 dùng để xóa bit về 0

Trước khi code cho hàm setbank mình sẽ #define các Opcode vào file .h để nhìn cho trực quan thay vì gõ thẳng mã hex nhìn không chuyên nghiệp 😊

The screenshot shows a code editor with the following content:

```
1//-----
2//dinh nghia cac Opcode
3#define ENC28J60_BIT_FIELD_SET    0x80
4#define ENC28J60_BIT_FIELD_CLR    0xA0
```

The code defines two constants: `ENC28J60_BIT_FIELD_SET` with value `0x80` and `ENC28J60_BIT_FIELD_CLR` with value `0xA0`. The code is written in C and uses preprocessor directives `#define`.

Các bước để ghi bank như sau:

1. Xóa dữ liệu của BSEL1 và BSEL0 trong ECON1 bằng nhóm lệnh có Opcode là Bit Field Clear (BFC)
2. Set dữ liệu bank mới của BSEL1 và BSEL0 vào các thanh ghi ECON1 bằng nhóm lệnh có Opcode là Bit Field Set (BFS)



```
1 static uint8_t Enc28j60Bank;  
2 static void ENC28J60_SetBank(uint8_t adres)  
3 {  
4     if ((adres&0x60)!=Enc28j60Bank) //neu bank hien tai khac bank da cai dat  
5     {  
6         ENC28J60_write_command(ENC28J60_BIT_FIELD_CLR,ECON1,ECON1_BSEL1|ECON1_BSEL0);  
7         Enc28j60Bank = adres&0x60; //luu lai de lan sau kiem tra  
8         ENC28J60_write_command(ENC28J60_BIT_FIELD_SET,ECON1,Enc28j60Bank>>5);  
9     }  
10}
```

Ở đây, mình có khởi tạo thêm biến **Enc28j60Bank** để lưu bank đã cài lại, như vậy lần sau chỉ cần kiểm tra nếu đã trùng bank rồi thì thôi không cần set lại nữa ! (nhớ thêm nguyên mẫu hàm vào file .h)

Tùy giờ khi #define địa chỉ của các thanh ghi hãy nhớ OR thêm bank tương ứng của nó vào nhé !

Viết các hàm điều khiển các thanh ghi thông thường

Để đọc ghi các thanh ghi thông thường mình sẽ sử dụng Opcode là Read Control Register=0x00 (RCR) và Write Control Register=0x40 (WCR), do đó mình tiếp tục #define cho 2 thằng này bên file .h nhé !



```
1#define ENC28J60_READ_CTRL_REG    0x00  
2#define ENC28J60_WRITE_CTRL_REG   0x40
```

Và đây là 2 hàm để đọc ghi vào các thanh ghi điều khiển



```
1 static void ENC28J60_writeByte(uint8_t adres,uint8_t data)
2 {
3     ENC28J60_SetBank(adres);
4     ENC28J60_write_command(ENC28J60_WRITE_CTRL_REG,adres,data);
5 }
6 //-----
7 static uint8_t ENC28J60_readByte(uint8_t adres)
8 {
9     ENC28J60_SetBank(adres);
10    return enc28j60_read_command(ENC28J60_READ_CTRL_REG,adres);
11 }
```

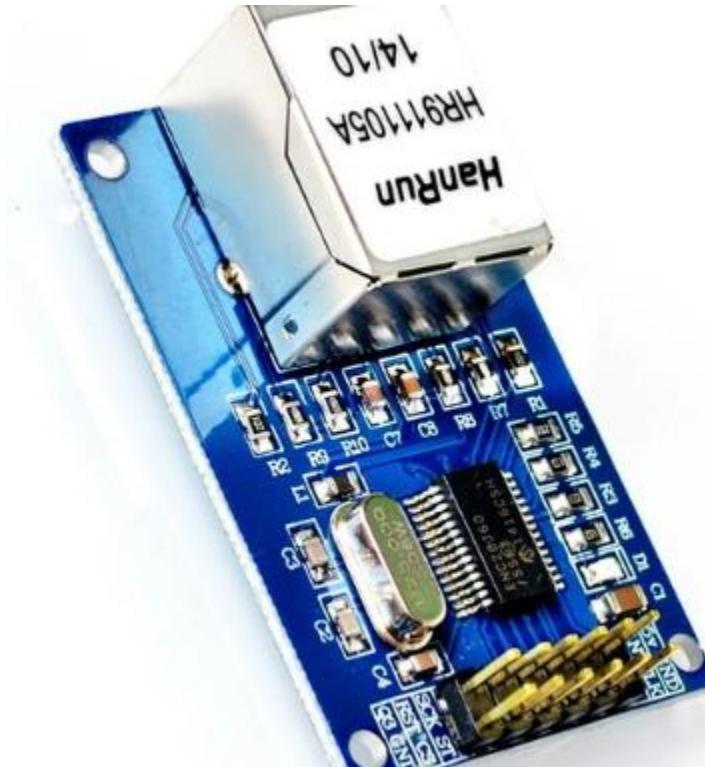
Trong bài tiếp , chúng ta sẽ tiếp tục hoàn thành chức năng khởi tạo ENC29J60, chứ viết bài dài quá các bạn chán không buồn đọc 😞

Download

Toàn bộ source code cho bài này các bạn tải [ở đây](#)

[ENC28J60] Bài 3: Khởi tạo module enc28j60 – giao tiếp enc28j60 (phần 2)

1 Tháng Tư, 2020 Đào Nguyễn Ethernet, IoT tutorial 1



Trong **phần trước** của chuỗi bài giao tiếp enc28j60 chúng ta đã viết các hàm cơ bản để đọc ghi vào các thanh ghi. Phần này chúng ta sẽ tiếp tục hoàn thiện chức năng khởi tạo chip !

Muốn biết làm sao để khởi tạo chip thì các bạn mở **datasheet** mục 6 người ta có hướng dẫn từng bước 1 luôn nhé !

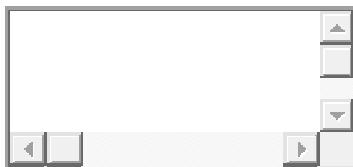
Các bạn chuyển tới hàm **ENC28J60_ini** chúng ta đã viết sẵn ngay từ đầu, đầu tiên mình sẽ reset mềm lại module bằng lệnh Opcode = System Command (Soft Reset) (SC) = 0xFF

Các bạn thêm #define cho nó vào file .h nhé



```
1#define ENC28J60_SOFT_RESET      0xFF
```

Trong hàm khởi tạo



```
1UART_putString("Dang khai tao ENC28J60 ...\\r\\n");
2ENC28J60_write_command(ENC28J60_SOFT_RESET,0x1F,0); //soft reset
```

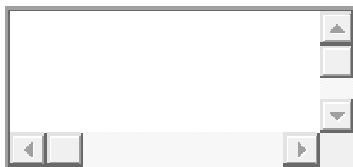
Bởi vì command thuộc nhóm reset nhận data là N/A tức là gì cũng được nên mình sẽ gửi byte 0 cho nó !

Trong datasheet trang 5 có nói, sau khi khởi tạo, nên có 1 khoảng thời gian trễ (khoảng 7500 chu kì máy) và nên kiểm tra bit CLKRDY trong ESTAT. Nếu bit này đã được set bằng 1 thì mọi thứ đã sẵn sàng. Do vậy mình sẽ thêm lệnh kiểm tra bit này nữa !



```
1 delay_ms(2);
2 while(!ENC28J60_read_command(ENC28J60_READ_CTRL_REG,ESTAT)&ESTAT_CLKRDY); //cho bit CLKRDY duoc
set
```

Hãy nhớ thêm định nghĩa cho các bit và thanh ghi ESTAT và file .h nhé !



```
1// ENC28J60 ESTAT Register Bit Definitions
2#define ESTAT      0x1D
3#define ESTAT_INT    0x80
4#define ESTAT_LATECOL 0x10
5#define ESTAT_RXBUSY  0x04
6#define ESTAT_TXABRT   0x02
7#define ESTAT_CLKRDY   0x01
```

Ngoài ra, do mình xài delay_ms nên chúng ta cũng cần add thêm thư viện <delay.h> mà phần mềm CodeVison AVR hỗ trợ vào file **enc28j60.h**

Ở đoạn chương trình trên, sau khi khởi tạo, mình delay 1 khoảng nhỏ rồi liên tục kiểm tra bit ESTAT_CLKRDY bằng vòng lặp while

Sau khi reset xong, để chắc chắn module có hoạt động, mình sẽ kiểm tra thử thanh ghi ERDPTL (ở 0x00 bank0)trong bảng **TABLE 3-2: ENC28J60 CONTROL REGISTER SUMMARY** thanh ghi này luôn có giá trị là 0xFA, ta sẽ căn cứ vào nó để kiểm tra module còn hoạt động không !

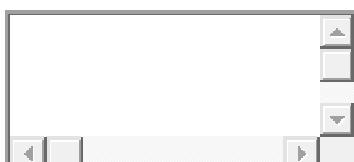
ENC28J60

TABLE 3-2: ENC28J60 CONTROL REGISTER SUMMARY

Register Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Reset	Details on Page
EIE	INTIE	PKTIE	DMAIE	LINKIE	TXIE	WOLIE	TXERIE	RXERIE	0000 0000	67
EIR	—	PKTIF	DMAIF	LINKIF	TXIF	WOLIF	TXERIF	RXERIF	-000 0000	68
ESTAT	INT	r	r	LATECOL	—	RXBUSY	TXABRT	CLKRDY ⁽¹⁾	0000 -000	66
ECON2	AUTOINC	PKTDEC	PWRSV	—	VRPS	—	—	—	100- 0---	16
ECON1	TXRST	RXRST	DMAST	CSUMEN	TXRTS	RXEN	BSEL1	BSEL0	0000 0000	15
ERDPTL	Read Pointer Low Byte ERDPT<7:0>								1111 1010	17
ERDPTH	Read Pointer High Byte (ERDPT<12:8>)								0 0101	17
EWRPTL	Write Pointer Low Byte (EWRPT<7:0>)								0000 0000	17
EWRPTH	—	—	—	Write Pointer High Byte (EWRPT<12:8>)					---0 0000	17
ETXSTL	TX Start Low Byte (ETXST<7:0>)								0000 0000	17

Các bạn định nghĩa thanh ghi này vào file .h nhé

```
// Bank 0 registers #define ERDPT (0x00|0x00)
```



```
1 if(ENC28J60_readByte(ERDPT) != 0xFA)ENC28J60_error(); //khai tao that bai
2 else
3 {
4
5 }
```

Đến thời điểm này, hàm khởi tạo module của mình sẽ như sau:



```
1 void ENC29J60_ini(void)
```

```

2 {
3 UART_putString("Dang khai tao ENC28J60 ...\\r\\n");
4 ENC28J60_write_command(ENC28J60_SOFT_RESET,0x1F,0); //soft reset
5 delay_ms(2);
6 while(!ENC28J60_read_command(ENC28J60_READ_CTRL_REG,ESTAT)& ESTAT_CLKRDY);
7 //cho bit CLKRDY duoc set
8 if(ENC28J60_readByte(ERDPT) != 0xFA)ENC28J60_error(); //khai tao that bai
9 else
10 {
11
12 }
13}

```

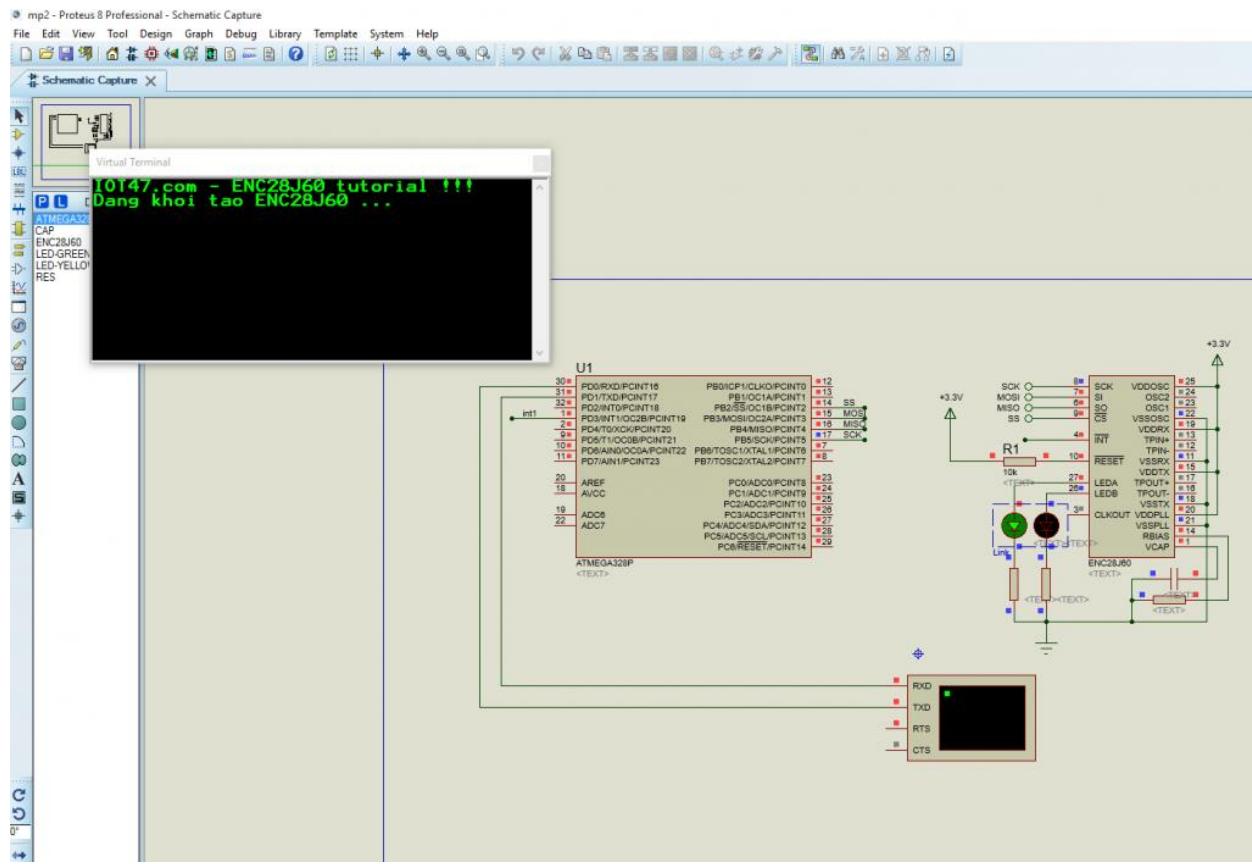
Trước khi viết tiếp hàm khởi tạo, chúng ta thử gọi hàm này vào hàm main trước vòng lặp while(1) để test qua tí đã chứ nhỉ 😊 viết nhiều rồi test thành quả tí đã

```

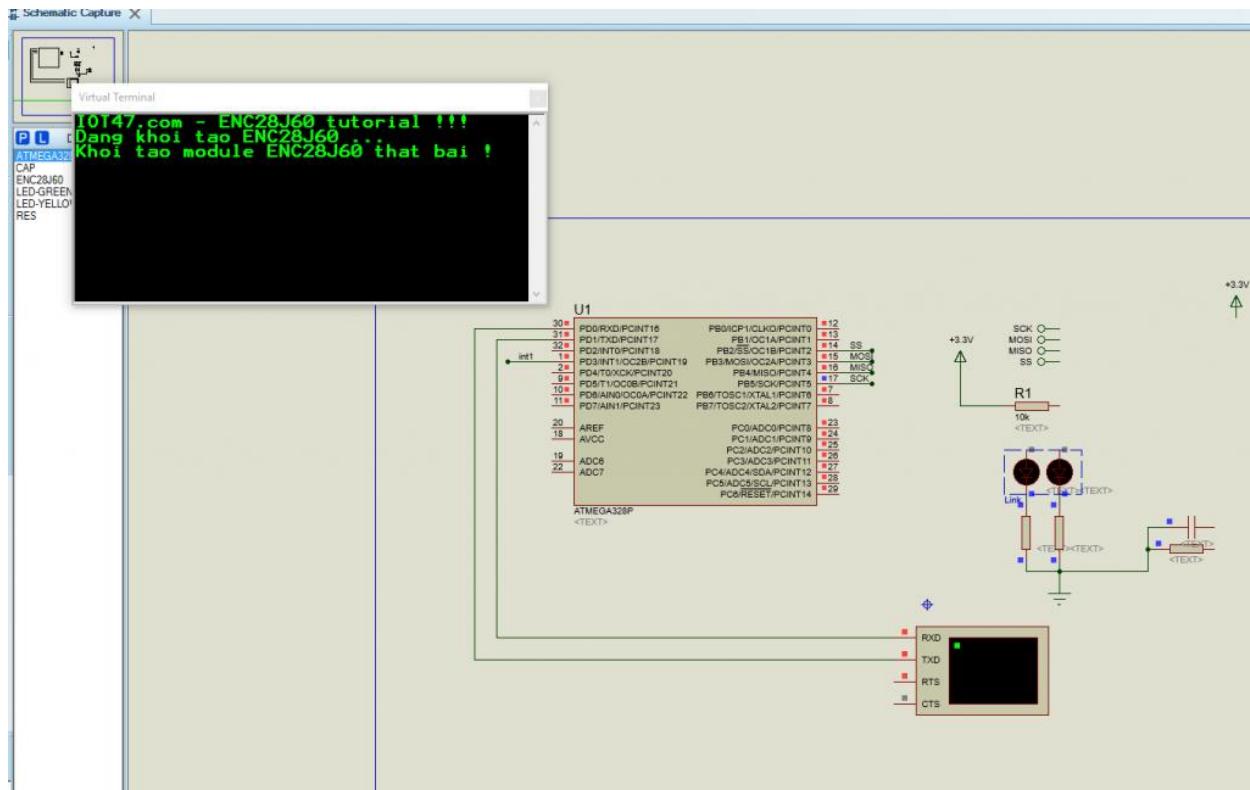
33
34     UART_init();
35     #asm("sei")
36
37     UART_putString("IOT47.com - ENC28J60 tutorial !!!\\r\\n");
38     ENC28J600_ini();
39     while (1)
40     {
41
42
43     }
44
45

```

Chạy mô phỏng thử khi có kết nối ic ENC28J60 thì sẽ không thấy thông báo thất bại !



Còn khi xóa ic ENC28J60 đi thì dòng khởi tạo thất bại ngay lập tức được in ra !

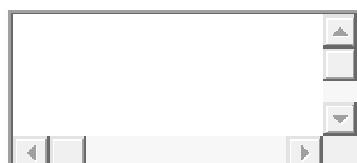


Rồi ok ! Tiếp tục viết thêm mã khởi tạo trong phần **else** của hàm khởi tạo nhé !

Các bạn có thể xem mục 6 (**INITIALIZATION**) trong **datasheet** để hiểu hơn về cách khởi tạo module nhé !

Mục 6.1 **datasheet** yêu cầu ta phải khởi tạo cho Receive Buffer qua thanh ghi **ERXST** và **ERXND**. Tương tự với Transmission Buffer cũng cần được khởi tạo tại **ETXST** và **ETXND**

Thực tế, khi coi bản đồ thanh ghi của chip (**TABLE 3-2: ENC28J60 CONTROL REGISTER SUMMARY**) các bạn sẽ không thay cái thanh ghi nào tên **ERXST** đâu, bởi nó là thanh ghi có độ dài tới 16bit, nên thằng ENC28J60 nó chia ra thành 2 thanh ghi nhỏ 8bit tên là **ERXSTL** và **ERXSTH**. Có rất nhiều thanh ghi 16bit như vậy, do đó mình sẽ viết thêm 1 hàm để ghi value 16bit vào 2 cho tiện



```

1 static void ENC28J60_writeByte16(uint8_t adres,uint16_t data)
2{
3 ENC28J60_writeByte(adres, data);
4 ENC28J60_writeByte(adres+1, data>>8);
5}

```

(thêm nguyên mẫu hàm vào file .h nhé)

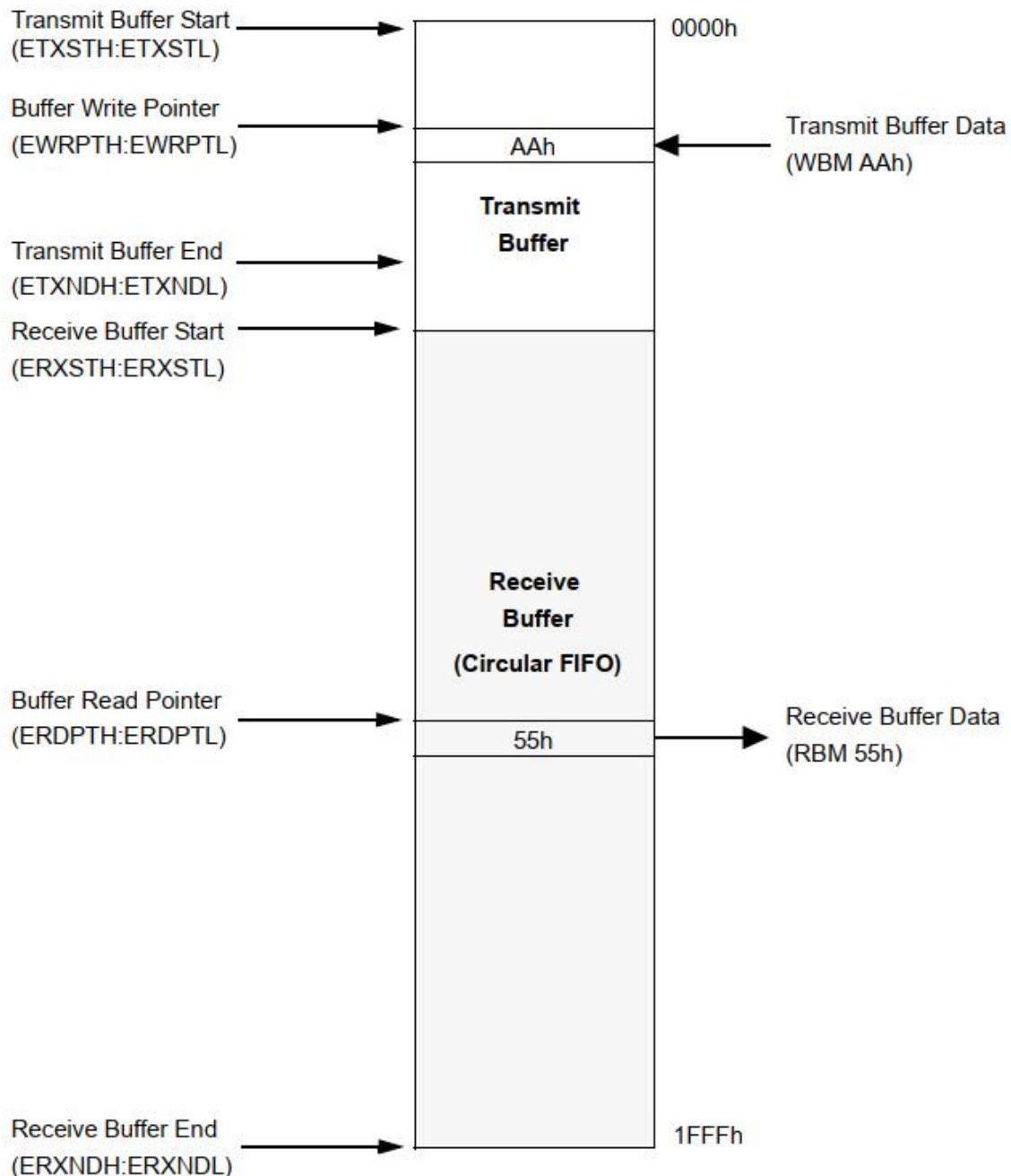
OK, quay trở lại hàm **ENC29J600_ini** chúng ta sẽ khởi tạo Receive Buffer (bộ đệm nhận) và (Transmission Buffer) như thế nào ?

Ethernet Buffer

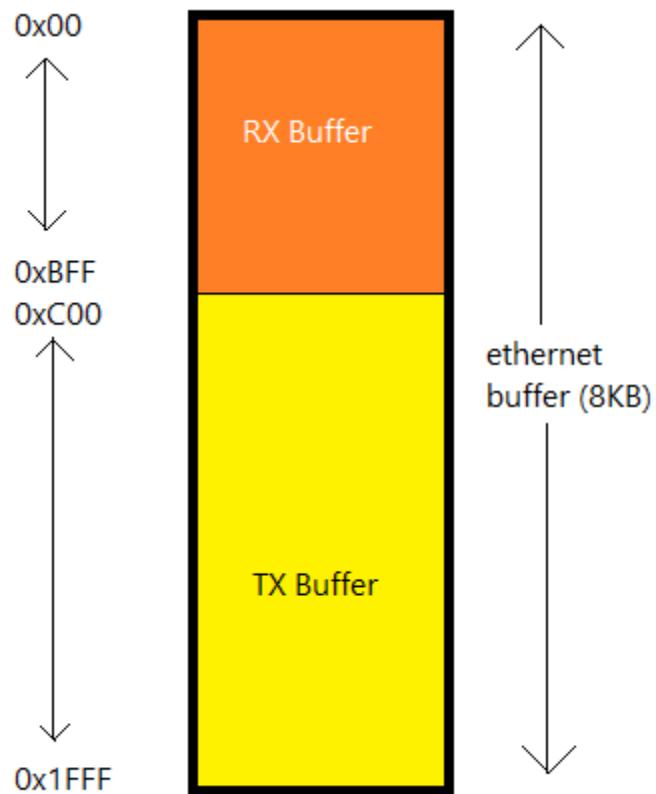
*Mục 3.2 **datasheet***

ENC28J60 có 8Kb Ram cho bộ đệm phục vụ truyền nhận các gói tin ! Chúng ta tùy ý cấu hình phần vùng cho miếng nào là nhận, miếng nào là truyền.

Địa chỉ của bộ đệm bắt đầu từ 0x00 đến 0x1FF



Ở đây mình sẽ chia cho bộ đệm nhận 3K bộ nhớ bắt đầu từ 0x00 đến 0x0BF, còn bộ đệm truyền là 5K bắt đầu từ 0xC00 tới hết (0x1FF)



Chúng ta #define các thông số sau vào file .h



```

1//-----
2#define RXSTART_INIT    0x0000 // start of RX buffer, room for 2 packets
3#define RXSTOP_INIT     0x0BFF // end of RX buffer
4//-----
5#define TXSTART_INIT    0x0C00 // start of TX buffer, room for 1 packet
6#define TXSTOP_INIT     0x11FF // end of TX buffer

```

Minh cũng #define địa chỉ của các thanh ghi **ERXST ERXND ETXST ETXND**
(minh gán địa chỉ của thanh ghi L cho nó)



```
1#define EWRPT      (0x02|0x00)
2#define ETXST       (0x04|0x00)
3#define ETXND       (0x06|0x00)
4#define ERXST       (0x08|0x00)
5#define ERXND       (0x0A|0x00)
6#define ERXRDPT     (0x0C|0x00)
7#define ERXWRPT     (0x0E|0x00)
```

Vì các thanh ghi này đều thuộc bank0 nên mình sẽ or thêm 0x00 (các bạn để cạnh thanh ghi **ERDPT** lúc nãy đã #define cho tiện theo dõi)

D:\Cac chuan giao tiep\Ethernet ENC28J60\test\enc28j60.h

	Notes	code.c	enc28j60.c	enc28j60.h
34	#define ECON1 0x1F			
35	#define ECON1_TXRST 0x80			
36	#define ECON1_RXRST 0x40			
37	#define ECON1_DMAST 0x20			
38	#define ECON1_CSUMEN 0x10			
39	#define ECON1_TXRTS 0x08			
40	#define ECON1_RXEN 0x04			
41	#define ECON1_BSEL1 0x02			
42	#define ECON1_BSEL0 0x01			
43	d_G000' declared			
44	00' declared, but			
45	100' declared, bu			
46	100' declared, bu			
47	G000' declared, l			
48	declared, but nev			
49				
50	// ENC28J60 ESTAT Register Bit Definitions			
51	#define ESTAT 0x1D			
52	#define ESTAT_INT 0x80			
53	#define ESTAT_LATECOL 0x10			
54	#define ESTAT_RXBUSY 0x04			
55	#define ESTAT_TXABRT 0x02			
56	#define ESTAT_CLKRDY 0x01			
57				
58	// Bank 0 registers			
59	#define ERDPT (0x00 0x00)			
60	#define EWRPT (0x02 0x00)			
61	#define ETXST (0x04 0x00)			
62	#define ETXND (0x06 0x00)			
63	#define ERXST (0x08 0x00)			
64	#define ERXND (0x0A 0x00)			
65	#define ERXRDPT (0x0C 0x00)			
66	#define ERXWRPT (0x0E 0x00)			
67				
68	//khai bao nguyen mau ham			
69	void SPI_SendByte(uint8_t _byte);			
70	uint8_t SPI_ReceiveByte(void);			
71	uint8_t SPI_ReceiveByte(void);			
72	void ENC28J60_write_command(uint8_t op,uint8_t adres, uint8_t data);			
73	static uint8_t enc28j60_read_command(uint8_t op,uint8_t adres);			
74	static void ENC28J60_SetBank(uint8_t adres);			
75	static void ENC28J60_writeByte(uint8_t adres,uint8_t data);			
76	static uint8_t ENC28J60_readByte(uint8_t adres);			
	static void ENC28J60_writeByte16(uint8_t adres,uint16_t data);			
	void ENC29J600_ini(void);			
	static void ENC28J60_error (void);			

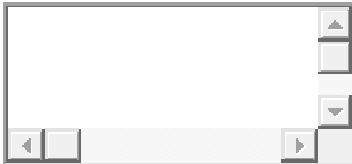
	#endif /* ENC28J60_H_ */			
	76			

Rồi, bỏ chương trình cấu hình buffer vào chỗ **else** của hàm **ENC29J600_ini** nào !



```
1 //cau hinh khich thuoc bo dem truyen nhan  
2 ENC28J60_writeByte16(ERXST,RXSTART_INIT);  
3 ENC28J60_writeByte16(ERXND,RXSTOP_INIT);  
4  
5 ENC28J60_writeByte16(ETXST,TXSTART_INIT);  
6 ENC28J60_writeByte16(ETXND,TXSTOP_INIT);
```

Đồng thời mình cũng reset con trỏ của bộ đệm truyền nhận về vị trí Start luôn nhé



```
1 //reset con tro RX TX ve vi tri start  
2 ENC28J60_writeByte16(ERXRDPT,RXSTART_INIT);  
3 ENC28J60_writeByte16(ERXWRPT,TXSTART_INIT);
```

Tiếp tục

Tiếp tục quá trình khởi tạo, ở mục 6.3 của **datasheet**, hắn có nói chúng ta nên cấu hình thanh ghi **ERXFCON**. OK thôi, xem thanh ghi này có gì nào...

REGISTER 8-1: ERXFCON: 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

bit 7 UCEN: Unicast Filter Enable bit

When ANDOR = 1:

1 = Packets not having a destination address matching the local MAC address will be discarded

0 = Filter disabled

When ANDOR = 0:

1 = Packets with a destination address matching the local MAC address will be accepted

0 = Filter disabled

bit 6 ANDOR: AND/OR Filter Select bit

1 = AND: Packets will be rejected unless all enabled filters accept the packet

0 = OR: Packets will be accepted unless all enabled filters reject the packet

bit 5 CRCEN: Post-Filter CRC Check Enable bit

1 = All packets with an invalid CRC will be discarded

0 = The CRC validity will be ignored

bit 4 PMEN: Pattern Match Filter Enable bit

When ANDOR = 1:

1 = Packets must meet the pattern match criteria or they will be discarded

0 = Filter disabled

When ANDOR = 0:

1 = Packets which meet the pattern match criteria will be accepted

0 = Filter disabled

bit 3 MPEN: Magic Packet Filter Enable bit

When ANDOR = 1:

1 = Packets must be Magic Packets for the local MAC address or they will be discarded

0 = Filter disabled

When ANDOR = 0:

1 = Magic Packets for the local MAC address will be accepted

0 = Filter disabled

bit 2 HTEN: Hash Table Filter Enable bit

When ANDOR = 1:

1 = Packets must meet the hash table criteria or they will be discarded

0 = Filter disabled

When ANDOR = 0:

1 = Packets which meet the hash table criteria will be accepted

0 = Filter disabled

bit 1 MCEN: Multicast Filter Enable bit

When ANDOR = 1:

1 = Packets must have the Least Significant bit set in the destination address or they will be discarded

0 = Filter disabled

When ANDOR = 0:

1 = Packets which have the Least Significant bit set in the destination address will be accepted

0 = Filter disabled

bit 0 BCEN: Broadcast Filter Enable bit

When ANDOR = 1:

1 = Packets must have a destination address of FF-FF-FF-FF-FF-FF or they will be discarded

0 = Filter disabled

When ANDOR = 0:

1 = Packets which have a destination address of FF-FF-FF-FF-FF-FF will be accepted

0 = Filter disabled

Thanh ghi này có chưa các bit điều khiển chức năng lọc các gói tin, đây là chức năng nâng cao, tạm thời mình sẽ không xài chức năng này, mình sẽ code nhưng để comment dòng lệnh này đi, chưa cần dùng vôi, sau này sẽ tính đến nó sau !



1//rx buffer filters

```
2 //ENC28J60_writeByte(ERXFCON,ERXFCON_UCEN|ERXFCON_ANDOR|ERXFCON_CRCEN); //set 3 bit
```

Cũng đừng quên thêm các #define cho thanh ghi này vào file .h nhé



```
1 // ENC28J60 ERXFCON Register Bit Definitions  
2 #define ERXFCON      (0x18|0x20) //boi vi no thuoc bank1 nen can or them 0x20  
3 #define ERXFCON_UCEN  0x80  
4 #define ERXFCON_ANDOR 0x40  
5 #define ERXFCON_CRCEN 0x20  
6 #define ERXFCON_PMEN   0x10  
7 #define ERXFCON_MPEN   0x08  
8 #define ERXFCON_HTEN   0x04  
9 #define ERXFCON_MCEN   0x02  
10#define ERXFCON_BCEN   0x01
```

Tiếp tục ở mục 6.4 của **datasheet** thì mục này chúng ta đã code ngay từ đâu lun rồi

Tiếp tục tới mục 6.5, nó có tên là **MAC Initialization Settings** tức là cấu hình địa chỉ MAC

Cấu hình địa chỉ MAC

Địa chỉ **MAC** là cái quái gì ? Nó là địa chỉ vật lý của 1 thiết bị mạng, dùng để phân biệt với các thiết bị khác trong mạng ! Nó là 1 địa chỉ cố định được gán vào với nhà sản xuất (tức chúng ta)

Địa chỉ MAC là một bộ sáu cặp hai ký tự, cách nhau bằng dấu hai chấm. Ví dụ **00:1B:44:11:3A:B7** là một địa chỉ MAC

Địa chỉ **MAC** khác địa chỉ **IP** chỗ nào

Địa chỉ IP được sử dụng để truyền dữ liệu từ mạng này sang mạng khác. Địa chỉ MAC được sử dụng để phân phối dữ liệu đến đúng thiết bị trên mạng.

Địa chỉ IP giống như địa chỉ nhà, còn địa chỉ MAC là tên người nhận

ENC28J60 là 1 thiết bị mạng, nó cũng cần có 1 địa chỉ MAC, và chính chúng ta là người gán địa chỉ MAC cho nó !

Các bước để cấu hình địa chỉ MAC **datasheet** 6.5 đã nói rất rõ, các bạn có thể tự đọc thêm nếu muốn nhé:

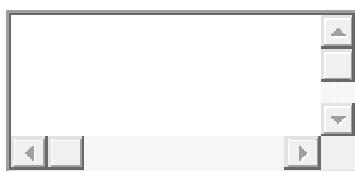
Đầu tiên là set các bit **MARXEN TXPAUS RXPAUS PASSALL** trong thanh ghi **MACON1**. Rồi clear thanh ghi **MACON2**.

Rồi set các bit **PADCFG0 TXCRCEN FRMLNEN** trong thanh ghi **MACON3**. Rồi set giá trị 0xC12 cho thanh ghi **MAIPG**.

Thanh ghi **MABBIPG** cũng nên được set giá trị 0x12 như datasheet khuyên.

Tương tự với thanh ghi **MAMXFL** cũng nên được cấu hình = 1518 để lại

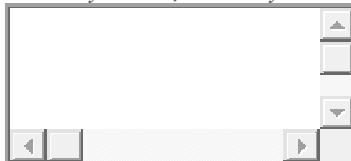
Không làm man nữa, mình sẽ đi vào code luôn, trước tiên phải thêm các #define cho các thanh ghi đã đúng không :)



```
1 // Bank 2 registers
2 #define MACON1      (0x00|0x40|0x80) //boi vi la bank 2 nen OR voi 0x40
3 #define MACON2      (0x01|0x40|0x80)
4 #define MACON3      (0x02|0x40|0x80)
5 #define MACON4      (0x03|0x40|0x80)
6 #define MABBIPG     (0x04|0x40|0x80)
7 #define MAIPG       (0x06|0x40|0x80)
8 #define MAMXFL      (0x0A|0x40|0x80)
9 #define MIREGADR    (0x14|0x40|0x80)
10#define MIWR        (0x16|0x40|0x80)
11// -- cau hinh vi tri cac bit trong MACON1---/
12#define MACON1_LOOPBK 0x10
13#define MACON1_TXPAUS 0x08
14#define MACON1_RXPAUS 0x04
15#define MACON1_PASSALL 0x02
16#define MACON1_MARXEN 0x01
17// -- cau hinh vi tri cac bit trong MACON3---/
18#define MACON3_PADCFG2 0x80
19#define MACON3_PADCFG1 0x40
20#define MACON3_PADCFG0 0x20
21#define MACON3_TXCRCEN 0x10
```

```
22#define MACON3_PHDRLEN 0x08  
23#define MACON3_HFRMLEN 0x04  
24#define MACON3_FRMLNEN 0x02  
25#define MACON3_FULDPX 0x01
```

Nếu để ý các bạn sẽ thấy các thanh ghi ở bank2 mình còn OR thêm 0x80, để làm gì lát mình sẽ nói

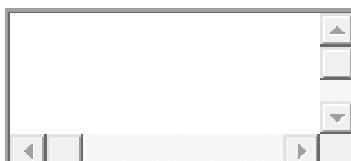


```
1 //cau hinh MAC  
2 ENC28J60_writeByte(MACON1,MACON1_MARXEN|MACON1_TXPAUS|MACON1_RXPAUS);  
3 ENC28J60_writeByte(MACON2,0x00);  
4 ENC28J60_write_command(ENC28J60_BIT_FIELD_SET,MACON3,MACON3_PADCFG0|MACON3_TXCRCEN|MACO  
N3_FRMLNEN);  
5 ENC28J60_writeByte16(MAIPG,0x0C12);  
6 ENC28J60_writeByte(MABBIPG,0x12);  
7 ENC28J60_writeByte(MAMXFL,1500);
```

Điền địa chỉ MAC

Chúng ta sẽ điền địa chỉ MAC cho module vào 6 thanh ghi từ **MMADR0** đến **MMADR5**

Tiếp tục #define 5 địa chỉ này vào file .h nhé, ngoài ra mình cũng #define thêm địa chỉ MAC của module vào



```
1// Bank 3 registers  
2#define MAADDR1      (0x00|0x60|0x80)  
3#define MAADDR0      (0x01|0x60|0x80)  
4#define MAADDR3      (0x02|0x60|0x80)  
5#define MAADDR2      (0x03|0x60|0x80)  
6#define MAADDR5      (0x04|0x60|0x80)  
7#define MAADDR4      (0x05|0x60|0x80)  
8#define MISTAT       (0xA|0x60|0x80)
```

Nếu để ý các bạn sẽ thấy các thanh ghi ở bank3 mình còn OR thêm 0x80, để làm gì lát mình sẽ nói

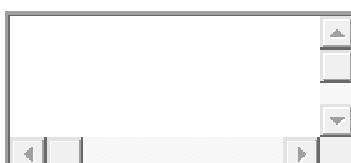
OK, giờ mình sẽ thêm code để điền mac vào thanh ghi chứa MAC nhé

Trước hàm **ENC29J600_ini** mình khai báo thêm 1 mảng **macaddr[6]** để lưu địa chỉ MAC mà mình thiết lập cho chip !



```
1 uint8_t macaddr [6] = {0x00,0x00,0x20,0x07,0x19,0x98};
```

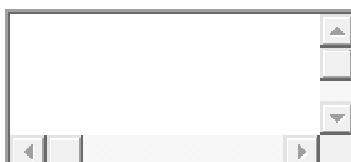
Tiếp tục code khởi tạo



```
1 ENC28J60_writeByte(MAADDR5,macaddr[0]); //ghi dia chi MAC vao chip
2 ENC28J60_writeByte(MAADDR4,macaddr[1]);
3 ENC28J60_writeByte(MAADDR3,macaddr[2]);
4 ENC28J60_writeByte(MAADDR2,macaddr[3]);
5 ENC28J60_writeByte(MAADDR1,macaddr[4]);
6 ENC28J60_writeByte(MAADDR0,macaddr[5]);
```

Để chắc chắn đã ghi MAC vào đúng, mình thử đọc ra xem thế nào nhé !

Mình sẽ khởi tạo thêm mảng **mymac[6]** để lưu địa chỉ mac đọc ra ! Đồng thời, khởi tạo thêm mảng **debug_string[60]** để phục vụ việc in ra màn serial debug



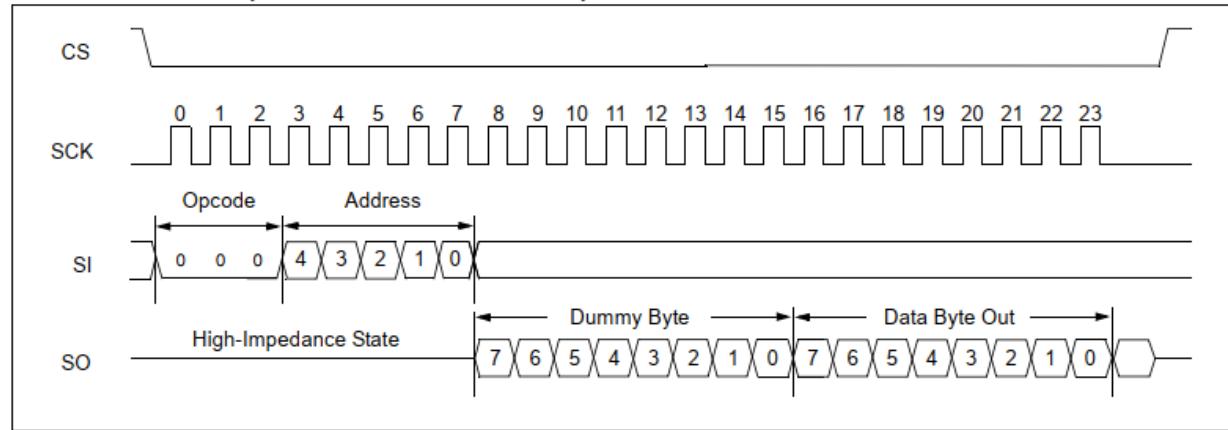
```
1 uint8_t mymac[6];
2 uint8_t debug_string[60]
```

Tiếp tục thêm code vô hàm khởi tạo

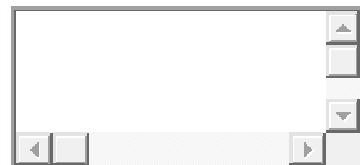
Sửa lại hàm **ENC28J60_read_command**

Chúng ta sẽ đọc địa chỉ MAC đã set bằng hàm **ENC28J60_read_command**, tuy nhiên hãy nhìn lại mục 4.2.1 trong **datasheet**

FIGURE 4-4: READ CONTROL REGISTER COMMAND SEQUENCE (MAC AND MII REGISTERS)



Khi đọc các thanh ghi liên quan đến MAC và MII thì byte đầu tiên đọc ra sẽ là byte dummy (byte rác), byte thứ hai mới đúng là data ta cần. Do vậy, hàm [ENC28J60_read_command](#) mình sẽ sửa lại như sau:



```

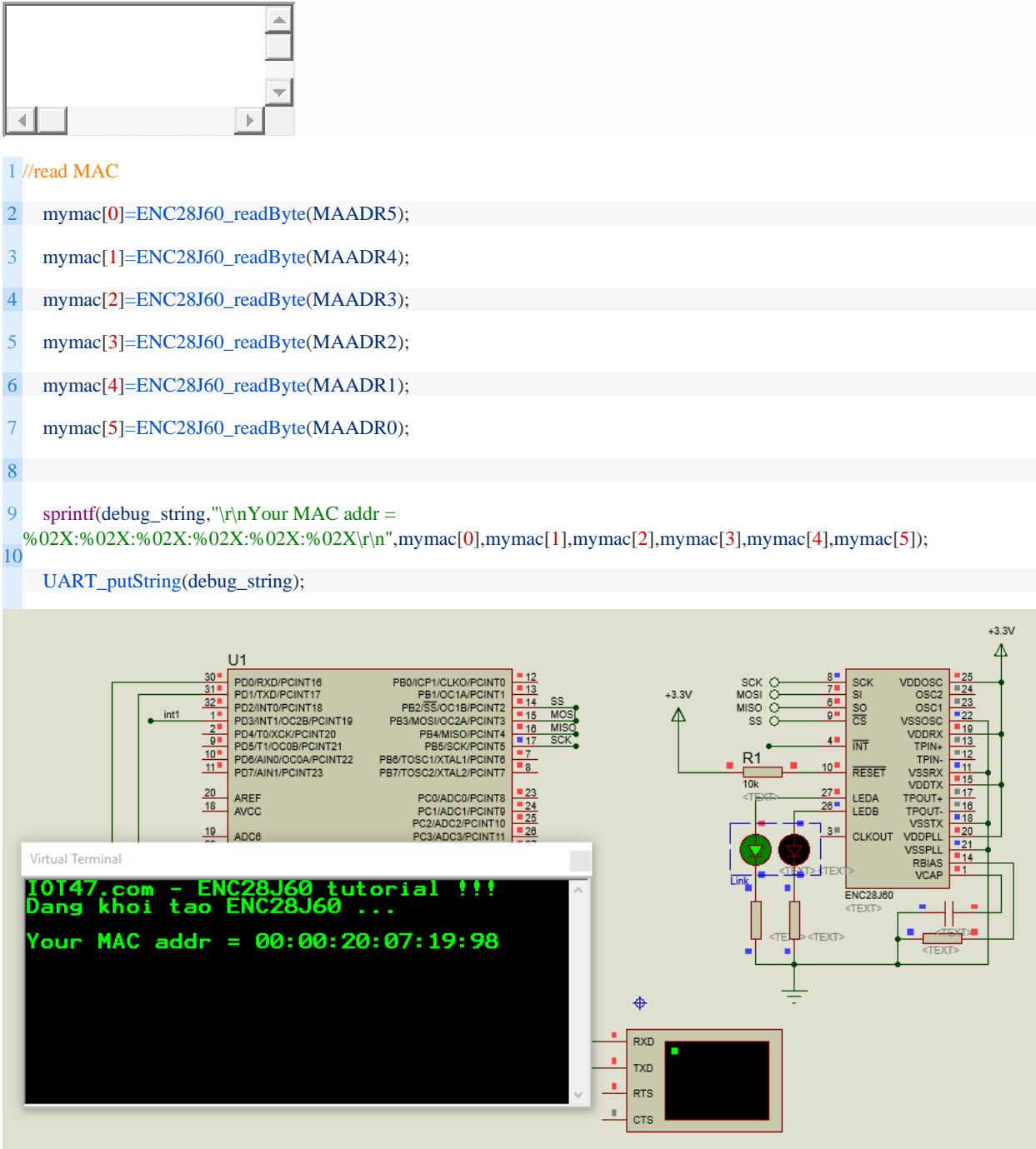
1 static uint8_t ENC28J60_read_command(uint8_t op,uint8_t address)
2 {
3     uint8_t result;
4     SS_SELECT();
5     SPI_SendByte(op|(address&0x1F));
6     if(address & 0x80) SPI_ReceiveByte();
7     result=SPI_ReceiveByte();
8     SS_DESELECT();
9     return result;
10}

```

Mình đã thêm vào dòng lệnh kiểm tra bit cao nhất của byte địa chỉ có được set không, nếu được set thì sẽ gọi hàm đọc nhưng không lấy kết quả vì đó là byte rác, lần đọc thứ 2 mình mới lấy ! Đó là lí do vì sao khi #define các byte địa chỉ có liên quan đến MAC ở trên mình có OR nó thêm với 0x80

Do các byte địa chỉ chỉ xài có 5bit thấp, 2 bit tiếp dùng để xác định bank, nên bit cuối (bit cao nhất) sẽ dùng để xác định byte này có thuộc loại MAC hay MII không để tránh byte rác (không hiểu mấy ông design con chip này tính kiểu gì luôn 😐)

Rồi, giờ đọc địa chỉ MAC đã ghi xem có đúng với ban đầu không !



Kết quả : chúng ta đã ghi thành công địa chỉ MAC

Nhu vậy, qua trình khởi tạo chip cơ bản là đã xong 70% !

Trong bài tiếp, chúng ta sẽ tiếp tục viết các hàm để làm việc với các thanh ghi thuộc lớp vật lí và hoàn thành hàm khởi tạo chip !

[ENC28J60] Bài 4: Khởi tạo module enc28j60 (phần 3) – lập trình enc28j60

1 Tháng Tư, 2020 Đào Nguyễn Ethernet, IoT tutorial 2



Tiếp tục với hàm khởi tạo chip ở [bài trước](#) trong chuỗi bài lập trình enc28j60, ở bài này chúng ta sẽ cấu hình và khởi tạo cho các thanh ghi thuộc nhóm **PHY**

Thông tin về các **PHY Registers** nằm ở phần 3.3 [datasheet](#)

Các thanh ghi thuộc **PHY** đều là loại 16bit và **không thể trực tiếp truy cập qua giao tiếp SPI**, các truy cập vào **PHY** đều gián tiếp thông qua các thanh ghi được gọi là **MII**

READING PHY REGISTERS

Để đọc 1 thanh ghi **PHY**

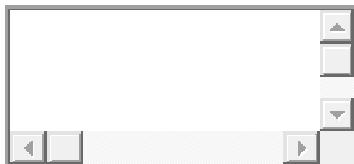
1. Ghi địa chỉ của thanh ghi đó vào thanh ghi **MIREGADR**
2. Set bit **MICMD.MIIRD** để bắt đầu thao tác đọc, bit **MISTAT.BUSY** (bit báo bận) sẽ được set
3. Đợi 10.24uS, thăm dò bit **MISTAT.BUSY** để chắc chắn hoạt động đọc hoàn thành
4. Khi hoàn thành, bit **MISTAT.BUSY** sẽ tự xóa, chúng ta xóa thủ công bit **MICMD.MIIRD** và lấy giá trị mong muốn từ thanh ghi **MIRDL** và **MIRDH**

WRITING PHY REGISTERS

Để ghi 1 thanh **PHY**

1. Ghi địa chỉ của thanh ghi đó vào thanh ghi **MIREGADR**
2. Ghi giá trị 8bit thấp vào **MIWRL**
3. Ghi giá trị 8bit cao vào **MIWRH**
4. Ngay sau đó **MISTAT.BUSY** tự động được set
5. Sau khoảng 10.24uS, quá trình ghi hoàn tất, bit bận sẽ tự xóa

Khá đơn giản phải không. Viết 2 hàm ghi và đọc **PHY** thôi nào 😊



```
1 static void ENC28J60_writePhy(uint8_t adres,uint16_t data) //ham ghi PHY
2 {
3     ENC28J60_writeByte(MIREGADR, adres);
4     ENC28J60_writeByte16(MIWR, data);
5     while(ENC28J60_readByte(MISTAT)&MISTAT_BUSY); //cho het ban
6 }
7 static uint16_t ENC28J60_readPhy(uint8_t adres) //ham doc PHY
8 {
9     uint16_t value;
10    ENC28J60_writeByte(MIREGADR, adres);
11    ENC28J60_writeByte(MICMD, MICMD_MIIRD); //set bit MICMD_MIIRD bat dau doc
12    while(ENC28J60_readByte(MISTAT)&MISTAT_BUSY); //cho het ban
13    ENC28J60_writeByte(MICMD,0x00); //clear bit MICMD_MIIRD
14    value = ENC28J60_readByte(MIRDL) + ((uint16_t)ENC28J60_readByte(MIRDH)<<8); //doc gia tri trong 2 thanh gho
15    MIWR
16    return value; //tra ve gia tri
17 }
```

Đừng quên thêm nguyên mẫu hàm vào file .h và thêm cả #define cho các thanh ghi **PHY** và **MI** nhé !



```
1//-----
2//MII read registers and bit
3#define MICMD          (0x12|0x40|0x80) //thanh ghi nay thuoc bank 2 nen OR 0x40
4#define MICMD_MIRD    0x01
5#define MIRDL          (0x18|0x40|0x80)
6#define MIRDH          (0x19|0x40|0x80)
7#define MISTAT_BUSY    0x01
```

Bây giờ cấu hình lớp vật lý **PHY** trong hàm khởi tạo

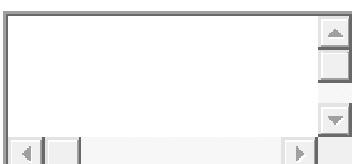
Quay trở lại mục 6.6 trong **datasheet** để xem nhung thứ cần khởi tạo nhé !

Dưới đây là 1 số khởi tạo thanh ghi **PHY** theo datasheet yêu cầu, mình cũng set các bit trong thanh ghi **EIE** để cho phép ngắt



```
1 //*****Advanced Initialisations*****
2 ENC28J60_writePhy(PHCON2,PHCON2_HDLDIS);
3 ENC28J60_writePhy(PHLCON,PHLCON_LACFG2|PHLCON_LBCFG2|PHLCON_LBCFG1|PHLCON_LBCFG0|PHLCO
N_LFRQ0|PHLCON_STRCH);
4 ENC28J60_SetBank (ECON1);
5 ENC28J60_write_command (ENC28J60_BIT_FIELD_SET, EIE, EIE_INTIE | EIE_PKTIE);
6 ENC28J60_writeByte(ECOCON,0x02);
7 delay_us(15);
8 ENC28J60_write_command (ENC28J60_BIT_FIELD_SET, ECON1, ECON1_RXEN); // cho phép nhan gói
```

Và thêm #define địa chỉ cho các thanh ghi và bit **PHY** và 1 số bit trong thanh ghi **EIE**



```
1 #define EIE      0x1B
2 // ENC28J60 EIE Register Bit Definitions
3 #define EIE_INTIE    0x80
4 #define EIE_PKTIE     0x40
5 #define EIE_DMAIE     0x20
6 #define EIE_LINKIE    0x10
7 #define EIE_TXIE       0x08
8 #define EIE_WOLIE      0x04
9 #define EIE_TXERIE     0x02
10#define EIE_RXERIE     0x01
11
12#define ECOCON   (0x15|0x60) //thanh ghi nay thuoc bank 3 nen OR 0x60
13
14// PHY registers
15#define PHCON1     0x00
16#define PHSTAT1    0x01
17#define PHHID1     0x02
18#define PHHID2     0x03
19#define PHCON2     0x10
20#define PHSTAT2    0x11
21#define PHIE       0x12
22#define PHIR       0x13
23#define PHLCON     0x14
24//-----
25#define PHCON2_HDLDIS 0x0100
26//-----
27// PHLCON
28#define PHLCON_LACFG3 0x0800
29#define PHLCON_LACFG2 0x0400
30#define PHLCON_LACFG1 0x0200
31#define PHLCON_LACFG0 0x0100
32#define PHLCON_LBCFG3 0x0080
33#define PHLCON_LBCFG2 0x0040
```

```
34#define PHLCON_LBCFG1 0x0020
35#define PHLCON_LBCFG0 0x0010
36#define PHLCON_LFRQ1 0x0008
37#define PHLCON_LFRQ0 0x0004
38#define PHLCON_STRCH 0x0002
```

Đối với 2 đèn LED trên cổng RJ45, nó cũng có hàn 1 thanh ghi **PHY**, nhưng mình sẽ mặc định và không cấu hình thêm

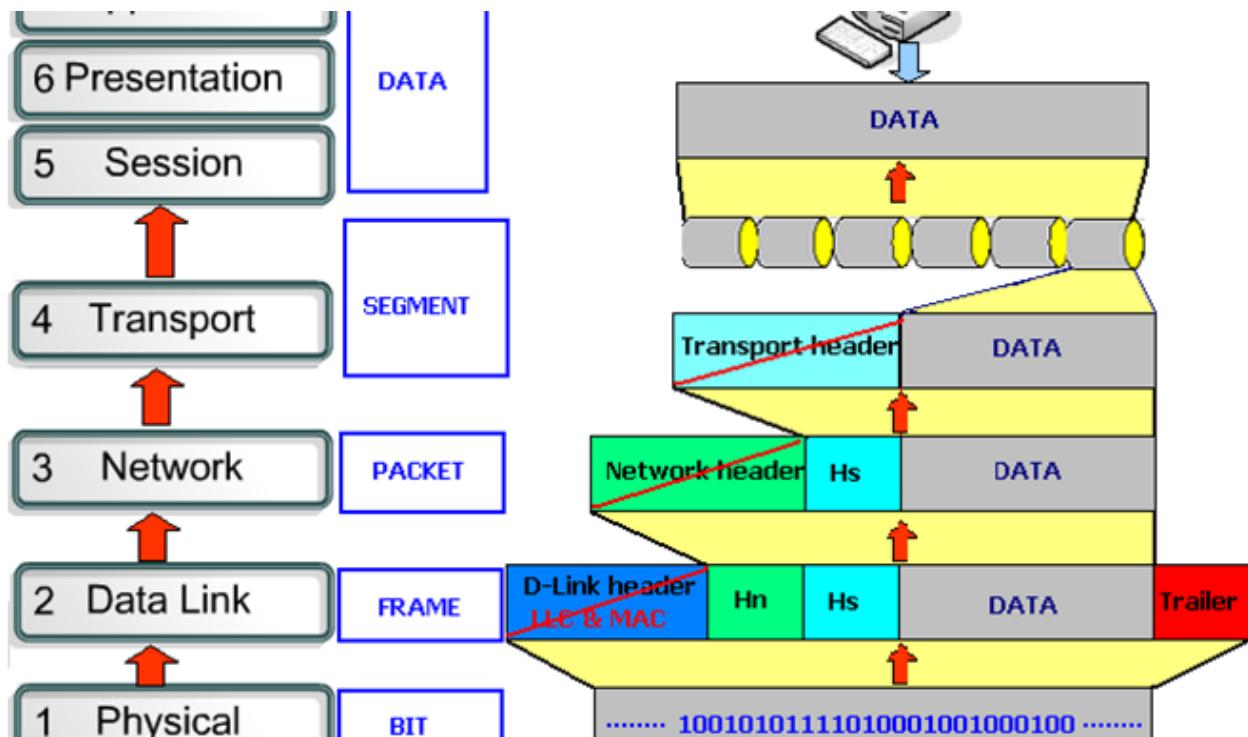
Vậy là quá trình khởi tạo cho module lúc khởi động đã hoàn tất !

Các bạn có thể tải source cho bài này [tại đây](#)

Trong bài [tiếp theo](#) – lập trình enc28j60, chúng ta sẽ làm việc với bộ đệm TX RX và viết các hàm để ghi đọc các gói tin trên bộ đệm !

[ENC28J60] Bài 5: Đọc ghi các gói tin trong bộ đệm – lập trình enc28j60

2 Tháng Tư, 2020 Đào Nguyễn Ethernet, IoT tutorial 2



Theo dõi toàn bộ tutorial lập trình enc28j60 [tại đây](#)

Viết chức năng đọc các gói tin (đọc bộ đệm ethernet)

Bây giờ chúng ta sẽ học cách nhận các gói, vì toàn bộ trao đổi trên mạng cục bộ được chia thành các gói.

Việc nhận các gói trong chip xảy ra thông qua **bộ đệm vòng**.

