**STM Bài 96. LAN8720. LWIP. máy khách TCP. Phần 1**

Đăng trên[15 Tháng Mười Một, 2017](https://narodstream.ru/stm-urok-96-lan8720-lwip-tcp-client-chast-1/)bởi [Narod Stream](https://narodstream.ru/author/admin/) Đăng trong [lập trình STM32](https://narodstream.ru/rub_stm32/)— [5 bình luận ↓](https://narodstream.ru/stm-urok-96-lan8720-lwip-tcp-client-chast-1/#comments)

Chúng tôi đã nghiên cứu kỹ hai vi mạch được thiết kế để truyền dữ liệu qua giao diện mạng LAN: đó là ENC28J60 và W5500. Đây là những con chip từ hai nhà sản xuất khác nhau. Cấp độ đầu tiên bao gồm hai cấp độ - kênh và vật lý, và cấp độ thứ hai - ngoài tất cả những điều này, nó còn là mạng và vận chuyển, tức là nó là một loại hiện thân của ngăn xếp TCP / IP đã hoàn thành ở cấp độ phần cứng. Điều này rất thuận tiện cho nhà phát triển, vì bạn không phải viết toàn bộ ngăn xếp, như chúng tôi đã làm với ENC28J60, nhưng nó áp đặt một số hạn chế đối với quyền tự do hành động, vì không thể thay đổi điều gì đó trong hoạt động của ngăn xếp, mặc dù có chế độ RAW cho phép bạn tắt ngăn xếp . Nhưng tôi đã không sử dụng chế độ này, vì mã rất có thể sẽ giống với mã của tất cả các bài học về vi mạch đầu tiên và tại sao chúng ta cần lặp lại - điều đó sẽ rất lãng phí thời gian,

Do đó, hôm nay chúng ta sẽ thử làm việc với một vi mạch khác truyền dữ liệu qua bus này - đó là **LAN8720** . Con chip này, giống như đại diện được đề cập đầu tiên, là đứa con tinh thần của **Microchip** , chỉ có điều nó chỉ chứa lớp vật lý trên bo mạch của nó. Có vẻ như điều này đặt ra một trách nhiệm to lớn đối với nhà phát triển, buộc anh ta phải tổ chức thành thạo nhiều cấp độ như vậy, nhưng hoàn toàn không phải vậy. Bởi vì thư viện **LWIP đến để giải cứu**. Thư viện này ban đầu được phát triển bởi Adam Dunkels tại Viện Kiến trúc Mạng và Máy tính Thụy Điển (CNA). Hiện tại, nhóm các nhà phát triển lwIP đang tham gia vào thư viện này, nó được phân phối theo giấy phép BSD đã sửa đổi, nghĩa là nó được sử dụng miễn phí. **LWIP (IP nhẹ)** là thư viện triển khai ngăn xếp giao thức với mã nguồn hỗ trợ các giao thức sau:

* IPv4 và IPv6 (Giao thức Internet v4 và v6)
* ICMP (Giao thức tin nhắn điều khiển Internet) để bảo trì và gỡ lỗi mạng
* IGMP (Giao thức quản lý nhóm Internet) để quản lý lưu lượng phát đa hướng
* UDP (Giao thức gói dữ liệu người dùng) • TCP (Giao thức điều khiển truyền dẫn)
* DNS (Máy chủ tên miền)
* SNMP (Giao thức quản lý mạng đơn giản)
* DHCP (Giao thức cấu hình máy chủ động)
* PPP (Giao thức điểm tới điểm)
* ARP (Giao thức phân giải địa chỉ)

Và đây không phải là tất cả các giao thức được thư viện này hỗ trợ, còn có một số giao thức khác.

Ngoài ra còn có ba API có thể được sử dụng để phát triển phần mềm giao tiếp

**RAW / API gốc** - một giao diện được sử dụng mà không cần hệ điều hành. Nó có một số nhược điểm, nhưng cũng có một số ưu điểm, chẳng hạn như cung cấp quyền tự do hành động, điều thường thiếu ở các lập trình viên cao cấp hơn.

**Netconn API** là API nối tiếp cấp cao yêu cầu hệ điều hành thời gian thực (RTOS). Netconn API cho phép hoạt động đa luồng.

**BSD Socket API** - Một API tương tự như Berkeley-Socket (được phát triển dựa trên API của Netconn).

Bây giờ chúng ta sẽ làm việc với giao diện RAW đầu tiên. Thứ nhất, vì chúng tôi chưa nghiên cứu kỹ về hệ điều hành FreeRTOS và thứ hai, chúng tôi chưa quen với việc tự phục vụ các sự kiện mạng và tôi nghĩ sẽ không quá khó để thực hiện nếu không sử dụng HĐH.

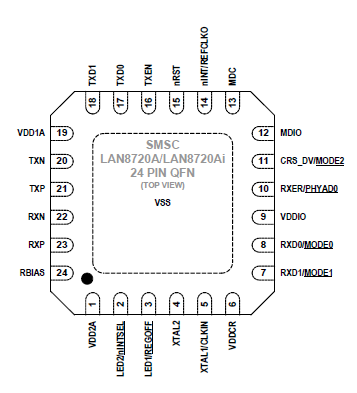
Không cần phải nói nhiều về bản thân chip **LAN8720**  , vì không có nhiều thanh ghi như W5500 có.

Điều đáng nói là, không giống như W5500, nó hỗ trợ đầy đủ công nghệ **Auto-MDIX** , thường không đủ hỗ trợ để kết nối các nút mà không sử dụng bộ tập trung và bộ định tuyến, tức là trực tiếp bằng cáp không chéo thông thường.

Ngoài ra, vi mạch này hỗ trợ trao đổi ở tốc độ 10/100 megabit mỗi giây ở cả chế độ bán song công và song công hoàn toàn, tất nhiên, chế độ Tự động đàm phán được hỗ trợ, cho phép các nút tự thương lượng về tốc độ trao đổi dữ liệu.