Chúng ta hãy xem một ví dụ nhận gói trong hình

FIGURE 7-3: SAMPLE RECEIVE PACKET LAYOUT

	<u>Address</u>	<u>Memory</u>	<u>Description</u>
Packet N – 1	101Fh		End of the Previous Packet
	1020h	0Eh	Low Byte
	1021h	10h	High Byte
	1022h	rsv[7:0]	
	1023h	rsv[15:8]	
	1024h	rsv[23:16]	
	1025h	rsv[30:24]	
	1026h	data[1]	
	1027h	data[2]	
Packet N	1059h	data[m-3]	Packet Data: Destination Address, Source Address, Type/Length, Data, Padding, CRC
	105Ah	data[m-2]	
	105Bh	data[m-1]	
	105Ch	data[m]	
	105Dh	crc[31:24]	
	105Eh	crc[23:16]	
		crc[15:8]	
		crc[7:0]	
			Byte Skipped to Ensure Even Buffer Address
			Start of the Next Packet

Trong bộ đệm sẽ chứa rất nhiều gói tin, cứ bắt đầu mỗi gói sẽ là 1 con trỏ dài 2 byte lưu vị trí của gói tin tiếp theo, tiếp đến 4 byte lưu trạng thái nhận, tiếp đó là dữ liệu của gói tin, và 4 byte cuối sẽ lưu giá trị checksum

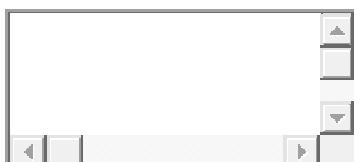
TABLE 7-3: RECEIVE STATUS VECTOR

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.

Trong trường vector status, 2 byte đầu tiên [0->15] chứa độ dài của gói tin, 2 byte tiếp chứa các trạng thái nhận liên quan đến gói đó, các bạn xem chú thích trên bảng nhé !

Thanh ghi EPKTCNT ở bank2 lưu số lượng gói đang nhận, vì vậy, chúng ta sẽ kiểm tra nó nếu > 0 tức là đang có gói tin trong bộ đệm, hãy đọc nó ra !

Giờ hãy tạo một hàm để đọc một gói trong tệp **enc28j60.c** và tạo một vài biến cục bộ để phục vụ tính toán và thêm 1 biến toàn cục để lưu vị trí cho gói tiếp theo (nhớ thêm nguyên mẫu hàm vào file .h)



```

1 uint16_t ENC28J60_read_packet(uint8_t *buf,uint16_t buflen)
2 {
3     uint16_t status;
4     uint16_t len=0;
5     if(ENC28J60_readByte(EPKTCNT)>0) // neu co goi tin
6     {
7

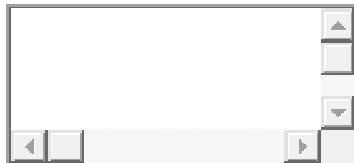
```

```

8 }
9 return len;
10}

```

Và tạo #define cho thanh ghi EPKTCNT



```

1// Bank 1 registers
2#define EPKTCNT      (0x19|0x20)

```

Mình sẽ tạo thêm 1 biến toàn cục tên là **NextPacketPtr**, biến này sẽ lưu vị trí của gói tin tiếp theo. Khi khởi động, gói tin sẽ bắt đầu ở vị trí **RXSTART_INIT** nên các bạn gán vị trí này cho biến **NextPacketPtr** ngay trong hàm khởi tạo luôn nhé !

```

uint8_t macaddr [6] = {0x00,0x00,0x20,0x07,0x19,0x98};
uint8_t mymac[6];
uint8_t debug_string[60];
static uint16_t NextPacketPtr; ←
void ENC28J60_ini(void)
{
    UART_putString("Dang khai tao ENC28J60 ... \r\n");
    ENC28J60_write_command(ENC28J60_SOFT_RESET,0x1F,0); //soft reset
    delay_ms(2);
    while(!ENC28J60_read_command(ENC28J60_READ_CTRL_REG,ESTAT) & ESTAT1
        if(ENC28J60_readByte(ERDPT) != 0xFA)ENC28J60_error(); //khai tao
    else
    {
        NextPacketPtr=RXSTART_INIT; ←
        //cau hinh kich thuoc bo dem truyen nhan
        ENC28J60_writeByte16(ERXST,RXSTART_INIT);
        ENC28J60_writeByte16(ERXND,RXSTOP_INIT);
    }
}

```

Tiếp tục quay trở lại hàm **ENC28J60_read_packet**, khi kiểm tra và thấy có gói tin mới, chúng ta sẽ đọc nó ra theo các bước sau:

1. Đặt con trỏ đọc ERDPT vào vị trí bắt đầu của gói tin
2. Đọc 2 byte đầu tiên trong bộ đệm, đây sẽ là vị trí bắt đầu của gói tin tiếp theo -> nó cũng chính là vị trí kết thúc của gói tin hiện tại
3. Đọc 2 byte tiếp theo trong bộ đệm, nó chính là độ dài của gói tin [Status vector 0:15]
4. Đọc 2 byte tiếp theo trong bộ đệm, nó chứa thông tin trạng thái nhận [Status vector 16:31]
5. Bắt đầu từ bây giờ chính là dữ liệu của gói, các bạn đọc từ vị trí này tới hết độ dài của gói tin nhé, chúng ta có thể trừ đi 4 byte checksum ở cuối nếu không cần kiểm tra lại checksum

6. Set **ERXRDP** tới vị trí của gói tin tiếp theo

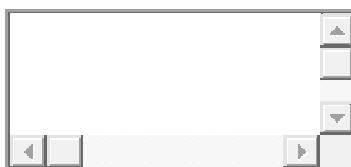
7. Sau khi đọc xong chúng ta set bit **PKTDEC** trong thanh ghi **ECON2** để **EPKTCNT** giảm đi 1

Để đọc buffer, các bạn phải gửi command ở nhóm lệnh có Opcode là **Read Buffer Memory** = 0x3A. Bởi vậy hãy thêm #define cho nó nhé



```
1#define ENC28J60_READ_BUF_MEM    0x3A
```

Minh sẽ tạo thêm 1 hàm để đọc 1 byte từ buffer cho tiện hen (nhớ thêm nguyên mẫu hàm cho file .h)



```
1uint8_t ENC28J60_read_Byte_Buffer(void)
2{
3    return ENC28J60_read_command(ENC28J60_READ_BUF_MEM,0);
4}
```

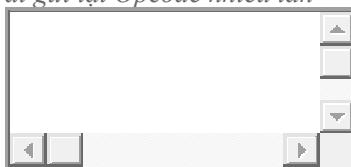
Do nhóm lệnh **Read Buffer Memory** nó nhận data vào là n/a nên chúng ta nạp vào là gì cũng được. Ở đây mình gửi value=0 luôn.

Đọc 1 byte thì đương nhiên sẽ có đọc nhiều byte, thay vì gọi hàm **ENC28J60_read_Byte_Buffer** liên tục sẽ chậm (do gửi 2 byte để đọc về 1 byte), chip hỗ trợ chúng ta 1 cách đọc nhanh hơn nhiều, *không cần phải gửi Opcode luôn*

Cách đọc 1 khối data lớn trong **Read Buffer Memory**

1. CS Enable
2. Gửi Opcode **Read Buffer Memory**
3. Liên tục đọc data ra qua giao thức SPI
4. CS Disable

Bí quyết nằm ở chỗ chúng ta không tắt chân CS nên chip sẽ không resetOpcode đã gửi, do vậy không cần gửi đi lại Opcode nhiều lần



```
1 void ENC28J60_ReadBuffer(uint16_t len, uint8_t* data)
```

```
2 {
```

```

3 SS_SELECT();
4 SPI_SendByte(ENC28J60_READ_BUF_MEM); //gui Opcode
5 while(len)
6 {
7     len--;
8     // read data
9     *data = (uint8_t)SPI_ReceiveByte();
10    data++;
11 }
12 *data='\0'; //set end buffer
13 SS_DESELECT();
14}

```

Được rồi, giờ viết nội dung cho hàm **ENC28J60_read_packet** nhé !



```

1 //Thiet lap con tro de doc du lieu tu goi tin nhan duoc
2 ENC28J60_writeByte16(ERDPT,NextPacketPtr);
3
4 //Doc gia tri con tro cua goi tin tiep theo
5 NextPacketPtr=ENC28J60_read_Byte_Buffer();
6 NextPacketPtr|=ENC28J60_read_Byte_Buffer()<8;
7
8 // Doc kich thuoc cua goi tin
9 len = ENC28J60_read_Byte_Buffer();
10    len |= ENC28J60_read_Byte_Buffer()<<8;
11
12 len-=4; // xoa 4byte checksum o cuoi
13 if(len>buflen) len=buflen;
14
15 //Doc trang thai cua bo nhan

```

```

16 status = ENC28J60_read_Byte_Buffer();
17 status |= ENC28J60_read_Byte_Buffer()<<8;
18
19 if ((status & 0x80)==0)len=0; //kiem tra bit Received Ok, neu err thi khong doc goi nay
20 else
21 {
22     //doc du lieu trong bo dem
23     ENC28J60_ReadBuffer(len, buf);
24 }
25
26 // Chuyen con tro du lieu nhan toi phan dau cua goi tin tiep theo.
27 if(NextPacketPtr-1>RXSTOP_INIT)ENC28J60_writeByte16(ERXRDPT,RXSTOP_INIT);
28 else ENC28J60_writeByte16(ERXRDPT,NextPacketPtr);
29 ENC28J60_write_command(ENC28J60_BIT_FIELD_SET,ECON2,ECON2_PKTDEC);

```

Nhớ #define địa chỉ cho thanh ghi ECON2



```

1// ENC28J60 ECON2 Register Bit Definitions
2#define ECON2      0x1E
3#define ECON2_PKTDEC 0x40

```

Viết chức năng gửi các gói tin (ghi bộ đệm ethernet)

Chương trình gửi gói cũng tương tự như đọc

FIGURE 7-2: SAMPLE TRANSMIT PACKET LAYOUT

Buffer Pointers	Address	Memory	Description
ETXST = 0120h	0120h	0Eh	Control
	0121h	data[1]	PHUGEEN, PPADEN, PCRCEN and POVERRIDE
	0122h	data[2]	
			Destination Address, Source Address, Type/Length and Data
ETXND = 0156h	0156h	data[m]	
	0157h	tsv[7:0]	
	0158h	tsv[15:8]	
	0159h	tsv[23:16]	
	016Ah	tsv[31:24]	
	016Bh	tsv[39:32]	
	016Ch	tsv[47:40]	
	016Dh	tsv[51:48]	
	016Eh		Status Vector Written by the Hardware
			Start of the Next Packet

Hàm gửi buffer ra cho chip ENC28J60



```

1 static void ENC28J60_writeBuf(uint16_t len,uint8_t* data)
2{
3     SS_SELECT();
4     SPI_SendByte(ENC28J60_WRITE_BUF_MEM);
5     while(len--)
6         SPI_SendByte(*data++);
7     SS_DESELECT();
8 }
```

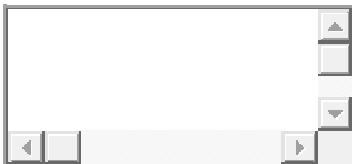
Thêm nguyên mẫu hàm và #define cho Opcode **ENC28J60_WRITE_BUF_MEM**



```
1#define ENC28J60_WRITE_BUF_MEM    0x7A
```

Các bước thiết lập để truyền gói tin – lập trình enc28j60 tutorial

1. Kiểm tra bit **TXRTS** trong thanh ghi **ECON1**, nếu nó đang được xóa thì chip sẵn sàng để truyền
2. Cài thanh ghi **EWRPT** (16bit) (con trỏ truyền) địa chỉ bắt đầu lưu bộ đệm truyền
3. Cài thanh ghi **ETXND** (16bit) địa chỉ của byte cuối cùng trong bộ đệm
4. Mở đầu bằng cách truyền byte control=0x00 vào bộ đệm
5. Truyền gói tin sang cho chip
6. Set bit **TXRTS** trong thanh ghi lên 1 để chip ENC28J60 bắt đầu truyền gói tin lên mạng



```
1 void ENC28J60_send_packet(uint8_t *buf,uint16_t buflen)
2 {
3     //cho qua trinh truyen truoc do hoan tat
4     while(ENC28J60_readByte(ECON1)&ECON1_TXRTS);
5
6     //Thiet lap con tro bat dau khong gian bo dem truyen
7     ENC28J60_writeByte16(EWRPT,TXSTART_INIT);
8
9     //Thiet lap con tro toi vi tri cua byte cuoi cua goi tin
10    ENC28J60_writeByte16(ETXND,TXSTART_INIT+buflen);
11
12    //truyen byte dau tien la 0x00 vao bo dem
13    ENC28J60_write_command(ENC28J60_WRITE_BUF_MEM,0,0);
14
15    //truyen goi tin vao bo dem
16    ENC28J60_writeBuf(buflen,buf);
17
18    //set bit TXRTS cho enc28j60 truyen goi tin
```

```
19 ENC28J60_write_command(ENC28J60_BIT_FIELD_SET,ECON1,ECON1_TXRTS);  
20}
```

Thêm nguyên mẫu cho hàm và thêm #define cho thanh ghi **EIR** vào file .h



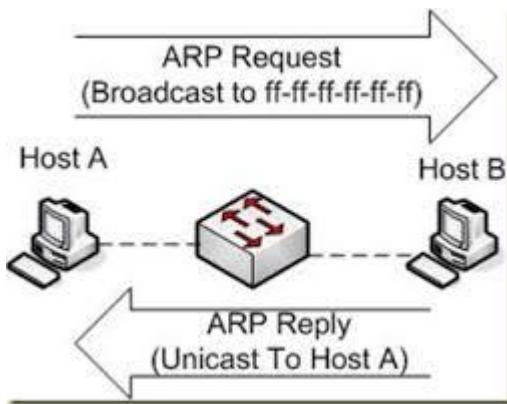
```
1// ENC28J60 EIR Register Bit Definitions  
2#define EIR      0x1C  
3#define EIR_PKTIF    0x40  
4#define EIR_DMAIF    0x20  
5#define EIR_LINKIF   0x10  
6#define EIR_TXIF     0x08  
7#define EIR_WOLIF    0x04  
8#define EIR_TXERIF   0x02  
9#define EIR_RXERIF   0x01
```

Như vậy, mình đã hướng dẫn các bạn viết xong các hàm cơ bản để giao tiếp với chip ENC28J60, bắt đầu từ bây giờ, chúng ta sẽ làm việc, mổ xé, bóc tách nội dung bên trong của các gói tin, xử lí các gói tin !

Các bạn có thể tải source của bài hôm nay [tại đây](#)

[ENC28J60] Bài 6: Giao thức ARP

3 Tháng Tư, 2020 Đào Nguyễn Ethernet, IoT tutorial 11



Như mình đã nói lần trước, để một gói tin được gửi tới chính xác thiết bị trong mạng LAN, nó cần có địa chỉ MAC của thiết bị đó. Giao thức ARP được sử dụng để từ 1 địa chỉ IP có thể tìm ra địa chỉ MAC của thiết bị đích hay nói cách khác là phân giải địa chỉ IP sang địa chỉ máy

Mình sẽ làm thử 1 demo gửi 1 gói tin ARP nhé !

Chúng ta mở project đã làm ở **bài 5**

Trong file code main, mình khai báo 1 gói tin arp như sau:



```
1 uint8_t arp[62]=
2 {
3     0xFF,0xFF,0xFF,0xFF,0xFF,0xFF, //board cat
4     0x00,0x00,0x20,0x07,0x19,0x98, //source MAC
5     0x08,0x06,          //Ethernet type - ARP
6     0x00,0x01,          //HTPYE
7     0x08,0x00,
8     0x06,0x04,
9     0x00,0x01,
10    0x00,0x00,0x20,0x07,0x19,0x98, //source MAC
```

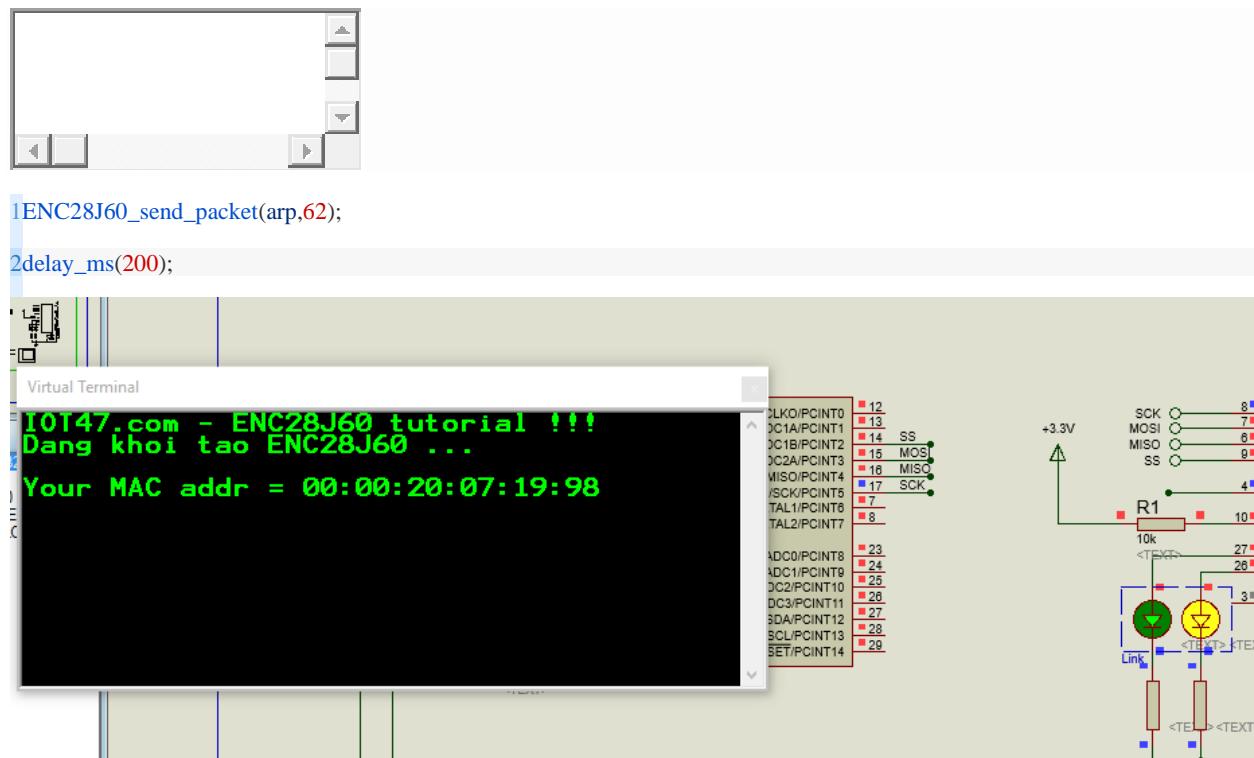
```

11 192,168,1,197,
12 0x00,0x00,0x00,0x00,0x00,0x00,
13 192,168,1,1,
14 0x00,0x00,0x00,0x00,0x00,0x00,
15 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
16};

```

Chú ý: Kiểm tra ip của bạn thuộc dạng 192.168.1.xxx hay 192.168.0.xxx để đổi 2 cái bên trên ip tương ứng nhé

Trong hàm main, vòng lặp while(1) mình sẽ gửi gói tin đi liên tục bằng hàm **ENC28J60_send_packet**



Chạy thử chương trình và mình thấy led vàng nhấp nháy liên tục, có nghĩa là đã có dữ liệu được gửi đi. Để có thể debug được dữ liệu này mình sẽ sử dụng 1 phần mềm chuyên bắt gói tin trên máy tính

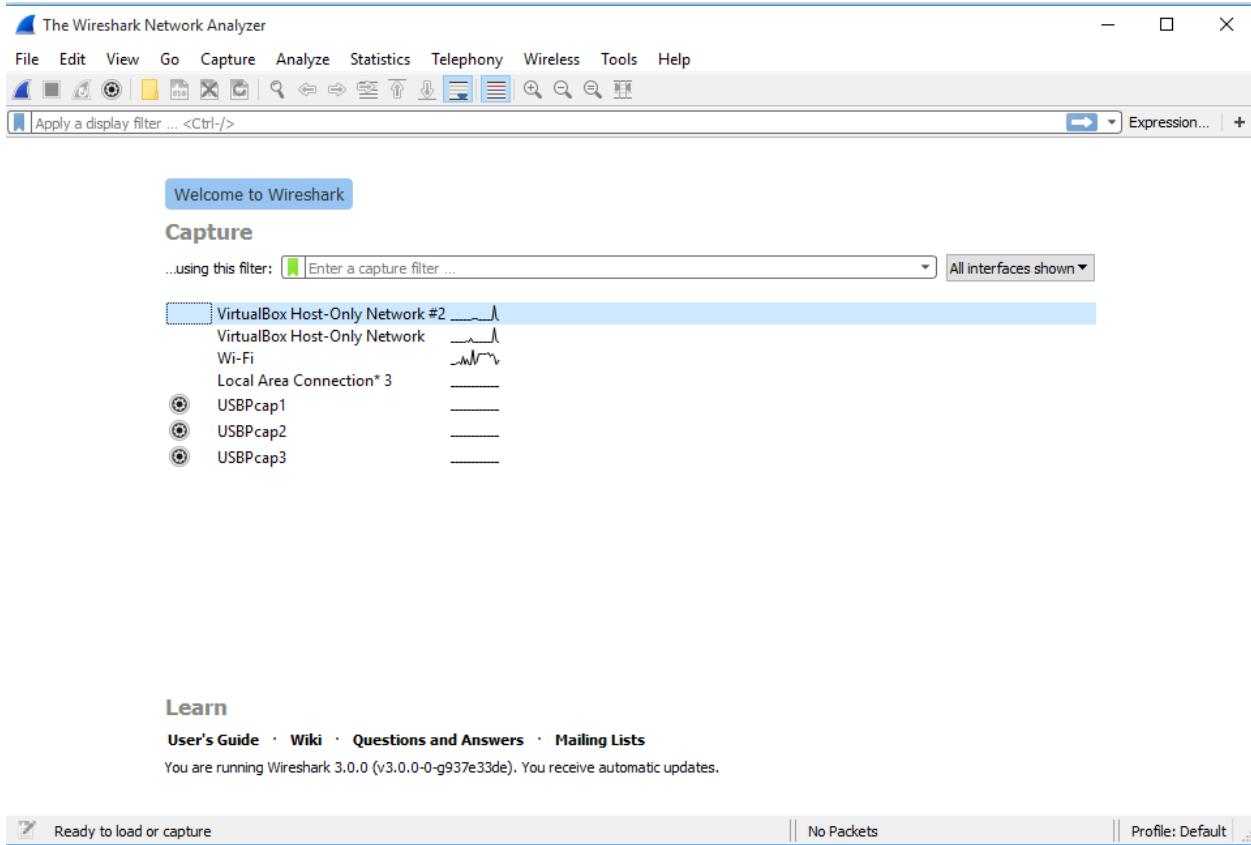
Giới thiệu phần mềm Wireshark

Wireshark là phần mềm chuyên bắt mọi gói dữ liệu ra vào máy tính thông qua Ethernet, wifi, usb. Do chúng ta sử dụng mô phỏng để chạy chíp ENC28J60 nên tất cả gói dữ liệu ra vào con chíp này cũng sẽ được phần mềm **Wireshark** bắt lại. Đây là công cụ mà hacker rất hay dùng

Chúng ta sẽ sử dụng phần mềm này để debug các gói tin trong suốt toàn bộ khóa học

Các bạn tải và cài đặt **Wireshark** nhé !

Sau khi cài phần mềm xong, mở lên, giao diện nó như vậy



Nếu bạn dùng wifi thì click chọn vào wifi, nếu dùng mạng dây thì chọn mạng dây tương ứng nhé !

Sau khi chọn wifi hoặc mạng dây thì màn hình real time sẽ hiện ra, ở đây sẽ liên tục hiện các gói tin vào ra máy tính theo thời gian thực

Capturing from Wi-Fi

No.	Time	Source	Destination	Protocol	Length	Info
93	3.118745	192.168.1.9	239.255.255.250	SSDP	167	M-SEARCH * HTTP/1.1
94	3.147511	172.217.161.130	192.168.1.15	TCP	54	443 → 52456 [ACK] Seq=544 Ack=202 Win=263 Len=0
95	3.148109	172.217.194.104	192.168.1.15	TLSv1.2	110	Application Data
96	3.148321	192.168.1.15	172.217.194.104	TCP	54	52367 → 443 [FIN, ACK] Seq=1 Ack=57 Win=64 Len=0
97	3.148326	192.168.1.15	172.217.194.104	TCP	54	[TCP Out-Of-Order] 52367 → 443 [FIN, ACK] Seq=1 Ack=5 Win=5 Len=0
98	3.200273	172.217.194.104	192.168.1.15	TCP	60	443 → 52367 [FIN, ACK] Seq=57 Ack=1 Win=255 Len=0
99	3.200315	192.168.1.15	172.217.194.104	TCP	54	52367 → 443 [ACK] Seq=2 Ack=58 Win=64 Len=0
100	3.200318	192.168.1.15	172.217.194.104	TCP	54	[TCP Dup ACK 99#1] 52367 → 443 [ACK] Seq=2 Ack=58 Win=0
101	3.217702	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
102	3.217706	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
103	3.236180	172.217.194.104	192.168.1.15	TCP	60	443 → 52367 [ACK] Seq=58 Ack=2 Win=255 Len=0
104	3.497002	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
105	3.497006	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
106	3.667971	192.168.1.9	224.0.0.251	MDNS	103	Standard query 0x004c PTR _googlecast._tcp.local, "QM...
107	3.770869	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
108	3.770874	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
109	4.056867	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
110	4.056871	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
111	4.239148	74.125.24.94	192.168.1.15	TCP	54	443 → 52357 [ACK] Seq=1 Ack=1 Win=255 Len=0
112	4.279042	Zte_98:95:1e	HonHaiPr_3d:74:66	ARP	60	Who has 192.168.1.15? Tell 192.168.1.1

Do có rất nhiều gói tin gửi nên mình sẽ dùng chức năng lọc để lọc lấy các gói tin arp

*Wi-Fi

No.	Time	Source	Destination	Protocol	Length	Info
65180	175.940289	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65181	175.940294	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65184	176.195723	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65185	176.195728	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65188	176.446108	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65189	176.446112	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65190	176.698977	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65191	176.698981	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65216	176.954429	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65217	176.954433	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65221	177.207111	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65222	177.207116	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65232	177.514922	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65233	177.514926	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65236	177.764621	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65237	177.764625	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65241	178.014948	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65242	178.014953	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65254	178.270892	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197
65255	178.270894	DIAB_07:19:98	Broadcast	ARP	62	Who has 192.168.1.1? Tell 192.168.1.197

> Frame 64314: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0

```

0000 ff ff ff ff ff ff 00 20 07 19 98 08 06 00 01 .....
0010 08 00 06 04 00 01 00 00 20 07 19 98 c0 a8 01 c5 .....
0020 00 00 00 00 00 00 c0 a8 01 01 00 00 00 00 00 00 .....
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

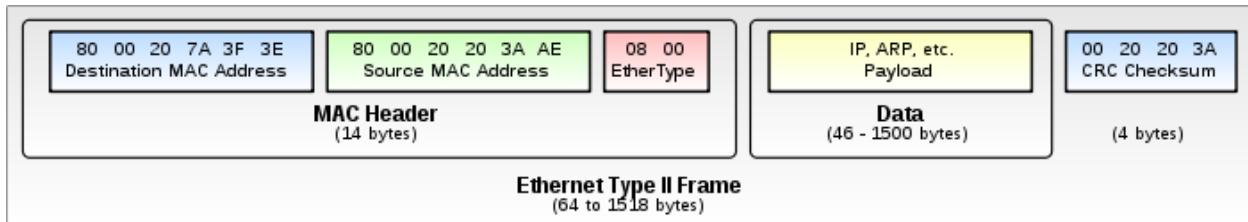
wireshark_Wi-Fi_20200403001831_a04160.pcapng | Packets: 65265 · Displayed: 1318 (2.0%) | Profile: Default

Các bạn thấy đây, phần mềm đã bắt được gói tin mà mình đã gửi đi, như vậy hàm **ENC28J60_send_packet** đã gửi thành công gói tin vào môi trường mạng LAN rồi !

Lan màn vậy đủ rồi, chúng ta sẽ vào chủ đề chính ngày hôm nay

Cấu trúc chung của 1 gói tin

Do chúng ta đang làm việc với chip enc29j60 sử dụng giao thức mạng ethernet, tất cả các gói tin truyền nhận đều phải tuân thủ quy tắc của **Ethernet Frame Type II**



Chúng ta cần quan tâm 14 byte đầu tiên :

Với 6 byte đầu tiên luôn là địa chỉ MAC của thằng nhận

6 byte tiếp theo là địa chỉ MAC của thằng gửi

2 byte tiếp theo sẽ nói lên giao thức phía sau của nó thuộc loại gì ? IP hay ARP ????

Giao thức ARP (Ether Type =0x0806)

<----- 14 byte -----> <--- 28 byte --->

Ethernet Frame Type II

ARP

<---- Ethernet Buffer ----->

Như đã nói bên trên, giao thức ARP dùng để lấy địa chỉ MAC của 1 thiết bị từ địa chỉ IP

Ví dụ ENC28J60 có ip 192.168.197 muốn gửi một tin nhắn cho máy tính có ip 192.168.1.4, trước tiên nó sẽ gửi 1 gói tin ARP request (broadcast) tới toàn bộ thiết bị trong mạng, gói tin ARP request đó có nội dung như sau:

Tao là 192.168.1.197 đây, MAC của tao là xxx , thằng nào là 192.168.1.4 xưng danh

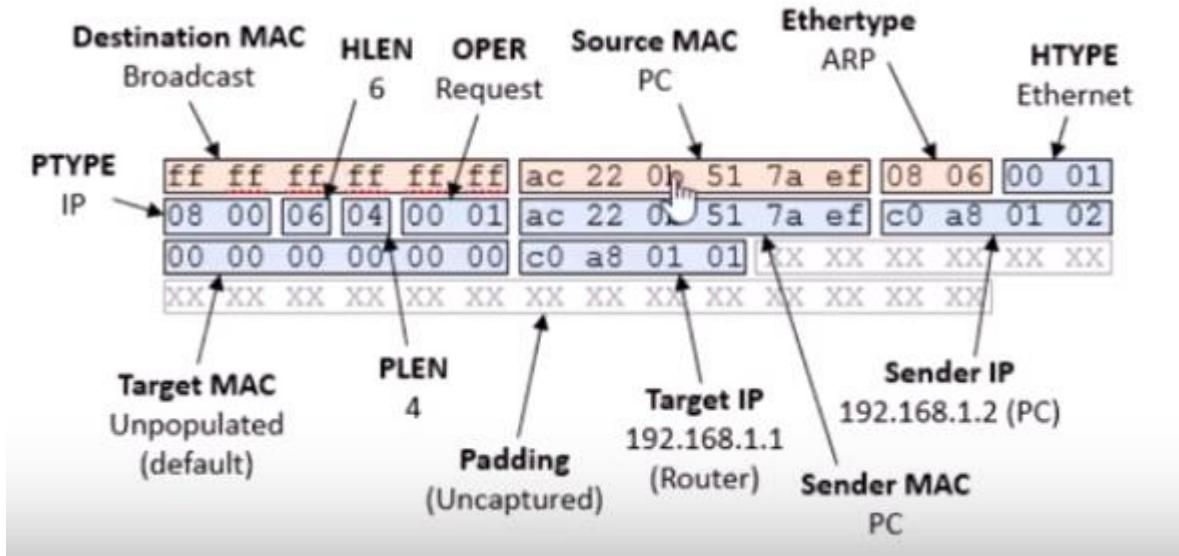
Toàn bộ thiết bị trong mạng đó sẽ nhận được tin nhắn này, những thằng nào không phải 192.168.1.4 sẽ ngó lơ bản tin này, chỉ riêng máy tính có địa chỉ 192.168.1.4 sẽ gửi 1 gói tin trả lời trực tiếp cho ENC28J60 có ip 192.168.1.197 do nó đã biết MAC của 192.168.1.197 (ARP reponse)

Tao là 192.168.1.4 đây, địa chỉ MAC của tao là AA:BB:CC:DD:EE:FF

ENC28J60 nhận được tin nhắn phải hồi và lưu địa chỉ MAC lại, bây giờ nó có thể gửi tin nhắn tới cho máy tính qua địa chỉ MAC

Bản tin ARP Request

Chúng ta sẽ ngâm cứu kỹ hơn cấu trúc của 1 ARP request



Giải thích cấu trúc :

- 6 byte đầu mặc định: 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF
- 6 byte tiếp theo là địa chỉ MAC của nguồn – tức là của chính thằng gửi request
- 2 byte 0x08 0x06 thể hiện đây là bản tin ARP (Ethernet Type)
- 2 byte 0x00 0x01 thể hiện htype
- 2 byte 0x80 0x00 thể hiện Protocol type, ở đây là IPv4
- Tiếp đến là 0x06 và 0x04 là trường length
- 2 byte 0x00 và 0x01 là opcode báo bản tin ARP này là request
- Sender MAC và SenderIP của thiết bị request
- Trường Taget MAC là của đích (tức thiết bị nhận request) mặc định là 6 byte 0x00 do ta chưa biết MAC của nó
- 4 byte cuối là IP của đích

Nói chung, chúng ta chỉ cần quan tâm tới SourceMAC SenderMAC SenderIP và TargetIP

Bản tin ARP Reponse

Bản tin ARP reponse có cấu trúc tương tự như các Request, chỉ khác nhau là Opcode sẽ là 0x0002 để thể hiện đây là reponse và Destination MAC và Target MAC sẽ không phải là 6 byte mặc định nữa mà là MAC của thiết bị đã gửi request

Dưới đây là ảnh chụp của 1 ARP reponse được bắt bởi Wireshark

76 4.620470	HonHaiPr_3d:74:66	DIAB_07:19:98	Broadcast	ARP	72 192.168.1.197 13 01 00:26:5E:3D:74:66
79 4.620470	DIAB_07:19:98		Broadcast	ARP	62 Who has 192.168.1.15? Tell 192.168.1.197
80 4.620475	DIAB_07:19:98		Broadcast	ARP	62 Who has 192.168.1.15? Tell 192.168.1.197
81 4.620496	HonHaiPr_3d:74:66	DIAB_07:19:98	ARP		42 192.168.1.15 is at 00:26:5E:3D:74:66
82 4.620497	HonHaiPr_3d:74:66	DIAB_07:19:98	ARP		42 192.168.1.15 is at 00:26:5E:3D:74:66
83 4.876545	DIAB_07:19:98		Broadcast	ARP	62 Who has 192.168.1.15? Tell 192.168.1.197
84 4.876551	DIAB_07:19:98		Broadcast	ARP	62 Who has 192.168.1.15? Tell 192.168.1.197
85 4.876572	HonHaiPr_3d:74:66	DIAB_07:19:98	ARP		42 192.168.1.15 is at 00:26:5E:3D:74:66
86 4.876572	HonHaiPr_3d:74:66	DIAB_07:19:98	ARP		42 192.168.1.15 is at 00:26:5E:3D:74:66
87 5.125034	DIAB_07:19:98		Broadcast	ARP	62 Who has 192.168.1.15? Tell 192.168.1.197

Protocol size: 4

```
0000 00 00 20 07 19 98 00 26 5e 3d 74 66 08 06 00 01  . . . . . & ^tf...
0010 08 00 06 04 00 02 00 26 5e 3d 74 66 c0 a8 01 0f  . . . . . & ^tf...
0020 00 00 20 07 19 98 c0 a8 01 c5  . . . . .
```

Packets: 95 · Displayed: 88 (92.6%) · Dropped: 0 (0.0%) ||| Profile: Default

Các bạn có thể thấy, sau khi mình cho ENC28J60 gửi 1 broadcast (arp request) tới cho IP của máy tính thì máy tính đã trả lời lại MAC của nó (**192.168.1.15 is at 00:26:5E:3D:74:66**)

Thời gian ARP

Khi 1 thiết bị đã lấy được MAC của thiết bị khác từ IP, nó sẽ lưu trữ lại vào bảng để lần gửi sau chỉ việc lấy ra dùng chứ không cần phải hỏi lại nữa. Tuy nhiên trong mạng LAN, IP của các thiết bị là IP động, nó thường hay thay đổi mỗi khi thiết bị khởi động lại. Vì vậy các gói tin ARP vẫn thường xuyên được gửi để cập nhật lại MAC cho chính xác. Tránh việc sử dụng các địa chỉ đã lỗi thời

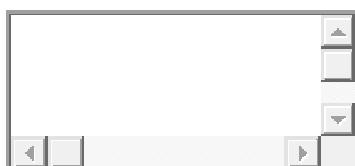
Thông thường, các bảng lưu ARP được cập nhật từ 30 đến 60s 1 lần

Tạo thư viện ARP

Trước khi viết thư viện **ARP**, mình sẽ tạo ra thư viện trung gian có tên là **NET** để làm trung gian giao tiếp giữa thư viện **ENC28J60** với thư viện **ARP** và các thư viện khác nữa !

Chúng ta tạo 2 tệp là **net.c** và **net.h** và add vào project tương tự như hướng dẫn ở các bài trước

Các bạn copy nội dung khôi tạo cho file net.h



```
1 #ifndef NET_H_
2 #define NET_H_
3 //-----
4 //Include cac thu vien can thiet
5 #include <mega328p.h>
6 #include <delay.h>
```

```

7 #include <string.h>
8 #include <stdlib.h>
9 #include <stdint.h>
10#include <stdio.h>
11#include <spi.h>
12#include "uart.h"
13#include "enc28j60.h"
14//-----
15
16//-----
17#endif /* ENC28J60_H_ */

```

Tương tự, copy nội dung khởi tạo cho file net.c



```

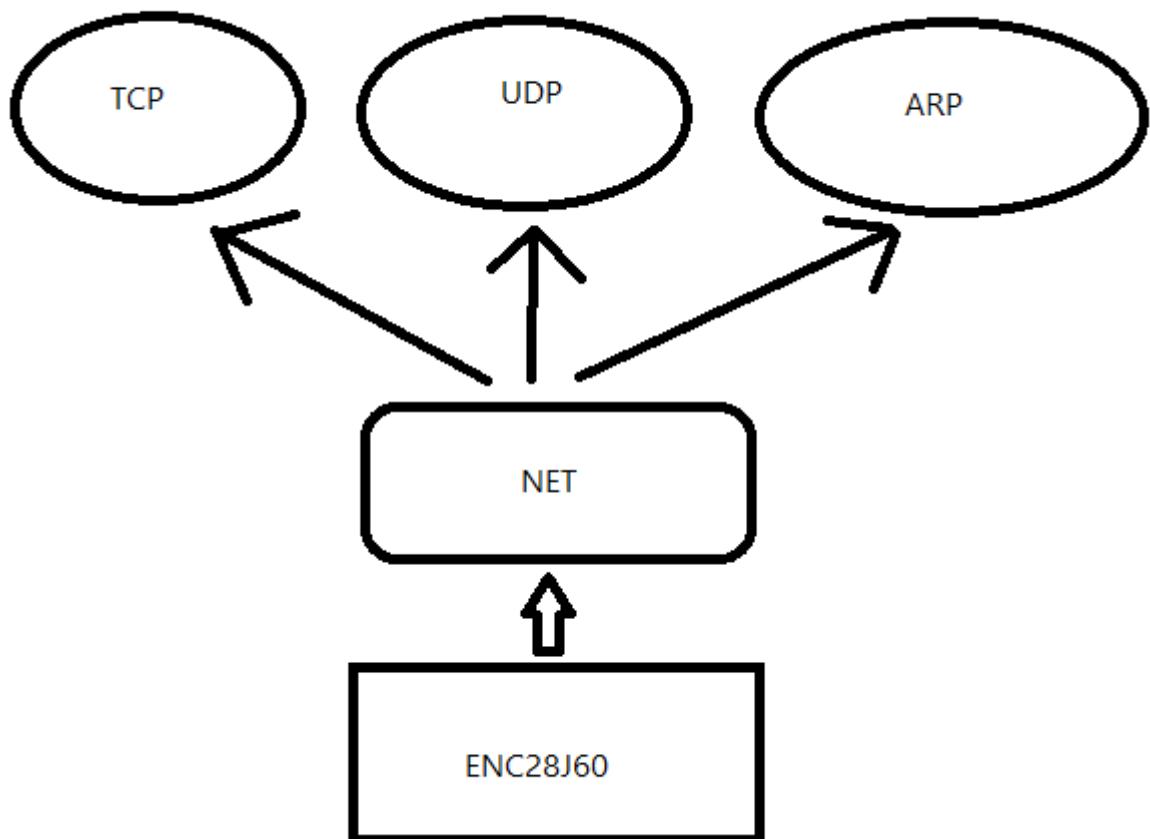
1#include "net.h"
2
3
4
5
6//-
7//End file

```

Chú ý: Các này đáng lẽ mình phải nhắc từ đâu, tất cả các file .h .c các bạn phải có 1 dòng trống ở cuối file nhé, không thì trình dịch sẽ báo lỗi

Nhiệm vụ của thư viện NET

- Phân tích gói tin xem nó thuộc loại gì, tùy vào loại của gói tin để gọi thư viện xử lý tương ứng. Ví dụ thư viện NET sẽ đọc trường **Ethernet Type** của gói tin. Nếu nó là 0x0806 thì đây là gói ARP => gọi thư viện ARP và ném gói này cho nó xử lý. Nếu là gói tin chưa được hỗ trợ thì bỏ qua hoặc thông báo lỗi gì đó tùy ứng dụng ...
- Tính toán checksum
- Gửi nhận gói tin
- Kiểm tra liên tục xem có gói tin mới nào không



hình hệ thống

Mô

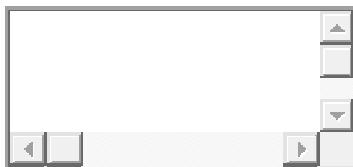
Để có thể xử lí, phân tích các gói tin, mình sẽ khởi tạo trên ram của chip vi điều khiển 1 bộ đệm dài 512 byte



Thêm định #define cho BUFFER_LENGTH

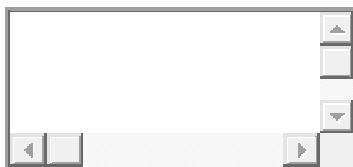


Hàm **NET_SendFrame** sẽ gọi lại hàm **ENC28J60_send_packet**



```
1 void NET_SendFrame(uint8_t *EthFrame,uint16_t len)
2 {
3     sprintf(debug_string,"Gửi gói tin ethernet có độ dài: %i\r\n",len);
4     UART_putString(debug_string);
5     ENC28J60_send_packet(EthFrame,len);
6 }
```

Hàm **NET_loop** có nhiệm vụ check có gói tin nào không để gửi gói tin cho hàm **NET_read** xử lý gói tin



```
1 void NET_loop(void)
2 {
3     uint16_t len;
4     while(1)
5     {
6         len=ENC28J60_read_packet(eth_buffer,BUFFER_LENGTH);
7         if(len==0) return;
8         else NET_read(eth_buffer,len);
9     }
10 }
```

Các bạn nhớ thêm nguyên mẫu hàm vào file *net.h*

Hàm **NET_loop** sẽ được gọi trong vòng lặp `while(1)` của hàm main để liên tục thăm dò gói tin

```

UART_init();
#asm("sei")

UART_putString("IOT47.com - ENC28J60 tutorial !!!\r\n");
ENC29J600_ini();
while (1)
{
    NET_loop();
}

```

Bây giờ chúng ta sẽ viết nội dung cho hàm **NET_read** cho nó mở gói tin và xử lý xem bên trong gói tin có gì



```

1 void NET_read(uint8_t *net_buffer,uint16_t len)
2{
3
4}

```

Khi 1 gói tin được gửi tới, chúng ta sẽ ngay lập tức biết nó có phải là gói tin ARP không bằng cách check trường **Ethernet Type**

- Nếu **Ethernet Type** = 0x0806 -> Đây là gói tin ARP
- Ngược lại nếu **Ethernet Type** = 0x0800 -> Đây là gói tin IP
- Ngược lại đây là 1 gói tin lạ, chúng ta sẽ bỏ qua

Chúng ta sẽ tạm thời chưa quan tâm gói tin IP vội, mình sẽ xử lí nó sau, trước mắt sẽ viết code để xử lí gói tin ARP

Cúc bước kiểm tra

1. Một bản tin ARP phải có ít nhất 42 byte, nếu không đủ 42 byte -> loại
2. Check **Ethernet Type** tại byte thứ 12 và 13 xem có phải 0x0806 không, nếu không phải -> loại
3. Tới đây, đã có thể chắc chắn đây là 1 gói tin ARP rồi, chúng ta sẽ lại gửi gói tin cho thư viện ARP xử lí



```

1 void NET_read(uint8_t *net_buffer,uint16_t len)
2{

```

```

3 //in ra do dai cua goi tin
4 sprintf(debug_string,"Da doc goi tin ra, do dai goi tin=%i\r\n",len);
5 UART_putString(debug_string);
6
7 //kiem tra xem co phai goi tin ARP khong
8 if(len>41)
9 {
10 if(net_buffer[12] == 0x08 && net_buffer[13] == 0x06)
11 {
12 //day dung la goi tin ARP
13 UART_putString("Day la goi tin ARP\r\n");
14 //gui goi tin cho thu vien ARP.h xu li
15 //ARP_read_packet(uint8_t *net_buffer,uint16_t len);
16 }
17 }
18}

```

Đương nhiên hàm **ARP_read_packet** mình chưa viết nên sẽ tạm comment nó chung ta sẽ viết nó ngay, nhưng sẽ viết trong thư viện **arp.c** chứ không viết ở file **net.c**

Test thử

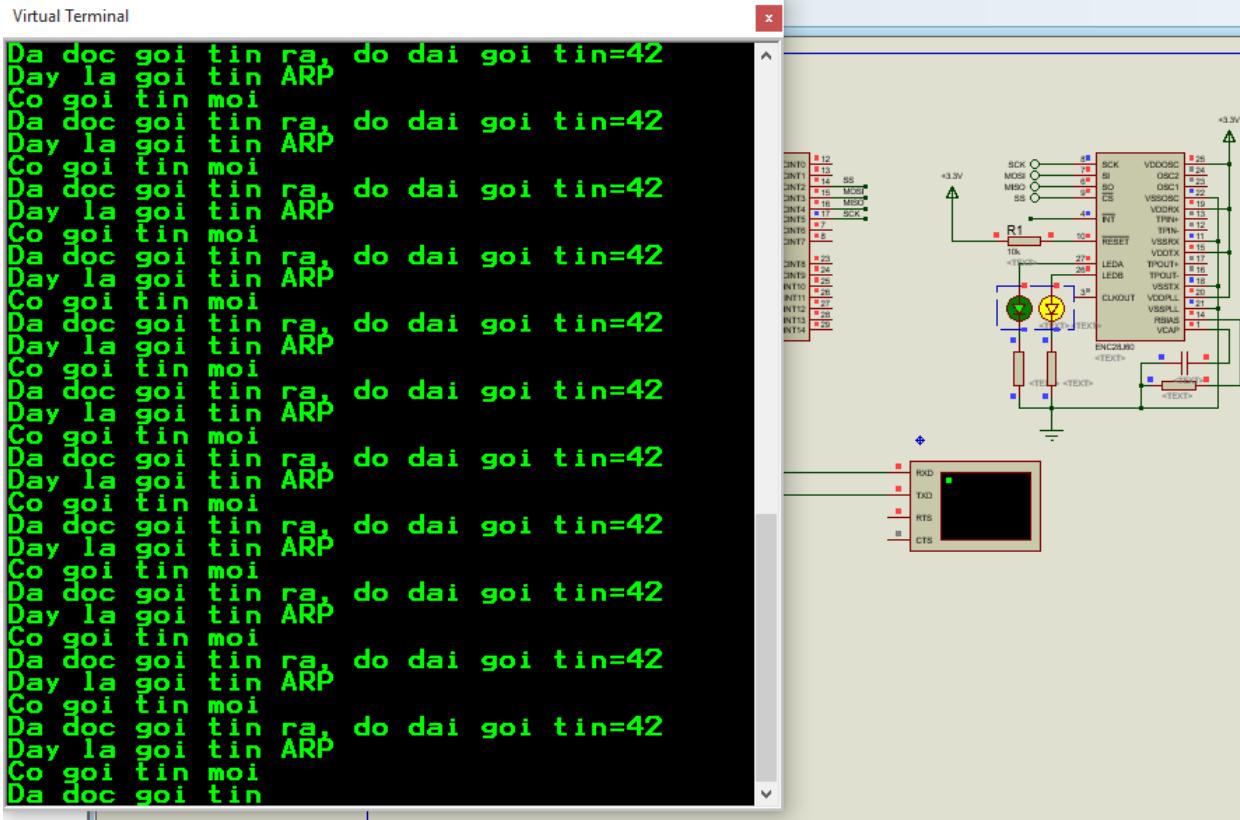
Code nhiều rồi mà không test thì cũng chán nhỉ ! Chúng ta sẽ test thử bằng cách cho điện thoại gửi thử các bản tin ARP để xem chương trình của chúng ta có nhận được các gói ARP này không nhé !

Để gửi được các broadcast (ARP request) đi, mình sẽ sử dụng 1 phần mềm Scan IP bất kì. Ví dụ ở điện thoại android mình sẽ xài **Network IP Scan**

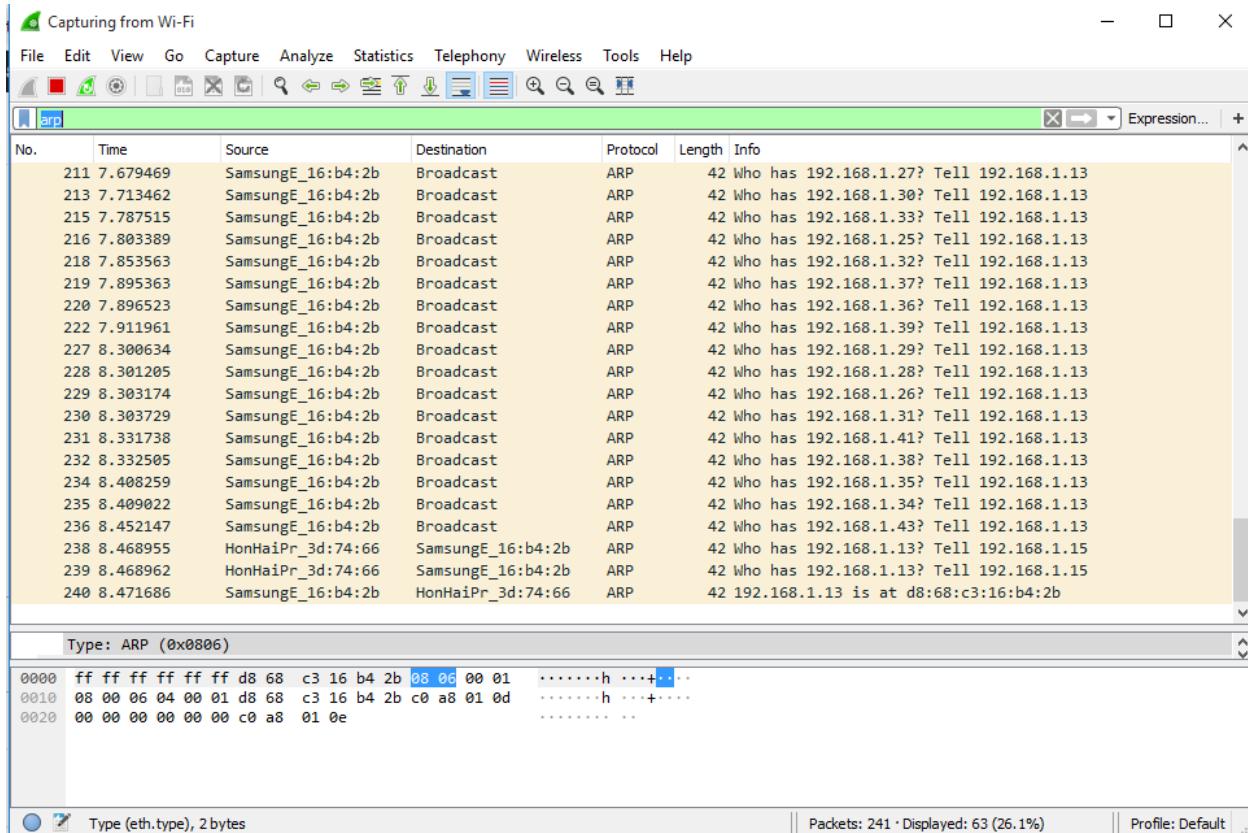
Các bạn có thể dùng phần mềm sacn bất kì

Bản chất của phần mềm scan IP là nó sẽ gửi các broadcast (tức ARP request) đến toàn mạng. Thiết bị nào có IP tương ứng sẽ trả lời lại cho nó

Bây giờ, mình chạy mô phỏng proteus và mở phần mềm scan lên cho nó scan và xem kết quả (mình cũng mở đồng thời WireShark lên nữa)



Các bạn thấy đó, chúng ta đã bắt được rất nhiều gói tin ARP request của phần mềm scan (toàn bộ thiết bị trong mạng wifi đều nhận được nha)



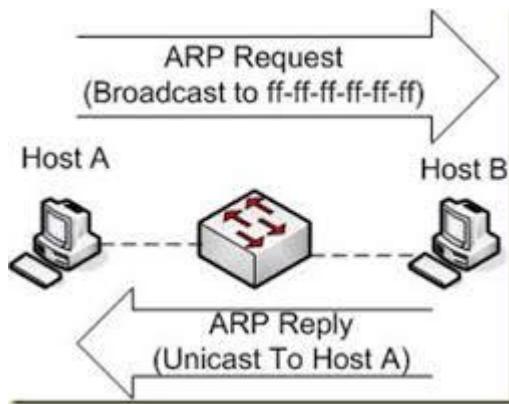
Đồng thời thì wireshark nó cũng bắt được rất nhiều gói tin ARP

Như vậy chúng ta đã bắt được các gói tin ARP. Trong phần tiếp theo của bài ARP, chúng ta sẽ tiếp tục hoàn thành thư viện xử lí gói tin ARP và viết hàm trả lời các bản tin ARP request nhé !

Các bạn có thể tải code cho bài học hôm nay [tại đây](#) !

[ENC28J60] Bài 7: Giao thức ARP (phần 2)

3 Tháng Tư, 2020 Đào Nguyễn Ethernet, IoT tutorial 1



Trong **bài trước**, chúng ta đã tìm hiểu về giao thức ARP và đã có thể bắt được cái gói tin ARP request được gửi tới. Và ở bài này, chúng ta sẽ viết chương trình để trả lời các ARP requests đó (ARP response)

Các gói tin ARP sẽ được xử lý ở thư viện ARP, nên mình sẽ tạo thêm 2 file `arp.c` và `arp.h` và add vào project như đã nói ở các bài trước (vào **Project -> Configure** để add file `arp.c`)

Ở file `arp.h` các bạn copy khôi code cơ bản

```
1 #ifndef ARP_H_
2 #define ARP_H_
3 //-----
4 //Include cac thu vien can thiet
5 #include <mega328p.h>
6 #include <delay.h>
7 #include <string.h>
8 #include <stdlib.h>
9 #include <stdint.h>
10 #include <stdio.h>
11 #include <spi.h>
12 #include "uart.h"
13 #include "net.h"
14 //-----
15
16 //-----
17#endif /* ARP_H_ */
```

Tương tự với file `arp.c`



```
1#include "arp.h"
2
3
4
5
6//-----
7//End file
```

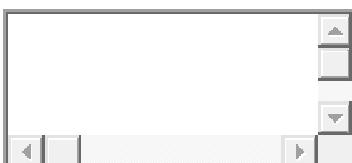
Trong file **net.h** các bạn add thêm thư viện **arp.h** vô

```
#ifndef NET_H_
#define NET_H_
//-----
//Include cac thu vien can thiet
#include <mega328p.h>
#include <delay.h>
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <spi.h>
#include "uart.h"
#include "enc28j60.h"
#include "arp.h"
//-----
```

Đồng thời trong file **net.c** , bỏ comment cho hàm **ARP_read_packet** vì chúng ta sẽ viết hàm này ngay bây giờ

*Giờ mình mới để ý lời gọi hàm **ARP_read_packet** trong file **net.c** mình viết sai. Các bạn sửa lại cho đúng nhé ! **ARP_read_packet(net_buffer,len);***

Tạo hàm **ARP_read_packet** trong file **arp.c** và khai báo nguyên mẫu hàm cho nó vào file **arp.h**



```
1void ARP_read_packet(uint8_t *ARP_Buff,uint16_t len)
2{
3
4}
```

Chúng ta đã nhận được 1 gói tin ARP, công việc bây giờ là :

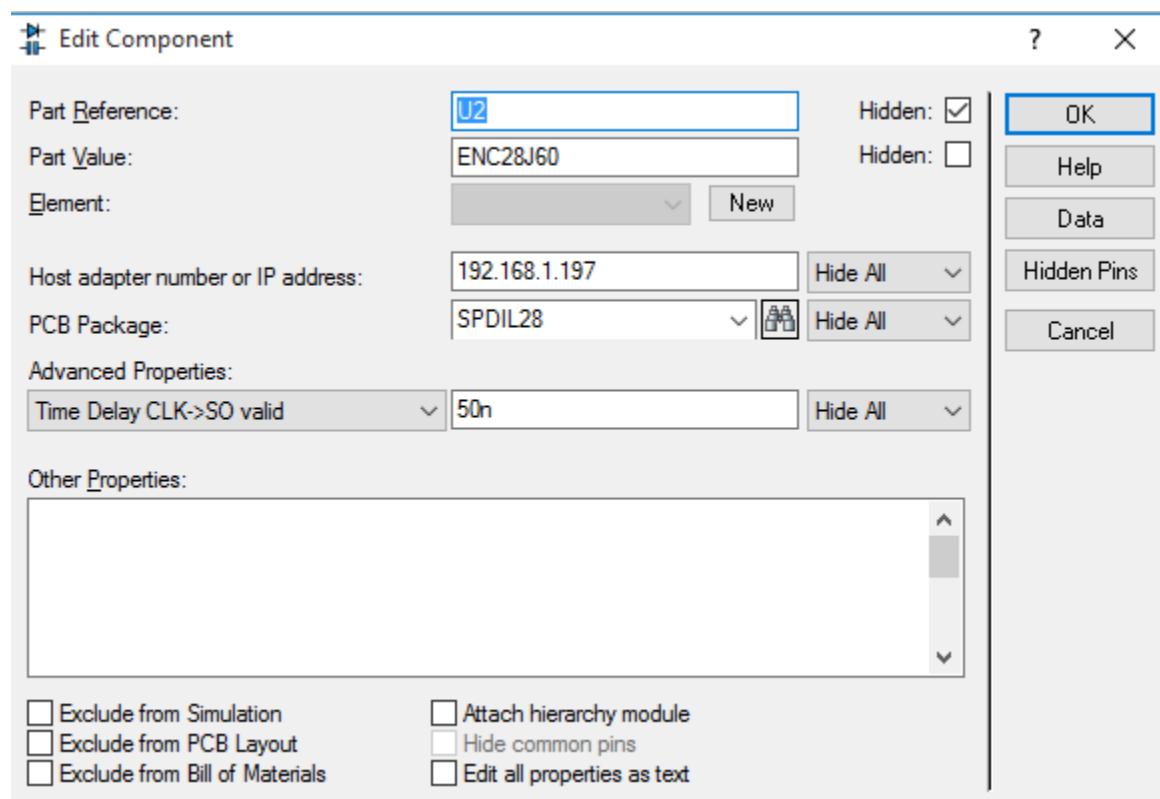
- Kiểm tra xem gói tin ARP này thuộc loại gì: ARP request hay ARP reponse bằng cách check trường Opcode tại vị trí byte thứ 20 21

- Nếu đó là ARP request thì phải chuẩn bị gói tin trả lời lại
- Nếu đó là ARP response, kiểm tra địa chỉ MAC đích xem có đúng là nó response cho mình không, nếu không trùng MAC của mình thì bỏ qua. Nếu đúng là nó gửi cho mình thì tiến hành lưu MAC của thằng đã response vào bảng để sử dụng

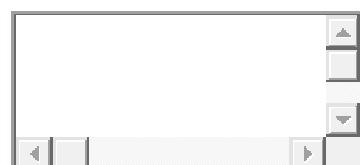
Chà ! Khối lượng công việc khá nhiều đây, chúng ta sẽ làm dần từng ý một

Bây giờ mình sẽ gán cho chip của chúng ta 1 địa chỉ IP, do IP này là IP cố định dễ có khả năng trùng IP với thiết bị trong mạng, nên mình sẽ lấy 1 IP có giá trị cao 1 chút 192.168.1.197 đi

Các bạn click chuột vào con chip ENC28J60 trong mô phỏng, điền IP mà mình gán cho nó vào

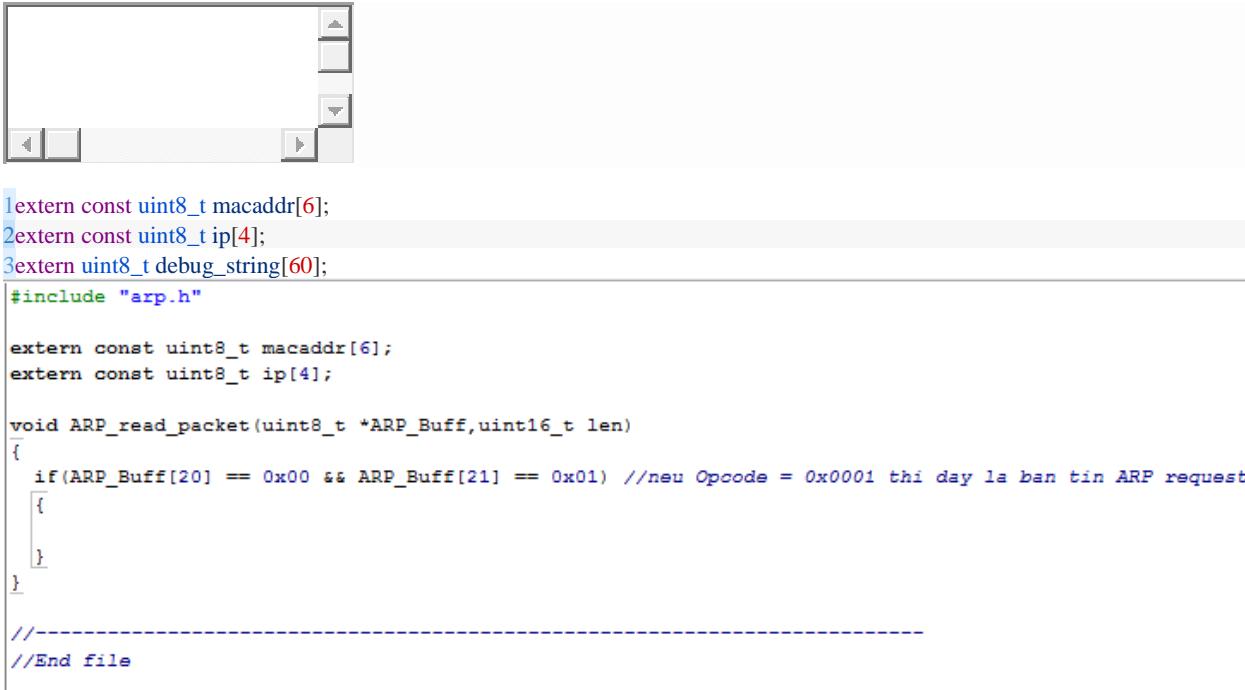


Tương tự, trong file enc28j60.c, cạnh chỗ khai báo MAC, mình cũng khai báo IP 192.168.1.197 vô



```
1const uint8_t ip [4] = { 192,168,1,197};
```

Và khai báo lại ip với mac ở các file thư viện .c khác bằng cú pháp **extern**



```

1 extern const uint8_t macaddr[6];
2 extern const uint8_t ip[4];
3 extern uint8_t debug_string[60];
4 #include "arp.h"

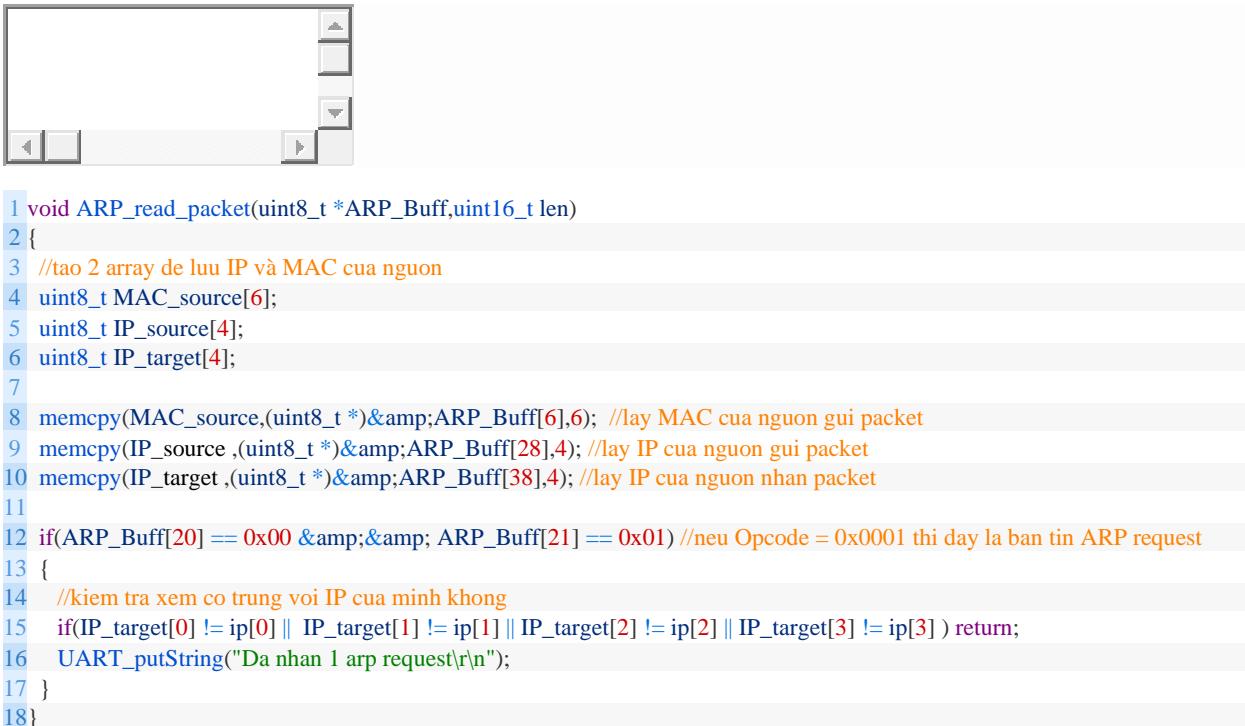
5 extern const uint8_t macaddr[6];
6 extern const uint8_t ip[4];

7 void ARP_read_packet(uint8_t *ARP_Buff,uint16_t len)
8 {
9     if(ARP_Buff[20] == 0x00 && ARP_Buff[21] == 0x01) //neu Opcode = 0x0001 thi day la ban tin ARP request
10    {
11    }
12 }

13 //-----
14 //End file

```

Tiếp tục với nội dung hàm `ARP_read_packet`



```

1 void ARP_read_packet(uint8_t *ARP_Buff,uint16_t len)
2 {
3     //tao 2 array de luu IP va MAC cua nguon
4     uint8_t MAC_source[6];
5     uint8_t IP_source[4];
6     uint8_t IP_target[4];
7
8     memcpy(MAC_source,(uint8_t *)&ARP_Buff[6],6); //lay MAC cua nguon gui packet
9     memcpy(IP_source ,(uint8_t *)&ARP_Buff[28],4); //lay IP cua nguon gui packet
10    memcpy(IP_target ,(uint8_t *)&ARP_Buff[38],4); //lay IP cua nguon nhan packet
11
12    if(ARP_Buff[20] == 0x00 && ARP_Buff[21] == 0x01) //neu Opcode = 0x0001 thi day la ban tin ARP request
13    {
14        //kiem tra xem co trung voi IP cua minh khong
15        if(IP_target[0] != ip[0] || IP_target[1] != ip[1] || IP_target[2] != ip[2] || IP_target[3] != ip[3] ) return;
16        UART.putString("Da nhan 1 arp request\r\n");
17    }
18}

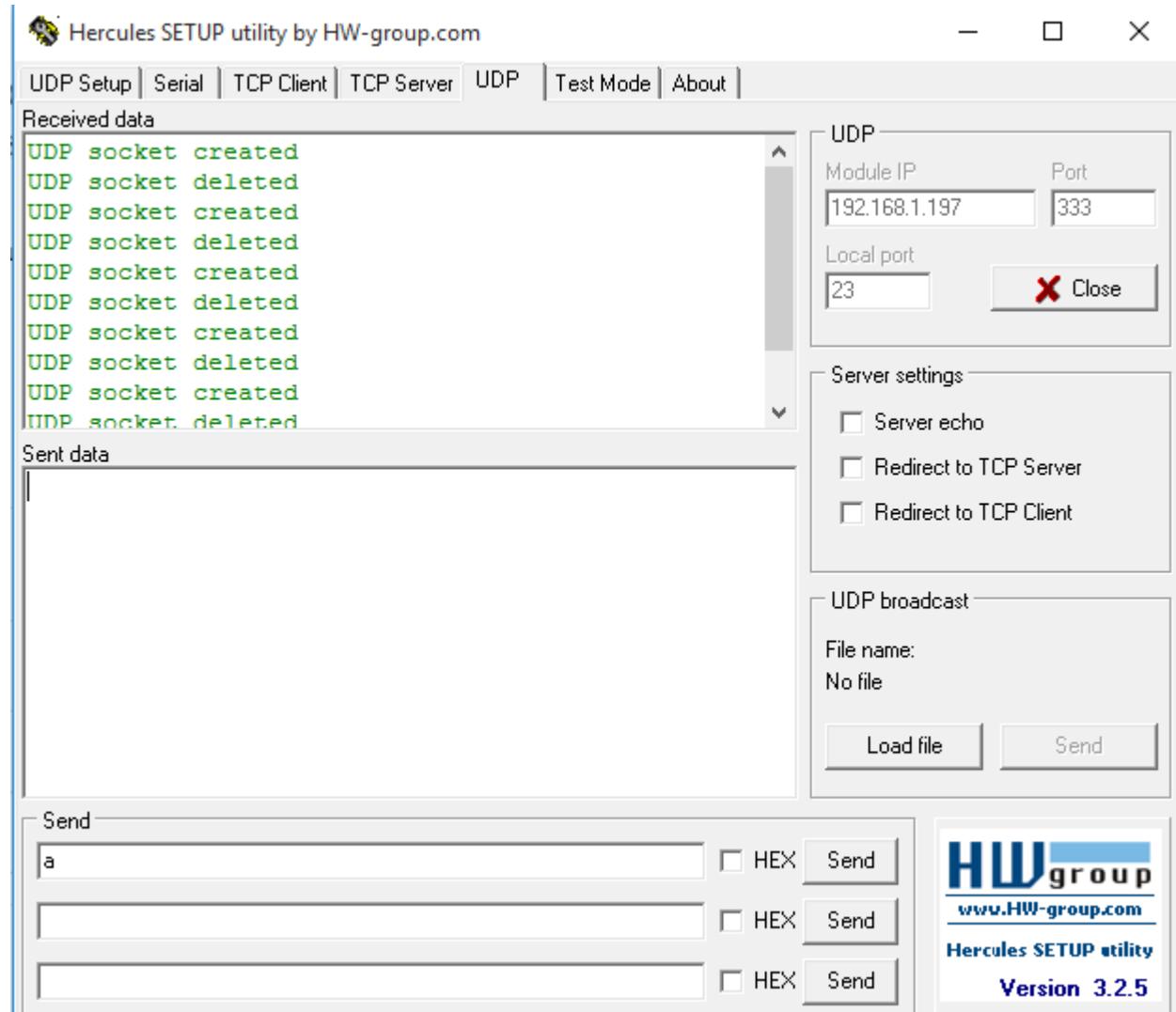
```

Giải thích: Sau khi lấy được các thông tin trong gói arp, chúng ta kiểm tra Opcode xem nó có phải là arp request không, nếu đúng tiếp tục kiểm tra xem ARP request này có ip đích trùng với ip của mình không ! nếu không trùng thì bỏ qua ! Nếu đúng thì sẽ in ra dòng debug là đã nhận được 1 arp request

Chúng ta sẽ test thử xem code đã hoạt động chưa nhé. Mình sẽ dùng máy tính để gửi 1 arp request tới ip của enc28j60

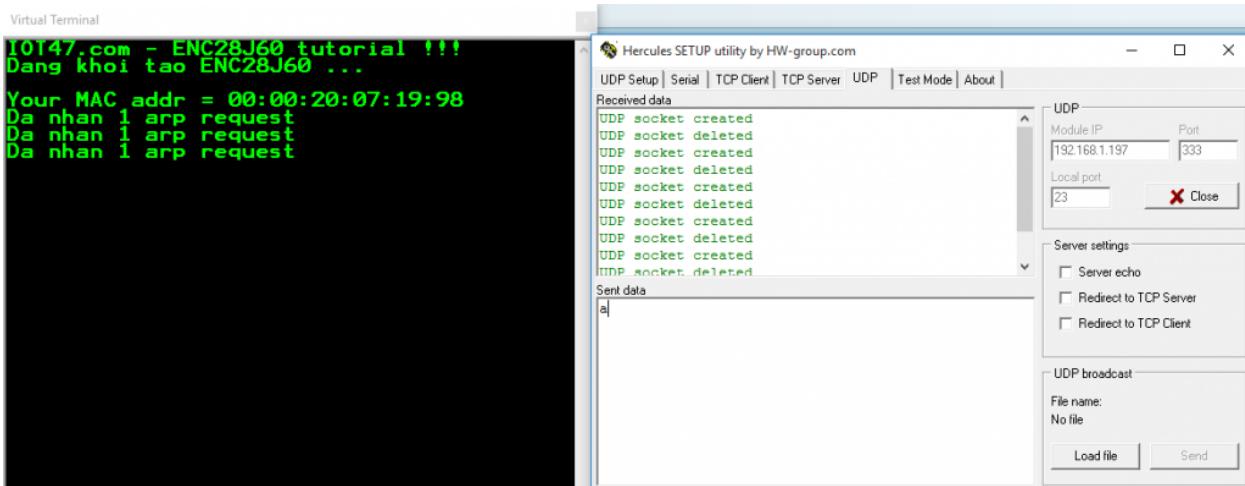
Trước đó mình sẽ commnet mấy dòng debug (**UART_putString**) ở các file **enc28j60.c** và **net.c** đi vì chúng ta đã test các hàm đó hoạt động rồi, in ra lăm cung ngứa mắt

Bây giờ mình sẽ dùng phần mềm herculer, chức năng UDP để cho nó gửi thử 1 arp request xem chíp có bắt được không nhé !



Chỗ module IP mình sẽ điền IP của ENC28j60 (ip đích)

Các bạn gửi thử 1 nội dung bất kì, phần mềm herculer sẽ tự động gửi request tới ip đã điền



OK, chúng ta đã bắt được arp request

Bây giờ mình sẽ viết thêm code để phản hồi lại nhé



```

1 void ARP_read_packet(uint8_t *ARP_Buff,uint16_t len)
2 {
3     struct
4     {
5         uint8_t MAC_dich[6];           // MAC dich
6         uint8_t MAC_nguon[6];         //MAC nguon
7         uint16_t Ethernet_type;       //ethernet type
8         uint16_t Hardwave_type;       //hardwave type
9         uint16_t Protocol_type;       //protocol type (ARP)
10        uint16_t Size;                //size
11        uint16_t Opcode;              //opcode
12        uint8_t MAC_sender[6];        //sender MAC
13        uint8_t IP_sender[4];         //sender IP
14        uint8_t MAC_target[6];        // Target MAC
15        uint8_t IP_target[4];         // Target IP
16    }ARP_reponse;
17
18 //tao 2 array de luu IP va MAC cua nguon
19 uint8_t MAC_source[6];
20 uint8_t IP_source[4];
21 uint8_t IP_target[4];
22
23 memcpy(MAC_source,(uint8_t *)&ARP_Buff[6],6); //lay MAC cua nguon gui packet
24 memcpy(IP_source ,(uint8_t *)&ARP_Buff[28],4); //lay IP cua nguon gui packet
25 memcpy(IP_target ,(uint8_t *)&ARP_Buff[38],4); //lay IP cua nguon nhan packet
26
27 if(ARP_Buff[20] == 0x00 && ARP_Buff[21] == 0x01) //neu Opcode = 0x0001 thi day la ban tin ARP request
28 {
29     //kiem tra xem co trung voi IP cua minh khong

```

```

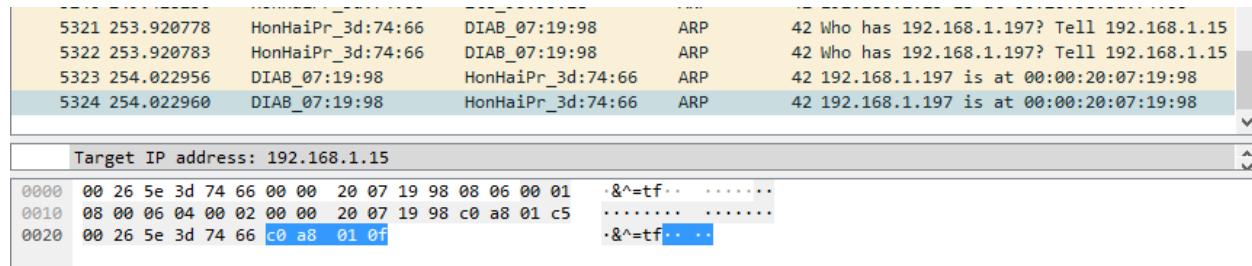
30 if(IP_target[0] != ip[0] || IP_target[1] != ip[1] || IP_target[2] != ip[2] || IP_target[3] != ip[3] ) return;
31 UART.putString("Da nhan 1 arp request\r\n");
32 //tao goi tin ARP reponse
33 memcpy(ARP_reponse.MAC_dich,MAC_source,6); //dia chi MAC cua thang nhan goi tin reponse
34 memcpy(ARP_reponse.MAC_target,MAC_source,6);
35 memcpy(ARP_reponse.IP_target,IP_source,4); //dia chi IP cua thang nhan goi tin reponse
36 memcpy(ARP_reponse.MAC_nguon,macaddr,6); //dia chi MAC cua thang gui ( chinh la MAC cua ENC28J60)
37 memcpy(ARP_reponse.MAC_sender,macaddr,6);
38 memcpy(ARP_reponse.IP_sender,ip,4); //dia chi IP cua thang gui goi tin reponse ( chinh la MAC cua ENC28J60)
39 ARP_reponse.Ethernet_type=0x0608;
40 ARP_reponse.Hardware_type=0x0100;
41 ARP_reponse.Protocol_type=0x0008;
42 ARP_reponse.Size=0x0406;
43 ARP_reponse.Opcode=0x0200;
44 UART.putString("Gui gon tin reply\r\n");
45 NET_SendFrame((uint8_t *)&ARP_reponse,42); //gui goi ARP reponse di
46 }
47}

```

Giải thích:

Mình tạo 1 Struct theo cấu trúc của 1 gói tin trong giao thức ARP, sau đó điền các thông tin như địa chỉ MAC đích, Mac nguồn, IP đích, IP nguồn và gọi hàm **NET_SendFrame** để gửi trả lại gói tin ARP reponse

Và giờ, Wireshark đã bắt được bản tin ARP reponse của ENC28J60



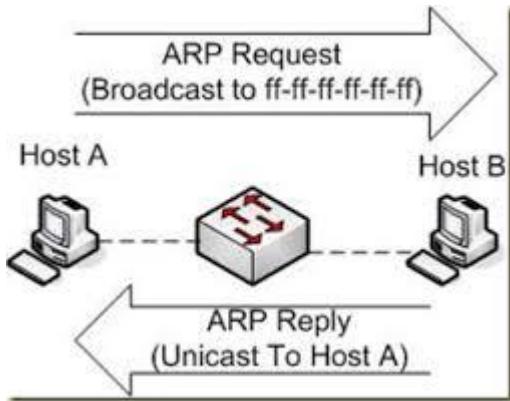
Thật tuyệt, tới thời điểm bây giờ chúng ta đã bắt được 1 gói tin ARP request và phản hồi lại cho thằng request 1 bản tin ARP reponse

Trong **bài tiếp theo**, chúng ta sẽ tự gửi đi 1 ARP request và bắt lại gói tin ARP reponse và thiết kế bảng lưu địa chỉ MAC cho giao thức ARP

Các bạn có thể tải source cho bài này [ở đây](#)

[ENC28J60] Bài 8: Giao thức ARP (phần 3)

4 Tháng Tư, 2020 Đào Nguyễn Ethernet, IoT tutorial 0



Tiếp tục chuỗi bài về giao thức arp, hôm nay chúng ta sẽ hoàn thành nốt chương trình gửi request, nhận reponse và tạo table để lưu các địa chỉ MAC của thiết bị trong mạng

Nếu chưa đọc **bài trước** các bạn có thể theo dõi [tại đây](#)



```
1 void ARP_send_request(uint8_t *ip_dest)
2 {
3     struct
4     {
5         uint8_t MAC_dich[6];           // MAC đích
6         uint8_t MAC_nguon[6];         // MAC nguồn
7         uint16_t Ethernet_type;       // ethernet type
8         uint16_t Hardwave_type;       // hardware type
9         uint16_t Protocol_type;       // protocol type (ARP)
10        uint16_t Size;               // size
11        uint16_t Opcode;             // opcode
12        uint8_t MAC_sender[6];        // sender MAC
13        uint8_t IP_sender[4];        // sender IP
14        uint8_t MAC_target[6];        // Target MAC
```

```

15 uint8_t IP_target[4];           // Target IP
16 }ARP_request;
17 uint8_t MAC_dest[6]={0xFF,0xFF,0xFF,0xFF,0xFF,0xFF};
18 uint8_t MAC_target[6]={0x00,0x00,0x00,0x00,0x00,0x00};
19 memcpy(ARP_request.MAC_dich,MAC_dest,6);
20 memcpy(ARP_request.MAC_target,MAC_target,6);
21 memcpy(ARP_request.IP_target,ip_dest,4);
22 memcpy(ARP_request.MAC_nguon,macaddr,6);
23 memcpy(ARP_request.MAC_sender,macaddr,6);
24 memcpy(ARP_request.IP_sender,ip,4);
25 ARP_request.Ethernet_type=0x0608;
26 ARP_request.Hardwave_type=0x0100;
27 ARP_request.Protocol_type=0x0008;
28 ARP_request.Size=0x0406;
29 ARP_request.Opcode=0x0100; //request
30 UART_putString("Gui gon tin arp request\r\n");
31 NET_SendFrame((uint8_t *)&ARP_request,42); //gui goi ARP reponse di
32}

```

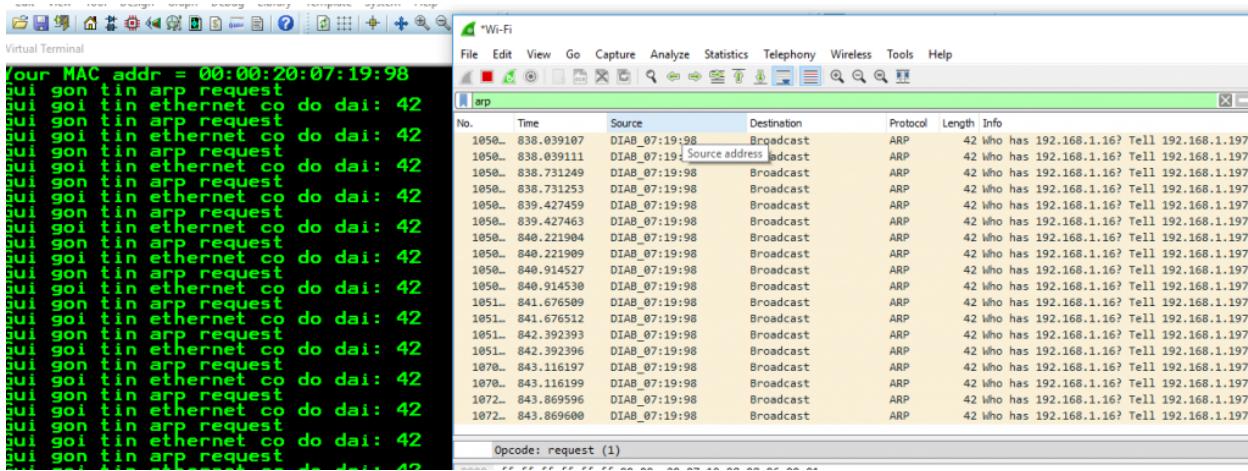
Hàm ARP request tương tự như reponse thôi, có opcode là khác. Gọi nó liên tục trong while(1) của hàm main để test thử xem đã gửi được đi chưa nhé !



```

1 //NET_loop();
2 ARP_send_request(ip_dest);
3 delay_ms(500);

```



Tuyệt vời, gói tin đã được gửi đi và wireshark đã bắt được các bản tin arp request

Bây giờ chúng ta sẽ tiếp tục viết mã đọc gói tin request trong hàm `ARP_read_packet`

1. Kiểm tra opcode, nếu đúng bằng 0x0002 thì đây là bản tin request
2. Kiểm tra target ip xem bản này có đúng là gửi cho mình không
3. Lấy MAC và ip của thiết bị đã gửi reponse và lưu lại vào bảng lưu trữ

Mình sẽ khởi tạo 1 table để lưu trữ ip và mac. Do bộ nhớ chip có hạn nên mình sẽ tạo bảng lưu với tối đa 5 địa chỉ MAC !



Viết thêm 1 số hàm để truy xuất bảng



```

1 int8_t ARP_table_checkIP(uint8_t *ip_check) //kiem tra xem co ton tai ip trong bang chua
2 {
3     int i;
4     for(i=0;i<5;i++)
5     {
6         if(ip_check[0] == ARP_table[i].ip[0] && ip_check[1] == ARP_table[i].ip[1] && ip_check[2] == ARP_table[i].ip[2] &&
7             ip_check[3] == ARP_table[i].ip[3])
8             return i+1; //tra ve vi tri cua ip trong table
9     }
10    return -1;
11 }

1 void ARP_table_setIP(uint8_t *ip_set, uint8_t *mac_set)
1 {
1
2     if(ARP_table_checkIP(ip_set) == -1) //neu chua ton tai IP trong table
3     {
4         sprintf(debug_string,"Da luu ip %i.%i.%i.%i =
1 %02X:%02X:%02X:%02X:%02X\r\n",ip_set[0],ip_set[1],ip_set[2],ip_set[3],mac_set[0],mac_set[1],mac_set[2],mac_se
4 t[3],mac_set[4],mac_set[5]);
5         UART.putString(debug_string);
6         memcpy(ARP_table[ARP_table_index].ip.ip_set,4);
7         memcpy(ARP_table[ARP_table_index].mac.mac_set,6);
8         ARP_table_index++;if(ARP_table_index==5)ARP_table_index=0; //neu so luong ip vuot qua muc cho phep thi quay lai
9     }
10
11    else
12        UART.putString("IP da ton tai trong bang\r\n");
13
14}
15
16
17
18
19
20
21
22
23

```

Trong hàm **ARP_read_packet**



```
1 else if(ARP_Buff[20] == 0x00 && ARP_Buff[21] == 0x02) //neu Opcode = 0x0001 thi day la ban tin ARP reponse
2 {
3     UART_putString("Da nhan 1 arp reponse\r\n");
4     ARP_table_setIP(IP_source,MAC_source);
5 }
```

Do cả ARP request hay ARP reponse đều cần kiểm tra IP đích xem có phải là của mình không nên mình sẽ bỏ nó lên đầu hàm luôn



```
1 //kiem tra xem co trung voi IP cua minh khong
2 if(IP_target[0] != ip[0] || IP_target[1] != ip[1] || IP_target[2] != ip[2] || IP_target[3] != ip[3] ) return;
```

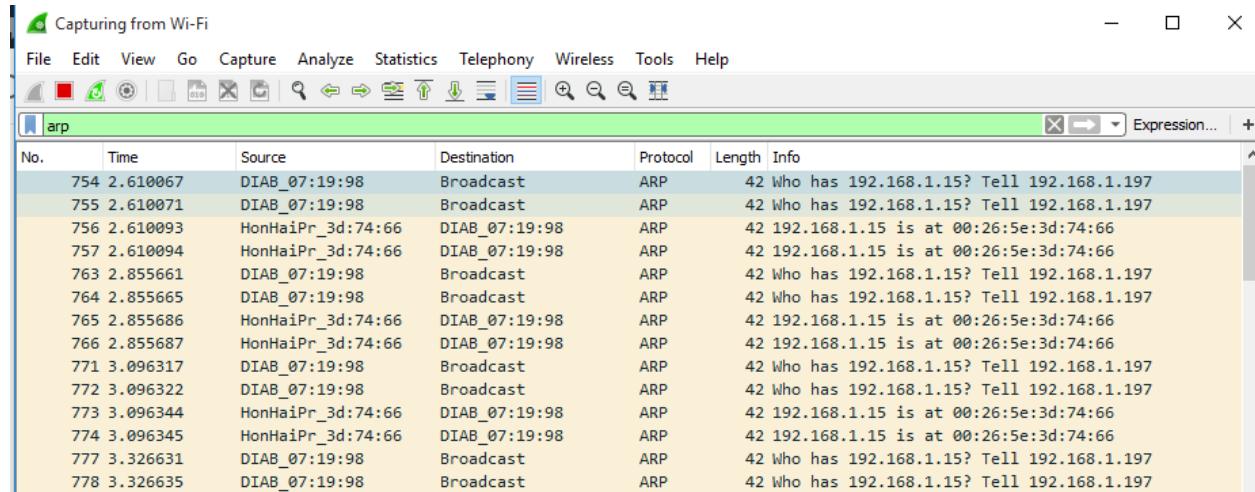
Bây giờ mình sẽ cho gửi các bản tin ARP_request đến cho máy tính để xem đã nhận được gói tin phản hồi và lưu vào bảng chưa nhé (các bạn vào cmd -> gõ ipconfig để lấy ip của máy tính, như máy tính mình là 192.168.1.15)

```
UART_putString("IOT47.com - ENC28J60 tutorial !!!\r\n");
ENC29J600_ini();
while (1)
{
    NET_loop();
    ARP_send_request(ip_dest);
    delay_ms(100);
}
```

Virtual Terminal

```
IOT47.com - ENC28J60 tutorial !!!  
Dang khai tao ENC28J60 ...  
  
Your MAC addr = 00:00:20:07:19:98  
Gui gon tin arp request  
Da nhan 1 arp reponse  
Da luu ip 192.168.1.15 = 00:26:5E:3D:74:66  
Gui gon tin arp request  
Da nhan 1 arp reponse  
IP da ton tai trong bang  
Gui gon tin arp request  
Da nhan 1 arp reponse  
IP da ton tai trong bang  
Gui gon tin arp request
```

Mình đã nhận thành công MAC của máy tính có IP 192.168.1.15



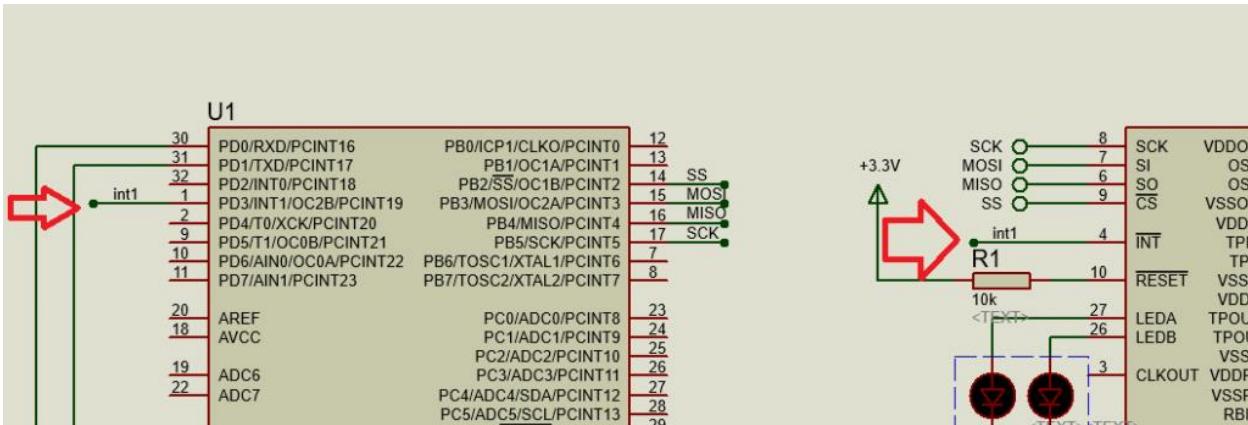
The screenshot shows the Wireshark interface with a capture window titled "arp". The packet list shows multiple ARP requests and responses. The columns include No., Time, Source, Destination, Protocol, Length, and Info. Most packets are ARP requests from "DIAB_07:19:98" to "Broadcast". One response is from "HonHaiPr_3d:74:66" to "DIAB_07:19:98". The "Info" column contains details like "Who has 192.168.1.15? Tell 192.168.1.197". The "Length" column shows values like 42 and 46.

No.	Time	Source	Destination	Protocol	Length	Info
754	2.610067	DIAB_07:19:98	Broadcast	ARP	42	Who has 192.168.1.15? Tell 192.168.1.197
755	2.610071	DIAB_07:19:98	Broadcast	ARP	42	Who has 192.168.1.15? Tell 192.168.1.197
756	2.610093	HonHaiPr_3d:74:66	DIAB_07:19:98	ARP	42	192.168.1.15 is at 00:26:5e:3d:74:66
757	2.610094	HonHaiPr_3d:74:66	DIAB_07:19:98	ARP	42	192.168.1.15 is at 00:26:5e:3d:74:66
763	2.855661	DIAB_07:19:98	Broadcast	ARP	42	Who has 192.168.1.15? Tell 192.168.1.197
764	2.855665	DIAB_07:19:98	Broadcast	ARP	42	Who has 192.168.1.15? Tell 192.168.1.197
765	2.855686	HonHaiPr_3d:74:66	DIAB_07:19:98	ARP	42	192.168.1.15 is at 00:26:5e:3d:74:66
766	2.855687	HonHaiPr_3d:74:66	DIAB_07:19:98	ARP	42	192.168.1.15 is at 00:26:5e:3d:74:66
771	3.096317	DIAB_07:19:98	Broadcast	ARP	42	Who has 192.168.1.15? Tell 192.168.1.197
772	3.096322	DIAB_07:19:98	Broadcast	ARP	42	Who has 192.168.1.15? Tell 192.168.1.197
773	3.096344	HonHaiPr_3d:74:66	DIAB_07:19:98	ARP	42	192.168.1.15 is at 00:26:5e:3d:74:66
774	3.096345	HonHaiPr_3d:74:66	DIAB_07:19:98	ARP	42	192.168.1.15 is at 00:26:5e:3d:74:66
777	3.326631	DIAB_07:19:98	Broadcast	ARP	42	Who has 192.168.1.15? Tell 192.168.1.197
778	3.326635	DIAB_07:19:98	Broadcast	ARP	42	Who has 192.168.1.15? Tell 192.168.1.197

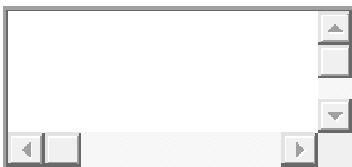
Phần mềm Wireshark cũng đã bắt được toàn bộ gói tin gửi nhận

Nhận các gói tin bằng ngắt

ENC28J60 hỗ trợ chân ngắt để báo có gói tin mới, khi có 1 gói tin mới chân ngắt sẽ kéo xuống mức 0, mình sẽ tận dụng điều này để nhận gói tin cho thuận tiện thay vì hỏi vòng ở trong **while(1)**, mình sẽ nối chân ngắt này với ngắt ngoài 1 của vi điều khiển ATmega328



Các bạn viết thêm hàm khởi tạo ngắt ngoài 1 trong file main



```

1void INT1_init()
2{
3 DDRD.3=0;// input mode
4 EICRA=0x08; //INT1 Mode: Falling Edge
5 EIMSK=0x02;
6 EIFR=0x02; //kich hoạt ngắt chan int1
7}

```

Nhớ gọi nó trong hàm main để khởi tạo ngắt nhé

Chúng ta sẽ gọi hàm **NET_loop** trong ngắt



```

1// External Interrupt 1 service routine
2interrupt [EXT_INT1] void ext_int1_isr(void)
3{
4 NET_loop();
5}

```

```

43 // SPI initialization
44 // SPI Type: Master
45 // SPI Clock Rate: 4000,000 kHz
46 // SPI Clock Phase: Cycle Start
47 // SPI Clock Polarity: Low
48 // SPI Data Order: MSB First
49
50 SPCR=0x50;
51 SPSR=0x00;
52
53 UART_init();
54 INT1_init();
55
56 UART_putString("IOT47.com - ENC28J60 tutorial !!!\r\n");
57 ENC29J600_ini();
58
59 #asm("sei")
60 while (1)
61 {
62     ARP_send_request(ip_dest);
63     delay_ms(500);
64 }
65

```

Virtual Terminal

```

IOT47.com - ENC28J60 tutorial !!!
Dang khai tao ENC28J60 ...

Your MAC addr = 00:00:20:07:19:98
Gui gon tin arp request
Gui gon tin arp request
Da nhan 1 arp reponse
Da luu ip 192.168.1.15 = 00:26:5E:3D:74:66
Gui gon tin arp request
Da nhan 1 arp reponse
IP da ton tai trong bang

```

Mọi thứ vẫn hoạt động trơn tru

Tới lúc này các bạn có thể xóa các dòng `UART_putString` ở các file `arp.c net.c enc28j60.c` đi để tránh in debug nhiều rồi mất !

Vẫn đè về thời gian tồn tại các bảng lưu trữ arp

Trong mạng LAN, IP của các thiết bị không cố định mà thường hay thay đổi, do đó, mình sẽ xóa các bảng lưu thường xuyên để tránh các ip lỗi thời. Thời gian này có thể từ 1 phút đến vài phút hoặc vài giờ. Mình sẽ sử dụng timer1 để tạo ra 1 bộ hẹn giờ xóa bảng lưu trong vòng 10 phút

Trước tiên mình tạo thêm hàm phục vụ clear bảng ở trong thư viện arp



```
1 uint32_t ARP_clear_time;
2 void ARP_clear_table(void)
3 {
4     uint8_t *end;
5     uint8_t *start;
6     if(++ARP_clear_time > COUNT_TICK)
7     {
8         ARP_clear_time=0;
9         //xoá bảng
10    start = (uint8_t *)&ARP_table;
11    end = start + sizeof(ARP_table);
12    for(;start<end;start++)
13        *start=0;
14    }
15}
```

Thêm định nghĩa thời gian xóa vào file .h



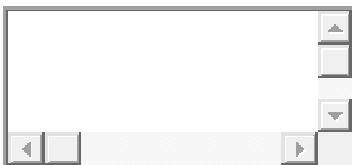
```
1//định nghĩa thời gian xóa bảng arp
2#define MAX_TIME_SAVE 10 //(phút)
3//-----
4#define COUNT_TICK (MAX_TIME_SAVE*60000)
5//-----
```

Trong file main, mình tạo file khởi tạo time1



```
1 uint32_t sys_tick;  
2 void TIMER1_init_1ms()  
3 {  
4     sys_tick=0;  
5     TCCR1B=0x02; // timer 1 2Mhz  
6     TIMSK1=0x01; // Timer/Counter 1 Interrupt(s) initialization  
7 }
```

Và gọi hàm xóa bảng trong ngắt



```
1 interrupt [TIM1_OVF] void timer1_ovf_isr(void) // Timer1 overflow interrupt service routine  
2 {  
3     TCNT1H=0xF8;  
4     TCNT1L=0x2D;  
5     sys_tick++;  
6     ARP_clear_table();  
7 }
```

Kết luận

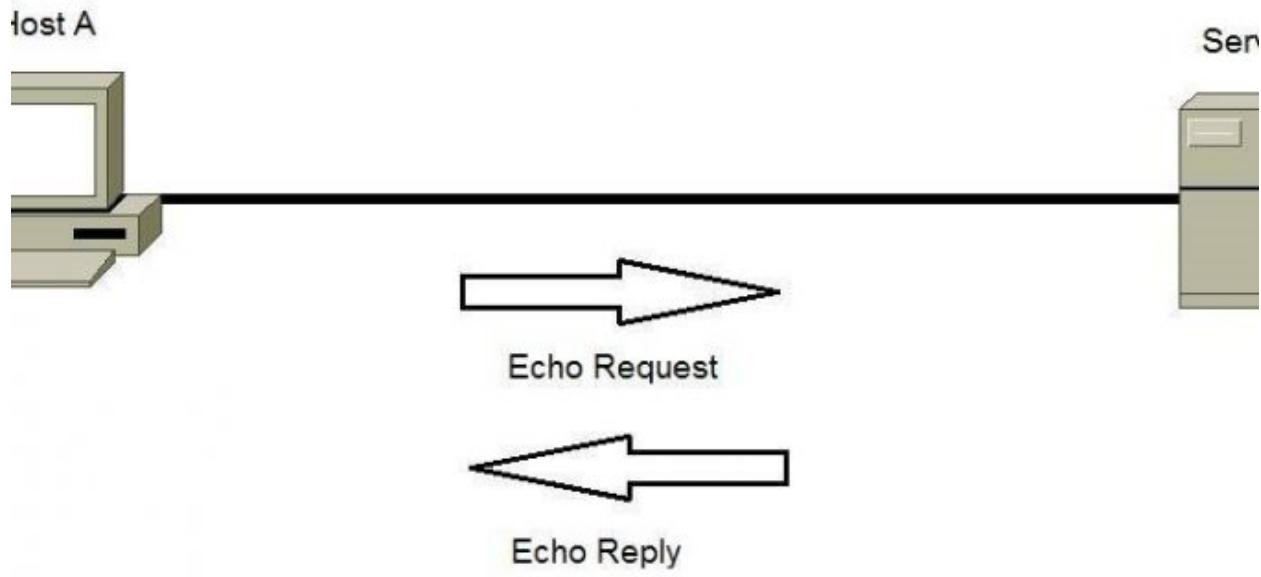
Như vậy chúng ta đã hoàn thành chương trình gửi nhận các gói tin ARP, ARP là giao đầu tiên chúng ta cần phải hoàn thành vì có nó chúng ta mới sử dụng tiếp các giao thức trong bộ giao thức IP được

Các bạn tải source cho bài này [tại đây](#)

Trong phần tiếp của chuỗi tutorila tự học lập trình enc28j60, chúng ta sẽ làm quen với giao thức ICMP và học cách trả lời lệnh ping

[ENC28J60] Bài 9: Giao thức ICMP

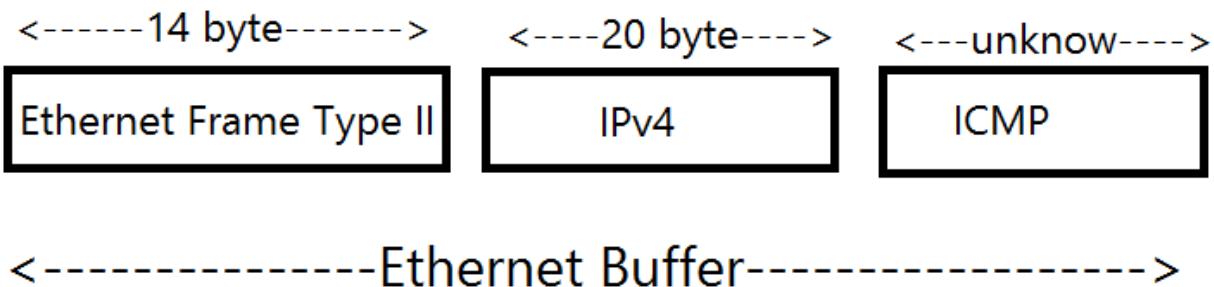
5 Tháng Tư, 2020 Đào Nguyễn Ethernet, IoT tutorial 3



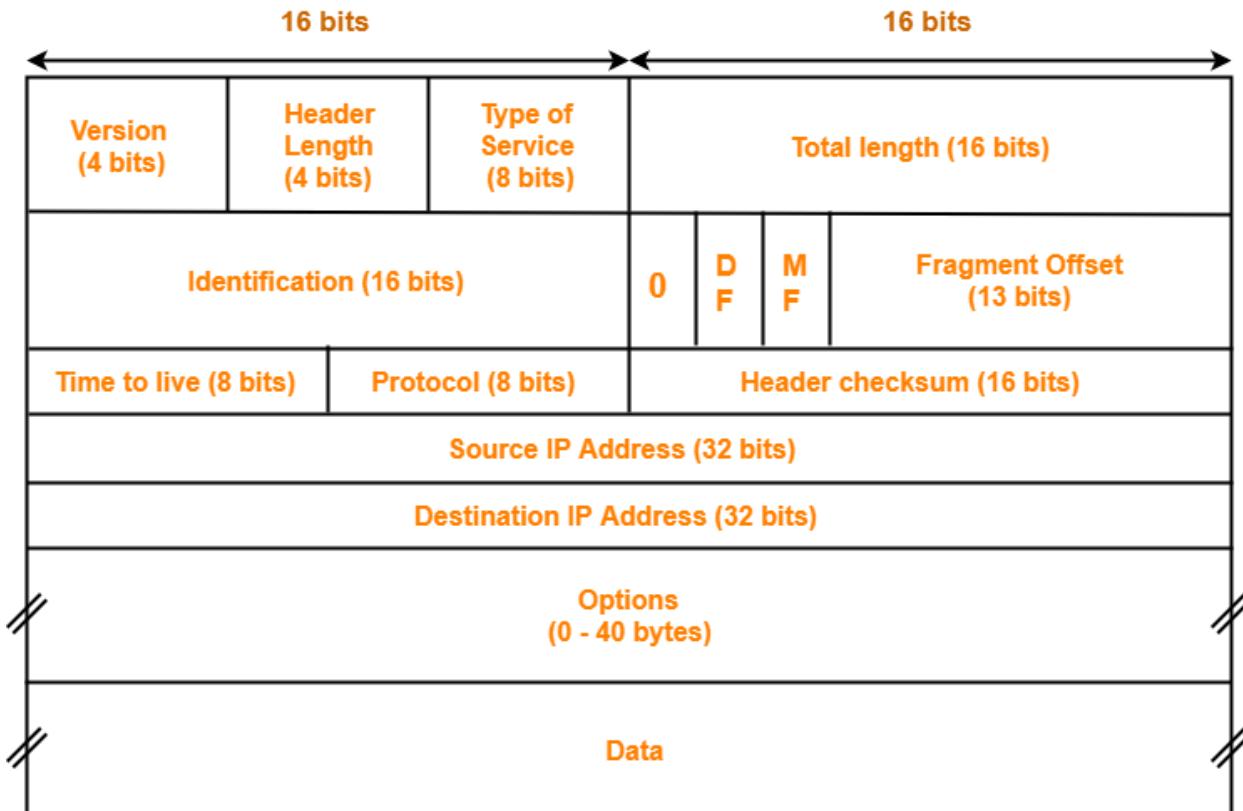
Tiếp tục trong bài học này, chúng ta sẽ tìm hiểu về giao thức **ICMP**

Xem bài trước : [Gói tin ARP](#)

Nếu như các bạn đã biết tới Ping, thì đó chính là **ICMP** đó.



Giao thức **ICMP** (Internet Control Message Protocol) **ICMP** được dùng để thông báo các lỗi xảy ra trong quá trình truyền đi của các gói dữ liệu trên mạng. Nó là 1 giao thức của IP (Internet Protocol). Do vậy, để làm việc với nó, chúng ta phải tìm hiểu về cấu trúc của các gói IP trước, cụ thể ở đây là **IPv4 Protocol**



IPv4 Header

IP Protocol có Ether Type là 0x0800 .Các bạn có thể tìm hiểu **kĩ hơn** các thuộc tính của từng trường, còn trong bài này, chúng ta chỉ cần quan tâm tới:

- Trường Protocol để xác định xem gói data này có phải là **ICMP** không !
- Source IP và Dest IP (đương nhiên phải quan tâm rồi)
- Data

1. Trường Protocol

Protocol = 0x01 => Đây là gói ICMP

```

> Frame 953357: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
> Ethernet II, Src: HonHaiPr_3d:74:66 (00:26:5e:3d:74:66), Dst: DIAB_07:19:98 (00:00:20:07:19:98)
< Internet Protocol Version 4, Src: 192.168.1.15, Dst: 192.168.1.197
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        Total Length: 60
        Identification: 0x2ad8 (10968)
    > Flags: 0x0000
        Time to live: 128
    > Protocol: ICMP (1)
        Header checksum: 0x8bc4 [validation disabled]
        [Header checksum status: Unverified]
        Source: 192.168.1.15
        Destination: 192.168.1.197
    0000  00 00 20 07 19 98 00 26  5e 3d 74 66 08 00 45 00  ... . . . & ^=tf..E.
    0010  00 3c 2a d8 00 00 80  01  8b c4 c0 a8 01 0f c0 a8  .<*....+ ..... .
    0020  01 c5 08 00 44 c7 00 01  08 94 61 62 63 64 65 66  ....D... . . abcdef
    0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76  ghiJKLMNOPQRSTUV
    0040  77 61 62 63 64 65 66 67  68 69  wabcdeFG hi

Byte 23: Protocol (ip.proto)  ↗

```

Close Help

Sử dụng phần mềm WireShark phân tích rất tiện lợi, vị trí byte protocol trong gói tin là 23

2. Trường Source IP và Dest IP

```

00 00 20 07 19 98 00 26  5e 3d 74 66 08 00 45 00  ... . . . & ^=tf..E.
00 3c 2a d8 00 00 80  01  8b c4 c0 a8 01 0f c0 a8  .<*....+ ..... .
01 c5 08 00 44 c7 00 01  08 94 61 62 63 64 65 66  ....D... . . abcdef
67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76  ghiJKLMNOPQRSTUV
77 61 62 63 64 65 66 67  68 69  wabcdeFG hi

Source IP   Dest IP
ICMP

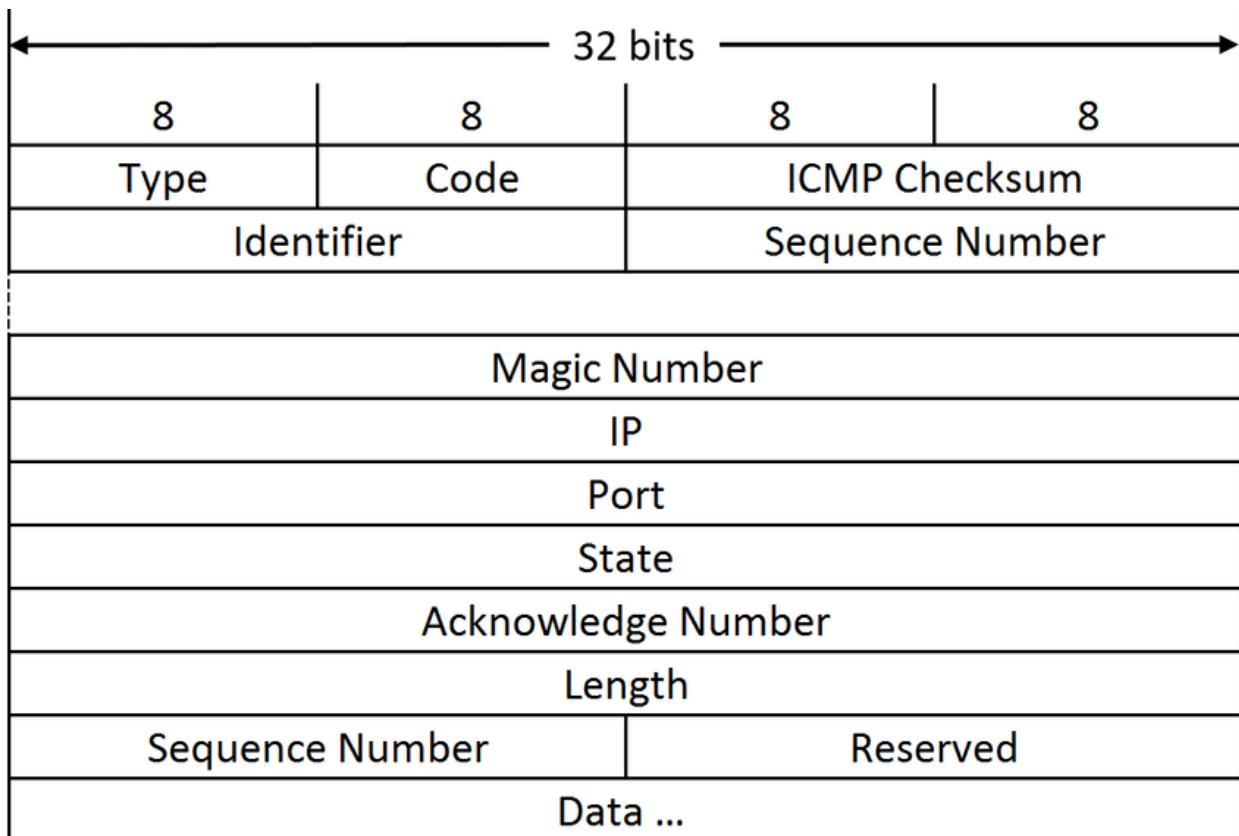
```

Tương tự, với source ip vị trí bắt đầu là 26
Với dest ip vị trí bắt đầu là 30

3. Trường data

Trường data chính là gói ICMP

Bây giờ chúng ta phân tích kỹ hơn về cấu trúc của gói ICMP



Chúng ta quan tâm tới các trường sau:

- Type
- Data

Nếu Type = 0x08 thì đây là 1 gói ping hỏi, còn type = 0x00 thì đây là ping trả lời, vì số lượng **kiểu** của ICMP quá nhiều nên trong bài này, mình chỉ hướng dẫn các bạn xử lý type = 0x08 (ping hỏi) còn mấy cái type khác chưa hỗ trợ nhé, chứ ngồi viết hết thì thốn lắm

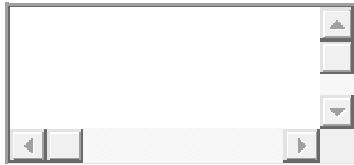
Trường Data chính là nội dung của tin nhắn

Để trả lời 1 gói ping thì chúng ta trả lời lại đúng cái nội dung tin nhắn đã nhận được là ok

Lí thuyết có vậy thôi, hết rồi đó, bắt tay vào code luôn

Do ICMP thuộc bộ giao thức IP, nên ta phải kiểm tra gói tin gửi tới có phải là gói IP không đã

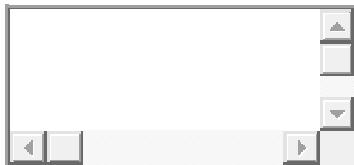
Trong thư viện `net.c` ngay dưới phần kiểm tra gói tin ARP, nếu gói tin đó không phải gói ARP, chúng ta tiếp tục check xem có phải là gói IP không



```
1 //kiem tra xem co phai goi tin IP khong
2 else if(net_buffer[12] == 0x08 && net_buffer[13] == 0x00)
3 {
4     //check ip
5     //UART_putString("Day la goi tin IP\r\n");
6     NET_ipRead(net_buffer,len);
7 }
```

Sau khi đã biết đây là gói tin IP, chúng ta tiếp tục kiểm tra xem nó có phải là gói ICMP không (hay là TCP UDP)

Mình sẽ tạo 1 hàm trên là `NET_ipRead` để kiểm tra



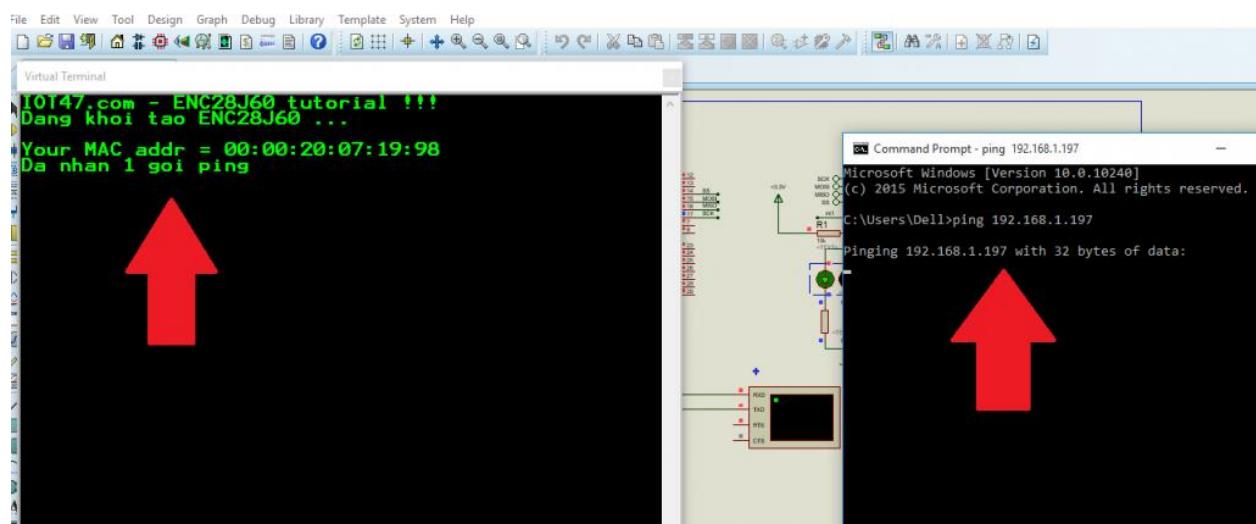
```
1 void NET_ipRead(uint8_t *IP_Frame,uint16_t len)
2 {
3     //chung ta se kiem tra xem goi tin do thuoc loai nao tcp hay udp hay icmp
4     if(IP_Frame[23] == 0x01) //it is ICMP packet
5     {
6         if(IP_Frame[34] == 8) //ping
7         {
8             UART_putString("Da nhan 1 goi ping\r\n");
9         }
10    }
11
12}
```

Làm tới đâu test tới đó, ngay bây giờ mình sẽ ping thử tới enc28j60 nhé ! Các bạn chạy mô phỏng và mở cmd gõ **ping 192.168.1.197**

Trong đó 192.168.1.197 là ip tĩnh của enc28j60 mà ta đã cài ở file **enc28j60.c**

```
C:\Users\Dell>ping 192.168.1.197
```

```
Pinging 192.168.1.197 with 32 bytes of data:
```



Toet vời ! Chúng ta đã bắt được 1 gói ping

Bây giờ viết mã để phản hồi cho gói tin ping này nào !

Tiếp tục tạo file thư viện **icmp.h** và **icmp.c** và add vào project

Copy nội dung khởi tạo cho file **icmp.h**



```
1 #ifndef ICMP_H_  
2 #define ICMP_H_  
3 //-----  
4 //Include cac thu vien can thiet  
5 #include <mega328p.h>  
6 #include <delay.h>  
7 #include <string.h>  
8 #include <stdlib.h>
```

```

9 #include <stdint.h>
10#include <stdio.h>
11#include <spi.h>
12#include "uart.h"
13#include "net.h"
14//-----
15
16//-----
17#endif /* ICMP_H_ */

```

Copy nội dung khởi tạo cho file `icmp.c`



```

1#include "icmp.h"
2extern const uint8_t macaddr[6];
3extern const uint8_t ip[4];
4extern uint8_t debug_string[60];
5
6//-
7//End file

```

Đồng thời add thư viện `icmp.h` vào file `net.h`

Sử dụng Struct để xử lí các gói tin 1 cách tiện lợi

Hiện tại mình đang sử dụng truy xuất array để xử lí các gói tin, nó ổn, nhưng không tiện lợi bằng **struct**.
Đương nhiên các bạn phải khá thành thạo **con trỏ** để có thể kết hợp nó với **struct** 1 cách nhuần nhuyễn. Mình sẽ code sử dụng lẩn lộn giữa array và struct dàn rồi sẽ chuyển dàn hoàn toàn sang xài struct cho tiện, vì sau này xử lí các gói tin TCP nó phức tạp lắm không dễ như mấy cái ARP với ICMP này đâu 😊

Mình sẽ khai báo 1 struct vào file `icmp.h` có cấu trúc tương tự như của 1 ICMP frame