Tôi nghĩ rằng phần còn lại của sự quyến rũ của vi mạch này, chúng ta sẽ cảm nhận được khi bắt đầu sử dụng nó. Ngoài ra, bạn luôn có thể tải xuống phiên bản mới nhất của tài liệu kỹ thuật cho vi mạch này trên trang web chính thức của nhà sản xuất và làm quen với tất cả những điều tinh tế của vi mạch này mà bạn quan tâm.

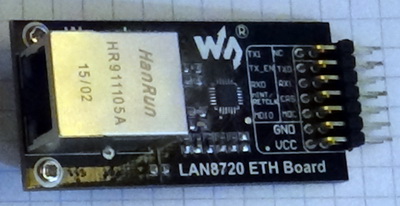
Hãy xem vị trí của các chân trên đó



Các chân này đại diện cho tất cả các chân cần thiết cho bus RMII, chúng ta sẽ nói về điều này sau một chút, cũng như tất cả các chân cần thiết để cung cấp lớp vật lý, và tất nhiên, có các chân để tạo xung nhịp với bộ cộng hưởng thạch anh, cũng như chân cắm điện.

Tôi có con chip này trên mô-đun **WaveShare** . Chúng tôi thường sử dụng các mô-đun từ công ty này.

Đây là những gì mô-đun của chúng tôi trông giống như



Tất cả các liên hệ được đánh dấu, nghĩa là bạn có thể sử dụng một cách an toàn.

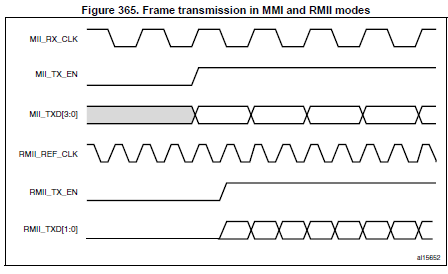
Chúng tôi chỉ có thể kết nối vi mạch này, cũng như mô-đun, với bất kỳ bộ điều khiển nào. Vì vi mạch này không có cấp độ kênh nên nó phải được hỗ trợ và tốt nhất là trong phần cứng bởi chính vi điều khiển. Là một bộ điều khiển như vậy, chúng tôi sẽ sử dụng STM32F407VG, nằm trên bảng STM32F4-Discovery yêu thích của chúng tôi, mà chúng tôi đã trải qua rất nhiều bài học trước đó và biết về nó từ lâu.

Hiện tại, đối với nhiều, không nhiều nhưng hầu như tất cả các bo mạch thế hệ mới của nó, STM cũng cài đặt một vi mạch tương tự, tuy thuộc thế hệ mới hơn nhưng tôi chưa nhận thấy sự khác biệt trong hoạt động. Nhưng vì Discovery 4 yêu quý của chúng tôi không có một vi mạch như vậy và tôi chắc chắn rằng nhiều người muốn tổ chức trao đổi dữ liệu mạng với bảng này, chúng tôi sẽ chỉ kết nối một mô-đun như vậy với nó, vì hai vi mạch mà chúng tôi đã xem xét trước đó với kênh mức hiện tại của họ với điều này, thật vô nghĩa khi kết nối bảng gỡ lỗi, bởi vì MK nằm trên đó đã có cấp kênh do phần cứng triển khai và tôi nghĩ rằng không có lý do gì để nó không hoạt động.

[Quảng cáo từ **RtbSape**](https://rtb.sape.ru/?utm_source=adsbyrtbsape&utm_medium=banner&utm_campaign=overlay)

Các vi mạch như vậy, chỉ có một lớp vật lý, được kết nối với bộ vi điều khiển STM32F407 thông qua một trong các giao diện liên kênh **MII** hoặc **RMII .**. Đây là những lốp xe tương tự, hơi khác nhau. Dữ liệu được truyền theo các hướng khác nhau trên các dây riêng biệt. MII sử dụng 4 dây để truyền và nhận dữ liệu cho mỗi hướng. Giao diện là đồng bộ, nghĩa là có một chân đồng bộ hóa, trong trường hợp sử dụng MII, các xung có tần số 25 megahertz được áp dụng. Do đó, chúng tôi truyền 4 bit thông tin cùng một lúc trong một lần đồng bộ hóa. Đây là cách chúng tôi đạt được tốc độ 100 megabit mỗi giây. Hơn nữa, chúng ta có thể truyền đồng thời với tốc độ như vậy theo cả hai hướng, vì chúng ta có 4 dây cho mỗi hướng. Trong trường hợp sử dụng RMII, chúng tôi đã có ít dây hơn, tức là chỉ có 2 dây cho mỗi hướng, nhưng do thực tế là xung nhịp đã được thực hiện ở tần số 50 megahertz, thì chúng tôi cũng truyền và nhận dữ liệu với tốc độ 100 megabit/giây. Đó là sự khác biệt giữa các giao diện này.

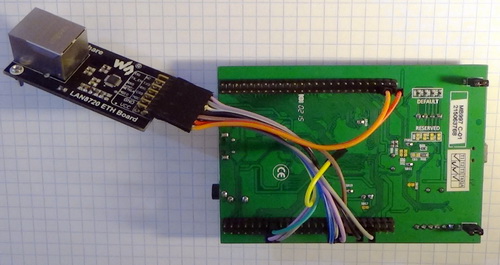
Tôi cũng sẽ đưa ra các ví dụ minh họa về truyền dữ liệu qua các giao diện từ tài liệu dành cho bộ điều khiển



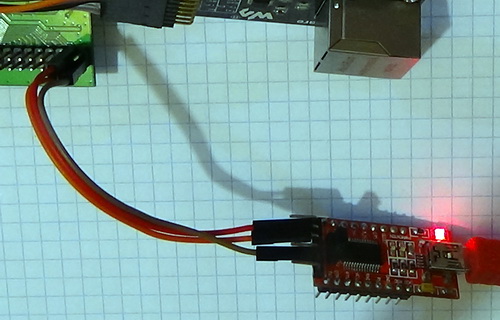
Bây giờ, tôi nghĩ nó đã trở nên rõ ràng hơn nhiều đối với mọi người. Chúng tôi thấy rằng xung nhịp trên giao diện RMII nhanh gấp đôi và có số chân cắm bằng một nửa.

Bạn và tôi sẽ sử dụng chính xác RMII, vì chúng tôi không có lựa chọn nào khác, vì chip LAN8720 của chúng tôi chỉ hoạt động trên một giao diện như vậy. Có một mặt tích cực cho điều này. Chúng tôi thắng về số lượng dây, nhưng điều này buộc chúng tôi phải chịu trách nhiệm về chất lượng của chính những dây này. Chúng tôi không thể sử dụng chúng quá dài, vì 50 megahertz để truyền dữ liệu không còn thấp nữa, mặc dù không cao lắm. Tôi không nghĩ bạn cần che chắn dây. Ít nhất tôi ổn với nó.

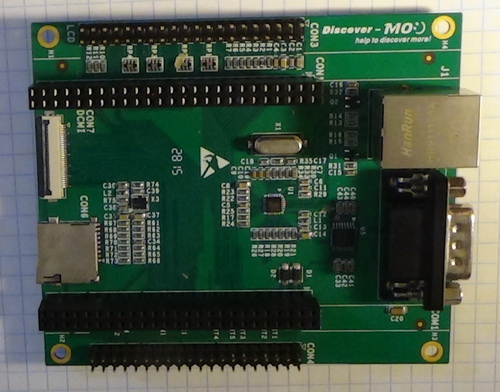
Đây là cách tôi có một vi mạch được kết nối với bảng (bấm vào hình ảnh để phóng to hình ảnh)

[](https://narodstream.ru/wp-content/uploads/2017/11/image03.jpg)

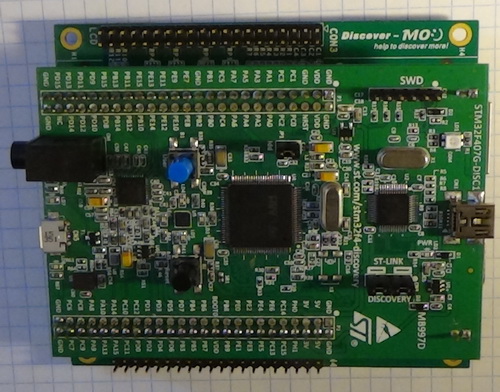
Chúng ta cũng hãy kết nối bộ điều hợp USB-TTL để điều khiển và giám sát qua giao diện USART. Chúng tôi sẽ kết nối nó với các liên hệ của xe buýt USART6



Tôi đã kết nối bộ điều hợp với bus USART6 không chỉ vì lý do thuận tiện cho vị trí của nó ở rìa mà còn vì khả năng tương thích với bo mạch STM32F4DIS **-BB** , mà tôi cũng có sẵn, trong đó có bảng gỡ lỗi STM32F4-DISCOVERY được lắp vào rất thuận tiện và rất tiện lợi , vì nó có nhiều giải pháp làm sẵn, chẳng hạn như chip LAN8720 đã được cài đặt sẵn trên bo mạch ở đó. Ngoài ra, bạn có thể kết nối màn hình và chân riêng biệt được hiển thị, máy quay video, còn có đầu đọc thẻ Micro-SD, nói chung là rất nhiều điều thú vị. Đây là góc nhìn từ trên xuống của cô ấy



Và đây là giao diện của bảng này với bảng gỡ lỗi STM32F4-DISCOVERY được chèn vào nó



Như chúng ta có thể thấy, USART được tạo trên bo mạch này dưới dạng đầu nối cổng COM và nó được kết nối sơ đồ với USART6. Thành thật mà nói, tôi vẫn chưa thể sử dụng nó. Nó vẫn chưa kết hợp với cổng COM ảo của tôi, tuy nhiên, đó là lý do tại sao tôi đã kết nối bộ điều hợp USB-TTL của mình với các tiếp điểm xe buýt USART6 mà không sử dụng bảng mở rộng này.

Đầu tiên, chúng tôi sẽ làm việc mà không có bảng mở rộng, sau đó, với sự cho phép của bạn, tôi sẽ chuyển sang nó, nó thuận tiện hơn cho tôi và bạn có thể tiếp tục sử dụng mô-đun từ WaweShare, cũng tiện lợi và rất rẻ . Tất nhiên, nếu bạn không có bo mạch mở rộng **STM32F4DIS-BB** , nếu có thì tôi khuyên bạn nên sử dụng nó. Và sau đây tôi sẽ nói về sự khác biệt chính giữa việc thiết lập một dự án bằng cách sử dụng bảng mở rộng và không sử dụng nó. Tôi đã không nhận thấy điều này ngay lập tức, điều mà nói chung tôi không được phép làm nếu không có bảng này.

Trong [**phần tiếp theo của**](https://narodstream.ru/stm-urok-96-lan8720-lwip-tcp-client-chast-2/) hướng dẫn, chúng ta sẽ tạo và thiết lập một dự án, viết một số chức năng và cố gắng thực sự kết nối với máy chủ TCP.

# STM Bài 96. LAN8720. LWIP. máy khách TCP. Phần 2

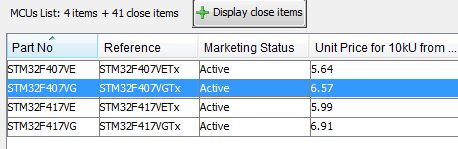
Đăng trên[22 Tháng Mười Một, 2017](https://narodstream.ru/stm-urok-96-lan8720-lwip-tcp-client-chast-2/)bởi [Narod Stream](https://narodstream.ru/author/admin/) Đăng trong [lập trình STM32](https://narodstream.ru/rub_stm32/)— [44 bình luận ↓](https://narodstream.ru/stm-urok-96-lan8720-lwip-tcp-client-chast-2/#comments)

Trong [**phần trước**](https://narodstream.ru/stm-urok-96-lan8720-lwip-tcp-client-chast-1/) của bài học, chúng ta đã làm quen với chip LAN8720, với các giao diện liên kênh và ngăn xếp giao thức LWIP.