```

1 typedef struct
2 {
3     uint8_t MAC_dich[6]; //-----
4     uint8_t MAC_nguon[6]; // | => It is Ethernet Frame II
5     uint16_t Ethernet_type; //-----
6
7     uint8_t Header_length; //-----| => IP
8     uint8_t Services; // |
9     uint16_t TotoLength; // |
10    uint16_t Identification; // |
11    uint16_t Flag; // |
12    uint8_t TimeToLive; // |
13    uint8_t Protocol; // |
14    uint16_t CheckSum; // |
15    uint8_t SourceIP[4]; // |
16    uint8_t DestIP[4]; //-----
17
18    uint8_t ICMP_Type; //-----| //ICMP
19    uint8_t ICMP_Code; // |
20    uint16_t ICMP_checkSum; // |
21    uint16_t ICMP_Identifier; // |
22    uint16_t ICMP_Squenquer; // |
23    uint8_t *data; //-----|
24 }ICMP_struct;

```

Struct phía trên là cấu trúc của 1 gói ping (gồm cả request và và reponse)

Sau khi nhận được gói tin, mình sẽ phang thẳng gói tin ấy vào struct bằng cách ép kiểu

Ở bên trên mình có kiểm tra xem có phải gói ping không ở trong file [net.c](#) (hàm [NET_ipRead](#)) mục đích là để test nhanh thôi chứ công việc này sẽ xử lí ở file [icmp.c](#). Do vậy các bạn xóa dòng kiểm tra này đi và gọi hàm [ICMP_read](#) để cho hàm này xử lí thì hơn

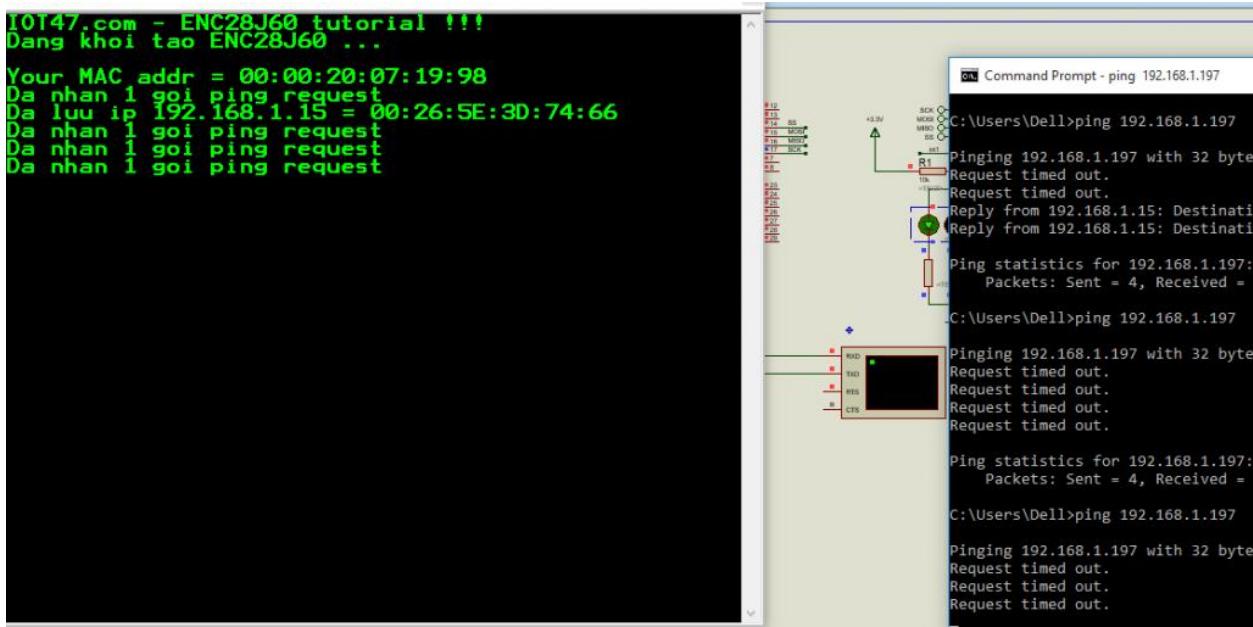
```
1 void NET_ipRead(uint8_t *IP_Frame,uint16_t len)
2 {
3     //chung ta se kiem tra xem goi tin do thuoc loai nao tcp hay udp hay icmp
4     if(IP_Frame[23] == 0x01) //it is ICMP packet
5     {
6         ICMP_read(IP_Frame,len);
7     }
8 }
```

Chúng ta sẽ viết hàm **ICMP_read** trong file **icmp.c** nhé



```
1 void ICMP_read(uint8_t *ICMP_Frame,uint16_t len)
2 {
3     ICMP_struct *ICMP_Ping_Packet = (ICMP_struct *)ICMP_Frame;
4
5     //kiem tra dia chi ip xem co phai no gui cho minh khong
6     if( memcmp(ICMP_Ping_Packet->DestIP,ip,4) )return; // dung memcmp de so sanh, neu khac thi thoat
7
8     //kiem tra xem co phai ping request khong ?
9     if(ICMP_Ping_Packet->ICMP_Type == 0x08)
10    {
11        UART.putString("Da nhan 1 goi ping request\r\n");
12        //phan hoi lai goi ping
13    }
14}
```

Mình sẽ test xem code đến chỗ này đã chạy ok chưa !



Đã bắt được gói ping, các bạn có thấy xài Struct như này chương trình nhìn thân thiện và chuyên nghiệp hơn hẳn không 😊

Nếu không hiểu dòng `ICMP_struct *ICMP_Ping_Packet = (ICMP_struct *)ICMP_Frame;`; thì đã đến lúc bạn dùng bài viết và đi tìm hiểu về con trỏ và struct rồi đó 😊

Gửi gói ping trả lời

Giờ chúng ta sẽ viết mã để gửi lại phản hồi

Mình sẽ tạo 1 hàm khác để gửi ping phản hồi



```
1void ICMP_ping_reply(ICMP_struct * ping_packet,uint16_t len)
2{
3
4}
```

Nhiệm vụ trong hàm phản hồi:

1. Đảo lộn Source Mac và Dest Mac lại
2. Sửa Type 0x08 (hỏi) thành 0x00 (trả lời)
3. Sửa lại checksum cho gói icmp

4. Gửi đi

Viết hàm tính checksum

Checksum là gì ?

Checksum về cơ bản là một giá trị được tính từ gói dữ liệu để kiểm tra tính toàn vẹn của nó. Thông qua **checksum**, chúng ta có thể kiểm tra xem dữ liệu nhận được có lỗi hay không. Điều này là do trong khi di chuyển trên mạng, một gói dữ liệu có thể bị hỏng và phải có một cách để biết rằng dữ liệu có bị hỏng hay không. Đây là lý do **checksum** được thêm vào header của mỗi gói tin. Ở phía nguồn gửi, checksum được tính toán và đặt trong tiêu đề dưới dạng một trường. Ở phía đích, checksum lại được tính toán và kiểm tra chéo với giá trị tổng kiểm tra hiện có trong tiêu đề để xem liệu gói dữ liệu có ổn hay không.

Cách tính checksum

Công tất cả các byte 16bit lại với nhau (nếu lẻ thì đắp thêm 0x00 cho chẵn), nếu kết quả > 16bit thì lại tiếp tục cộng 16bit lại cho đến khi được 1 byte 16bit hoàn chỉnh, sau đó lấy nghịch đảo

Ví dụ, 1 gói ICMP có nội dung như sau:

0x00, 0x00, **0x49, 0x66**, 0x00, 0x01, 0x0B, 0xF5, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66,
0x67, 0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f, 0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76,
0x77, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69

Thì phần mình to đỗ chính là 2 byte check sum. Chúng ta ghép các byte 8 thành byte 16 rồi cộng hết lại với nhau (vì đang tính checksum nên coi checksum=0x0000)

0x0000 + 0x0001 + 0x0BF5 + 0x6162 + 0x6364 + 0x6566 + 0x6768 + 0x696A + 0x6B6C + 0x6D6E +
0x6F70 + 0x7172 + 0x7374 + 0x7576 + 0x7761 + 0x6263 + 0x6465 + 0x6667 + 0x6869 = **0x6B693**

Tiếp tục cộng kết quả bằng các byte 16

Tức 0x0006 + 0xB693 = **0xB699**

Lấy nghịch đảo => checksum = **0x4966**

Trong file **icmp.c** mình sẽ viết hàm tính checksum



```
1 uint16_t ICMP_checksum(ICMP_struct *icmp_packet,uint16_t icmp_len )
2 {
3     uint32_t checksum=0;
4     uint8_t *ptr;
5     icmp_packet->ICMP_checkSum=0; //reset check sum
```

```

6  icmp_len=34; //bo qua truong ethernet va truong ip
7  ptr = &icmp_packet->ICMP_Type;
8  while(icmp_len>1) //cong het cac byte16 lai
9  {
10    checksum += (uint16_t) (((uint32_t)*ptr<<8)|*(ptr+1));
11    ptr+=2;
12    icmp_len-=2;
13 }
14 if(icmp_len) checksum+=((uint32_t)*ptr)<<8; //neu con le 1 byte
15 while (checksum>>16) checksum=(uint16_t)checksum+(checksum>>16);
16
17 //nghich dao bit
18 checksum=~checksum;
19
20 //hoan vi byte thap byte cao
21 return ( ((checksum>>8)&0xFF) | ((checksum<<8)&0xFF00) );
22}

```

Giải thích: Mình giảm len đi 34 vì chúng ta không tính checksum cho các byte thuộc gói ethernet và gói ip (nó chiếm 34 byte). Con trả ptr bắt đầu từ vị trí của trường Type của gói ICMP

Vòng lặp while sẽ ghép 2 byte cạnh nhau thành byte lớn 16bit rồi cộng dồn lại, sau đó kiểm tra nếu lẻ 1 byte thì nhét thêm 0x00 vào đít bằng lệnh dịch lên cao và cộng lại

Sau khi cộng tổng xong thì kiểm tra tổng này có lớn hơn 16bit không để cộng lại tiếp đến khi con 16bit thì thôi.

Sau đó nghịch đảo byte checksum lại, giá trị checksum trong struct cũng bị lộn thứ tự (byte thấp đứng trước byte cao) nên mình hoán vị byte lại

Viết hàm ICMP reply



```

1 void ICMP_ping_reply(ICMP_struct * ping_packet,uint16_t len)
2 {
3  uint8_t buffer_temp[6]; //buff trung gian phuc vu hoan doi

```

```

4
5 //hoan vi MAC
6 memcpy(buffer_temp,ping_packet->MAC_nguon,6);      //copy mac nguon vao bo nho tam
7 memcpy(ping_packet->MAC_nguon,ping_packet->MAC_dich,6); //copy mac dich vao mac nguon
8 memcpy(ping_packet->MAC_dich,buffer_temp,6);
9
10 //hoan vi IP
11 memcpy(buffer_temp,ping_packet->SourceIP,4);      //copy ip nguon vao bo nho tam
12 memcpy(ping_packet->SourceIP,ping_packet->DestIP,4); //copy ip dich vao mac nguon
13 memcpy(ping_packet->DestIP,buffer_temp,4);
14
15 //make packet reply
16 ping_packet->ICMP_Type = 0x00; //type = reply
17 ping_packet->ICMP_checkSum = ICMP_checksum(ping_packet,len);
18
19 NET_SendFrame((uint8_t *)ping_packet,len); //gui phan hoi
20}

```

Quá rõ ràng rồi, không cần giải thích nữa nhỉ 😊

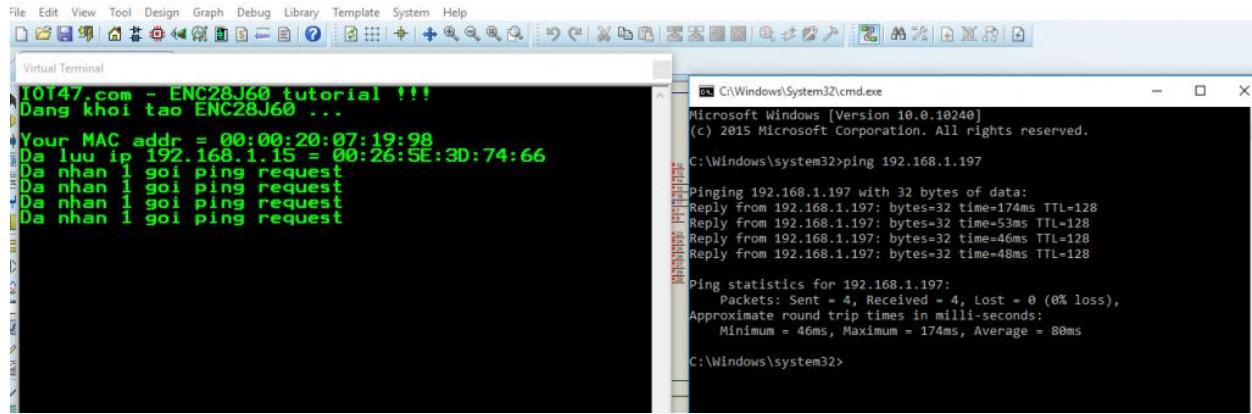
Đừng quên gọi hàm **reply** trong hàm **ICMP_read** nhé !

```

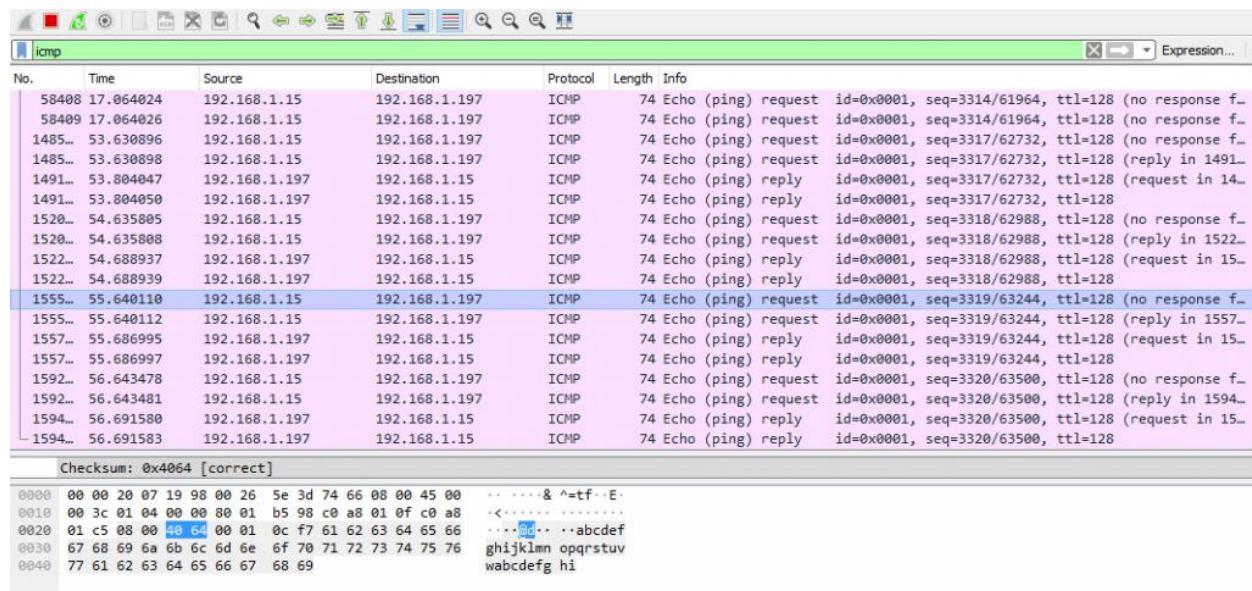
//kiem tra xem co phai ping request khong ?
if(ICMP_Ping_Packet->ICMP_Type == 0x08)
{
    UART.putString("Da nhan 1 goi ping request\r\n");
    //phan hoi lai goi ping
    ICMP_ping_reply(ICMP_Ping_Packet,len);
}

```

Giờ chạy mô phỏng và ping thử tới enc28j60 của chúng ta thôi



Máy tính đã nhận được ping phản hồi không trượt phát nào 😊



WireShark cũng bắt được cái gói ping request và ping reply

Dowload

Toàn bộ soure cho bài giao thức icmp các bạn có thể tải [tại đây](#)

Trong **bài tiếp theo**, chúng ta sẽ tìm hiểu về giao thức UDP, cũng là 1 giao thức nằm trong IP và làm 1 project đó là điều khiển led từ xa qua mạng lan bằng UDP

[ENC28J60] Bài 10: Giao thức UDP

6 Tháng Tư, 2020 Đào Nguyễn Ethernet, IoT tutorial 5

+	Bits 0 - 15	16 - 31
0	Source Port	Destination Port
32	Length	Checksum
64	Data	

Tiếp tục **bài trước** trong bài này, chúng ta sẽ tìm hiểu về giao thức UDP. **UDP (User Datagram Protocol)** là một trong những giao thức cốt lõi của **giao thức TCP/IP**. Dùng UDP, chương trình trên **mạng máy tính** có thể gửi những dữ liệu ngắn được gọi là **datagram** tới máy khác.

Giao thức UDP không cung cấp sự tin cậy và thứ tự truyền nhận mà **TCP** làm, các gói dữ liệu có thể đến không đúng thứ tự hoặc bị mất mà không có thông báo. Tuy nhiên UDP nhanh và hiệu quả hơn đối với các mục tiêu nhu cầu kích thước nhỏ và yêu cầu khắt khe về thời gian.

Do bản chất không trạng thái của nó nên nó hữu dụng đối với việc trả lời các truy vấn nhỏ với số lượng lớn người yêu cầu.

Cổng

UDP dùng cổng để cho phép các giao tiếp giữa các ứng dụng diễn ra.

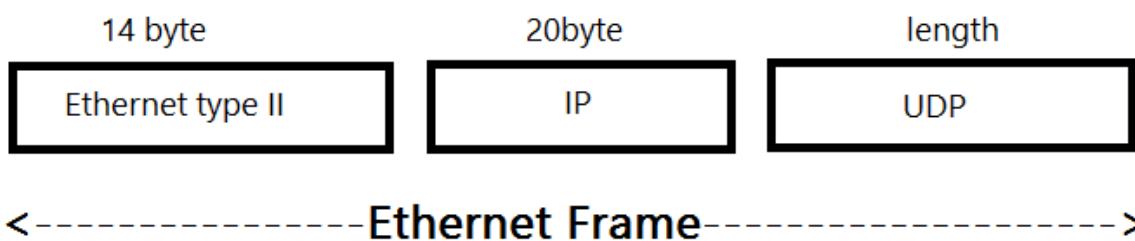
Cổng dùng 16 **bit** để đánh địa chỉ, vì vậy số của cổng nằm trong khoảng 0 đến 65.535. Cổng 0 được để dành và không nên sử dụng.

Cổng từ 1 đến 1023 được gọi là cổng “well-known” và trên các **hệ điều hành** tựa **Unix**, việc gắn kết tới một trong những cổng này đòi hỏi quyền **root**.

Cổng 1024 đến 49.151 là **cổng đã đăng ký**.

Cổng từ 49.152 đến 65.535 là các cổng tạm, được dùng chủ yếu bởi client khi liên lạc với server.

Cấu trúc chung



Nhận dạng

UDP có mã Protocol (thuộc gói IP) là 0x11

Cấu trúc gói UDP

UDP là giao thức hướng thông điệp nhỏ nhất của **tầng giao vận**

7	Tầng ứng dụng	HTTP, SMTP, SNMP, FTP, Telnet, ECHO, SIP, SSH, NFS, RTSP, XMPP, Whois, ENRP
6	Tầng trình diễn	XDR, ASN.1, SMB, AFP, NCP
5	Tầng phiên	ASAP, TLS, SSH, ISO 8327 / CCITT X.225, RPC, NetBIOS, ASP
4	Tầng giao vận	TCP, UDP, RTP, SCTP, SPX, ATP, IL
3	Tầng mạng	IP, ICMP, IGMP, IPX, BGP, OSPF, RIP, IGRP, EIGRP, ARP, RARP, X.25
2	Tầng liên kết dữ liệu	Ethernet, Token ring, HDLC, Frame relay, ISDN, ATM, 802.11 WiFi, FDDI, PPP
1	Tầng vật lý	10BASE-T, 100BASE-T, 1000BASE-T, SONET/SDH, T-carrier/E-carrier, các tầng vật lý khác thuộc 802.11

Vị trí của UDP trong các tầng

UDP không đảm bảo cho các tầng phía trên thông điệp đã được gửi đi và người gửi cũng không có trạng thái thông điệp UDP một khi đã được gửi

+	Bits 0 - 15	16 - 31
0	Source Port	Destination Port
32	Length	Checksum
64	Data	

Cấu trúc của gói UDP

Không có gì để giải thích, cấu trúc của các gói UDP rất đơn giản

Chúng ta sẽ bắt tay vào viết thư viện udp luôn. Tiếp tục tạo các file `udp.h` và `udp.c`

Sao chép mã khởi tạo cho file .h



```
1 ifndef UDP_H_
2 define UDP_H_
3 //-----
4 //Include cac thu vien can thiet
5 #include <mega328p.h>
6 #include <delay.h>
7 #include <string.h>
8 #include <stdlib.h>
9 #include <stdint.h>
10#include <stdio.h>
11#include <spi.h>
12#include "uart.h"
13#include "net.h"
14
15//-----
16
17
18//-----
19#endif /* UDP_H_ */
```

Tiếp tục mã khởi tạo cho file .c



```
1 #include "udp.h"
2
3 extern const uint8_t macaddr[6];
4 extern const uint8_t ip[4];
5 extern uint8_t debug_string[60];
```

```
6  
7  
8  
9  
10//-----  
11//end file
```

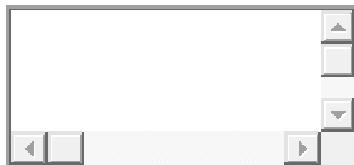
Khởi tạo cấu trúc cho frame ở file .h



```
1 typedef struct  
2 {  
3     uint8_t MAC_dich[6]; //-----|  
4     uint8_t MAC_nguon[6]; // | => It is Ethernet Frame II  
5     uint16_t Ethernet_type; //-----|  
6  
7     uint8_t Header_length; //-----| => IP  
8     uint8_t Services; // |  
9     uint16_t TotoLength; // |  
10    uint16_t Identification; // |  
11    uint16_t Flag; // |  
12    uint8_t TimeToLive; // |  
13    uint8_t Protocol; // |  
14    uint16_t CheckSum; // |  
15    uint8_t SourceIP[4]; // |  
16    uint8_t DestIP[4]; //-----|  
17  
18    uint16_t UDP_Source_Port;//-----| //UDP  
19    uint16_t UDP_Dest_Port; // |  
20    uint16_t UDP_Length; // |  
21    uint16_t UDP_Checksum; // |
```

```
22 uint8_t data[]; //-----|
23 }UDP_struct;
```

Trong thư viện `net.c` tại hàm `NET_ipRead` chúng ta kiểm tra trường Protocol để xem có phải gói UDP, nếu đúng thì ném gói này cho thư viện `udp.c` xử lí



```
1 else if(IP_Frame[23] == 0x11) //it is UDP packet
2 {
3     UDP_read(IP_Frame,len);
4 }
```

(nhớ thêm nguyên mẫu hàm vào file `udp.h` và add thư viện `udp.h` vào file `net.h`)

```
void NET_ipRead(uint8_t *IP_Frame,uint16_t len)
{
    //chung ta se kiem tra xem goi tin do thuoc loai nao tcp hay udp hay icmp
    if(IP_Frame[23] == 0x01) //it is ICMP packet
    {
        ICMP_read(IP_Frame,len);
    }
    else if(IP_Frame[23] == 0x11) //it is UDP packet
    {
        UDP_read(IP_Frame,len);
    }
}
```

Bây giờ mình sẽ tạo hàm `UDP_read` trong thư viện `udp.c`

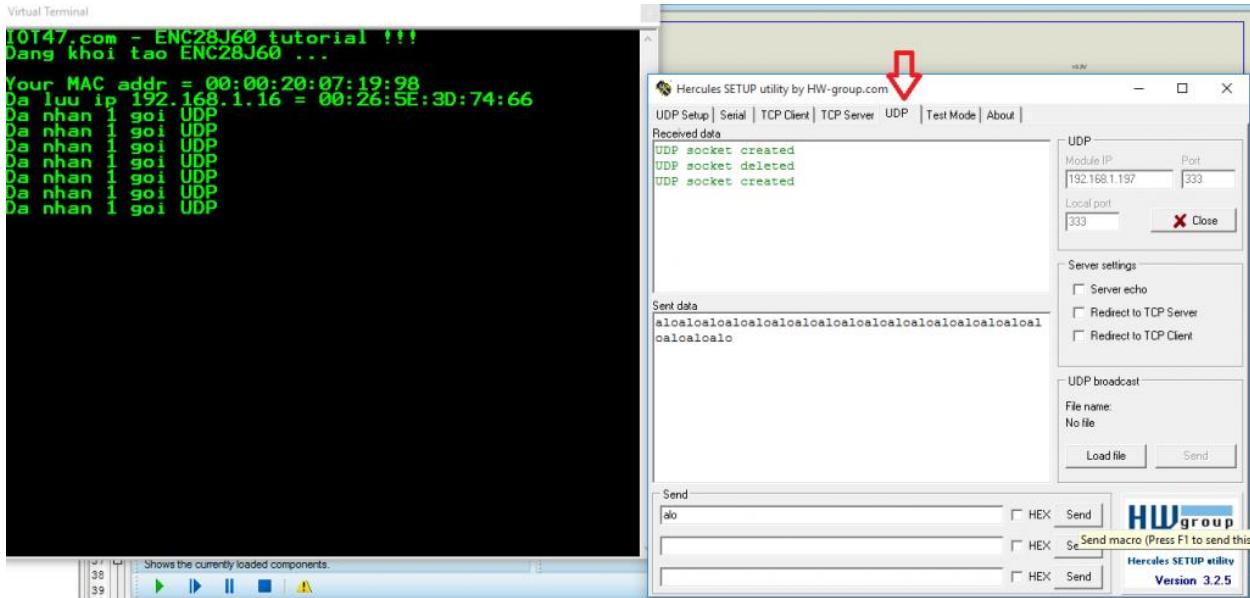


```
1 void UDP_read(uint8_t *UDP_Frame,uint16_t len)
2 {
3     UDP_struct *UDP_Struct_Frame = (UDP_struct *)UDP_Frame;
4
5     //kiem tra dia chi ip xem co phai no gui cho minh khong
6     if( memcmp(UDP_Struct_Frame->DestIP,ip,4) )return; // dung memcmp de so sanh, neu khac thi thoat
```

7

```
8 UART.putString("Da nhan 1 goi UDP\r\n");
9}
```

Mình sẽ chạy thử mô phỏng và cho phần mềm Hercules gửi thử các gói UDP đến ENC28J60 xem chúng ta đã nhận được gói UDP nào chưa



Toet vời ! Đã bắt được các gói UDP

Ngoài IP , UDP còn sử dụng thêm Port (cổng) để xác định nguồn và đích nhận dữ liệu ! Mình sẽ cho ENC28J60 cho phép nhận tin nhắn từ tất cả các cổng ! Do vậy chúng ta chỉ cần quan tâm Source Port (với tin gửi đến ENC28J60) và Dest Port (với tin mà ENC28J60 gửi đi)

Chúng ta sẽ đọc trường Length để xác định độ dài của tin nhắn gửi đến và in ra màn hình debug nhé.

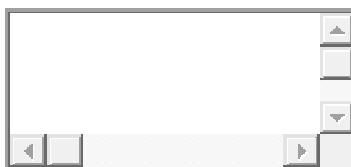
Các bạn mở file `net.h` rồi thêm vào 2 #define này nhé



```
1#define swap16(a) (((a)>>8)&0xff)|(((a)<<8)&0xff00))
2#define swap32(a) (((a)>>24)&0xff)|(((a)>>8)&0xff00)|(((a)<<8)&0xff0000)|(((a)<<24)&0xff000000))
```

Do các biến kiểu 16 và 32 trong struct có byte thấp đứng trước byte cao nên mình #define thêm macro để đảo ngược nó lại

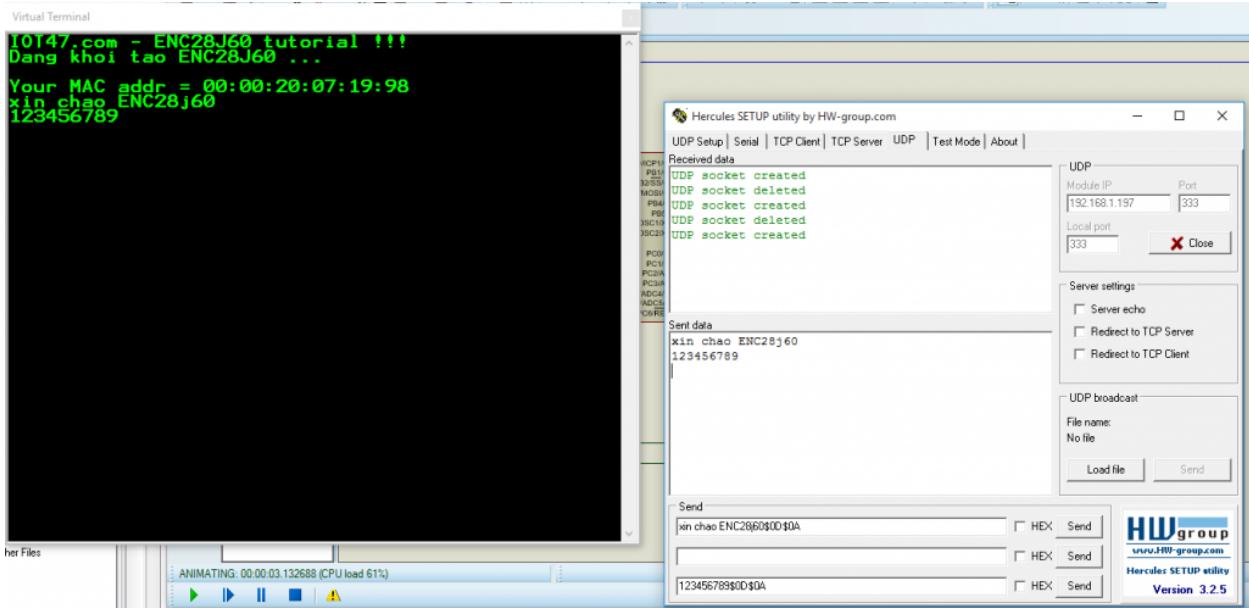
Tiếp tục với hàm **UDP_read**, mình sẽ in nội dung của tin nhắn ra



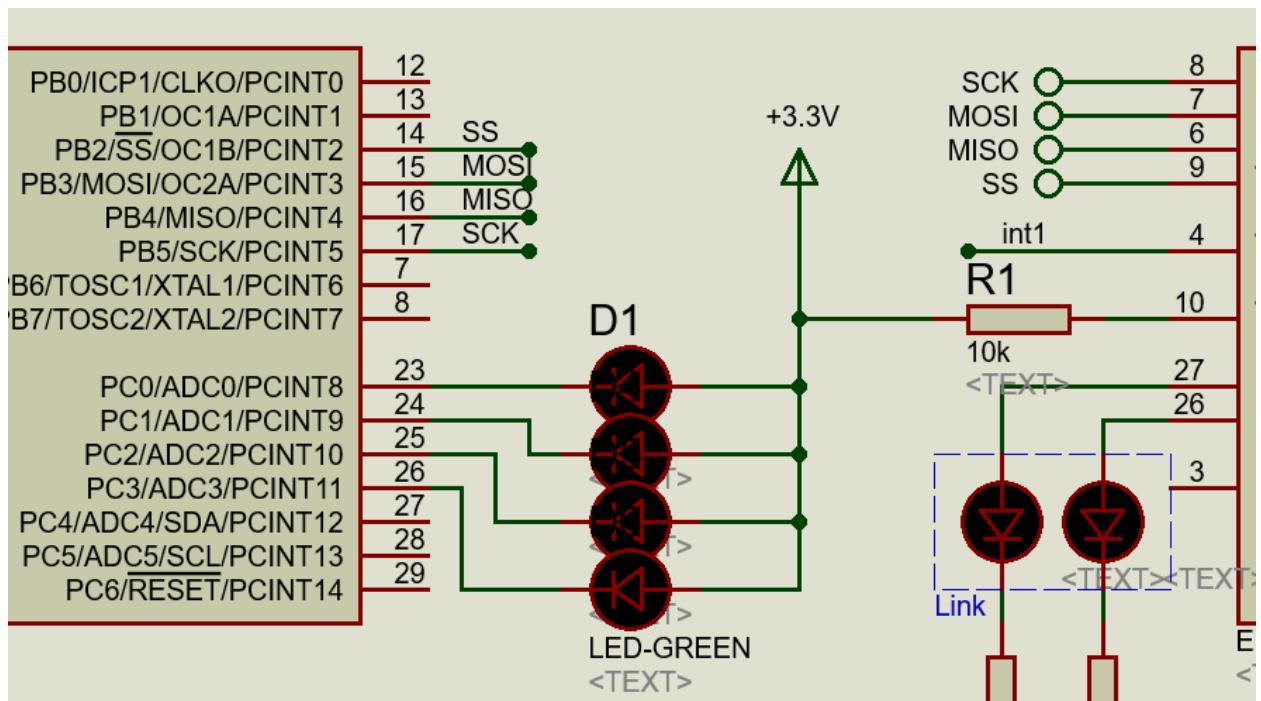
```
IUART_putString(UDP_Struct_Frame->data);

void UDP_read(uint8_t *UDP_Frame,uint16_t len)
{
    UDP_struct *UDP_Struct_Frame = (UDP_struct *)UDP_Frame;
    UDP_Struct_Frame1= *(UDP_struct *)UDP_Frame;
    //kiem tra dia chi ip xem co phai no gui cho minh khong
    if( memcmp(UDP_Struct_Frame->DestIP,ip,4) )return; // dung memcmp de so sanh, neu khac thi thoat

    UART_putString(UDP_Struct_Frame->data);
}
```



Mình sẽ vẽ thêm 4 con LED vào để test thử chức năng điều khiển LED nhé



Mình sẽ xài hàm strstr để check



```

1 if(strstr(UDP_Struct_Frame->data,"LED1=ON"))PORTC.0=0;
2 else if(strstr(UDP_Struct_Frame->data,"LED1=OF"))PORTC.0=1;
3
4 else if(strstr(UDP_Struct_Frame->data,"LED2=ON"))PORTC.1=0;
5 else if(strstr(UDP_Struct_Frame->data,"LED2=OF"))PORTC.1=1;
6
7 else if(strstr(UDP_Struct_Frame->data,"LED3=ON"))PORTC.2=0;
8 else if(strstr(UDP_Struct_Frame->data,"LED3=OF"))PORTC.2=1;
9
10 else if(strstr(UDP_Struct_Frame->data,"LED4=ON"))PORTC.3=0;
11 else if(strstr(UDP_Struct_Frame->data,"LED4=OF"))PORTC.3=1;

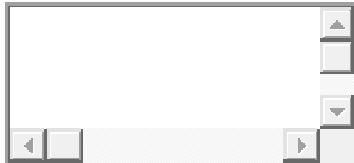
```

Và test gửi tin nhắn điều khiển bằng phần mềm Hercules

Xây dựng hàm gửi tin nhắn UDP

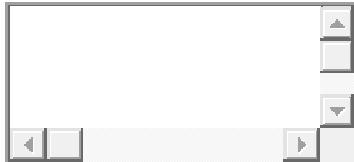
Để gửi gói tin UDP tới 1 địa chỉ IP, chúng ta cần biết MAC tương ứng của IP đó. Do vậy, mình sẽ check IP trong bàn ARP_table, nếu có rồi thì lấy xài, nếu chưa có thì phát hành bản tin ARP_request để lấy IP của nó

Mình sẽ khai báo 1 cổng mặc định để cho ENC28j60 gửi đi là cổng 20798. Các bạn thêm #define cho nó vào file **udp.h**



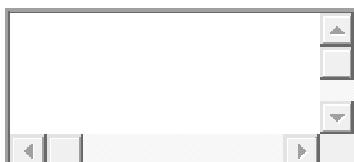
```
1#define ENC28J60_UDP_PORT 20798
```

Trong thư viện **arp.c** mình sẽ viết thêm API để lấy MAC trong bảng



```
1 void ARP_table_get_MAC(uint8_t *ip_check,uint8_t * MAC_dest)
2{
3 int i=ARP_table_checkIP(ip_check) - 1;
4 memcpy(MAC_dest,ARP_table[i].mac,6);
5}
```

Đừng quên thêm nguyên mẫu hàm cho nó vào file **arp.h**. Chúng ta sẽ bắt tay vào viết hàm **UDP_send**



```
1 uint8_t UDP_send(uint8_t *IP_dest,uint16_t port_dest,uint8_t *data,uint32_t timeout)
2 {
3     uint32_t count=0;
4     uint16_t length_mess,frame_length;
5     UDP_struct UDP_Struct_Frame;
6     length_mess = strlen(data);
7     while(1)
```

```

8  {
9    if(ARP_table_checkIP(IP_dest) != -1)break; // da nhan dc MAC
10   ARP_send_request(IP_dest);
11   delay_ms(50);
12   count+=50;
13   if(count >= timeout)
14   {
15     UART_putString("Chua gui goi tin udp\r\n");
16     return 0; //gui that bai
17   }
18 }

19

20 //make Ethernet II
21 ARP_table_get_MAC(IP_dest,UDP_Struct_Frame.MAC_dich); //mac cua thang nhan
22 memcpy(UDP_Struct_Frame.MAC_nguon,macaddr,6); //mac cua enc28j60
23 UDP_Struct_Frame.Ethernet_type = 0x0008; // Type = 0x800 = IP
24
25 //make IP packet
26 UDP_Struct_Frame.Header_length = 0x45;
27 UDP_Struct_Frame.Services=0x00;
28 UDP_Struct_Frame.TotoLength=swap16(length_mess+8+20);
29 UDP_Struct_Frame.Identification=0x2111;
30 UDP_Struct_Frame.Flag=0x0000;
31 UDP_Struct_Frame.TimeToLive=0x80;
32 UDP_Struct_Frame.Protocol=0x11; //UDP
33 UDP_Struct_Frame.CheckSum=0x0000;
34 memcpy(UDP_Struct_Frame.SourceIP,ip,4); //ip cua enc28j60
35 memcpy(UDP_Struct_Frame.DestIP,IP_dest,4); //ip cua thang nhan
36 //tinh checksum goi ip
37 UDP_Struct_Frame.CheckSum=NET_ipchecksum((uint8_t *)&UDP_Struct_Frame.Header_length);
38
39
40 //make UDP packet

```

```

41 UDP_Struct_Frame.UDP_Source_Port = swap16(ENC28J60_UDP_PORT);
42 UDP_Struct_Frame.UDP_Dest_Port = swap16(port_dest);
43 UDP_Struct_Frame.UDP_Length = swap16(length_mess + 8);
44 UDP_Struct_Frame.UDP_Checksum=0x0000;
45 strcpy((char*)UDP_Struct_Frame.data,data); //copy data to struct
46 //tinh checksum cho udp
47 UDP_Struct_Frame.UDP_Checksum = UDP_checksum(&UDP_Struct_Frame);
48
49 frame_length = swap16(UDP_Struct_Frame.UDP_Length) + 34;
50 NET_SendFrame((uint8_t *)&UDP_Struct_Frame,frame_length); //gui goi tin udp
51 return 1; //da gui
52}

```

Giải thích: Để gửi các gói tin UDP đi, mình sẽ phát hành một broadcast (ARP request) tới địa chỉ IP nhận, chờ nó phản hồi để lấy MAC, nếu không thấy phản hồi (hết time out) thì không gửi. Sau khi có phản hồi MAC. Chúng ta make gói tin và gửi đi

Trong hàm gửi đi, mình có viết thêm 2 hàm tính checksum cho gói IP và gói UDP. Giao thức UDP không quan trọng checksum, nếu không cần checksum, các bạn có thể để checksum mặc định = 0x00, nhưng vì đang học nên chúng ta cứ viết hàm tính cho nó nhé

Tính checksum

Trước tiên là hàm checksum cho gói IP, các bạn quay trở lại file [net.c](#) để tạo thêm hàm checksum cho gói IP. Nội dung các hàm checksum nó ý chang nhau, chăng qua địa chỉ bắt đầu và độ dài của mỗi loại checksum khác nhau 1 tí thôi, nên mình không giải thích lại hàm checksum nữa !



```

1 uint16_t NET_ipchecksum(uint8_t *IP_packet_start)
2 {
3     uint32_t checksum=0;
4     uint16_t length=20;
5     while(length) //cong het cac byte16 lai
6     {
7         checksum += (uint16_t) (((uint32_t)*IP_packet_start<<8)*(IP_packet_start+1));

```

```

8     IP_packet_start+=2;
9         length-=2;
10    }
11    while (checksum>>16) checksum=(uint16_t)checksum+(checksum>>16);
12    //nghịch dao bit
13    checksum=~checksum;
14    //hoan vi byte thap byte cao
15    return swap16(checksum);
16}

```

Nhớ khai báo nguyên mẫu hàm vào file [net.h](#) nhé nhé

Tiếp tục viết thêm hàm checksum cho gói UDP ở file [udp.c](#)

Checksum của gói UDP gồm: IP Protocol (0x11) + IP source + IP dest + toàn bộ UDP packet + UDP packet length



```

1 uint16_t UDP_checksum(UDP_struct *UDP_Struct_Frame)
2 {
3     uint32_t checksum;
4     uint8_t *ptr;
5     uint16_t length;
6     UDP_Struct_Frame->UDP_Checksum=0; //reset check sum
7     length = swap16(UDP_Struct_Frame->UDP_Length) + 8;
8     ptr = (uint8_t *)&UDP_Struct_Frame->SourceIP;
9
10    checksum=0x11 + length-8;
11    while(length>1) //cong het cac byte16 lai
12    {
13        checksum += (uint16_t) (((uint32_t)*ptr<<8)|*(ptr+1));
14        ptr+=2;
15        length-=2;

```

```

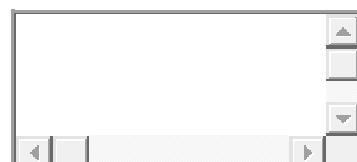
16 }
17 if(length) checksum+=((uint32_t)*ptr)<<8; //neu con le 1 byte
18 while (checksum>>16) checksum=(uint16_t)checksum+(checksum>>16);
19 //nghich dao bit
20 checksum=~checksum;
21 //hoan vi byte thap byte cao
22 return swap16(checksum);
23}

//-----
15
16 typedef struct
17 {
18     uint8_t MAC_dich[6];      //-----| => It is Ethernet Frame II
19     uint8_t MAC_nguon[6];    //-----|
20     uint16_t Ethernet_type; //-----|
21
22     uint8_t Header_length;  //-----| => IP
23     uint8_t Services;       //-----|
24     uint16_t TotoLength;    //-----|
25     uint16_t Identification; //-----|
26     uint16_t Flag;          //-----|
27     uint8_t TimeToLive;     //-----|
28     uint8_t Protocol;       //-----|
29     uint16_t CheckSum;      //-----|
30     uint8_t SourceIP[4];    //-----|
31     uint8_t DestIP[4];      //-----|
32
33     uint16_t UDP_Source_Port; //-----| //UDP
34     uint16_t UDP_Dest_Port;   //-----|
35     uint16_t UDP_Length;     //-----|
36     uint16_t UDP_Checksum;   //-----|
37     uint8_t data[];          //-----| udp messenger
38 }UDP_struct;
39 //-----
40 #define ENC28J60_UDP_PORT 20798
41 //-----

```

tính checksum cho udp

OK rồi ! Giờ trong hàm nhận udp điều khiển LED mình sẽ phản hồi lại cho máy tính nhé



```

1     if strstr(UDP_Struct_Frame->data,"LED1=ON")){PORTC.0=0;UDP_send(UDP_Struct_Frame-
2 >SourceIP,swap16(UDP_Struct_Frame->UDP_Source_Port),"Da bat LED1\r\n",1000);}
3     else if strstr(UDP_Struct_Frame->data,"LED1=OF")){PORTC.0=1;UDP_send(UDP_Struct_Frame-
4 >SourceIP,swap16(UDP_Struct_Frame->UDP_Source_Port),"Da tat LED1\r\n",1000);}
5

```

```

6 else if(strcmp(UDP_Struct_Frame->data,"LED2=ON")){PORTC.1=0;UDP_send(UDP_Struct_Frame-
>SourceIP,swap16(UDP_Struct_Frame->UDP_Source_Port),"Da bat LED2\r\n",1000);}
7
8 else if(strcmp(UDP_Struct_Frame->data,"LED2=OF")){PORTC.1=1;UDP_send(UDP_Struct_Frame-
>SourceIP,swap16(UDP_Struct_Frame->UDP_Source_Port),"Da tat LED2\r\n",1000);}
9
10
11 else if(strcmp(UDP_Struct_Frame->data,"LED3=ON")){PORTC.2=0;UDP_send(UDP_Struct_Frame-
>SourceIP,swap16(UDP_Struct_Frame->UDP_Source_Port),"Da bat LED3\r\n",1000);}
12
13 else if(strcmp(UDP_Struct_Frame->data,"LED3=OF")){PORTC.2=1;UDP_send(UDP_Struct_Frame-
>SourceIP,swap16(UDP_Struct_Frame->UDP_Source_Port),"Da tat LED3\r\n",1000);}

14
15 else if(strcmp(UDP_Struct_Frame->data,"LED4=ON")){PORTC.3=0;UDP_send(UDP_Struct_Frame-
>SourceIP,swap16(UDP_Struct_Frame->UDP_Source_Port),"Da bat LED4\r\n",1000);}
16
17 else if(strcmp(UDP_Struct_Frame->data,"LED4=OF")){PORTC.3=1;UDP_send(UDP_Struct_Frame-
>SourceIP,swap16(UDP_Struct_Frame->UDP_Source_Port),"Da tat LED4\r\n",1000);}

```

Các bạn có thể tải source cho bài này [tại đây](#)

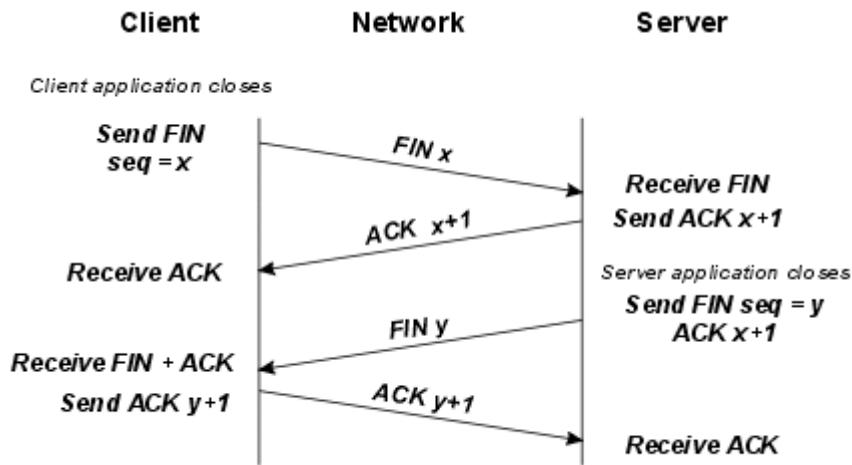
Như vậy, chúng ta đã hoàn thành các giao thức cơ bản, nếu hiểu hết các bài học tới thời điểm này, các bạn hoàn toàn có thể tự nghiên cứu để viết thêm TCP – giao thức chính, cũng là giao thức chúng ta cần nhất. Thực tế, giao thức TCP khó hơn các giao thức chúng ta đã học rất nhiều. Trong thời gian tới, mình sẽ cố gắng làm thêm về TCP cho các bạn tham khảo !

Lộ trình sắp tới sẽ là TCP Server -> TCP Client -> Socket -> MQTT

Các bạn giúp mình chia sẻ rộng rãi tutorial để mình có động lực viết thêm, và nếu thấy khóa học giúp ích ít nhiều cho các bạn, các bạn có thể donate ít mì tôm cho mình tại thông tin ở cuối website ! Rất cảm ơn sự ủng hộ của các bạn !

[ENC28J60] Bài 11: Giao thức TCP (lý thuyết)

8 Tháng Tư, 2020 Đào Nguyễn Ethernet, IoT tutorial 0



Trong **bài trước** chúng ta đã có thể làm việc với giao thức UDP, tiếp tục trong bài hôm nay sẽ là giao thức TCP, một giao thức cốt lõi của giao thức IP

Hoạt động của giao thức

Không như giao thức **UDP** – giao thức có thể lập tức gửi gói tin mà không cần thiết lập kết nối, TCP đòi hỏi thiết lập kết nối trước khi bắt đầu gửi dữ liệu và kết thúc kết nối khi việc gửi dữ liệu hoàn tất. Cụ thể, các kết nối TCP có ba pha:

1. Thiết lập kết nối
2. Truyền dữ liệu
3. Kết thúc kết nối

Thiết lập kết nối

Để thiết lập một kết nối, TCP sử dụng một quy trình **bắt tay** 3 bước (*3-way handshake*). Trước khi client thử kết nối với một server, server phải đăng ký một cổng và mở cổng đó cho các kết nối: đây được gọi là **mở bị động**. Một khi mở bị động đã được thiết lập thì một client có thể bắt đầu mở chủ động. Để thiết lập một kết nối, quy trình bắt tay 3 bước xảy ra như sau:

1. Client yêu cầu mở cổng dịch vụ bằng cách gửi gói tin SYN (gói tin TCP) tới server, trong gói tin này, tham số **sequence number** được gán cho một giá trị ngẫu nhiên **X**.
2. Server hồi đáp bằng cách gửi lại phía client bản tin SYN-ACK, trong gói tin này, tham số **acknowledgment number** được gán giá trị bằng **X + 1**, tham số **sequence number** được gán ngẫu nhiên một giá trị **Y**.

- Để hoàn tất quá trình **bắt tay ba bước**, client tiếp tục gửi tới server bản tin ACK, trong bản tin này, tham số **sequence number** được gán cho giá trị bằng X + 1 còn tham số **acknowledgment number** được gán giá trị bằng Y + 1

Tại thời điểm này, cả client và server đều được xác nhận rằng, một kết nối đã được thiết lập.

Truyền dữ liệu

Một số đặc điểm cơ bản của TCP để phân biệt với **UDP**:

- Truyền dữ liệu không lỗi (do có cơ chế sửa lỗi/truyền lại)
- Truyền các gói dữ liệu theo đúng thứ tự
- Truyền lại các gói dữ liệu mất trên đường truyền
- Loại bỏ các gói dữ liệu trùng lặp
- Cơ chế hạn chế tắc nghẽn đường truyền

Ở hai bước đầu tiên trong ba bước bắt tay, hai máy tính trao đổi một số thứ tự gói ban đầu (*Initial Sequence Number -ISN*). Số này có thể chọn một cách ngẫu nhiên. Số thứ tự này được dùng để đánh dấu các khối dữ liệu gửi từ mỗi máy tính. Sau mỗi byte được truyền đi, số này lại được tăng lên. Nhờ vậy ta có thể sắp xếp lại chúng khi tới máy tính kia bát ké các gói tới nơi theo thứ tự thế nào.

Trên lý thuyết, mỗi byte gửi đi đều có một số thứ tự và khi nhận được thì máy tính nhận gửi lại tin báo nhận (ACK). Trong thực tế thì chỉ có byte dữ liệu đầu tiên được gán số thứ tự trong trường số thứ tự của gói tin và bên nhận sẽ gửi tin báo nhận bằng cách gửi số thứ tự của byte đang chờ.

Ví dụ: Máy tính A gửi 4 byte với số thứ tự ban đầu là 100 (theo lý thuyết thì 4 byte sẽ có thứ tự là 100, 101, 102, 103) thì bên nhận sẽ gửi tin báo nhận có nội dung là 104 vì đó là thứ tự của byte tiếp theo nó cần. Bằng cách gửi tin báo nhận là 104, bên nhận đã ngầm thông báo rằng nó đã nhận được các byte 100, 101, 102 và 103. Trong trường hợp 2 byte cuối bị lỗi thì bên nhận sẽ gửi tin báo nhận với nội dung là 102 vì 2 byte 100 và 101 đã được nhận thành công.

Giả sử ta có 10.000 byte được gửi đi trong 10 gói tin 1.000 byte và có 1 gói tin bị mất trên đường truyền. Nếu gói bị mất là gói đầu tiên thì bên gửi sẽ phải gửi lại toàn bộ 10 gói vì không có cách nào để bên nhận thông báo nó đã nhận được 9 gói kia. Vấn đề này được giải quyết trong giao thức **SCTP** (*Stream Control Transmission Protocol – “Giao thức điều khiển truyền vận dòng”*) với việc bổ sung báo nhận chọn lọc.

Kết thúc kết nối

Để kết thúc kết nối hai bên sử dụng quá trình bắt tay 4 bước và chiều của kết nối kết thúc độc lập với nhau. Khi một bên muốn kết thúc, nó gửi đi một gói tin FIN và bên kia gửi lại tin báo nhận ACK. Vì vậy, một quá trình kết thúc tiêu biểu sẽ có 2 cặp gói tin trao đổi.

Một kết nối có thể tồn tại ở dạng “nửa mở”: một bên đã kết thúc gửi dữ liệu nên chỉ nhận thông tin, bên kia vẫn tiếp tục gửi.

Cấu trúc gói tin

+	Bิต 0 - 3	4 - 9	10 - 15	16 - 31
0	Source Port			Destination Port
32	Sequence Number			
64	Acknowledgement Number			
96	Data Offset	Reserved	Flags	Window
128	Checksum			Urgent Pointer
160	Options (optional)			
160/192+	Data			

Source port : Số hiệu của cổng tại máy tính gửi

Destination port : Số hiệu của cổng tại máy tính nhận.

Sequence number Trường này có 2 nhiệm vụ. Nếu cờ SYN bật thì nó là số thứ tự gói ban đầu và byte đầu tiên được gửi có số thứ tự này cộng thêm 1. Nếu không có cờ SYN thì đây là số thứ tự của byte đầu tiên

Acknowledgement number Nếu cờ ACK bật thì giá trị của trường chính là số thứ tự gói tin tiếp theo mà bên nhận cần.

Data offset Trường có độ dài 4 bit quy định độ dài của phần header (tính theo đơn vị từ 32 bit). Phần header có độ dài tối thiểu là 5 từ (160 bit) và tối đa là 15 từ (480 bit).

Reserved Dành cho tương lai và có giá trị là 0

Flags (hay Control bits) : URG ACK PSH RST SYN FIN

Window Số byte có thể nhận bắt đầu từ giá trị của trường báo nhận (ACK)

Checksum : 16 bit kiểm tra cho toàn bộ gói TCP và 1 phần của gói IP

Urgent pointer Nếu cờ URG bật thì giá trị trường này chính là số từ 16 bit mà số thứ tự gói tin (*sequence number*) cần dịch trái.

Options Đây là trường tùy chọn. Nếu có thì độ dài là bội số của 32 bit.

Dữ liệu Dữ liệu của gói TCP

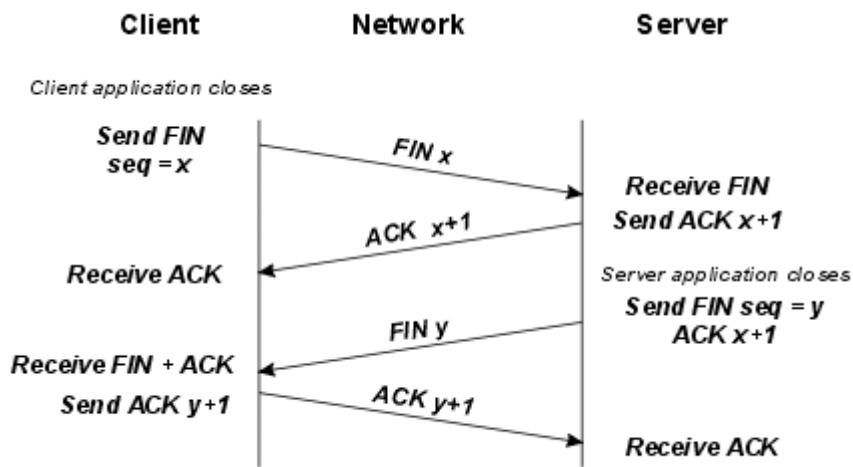
Nguồn : <https://vi.wikipedia.org/wiki/TCP>

Bài tiếp theo: Thiết lập kết nối TCP

Theo dõi toàn bộ bài học [tại đây](#)

[ENC28J60] Bài 12: Giao thức TCP – thiết lập kết nối

8 Tháng Tư, 2020 Đào Nguyễn Ethernet, IoT tutorial 0



Bạn nào chưa tìm hiểu về lí thuyết của giao thức TCP thì xem lại **bài trước** nhé !

Hãy chắc chắn vững vàng lí thuyết ở **bài trước** thì bài này mới hiểu vì mình không nhắc lại mà đi vào code luôn 😊

Khởi tạo thư viện tcp

Tiếp tục tạo 2 file **tcp.h** và **tcp.c** và copy nội dung khởi tạo và add file thư viện vào project

Nội dung khởi tạo cho file **tcp.h**

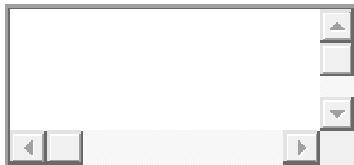
```
1 #ifndef TCP_H_
2 #define TCP_H_
3 //-----
4 //Include cac thu vien can thiet
5 #include <mega32p.h>
```

```

6 #include <delay.h>
7 #include <string.h>
8 #include <stdlib.h>
9 #include <stdint.h>
10#include <stdio.h>
11#include <spi.h>
12#include "uart.h"
13#include "net.h"
14
15
16//-----
17void TCP_read(uint8_t *TCP_Frame,uint16_t len);
18//-----
19#endif /* TCP_H_ */

```

Nội dung khởi tạo cho file **tcp.c**