Tốt. Cuối cùng chúng ta hãy chuyển sang dự án của chúng ta.

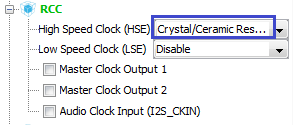
Nhiệm vụ của chúng tôi là tạo một máy khách TCP có thể chủ động kết nối và ngắt kết nối với máy chủ TCP và chương trình của chúng tôi cũng có thể nhận và xem dữ liệu đến ở dạng văn bản, đồng thời gửi dữ liệu tương tự tới máy chủ TCP. Tức là, một kiểu trò chuyện sử dụng giao thức TCP sẽ phải xuất hiện.

Chúng ta sẽ tạo một project mới. Với tư cách là bộ điều khiển, chúng tôi sẽ chọn tương ứng bộ điều khiển **STM32F407VGTx của chúng tôi**

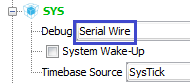


Sau đó, chúng tôi sẽ thiết lập dự án của chúng tôi.

Trước hết là RCC



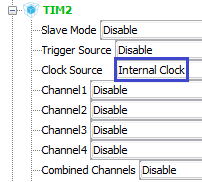
Sau đó lập trình viên



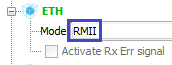
Kích hoạt USART

hình ảnh10

Hãy bật hẹn giờ cho tương lai, có lẽ trong dự án này, chúng tôi sẽ không cần nó



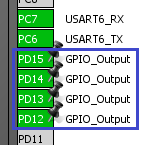
Bây giờ Ethernet



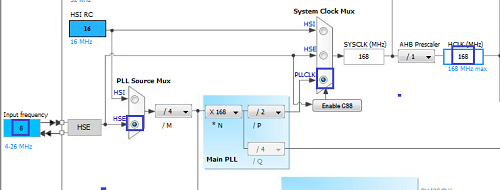
Sau đó, chúng tôi sẽ kết nối thư viện ngăn xếp giao thức LWIP

hinh13

Bật chân của đèn LED cho đầu ra, điều này sẽ mang lại sự thuận tiện và rõ ràng trong việc gỡ lỗi dự án

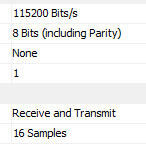


Hãy vào **Cấu hình đồng hồ** và định cấu hình bộ chia và bộ nhân để đặt tần số cần thiết (nhấp vào hình ảnh để phóng to hình ảnh)

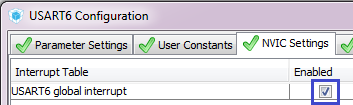
[](https://narodstream.ru/wp-content/uploads/2017/11/image15.png)

Hãy đi đến Cấu hình.

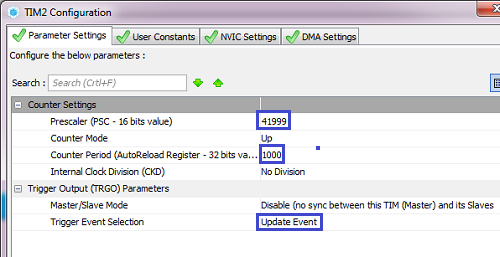
Nếu bạn có các tham số như vậy trong **USART6 , thì chúng tôi không chạm vào bất cứ thứ gì ở đây**



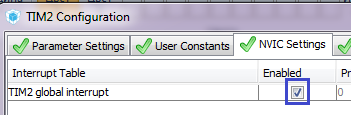
Chuyển đến tab USART6 với các ngắt và bật chúng



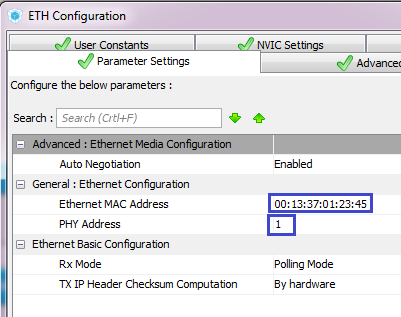
Sau đó, hãy thiết lập bộ đếm thời gian của chúng tôi



Đồng thời bật bộ hẹn giờ ngắt

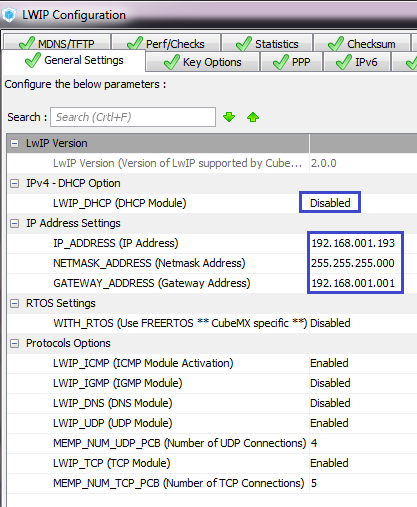


Bây giờ ETH. Tab có tham số

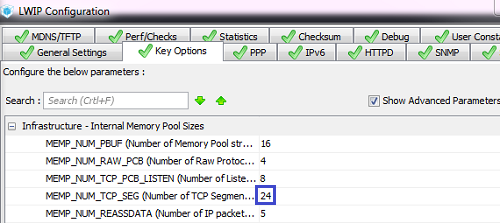


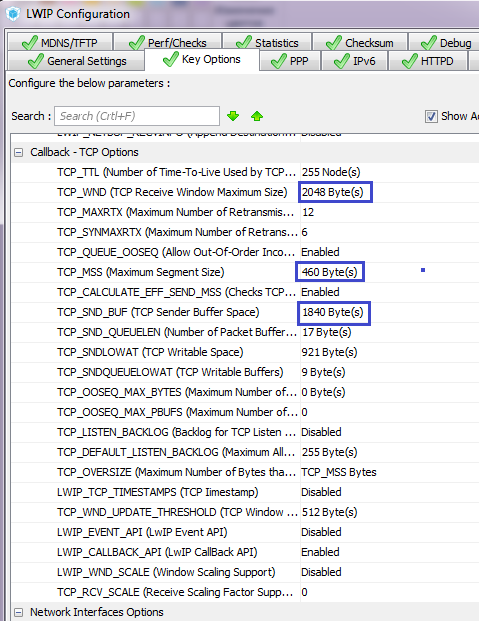
Chúng tôi đặt địa chỉ MAC mong muốn ở đó, cũng như địa chỉ đăng ký để truy cập chip thông qua RMII. Chúng tôi để lại **1** . Và đây là sự khác biệt! **1** - đây là trường hợp sử dụng mô-đun từ WaveShare và **0** - trong trường hợp sử dụng bảng mở rộng STM32F4DIS-BB.

Bây giờ hãy thiết lập LWIP. Đầu tiên, tab **Cài đặt chung** . Hãy tắt DHCP, chúng tôi sẽ sử dụng địa chỉ tĩnh để chúng tôi không liên tục tìm ra địa chỉ của nút của chúng tôi trên một nút khác. Vâng, cũng cho các mục đích trong tương lai. Chúng tôi sẽ kết nối trực tiếp hai bộ điều khiển với nhau thông qua giao diện mạng LAN và không chắc rằng bộ điều khiển khác sẽ phân phối chính xác IP cho bộ điều khiển của chúng tôi



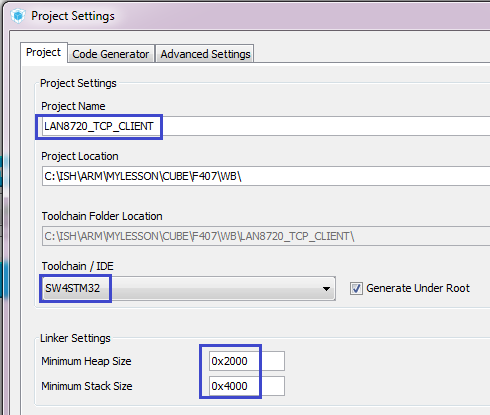
Sau đó, hãy chuyển đến tab **Tùy chọn chính** , đảm bảo rằng hộp kiểm có cài đặt nâng cao đã được chọn và thay đổi một số tham số tại đây để giữ cho kích thước cửa sổ quen thuộc với chúng tôi - 2048





Chúng tôi không chạm vào hoặc kết nối bất kỳ giao thức nào nữa.

Hãy chuyển đến cài đặt dự án, tăng ngăn xếp và đống và thiết lập tạo dự án cho Bàn làm việc hệ thống, tốt, hãy đặt tên cho dự án của chúng ta



Áp dụng cài đặt, tạo dự án, truy cập **Bàn làm việc hệ thống** , kết nối dự án đã tạo của chúng tôi ở đó, chuyển đến cài đặt của nó, xóa tất cả cấu hình gỡ lỗi, nếu có và đặt mức tối ưu hóa thành **1** như bình thường .

Hãy cố gắng xây dựng dự án của chúng tôi.

Hãy chuyển đến tệp **main.c**  và kết nối cấu trúc cho giao diện mạng của chúng tôi

/\* USER CODE BEGIN PV \*/

/\* Private variables ---------------------------------------------------------\*/

**extern struct netif gnetif;**

/\* USER CODE END PV \*/

Hãy viết mã tiêu chuẩn sau trong một vòng lặp vô hạn để đảm bảo hoạt động liên tục của ngăn xếp của chúng ta

/\* USER CODE BEGIN 3 \*/

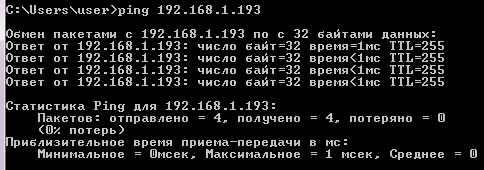
**ethernetif\_input(&gnetif);**

**sys\_check\_timeouts();**

}

/\* USER CODE END 3 \*/

Kết nối mạch của chúng tôi, lắp ráp mã, flash bộ điều khiển và thử ping mô-đun của chúng tôi



Hãy tạo hai tệp để làm việc với mạng - **net.h** và **net.c** với nội dung ban đầu sau

**mạng.h:**

**#ifndef NET\_H\_**

**#define NET\_H\_**

**//-----------------------------------------------**

**#include "stm32f4xx\_hal.h"**

**#include <string.h>**

**#include <stdlib.h>**

**#include <stdint.h>**

**#include "lwip.h"**

**#include "lwip/tcp.h"**

**//-----------------------------------------------**

**#endif /\* NET\_H\_ \*/**

**net.c**

**#include "net.h"**

**//-----------------------------------------------**

**extern UART\_HandleTypeDef huart6;**

**//-----------------------------------------------**

Chúng tôi cũng sẽ đưa thư viện của mình vào tệp **main.c**

/\* USER CODE BEGIN Includes \*/

**#include "net.h"**

/\* USER CODE END Includes \*/

Hãy chuyển sang tệp tiêu đề **net.h.**

Vì chúng tôi đang viết một ứng dụng khách TCP, bằng cách nào đó chúng tôi sẽ cần đưa ra lệnh để kết nối với máy chủ TCP. Để làm điều này, chúng ta sẽ cần biết địa chỉ IP cũng như địa chỉ cổng. Tất nhiên, không khó để nhận ra chúng, nhưng chúng có thể khác nhau mỗi lần, vì vậy chúng tôi không thể mã hóa chúng vào dự án của mình. Chúng ta phải bằng cách nào đó chuyển chúng đến dự án mỗi lần như các tham số của lệnh kết nối. Để làm điều này, xe buýt USART của chúng tôi sẽ phục vụ chúng tôi và chúng tôi sẽ thực hiện việc này từ chương trình đầu cuối, sau đó phân tích cú pháp dòng xuất phát từ đó trong dự án. Do đó, để bắt đầu, hãy viết một cấu trúc với các thuộc tính của giao diện USART của chúng tôi

#include "lwip/tcp.h"