```

1 #include "tcp.h"
2 extern const uint8_t macaddr[6];
3 extern const uint8_t ip[4];
4 extern uint8_t debug_string[60];
5
6 void TCP_read(uint8_t *TCP_Frame,uint16_t len)
7 {
8
9 }
10
11
12//-----
13//end file

```

Các bạn add thư viện `tcp.h` vào thư viện `net.h` và tiến hành kiểm tra gói IP có protocol = 0x06 ở hàm `NET_ipRead` để néo gói cho thư viện tcp xử lí

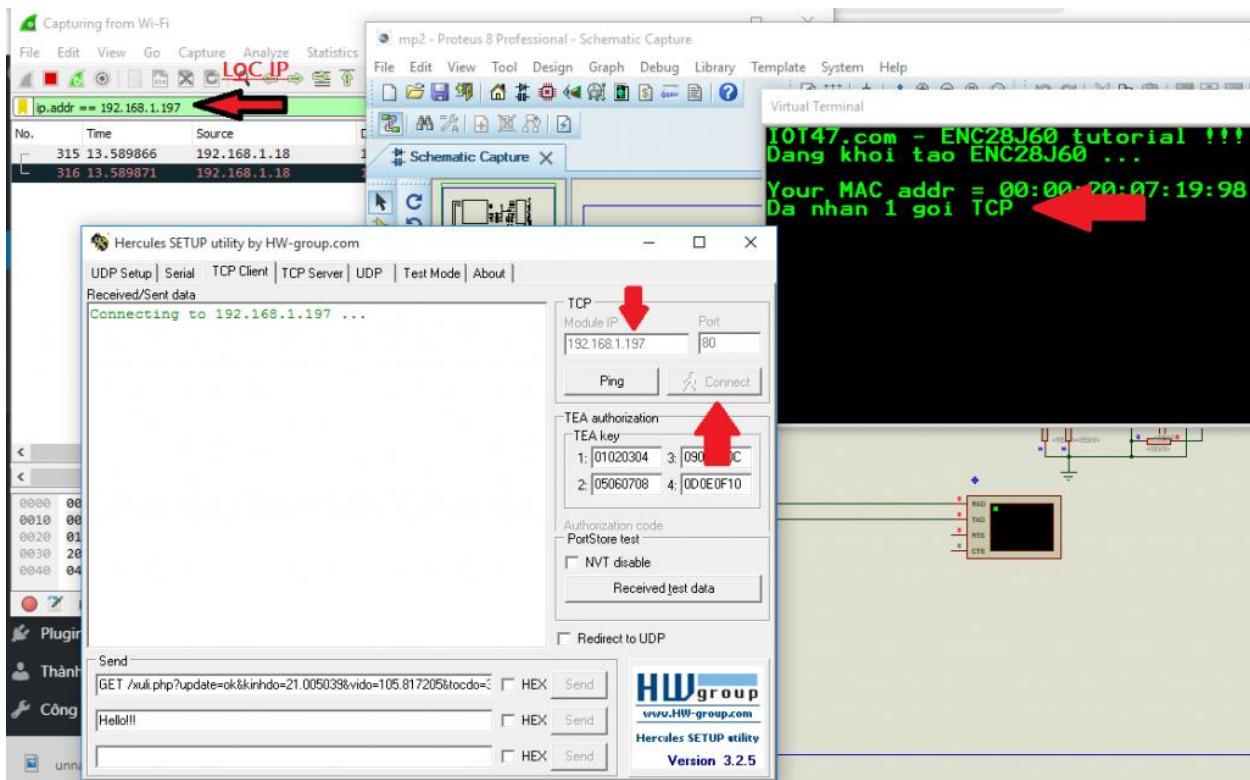
```
void NET_ipRead(uint8_t *IP_Frame, uint16_t len)
{
    //chung ta se kiem tra xem goi tin do thuoc loai nao tcp hay udp hay icmp
    if(IP_Frame[23] == 0x01) //it is ICMP packet
    {
        ICMP_read(IP_Frame, len);
    }
    else if(IP_Frame[23] == 0x11) //it is UDP packet
    {
        UDP_read(IP_Frame, len);
    }
    else if(IP_Frame[23] == 0x06) //it is TCP packet
    {
        TCP_read(IP_Frame, len);
    }
}
```

Vậy là xong quá trình khởi tạo mà chắc các bạn đã quá quen thuộc. Từ giờ là công việc của thư viện tcp
Thiết lập kết nối cho giao thức TCP

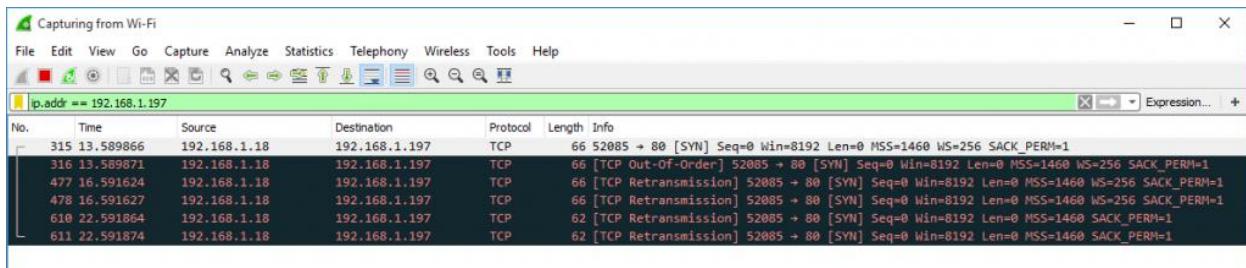
Chúng ta sẽ bắt xem đã nhận được gói tcp nào chưa nhé. Trong file hàm `TCP_read` mình thêm dòng debug

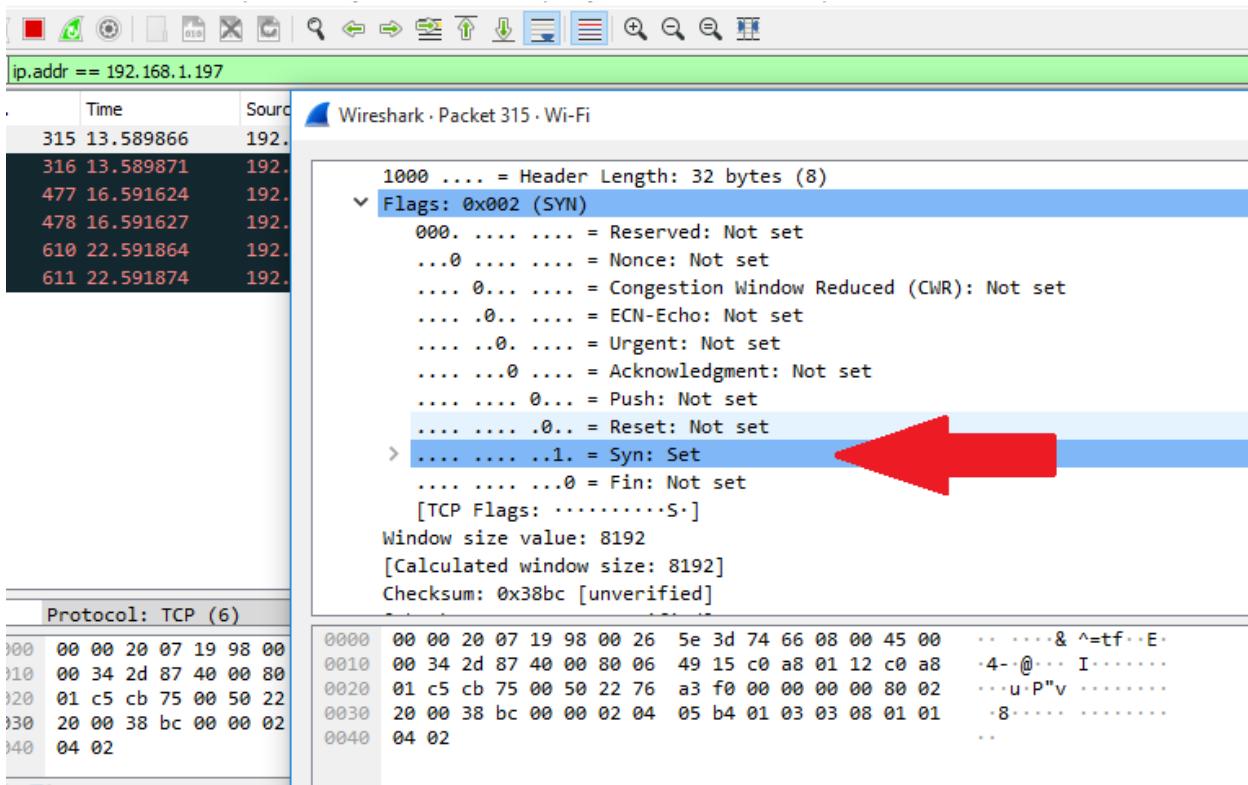


Chạy mô phỏng, mở Wireshark, chọn chế độ lọc IP để tránh show các gói không cần thiết, và mở phần mềm Hercules, tab TCP Client, điền IP của ENC28j60 là 192.168.1.197 và ấn Connect



Chúng ta đã bắt dc các gói TCP gửi đến khi ấn Connect, đồng thời WireShark cũng bắt được và Show cụ thể các gói tin ra





Click vào gói tin đầu tiên và mình đã thấy cơ Syn được SET, như vậy gói tin mở đầu của việc bắt tay đã được gửi tới cho ENC28J60. Ngay bây giờ chúng ta phải viết hàm để phản hồi lại gói tin này bằng các việc sau:

- Set cờ SYN và cờ ACK
- Trường **acknowledgment number** = **sequence number +1**
- **sequence number** = ngẫu nhiên

Mình sẽ tạo 1 Struct giống như đã tạo ở bài UDP và bỏ vào file tcp.h

```

1 //-----
2 typedef struct
3 {
4     uint8_t MAC_dich[6]; //-----
5     uint8_t MAC_nguon[6]; // | => It is Ethernet Frame II
6     uint16_t Ethernet_type; //-----
7

```

```

8  uint8_t Header_length; //-----| => IP
9  uint8_t Services; // | |
10 uint16_t TotoLength; // | |
11 uint16_t Identification; // | |
12 uint16_t Flag; // | |
13 uint8_t TimeToLive; // | |
14 uint8_t Protocol; // | |
15 uint16_t CheckSum; // | |
16 uint8_t SourceIP[4]; // | |
17 uint8_t DestIP[4]; //-----|
18
19 uint16_t Source_Port;//-----| //TCP
20 uint16_t Dest_Port; // | |
21 uint32_t Sequence_Number; // | |
22 uint32_t Acknowledgement; // | |
23 uint8_t data_offset; // | |
24 uint8_t TCP_Flags; // | |
25 uint16_t Window; // | |
26 uint16_t TCP_Checksums; // | |
27 uint16_t Urgent_Pointer; // | |
28 uint8_t data[]; //-----|
29 }TCP_struct;
30 //dinh nghia cac macro cho truong Flags
31 #define TCP_CWR 0x80
32 #define TCP_ECE 0x40
33 #define TCP_URG 0x20
34 #define TCP_ACK 0x10
35 #define TCP_PSH 0x08
36 #define TCP_RST 0x04
37 #define TCP_SYN 0x02
38 #define TCP_FIN 0x01
39 //-----|

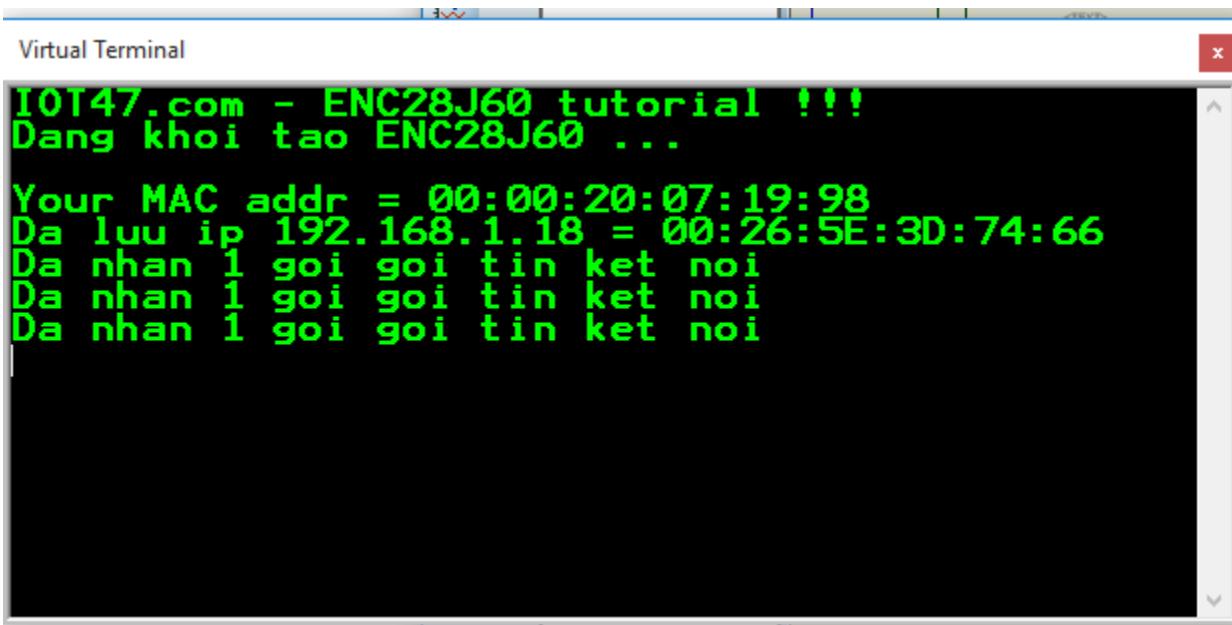
```

Bắt đầu viết mã cho hàm **TCP_read**, mình sẽ debug từng tí một, trước tiên là kiểm cờ SYN



```
1 void TCP_read(uint8_t *TCP_Frame,uint16_t len)
2 {
3     TCP_struct *TCP_Struct_Frame = (TCP_struct *)TCP_Frame;
4     //kiem tra dia chi ip xem co phai no gui cho minh khong
5     if( memcmp(TCP_Struct_Frame->DestIP,ip,4) )return; // dung memcmp de so sanh, neu khac thi thoat
6     if(TCP_Struct_Frame->TCP_Flags == TCP_SYN)
7     {
8         UART.putString("Da nhan 1 goi goi tin ket noi\r\n");
9     }
10}
```

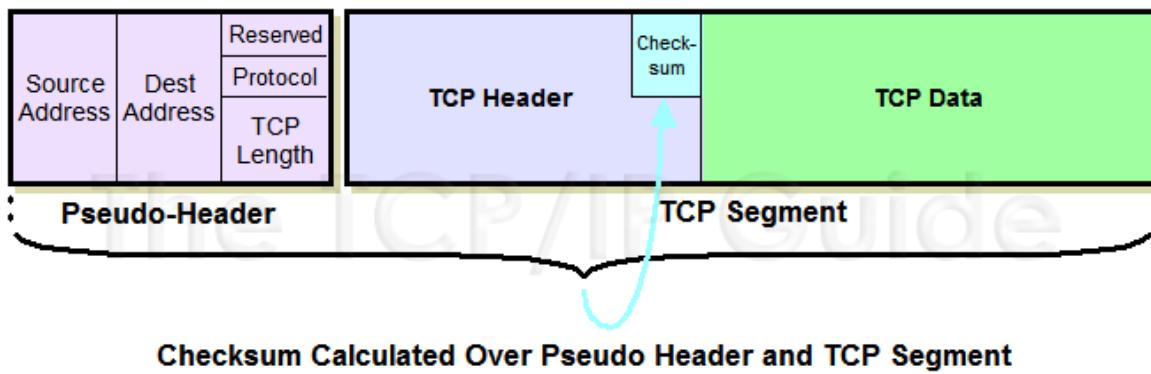
Tiếp tục dùng Hercules gửi thử 1 tin Connect



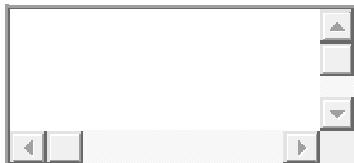
Virtual Terminal

```
IOT47.com - ENC28J60 tutorial !!!
Dang khai tao ENC28J60 ...
Your MAC addr = 00:00:20:07:19:98
Da luu ip 192.168.1.18 = 00:26:5E:3D:74:66
Da nhan 1 goi goi tin ket noi
Da nhan 1 goi goi tin ket noi
Da nhan 1 goi goi tin ket noi
```

OK rồi, giờ phải phản hồi lại, để phản hồi thì phải tính kèm checksum, nên mình sẽ tạo hàm tính checksum cho giao thức TCP hen



Checksum của TCP gồm toàn bộ gói TCP cộng thêm 8 byte IP, 2byte TCP length và 1 byte TCP Protocol (tương tự gói checksum của UDP)



```

1 uint16_t TCP_checksum(TCP_struct *TCP_Struct_Frame)
2 {
3     uint32_t checksum;
4     uint8_t *ptr;
5     uint16_t length;
6     length = swap16(TCP_Struct_Frame->TotoLength) - 20 + 8; //tinh length bat dau tu checksum
7     ptr = (uint8_t *)&TCP_Struct_Frame->SourceIP;      //dia chi bat dau tinh checksum
8
9     checksum=6 + length - 8;
10    while(length>1) //cong het cac byte16 lai
11    {
12        checksum += (uint16_t) (((uint32_t)*ptr<<8)*(ptr+1));
13        ptr+=2;
14        length-=2;
15    };
16    if(length) checksum+=((uint32_t)*ptr)<<8; //neu con le 1 byte
17    while (checksum>>16) checksum=(uint16_t)checksum+(checksum>>16);

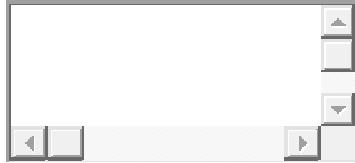
```

```

18 //nghich dao bit
19 checksum=~checksum;
20 //hoan vi byte thap byte cao
21 return swap16(checksum);
22}

```

Quay trở lại với hàm **TCP_read**



```

1 void TCP_read(uint8_t *TCP_Frame,uint16_t len)
2 {
3     uint16_t port;
4     TCP_struct *TCP_Struct_Frame = (TCP_struct *)TCP_Frame;
5     //kiem tra dia chi ip xem co phai no gui cho minh khong
6     if( memcmp(TCP_Struct_Frame->DestIP,ip,4) )return; // dung memcmp de so sanh, neu khac thi thoat
7     if(TCP_Struct_Frame->TCP_Flags == TCP_SYN)
8     {
9         //reply voi SYN|ACK
10        //make reply
11        memcpy(TCP_Struct_Frame->MAC_dich,TCP_Struct_Frame->MAC_nguon,6);
12        memcpy(TCP_Struct_Frame->MAC_nguon,macaddr,6);
13        TCP_Struct_Frame->CheckSum=0;
14        memcpy(TCP_Struct_Frame->DestIP,TCP_Struct_Frame->SourceIP,4); //hoan vi source, dest
15        memcpy(TCP_Struct_Frame->SourceIP,ip,4); //ip cua minh
16        port = TCP_Struct_Frame->Source_Port;
17        TCP_Struct_Frame->Source_Port = TCP_Struct_Frame->Dest_Port;
18        TCP_Struct_Frame->Dest_Port = port;
19        TCP_Struct_Frame->Acknowledgement = swap32(swap32(TCP_Struct_Frame->Sequence_Number) + 1);
20        TCP_Struct_Frame->Sequence_Number = swap32(2071998);
21        TCP_Struct_Frame->TCP_Flags = TCP_SYN | TCP_ACK;
22        TCP_Struct_Frame->TCP_Checksums=0;

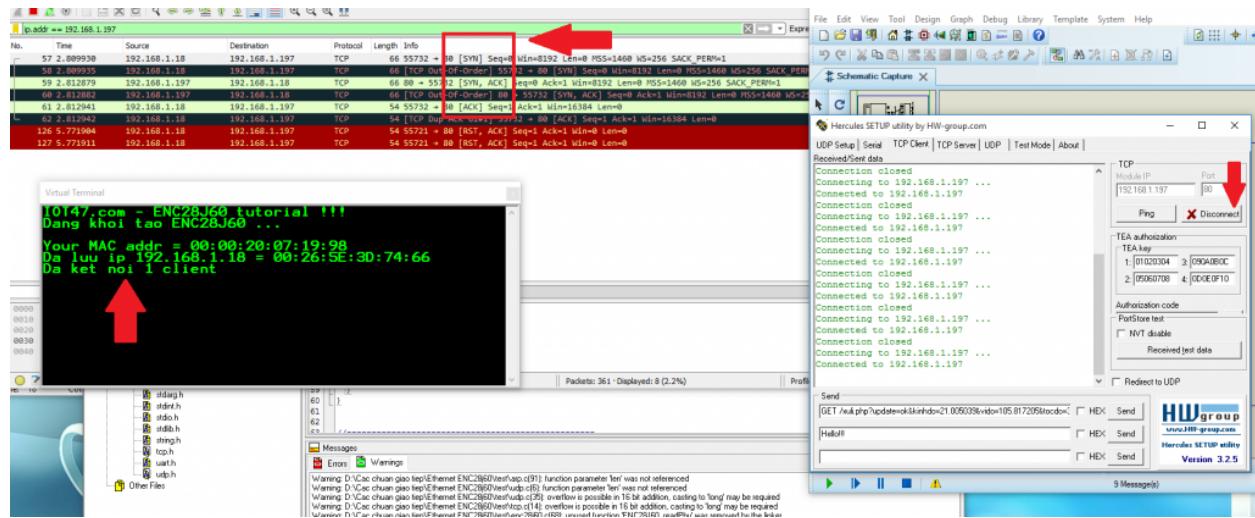
```

```

23     TCP_Struct_Frame->Urgent_Pointer=0;
24     TCP_Struct_Frame->CheckSum = NET_ipchecksum((uint8_t *)&TCP_Struct_Frame->Header_length); //tinh checksum
25     cho goi IO
26
27     TCP_Struct_Frame->TCP_Checksums = TCP_checksum(TCP_Struct_Frame);
28
29     NET_SendFrame((uint8_t *)TCP_Struct_Frame,len); //gui goi tin tcp reply
30
31     if(TCP_Struct_Frame->TCP_Flags == TCP_ACK)
32     {
33         UART.putString("Da ket noi 1 client\r\n");
34     }
35

```

Biên dịch và chạy lại mô phỏng. Bây giờ khi ấn Connect trên Hercules chúng ta đã kết nối thành công tới server 😊



WireShark cũng đã bắt lại toàn bộ quá trình !

Ngắt kết nối giao thức TCP

Hãy thử ấn Disconnect trên Hercules, bạn sẽ thấy ngay 1 gói tin với cờ FIN và ACK (Wikipedia nó không hề lừa mình 😊)

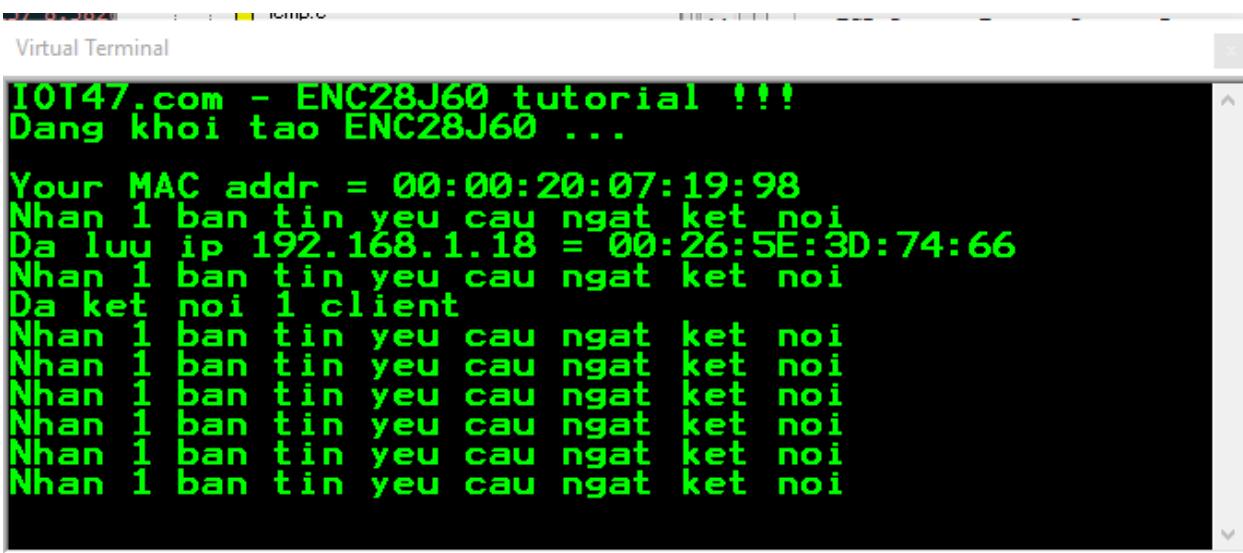
Tiếp tục bắt 2 cờ này và phản hồi lại nào (cứ theo lí thuyết mà phang thôi 😊)

Trong hàm **TCP_read**



```
1 else if(TCP_Struct_Frame->TCP_Flags == (TCP_FIN|TCP_ACK))  
2 {  
3     UART_putString("Nhan 1 ban tin yeu cau ngat ket noi\r\n");  
4  
5 }
```

Vẫn với phương châm test từng tí môt. ! Tiếp tục test tiếp bằng debug



Rồi ! Giờ xóa dòng debug đi và gửi phản hồi lại

Code make header gần như copy y chang, chỉ sửa lại **sequence number** và **acknowledgment number** và tính lại checksum

Mình khai báo thêm `uint32_t dat_ack` ở đầu hàm



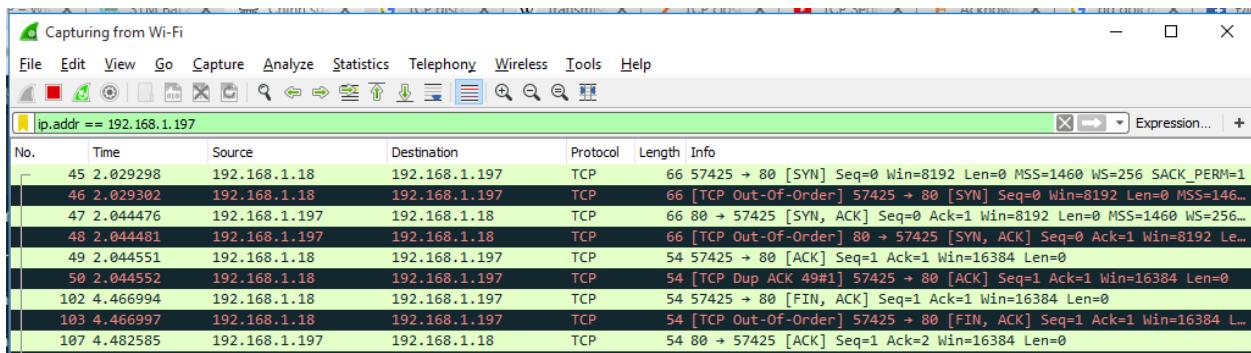
```
1 else if(TCP_Struct_Frame->TCP_Flags == (TCP_FIN|TCP_ACK))  
2 {  
3     //reply voi ACK
```

```

4 //make reply
5 memcpy(TCP_Struct_Frame->MAC_dich,TCP_Struct_Frame->MAC_nguon,6);
6 memcpy(TCP_Struct_Frame->MAC_nguon,macaddr,6);
7 TCP_Struct_Frame->CheckSum=0;
8 memcpy(TCP_Struct_Frame->DestIP,TCP_Struct_Frame->SourceIP,4); //hoan vi source, dest
9 memcpy(TCP_Struct_Frame->SourceIP,ip,4);           //ip cua minh
10 port = TCP_Struct_Frame->Source_Port;
11 TCP_Struct_Frame->Source_Port = TCP_Struct_Frame->Dest_Port;
12 TCP_Struct_Frame->Dest_Port = port;
13 dat_ack = TCP_Struct_Frame->Acknowledgement;
14 TCP_Struct_Frame->Acknowledgement = swap32(swap32(TCP_Struct_Frame->Sequence_Number) + 1);
15 TCP_Struct_Frame->Sequence_Number = dat_ack;
16 TCP_Struct_Frame->TCP_Flags = TCP_ACK;
17 TCP_Struct_Frame->TCP_Checksums=0;
18 TCP_Struct_Frame->Urgent_Pointer=0;
19 TCP_Struct_Frame->CheckSum = NET_ipchecksum((uint8_t *)&TCP_Struct_Frame->Header_length); //tinh checksum
cho goi IO
20 TCP_Struct_Frame->TCP_Checksums = TCP_checksum(TCP_Struct_Frame);
21
22 NET_SendFrame((uint8_t *)TCP_Struct_Frame,len); //gui goi tin tcp reply
23 }

```

Giờ ngắt kết nối thử nhé



WireShark đã bắt được gói tin phản hồi ngắt kết nối của ENC28J60 (ACK)

Hiện tại mới chỉ Client ngắt kết nối thôi, server tiếp tục gửi cờ FIN|ACK để ngắt kết nối pha thứ 2



```

1 TCP_Struct_Frame->TCP_Flags = TCP_FIN|TCP_ACK;
2 TCP_Struct_Frame->TCP_Checksums=0;
3 TCP_Struct_Frame->TCP_Checksums = TCP_checksum(TCP_Struct_Frame);
4
5 NET_SendFrame((uint8_t *)TCP_Struct_Frame,len); //gui goi tin tcp reply

```

Capturing from Wi-Fi

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

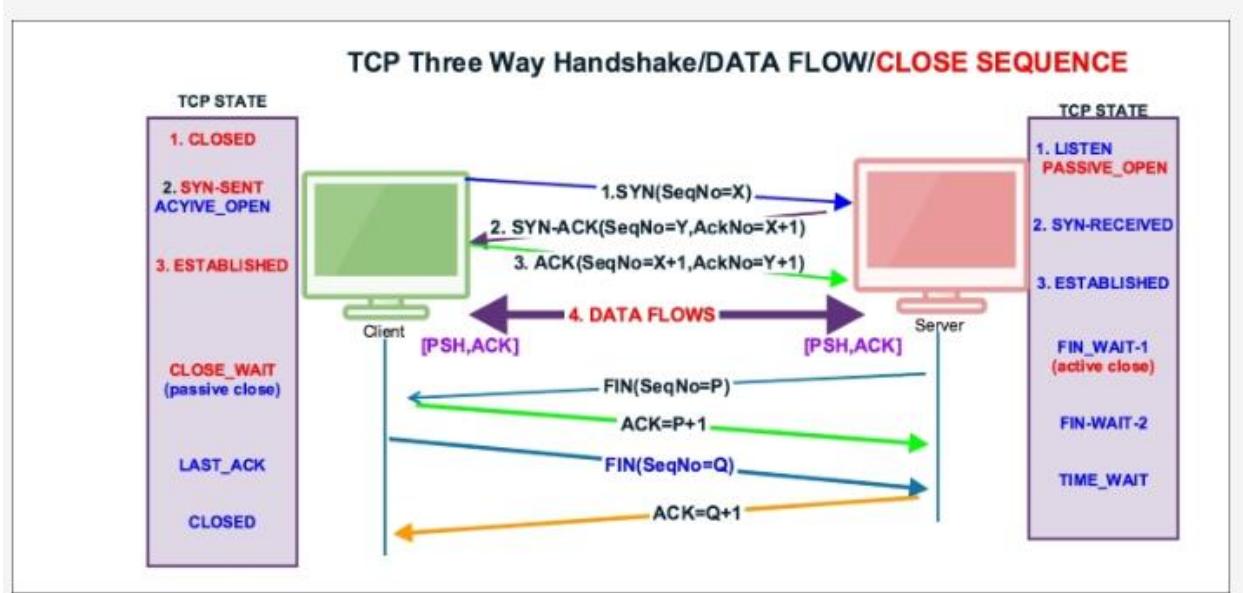
Filter: p.addr == 192.168.1.197

Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
45	2.029298	192.168.1.18	192.168.1.197	TCP	66	57425 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
46	2.029302	192.168.1.18	192.168.1.197	TCP	66	[TCP Out-Of-Order] 57425 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=146...
47	2.044476	192.168.1.197	192.168.1.18	TCP	66	80 → 57425 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256...
48	2.044481	192.168.1.197	192.168.1.18	TCP	66	[TCP Out-Of-Order] 80 → 57425 [SYN, ACK] Seq=0 Ack=1 Win=8192 Le...
49	2.044551	192.168.1.18	192.168.1.197	TCP	54	57425 → 80 [ACK] Seq=1 Ack=1 Win=16384 Len=0
50	2.044552	192.168.1.18	192.168.1.197	TCP	54	[TCP Dup ACK 49#1] 57425 → 80 [ACK] Seq=1 Ack=1 Win=16384 Len=0
102	4.466994	192.168.1.18	192.168.1.197	TCP	54	57425 → 80 [FIN, ACK] Seq=1 Ack=1 Win=16384 Len=0
103	4.466997	192.168.1.18	192.168.1.197	TCP	54	[TCP Out-Of-Order] 57425 → 80 [FIN, ACK] Seq=1 Ack=1 Win=16384 L...
107	4.482585	192.168.1.197	192.168.1.18	TCP	54	80 → 57425 [ACK] Seq=1 Ack=2 Win=16384 Len=0
108	4.482589	192.168.1.197	192.168.1.18	TCP	54	[TCP Dup ACK 107#1] 80 → 57425 [ACK] Seq=1 Ack=2 Win=16384 Len=0
109	4.483442	192.168.1.197	192.168.1.18	TCP	54	80 → 57425 [FIN, ACK] Seq=1 Ack=2 Win=16384 Len=0
110	4.483444	192.168.1.197	192.168.1.18	TCP	54	[TCP Out-Of-Order] 80 → 57425 [FIN, ACK] Seq=1 Ack=2 Win=16384 L...
111	4.483463	192.168.1.18	192.168.1.197	TCP	54	57425 → 80 [ACK] Seq=2 Ack=2 Win=16384 Len=0

Server phản hồi FIN|ACK và client trả lời lại ACK, hoàn tất đóng kết nối

Tóm tắt quy trình

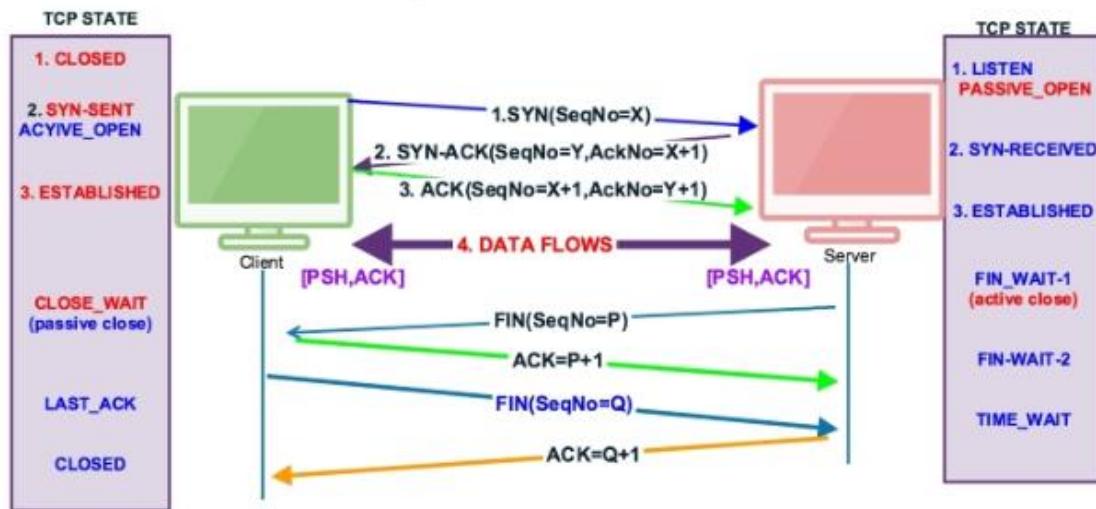


Các bạn có thể tải file cho bài này [tại đây](#)

[ENC28J60] Bài 13: Giao thức TCP – truyền nhận dữ liệu

10 Tháng Tư, 2020 Đào Nguyễn Ethernet, IoT tutorial 2

TCP Three Way Handshake/DATA FLOW/CLOSE SEQUENCE



Trong **bài trước**, chúng ta đã làm quen với việc đóng mở kết nối của giao thức TCP, tiếp tục bài này mình sẽ hướng dẫn các bạn truyền nhận dữ liệu trong giao thức TCP

Nhận dữ liệu từ client gửi lên

Khi client gửi dữ liệu lên cho server, nó sẽ set cờ PSH|ACK, chúng ta sẽ kiểm tra cờ này trong hàm **TCP_read** nhé



```
1 else if(TCP_Struct_Frame->TCP_Flags == (TCP_PSH|TCP_ACK))  
2 {  
3  
4 }  
\
```

Tính độ dài của dữ liệu gửi đến

Căn cứ vào trường **IP->Totolength** để tính độ dài, **IP->Totolength** là độ dài của gói IP + TCP. Do đó mình sẽ trừ đi 20 (độ dài mặc định của gói IP) và căn cứ vào trường **TCP->dataOffset** để tính độ dài phần header của gói TCP

Trường **TCP->dataOffset** có độ dài 4 bit và là 4bit cao nên mình $>>4$. Mỗi giá trị trong **dataOffset** tương ứng 4byte nên ta nhân 4 lên. ($TCP->dataOffset >> 4$) * 4 = $TCP->dataOffset >> 2$

Tóm lại: Độ dài của trường TCP data = Totolength – 20 – (dataOffset >> 2)

Sau khi tính được độ dài thì mình sẽ in từng byte data ra màn hình debug bằng lệnh **UART_putChar**



1 //tinh do dai cua goi data va in ra man hinh

```
2     dataLength = swap16(TCP_Struct_Frame->TotalLength) - 20 - (TCP_Struct_Frame->data_offset >> 2); // (>> 4)*4 = >> 2
3     sprintf(debug_string, "%u:", dataLength);
4     UART.putString(debug_string);
5     for(i=0;i<dataLength;i++)
6         UART.putChar(TCP_Struct_Frame->data[i]);
```

Phản hồi lại ACK cho client

Khi nhận được gói tin, chúng ta bắt buộc phải phản hồi lại cho client bằng cờ ACK, mình sẽ tận dụng luồng gói dữ liệu client vừa gửi đến và cắt bỏ đi trường TCP->Data và remake lại 1 vài thông tin, cụ thể như sau:

- **Totolength** phải giảm đi 1 lượng bằng dữ liệu của TCP vì mình cắt bỏ đi mà
- **Acknowledgement** sẽ không phải là + thêm 1 nữa mà phải cộng thêm độ dài của trường **TCP->data** Như vậy bên client nhận được gói tin ACK sẽ biết chắc chắn là sever đã nhận được đúng từng này data rồi. Còn nếu sai client tự động gửi lại. Bởi cơ chế phản hồi này nên giao thức TCP rất đáng tin cậy

Toàn bộ code xử lí cờ PSH|ACK, các bạn khai báo thêm các biến cục bộ **i,dataLength** trên đầu hàm nhé !



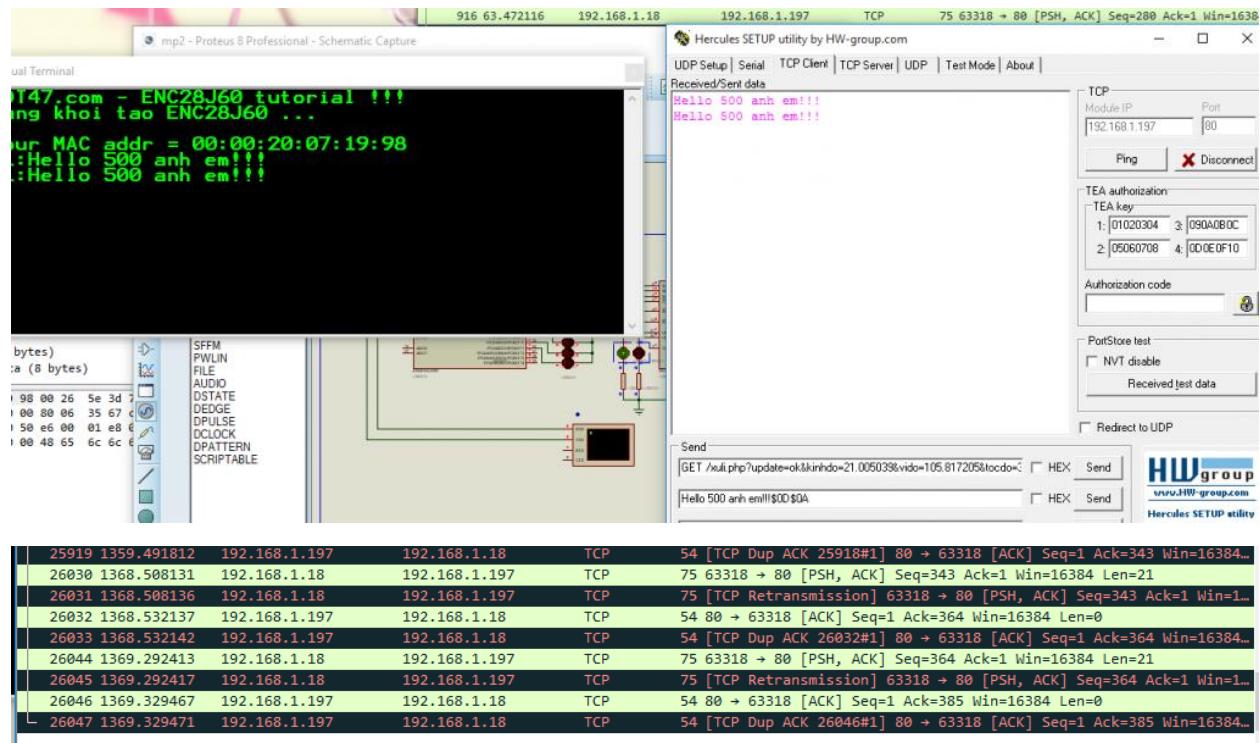
```
1 else if(TCP_Struct_Frame->TCP_Flags == (TCP_PSH|TCP_ACK))
2 {
```

```

3 //tinh do dai cua goi data va in ra man hinh
4 datalength= swap16(TCP_Struct_Frame->TotoLength) -20 - (TCP_Struct_Frame->data_offset >> 2); // ( >> 4)*4 = >> 2
5 sprintf(debug_string,"%u:",datalength);
6 UART.putString(debug_string);
7 for(i=0;i<datalength;i++)
8   UART_putChar(TCP_Struct_Frame->data[i + ((TCP_Struct_Frame->data_offset >> 2) - 20 )]);
9 UART.putString("\r\n");
10
11 //make reply
12 len=datalength; //resize len
13
14 memcpy(TCP_Struct_Frame->MAC_dich,TCP_Struct_Frame->MAC_nguon,6);
15 memcpy(TCP_Struct_Frame->MAC_nguon,macaddr,6);
16 TCP_Struct_Frame->CheckSum=0;
17 TCP_Struct_Frame->TotoLength = swap16(40); //defual totolength
18 TCP_Struct_Frame->data_offset = 0x50; //defual data_offset
19 memcpy(TCP_Struct_Frame->DestIP,TCP_Struct_Frame->SourceIP,4); //hoan vi source, dest
20 memcpy(TCP_Struct_Frame->SourceIP,ip,4); //ip cua minh
21 port = TCP_Struct_Frame->Source_Port;
22 TCP_Struct_Frame->Source_Port = TCP_Struct_Frame->Dest_Port;
23 TCP_Struct_Frame->Dest_Port = port;
24 dat_ack = TCP_Struct_Frame->Acknowledgement;
25 TCP_Struct_Frame->Acknowledgement = swap32(swap32(TCP_Struct_Frame->Sequence_Number) + datalength);
26 TCP_Struct_Frame->Sequence_Number = dat_ack;
27 TCP_Struct_Frame->TCP_Flags = TCP_ACK;
28 TCP_Struct_Frame->TCP_Checksums=0;
29 TCP_Struct_Frame->Urgent_Pointer=0;
30 TCP_Struct_Frame->CheckSum = NET_ipchecksum((uint8_t *)&TCP_Struct_Frame->Header_length); //tinh checksum
31 cho goi IO
32 TCP_Struct_Frame->TCP_Checksums = TCP_checksum(TCP_Struct_Frame);
33
34 NET_SendFrame((uint8_t *)TCP_Struct_Frame,len); //gui goi tin tcp reply
}

```

Tiếp tục test gửi thử bằng chức năng TCP Client của phần mềm Hercules



WireShark cũng đã bắt được đầy đủ các gói tin

Viết bản tin phản hồi



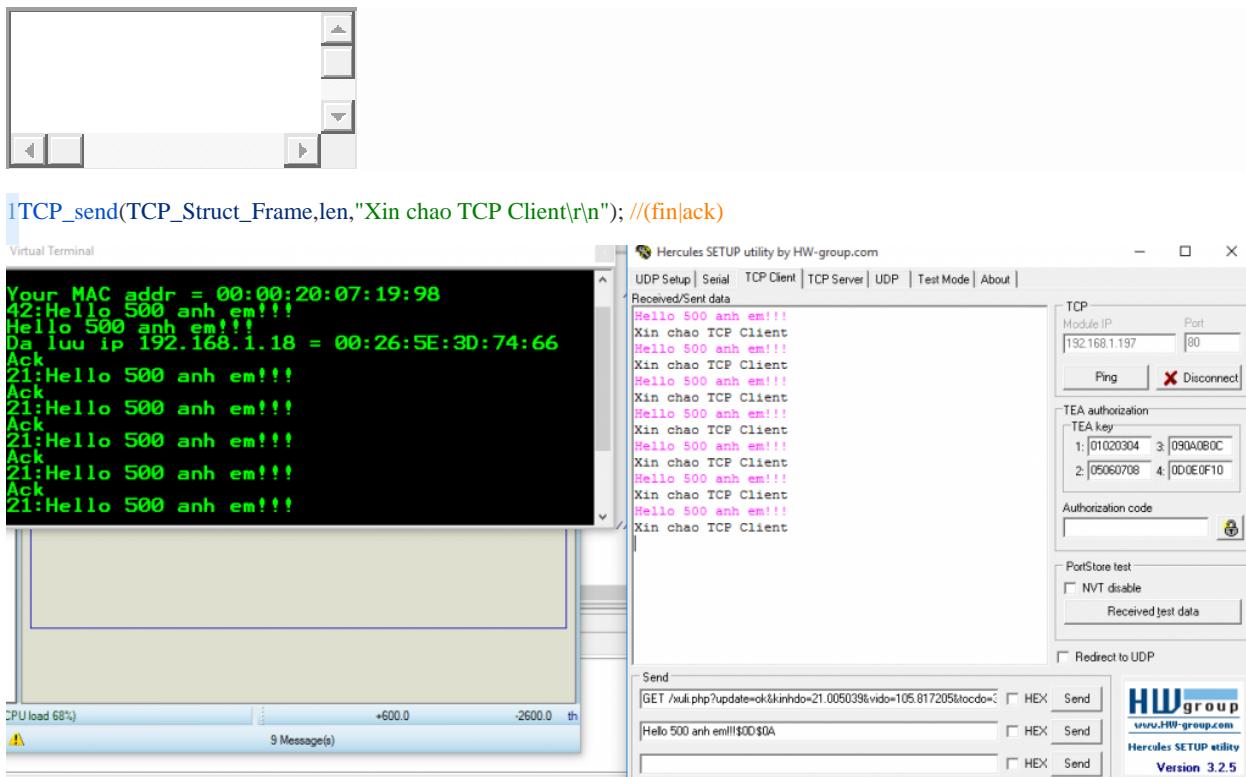
```
1 void TCP_send(TCP_struct *TCP_Struct_Frame,uint16_t len,uint8_t *data)
2 {
3     uint16_t data_length;
4     strcpy(TCP_Struct_Frame->data,data);
5     data_length=strlen(data);
6     len+=data_length;
7     TCP_Struct_Frame->TotoLength = swap16(swap16(TCP_Struct_Frame->TotoLength) + data_length); //make totolength
8     TCP_Struct_Frame->CheckSum=0;
9     TCP_Struct_Frame->TCP_Checksums=0;
10    TCP_Struct_Frame->TCP_Flags = TCP_PSH|TCP_ACK;
11 }
```

```

12 TCP_Struct_Frame->CheckSum = NET_ipchecksum((uint8_t *)&TCP_Struct_Frame->Header_length); //tinh checksum
cho goi IO
13
14 TCP_Struct_Frame->TCP_Checksums = TCP_checksum(TCP_Struct_Frame);
15
16 NET_SendFrame((uint8_t *)TCP_Struct_Frame,len);
17 }

```

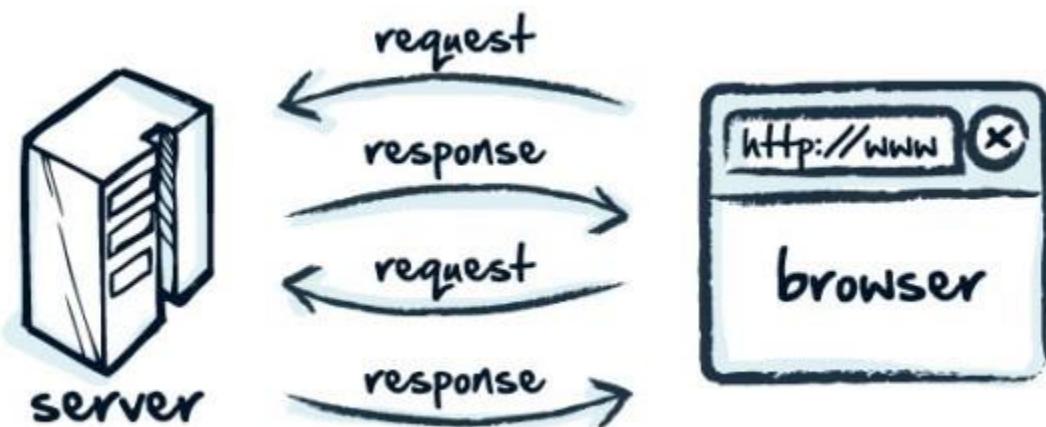
Ngay khi nhận được tin nhắn và gửi trả ACK, mình sẽ gọi hàm reply lại



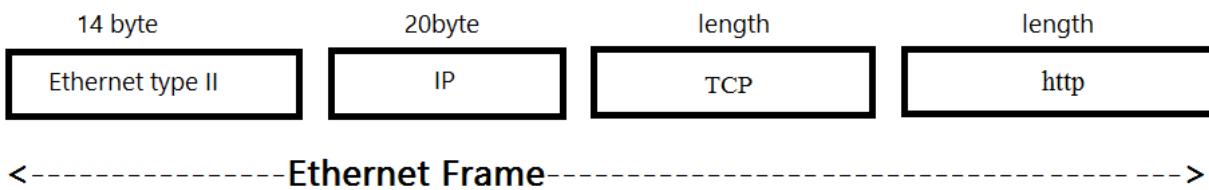
Phần mềm Hercules đã nhận được gói tin phản hồi, tuy nhiên, ở hàm trên, mình mới phản hồi như kiểu UDP chứ chưa kiểm tra lại xem đã gửi thành công chưa, vì như đã nói lúc nãy, TCP có cơ chế gửi và kiểm tra rất chặt chẽ, gửi 1 lúc chưa thấy cờ ACK phản hồi sẽ phải gửi lại. Tuy nhiên vấn đề này không dễ, chúng ta sẽ hoàn thiện nó trong các bài tiếp.

[ENC28J60] Bài 14: Giao thức HTTP – Tạo Webserver với ENC28J60

13 Tháng Tư, 2020 Đào Nguyễn Ethernet, IoT tutorial 0



Giao thức **HTTP** (Tiếng Anh: **HyperText Transfer Protocol** – *Giao thức truyền tải siêu văn bản*) là một giao thức thuộc tầng ứng dụng (lớp trên của giao thức TCP) nổi đơn giản là nó hoạt động trên nền **giao thức TCP** – nó chính là trường data của giao thức TCP



Mình đã có viết một bài về giao thức http, các bạn tham khảo tại đây: <http://iot47.com/iot-bai-4-gioi-thieu-ngoan-nhu-html-va-mo-hinh-http-request-reponse/>

HTTP Request

Client gửi 1 bản tin tới Server, Server tiếp nhận, phân tích và trả về nội dung mà client muốn. 1 HTTP Request cơ bản gồm các request line và data. Request line có cấu trúc như sau:

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Các request line sẽ được kết thúc bằng 2 lần xuống dòng (\r\n\r\n)

Trăm nghe không bằng 1 thấy, tại sao không test thử luôn nha 😊

Sử dụng project đã làm ở [bài 13](#), chạy mô phỏng, mở trình duyệt, gõ địa chỉ IP của ENC28j60 (như của mình là 192.168.1.197) các bạn sẽ thấy máy tính gửi lên 1 cái truy vấn tương tự như trên

```
Virtual Terminal
IOT47.com - ENC28J60 tutorial !!!
Dang khai tao ENC28J60 ...
Your MAC addr = 00:00:20:07:19:98
Da luu ip 192.168.1.18 = 00:26:5E:30:74:66
Ack
435:GET / HTTP/1.1
Host: 192.168.1.197
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/8
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,appli
Accept-Encoding: gzip, deflate
Accept-Language: vi
```

Chúng ta sẽ nghiên cứu kỹ hơn về http request ở bài TCP Client, còn bài này đang làm TCP Server nên tập trung vào http reponse nhé

HTTP Reponse

Khi nhận 1 request và phân tích request, server đã biết client muốn gì, nó sẽ trả lời lại nội dung tương ứng cho server. 1 HTTP Reponse có cấu trúc cơ bản như sau:

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

Status

mã
HTML

Về cơ bản, chúng ta có thể rút gọn các Status Line xuống còn 2 dòng như sau:

HTTP/1.1 200 OK

Content-Type: text/html

HTTP/1.1 ám chỉ phiên bản HTTP, nó đã được phát triển lên 2.0. Nhưng chưa được đưa vào sử dụng
200 là Status code, ám chỉ server đã tiếp nhận và xử lý request của client thành công

Content-Type: text/html ám chỉ phần data mà server sẽ gửi thuộc loại mã html

Các status line sẽ được kết thúc bằng 2 lần xuống dòng (`\r\n\r\n`)

Và sau khi gửi xong các statusline thì chính là data của server gửi xuống dưới dạng các mã HTML

Ngôn ngữ HTML

Sau khi nhận được dữ liệu Server trả về (tức các đoạn code html), trình duyệt web sẽ phân tích các mã html đó và tạo ra giao diện hiển thị cho người dùng.

Ngôn ngữ HTML được tạo bởi các cặp thẻ, nó tạo ra khung sườn cho trang Web, kết hợp với CSS và JavaScript tạo nên bộ 3 tam giác hợp bích.

Cấu trúc cơ bản của 1 trang HTML



```
1<!DOCTYPE html>
2 <html lang="en">
3 <head></head>
4 <body>
5   Đây là 1 trang web
6 </body>
7 </html>
```

1 số thẻ thường dùng

Thẻ <h1></h1> tạo ra phần chữ tiêu đề (chữ khá to và đậm)
Ví dụ



```
1<h1>Đây là thẻ H1</h1>
```

Đây là thẻ H1

Các bạn tự tìm hiểu thêm về ngôn ngữ html trên mạng nhé, chúng ta sẽ quay lại chủ đề chính ngày hôm nay

Tạo webserver với ECN28J60

Tiếp tục với code đã viết ở **bài 13**, tại chỗ



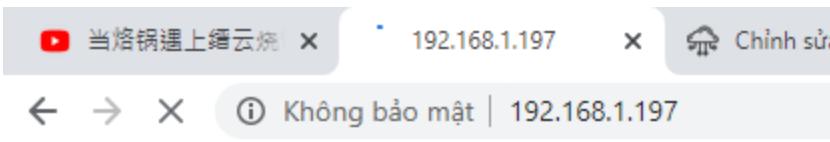
```
1TCP_send(TCP_Struct_Frame,len,"Xin chao TCP Client\r\n");
```

Các bạn xóa dòng phản hồi này đi vì nó không cần thiết nữa, thay vào đó, mình sẽ thêm 1 lệnh kiểm tra data gửi đến xem lệnh gửi đến có phải là truy vấn GET không, nếu đúng thì trả về http response cùng với 1 mã html đơn giản



```
1 if (strncmp((char*)TCP_Struct_Frame->data,"GET /", 5) == 0)
2 {
3     TCP_send(TCP_Struct_Frame,len,"HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<h1>IoT47.com xin chao cac
4     ban</h1>");
5 }
```

Các bạn chạy mô phỏng và quay lại trình duyệt web để truy cập vào IP của enc28j60



IoT47.com xin chao cac ban

Tuyệt ! 1 trang web siêu đơn giản đã được tạo ra !

Tạo thư viện webserver

Đó chỉ là 1 demo nhỏ thôi, để tạo 1 webserver hoàn chỉnh sẽ phức tạp hơn, vì 1 trang web lớn thì không thể gửi 1 phát xong ngay được mà phải chia nhỏ thành các gói tin để gửi lần lượt. Chúng ta sẽ làm điều đó trong 1 thư viện riêng nhé

Các bạn tạo 2 file thư viện `http.c` và `http.h` và add vào project

Thêm mã khởi tạo cho file `http.h`



```
1 #ifndef HTTP_H_
2 #define HTTP_H_
3 //-----
```

```

4 //Include cac thu vien can thiet
5 #include <mega328p.h>
6 #include <delay.h>
7 #include <string.h>
8 #include <stdlib.h>
9 #include <stdint.h>
10#include <stdio.h>
11#include <spi.h>
12#include "uart.h"
13#include "tcp.h"
14
15//-----
16
17//-----
18#endif /* HTTP_H_ */

```

Và mã khởi tạo cho file [http.c](#)



```

1 #include "http.h"
2
3 extern const uint8_t macaddr[6];
4 extern const uint8_t ip[4];
5 extern uint8_t debug_string[60];
6
7
8
9
10//-----

```

Tóm tắt quy trình khi bạn truy cập vào webserver

- Client (tức trình duyệt) gửi bản tin connect (SYN) server trả lời và hoàn tất việc bắt tay
- Client gửi http request tới server (PSH|ACK)

- Server nhận được http request và gửi trả mã html, do chip AVR có giới hạn RAM nên mình đã khởi tạo bộ đệm là 512 byte (trong net.c) , do vậy 1 lần gửi dữ liệu đi chỉ được tối đa 12 byte, lúc này sẽ có 3 trường hợp
 1. Gửi gói 1 lần xong luôn
 2. Gửi gói lần cuối
 3. Gửi gói n lần nữa mới đến lần cuối
- Sau khi gửi xong chúng ta sẽ đóng kết nối với client

Chúng ta sẽ xử lý tất cả các trường hợp nhé, trước tiên mình phải tạo ra 1 giao diện web đã, mình sẽ sử dụng giao diện web đã viết ở tutorial **IOT-ESP8266 bài 5** – nó có dạng hiển thị như này



Còn đây là mã nguồn của nó



```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
5     <title>Điều khiển thiết bị</title>
6     <meta name="viewport" content="width=device-width, initial-scale=1">
7     <style>
8       .b{width: 100px;height: 40px;font-size: 21px;color: #FFF;background-color:#4caf50;border-radius: 10px;}
9
10      .t{width: 100px;height: 40px;font-size: 21px;color: #FFF;background-color:#f44336;border-radius: 10px;}
```

```

11      </style>
12</head>
13<body>
14<div style="width: 330px; height: auto; margin: 0 auto; margin-top: 70px">
15<h1 align="center">Điều khiển thiết bị qua WIFI</h1>
16      <table align="center">
17          <tr>
18              <td><a href='/bat1'><button class='b'>Bật 1</button></a></td>
19              <td><a href='/tat1'><button class='t'>Tắt 1</button></a></td>
20          <tr>
21          <tr>
22              <td><a href='/bat2'><button class='b'>Bật 2</button></a></td>
23              <td><a href='/tat2'><button class='t'>Tắt 2</button></a></td>
24          <tr>
25          <tr>
26              <td><a href='/bat3'><button class='b'>Bật 3</button></a></td>
27              <td><a href='/tat3'><button class='t'>Tắt 3</button></a></td>
28          <tr>
29          <tr>
30              <td><a href='/bat4'><button class='b'>Bật 4</button></a></td>
31              <td><a href='/tat4'><button class='t'>Tắt 4</button></a></td>
32          <tr>
33      </table>
34</div>
35</body>
</html>

```

Trong mã trên mình có xài kí tự tiếng việt UTF-8, mà phần mềm AVRCodevision không hỗ trợ editer utf-8 nên mình sẽ chuyển sang dạng HEX để code webserver của chúng ta hiện được tiếng Việt nhé !
 Minh xài tool online này để chuyển : <https://onlineutf8tools.com/convert-utf8-to-hexadecimal>

Trước khi chuyển đổi thì mình cũng nhét thêm mấy dòng Status Line ở đầu luôn, như vậy toàn bộ code web sẽ như sau:



```
1 HTTP/1.1 200 OK
2 Content-Type: text/html
3
4 <!DOCTYPE html>
5 <html>
6 <head>
7     <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
8     <title>Điều khiển thiết bị</title>
9     <meta name="viewport" content="width=device-width, initial-scale=1">
10    <style>
11        .b{width: 100px;height: 40px;font-size: 21px;color: #FFF;background-color:#4caf50;border-radius:
12            10px;}
13        .t{width: 100px;height: 40px;font-size: 21px;color: #FFF;background-color:#f44336;border-radius: 10px;}
14    </style>
15 </head>
16 <body>
17 <div style="width: 330px;height: auto;margin: 0 auto; margin-top: 70px">
18     <h1 align="center">Điều khiển thiết bị qua WIFI</h1>
19     <table align="center">
20         <tr>
21             <td><a href='/bat1'><button class='b'>Bật 1</button></a></td>
22             <td><a href='/tat1'><button class='t'>Tắt 1</button></a></td>
23         <tr>
24             <td><a href='/bat2'><button class='b'>Bật 2</button></a></td>
25             <td><a href='/tat2'><button class='t'>Tắt 2</button></a></td>
26         <tr>
27             <td><a href='/bat3'><button class='b'>Bật 3</button></a></td>
28             <td><a href='/tat3'><button class='t'>Tắt 3</button></a></td>
```

```

31      <tr>
32      <tr>
33          <td><a href='/bat4'><button class='b'>Bật 4</button></a><td>
34          <td><a href='/tat4'><button class='t'>Tắt 4</button></a><td>
35      <tr>
36      </table>
37</div>
38</body>
</html>

```

onlineutf8tools.com/convert-utf8-to-hexadecimal
you will instantly get base 16 numbers on the right. Free, quick, and very powerful. Import UTF8 – get base 16. Created by geeks from team Browserling.

❶ love ascii more than utf8? [~]

We also have [Online ASCII Tools](#) – utilities for working with ASCII character set. Check it out!

utf8	hexadecimal
<pre> <tr> <td><button class='b'>Bật 4</button><td> <td><button class='t'>Tắt 4</button><td> <tr> <td><button class='b'>Bật 3</button><td> <td><button class='t'>Tắt 3</button><td> <tr> <td><button class='b'>Bật 4</button><td> <td><button class='t'>Tắt 4</button><td> </table> </div> </body> </html> </pre>	<pre> 0x3c 0x21 0x44 0x4f 0x43 0x54 0x59 0x50 0x45 0x20 0x68 0x74 0x6d 0x6c 0x3e 0x0a 0x3c 0x68 0x74 0x6d 0x6c 0x3e 0x0a 0x3c 0x68 0x65 0x61 0x64 0x3e 0x0a 0x69 0x20 0x3c 0x6d 0x65 0x74 0x61 0x20 0x68 0x74 0x74 0x70 0x2d 0x65 0x71 0x75 0x69 0x76 0x3d 0x22 0x43 0x6f 0x6e 0x74 0x65 0x6e 0x74 0x2d 0x54 0x79 0x70 0x65 0x22 0x20 0x63 0x6f 0x6e 0x74 0x65 0x6e 0x74 0x3d 0x22 0x74 0x65 0x78 0x74 0x2f 0x68 0x74 0x6d 0x6c 0x3b 0x20 0x63 0x68 0x61 0x72 0x73 0x65 0x74 0x3d 0x75 0x74 0x66 0x2d 0x38 0x22 0x3e 0x0a 0x09 0x3c 0x74 0x69 0x74 0x6c 0x65 0x3e 0xc4 0x90 0x69 0xe1 0xbb 0x81 0x75 0x20 0x6b 0x68 0x69 0xe1 0xbb 0x83 0x6e 0x20 0x74 0x68 0x69 0xe1 0xba 0xbff 0x74 0x20 0x62 0x6e 0xbb 0x8b 0x3c 0x2f 0x74 0x69 0x74 0x6c 0x65 0x3e 0x0a 0x09 0x3c 0x6d 0x65 0x74 0x61 0x20 0x6e 0x61 0x6d 0x65 0x3d 0x22 0x76 0x69 0x65 0x77 0x70 0x6f 0x72 0x74 0x22 0x20 0x63 0x6f 0x6e 0x74 0x65 0x6e 0x74 0x3d 0x22 0x77 0x69 0x64 0x74 0x68 0x3d 0x64 0x65 0x76 0x69 0x63 0x65 0x2d 0x77 0x69 0x64 0x74 0x68 0x2c 0x20 0x69 0x6e 0x69 0x74 0x69 0x61 0x6c 0x2d 0x73 0x63 0x61 0x6c 0x65 0x3d 0x31 0x22 0x3e 0x0a 0x09 0x3c 0x73 0x74 0x79 </pre>

Như vậy là có cục data webserver,các bạn bỏ vào file [http.c](#) và đừng quên đưa nó vào bộ nhớ flash

```

1 flash uint8_t web_page[]=
2 {
3     0x48 ,0x54 ,0x54 ,0x50 ,0x2f ,0x31 ,0x2e ,0x31 ,0x20 ,0x32 ,0x30 ,0x30 ,0x20 ,0x4f ,0x4b ,0x0a
4     ,0x43 ,0x6f ,0x6e ,0x74 ,0x65 ,0x6e ,0x74 ,0x2d ,0x54 ,0x79 ,0x70 ,0x65 ,0x3a ,0x20 ,0x74 ,0x65
5     ,0x78 ,0x74 ,0x2f ,0x68 ,0x74 ,0x6d ,0x6c ,0x0a ,0x0a ,0x3c ,0x21 ,0x44 ,0x4f ,0x43 ,0x54 ,0x59

```

6,0x50 ,0x45 ,0x20 ,0x68 ,0x74 ,0x6d ,0x6c ,0x3e ,0x0a ,0x3c ,0x68 ,0x74 ,0x6d ,0x6c ,0x3e ,0x0a
7,0x3c ,0x68 ,0x65 ,0x61 ,0x64 ,0x3e ,0x0a ,0x09 ,0x20 ,0x3c ,0x6d ,0x65 ,0x74 ,0x61 ,0x20 ,0x68
8,0x74 ,0x74 ,0x70 ,0x2d ,0x65 ,0x71 ,0x75 ,0x69 ,0x76 ,0x3d ,0x22 ,0x43 ,0x6f ,0x6e ,0x74 ,0x65
9,0x6e ,0x74 ,0x2d ,0x54 ,0x79 ,0x70 ,0x65 ,0x22 ,0x20 ,0x63 ,0x6f ,0x6e ,0x74 ,0x65 ,0x6e ,0x74
10,0x3d ,0x22 ,0x74 ,0x65 ,0x78 ,0x74 ,0x2f ,0x68 ,0x74 ,0x6d ,0x6c ,0x3b ,0x20 ,0x63 ,0x68 ,0x61
11,0x72 ,0x73 ,0x65 ,0x74 ,0x3d ,0x75 ,0x74 ,0x66 ,0x2d ,0x38 ,0x22 ,0x3e ,0x0a ,0x09 ,0x3c ,0x74
12,0x69 ,0x74 ,0x6c ,0x65 ,0x3e ,0xc4 ,0x90 ,0x69 ,0xe1 ,0xbb ,0x81 ,0x75 ,0x20 ,0x6b ,0x68 ,0x69
13,0xe1 ,0xbb ,0x83 ,0x6e ,0x20 ,0x74 ,0x68 ,0x69 ,0xe1 ,0xba ,0xbf ,0x74 ,0x20 ,0x62 ,0xe1 ,0xbb
14,0x8b ,0x3c ,0x2f ,0x74 ,0x69 ,0x74 ,0x6c ,0x65 ,0x3e ,0x0a ,0x09 ,0x3c ,0x6d ,0x65 ,0x74 ,0x61
15,0x20 ,0x6e ,0x61 ,0x6d ,0x65 ,0x3d ,0x22 ,0x76 ,0x69 ,0x65 ,0x77 ,0x70 ,0x6f ,0x72 ,0x74 ,0x22
16,0x20 ,0x63 ,0x6f ,0x6e ,0x74 ,0x65 ,0x6e ,0x74 ,0x3d ,0x22 ,0x77 ,0x69 ,0x64 ,0x74 ,0x68 ,0x3d
17,0x64 ,0x65 ,0x76 ,0x69 ,0x63 ,0x65 ,0x2d ,0x77 ,0x69 ,0x64 ,0x74 ,0x68 ,0x2c ,0x20 ,0x69 ,0x6e
18,0x69 ,0x74 ,0x69 ,0x61 ,0x6c ,0x2d ,0x73 ,0x63 ,0x61 ,0x6c ,0x65 ,0x3d ,0x31 ,0x22 ,0x3e ,0x0a
19,0x09 ,0x3c ,0x73 ,0x74 ,0x79 ,0x6c ,0x65 ,0x3e ,0x0a ,0x09 ,0x09 ,0x2e ,0x62 ,0x7b ,0x77 ,0x69
20,0x64 ,0x74 ,0x68 ,0x3a ,0x20 ,0x31 ,0x30 ,0x30 ,0x70 ,0x78 ,0x3b ,0x68 ,0x65 ,0x69 ,0x67 ,0x68
21,0x74 ,0x3a ,0x20 ,0x34 ,0x30 ,0x70 ,0x78 ,0x3b ,0x66 ,0x6f ,0x6e ,0x74 ,0x2d ,0x73 ,0x69 ,0x7a
22,0x65 ,0x3a ,0x20 ,0x32 ,0x31 ,0x70 ,0x78 ,0x3b ,0x63 ,0x6f ,0x6c ,0x6f ,0x72 ,0x3a ,0x20 ,0x23
23,0x46 ,0x46 ,0x46 ,0x3b ,0x62 ,0x61 ,0x63 ,0x6b ,0x67 ,0x72 ,0x6f ,0x75 ,0x6e ,0x64 ,0x2d ,0x63
24,0x6f ,0x6c ,0x6f ,0x72 ,0x3a ,0x23 ,0x34 ,0x63 ,0x61 ,0x66 ,0x35 ,0x30 ,0x3b ,0x62 ,0x6f ,0x72
25,0x64 ,0x65 ,0x72 ,0x2d ,0x72 ,0x61 ,0x64 ,0x69 ,0x75 ,0x73 ,0x3a ,0x20 ,0x31 ,0x30 ,0x70 ,0x78
26,0x3b ,0x7d ,0x0a ,0x09 ,0x09 ,0x2e ,0x74 ,0x7b ,0x77 ,0x69 ,0x64 ,0x74 ,0x68 ,0x3a ,0x20 ,0x31
27,0x30 ,0x30 ,0x70 ,0x78 ,0x3b ,0x68 ,0x65 ,0x69 ,0x67 ,0x68 ,0x74 ,0x3a ,0x20 ,0x34 ,0x30 ,0x70
28,0x78 ,0x3b ,0x66 ,0x6f ,0x6e ,0x74 ,0x2d ,0x73 ,0x69 ,0x7a ,0x65 ,0x3a ,0x20 ,0x32 ,0x31 ,0x70
29,0x78 ,0x3b ,0x63 ,0x6f ,0x6c ,0x6f ,0x72 ,0x3a ,0x20 ,0x23 ,0x46 ,0x46 ,0x3b ,0x62 ,0x61
30,0x63 ,0x6b ,0x67 ,0x72 ,0x6f ,0x75 ,0x6e ,0x64 ,0x2d ,0x63 ,0x6f ,0x6c ,0x6f ,0x72 ,0x3a ,0x23
31,0x66 ,0x34 ,0x34 ,0x33 ,0x33 ,0x36 ,0x3b ,0x62 ,0x6f ,0x72 ,0x64 ,0x65 ,0x72 ,0x2d ,0x72 ,0x61
32,0x64 ,0x69 ,0x75 ,0x73 ,0x3a ,0x20 ,0x31 ,0x30 ,0x70 ,0x78 ,0x3b ,0x7d ,0x0a ,0x09 ,0x3c ,0x2f
33,0x73 ,0x74 ,0x79 ,0x6c ,0x65 ,0x3e ,0x0a ,0x3c ,0x2f ,0x68 ,0x65 ,0x61 ,0x64 ,0x3e ,0x0a ,0x3c
34,0x62 ,0x6f ,0x64 ,0x79 ,0x3e ,0x0a ,0x3c ,0x64 ,0x69 ,0x76 ,0x20 ,0x73 ,0x74 ,0x79 ,0x6c ,0x65
35,0x3d ,0x22 ,0x77 ,0x69 ,0x64 ,0x74 ,0x68 ,0x3a ,0x20 ,0x33 ,0x33 ,0x30 ,0x70 ,0x78 ,0x3b ,0x68
36,0x65 ,0x69 ,0x67 ,0x68 ,0x74 ,0x3a ,0x20 ,0x61 ,0x75 ,0x74 ,0x6f ,0x3b ,0x6d ,0x61 ,0x72 ,0x67
37,0x69 ,0x6e ,0x3a ,0x20 ,0x30 ,0x20 ,0x61 ,0x75 ,0x74 ,0x6f ,0x3b ,0x6d ,0x61 ,0x72 ,0x67 ,0x69
38,0x6e ,0x2d ,0x74 ,0x6f ,0x70 ,0x3a ,0x20 ,0x37 ,0x30 ,0x70 ,0x78 ,0x22 ,0x3e ,0x0a ,0x3c ,0x68

39,0x31 ,0x20 ,0x61 ,0x6c ,0x69 ,0x67 ,0x6e ,0x3d ,0x22 ,0x63 ,0x65 ,0x6e ,0x74 ,0x65 ,0x72 ,0x22
40,0x3e ,0xc4 ,0x90 ,0x69 ,0xe1 ,0xbb ,0x81 ,0x75 ,0x20 ,0x6b ,0x68 ,0x69 ,0xe1 ,0xbb ,0x83 ,0x6e
41,0x20 ,0x74 ,0x68 ,0x69 ,0xe1 ,0xba ,0xbf ,0x74 ,0x20 ,0x62 ,0xe1 ,0xbb ,0x8b ,0x20 ,0x71 ,0x75
42,0x61 ,0x20 ,0x57 ,0x49 ,0x46 ,0x49 ,0x3c ,0x2f ,0x68 ,0x31 ,0x3e ,0x0a ,0x09 ,0x3c ,0x74 ,0x61
43,0x62 ,0x6c ,0x65 ,0x20 ,0x61 ,0x6c ,0x69 ,0x67 ,0x6e ,0x3d ,0x22 ,0x63 ,0x65 ,0x6e ,0x74 ,0x65
44,0x72 ,0x22 ,0x3e ,0x0a ,0x09 ,0x3c ,0x74 ,0x72 ,0x3e ,0x0a ,0x09 ,0x09 ,0x09 ,0x3c ,0x74
45,0x64 ,0x3e ,0x3c ,0x61 ,0x20 ,0x68 ,0x72 ,0x65 ,0x66 ,0x3d ,0x27 ,0x2f ,0x62 ,0x61 ,0x74 ,0x31
46,0x27 ,0x3e ,0x3c ,0x62 ,0x75 ,0x74 ,0x6f ,0x6e ,0x20 ,0x63 ,0x6c ,0x61 ,0x73 ,0x73 ,0x3d
47,0x27 ,0x62 ,0x27 ,0x3e ,0x42 ,0xe1 ,0xba ,0xad ,0x74 ,0x20 ,0x31 ,0x3c ,0x2f ,0x62 ,0x75 ,0x74
48,0x74 ,0x6f ,0x6e ,0x3e ,0x3c ,0x2f ,0x61 ,0x3e ,0x3c ,0x74 ,0x64 ,0x3e ,0x0a ,0x09 ,0x09 ,0x09
49,0x3c ,0x74 ,0x64 ,0x3e ,0x3c ,0x61 ,0x20 ,0x68 ,0x72 ,0x65 ,0x66 ,0x3d ,0x27 ,0x2f ,0x74 ,0x61
50,0x74 ,0x31 ,0x27 ,0x3e ,0x3c ,0x62 ,0x75 ,0x74 ,0x6f ,0x6e ,0x20 ,0x63 ,0x6c ,0x61 ,0x73
51,0x73 ,0x3d ,0x27 ,0x74 ,0x27 ,0x3e ,0x54 ,0xe1 ,0xba ,0xaf ,0x74 ,0x20 ,0x31 ,0x3c ,0x2f ,0x62
52,0x75 ,0x74 ,0x74 ,0x6f ,0x6e ,0x3e ,0x3c ,0x2f ,0x61 ,0x3e ,0x3c ,0x74 ,0x64 ,0x3e ,0x0a ,0x09
53,0x20 ,0x20 ,0x20 ,0x20 ,0x3c ,0x74 ,0x72 ,0x3e ,0x0a ,0x09 ,0x20 ,0x20 ,0x20 ,0x20 ,0x3c ,0x74
54,0x72 ,0x3e ,0x0a ,0x09 ,0x09 ,0x3c ,0x74 ,0x64 ,0x3e ,0x3c ,0x61 ,0x20 ,0x68 ,0x72 ,0x65
55,0x66 ,0x3d ,0x27 ,0x2f ,0x62 ,0x61 ,0x74 ,0x32 ,0x27 ,0x3e ,0x3c ,0x62 ,0x75 ,0x74 ,0x6f
56,0x6e ,0x20 ,0x63 ,0x6c ,0x61 ,0x73 ,0x73 ,0x3d ,0x27 ,0x62 ,0x27 ,0x3e ,0x42 ,0xe1 ,0xba ,0xad
57,0x74 ,0x20 ,0x32 ,0x3c ,0x2f ,0x62 ,0x75 ,0x74 ,0x6f ,0x6e ,0x3e ,0x3c ,0x2f ,0x61 ,0x3e
58,0x3c ,0x74 ,0x64 ,0x3e ,0x0a ,0x09 ,0x09 ,0x09 ,0x3c ,0x74 ,0x64 ,0x3e ,0x3c ,0x61 ,0x20 ,0x68
59,0x72 ,0x65 ,0x66 ,0x3d ,0x27 ,0x2f ,0x74 ,0x61 ,0x74 ,0x32 ,0x27 ,0x3e ,0x3c ,0x62 ,0x75 ,0x74
60,0x74 ,0x6f ,0x6e ,0x20 ,0x63 ,0x6c ,0x61 ,0x73 ,0x73 ,0x3d ,0x27 ,0x74 ,0x27 ,0x3e ,0x54 ,0xe1
61,0xba ,0xaf ,0x74 ,0x20 ,0x32 ,0x3c ,0x2f ,0x62 ,0x75 ,0x74 ,0x74 ,0x6f ,0x6e ,0x3e ,0x3c ,0x2f
62,0x61 ,0x3e ,0x3c ,0x74 ,0x64 ,0x3e ,0x0a ,0x09 ,0x20 ,0x20 ,0x20 ,0x20 ,0x3c ,0x74 ,0x72 ,0x3e
63,0x0a ,0x09 ,0x20 ,0x20 ,0x20 ,0x3c ,0x74 ,0x72 ,0x3e ,0x0a ,0x09 ,0x09 ,0x09 ,0x3c ,0x74
64,0x64 ,0x3e ,0x3c ,0x61 ,0x20 ,0x68 ,0x72 ,0x65 ,0x66 ,0x3d ,0x27 ,0x2f ,0x62 ,0x61 ,0x74 ,0x33
65,0x27 ,0x3e ,0x3c ,0x62 ,0x75 ,0x74 ,0x6f ,0x6e ,0x20 ,0x63 ,0x6c ,0x61 ,0x73 ,0x73 ,0x3d
66,0x27 ,0x62 ,0x27 ,0x3e ,0x42 ,0xe1 ,0xba ,0xad ,0x74 ,0x20 ,0x33 ,0x3c ,0x2f ,0x62 ,0x75 ,0x74
67,0x74 ,0x6f ,0x6e ,0x3e ,0x3c ,0x2f ,0x61 ,0x3e ,0x3c ,0x74 ,0x64 ,0x3e ,0x0a ,0x09 ,0x09 ,0x09
68,0x3c ,0x74 ,0x64 ,0x3e ,0x3c ,0x61 ,0x20 ,0x68 ,0x72 ,0x65 ,0x66 ,0x3d ,0x27 ,0x2f ,0x74 ,0x61
69,0x74 ,0x33 ,0x27 ,0x3e ,0x3c ,0x62 ,0x75 ,0x74 ,0x74 ,0x6f ,0x6e ,0x20 ,0x63 ,0x6c ,0x61 ,0x73
70,0x73 ,0x3d ,0x27 ,0x74 ,0x27 ,0x3e ,0x54 ,0xe1 ,0xba ,0xaf ,0x74 ,0x20 ,0x33 ,0x3c ,0x2f ,0x62
71,0x75 ,0x74 ,0x74 ,0x6f ,0x6e ,0x3e ,0x3c ,0x2f ,0x61 ,0x3e ,0x3c ,0x74 ,0x64 ,0x3e ,0x0a ,0x09

```

72,0x20 ,0x20 ,0x20 ,0x20 ,0x3c ,0x74 ,0x72 ,0x3e ,0x0a ,0x09 ,0x20 ,0x20 ,0x20 ,0x3c ,0x74
73,0x72 ,0x3e ,0x0a ,0x09 ,0x09 ,0x3c ,0x74 ,0x64 ,0x3e ,0x3c ,0x61 ,0x20 ,0x68 ,0x72 ,0x65
74,0x66 ,0x3d ,0x27 ,0x2f ,0x62 ,0x61 ,0x74 ,0x34 ,0x27 ,0x3e ,0x3c ,0x62 ,0x75 ,0x74 ,0x74 ,0x6f
75,0x6e ,0x20 ,0x63 ,0x6c ,0x61 ,0x73 ,0x73 ,0x3d ,0x27 ,0x62 ,0x27 ,0x3e ,0x42 ,0xe1 ,0xba ,0xad
76,0x74 ,0x20 ,0x34 ,0x3c ,0x2f ,0x62 ,0x75 ,0x74 ,0x6f ,0x6e ,0x3e ,0x3c ,0x2f ,0x61 ,0x3e
77,0x3c ,0x74 ,0x64 ,0x3e ,0x0a ,0x09 ,0x09 ,0x09 ,0x3c ,0x74 ,0x64 ,0x3e ,0x3c ,0x61 ,0x20 ,0x68
78,0x72 ,0x65 ,0x66 ,0x3d ,0x27 ,0x2f ,0x74 ,0x61 ,0x74 ,0x34 ,0x27 ,0x3e ,0x3c ,0x62 ,0x75 ,0x74
79,0x74 ,0x6f ,0x6e ,0x20 ,0x63 ,0x6c ,0x61 ,0x73 ,0x73 ,0x3d ,0x27 ,0x74 ,0x27 ,0x3e ,0x54 ,0xe1
80,0xba ,0xaf ,0x74 ,0x20 ,0x34 ,0x3c ,0x2f ,0x62 ,0x75 ,0x74 ,0x6f ,0x6e ,0x3e ,0x3c ,0x2f
81,0x61 ,0x3e ,0x3c ,0x74 ,0x64 ,0x3e ,0x0a ,0x09 ,0x20 ,0x20 ,0x20 ,0x20 ,0x3c ,0x74 ,0x72 ,0x3e
82,0x09 ,0x0a ,0x09 ,0x3c ,0x2f ,0x74 ,0x61 ,0x62 ,0x6c ,0x65 ,0x3e ,0x0a ,0x3c ,0x2f ,0x64 ,0x69
83,0x76 ,0x3e ,0x0a ,0x3c ,0x2f ,0x62 ,0x6f ,0x64 ,0x79 ,0x3e ,0x0a ,0x3c ,0x2f ,0x68 ,0x74 ,0x6d
84,0x6c ,0x3e
85};

```

Nếu không có nhu cầu xài tiếng việt thì không cần chuyển sang dạng HEX như này đâu

Giờ sẽ là công việc khó khăn hơn, viết code để đầy cục data này đi khi có lệnh GET từ browser

Do phần header bắt đầu từ 0 đến trường IP->Urgent_Pointer là 54 byte, bộ đệm của chúng ta max là 512 byte nên mình còn dư $512 - 54 = 458$ byte cho dữ liệu http. Nếu dữ liệu http > 458 thì phải cắt nhỏ ra gửi

Mình sẽ tạo 1 số biến toàn cục để lưu trữ số lượng gói tin đã gửi, đang gửi, trạng thái gửi, số hiệu gói tin vừa gửi trong [http.c](#)



```

1uint8_t status=0;
2uint32_t data_num=0;
3uint32_t Sequence_Number=0;

```

Đồng thời viết luôn 1 vài phương thức Set Get để các thư viện khác có thể truy cập vào các biến này



```

1 uint8_t HTTP_get_Status(void)
2 {
3     return status;
4 }
5 void HTTP_set_Status(uint8_t s)
6 {
7     status=s;
8 }
9 uint32_t HTTP_get_Sequence_Number(void)
10{
11    return Sequence_Number;
12}
13void HTTP_set_Sequence_Number(uint32_t SequenceN)
14{
15    Sequence_Number = SequenceN;
16}
17void HTTP_pack_init(void)
18{
19    status=1; //bat dau gui tin http dau tien
20    data_num=0;
21}

```

Đừng quên thêm nguyên mẫu hàm cho nó vào file **html.h** nhé

Chỉnh lại code ở bài 13 một chút cho gọn

Minh sẽ tách 1 số đoạn code thiết lập struct tcp sang 1 hàm riêng cho gọn hàm **TCP_read**. Cụ thể, tạo thêm 1 hàm tên là **TCP_make_header** trong file **tcp.c**



```

1 void TCP_make_header(TCP_struct *TCP_Struct_Frame,uint16_t len,uint8_t type)
2 {
3     uint32_t dat_ack;

```

```

4  uint16_t port,datalength,i;
5  if(type == FOR_SYN)
6  {
7      //reply voi SYN|ACK
8      //make reply
9      memcpy(TCP_Struct_Frame->MAC_dich,TCP_Struct_Frame->MAC_nguon,6);
10     memcpy(TCP_Struct_Frame->MAC_nguon,macaddr,6);
11     TCP_Struct_Frame->CheckSum=0;
12     memcpy(TCP_Struct_Frame->DestIP,TCP_Struct_Frame->SourceIP,4); //hoan vi source, dest
13     memcpy(TCP_Struct_Frame->SourceIP,ip,4);           //ip cua minh
14     port = TCP_Struct_Frame->Source_Port;
15     TCP_Struct_Frame->Source_Port = TCP_Struct_Frame->Dest_Port;
16     TCP_Struct_Frame->Dest_Port = port;
17     TCP_Struct_Frame->Acknowledgement = swap32(swap32(TCP_Struct_Frame->Sequence_Number) + 1);
18     TCP_Struct_Frame->Sequence_Number = swap32(2071998);
19     TCP_Struct_Frame->TCP_Flags = TCP_SYN | TCP_ACK;
20     TCP_Struct_Frame->TCP_Checksums=0;
21     TCP_Struct_Frame->Urgent_Pointer=0;
22     TCP_Struct_Frame->CheckSum = NET_ipchecksum((uint8_t *)&TCP_Struct_Frame->Header_length); //tinh checksum
23     cho goi IO
24     TCP_Struct_Frame->TCP_Checksums = TCP_checksum(TCP_Struct_Frame);
25 }
26 else if(type == FOR_FIN)
27 {
28     //reply voi ACK
29     //make reply
30     memcpy(TCP_Struct_Frame->MAC_dich,TCP_Struct_Frame->MAC_nguon,6);
31     memcpy(TCP_Struct_Frame->MAC_nguon,macaddr,6);
32     TCP_Struct_Frame->CheckSum=0;
33     memcpy(TCP_Struct_Frame->DestIP,TCP_Struct_Frame->SourceIP,4); //hoan vi source, dest
34     memcpy(TCP_Struct_Frame->SourceIP,ip,4);           //ip cua minh
35     port = TCP_Struct_Frame->Source_Port;
36     TCP_Struct_Frame->Source_Port = TCP_Struct_Frame->Dest_Port;
37     TCP_Struct_Frame->Dest_Port = port;

```

```

37 dat_ack = TCP_Struct_Frame->Acknowledgement;
38 TCP_Struct_Frame->Acknowledgement = swap32(swap32(TCP_Struct_Frame->Sequence_Number) + 1);
39 TCP_Struct_Frame->Sequence_Number = dat_ack;
40 TCP_Struct_Frame->TCP_Flags = TCP_ACK;
41 TCP_Struct_Frame->TCP_Checksums=0;
42 TCP_Struct_Frame->Urgent_Pointer=0;
43 TCP_Struct_Frame->CheckSum = NET_ipchecksum((uint8_t *)&TCP_Struct_Frame->Header_length); //tinh checksum
44 //cho goi IO
45 TCP_Struct_Frame->TCP_Checksums = TCP_checksum(TCP_Struct_Frame);
46 }
47 else if(type == FOR_PSH_ACK)
48 {
49 //tinh do dai cua goi data va in ra man hinh
50 dataLength= swap16(TCP_Struct_Frame->TotoLength) -20 - (TCP_Struct_Frame->data_offset >> 2); // ( >> 4)*4 = >> 2
51 for(i=0;i<dataLength;i++)
52     UART_putChar(TCP_Struct_Frame->data[i]);
53
54 //make reply
55 len=dataLength; //resize len
56
57 memcpy(TCP_Struct_Frame->MAC_dich,TCP_Struct_Frame->MAC_nguon,6);
58 memcpy(TCP_Struct_Frame->MAC_nguon,macaddr,6);
59 TCP_Struct_Frame->CheckSum=0;
60 TCP_Struct_Frame->TotoLength = swap16(40); //defaul totolength
61 TCP_Struct_Frame->data_offset = 0x50; //defaul data_offset
62 memcpy(TCP_Struct_Frame->DestIP,TCP_Struct_Frame->SourceIP,4); //hoan vi source, dest
63 memcpy(TCP_Struct_Frame->SourceIP,ip,4); //ip cua minh
64 port = TCP_Struct_Frame->Source_Port;
65 TCP_Struct_Frame->Source_Port = TCP_Struct_Frame->Dest_Port;
66 TCP_Struct_Frame->Dest_Port = port;
67 dat_ack = TCP_Struct_Frame->Acknowledgement;
68 TCP_Struct_Frame->Acknowledgement = swap32(swap32(TCP_Struct_Frame->Sequence_Number) + dataLength);
69 TCP_Struct_Frame->Sequence_Number = dat_ack;
70 TCP_Struct_Frame->TCP_Flags = TCP_ACK;

```

```

70 TCP_Struct_Frame->TCP_Checksums=0;
71 TCP_Struct_Frame->Urgent_Pointer=0;
72 TCP_Struct_Frame->CheckSum = NET_ipchecksum((uint8_t *)&TCP_Struct_Frame->Header_length); //tinh checksum
    cho goi IO
73 TCP_Struct_Frame->TCP_Checksums = TCP_checksum(TCP_Struct_Frame);
74 }
75 }

```

Và định nghĩa thêm các tham số của đối số **type** vào file **tcp.h**



```

1 #define FOR_SYN    0
2 #define FOR_FIN    1
3 #define FOR_PSH_ACK 2

```

Hàm **TCP_make_header** sẽ kiểm tra tham số **type** để xử lí Struct tcp cho phù hợp

Bây giờ trong hàm **TCP_read**, chúng ta có thể viết gọn lại bằng cách gọi hàm **TCP_make_header**. Đây là hàm **TCP_read** gọn nhẹ !



```

1 void TCP_read(uint8_t *TCP_Frame,uint16_t len)
2 {
3     TCP_struct *TCP_Struct_Frame = (TCP_struct *)TCP_Frame;
4     //kiem tra dia chi ip xem co phai no gui cho minh khong
5     if( memcmp(TCP_Struct_Frame->DestIP,ip,4) )return; // dung memcmp de so sanh, neu khac thi thoat
6     if(TCP_Struct_Frame->TCP_Flags == TCP_SYN)
7     {
8         TCP_make_header(TCP_Struct_Frame,len,FOR_SYN);
9         NET_SendFrame((uint8_t *)TCP_Struct_Frame,len); //gui goi tin tcp reply
10    }

```

```

11 else if(TCP_Struct_Frame->TCP_Flags == TCP_ACK)
12 {
13   UART_putString("Ack\r\n");
14 }
15 else if(TCP_Struct_Frame->TCP_Flags == (TCP_FIN|TCP_ACK))
16 {
17   TCP_make_header(TCP_Struct_Frame,len,FOR_FIN);
18   NET_SendFrame((uint8_t *)TCP_Struct_Frame,len); //gui goi tin tcp reply
19
20   TCP_Struct_Frame->TCP_Flags = TCP_FIN|TCP_ACK;
21   TCP_Struct_Frame->TCP_Checksums=0;
22   TCP_Struct_Frame->TCP_Checksums = TCP_checksum(TCP_Struct_Frame);
23
24   NET_SendFrame((uint8_t *)TCP_Struct_Frame,len); //gui goi tin tcp reply
25 }
26 else if(TCP_Struct_Frame->TCP_Flags == (TCP_PSH|TCP_ACK))
27 {
28   TCP_make_header(TCP_Struct_Frame,len,FOR_PSH_ACK);
29   if (strcmp((char*)TCP_Struct_Frame->data,"GET /", 5) == 0)
30   {
31     //co 1 request gui toi kia
32   }
33 }
34}

```

OK ! Bây giờ chúng ta sẽ bắt đầu gửi trả nội dung trang web cho client khi có 1 request gửi tới

Do trang web của chúng ta khá dài nên sẽ phải chia thành nhiều gói nhỏ để gửi. Khi gửi nhiều gói tin :

Phương pháp đơn giản nhất là : Khi server trả về cho client 1 gói tin, server sẽ chờ nhận được phản hồi ACK của client để gửi tiếp, mà cái phản hồi ACK đấy phải chính xác là của gói tin mà server đã gửi. Chúng ta sẽ check Sequence Number (số hiệu nhận dạng gói tin) xem có trùng với Sequence Number lúc gửi đi không ! Nếu trùng thì có thể chắc chắn 100% client đã nhận được chính xác gói tin mà server đã gửi cho nó. Lúc này chúng ta mới gửi gói tin tiếp theo!

Thực ra đáng lẽ còn phải có 1 trường hợp nữa là server sẽ làm gì khi chờ mãi mà không nhận được ACK phản hồi, nhưng nó sẽ làm phức tạp hóa vấn đề lên rất nhiều. Chúng ta sẽ tạm bỏ qua các vấn đề này và xử lý nó khi làm việc với phương pháp lập trình non-blocking

Mình tạo thêm biến **status** trong thư viện **http.c** và sẽ sử dụng nó với mục đích là kiểm tra xem server có đang “bắn” trả lời request GET cho thằng nào không. Nếu == 1 thì là đang bắn rồi, thằng nào đang GET thì chờ chút đê. Nếu == 0 thì là đang rảnh, có thể phản hồi ngay

Sử lại hàm **TCP_send** 1 chút

Ở hàm cũ thì mình chỉ nhận dữ liệu và tự tính toán độ dài dữ liệu trong hàm **TCP_send**

Mình sẽ sửa lại thành nhận dữ liệu kèm cả độ dài của dữ liệu và thêm 1 cái offset để căn chỉnh dữ liệu cho tiện. Bởi vì data web của chúng ta lưu vào bộ nhớ **flash** của chip atmega328, mà truy cập flash ở AVR có hơi vất và hơn các dòng chip khác 1 chút.

Đây là hàm **TCP_send** viết lại



```
1 void TCP_send(TCP_struct *TCP_Struct_Frame,uint16_t len,flash uint8_t *data,uint16_t data_length)
2 {
3     uint16_t i;
4     for(i=0;i<data_length;i++)TCP_Struct_Frame->data[i] = data[i];
5     len+=data_length;
6     TCP_Struct_Frame->TotoLength = swap16(swap16(TCP_Struct_Frame->TotoLength) + data_length); //make totolength
7     TCP_Struct_Frame->CheckSum=0;
8     TCP_Struct_Frame->TCP_Checksums=0;
9     TCP_Struct_Frame->TCP_Flags = TCP_PSH|TCP_ACK;
10
11    TCP_Struct_Frame->CheckSum = NET_ipchecksum((uint8_t *)&TCP_Struct_Frame->Header_length); //tinh checksum
12    cho goi IO
13    TCP_Struct_Frame->TCP_Checksums = TCP_checksum(TCP_Struct_Frame);
14
15    NET_SendFrame((uint8_t *)TCP_Struct_Frame,len);
16 }
```

Như vậy khi gọi hàm **TCP_send** chúng ta phải nhập vào thêm 2 thông tin đó là:

- **data** : Đây là 1 con trỏ dạng flash trả tới nơi lưu trữ nội dung web được lưu ở bộ nhớ flash
- **data_length** : Độ dài của data sẽ gửi đi là bao nhiêu (0 -> 458)

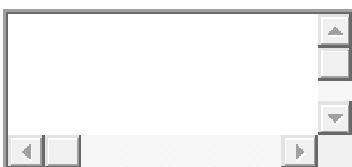
Thư viện http sẽ có nhiệm vụ cung cấp cho bên tcp biết 2 thông số này, vì vậy mình sẽ viết 1 hàm làm việc đó



```
1 uint8_t HTTP_get_data_num(flash uint8_t** data,uint16_t* data_length)
2 {
3     if(data_num >= sizeof(web_page)) //da gui xong data, dong ket noi
4     {
5         UART_putString("da gui xong tap tin html\r\n");
6         return 1;//end
7     }
8     else if( (sizeof(web_page) - data_num) > 458 ) //neu du lieu can gui > 458 (con gui tiep nua)
9     *data_length=458;
10    else if( (sizeof(web_page) - data_num) <= 458 ) //neu du lieu can gui <= 458 (gui lan cuoi)
11    *data_length = sizeof(web_page) - data_num;
12    *data=web_page+data_num;
13    data_num+=*data_length;
14    return 0;
15}
```

Ngoài việc cung cấp 2 thông số **data** và **data_length** qua phương pháp truy cập gián tiếp qua con trỏ (đối với **data_length**) và thông qua con trỏ 2 cấp (đối với **data** ở dạng flash). Hàm **HTTP_get_data_num** còn trả về 1 – tức dữ liệu gửi đi hết rồi, và 0 – vẫn còn dữ liệu để gửi tiếp

<tại hàm **TCP_read**> chõ //có 1 request gửi tới kia
Chúng ta sẽ bắt đầu trả lời gói tin đầu tiên ngay tại đây



```
1     if(HTTP_get_status() != 1)
2     {
3         HTTP_pack_init(); //bat dau gui 1 goi tin
4         TCP_make_header(TCP_Struct_Frame,len,FOR_PSH_ACK);
5         if(HTTP_get_data_num(&data_send,&data_send_len) == 0)
6         {
```

```

7     TCP_send(TCP_Struct_Frame,len,data_send,data_send_len);
8     HTTP_set_Sequence_Number(TCP_Struct_Frame->Acknowledgement); //luu so hieu goi tin da gui lai
9 }
10 }
```

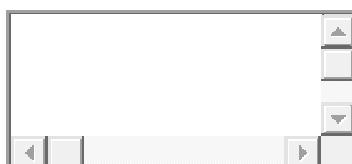
Các gói tin tiếp theo sẽ được gửi khi chúng ta nhận được cờ ACK
 <Tại chỗ nhận cờ ACK trong hàm **TCP_read**>



```

1 if(HTTP_get_Status() == 1) //neu dang co nhan vu gui cac goi tin http
2 {
3     if(TCP_Struct_Frame->Sequence_Number == HTTP_get_Sequence_Number()) //neu goi tin truoc da thanh cong
4     {
5         TCP_make_header(TCP_Struct_Frame,len,FOR_PSH_ACK);
6         if(HTTP_get_data_num(&data_send,&data_send_len) == 0)
7         {
8             TCP_send(TCP_Struct_Frame,len,data_send,data_send_len);
9             HTTP_set_Sequence_Number(TCP_Struct_Frame->Acknowledgement); //luu so hieu goi tin da gui lai
10        }
11    else
12    {
13
14    }
15  }
16 }
```

Tổng quan, toàn bộ hàm **TCP_read** lúc này sẽ như sau:



```
1 void TCP_read(uint8_t *TCP_Frame,uint16_t len)
```

```

2 {
3     flash uint8_t *data_send=0;
4     uint16_t data_send_len=0;
5
6     TCP_struct *TCP_Struct_Frame = (TCP_struct *)TCP_Frame;
7     //kiem tra dia chi ip xem co phai no gui cho minh khong
8     if( memcmp(TCP_Struct_Frame->DestIP,ip,4) )return; // dung memcmp de so sanh, neu khac thi thoat
9     if(TCP_Struct_Frame->TCP_Flags == TCP_SYN)
10    {
11        TCP_make_header(TCP_Struct_Frame,len,FOR_SYN);
12        NET_SendFrame((uint8_t *)TCP_Struct_Frame,len); //gui goi tin tcp reply
13    }
14 else if(TCP_Struct_Frame->TCP_Flags == TCP_ACK)
15    {
16     UART.putString("Ack\r\n");
17     if(HTTP_get_status()==1) //neu dang co nhan vu gui cac goi tin http
18    {
19        if(TCP_Struct_Frame->Sequence_Number==HTTP_get_Sequence_Number()) //neu goi tin truoc da gui thanh cong
20    {
21        TCP_make_header(TCP_Struct_Frame,len,FOR_PSH_ACK);
22        if(HTTP_get_data_num(&data_send,&data_send_len) == 0)
23        {
24            TCP_send(TCP_Struct_Frame,len,data_send,data_send_len);
25            HTTP_set_Sequence_Number(TCP_Struct_Frame->Acknowledgement); //luu so hieu goi tin da gui lai
26        }
27    else
28    {
29        //gui het roi,khong con gi de gui nua
30    }
31    }
32    }
33 }
34 else if(TCP_Struct_Frame->TCP_Flags == (TCP_FIN|TCP_ACK))

```

```

35 {
36     TCP_make_header(TCP_Struct_Frame,len,FOR_FIN);
37     NET_SendFrame((uint8_t *)TCP_Struct_Frame,len); //gui goi tin tcp reply
38
39     TCP_Struct_Frame->TCP_Flags = TCP_FIN|TCP_ACK;
40     TCP_Struct_Frame->TCP_Checksums=0;
41     TCP_Struct_Frame->TCP_Checksums = TCP_checksum(TCP_Struct_Frame);
42
43     NET_SendFrame((uint8_t *)TCP_Struct_Frame,len); //gui goi tin tcp reply
44 }
45 else if(TCP_Struct_Frame->TCP_Flags == (TCP_PSH|TCP_ACK))
46 {
47     TCP_make_header(TCP_Struct_Frame,len,FOR_PSH_ACK);
48     if (strcmp((char*)TCP_Struct_Frame->data,"GET /", 5) == 0)
49     {
50         if(HTTP_get_status() != 1)
51         {
52             HTTP_pack_init(); //bat dau gui 1 goi tin
53             if(HTTP_get_data_num(&data_send,&data_send_len) == 0)
54             {
55                 TCP_send(TCP_Struct_Frame,len,data_send,data_send_len);
56                 HTTP_set_Sequence_Number(TCP_Struct_Frame->Acknowledgement); //luu so hieu goi tin da gui lai
57             }
58         }
59     }
60 }
61 }
```

Vát vả rồi 😞 chạy mô phỏng và test thử thôi



Pơ phêch ! Web server của chúng ta đã hiện nguyên hình

Chúng ta sẽ hoàn thành nốt công việc đóng kết nối với client (thanh loadding sẽ ngừng quay) và gọi hàm xóa biến status về 0 để báo server rảnh, sẵn sàng phản hồi các truy vấn khác

tại chỗ *//gửi hết rồi, không còn gì để gửi nữa*



```
1HTTP_set_status(0);
```

```
2 //đóng kết nối
```

```
3     TCP_Struct_Frame->TCP_Flags = TCP_FIN|TCP_ACK;
4     TCP_Struct_Frame->TCP_Checksums=0;
5     TCP_Struct_Frame->TCP_Checksums = TCP_checksum(TCP_Struct_Frame);
6
7     NET_SendFrame((uint8_t *)TCP_Struct_Frame,len); //gui goi tin ngat ket noi
```

Chạy thử



OK rồi ! Trình duyệt web đã ngừng loadding sau khi đã nhận xong data !

Các bạn có thể xóa phần in dữ liệu debug ra màn hình trong hàm `TCP_make_header` để chương trình phản hồi nhanh hơn (nhanh hơn hẳn luôn @@)

```

75 //tinh do dai cua goi data va in ra man hinh
76 dataLength= swap16(TCP_Struct_Frame->TotalLength) - 20 - (TCP_Struct_Frame->data_offset >> 2); // (>> 4)*4 = >> 2
77 // for(i=0;i<dataLength;i++)
78 //     UART_putChar(TCP_Struct_Frame->data[i]);
79

```

Xóa luôn máy dòng debug trong thư viện http

```

} uint8_t HTTP_get_data_num(flash uint8_t** data,uint16_t* data_length)
{
    if(data_num >= sizeof(web_page)) //da gui xong data, dong ket noi
    {
        //UART_putString("da gui xong tap tin html\r\n");
        return 1;//end
    }
}

```

Nói chung chỗ nào có chạy hàm UART thì xóa đi để web load ở tốc độ bàn thờ

Điều khiển thiết bị

Tiếp theo là tiết mục điều khiển thiết bị thành thảnh

Trước lệnh kiểm tra “GET / ” trong hàm **TCP_read** chỗ nhận cờ **TCP_PSH|TCP_ACK**, các bạn thêm vài dòng lệnh so sánh chuỗi rồi bật tắt thiết bị là được.