**//-----------------------------------------------**

**typedef struct USART\_prop{**

**uint8\_t usart\_buf[26];**

**uint8\_t usart\_cnt;**

**uint8\_t is\_tcp\_connect;//статус попытки создать соединение TCP с сервером**

**uint8\_t is\_text;//статус попытки передать текст серверу**

**} USART\_prop\_ptr;**

**//-----------------------------------------------**

Bây giờ, hãy chuyển đến tệp **net.c** và tạo một biến kiểu cấu trúc của chúng tôi

extern UART\_HandleTypeDef huart6;

**USART\_prop\_ptr usartprop;**

Chúng tôi cũng sẽ tạo một chức năng để khởi tạo mạng của mình bằng cách đặt lại giá trị của các trường cấu trúc trong đó

USART\_prop\_ptr usartprop;

**//-----------------------------------------------**

**void net\_ini(void)**

**{**

**usartprop.usart\_buf[0]=0;**

**usartprop.usart\_cnt=0;**

**usartprop.is\_tcp\_connect=0;**

**usartprop.is\_text=0;**

**}**

**//-----------------------------------------------**

Hãy thêm một nguyên mẫu cho chức năng của chúng tôi trong tệp tiêu đề và gọi nó trong hàm **main()** trong tệp **main.c** , đồng thời bật bộ đếm thời gian của chúng tôi ở đó

/\* USER CODE BEGIN 2 \*/

**HAL\_TIM\_Base\_Start\_IT(&htim2);**

**net\_ini();**

/\* USER CODE END 2 \*/

Hãy quay lại **net.c** và thêm chức năng xử lý ngắt USART để nhận dữ liệu

**//-----------------------------------------------**

**void UART6\_RxCpltCallback(void)**

**{**

**}**

**//-----------------------------------------------**

Hãy tạo một nguyên mẫu cho hàm này, quay lại main.c và thêm một trình xử lý ngắt thực sự ở đó, trong đó chúng ta sẽ gọi hàm của mình

/\* USER CODE BEGIN 4 \*/

**//-----------------------------------------------**

**void HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart)**

**{**

**if(huart==&huart6)**

**{**

**UART6\_RxCpltCallback();**

**}**

**}**

**//-----------------------------------------------**

/\* USER CODE END 4 \*/

Hãy quay lại tệp net.c và thêm một biến toàn cục dưới dạng một mảng chuỗi

USART\_prop\_ptr usartprop;

**char str[30];**

Một lần nữa, hãy quay lại tệp **main.c** và kết nối mảng chuỗi của chúng ta ở đó

extern struct netif gnetif;

**extern char str[30];**

Trong hàm **main()** , chúng ta sẽ gọi lệnh nhận ký tự, nếu không USART của chúng ta sẽ không bao giờ bắt đầu nhận

net\_ini();

**HAL\_UART\_Receive\_IT(&huart6,(uint8\_t\*)str,1);**

Sau đó, hãy quay lại tệp **net.c** , chuyển đến trình xử lý của chúng tôi, tạo một biến ở đó và xử lý trường hợp vượt quá kích thước của bộ đệm dòng và cuối cùng, chúng tôi sẽ tăng bộ đếm byte của mình trong bộ đệm và gọi dữ liệu nhận lại lệnh

void UART6\_RxCpltCallback(void)

{

**uint8\_t b;**

**b = str[0];**

**//если вдруг случайно превысим длину буфера**

**if (usartprop.usart\_cnt>25)**

**{**

**usartprop.usart\_cnt=0;**

**HAL\_UART\_Receive\_IT(&huart6,(uint8\_t\*)str,1);**

**return;**

**}**

**usartprop.usart\_cnt++;**

**HAL\_UART\_Receive\_IT(&huart6,(uint8\_t\*)str,1);**

}

Sau khi kiểm tra nếu vượt quá bộ đệm, chúng ta sẽ nhập ký tự nhận được vào bộ đệm

  return;

}

**usartprop.usart\_buf[usartprop.usart\_cnt] = b;**

usartprop.usart\_cnt++;

Nếu gặp ký tự xuống dòng thì ta thêm ký tự cuối dòng vào phần tử tiếp theo của mảng chuỗi

usartprop.usart\_buf[usartprop.usart\_cnt] = b;

**if(b==0x0A)**

**{**

**usartprop.usart\_buf[usartprop.usart\_cnt+1]=0;**

**}**

Ở trên, chúng tôi thêm một chức năng để phân tích chuỗi nhận được của chúng tôi

**//-----------------------------------------------**

**void string\_parse(char\* buf\_str)**

**{**

**}**

**//-----------------------------------------------**

Hãy quay lại trình xử lý của chúng ta và gọi hàm này trong điều kiện

if(b==0x0A)

{

  usartprop.usart\_buf[usartprop.usart\_cnt+1]=0;

**string\_parse((char\*)usartprop.usart\_buf);**

Sau đó, trong cùng một điều kiện, sau khi phân tích chuỗi, chúng tôi đặt lại bộ đếm của mình, gửi lệnh nhận một ký tự tới bus USART, ký tự này sẽ đợi ký tự tiếp theo từ bus và thoát khỏi chức năng của chúng tôi

  string\_parse((char\*)usartprop.usart\_buf);

**usartprop.usart\_cnt=0;**

**HAL\_UART\_Receive\_IT(&huart6,(uint8\_t\*)str,1);**

**return;**

}

Trong chức năng phân tích chuỗi, hiển thị chuỗi đã nhận trong chương trình đầu cuối để thấy rằng chúng tôi đã nhận mọi thứ chính xác

void string\_parse(char\* buf\_str)

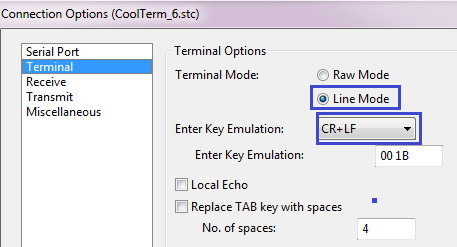
{

**HAL\_UART\_Transmit(&huart6, (uint8\_t\*)buf\_str,strlen(buf\_str),0x1000);**

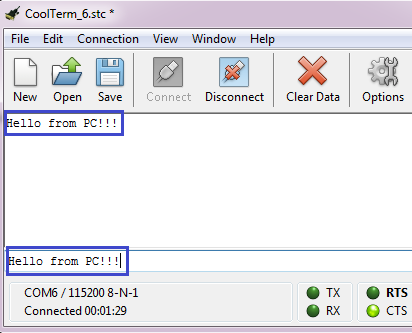
}

Hãy thu thập mã, flash bộ điều khiển, chạy chương trình đầu cuối. Lần này tôi sử dụng Cool Term vì nó có khả năng cấu hình truyền dữ liệu linh hoạt.