```

1 if strstr(TCP_Struct_Frame->data,"bat1"))PORTC.0=0;
2 else if strstr(TCP_Struct_Frame->data,"tat1"))PORTC.0=1;
3 else if strstr(TCP_Struct_Frame->data,"bat2"))PORTC.1=0;
4 else if strstr(TCP_Struct_Frame->data,"tat2"))PORTC.1=1;
5 else if strstr(TCP_Struct_Frame->data,"bat3"))PORTC.2=0;
6 else if strstr(TCP_Struct_Frame->data,"tat3"))PORTC.2=1;
7 else if strstr(TCP_Struct_Frame->data,"bat4"))PORTC.3=0;
8 else if strstr(TCP_Struct_Frame->data,"tat4"))PORTC.3=1;

```

Download

Tải full source bài 14 [tại đây](#)

[ENC28J60] Bài 15: Giao thức DHCP – lấy IP động



guao-thuc-dhcp

Trong các bài học trước, chúng ta đã quen với việc sử dụng ip tĩnh được gán sẵn trong chương trình. Tiếp tục bài này, chúng ta sẽ làm quen với giao thức DHCP . DHCP – **Dynamic Host Configuration Protocol** giúp cấp ip một cách tự động và lấy các thông tin cấu hình khác như gateway

Quá trình đạt được địa chỉ của giao thức dhcp gồm các bước sau:

- Bước 1: Thiết bị phát tán 1 thông điệp DHCP Discover có chứa địa chỉ MAC, tên thiết bị ...
- Bước 2: Máy chủ nhận thông điệp và chuẩn bị địa chỉ IP cho thiết bị và phát tán bản tin offer lên mạng
- Bước 3: Thiết bị nhận thông điệp, lấy địa chỉ IP và gửi bản tin DHCP Request
- Bước 4: Máy chủ nhận bản tin và trả lời lại bằng bản tin ACK

Cấu trúc gói tin

Gói tin **Discovery**

Example DHCPDISCOVER message

Ethernet: source=sender's MAC; destination=FF:FF:FF:FF:FF:FF						
IP: source=0.0.0.0; destination=255.255.255.255						
UDP: source port=68; destination port=67						
Octet 0	Octet 1	Octet 2	Octet 3			
OP	HTYPE	HLEN	HOPS			
0x01	0x01	0x06	0x00			
XID						
0x3903F326						
SECS	FLAGS					
0x0000	0x0000					
CIADDR (Client IP address)						
0x00000000						
YIADDR (Your IP address)						
0x00000000						
SIADDR (Server IP address)						
0x00000000						
GIADDR (Gateway IP address)						
0x00000000						
CHADDR (Client hardware address)						
0x00053C04						
0x8D590000						
0x00000000						
0x00000000						
192 octets of 0s, or overflow space for additional options; BOOTP legacy .						
Magic cookie						
0x63825363						
DHCP options						
0x350101 53: 1 (DHCP Discover)						
0x3204c0a80164 50: 192.168.1.100 requested						
0x370401030f06 55 (Parameter Request List):						
<ul style="list-style-type: none"> • 1 (Request Subnet Mask), • 3 (Router), • 15 (Domain Name), • 6 (Domain Name Server) 						
0xff 255 (Endmark)						

Gói tin Offer

DHCPOFFER message

Ethernet: source=sender's MAC; destination=client mac address						
IP: source=192.168.1.1; destination=255.255.255.255						
UDP: source port=67; destination port=68						
Octet 0	Octet 1	Octet 2	Octet 3			
OP	HTYPE	HLEN	HOPS			
0x02	0x01	0x06	0x00			
XID						
0x3903F326						
SECS	FLAGS					
0x0000	0x0000					
CIADDR (Client IP address)						
0x00000000						
YIADDR (Your IP address)						
0xC0A80164 (192.168.1.100)						
SIADDR (Server IP address)						
0xC0A80101 (192.168.1.1)						
GIADDR (Gateway IP address)						
0x00000000						
CHADDR (Client hardware address)						
0x00053C04						
0x8D590000						
0x00000000						
0x00000000						
192 octets of 0s; BOOTP legacy.						
Magic cookie						
0x63825363						
DHCP options						
53: 2 (DHCP Offer)						
1 (subnet mask): 255.255.255.0						
3 (Router): 192.168.1.1						
51 (IP address lease time): 86400s (1 day)						
54 (DHCP server): 192.168.1.1						
6 (DNS servers):						
<ul style="list-style-type: none"> • 9.7.10.15, • 9.7.10.16, • 9.7.10.18 						

Gói tin Request

DHCPREQUEST message					
Ethernet: source=sender's MAC; destination=FF:FF:FF:FF:FF:FF					
IP: source=0.0.0.0; destination=255.255.255.255; ^[a] UDP: source port=68; destination port=67					
Octet 0	Octet 1	Octet 2	Octet 3		
OP	HTYPE	HLEN	HOPS		
0x01	0x01	0x06	0x00		
XID					
0x3903F326					
SECS		FLAGS			
0x0000		0x0000			
CIADDR (Client IP address)					
0x00000000					
YIADDR (Your IP address)					
0x00000000					
SIADDR (Server IP address)					
0xC0A80101 (192.168.1.1)					
GIADDR (Gateway IP address)					
0x00000000					
CHADDR (Client hardware address)					
0x00053C04					
0x8D590000					
0x00000000					
0x00000000					
192 octets of 0s; BOOTP legacy.					
Magic cookie					
0x63825363					
DHCP options					
53: 3 (DHCP Request)					
50: 192.168.1.100 requested					
54 (DHCP server): 192.168.1.1					

Gói tin Acknowledgement

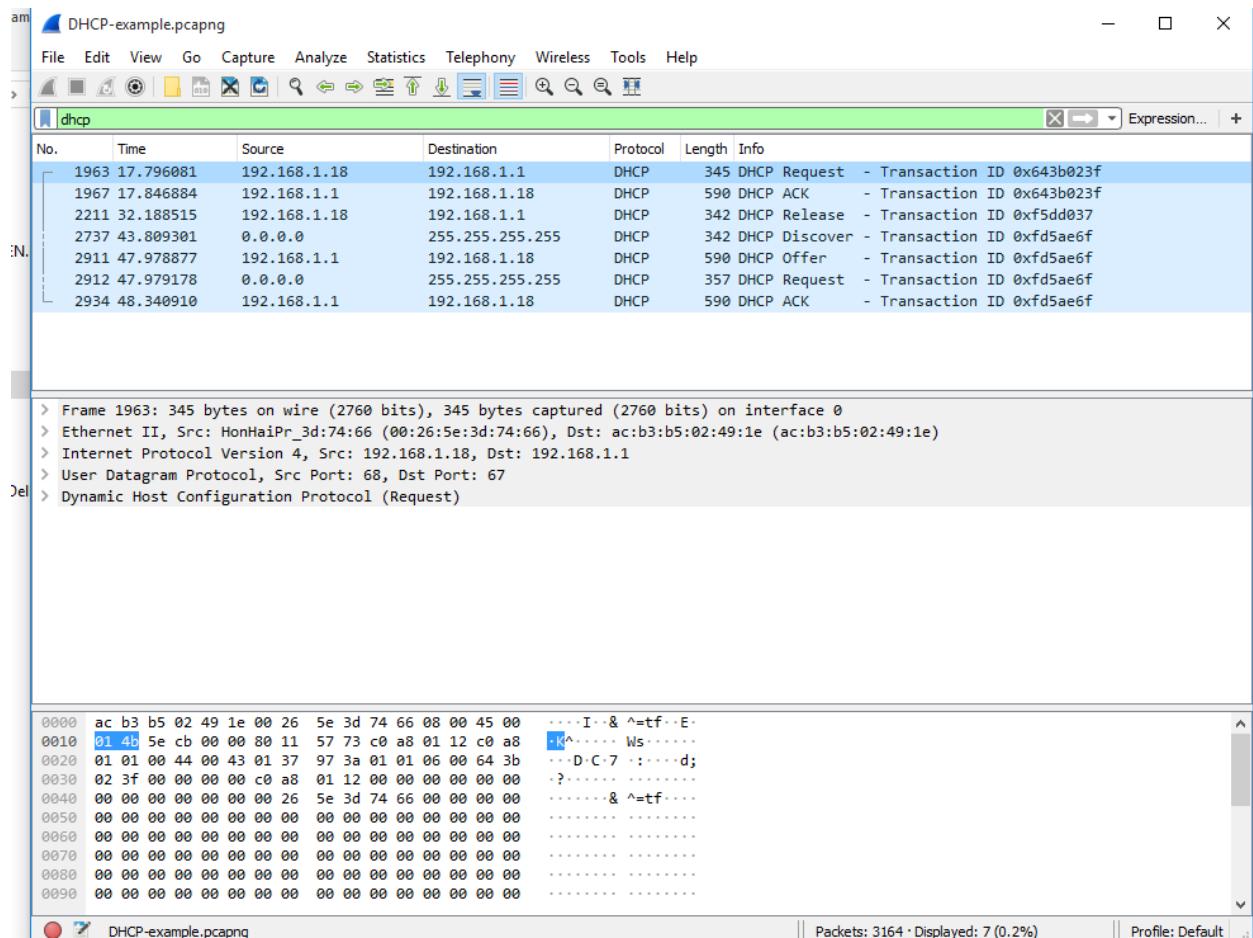
DHCPACK message

Ethernet: source=sender's MAC; destination=client's MAC			
IP: source=192.168.1.1; destination=255.255.255.255			
UDP: source port=67; destination port=68			
Octet 0	Octet 1	Octet 2	Octet 3
OP	HTYPE	HLEN	HOPS
0x02	0x01	0x06	0x00
XID			
0x3903F326			
SECS		FLAGS	
0x0000		0x0000	
CIADDR (Client IP address)			
0x00000000			
YIADDR (Your IP address)			
0xC0A80164 (192.168.1.100)			
SIADDR (Server IP address)			
0xC0A80101 (192.168.1.1)			
GIADDR (Gateway IP address switched by relay)			
0x00000000			
CHADDR (Client hardware address)			
0x00053C04			
0x8D590000			
0x00000000			
0x00000000			
192 octets of 0s. BOOTP legacy			
Magic cookie			
0x63825363			
DHCP options			
53: 5 (DHCP ACK) or 6 (DHCP NAK)			
1 (subnet mask): 255.255.255.0			
3 (Router): 192.168.1.1			
51 (IP address lease time): 86400s (1 day)			
54 (DHCP server): 192.168.1.1			
6 (DNS servers):			
• 9.7.10.15,			
• 9.7.10.16,			
• 9.7.10.18			

Các bạn tìm hiểu chi tiết hơn tại đây

- https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol
- <https://medium.com/@bromiley/full-packet-friday-dhcp-abbc6b7b3c77>

Tải file chụp gói tin bằng wireshark **tại đây**



Dùng bộ lọc để lọc ra các gói tin dhcp nhé

Lập trình

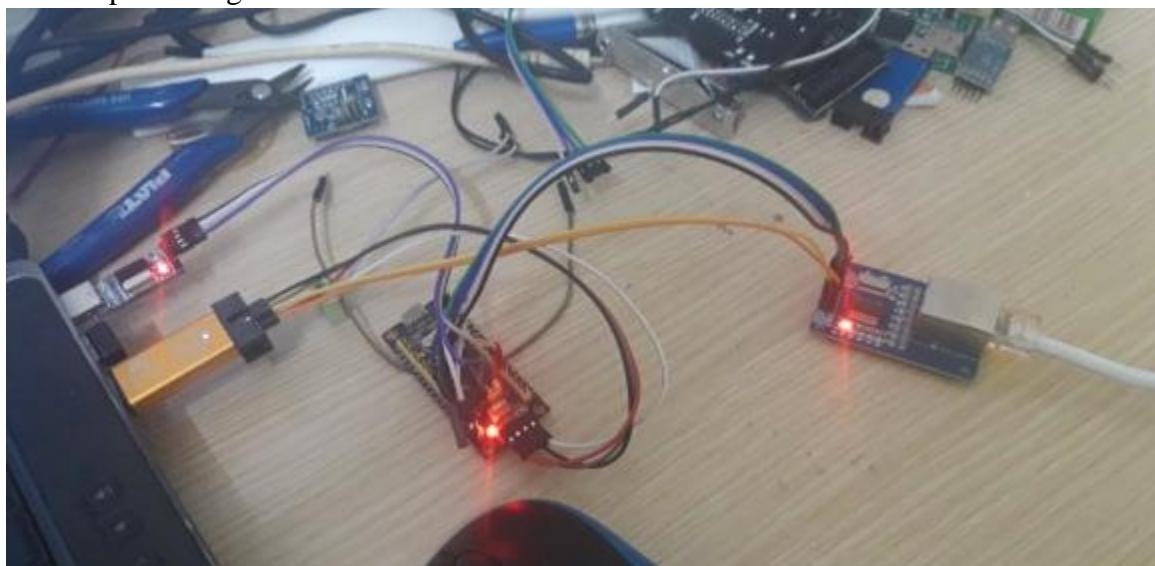
Do phần mềm mô phỏng không hỗ trợ giao thức dhcp và các giao thức về sau nên bắt đầu từ bài này mình sẽ chuyển sang dùng module thật. Phần cứng mình cũng sẽ chuyển luôn qua kit stm32f103c8t6 vì mình đang có kit này :v

Chuẩn bị đồ nghề

- Với stm32f103c8t6 thì mình sử dụng thư viện HAL, nên các bạn chuẩn bị HAL CUBEMX để sinh code nhé
- Module USB-TTL để debug dữ liệu, phần mềm Hercules
- Kit STM32, mạch nạp

- Moudle ENC28J60
- Modem mạng, dây nối ...

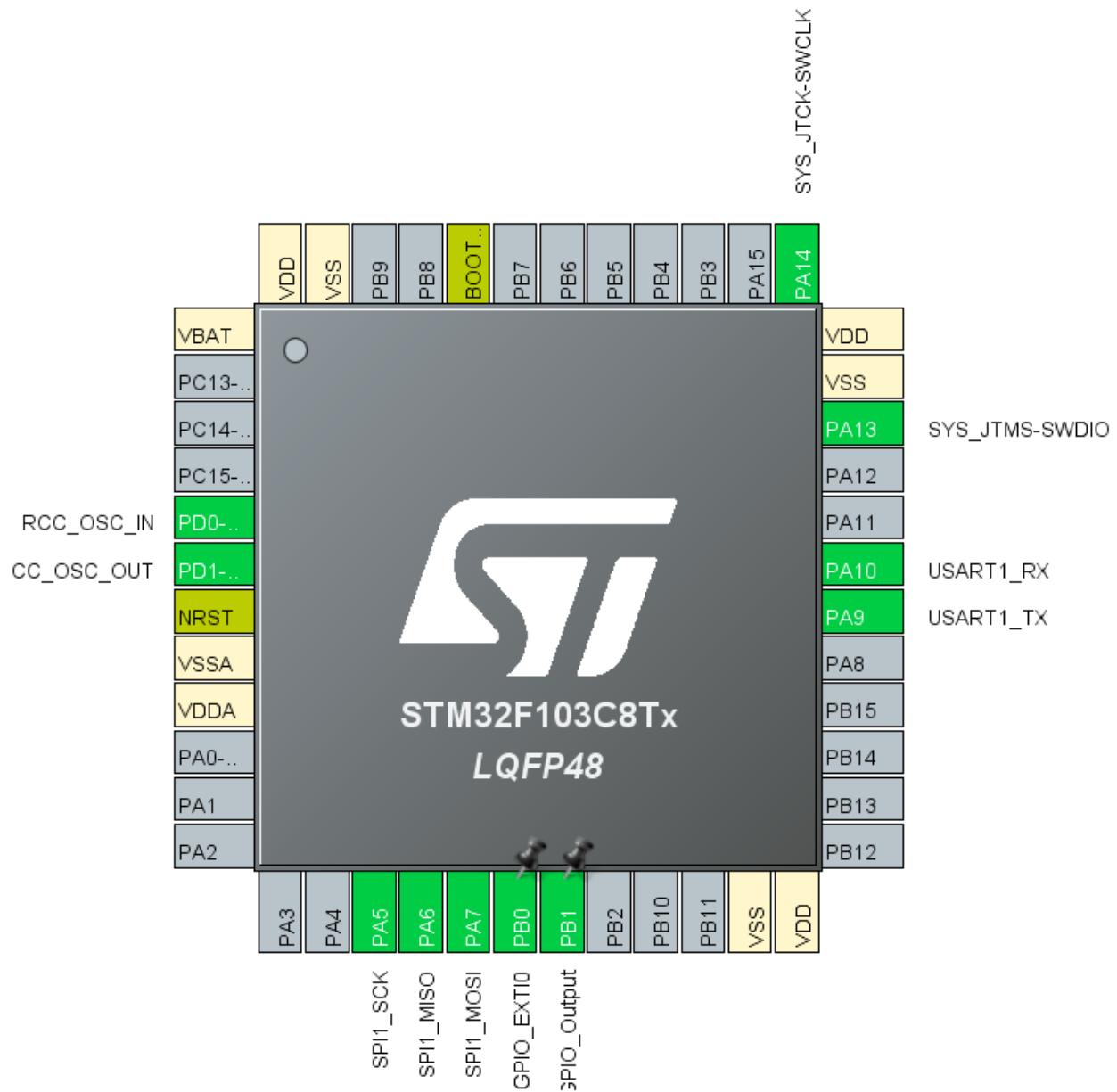
Kết nối phần cứng



ENC28J60	STM32F103C6
MOSI	A7
MISO	A6
SCK	A5
CS	B1
INT	B0
GND	GND
5V	5V
PL2303 (USB – TTL)	STM32F103C8
RX	A9
GND	GND

Nói chung, mình sẽ dùng bộ SPI1 và USART1 để debug

Khởi tạo project với CUBEMX



RCC Mode and Configuration

Mode

High Speed Clock (HSE)

Low Speed Clock (LSE)

Master Clock Output

Categories A-Z

- DMA
- GPIO
- IWDG
- NVIC
- RCC
- SYS
- WWDG

Bật thạch anh ngoài

Screenshot of the STM32CubeMX software interface showing the configuration of Timer 2.

Left Panel (Hardware Catalog):

- DMA
- GPIO
- IWDG
- NVIC
- RCC**
- SYS**
- WWDG

Analog

Timers

- RTC
- TIM1
- TIM2** (highlighted with a blue arrow)
- TIM3** (highlighted with a red arrow)
- TIM4

Connectivity

- CAN
- I2C1
- I2C2
- SPI1**
- SPI2
- USART1**
- USART2
- USART3
- USB

Computing

Right Panel (Configuration):

Timer 2 Configuration:

- Slave Mode: Disable
- Trigger Source: Disable
- Clock Source: Internal Clock** (highlighted with a red arrow)
- Channel1: Disable
- Channel2: Disable
- Channel3: Disable
- Channel4: Disable
- Combined Channels: Disable

Checkboxes:

- Use ETR as Clearing Source
- XOR activation
- One Pulse Mode

Configuration Tab:

Reset Configuration

<input checked="" type="checkbox"/> NVIC Settings	<input checked="" type="checkbox"/> DMA Settings
<input checked="" type="checkbox"/> Parameter Settings	<input checked="" type="checkbox"/> User Constants

Configure the below parameters :

Search (Ctrl+F) ⌂ ⌂

Counter Settings:

- Prescaler (PSC - 16 bits ... 7200-1) (highlighted with a red arrow)
- Counter Mode: Up
- Counter Period (AutoRelo... 10000) (highlighted with a red arrow)
- Internal Clock Division (C... No Division)
- auto-reload preload: Disable

Cấu hình timer 2

Additional S

Categories A->Z

DMA
GPIO
IWDG
NVIC
✓ RCC
✓ SYS
WWDG

Analog >

Timers >

RTC
TIM1
✓ TIM2
▲ TIM3
TIM4

Connectivity >

CAN
I2C1
I2C2
✓ SPI1
SPI2
✓ USART1
USART2
USART3
USB

Computing >

CRC

USART1 Mode and Configuration

Mode Asynchronous

Hardware Flow Control (RS232) Disable

Configuration

Reset Configuration

DMA Settings GPIO Settings

Parameter Settings User Constants NVIC Settings

Configure the below parameters :

Search (Ctrl+F)

Basic Parameters

Baud Rate	9600 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

Advanced Parameters

Data Direction	Receive and Transmit
Over Sampling	16 Samples

Cấu hình USART1

The screenshot shows the STM32CubeMX software interface. On the left, there is a tree view of hardware components:

- Categories**: DMA, GPIO, IWDG, NVIC, RCC, SYS, WWDG.
- Analog**: RTC, TIM1, TIM2, TIM3, TIM4.
- Timers**: RTC, TIM1, TIM2, TIM3, TIM4.
- Connectivity**: CAN, I2C1, I2C2, SPI1, SPI2, USART1, USART2, USART3, USB.

The **SPI1** node under Connectivity is selected and highlighted in blue.

SPI1 Mode and Configuration tab (top right):

- Mode: Full-Duplex Master
- Hardware NSS Signal: Disable

Configuration tab (bottom right):

- Reset Configuration
- Checkboxes: DMA Settings (checked), GPIO Settings (checked), Parameter Settings (checked), User Constants (checked), NVIC Settings (checked).
- Configure the below parameters:
 - Frame Format: Motorola
 - Data Size: 8 Bits
- Search (Ctrl+F) input field.
- Basic Parameters section with Frame Format set to Motorola and Data Size set to 8 Bits.

Cấu hình SPI1

The screenshot shows the STM32CubeMX software interface. On the left, there is a tree view of hardware components:

- Categories**: DMA, GPIO, IWDG, NVIC, RCC, SYS, WWDG.
- GPIO**: PB0, PB1.

The **GPIO** node under Categories is selected and highlighted in blue.

Search Signals section (top right):

- Search (Ctrl+F) input field.
- Show only Modified Pins checkbox (unchecked).

Pin ... table (bottom right):

Pin ...	Signal o...	GPIO ou...	GPIO m...	GPIO P...	Maximu...	User Label	Modified
PB0	n/a	n/a	External...	Pull-up	n/a		<input checked="" type="checkbox"/>
PB1	n/a	High	Output ...	Pull-up	High		<input checked="" type="checkbox"/>

Cấu hình GPIO, ngắt

System Core

- DMA
- GPIO
- IWDG
- NVIC**
- RCC
- SYS
- WWDG

Analog

Timers

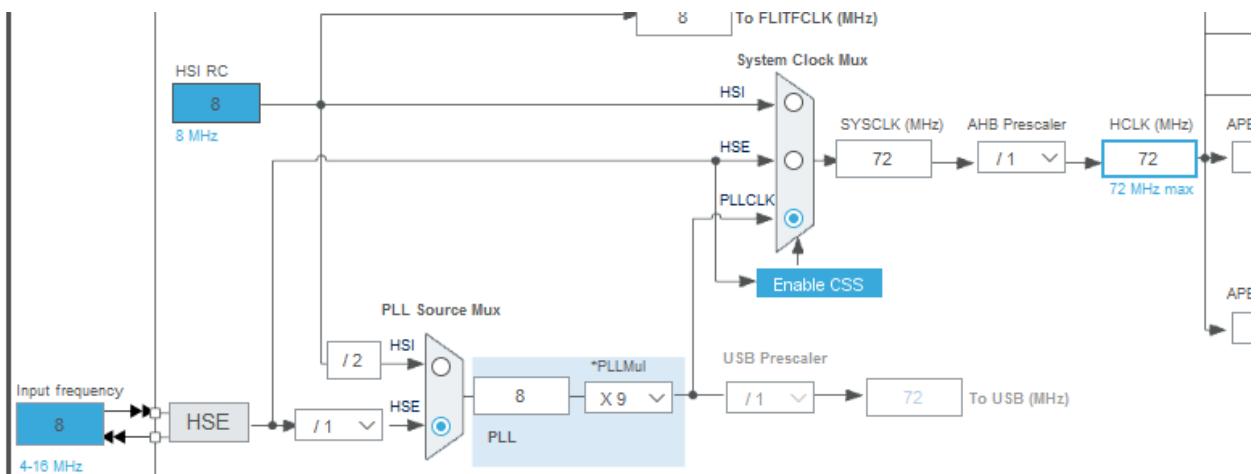
- RTC
- TIM1
- TIM2**
- TIM3**
- TIM4

Connectivity

NVIC Interrupt Table	Enabled	Preemption Priority
Non maskable interrupt	<input checked="" type="checkbox"/>	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0
Memory management fault	<input checked="" type="checkbox"/>	0
Prefetch fault, memory access fault	<input checked="" type="checkbox"/>	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0
Debug monitor	<input checked="" type="checkbox"/>	0
Pendable request for system service	<input checked="" type="checkbox"/>	0
Time base: System tick timer	<input checked="" type="checkbox"/>	0
PVD interrupt through EXTI line 16	<input type="checkbox"/>	0
Flash global interrupt	<input type="checkbox"/>	0
RCC global interrupt	<input type="checkbox"/>	0
EXTI line0 interrupt	<input checked="" type="checkbox"/>	0
TIM2 global interrupt	<input checked="" type="checkbox"/>	0
SPI1 global interrupt	<input type="checkbox"/>	0
USART1 global interrupt	<input type="checkbox"/>	0

Kích hoạt ngắt timer và ngắt ngoài

Trong phần Clock Configuration, chọn thạch anh 8M, tốc độ 72Mhz



Sinh code tùy theo trình dịch của bạn, sau đó đó tải thư viện cho các bài trước mà mình đã “remake” lại cho phù hợp với thư viện HAL. Còn ở đây, mình sử dụng trình dịch KeilC

Các bạn tải thư viện [tại đây](#)

Hướng dẫn add thư viện vào KeilC

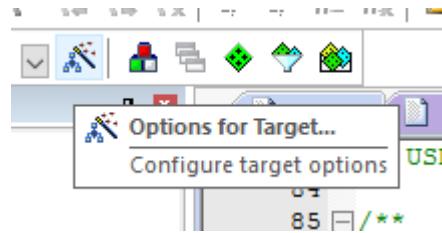
Các bạn sử dụng trình dịch khác thì tham khảo cách add thư viện trên google nhé

Sau khi sinh code và mở project, các bạn giải nén và copy thư mục thư viện tên là “IOT47_lib” vào thư mục **Drivers**

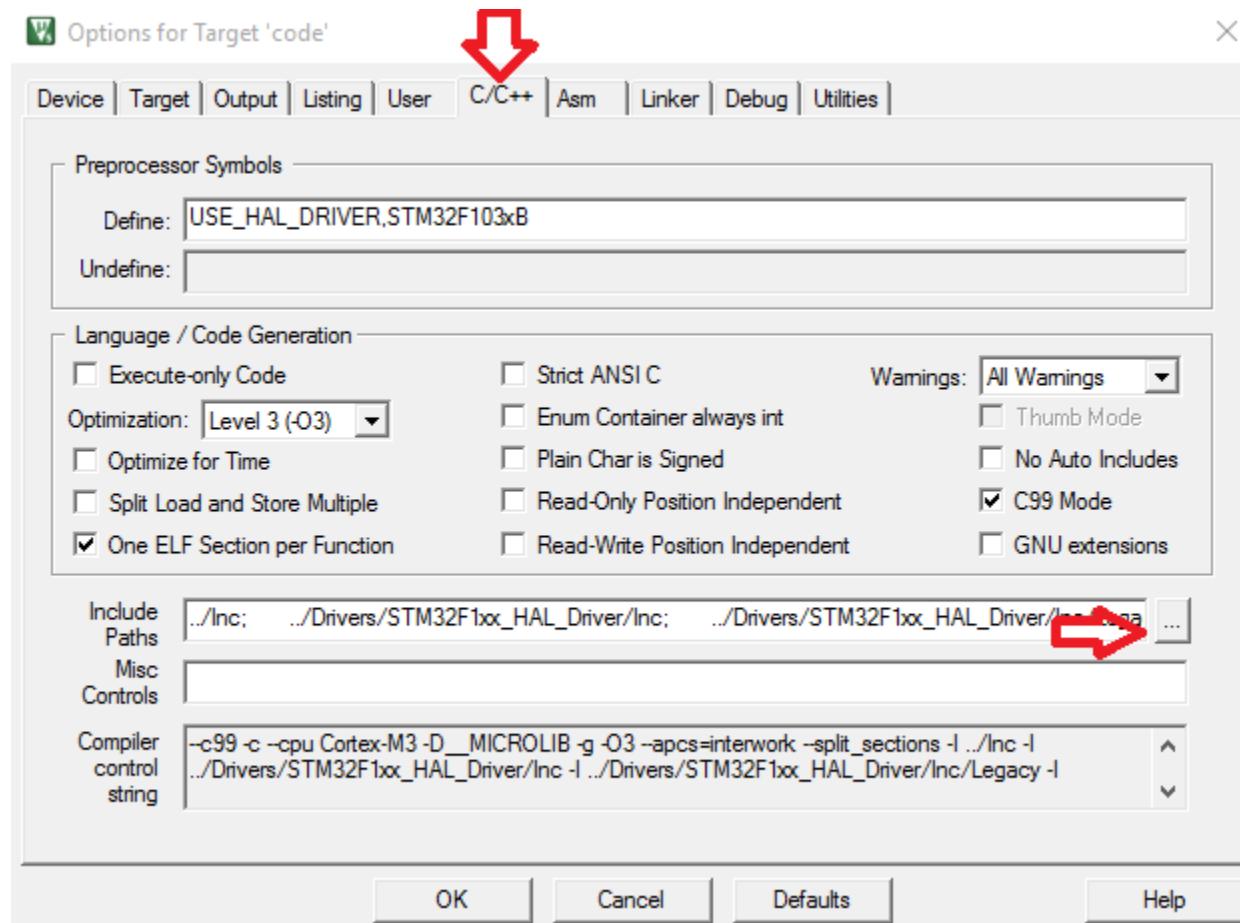
CMSIS	15/05/2020 09:23	File folder
IOT47_lib	15/05/2020 09:25	File folder
STM32F1xx_HAL_Driver	15/05/2020 09:23	File folder

copy thư viện vào

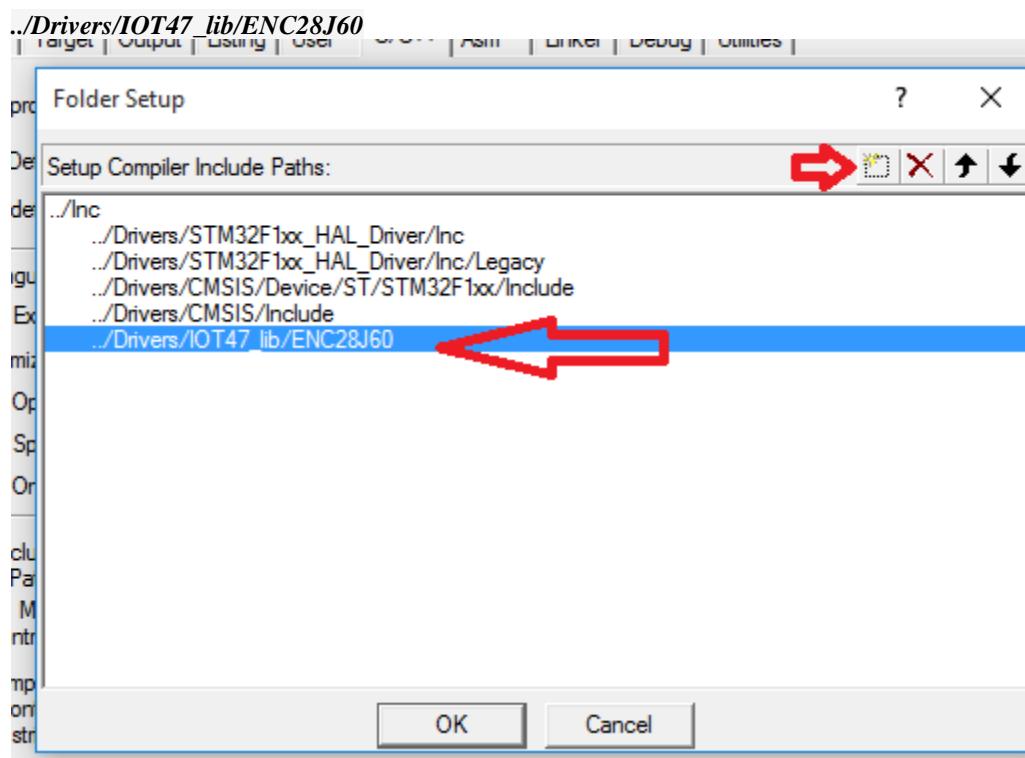
Trong KeilC, click vào Option for Target



Chọn thẻ **C/C++ -> Include Paths**

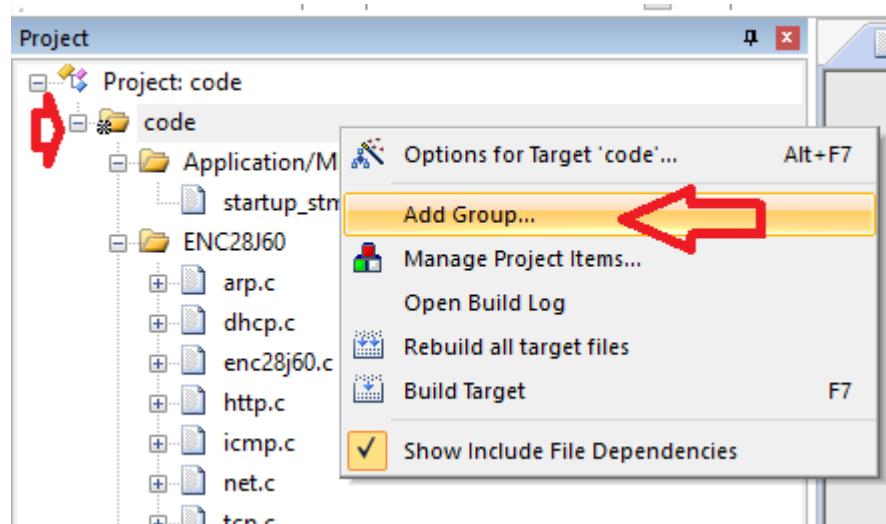


Add dòng này vào dưới cùng

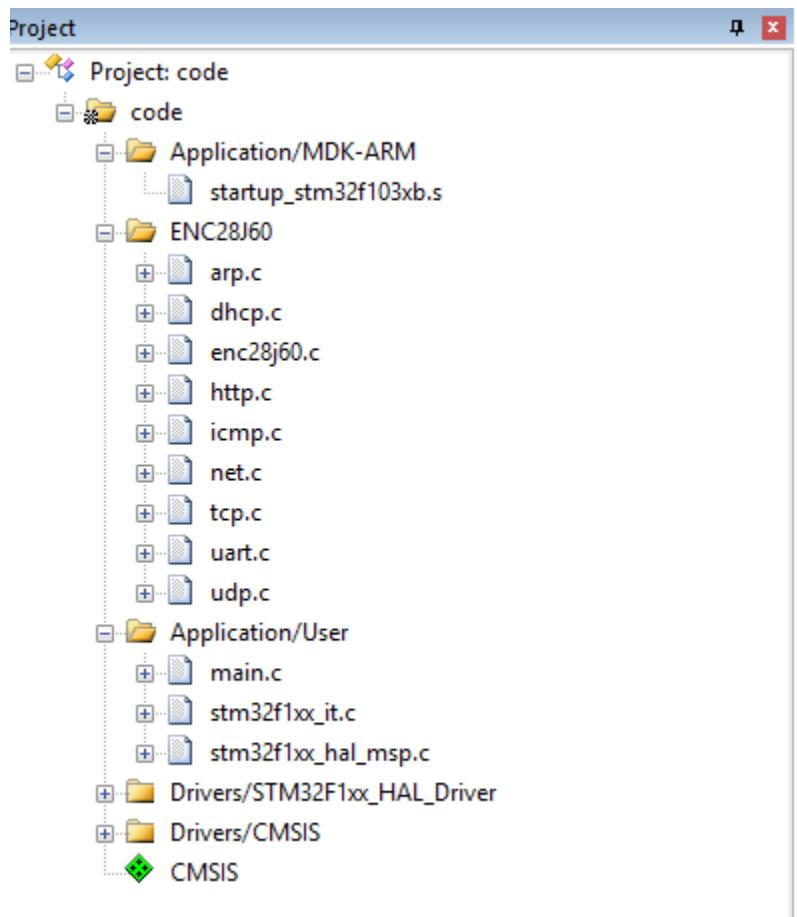


Để dễ quản lý, chúng ta sẽ đưa tất cả các file thư viện .c và trong cây Project và nhóm riêng vào trong group tên là ENC28J60 nhé

Các bạn click chuột phải vào project trong cây Project và chọn Add group, group New Folder được tạo ,click và đổi tên group thành tên gì các bạn muốn, ở đây mình sẽ đổi tên là ENC28J60



Click đúp vào group vừa tạo và tiến hành add tất cả các file .c vào cây quản lí project



Bây giờ add các thư viện cần thiết vào file main và biên dịch thử nhé



```
1#include "enc28j60.h"
2#include "net.h"
3#include "uart.h"
```

```

17 | ****
18 | */
19 | /* USER CODE END Header */
20 |
21 | /* Includes -----*/
22 | #include "main.h"
23 |
24 | /* Private includes -----*/
25 | /* USER CODE BEGIN Includes */
26 | #include "enc28j60.h"
27 | #include "net.h"
28 | #include "uart.h"
29 | /* USER CODE END Includes */
30 |
31 | /* Private typedef -----*/
32 | /* USER CODE BEGIN PTD */
33 |
34 | /* USER CODE END PTD */
35 |
36 | /* Private define -----*/
37 | /* USER CODE BEGIN PD */
38 | /* USER CODE END PD */

```

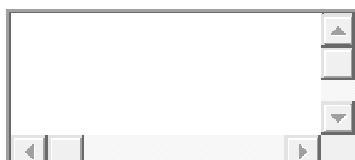
Build Output

```

Build started: Project: code
*** Using Compiler 'V5.06 update 6 (build 750)', folder
Build target 'code'
"code\code.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:00
Load "code\code.axf"
Erase Done.
Programming Done.
Verify OK.
Application running ...
Flash Load finished at 21:45:57

```

Thêm chương trình ngắt timer và ngắt ngoài



```

1 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
2 {
3     if(htim->Instance == TIM2)
4     {
5         ARP_clear_table();
6     }

```

```

7 }
8 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
9 {
10 if(GPIO_Pin == GPIO_PIN_0 && net_readEnable())
11 {
12     NET_loop();
13 }
14}

/* USER CODE BEGIN 0 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Instance == TIM2)
    {
        ARP_clear_table();
    }
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if(GPIO_Pin == GPIO_PIN_0 && net_readEnable())
    {
        NET_loop();
    }
}

```

Và thêm hàm khởi tạo vào đầu chương trình (trong hàm main, trước while(1))



```

1UART_init(&uart1);
2ENC29J600_ini(&hspi1);
3HAL_TIM_Base_Start_IT(&htim2);

```

```

99  HAL_Init();
100 /* USER CODE BEGIN Init */
101
102 /* USER CODE END Init */
103
104 /* Configure the system clock */
105 SystemClock_Config();
106
107 /* USER CODE BEGIN SysInit */
108
109 /* USER CODE END SysInit */
110
111 /* Initialize all configured peripherals */
112 MX_GPIO_Init();
113 MX_SPI1_Init();
114 MX_USART1_UART_Init();
115 MX_TIM2_Init();
116
117 /* USER CODE BEGIN 2 */
118 UART_init(&huart1);
119 ENC29J600_ini(&hspil1);
120 HAL_TIM_Base_Start_IT(&htim2);
121 /* USER CODE END 2 */
122

```

Các bạn có thể test lại các giao thức xem thư viện hoạt động ok chưa nhé, bây giờ sẽ đi vào phần chính là viết thư viện dhcp (trong file thư viện mình để đã có sẵn thư viện dhcp rồi đó)

Thư viện DCHP

File dhcp.h



```

1 #ifndef DHCP_H_
2 #define DHCP_H_
3 //-----
4 //Include cac thu vien can thiet
5 #include "stm32f1xx_hal.h"
6 #include <string.h>
7 #include <stdlib.h>
8 #include <stdint.h>
9 #include <stdio.h>

```

```
10#include "udp.h"
11#include "uart.h"
12//-----
13#define Port_src 68
14#define Port_dst 67
15#define Boardcast {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF}
16#define ip_src {0,0,0,0}
17#define ip_dst {0xFF,0xFF,0xFF,0xFF}
18//-----
19typedef struct
20 {
21     uint8_t op; //1
22     uint8_t htype; //1
23     uint8_t hlen; //6
24     uint8_t hops; //0
25     uint32_t id;
26     uint16_t seconds;
27     uint16_t flags;
28     uint8_t client_ip[4];
29     uint8_t your_id[4];
30     uint8_t server_id[4];
31     uint8_t gateway_id[4];
32     uint8_t client_mac[6];
33     uint8_t client_padding[10];
34     uint8_t server_name[64];
35     uint8_t boot_file_name[128];
36     uint32_t magic_cookie;
37
38 //DHCP OPTION
39     uint16_t option53; //0x3501 // Option: (53) DHCP Message Type (Discover)
40     uint8_t option53_dhcp; //0x01 ;//discover
41
42     uint16_t option61; //0x3D07 //Option: (61) Client identifier
```

```

43  uint8_t option61_hard_type; // Hardware type: Ethernet (0x01)
44  uint8_t option61_mac[6]; //Client MAC address
45
46  uint16_t option12; //0x0C05 Option: (12) Host Name (IOT47)
47  uint8_t option12_hostname[5];
48
49  uint16_t option55;//0x3705 //Option: (55) Parameter Request List
50  uint8_t option55_subnet_mask; //Parameter Request List Item: (1) Subnet Mask
51  uint8_t option55_router; //Parameter Request List Item: (3) Router
52  uint8_t option55_domain_name_sv;//Parameter Request List Item: (6) Domain Name Server
53  uint8_t optionEND;//0xFF
54 }__attribute__((packed)) DHCP_discover_segment;
55
56 typedef struct
57 {
58     uint8_t opcode;
59     uint8_t dump1[15];
60     uint8_t *option;
61 }__attribute__((packed)) DHCP_return_segment;
62 //-----
63 uint8_t DHCP_get_ip(uint32_t timeout);
64 void dhcp_discover(void);
65 void dhcp_getGateway(uint8_t *dhcp_mess,uint8_t *gate_mac,uint8_t *gate_ip);
66 //-----
67
68#endif /* ARP_H_ */

```

File dhcp.c



```
1 #include "dhcp.h"
```

```
2
3 extern const uint8_t macaddr[6];
4 extern uint8_t ip[4];
5 extern char debug_string[60];
6 extern uint8_t gateway_ip[4];
7 extern uint8_t gateway_mac[6];
8
9 uint8_t boar_cast[6]=Boardcast;
10 uint8_t ip_nguong[4]=ip_src;
11 uint8_t ip_dich[ 4]=ip_dst;
12
13
14 uint8_t getip_ok=0;
15 uint8_t DHCP_get_ip(uint32_t timeout)
16 {
17     uint32_t t = HAL_GetTick();
18     while(1)
19     {
20         if( (HAL_GetTick() - t) > timeout)
21             return 0; //time out
22         dhcp_discover();
23         HAL_Delay(1000);
24         if(getip_ok) return 1;
25     }
26 }
27 void dhcp_discover(void)
28 {
29     uint16_t dhcp_discover_length,i;
30
31     DHCP_Discover_Segment discover;
32
33     discover.op = 1;
34     discover.htype=1;
35     discover.hlen=6;
```

```
2 discover.hops=0;
6 discover.id = swap32((uint32_t)0x20071999);
2 discover.seconds = 0;
2 discover.flags=0;
8
2 memcpy(discover.client_ip,ip_nguon,4);
9 memcpy(discover.your_id,ip_nguon,4);
3 memcpy(discover.server_id,ip_nguon,4);
3 memcpy(discover.gateway_id,ip_nguon,4);
1 memcpy(discover.client_mac,macaddr,6);
3 for(i=0;i<10;i++)discover.client_padding[i]=0x00;
3 for(i=0;i<64;i++)discover.server_name[i]=0x00;
3 for(i=0;i<128;i++)discover.boot_file_name[i]=0x00;
3 discover.magic_cookie=swap32((uint32_t)0x63825363);
4
3
5 discover.option53= 0x0135;
3 discover.option53_dhcp=0x01;
6 discover.option61=0x073D;
3
7 discover.option61_hard_type=0x01;
3 memcpy(discover.option61_mac,macaddr,6);
8
3
9 discover.option12 = 0x050C;
4 discover.option12_hostname[0]= 'T';
0 discover.option12_hostname[1]= 'O';
4 discover.option12_hostname[2]= 'T';
1 discover.option12_hostname[3]= '4';
4 discover.option12_hostname[4]= '7';
2
3
4 discover.option55=0x0337;
4 discover.option55_subnet_mask=1;
4 discover.option55_router=3;
5 discover.option55_domain_name_sv=6;
4 discover.optionEND=0xFF; //end
6
```

```

4
7    dhcp_discover_length=sizeof(discover);
4
8    UDP_send_with_mac(boar_cast,ip_dich,Port_dst,(uint8_t *)macaddr,ip_ngouon,Port_src,(uint8_t_t
*)&discover,dhcp_discover_length);
4
9 }

5 void dhcp_getGateway(uint8_t *dhcp_mess,uint8_t *gate_mac,uint8_t *gate_ip)
0 {
5     DHCP_return_segment *DHCP_return = (DHCP_return_segment *)dhcp_mess;
1

5
2     if(getip_ok == 0)
3     {
5         getip_ok=1;
5
4         memcpy(ip,DHCP_return->offfer_ip,4);
5
5         memcpy(gateway_ip,gate_ip,4);
5
5         memcpy(gateway_mac,gate_mac,6);
5
6
5         sprintf(debug_string,"Your IP:%i.%i.%i.%i\r\n",ip[0],ip[1],ip[2],ip[3]);
7         UART.putString(debug_string);
5
5         sprintf(debug_string,"GateWay
8 IP:%i.%i.%i.%i\r\n",gateway_ip[0],gateway_ip[1],gateway_ip[2],gateway_ip[3]);
5
5         UART.putString(debug_string);
9
5         sprintf(debug_string,"GateWay
6 MAC:%02X:%02X:%02X:%02X:%02X\r\n",gateway_mac[0],gateway_mac[1],gateway_mac[2],gateway_mac[3],gate
0 way_mac[4],gateway_mac[5]);
5
6         UART.putString(debug_string);
1
5         dhcp_discover();
6
2     }
6
5
6
6
6
7

```

6 //-----END FILE-----//

8

6

9

7

0

7

1

7

2

7

3

7

4

7

5

7

6

7

7

7

8

7

9

8

0

8

1

8

2

8

3

8

4

8

5

8

6

8

7

8

8

```
8  
9  
9  
0  
9  
1  
9  
2  
9  
3  
9  
4  
9  
5  
9  
6  
9  
7  
9  
8  
9  
9
```

Trong hàm main gọi hàm **DHCP_get_ip** với tham số timeout là 10000, tức 10 giây



```
1UART_putString("Get IP ...\\r\\n");  
2if(!DHCP_get_ip(10000))UART_putString("DHCP error !\\r\\n");//get dynamic ip, timeout 10s
```

Hàm **DHCP_get_ip** trả về 0 tức là timeout, 1 là OK và sẽ in ra màn hình Serial ip mà ENC28J60 được mode wifi phát

```

HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

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

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

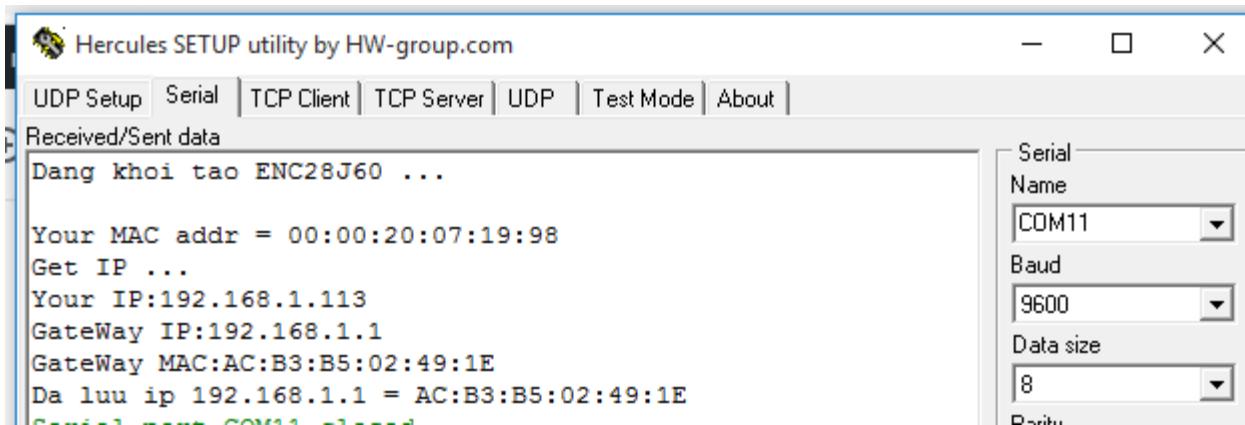
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_SPI1_Init();
MX_USART1_UART_Init();
MX_TIM2_Init();
/* USER CODE BEGIN 2 */
UART_init(&huart1);
ENC29J600_ini(&hsp1);
HAL_TIM_Base_Start_IT(&htim2);

UART.putString("Get IP ...\\r\\n");
if(!DHCP_get_ip(10000))UART.putString("DHCP error !\\r\\n");//get dynamic ip, timeout 10s

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{

```

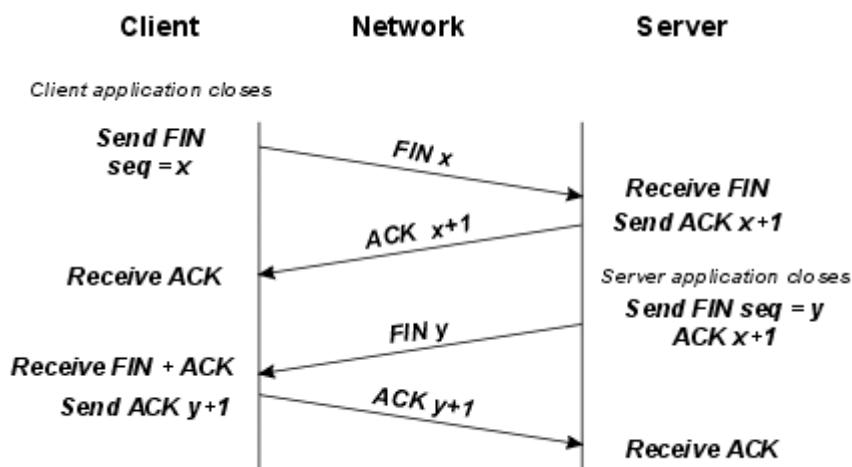


Chúng ta đã lấy được IP cá nhân, IP của Gateway và Mac của Gate, đây sẽ là các giá trị quan trọng được sử dụng cho các giao thức tiếp theo

Tải full project với cubemx và keilc [tại đây](#)

[ENC28J60] Bài 16: Giao thức TCP Client – Kết nối tới 1 server trong LAN

18 Tháng Năm, 2020 Đào Nguyễn Ethernet, IoT tutorial 3



Chúng ta đã làm quen với giao thức TCP Client chế độ Server, với TCP Client, mọi thứ khó hơn rất nhiều, cũng giống như các bài trước, mình sẽ làm với từng phần nhỏ một. Ở bài này chúng ta sẽ thử cho ENC28J60 kết nối với một TCP trong mạng LAN nhé !

Chú ý: Mình đã chuyển project sang chip stm32f103c8t6, các bạn phải đọc **bài 15** trước nhé

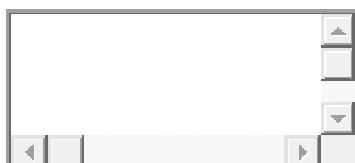
<http://iot47.com/enc28j60-bai-15-giao-thuc-dhcp-lay-ip-dong/>

Các bước thiết lập kết nối

- Client gửi bản tin SYN với 1 Sequence_Number bất kì
- Server sẽ trả về bản tin SYN|ACK với Acknowledgement = Sequence_Number +1
- Client trả lại với bản tin ACK

Hoàn tất kết nối. Sau khi kết nối thành công, chúng ta sẽ không cho Client kết nối lại nữa nhé vì điều này là không cần thiết

Ở trong file `tcp.h` chúng ta định nghĩa thêm 1 struct để lưu các thông tin kết nối với Server



```

2 {
3     uint32_t Sequence_Number;
4     uint32_t Acknowledgement;
5     uint8_t tcp_status;
6     uint16_t port;
7     uint16_t client_port;
8     uint8_t mac_defaul[6];
9     uint8_t ip_defaul[4];
10} __attribute__ ((packed)) TCP_client;

```

Khởi tạo Struct TCP_client vào đầu file tcp.c nhé



```
1TCP_client tcp_client1;
```

Biến tcp_status

tcp_status = 1 => đã gửi gói tin SYN, đang chờ SYN|ACK
 tcp_status = 2 => đã nhận được SYN|ACK, đã phản hồi ACK, kết nối thành công

Trong hàm **TCP_make_header** chúng ta thêm đoạn code make gói tin connect, gói tin connect sẽ mặc định là có 66 byte data



```

1 memcpy(TCP_Struct_Frame->MAC_dich,tcp_client1.mac_defaul,6);
2 memcpy(TCP_Struct_Frame->MAC_nguon,macaddr,6);
3 TCP_Struct_Frame->Ethernet_type = 0x0008;
4
5 TCP_Struct_Frame->Header_length = 0x45;      //make IP
6 TCP_Struct_Frame->Services=0x0000;
7 TCP_Struct_Frame->TotoLength = swap16(52); //do dai cua goi tin IP mac dinh la 0x0034 (52 byte) (66-14)
8 TCP_Struct_Frame->Identification=0xBF2B;

```

```

9   TCP_Struct_Frame->Flag=0x0040;
10  TCP_Struct_Frame->TimeToLive=0x80;
11  TCP_Struct_Frame->Protocol=0x06; //tcp
12  TCP_Struct_Frame->CheckSum=0x0000;
13  memcpy(TCP_Struct_Frame->SourceIP,ip,4);
14
15          tcp_client1.client_port = 3456;
16  TCP_Struct_Frame->Source_Port = swap16(tcp_client1.client_port);
17          TCP_Struct_Frame->Sequence_Number=swap32(tcp_client1.Sequence_Number);
18  TCP_Struct_Frame->Acknowledgement=0;
19  TCP_Struct_Frame->data_offset=0x80;
20  TCP_Struct_Frame->TCP_Flags = TCP_SYN;
21  TCP_Struct_Frame->Window = swap16(8192); // thong bao cho server biet bo dem nhan toi da
22  TCP_Struct_Frame->TCP_Checksums=0x0000;
23  TCP_Struct_Frame->Urgent_Pointer=0x0000;
24  //option
25  TCP_Struct_Frame->data[0]=0x02;
26  TCP_Struct_Frame->data[1]=0x04;
27  TCP_Struct_Frame->data[2]=0x05;
28  TCP_Struct_Frame->data[3]=0xb4;
29  TCP_Struct_Frame->data[4]=0x01;
30  TCP_Struct_Frame->data[5]=0x03;
31  TCP_Struct_Frame->data[6]=0x03;
32  TCP_Struct_Frame->data[7]=0x08;
33  TCP_Struct_Frame->data[8]=0x01;
34  TCP_Struct_Frame->data[9]=0x01;
35  TCP_Struct_Frame->data[10]=0x04;
36  TCP_Struct_Frame->data[11]=0x02;

```

Và định nghĩa **FOR_CONNECT** có giá trị là 5 vào file **tcp.h**



```
1#define FOR_ACK      5

#define FOR_SYN        0
#define FOR_FIN        1
#define FOR_PSH_ACK   2
#define FOR_DISCONNECT 3
#define FOR_CONNECT    4
#define FOR_ACK        5
```

Viết hàm gửi gói tin SYN



```
1 uint8_t TCP_sendSYN(uint8_t *IP_server,uint16_t port,uint32_t timeout)
2 {
3     TCP_struct *TCP_Struct_Frame = (TCP_struct *)eth_buffer;
4     uint32_t count=0;
5
6     if(IP_server[0] == 192 && IP_server[1] == 168) //local ip
7     {
8         while(1)
9         {
10            if(ARP_table_checkIP(IP_server) != -1)
11            {
12                ARP_table_get_MAC(IP_server,tcp_client1.mac_default);
13                break; // da nhan dc MAC
14            }
15            ARP_send_request(IP_server);
16            HAL_Delay(50);
17            count+=50;
18            if(count >= timeout)
19            {
20                return 0; //gui that bai
21            }
22        }
23    }
```

```

24         tcp_client1.tcp_status = 1;
25
26         tcp_client1.port = port;
27
28         memcpy(tcp_client1.ip_defaul,IP_server,4); //save
29
30 //chinh sua thong tin nguoi nhan
31 TCP_make_header(TCP_Struct_Frame,66,FOR_CONNECT); //ban tin connect co do dai mac dinh la 66
32 memcpy(TCP_Struct_Frame->DestIP,IP_server,4);
33 TCP_Struct_Frame->Dest_Port = swap16(port);
34
35         TCP_Struct_Frame->CheckSum = NET_ipchecksum((uint8_t *)&TCP_Struct_Frame->Header_length); //tinh checksum cho goi IP
36
37         TCP_Struct_Frame->TCP_Checksums = TCP_checksum(TCP_Struct_Frame);
38
39         sprintf(debug_string,"Connect to %i.%i.%i.%i:%i ... \r\n",IP_server[0],IP_server[1],IP_server[2],IP_server[3],port);
40         UART.putString(debug_string);
41         NET_SendFrame((uint8_t *)TCP_Struct_Frame,66);
42
43         net_readSetStatus(1);
44
45         return 1;
46     }

```

Để gửi gói tin connect vào 1 thiết bị trong mạng LAN, cần phải có địa chỉ MAC, mình sử dụng hàm [ARP_table_get_MAC](#) để lấy

Hàm này trả về 0 nếu không thể lấy MAC thành công, và trả về 1 nếu lấy thành công, đồng thời nó cũng lưu địa chỉ MAC đã lấy được vào struct [tcp_client1](#) luôn

Sau khi gửi xong bản tin SYN, mình sẽ phải chờ bản tin SYN|ACK của server trả về, do phải chờ nên ta sẽ cần 1 cái timeout, nếu quá hạn thì thử gửi lại xem sao.

Hàm [TCP_Connect](#) sẽ làm nhiệm vụ trên



```

1 uint8_t TCP_Connect(uint8_t *IP_server,uint16_t port,uint32_t timeout,int8_t Try_reconnect)
2 {
3     uint32_t t;

```

```

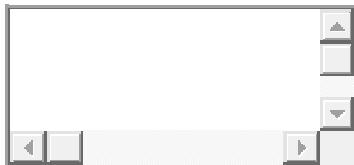
4     if( tcp_client1.tcp_status==2)
5     {
6         UART_putString("Da ket noi roi !\r\n");
7         return 2;
8     }
9     tcp_client1.Sequence_Number = rand(); //save Sequence_Number
10    do
11    {
12        if(!TCP_sendSYN(IP_server,80,5000)) // gửi bản tin SYN
13        {
14            UART_putString("Ket noi that bai 1 \r\n");
15            return 0; //gui that bai
16        }
17        else
18        {
19            t = HAL_GetTick();
20            while(1)
21            {
22                if(tcp_client1.tcp_status == 2)
23                {
24                    UART_putString("Ket noi thanh cong\r\n");
25                    return 1;
26                }
27                if(HAL_GetTick() - t > timeout)
28                {
29                    break;
30                }
31            }while(Try_reconnect--);
32            UART_putString("Ket noi that bai 2\r\n");
33            return 0; //gui that bai
34}

```

Ở đây, mình thêm vào tham số `Try_reconnect`, tức là số lần cố gắng thử kết nối lại nếu kết nối thất bại. Hàm này sẽ trả về 0 nếu kết nối hoàn toàn thất bại, 1 nếu kết nối thành công, 2 nếu đã kết nối rồi. Không kết nối nữa !

Nhận bản tin SYN|ACK

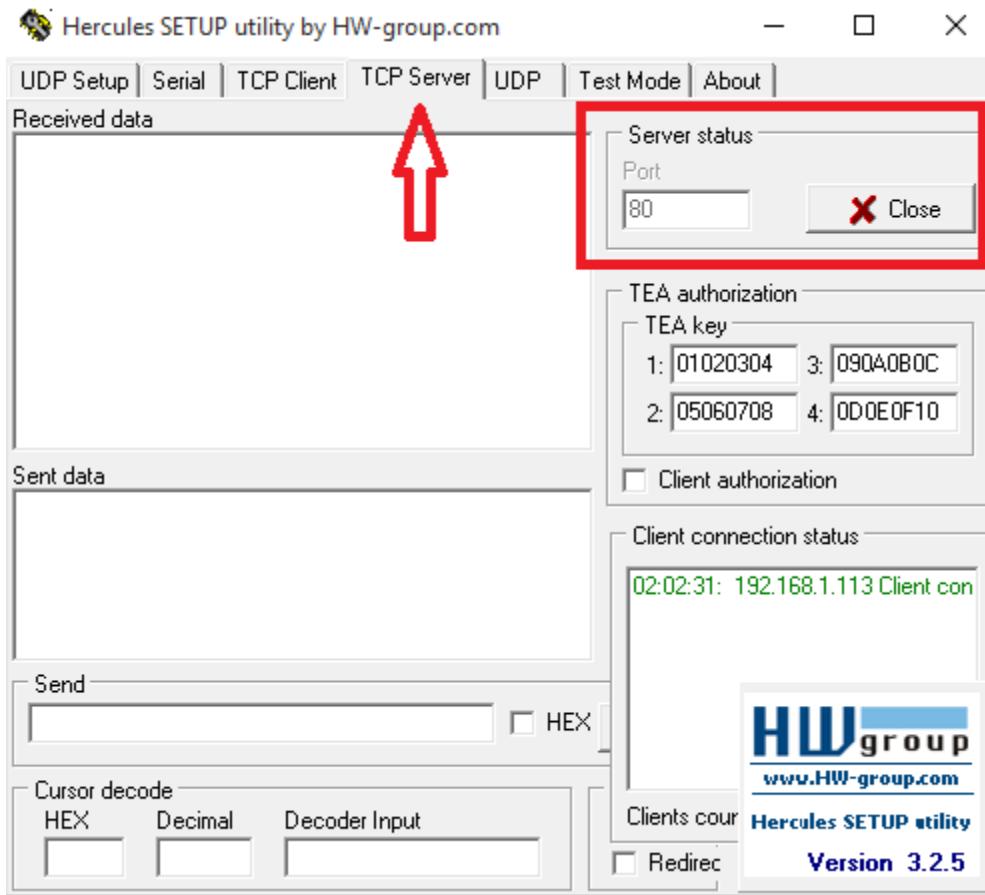
Trong hàm `tcp_read`, chúng ta thêm code kiểm tra cờ SYN|ACK



```
1     if(TCP_Struct_Frame->TCP_Flags == (TCP_SYN|TCP_ACK))
2     {
3         if(tcp_client1.tcp_status == 1 || tcp_client1.tcp_status == 2)
4         {
5             if(swap32(TCP_Struct_Frame->Acknowledgement) == tcp_client1.Sequence_Number+1)
6             {
7                 tcp_client1.Sequence_Number+=1;
8                 tcp_client1.Acknowledgement = swap32(TCP_Struct_Frame-
9 >Sequence_Number) + 1;
10                tcp_client1.tcp_status=2; //ket noi thanh cong
11                TCP_make_header(TCP_Struct_Frame,len,FOR_ACK);
12                NET_SendFrame((uint8_t *)TCP_Struct_Frame,len); //gui goi tin tcp ack reply
13                UART.putString("Connect OK\r\n");
14            }
15        }
```

OK. Bây giờ test thử thôi

Khởi tạo server TCP trên máy tính bằng phần mềm Hercules -> TCP Server. Chọn Port 80 -> Listen



Trong file code chính (main.c) mình sẽ add địa chỉ server của máy tính mình đang dùng, nếu các bạn không biết IP của máy tính thì vào cmd gõ ipconfig hoặc tra google với từ khóa “**cách kiểm tra ip máy tính**” hoặc tham khảo hướng dẫn [ở đây](#)

Ví dụ, laptop của mình có ip 192.168.1.18



```

8 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef)
9 {
10     if(htim->Instance == TIM2)
11     {
12         ARP_clear_table();
13     }
14 }
15 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
16 {
17     if(GPIO_Pin == GPIO_PIN_0 && net_readEnable())
18     {
19         NET_loop();
20     }
21 }
22 uint8_t ip_server[4]={192,168,1,18}; ←
23 /* USER CODE END 0 */
24

```

Quay trở lại hàm main và gọi **TCP_Connect(ip_server,80,5000,5);**

```

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_SPI1_Init();
MX_USART1_UART_Init();
MX_TIM2_Init();
/* USER CODE BEGIN 2 */
UART_init(&huart1);
ENC29J600_ini(&hspil);
HAL_TIM_Base_Start_IT(&htim2);

UART.putString("Get IP ...\\r\\n");
if(!DHCP_get_ip(10000))UART.putString("DHCP error !\\r\\n");//get dynamic ip, timeout 10s

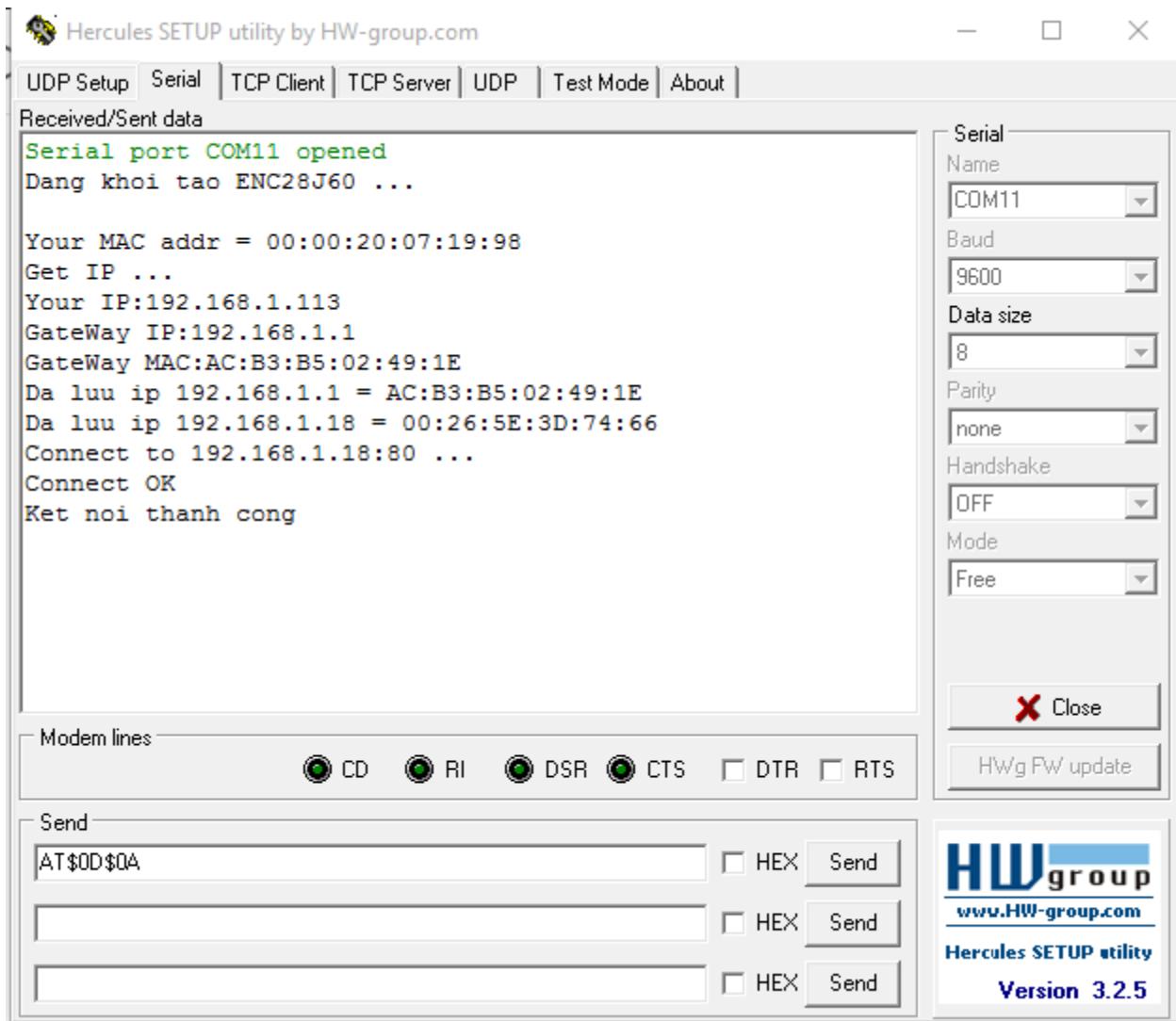
TCP_Connect(ip_server,80,5000,5); ←
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{

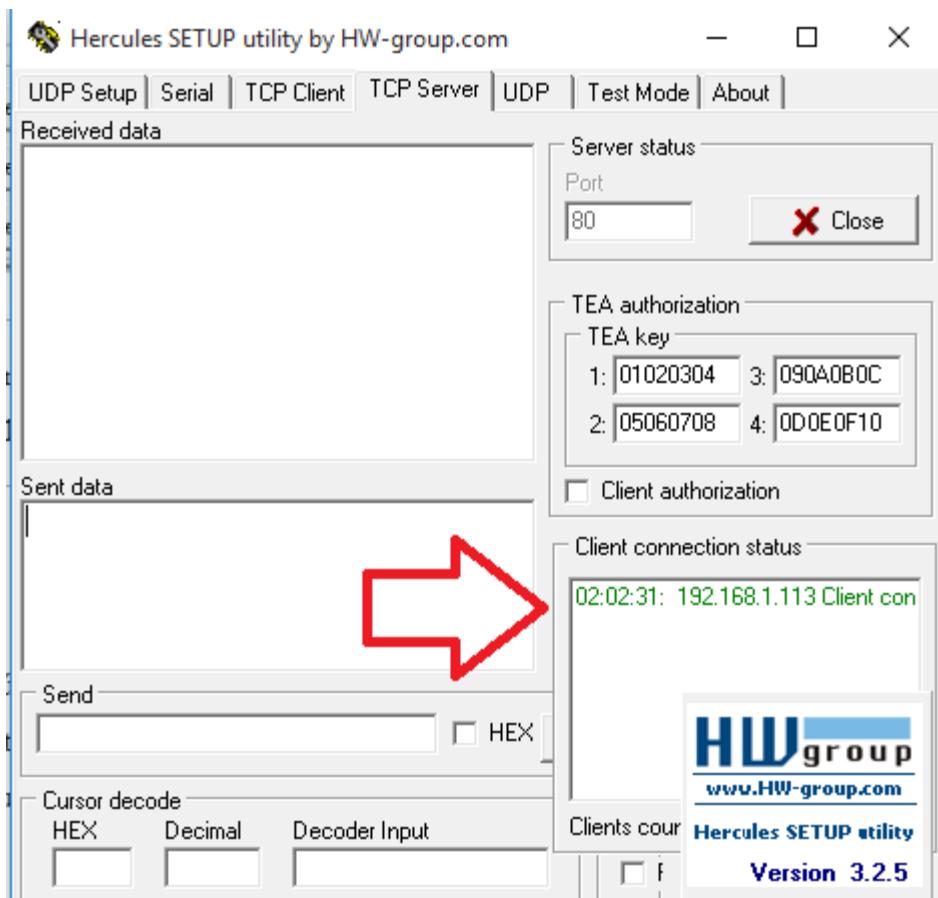
```

Ở đây, tham số 80 chính là port của server mà mình đã lắng nghe ở phần mềm Hercules, 5000 là timeout của mỗi lần connect và 5 là số lần cố gắng thử kết nối lại nếu thất bại

Chạy chương trình và xem màn hình debug



Như vậy mình đã kết nối thành công tới máy chủ do laptop của mình tạo ra. Và trong tab TCP_Server của phần mềm Hercules cũng sẽ có 1 thông báo có 1 client connect tới



Ngắt kết nối

Server có thể chủ động ngắt kết nối bằng bản tin FIN, do vậy chúng ta bắt sự kiện này ở bản tin FIN|ACK đồng thời kiểm tra **Acknowledgement** xem có trùng với **Sequence_Number** đã lưu ở trong Struct **tcp_client1** không ! Nếu đúng thì mình sẽ reset lại **tcp_status**

Trong hàm **TCP_read** tại chỗ kiểm tra cờ **TCP_FIN|TCP_ACK**, thêm



```

1 if(swap32(TCP_Struct_Frame->Acknowledgement) == tcp_client1.Sequence_Number)
2 {
3     tcp_client1.tcp_status=0;
4     UART_putString("Server da ngat ket noi !\r\n");
5 }
6 else if(swap32(TCP_Struct_Frame->Acknowledgement) == (tcp_client1.Sequence_Number+1))

```

```

7     {
8     tcp_client1.tcp_status=0;
9     UART_putString("Da ngat ket noi Server!\r\n");
10    }

```

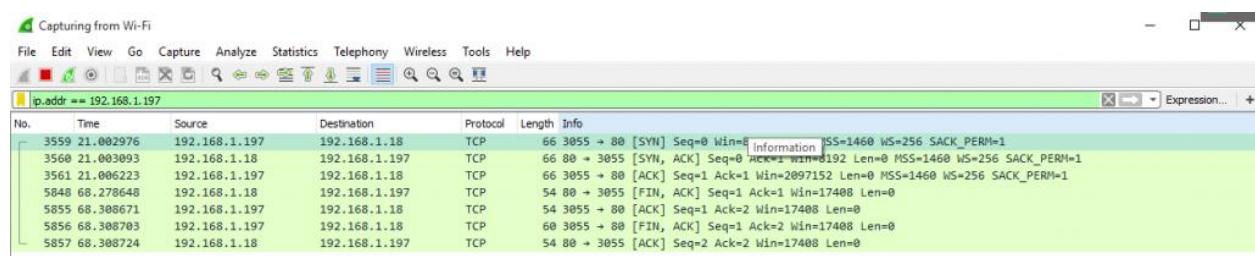
Sau khi kết nối thành công, hãy ấn nút Close trên tab TCP_Server trên tab Hercules, điều này sẽ khiến phần mềm tự động ngắt kết nối với tất cả client

```

Your MAC addr = 00:00:20:07:19:98
Get IP ...
Your IP:192.168.1.113
GateWay IP:192.168.1.1
GateWay MAC:AC:B3:B5:02:49:1E
Da luu ip 192.168.1.1 = AC:B3:B5:02:49:1E
Da luu ip 192.168.1.18 = 00:26:5E:3D:74:66
Connect to 192.168.1.18:80 ...
Connect to 192.168.1.18:80 ...
Connect OK
Ket noi thanh cong
Da ngat ket noi voi server !

```

Và đây là toàn bộ quá trình kết nối và ngắt được Wireshark ghi lại



Client chủ động ngắt kết nối

Client sẽ chủ động gửi FIN|ACK để ngắt kết nối tới Server

Thêm phần make header cho gói disconnect trong hàm **TCP_make_header**



```

1     else if(type == FOR_DISCONNECT)
2 {
3     memcpy(TCP_Struct_Frame->MAC_dich,tcp_client1.mac_defaul,6);

```

```

4 memcpy(TCP_Struct_Frame->MAC_nguon,macaddr,6);
5 TCP_Struct_Frame->Ethernet_type = 0x0008;
6
7 TCP_Struct_Frame->Header_length = 0x45; //make IP
8 TCP_Struct_Frame->Services=0x0000;
9 TCP_Struct_Frame->TotoLength = swap16(40); //do dai cua goi tin IP mac dinh la 0x0028 (40 byte) ( 54 -14)
10 TCP_Struct_Frame->Identification=0x4470;
11 TCP_Struct_Frame->Flag=0x0040;
12 TCP_Struct_Frame->TimeToLive=0x80;
13 TCP_Struct_Frame->Protocol=0x06; //tcp
14 TCP_Struct_Frame->CheckSum=0x0000;
15 memcpy(TCP_Struct_Frame->SourceIP,ip,4);
16
17 TCP_Struct_Frame->Source_Port = swap16(tcp_client1.client_port);
18 TCP_Struct_Frame->Sequence_Number=swap32(tcp_client1.Sequence_Number);
19 TCP_Struct_Frame->Acknowledgement=swap32(tcp_client1.Acknowledgement);
20 TCP_Struct_Frame->data_offset=0x50;
21 TCP_Struct_Frame->TCP_Flags = TCP_FIN|TCP_ACK;
22 TCP_Struct_Frame->Window = 0x4400; // thong bao cho server biet bo dem nhan toi da
23 TCP_Struct_Frame->TCP_Checksums=0x0000;
24 TCP_Struct_Frame->Urgent_Pointer=0x0000;
25 }

```

Và tạo thêm hàm **TCP_Disconnect**



```

1 uint8_t TCP_Disconnect(uint32_t timeout)
2 {
3     uint32_t t;
4     if( tcp_client1.tcp_status==2)
5     {

```

```

6         TCP_struct *TCP_Struct_Frame = (TCP_struct *)eth_buffer;
7         net_readSetStatus(0);
8 //chinh sua thong tin nguoi nhan
9         memcpy(TCP_Struct_Frame->DestIP,tcp_client1.ip_defaul,4);
10        TCP_Struct_Frame->Dest_Port = swap16(tcp_client1.port);
11        TCP_make_header(TCP_Struct_Frame,54,FOR_DISCONNECT); //ban tin disconnect co do dai mac dinh la 54
12        TCP_Struct_Frame->CheckSum = NET_ipchecksum((uint8_t *)&TCP_Struct_Frame->Header_length); //tinh checksum
13 //cho goi IP
14        TCP_Struct_Frame->TCP_Checksums = TCP_checksum(TCP_Struct_Frame);
15
16        NET_SendFrame((uint8_t *)TCP_Struct_Frame,54);
17        net_readSetStatus(1);
18        t = HAL_GetTick();
19        while(1)
20        {
21                if( tcp_client1.tcp_status == 0 ) //ngat ket noi thanh cong
22                {
23                        return 1;
24                }
25                if(HAL_GetTick() - t > timeout)
26                {
27                        UART.putString("Ngat ket noi that bai\r\n");
28                        return 0;
29                }
30        }
31        return 2; //chua co ket noi de ngat
32    }

```

Ở hàm main, sau khi kết nối thành công mình sẽ thử ngắt kết nối luôn để test hàm ngắt kết nối