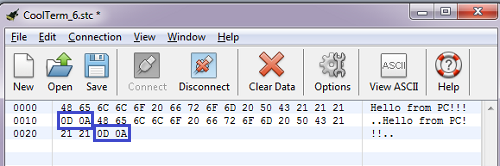
Hãy đi vào phần cài đặt của chương trình này, định cấu hình cổng và tốc độ của chúng tôi (tôi nghĩ điều này không đáng để hiển thị - bạn có thể tự xử lý), đồng thời thiết lập việc chuyển các ký tự xuống dòng và xuống dòng ở cuối dòng



Hãy kết nối với cổng và thử gửi một cái gì đó



Như chúng ta có thể thấy, tiếng vang hoạt động với chúng ta, điều đó có nghĩa là chúng ta đã bắt đúng dòng trên bộ điều khiển. Để làm rõ rằng chúng tôi cũng nhận được và chúng tôi cũng không làm mất chúng - các ký tự xuống dòng và xuống dòng có thể được chuyển bằng cách chuyển dòng tiếp theo, nó sẽ phải được lặp lại từ một dòng mới và cũng bằng cách nhấn View Hex trên thanh công cụ của chương trình đầu cuối, sau đó chúng ta sẽ thấy mã của các ký tự đến



Chúng tôi thấy các biểu tượng đang đến.

Hãy quay lại chức năng của chúng ta để phân tích chuỗi và tiếp tục viết phần thân của nó.

Và trước khi viết mã thân hàm, hãy đồng ý rằng chúng ta sẽ gửi lệnh sau để kết nối với máy chủ

**t:** địa chỉ IP máy chủ: địa chỉ cổng máy chủ

Và ngắt kết nối bằng một lệnh tương tự

**c:** địa chỉ IP máy chủ: địa chỉ cổng máy chủ

Về lý thuyết, bạn không cần chỉ định địa chỉ để ngắt kết nối, nhưng đột nhiên chúng tôi sẽ có một số kết nối và chúng tôi muốn ngắt chính xác một trong số chúng. Hiện tại chúng ta sẽ chỉ có một, nhưng trong tương lai mọi thứ đều có thể.

Và bây giờ cơ thể

  HAL\_UART\_Transmit(&huart6, (uint8\_t\*)buf\_str,strlen(buf\_str),0x1000);

**// если команда попытки соединения ("t:")**

**if (strncmp(buf\_str,"t:", 2) == 0)**

**{**

**}**

**//статус попытки разорвать соединение ("c:")**

**else if (strncmp(buf\_str,"c:", 2) == 0)**

**{**

**}**

**else**

**{**

**}**

}

Hiện tại, chúng tôi sẽ chỉ có ba tùy chọn cho các dòng - một lệnh để kết nối, một lệnh để ngắt kết nối, cũng như văn bản thuần túy để trò chuyện.

Ở trên ta thêm chức năng xử lý lệnh

**//-----------------------------------------------**

**void net\_cmd(char\* buf\_str)**

**{**

**}**

**//-----------------------------------------------**

Hãy xử lý lệnh kết nối với máy chủ bằng cách nhập một mã nhất định vào phần thân của điều kiện tương ứng trong hàm phân tích cú pháp chuỗi

if (strncmp(buf\_str,"t:", 2) == 0)

{

**usartprop.usart\_cnt-=1;**

**usartprop.is\_tcp\_connect=1;//статус попытки создать соединение TCP с сервером**

**net\_cmd(buf\_str+2);**

**HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_12, *GPIO\_PIN\_SET*);**

}

Chúng tôi sẽ thực hiện một thao tác tương tự trong điều kiện ngắt kết nối với máy chủ

else if (strncmp(buf\_str,"c:", 2) == 0)

{

**usartprop.usart\_cnt-=1;**

**usartprop.is\_tcp\_connect=2;//статус попытки разорвать соединение TCP с сервером**

**net\_cmd(buf\_str+2);**

**HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_13, *GPIO\_PIN\_SET*);**

}

Chúng tôi sẽ giải quyết các dòng sau. Trong khi chờ đợi, hãy chuyển sang chức năng xử lý lệnh, thêm một mảng gồm bốn phần tử cho địa chỉ IP, một biến và hai điều kiện vào đó

void net\_cmd(char\* buf\_str)

{

**uint8\_t ip[4];**

**uint16\_t port;**

**if(usartprop.is\_tcp\_connect==1)//статус попытки создать соединение TCP с сервером**

**{**

**}**

**if(usartprop.is\_tcp\_connect==2)//статус попытки разорвать соединение TCP с сервером**

**{**

**}**

}

Ở trên, chúng tôi sẽ thêm các chức năng để phân tích chuỗi với giá trị của địa chỉ IP và cổng, chúng tôi đã sử dụng các chức năng như vậy trong các dự án trước đó, vì vậy sẽ không có ý nghĩa gì khi lên án chúng. Vì vậy, tôi sẽ chỉ cung cấp mã của họ

**port\_extract**

**ip\_extract**

Chúng tôi cũng sẽ tạo các biến toàn cầu để lưu trữ địa chỉ máy chủ của chúng tôi, nơi chúng tôi hy vọng sẽ sớm kết nối.