```
1 if(TCP_Connect(ip_server,80,5000,1))
2 {
3     HAL_Delay(100);
4     TCP_Disconnect(5000); //ngắt kết nối với timeout là 5s
5 }
```

Your MAC addr = 00:00:20:07:19:98
Get IP ...
Your IP:192.168.1.113
GateWay IP:192.168.1.1
GateWay MAC:AC:B3:B5:02:49:1E
Da luu ip 192.168.1.1 = AC:B3:B5:02:49:1E
Da luu ip 192.168.1.18 = 00:26:5E:3D:74:66
Connect to 192.168.1.18:80 ...
Connect OK
Ket noi thanh cong
Da ngat ket noi Server!

Tải FULL Source [tại đây](#)

<https://drive.google.com/open?id=1LNJPPLRa7aYyXL2uvuLArp7aOn3KCLA>

Lưu ý: Thư viện enc28j60 nằm ở : |code|Drivers|IOT47_lib|ENC28J60

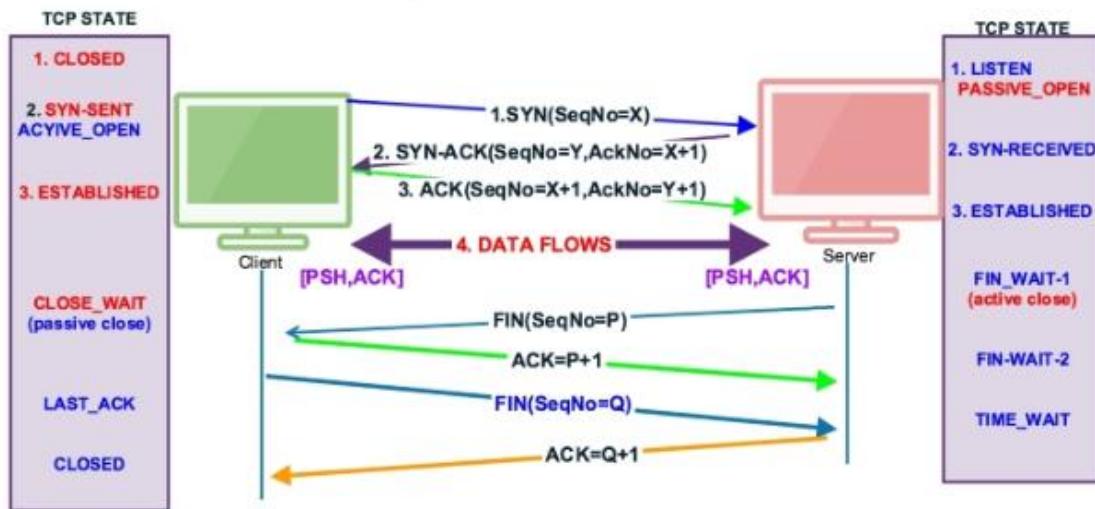
Theo dõi toàn bộ tutorial enc28j60 [tại đây](#)

Related posts:

[ENC28J60] Bài 17: Giao thức TCP Client – Gửi nhận dữ liệu

1 Tháng Sáu, 2020 Đào Nguyễn Ethernet, IoT tutorial 0

TCP Three Way Handshake/DATA FLOW/CLOSE SEQUENCE



Tiếp tục chuỗi tutorial lập trình giao tiếp ENC28J60 với giao thức TCP, trong bài này chúng ta sẽ hoàn thành các hàm gửi nhận data trong giao thức TCP Client. Về cơ bản, cũng tương tự như với TCP server thôi.

Nhận dữ liệu từ server



```
1 void TCP_readData(TCP_struct *TCP_Struct_Frame,uint16_t len)
2 {
3     uint16_t data_len,data_offset,i;
4     //kiểm tra số hiệu gói tin
5     if(swap32(TCP_Struct_Frame->Sequence_Number) == tcp_client1.Acknowledgement) //nếu chính xác thì nhận,
6         không thì bỏ qua vì có thể nó là gói tin thất lạc thôi
7     {
8         data_len= swap16(TCP_Struct_Frame->TotalLength) -20 - (TCP_Struct_Frame->data_offset >> 2); //tính độ dài chuỗi tin nhắn gửi tới
```

```

9     data_offset = (TCP_Struct_Frame->data_offset >> 2) - 20; //tính điểm bắt đầu của data
10
11     //in ra màn hình
12     for(i=0;i<data_len;i++)
13         UART_putChar(TCP_Struct_Frame->data[i+data_offset]);
14
15     TCP_make_header(TCP_Struct_Frame,len,FOR_PSH_ACK);
16
17     //lúc này client sẽ chủ động + độ dài data mà nó nhận dc vào Sequence_Number
18     tcp_client1.Sequence_Number = swap32(TCP_Struct_Frame->Sequence_Number);
19     tcp_client1.Acknowledgement = swap32(TCP_Struct_Frame->Acknowledgement);
20     TCP_send(TCP_Struct_Frame,54,(uint8_t *)"",0); //trả lời lại ack
21
22 }
}

```

Giải thích: Sau khi nhận được data từ server mình in ra màn hình, lưu lại các số hiệu gói tin và chả lời lại với ACK hoặc PSH|ACK cũng được

Server gửi dữ liệu sẽ dùng cờ PSH|ACK do vậy trong hàm TCP_read nhánh cờ **TCP_PSH|TCP_ACK** mình sẽ kiểm tra `tcp_client` status, nếu đã có kết nối với server thì gọi hàm **TCP_readData**

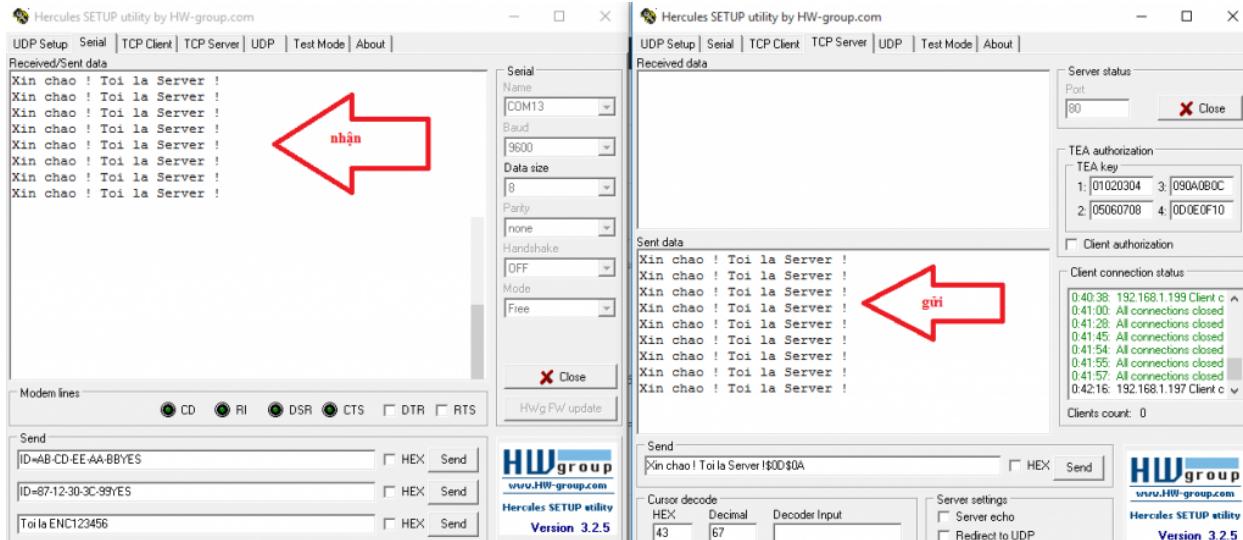


```

1 if(tcp_client1.tcp_status == 2) //neu đã kết nối với server
2 {
3     TCP_readData(TCP_Struct_Frame,len);
4 }

```

Bây giờ mở hercules mode `TCP_server`, mở thêm tab debug uart và gửi thử dữ liệu đi nào



Gửi dữ liệu lên server

Mình sẽ thêm nhánh maker header FOR_SEND cho hàm `TCP_make_header`



```

1     else if(type == FOR_SEND)
2     {
3         memcpy(TCP_Struct_Frame->MAC_dich,tcp_client1.mac_defaul,6);
4         memcpy(TCP_Struct_Frame->MAC_nguon,macaddr,6);
5         TCP_Struct_Frame->Ethernet_type = 0x0008;
6
7         memcpy(TCP_Struct_Frame->SourceIP,ip,4);
8         memcpy(TCP_Struct_Frame->DestIP,tcp_client1.ip_defaul,4);
9
10        TCP_Struct_Frame->Header_length = 0x45;      //make IP
11        TCP_Struct_Frame->Services=0x0000;
12        TCP_Struct_Frame->Identification=0x4470;
13        TCP_Struct_Frame->Flag=0x0040;
14        TCP_Struct_Frame->TimeToLive=0x80;
15        TCP_Struct_Frame->Protocol=0x06; //tcp
16        TCP_Struct_Frame->CheckSum=0x0000;

```

```

17     TCP_Struct_Frame->TotoLength = swap16(40);
18     memcpy(TCP_Struct_Frame->SourceIP,ip,4);
19
20     TCP_Struct_Frame->Source_Port = swap16(tcp_client1.client_port);
21     TCP_Struct_Frame->Dest_Port = swap16(tcp_client1.port);
22     TCP_Struct_Frame->Sequence_Number=swap32(tcp_client1.Sequence_Number);
23     TCP_Struct_Frame->Acknowledgement=swap32(tcp_client1.Acknowledgement);
24     TCP_Struct_Frame->data_offset=0x50;
25     TCP_Struct_Frame->Window = 0xCA01; // thong bao cho server biết bo dem nhan toi da
26     TCP_Struct_Frame->TCP_Checksums=0x0000;
27     TCP_Struct_Frame->Urgent_Pointer=0x0000;
28 }

```

Và viết hàm để sendData lên server

Hàm này sẽ nhận vào dữ liệu, độ dài của dữ liệu, và 1 timeout, sau khi gửi dữ liệu đi, server sẽ phản hồi lại cờ ACK, do vậy mình dùng vòng lặp vô tận while(1) để kiểm tra biến status đã nhận được cờ ACK chưa, nếu chưa và quá timeout thì thoát và báo gửi lỗi ! Ngoài ra, khi hàm này được gọi mà chưa kết nối server thì mình cũng thoát luôn mà không cần gửi vì điều đó vô nghĩa



```

1 uint8_t TCP_sendData(uint8_t *data,uint16_t data_length,uint32_t timeout)
2 {
3     uint32_t t;
4     TCP_struct *TCP_Struct_Frame = (TCP_struct *)eth_buffer;
5
6     if(tcp_client1.tcp_status != 2 )
7     {
8         UART.putString("Khong co ket noi\r\n");
9         return 2;
10    }
11
12    net_SetStatus(0);
13    TCP_make_header(TCP_Struct_Frame,0,FOR_SEND);

```

```

14     TCP_send(TCP_Struct_Frame,54,data,data_length);
15
16     t = HAL_GetTick();
17
18     tcp_client1.tcp_status = 3; //check ack
19
20     net_SetStatus(1);
21
22     while(1)
23
24         {
25
26             if(tcp_client1.tcp_status == 2)
27
28                 {
29
30                     UART_putString("Gui thanh cong\r\n");
31
32                     return 1;
33
34                 }
35
36             }
37
38         }
39
40     }
41
42
43 }
```

tcp_client1.status = 3 // đang cần check cờ ack

Và giờ trong hàm **TCP_read** nhánh bắt cờ ACK, mình sẽ thêm đoạn code kiểm tra xem cờ ACK này có phải là ACK của gói tin ta vừa gửi đi không

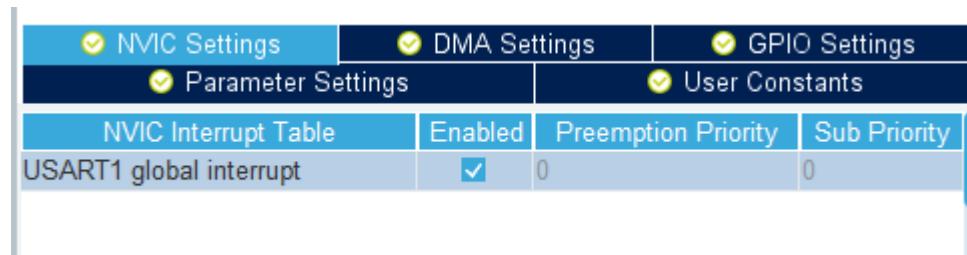


```

1 if(tcp_client1.tcp_status == 3)
2
3     {
4
5         tcp_client1.Sequence_Number = swap32(TCP_Struct_Frame->Acknowledgement);
6
7         tcp_client1.Acknowledgement = swap32(TCP_Struct_Frame->Sequence_Number);
8
9         tcp_client1.tcp_status = 2;
10
11    }
```

Thực chất, để đảm bảo chắc chắn cái ACK này là phản hồi của gói tin mình vừa gửi đi, ta phải kiểm tra kĩ hơn số hiệu gói tin nữa

Mình sẽ viết thêm 1 vài hàm nhận data từ UART để tiên cho việc gửi dữ liệu đi. Tức là mình sẽ gõ tin nhắn cần gửi vào Tab UART hercules, truyền xuống cho chip để chip gửi lên server.
Các bạn bật ngắt UART lên



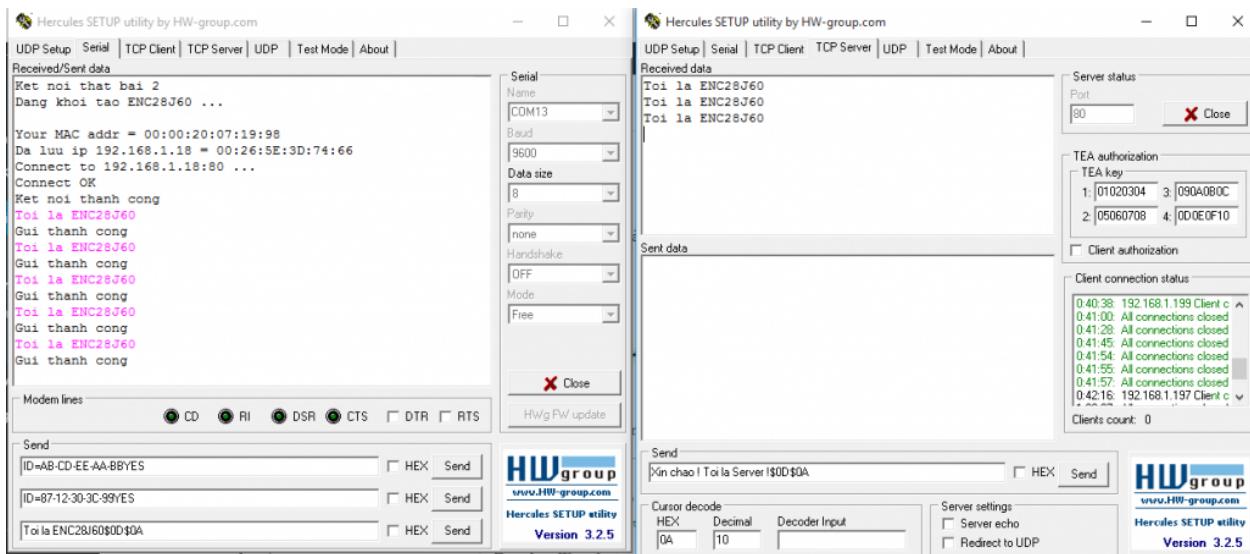
```
uint8_t RX_BUFF[99],RX_DATA,RX_index;
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    RX_BUFF[RX_index]=RX_DATA;
    RX_index++;if(RX_index>=99)RX_index=0;
    HAL_UART_Receive_IT(&huart1,(uint8_t *)&RX_DATA,1);
}
void clear_RX()
{
    for(int i=0;i<99;i++)
        RX_BUFF[i]=0;
    RX_index=0;
}
```

Và nhận từng byte data bằng ngắt UART, dừng quên gọi hàm cho phép nhận

```
/* USER CODE BEGIN 2 */
UART_init(&huart1);
ENC29J600_ini(&hspi1);
HAL_TIM_Base_Start_IT(&htim2);
HAL_UART_Receive_IT(&huart1,(uint8_t *)&RX_DATA,1);
```

Cuối cùng là trong hàm main, check data gửi tới để gọi hàm send lên server

```
if(RX_index!=0)
{
    HAL_Delay(50);//wait data
    TCP_sendData(RX_BUFF,strlen(RX_BUFF),5000); //gửi tin nhắn với timeout là 5s
    clear_RX();
}
```

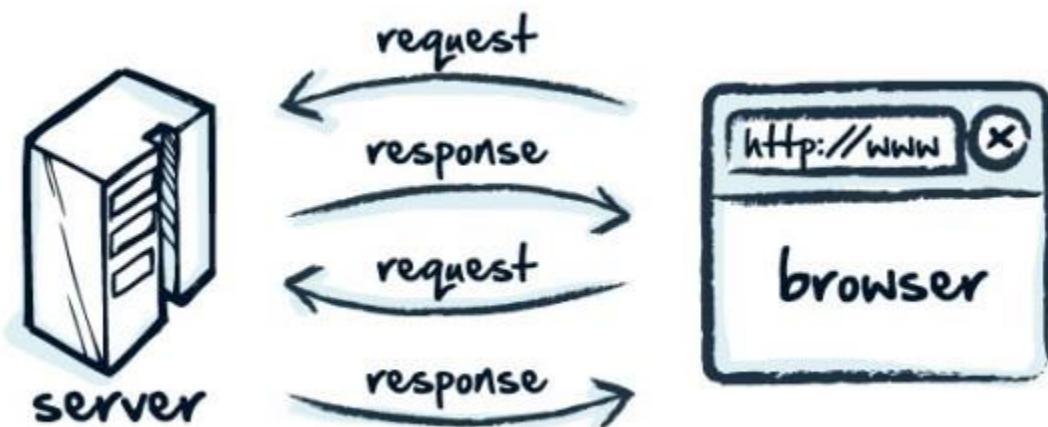


Tải FULL source cho bài này [tại đây](#)

<https://drive.google.com/file/d/1yDZA2OIVuso6ORLCL9TyTu-s6T-ojWJp/view?usp=sharing>

[ENC28J60] Bài 18: Giao thức HTTP Client, khám phá internet và gửi data lên thingspeak

1 Tháng Sáu, 2020 Đào Nguyễn Ethernet, IoT tutorial 2



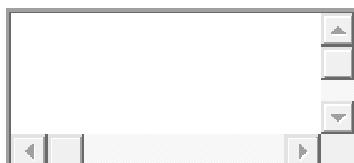
Chúng ta đã làm việc rất nhiều trong mạng LAN rồi, đã đến lúc đi ngoài và thăm thú thế giới internet đầy rẫy nguy hiểm ngoài kia

Trước tiên, hãy hoàn thiện lại giao thức TCP Client ở [bài 17](#) đã nhé, vì muốn giao tiếp với bên ngoài, chúng ta phải thông qua gateway, nhờ vào dhcp, chúng ta đã lấy được IP và MAC của gateway, mình sẽ extern 2 biến `gateway_ip` và `gateway_mac` và thư viện tcp



```
1extern uint8_t gateway_ip[4];  
2extern uint8_t gateway_mac[6];
```

Trong hàm TCP_sendSYN, thay vì phát bản tin ARP lấy MAC của các thiết bị trong LAN, chúng ta sẽ dùng MAC của gateway, do vậy hay thêm rẽ nhánh vào lệnh `if(IP_server[0] == 192 && IP_server[1] == 168):`



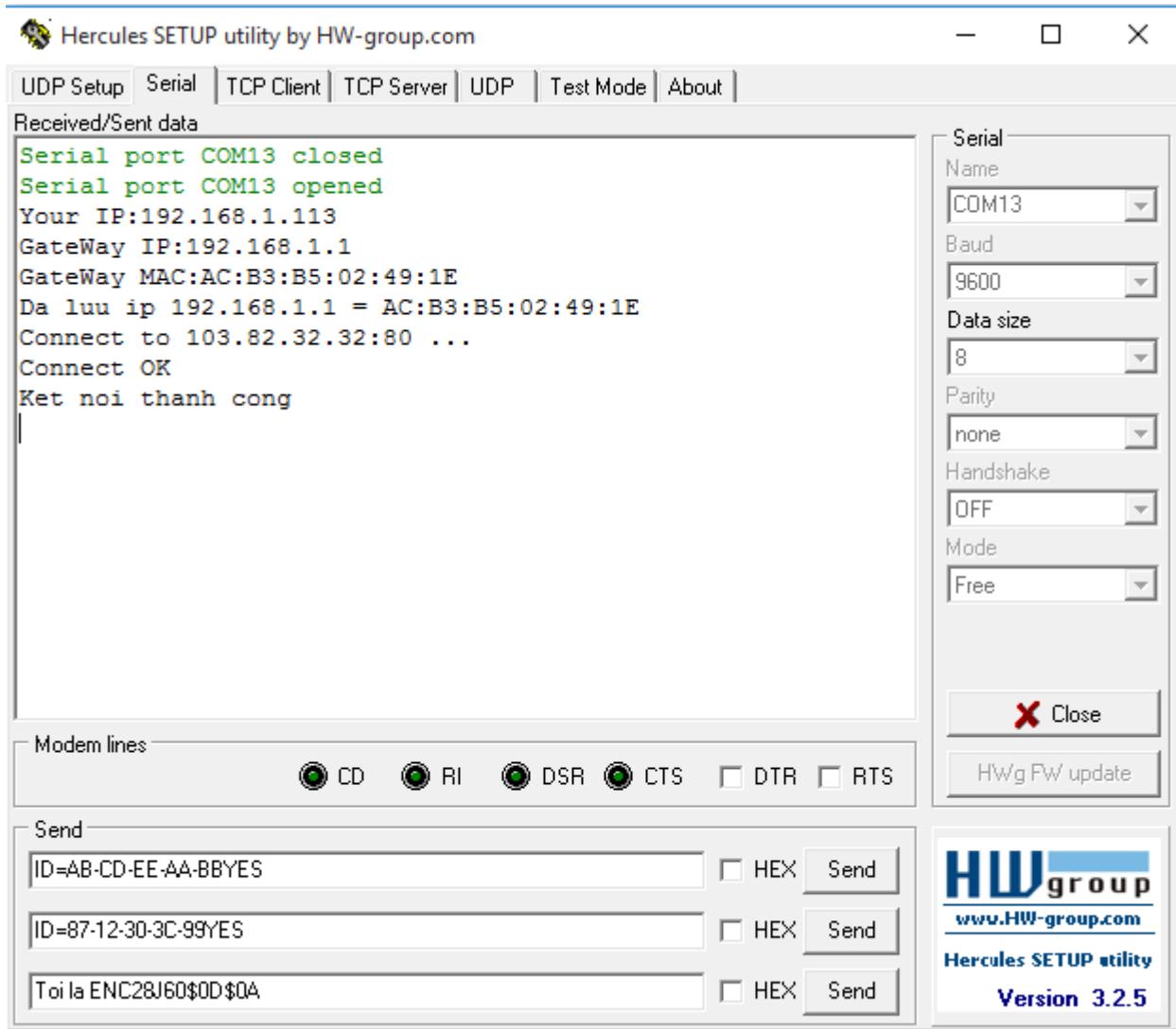
```
1 else
2 {
3     memcpy(tcp_client1.mac_defaul,gateway_mac,6);
4 }
```

Bây giờ qua lại main và sửa ip_server thành 103.82.32.32 nhé (chính là IP của website này đó)



```
1 uint8_t ip_server[4]={103,82,32,32};
```

Kết nối thử nào

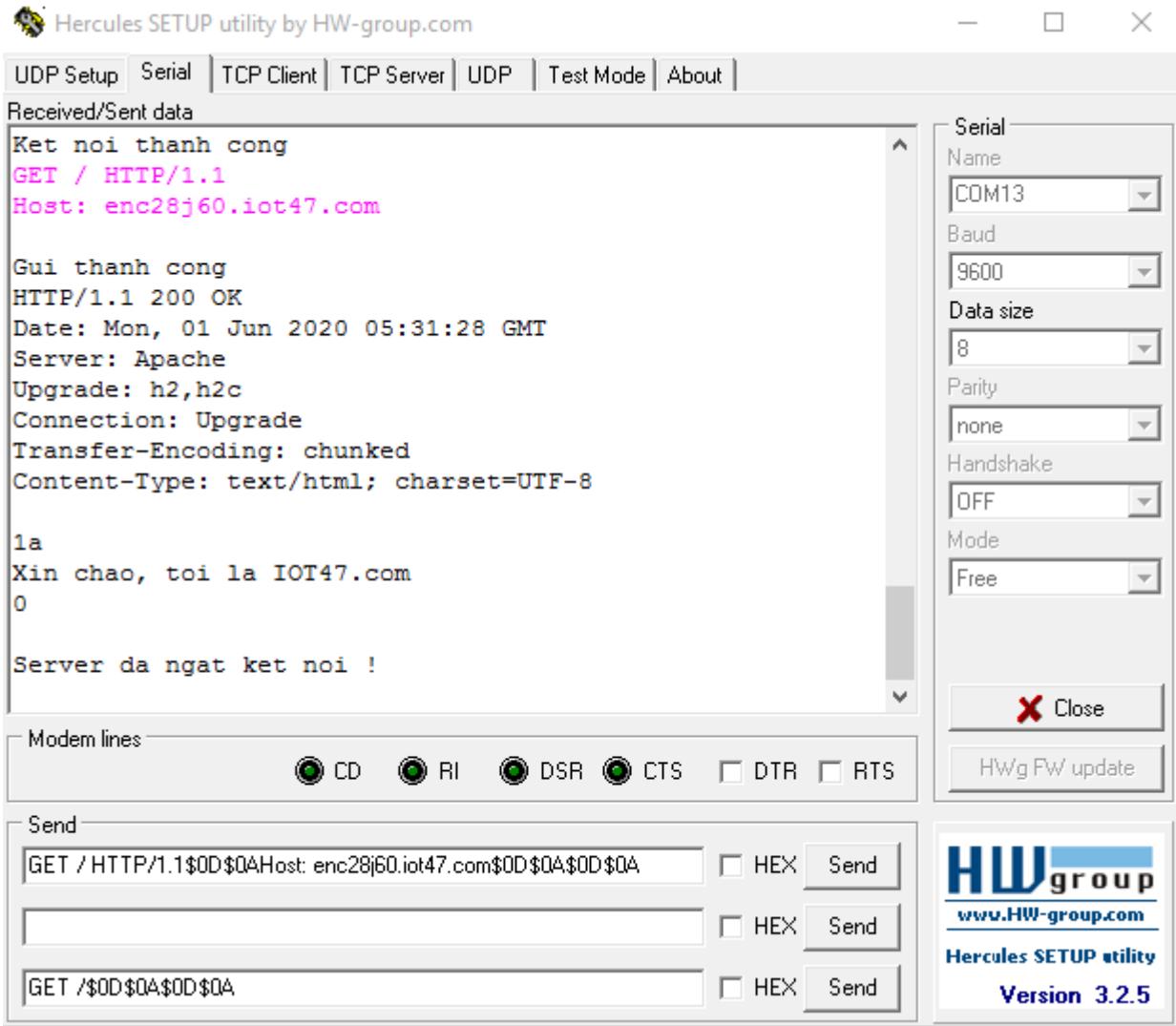


Tot vời, đây là lần đầu tiên chúng ta ra được internet

Mình đã tạo 1 trang web đơn giản có địa chỉ <http://enc28j60.iot47.com/>
Khi các bạn ấn vào sẽ thấy nội dung Xin chao,toi la IOT47.com

Bây giờ hãy cho ENC28J60 gửi 1 request GET tới trang web này nhé. Trong hercules UART mình sẽ gửi xuống

GET / HTTP/1.1\$0D\$0AHost: enc28j60.iot47.com\$0D\$0A\$0D\$0A



Ú òa ! Chúng ta đã tải được mã html của trang web <http://enc28j60.iot47.com/> . Ngay sau khi gửi request xong, do không có thuộc tính keep alive nên server đã chủ động đóng kết nối ngay ! Các bạn hãy thử gửi request tương tự tới google, thingspeak để tải mã html trang web của họ xem nhé 😊

Hoàn thiện thư viện HTTP Client

Chúng ta sẽ hoàn thiện thư viện http.c

Tạo hàm HTTP_request

```
void HTTP_request(uint8_t *IP_server,uint16_t port,uint8_t *data,uint16_t length,uint32_t timeout) {
    TCP_Connect(IP_server,port,timeout,0); //ket noi TCP_sendData(data,length,3000); //gửi tin nhắn với timeout là 3s }
```

Các bạn có thể test hàm bằng cách liên tục gọi request lên server



```
1HTTP_request(ip_server,80,(uint8_t *)"GET / HTTP/1.1\r\nHost: enc28j60.iot47.com\r\n\r\n",44,3000);
```

```
2      HAL_Delay(3000);
```

Demo gửi dữ liệu lên thingspeak

Trong bài 2 của khóa ESP8266 mình đã demo gửi lên thingspeak với tập lệnh AT rồi, lần này cũng tương tự nhé

Thingspeak là 1 nền tảng cung cấp các hoạt động giám sát dữ liệu, mình sẽ demo gửi thử nhiệt độ lên

Các bạn vào <https://thingspeak.com/> đăng kí tài khoản và đăng nhập
Trong mục, channel, hãy thêm 1 feild dưới dạng chart (biểu đồ)

New Channel

Search by

Name

Click on a visualization to add it to the Channel

Channel Location

Field 1 Chart

Field 2 Chart

Thêm biểu đồ

Save Cancel

Vào tab API_Key, người ta rất tâm lí tạo sẵn request mẫu cho mình rồi kia

The screenshot shows the Thingspeak API Keys interface. At the top, there are tabs: Private View, Public View, Channel Settings, Sharing, API Keys (highlighted), and Data Import / Export. A red arrow points up from the Write API Key section to the API Keys tab. The Write API Key section contains a text input field with the key '0JH8D4CLZMDXZ6MM' and a 'Generate New Write API Key' button. The Read API Keys section contains a text input field with the key 'Y3C0DLWAPRI4U9GV' and a note input field. Below these are 'Save Note' and 'Delete API Key' buttons. To the right, there is a 'Help' section with instructions about API keys, 'API Keys Settings' with bullet points, and 'API Requests' with examples for writing and reading channel feeds.

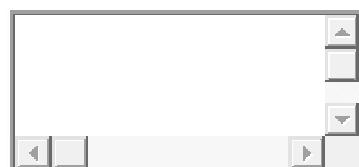
OK, giờ gửi thử request lên xem nào, mình sẽ tạo 1 mảng `uint8_t thingspeak_data[100]`; để có thể add nhiệt độ vào qua hàm sprintf

Cũng đừng quên đổi địa chỉ ip_server sang server của thingspeak nhé



```
1 uint8_t ip_server[4]={184,106,153,149};
```

Được rồi, giờ gửi dữ liệu lên thôi, phần nhiệt độ mình sẽ dùng hàm rand() để tạo chữ cũng k có cảm biến thật (include thư viện math.h để gọi hàm rand) và chia lấy dư cho 99 để giới hạn kết quả random trong phạm vi 0 đến 99



```
1 sprintf((char *)thingspeak_data,"GET
2 http://api.thingspeak.com/update?api_key=0JH8D4CLZMDXZ6MM&field2=%i\r\n\r\n",rand()%99);
3 HTTP_request(ip_server,80,thingspeak_data,strlen((char *)thingspeak_data),3000);
4 HAL_Delay(5000);
```

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

    /* USER CODE BEGIN 3 */
    sprintf((char *)thingspeak_data,"GET http://api.thingspeak.com/update?api_key=0JH8D4CLZMDXZ6MM&field1=%i\r\n\r\n",rand()%99);
    HTTP_request(ip_server,80,thingspeak_data,strlen((char *)thingspeak_data),3000);
    HAL_Delay(500);
}
/* USER CODE END 3 */
}
```

Trong while main mình sẽ cho gửi liên tục mỗi 5s 1 lần nhiệt độ

Và mình sẽ show live biểu đồ đó ở đây cho các bạn xem nhé ⓘ

Có vẻ cái anh thingspeak này mới thay đổi cơ chế, tầm 10 tới 20s mới nhận quest 1 lần thì phải. Nếu trả về 0 là quest đó server không chấp nhận vì gửi nhanh quá

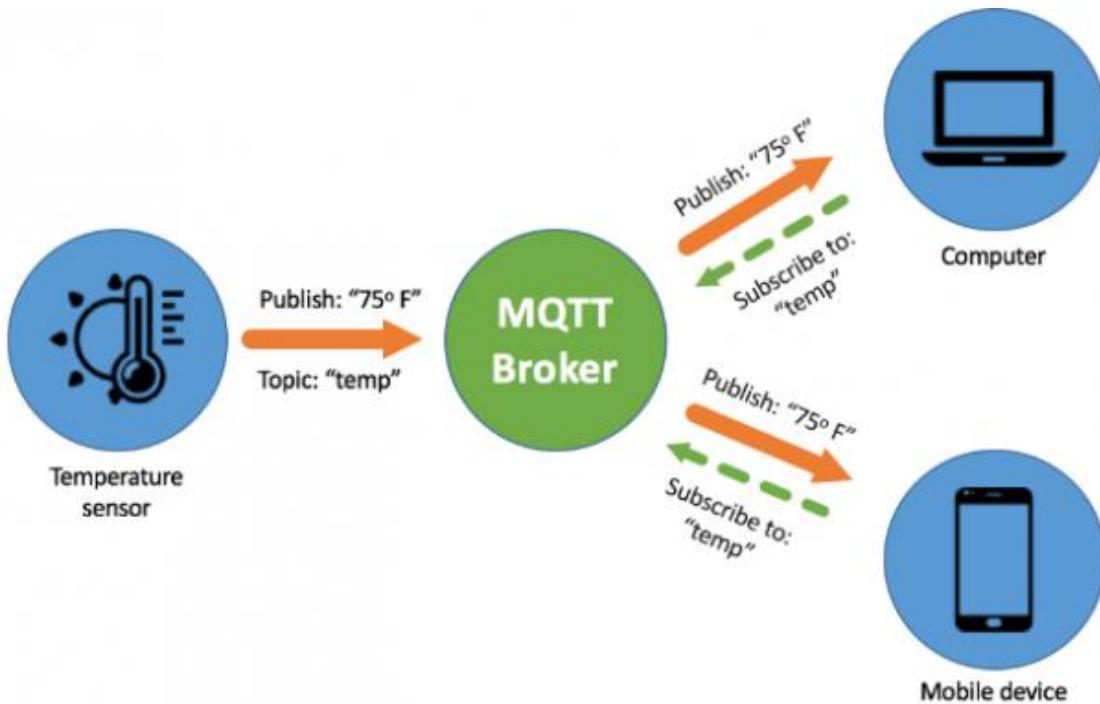
Download

Các bạn có thể tải toàn bộ source cho bài này **tại đây**:

https://drive.google.com/file/d/1qlJ1PhpDJamWCp_xTqw0Mq44o67t06pq/view?usp=sharing

[ENC28J60] Bài 19: Giao thức MQTT

1 Tháng Sáu, 2020 Đào Nguyễn Ethernet, IoT tutorial 2



Giao thức MQTT là giao thức được sử dụng rất nhiều trong thế giới IOT. MQTT có thể chạy trên nền TCP hoặc Socket, do chúng ta mới làm tới TCP nên đương nhiên bài này sẽ hướng dẫn các bạn giao thức TCP trên nền tảng TCP mà chúng ta đã viết từ các bài trước

MQTT cần có 1 broker là trung tâm của mọi luồng dữ liệu, mình sẽ sử dụng broker của trang [hive mqtt broker.hivemq.com](http://broker.hivemq.com)

Nhà cung cấp này cho trung ta 1 broker công khai miễn phí và không bảo mật do vậy rất phù hợp để học tập và thử nghiệm. MQTT trên nền TCP sẽ được phục vụ tại cổng 1883To

You can access the broker at:

Broker: broker.hivemq.com

TCP Port: 1883

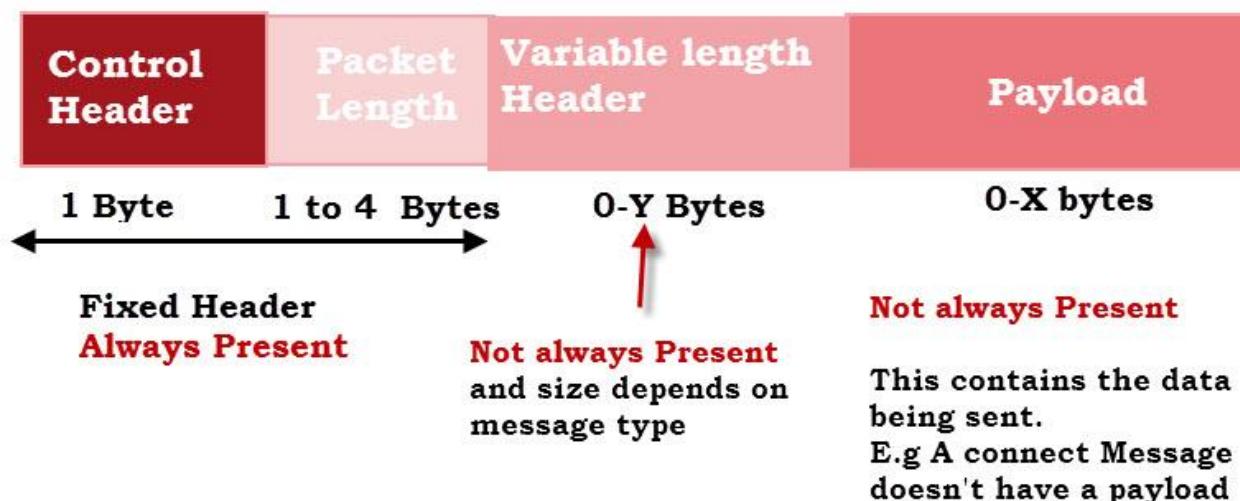
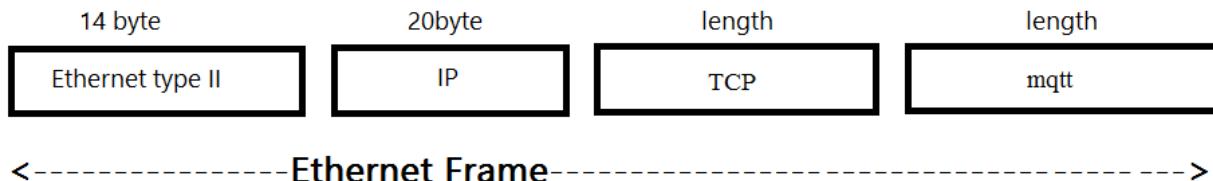
Websocket Port: 8000

Topic, Publish, Subscribe

Đây là 3 khái niệm cơ bản của MQTT, **topic** là 1 chủ đề, hiểu 1 cách đơn giản là đích đến của dữ liệu. **Publish** là hành động xuất bản 1 tin nhắn tới topic nào đó, và **Subscriptions** – đăng ký nhận dữ liệu từ topic nào đó

Ví dụ: Máy tính đăng ký với broker tôi muốn nhận dữ liệu từ topic **cảm biến**, mạch thu thập nhiệt độ sẽ thường xuyên gửi dữ liệu nhiệt độ nó đo được lên topic **cảm biến**. Do máy tính đã đăng ký nhận dữ liệu ở topic này nên máy tính sẽ đc broker gửi tin nhắn này cho. Máy tính cũng có thể đăng ký nhận dữ liệu từ các topic khác nữa

Cấu trúc gói



MQTT Standard Packet Structure

Để làm việc được với MQTT ta cần ít nhất 4 gói tin cơ bản gồm: Gói connect, gói disconnect, gói publish, gói subscribe

Trường **Control Header** : 1 byte , nó có tác dụng điều hướng gói tin mqtt này thuộc loài nào, với:

Gói connect: 0x10

Gói publish: 0x30

Gói subscribe : 0x80

Gói unsubscribe : 0xA0

Gói disconnect: 0xE0

Gói CONNACK : 0x20 (khi gửi in connect m sẽ check gói này để biết kết nối ok không)

Gói PUBACK : 0x40 (khi gửi tin nhắn đi mình check gói này để biết bên nhận đã nhận được chưa)

Gói SUBACK : 0x90 (khi gửi tin nhắn đăng ký topic mình check gói này để biết đã đăng ký được chưa)
Gói UNSUBACK : 0xB0 (khi gửi gói tin hủy đăng ký topic mình check gói này để biết hủy thành công chưa)

Ở byte **Control Header**, 4 byte đầu là số hiệu của gói, 4 byte sau là các option tùy chọn. Ví dụ = 0 thì không cần phản hồi, bằng 2 là có phản hồi. Cụ thể, bạn gửi gói tin connect với Header là 0x10 thì nó sẽ không phản hồi lại gói CONNACK, còn gửi 0x12 thì nó sẽ gửi lại CONNACK để chắc chắn 100% đã kết nối thành công

Trường **Packet Length** mô tả phía sau nó còn bao nhiêu byte nữa (không tính bản thân nó)

Trường này không cố định số lượng byte, tối đa 4 và tối thiểu 1. Bit cao nhất của byte sẽ xác định xem byte phía sau nó có thuộc Packet Length không ! Do đó chỉ có 7bit dùng để mã hóa dữ liệu

Ví dụ 1: Gói tin phía sau còn 31 byte thì ta cần 1 byte cho Packet Length
=> Packet Length = 0x1F

Ví dụ 2: Gói tin phía sau còn 321 byte thì ta sẽ cần 2 byte cho Packet Length với byte1 là byte thấp và byte2 là byte cao. Cú mỗi giá trị của byte cao sẽ = 128 lần byte thấp. Ta tách $321 = 65 + 2 * 128$

=> Packet Length = 0x41 0x02

Bit cao nhất của byte1 phải được set lên 1 để báo vẫn còn 1 byte nữa cho Packet Length nên phải sửa thành
Packet Length = 0xC1 0x02

Với 4 byte Packet Length, ta sẽ mã hóa được tối đa 268,435,455 byte dữ liệu

From	To
0 (0x00)	127 (0x7F)
28 (0x80, 0x01)	16 383 (0xFF, 0x7F)
16 384 (0x80, 0x80, 0x01)	2 097 151 (0xFF, 0xFF, 0x7F)
2 097 152 (0x80, 0x80, 0x80, 0x01)	268 435 455 (0xFF, 0xFF, 0xFF, 0x7F)

Trong bài này, để cho đơn giản mình sẽ mặc định chỉ sử dụng 1 byte cho **Packet Length**

Tiếp tục tới trường thứ 3 là **Variable length Header**, nó gồm 4 trường nhỏ sau: Protocol Name, Protocol Level, Connect Flags, and Keep Alive

- Protocol Name: (6byte) 0x00 0x04 0x4D 0x51 0x54 0x54
- Protocol Level (version): (1byte) 0x04 = Version: MQTT v3.1.1 (4)
- Connect Flags: (1 byte) 0x??
- Keep Alive: (2 byte) 0x?? 0x??

Các bạn chỉ cần quan tâm tới Connect Flags

Cấu trúc của nó như sau:

Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
byte 8	X	X	X	X	X	X	X	0

Bit 1 chúng ta sẽ mặc định để bằng 1

Các bit khác các bạn xài cái nào thì set nó lên. Ví dụ mình không cần user, password thì chỉ cần 0x02 là ok, nếu set cái nào lên thì phải mô tả nó trong payload nhé

Payload là trường sẽ chứa các thông tin để phục vụ việc connect như ID_Client, User, Password. Thông tin trong trường này phải tuân thủ thứ tự sau:

Client ID -> Will Topic -> Will Message -> User Name -> Password

Chú ý: Ở trên byte Connect Flags cái nào không set thì bỏ qua nó trong Payload nhé !

Còn Keep Alive là 2 byte chứa thời gian được tính bằng giây. Đó là khoảng thời gian tối đa được phép trôi qua giữa điểm mà Client hoàn thành việc truyền một Control Packet và điểm mà nó bắt đầu gửi tiếp theo. Nói chung cứ để từ 10 đến 60s tùy các bạn

Tạo thư viện MQTT

Các bạn tạo thêm 2 file [mqtt.c](#) và [mqtt.h](#) cũng như thêm các mã khởi tạo thư viện cơ bản mà chúng ta đã quá quen thuộc



```

1 #ifndef MQTT_H_
2 #define MQTT_H_
3 //-----
4 //Include cac thu vien can thiet
5 #include "stm32f1xx_hal.h"
6 #include <string.h>
7 #include <stdlib.h>
8 #include <stdint.h>
9 #include <stdio.h>
10#include "uart.h"

```

```

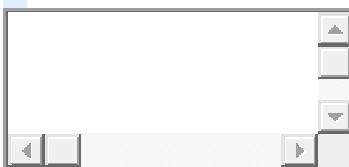
11#include "tcp.h"
12//-----
13typedef struct
14{
15    uint8_t control_header;
16    uint8_t length;
17    uint8_t Protocol_name[6]; //MQTT (0x00 0x04 0x4D 0x51 0x54 0x54)
18    uint8_t Protocol_version;
19    uint8_t flag;
20    uint16_t Keep_alive;
21    uint16_t data_length;
22    uint8_t data[];
23}__attribute__((packed)) MQTT_connect_struct; //struct size = size_data + 14
24typedef struct
25{
26    uint8_t control_header;
27    uint8_t length;
28    uint16_t topic_length;
29    uint8_t data[];
30}__attribute__((packed)) MQTT_mess_struct; //struct size = size_data + 4
31typedef struct
32{
33    uint8_t control_header;
34    uint8_t length;
35    uint16_t mess_identifier; //0x0001
36    uint16_t topic_length;
37    uint8_t data[];
38}__attribute__((packed)) MQTT_sub_struct; //struct size = size_data + 6 + 1 (ket thuc co 1 byte 0x00 nua)
39typedef struct
40{
41    uint8_t *topic;
42    uint16_t topic_length;
43    uint8_t *mess;

```

```

44     uint16_t mess_length;
45 }__attribute__((packed)) MQTT_callback_typedef;
46typedef void (*mqtt_fun_typedef)(MQTT_callback_typedef *);
47//-----
48void MQTT_connect(uint8_t *ip_broker,uint16_t port,char *client_id);
49void MQTT_publish(char *topic,char *mess);
50void MQTT_subscriber(char *topic); //gui 1 tin nhan toi topic bat ki
51void MQTT_read(uint8_t *mqtt_data,uint16_t mqtt_length);
52void MQTT_registerCallback(mqtt_fun_typedef mqtt);
53//-----
54#endif /* MQTT_H_ */

```



```

1#include "mqtt.h"
2extern const uint8_t macaddr[6];
3extern uint8_t ip[4];
4extern char debug_string[60];
5extern uint8_t eth_buffer[BUFFER_LENGTH];
6extern uint8_t gateway_ip[4];
7extern uint8_t gateway_mac[6];
8
9//-----END FILE-----//

```

Hàm Connect



```

1 void MQTT_connect(uint8_t *ip_broker,uint16_t port,char *client_id)
2 {
3     uint8_t *mqtt_ptr;
4     uint16_t struct_len,data_len;

```

```

5     data_len = strlen(client_id);
6     struct_len = data_len+14; //day la kich thuoc cua toan bo goi tin MQTT
7
8     mqtt_ptr = calloc(struct_len,sizeof(uint8_t)); //su dung cap phat dong
9
10    MQTT_connect_struct *MQTT = (MQTT_connect_struct *)mqtt_ptr; //khoi tao cau truc mqtt len vung nho dc cap
11    phat
12    MQTT->control_header = 0x10; //connect
13
14    MQTT->data_length = swap16(data_len);
15    MQTT->length = 12 + data_len;
16    memcpy(MQTT->Protocol_name,"\\x00\\x04MQTT",6);
17    MQTT->Protocol_version = 0x04; //Version: MQTT v3.1.1
18    MQTT->flag = 0x02;
19    MQTT->Keep_alive = swap16(60);
20    memcpy(MQTT->data,client_id,data_len);
21
22    UART.putString("Ket noi toi broker MQTT\\r\\n");
23    TCP_request(ipBroker,port,mqtt_ptr,struct_len,3000);
24
25    free(mqtt_ptr); //giai phong vung nho
}

```

Chú ý: Các bạn tạo thêm hàm TCP_request trong thư viện tcp.c nhé



```

1void TCP_request(uint8_t *IP_server,uint16_t port,uint8_t *data,uint16_t length,uint32_t timeout)
2{
3    TCP_Connect(IP_server,port,timeout,0); //ket noi
4    TCP_sendData(data,length,3000); //gửi tin nhắn với timeout là 3s
5}

```

Hàm Publish



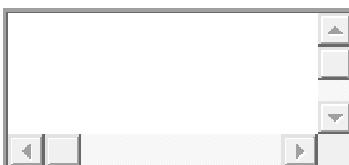
```
1 void MQTT_publish(char *topic,char *mess) //gui 1 tin nhan toi topic bat ki
2 {
3     uint8_t *mqtt_ptr;
4     uint16_t topic_length = strlen(topic);
5     uint16_t mess_length = strlen(mess);
6     uint16_t struct_len = topic_length + mess_length + 4; //tinh toan do dai cua struct
7
8     mqtt_ptr = calloc(struct_len,sizeof(uint8_t)); //su dung cap phat dong
9     MQTT_mess_struct *MQTT = (MQTT_mess_struct *)mqtt_ptr; //su dung cap phat dong de khai tao con tro struct
10    mqtt
11
12    MQTT->control_header = 0x30; //publish
13    MQTT->length = topic_length + mess_length + 2;
14    MQTT->topic_length = swap16(topic_length);
15    memcpy(MQTT->data,topic,topic_length); //copy topic vao struct
16    memcpy(MQTT->data+topic_length,mess,mess_length); //copy tin nhan vao struct
17
18    TCP_sendData(mqtt_ptr,struct_len,3000); //gui tin nhan voi timeout la 3s
19    free(mqtt_ptr); //giai phong vung nho
20 }
```

2 hàm trên mình sử dụng cấp phát động để khởi tạo bộ nhớ cho struct

Test thử

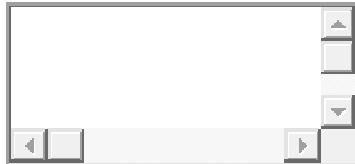
Chúng ta sẽ kết nối tới broker của hive MQTT và gửi thử 1 vài tin nhắn lên để kiểm tra chương trình hoạt động ok chưa nhé !

Trước tiên trong file main hãy đổi ip server thành 3.127.99.166 vì đó chính là ip của hive mqtt. Mình cũng tạo thêm 1 buff để lưu tin nhắn sẽ gửi và 1 biến i lưu số hiệu tin nhắn



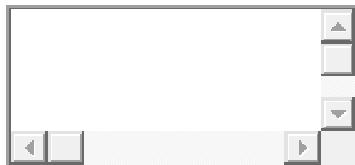
```
1uint8_t ip_server[4]={3,127,99,166};  
2uint8_t buff[100],i;
```

Trong hàm main, trước while(1) mình sẽ gọi hàm MQTT connect với id của client là enc28j60



```
1MQTT_connect(ip_server,1883,"enc28j60");
```

Và trong main cứ 2s gửi 1 tin nhắn lên topic tên là IOT47



```
1sprintf(buff,"Xin chao,toi la ENC28J60,tin nhan so %i",i++);  
2MQTT_publish("IOT47",buff);  
3HAL_Delay(2000);
```

OK, mình sẽ show giao diện debug ở ngay đây

Demo gửi nhận MQTT

Tin nhắn gửi tới topic IOT47: ...

Topic:

Tin nhắn:

Gửi

Demo gửi nhận MQTT

Tin nhắn từ esp8266: Xin chao,toi la ENC28J60, tin nhan so 3

Topic:

Tin nhắn:

Gửi

Related posts:



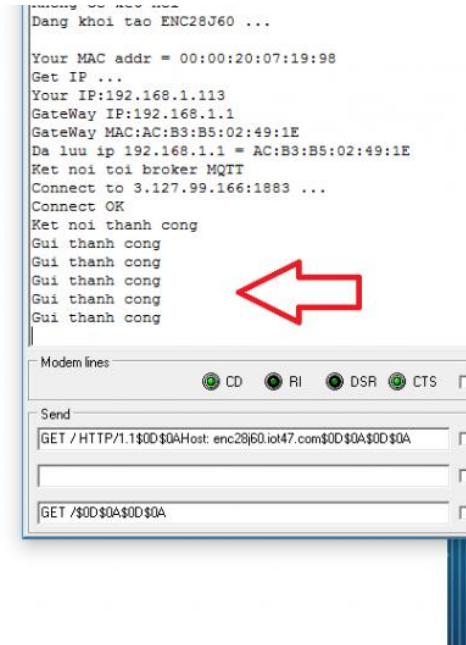
Mình đã nhận được tin nhắn gửi lên rồi web ! Chúc các bạn thành công 😊

Viết hàm Subscriber

Hàm subscriber



```
1 void MQTT_subscriber(char *topic) //gui 1 tin nhan toi topic bat ki
2 {
3     uint8_t *mqtt_ptr;
4     uint16_t topic_length = strlen(topic);
5     uint16_t struct_len = topic_length + 6 + 1; //tinh toan do dai cua struct
6
7     mqtt_ptr = calloc(struct_len,sizeof(uint8_t)); //su dung cap phat dong
8     mqtt_ptr[struct_len-1]=0; //end point = zero
9
10    MQTT_sub_struct *MQTT = (MQTT_sub_struct *)mqtt_ptr;
11
12    MQTT->control_header = 0x80; // sub
```



```

13     MQTT->length = struct_len-2;
14     MQTT->mess_identifier = swap16(0x0001);
15     MQTT->topic_length = swap16(topic_length);
16     memcpy(MQTT->data,topic,topic_length); //copy topic vao struct
17
18     UART_putString("Dang ki topic:");UART_putString(topic);UART_putString("\r\n");
19     TCP_sendData(mqtt_ptr,struct_len,3000); //gửi tin nhắn tcp với timeout là 3s
20     free(mqtt_ptr); //giai phong vung nho
21}

```

Xử lí các gói tin xác nhận (ACK)

Để chắc chắn việc kết nối, đăng ký thành công 100%, chúng ta sẽ bắt thêm các bản tin ACK nữa nhé, cụ thể mình sẽ bắt gói SUBACK, và CONACK, tuy nhiên mình sẽ để các bạn tự viết thêm chương trình bắt các gói xác nhận này nếu muốn, mình sẽ demo việc bắt gói Publish

Bây giờ mình sẽ viết phương thức MQTT_read để bên thư viện TCP gọi nó ra. Nó giống như việc thằng TCP nhận đc 1 gói tin nhìn giống gói MQTT nên sẽ gọi hàm MQTT_read và bảo "ê, có gói tin na ná mqtt này, tự đi mà xử lí tau không quan tâm"

Quay lại thư viện TCP, mở `tcp.h` và add thư viện `mqtt.h` vô

Ở cuối hàm `TCP_readData` thêm phần kiểm tra xem có gói tin mqtt nào không rồi gọi hàm `MQTT_read` cho em nó xử lí

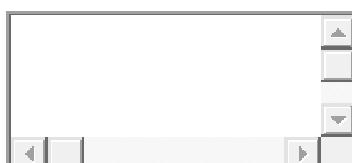


```

1//kiem tra goi tin xem co phai la goi MQTT publish khong
2
3         if(TCP_Struct_Frame->data[data_offset] == 0x30) //bat publish
4             MQTT_read(&TCP_Struct_Frame->data[data_offset],data_len);

```

OK ! bây giờ quay lại hàm `MQTT_read` và chúng ta sẽ xử lí data ở đó



```

1 void MQTT_read(uint8_t *mqtt_data,uint16_t mqtt_length)
2 {

```

```

3     MQTT_callback_typedef mqtt;
4     //kiem tra do dai cua topic mqtt
5     if((mqtt_data[1] & 0x80) == 0)
6     {
7         mqtt.topic_length = swap16(mqtt_data[1]);
8         mqtt.topic = &mqtt.topic[4];
9         mqtt.mess = &mqtt.topic[4+mqtt_data[1]];
10        mqtt.mess_length = mqtt_length - 4+mqtt_data[1];
11
12        UART_putString("Da nhan goi MQTT\r\nTopic:");
13        for(int i=0;i<mqtt.topic_length;i++)
14            UART_putChar(mqtt.topic[i]);
15        UART_putString("\r\nMess:");
16        for(int i=0;i<mqtt.mess_length;i++)
17            UART_putChar(mqtt.mess[i]);
18        UART_putString("\r\n");
19    }
20    else
21        UART_putString("Tin nhan qua dai ! khong ho tro\r\n");
22}

```

Trước khi viết tiếp mình sẽ test thử xem có đúng là đã nhận được bản tin publish nào chưa đã

Quay về, hàm main, sau khi connect tới broker xong, gọi hàm MQTT_subscriber

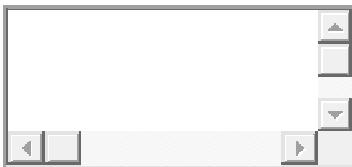


```
1MQTT_subscriber("iot47_hehe");
```

Và ở trang này các bạn cuộn lên giao diện MQTT debug mình đã viết bên trên, thử gửi 1 tin nhắn tới topic **iot47_hehe** xem

Đăng kí call back nhận tin nhắn

Các bạn có thể viết code thực thi gì đó ở hàm MQTT read luôn, nhưng để độc lập các thư viện mình sẽ tạo hàm callback để có thể viết hàm xử lí ở file main. Minh sẽ tạo 1 con trỏ hàm để trả tới hàm cần cần gọi, các bạn mở file [mqtt.c](#) và khởi tạo 1 con trỏ hàm ở đầu file



Và tạo thêm hàm đăng kí callback



```
2{  
3     mqtt_function = mqtt; //tro vao ham can goi  
4}
```

Khi chúng ta gọi hàm này và nạp vào địa chỉ của hàm cần callback, biến toàn cục [mqtt_function](#) sẽ lưu lại địa chỉ của hàm đó

Bây giờ, trong hàm MQTT_read, tiến hành gọi hàm callback



Hàm [MQTT_read](#) lúc này như sau:



```

2 {
3     MQTT_callback_typedef mqtt;
4     //kiem tra do dai cua topic mqtt
5     if((mqtt_data[1] & 0x80) == 0)
6     {
7         mqtt.topic_length = swap16(mqtt_data[1]);
8         mqtt.topic = &mqtt.topic[4];
9         mqtt.mess = &mqtt.topic[4+mqtt_data[1]];
10        mqtt.mess_length = mqtt_length - 4+mqtt_data[1];
11
12        mqtt_function(&mqtt); // goi ham thuc thi
13    }
14    else
15        UART_putString("Tin nhan qua dai ! khong ho tro\r\n");
16}

```

Bây giờ các bạn có thể quay về file main, tạo 1 hàm với 1 tên bất kì, ví dụ



```

1 void mqtt_callback(MQTT_callback_typedef *mqtt)
2 {
3     //your code
4 }

```

Hàm này nhập vào con trỏ struct chứa các thông tin topic, mess để các bạn so sánh hay xử lí gì thì tùy

Sau đó, sau khi kết nối tới broker, gọi
`MQTT_registerCallback(&mqtt_callback);`

Lúc này, mỗi khi có tin nhắn gửi tới, hàm `mqtt_callback` sẽ tự động được gọi

Callback là 1 cách rất hay ho để tăng tính độc lập giữa các thư viện

Gói ping

Chúng ta phải thường xuyên ping tới broker để nó biết client vẫn đang hoạt động, khi gửi ping request thì broker cũng sẽ gửi ping response. Vậy là cả 2 thằng đều biết là cả 2 vẫn hoạt động bình thường

Thời gian ping phải nhỏ hơn thời gian keep alive, nếu không broker sẽ ngắt kết nối và cho rằng client đã offline. Lúc này broker sẽ phát đi 1 tin nhắn tới cho các client khác trong mạng đã đăng ký topic bảo rằng thằng này đã offline (đương nhiên muốn làm thế thì client phải chủ động thỏa thuận với broker topic sẽ nhận và nội dung sẽ nhận khi tôi offline), hiểu nôm na như để lại di chúc vậy

Do vậy, MQTT có khái niệm **will retain**

< mình sẽ cập nhật hàm đăng ký gói tin này sau> trong phần này sẽ tập trung viết gói ping

Gói Disconnect