#include "net.h"

//-----------------------------------------------

**uint8\_t ipaddr\_dest[4];**

**uint16\_t port\_dest;**

Chúng tôi cũng sẽ thêm một mảng chuỗi khác để chuẩn bị và xuất dữ liệu dịch vụ khác nhau sang USART

char str[30];

**char str1[100];**

Trên cùng, sau khi khai báo các biến, thêm chức năng kết nối với máy chủ thông qua giao thức TCP

char str1[100];

**//-----------------------------------------------**

**void tcp\_client\_connect(void)**

**{**

**}**

**//-----------------------------------------------**

Bây giờ hãy để nó nằm rỗng, chúng ta chỉ cần gọi một cái gì đó trong hàm xử lý lệnh **net\_cmd** của chúng ta , chúng ta sẽ quay lại để viết phần thân của điều kiện kết nối máy chủ

if(usartprop.is\_tcp\_connect==1)//статус попытки создать соединение TCP с сервером

{

**ip\_extract(buf\_str,usartprop.usart\_cnt-1,ipaddr\_dest);**

**port\_dest=port\_extract(buf\_str,usartprop.usart\_cnt-1);**

**usartprop.usart\_cnt=0;**

**usartprop.is\_tcp\_connect=0;**

**tcp\_client\_connect();**

**sprintf(str1,"%d.%d.%d.%d:%u\r\n", ipaddr\_dest[0],ipaddr\_dest[1],ipaddr\_dest[2],ipaddr\_dest[3],port\_dest);**

**HAL\_UART\_Transmit(&huart6,(uint8\_t\*)str1,strlen(str1),0x1000);**

**HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_13, *GPIO\_PIN\_SET*);**

}

Trong phần thân của điều kiện này, chúng ta sẽ lấy địa chỉ máy chủ từ chuỗi, ghi chúng vào biến toàn cục, đặt lại biến trạng thái, gọi chức năng kết nối máy chủ, hiển thị địa chỉ trong chương trình đầu cuối và bật đèn LED.

Phía trên chức năng khởi tạo, thêm một chức năng khác sẽ được gọi khi tạo kết nối đến máy chủ, cái gọi là chức năng gọi lại)

**//-----------------------------------------------**

**static err\_t tcp\_client\_connected(void \*arg, struct tcp\_pcb \*tpcb, err\_t err)**

**{**

**return err;**

**}**

**//-----------------------------------------------**

Dưới dạng đối số, hàm này sẽ nhận một con trỏ tới các đối số, cũng như một con trỏ tới cấu trúc của kết nối TCP và trạng thái của một lỗi có thể xảy ra, mà chúng tôi cũng sẽ quay lại khi thoát khỏi chức năng này.

Hãy thêm một nguyên mẫu cho chức năng này trong cùng một tệp (không phải trong tiêu đề) cao hơn

char str1[100];

//-----------------------------------------------

**static err\_t tcp\_client\_connected(void \*arg, struct tcp\_pcb \*tpcb, err\_t err);**

//-----------------------------------------------

Chúng ta cũng hãy thêm một biến toàn cục của kiểu cấu trúc kết nối TCP và một bộ đếm tin nhắn

char str1[100];

**struct tcp\_pcb \*client\_pcb;**

**\_\_IO uint32\_t message\_count=0;**

Và cũng thêm một mảng cho dữ liệu

char str1[100];

**u8\_t data[100];**

Bây giờ chúng ta quay lại viết phần thân của hàm kết nối với server **tcp\_client\_connect**

void tcp\_client\_connect(void)

{

**ip\_addr\_t DestIPaddr;**

**client\_pcb = tcp\_new();**

**if (client\_pcb != NULL)**

**{**

**IP4\_ADDR( &DestIPaddr, ipaddr\_dest[0], ipaddr\_dest[1], ipaddr\_dest[2], ipaddr\_dest[3]);**

**tcp\_connect(client\_pcb,&DestIPaddr,port\_dest,tcp\_client\_connected);**

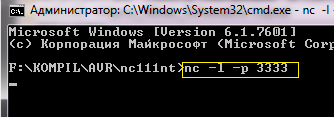
**HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_14, *GPIO\_PIN\_SET*);**

**}**

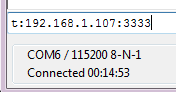
}

Chúng tôi đã tạo một phiên bản của cấu trúc kết nối, sau đó nhập địa chỉ IP của máy chủ vào biến và gọi hàm kết nối máy chủ, vì các tham số chúng tôi đã chuyển thể hiện cấu trúc, địa chỉ máy chủ, cổng máy chủ, cũng như chức năng gọi lại sẽ là được gọi trong trường hợp kết nối với máy chủ. Mặc dù chúng tôi vẫn phải viết rất nhiều chức năng đảm bảo hoạt động đáng tin cậy của máy khách, nhưng nếu bây giờ chúng tôi lắp ráp mã và cố gắng kết nối với máy chủ, thì rất có thể chúng tôi đã thành công. Hãy thử.

Chúng tôi thu thập mã, flash bộ điều khiển, sau đó chạy chương trình netcat trên PC để tạo máy chủ TCP sẽ lắng nghe trên một cổng cụ thể



Chúng ta cũng hãy khởi chạy bộ phân tích lưu lượng Wireshark và ra lệnh trong chương trình đầu cuối để kết nối với cổng máy chủ 3333



Hãy xem WireShark

hình ảnh31

Chúng tôi đã kết nối thành công với máy chủ. Vì vậy, chúng tôi đang đi đúng hướng.

Trong [**phần tiếp theo của**](https://narodstream.ru/stm-urok-96-lan8720-lwip-tcp-client-chast-3/) bài học, chúng ta sẽ viết một số chức năng và tìm hiểu cách ngắt kết nối chính xác với máy khách, cũng như gửi và nhận dữ liệu từ nó.

# STM Bài 96. LAN8720. LWIP. máy khách TCP. Phần 3

Đăng trên[23 Tháng Mười Một, 2017](https://narodstream.ru/stm-urok-96-lan8720-lwip-tcp-client-chast-3/)bởi [Narod Stream](https://narodstream.ru/author/admin/) Đăng trong [lập trình STM32](https://narodstream.ru/rub_stm32/)— [12 bình luận ↓](https://narodstream.ru/stm-urok-96-lan8720-lwip-tcp-client-chast-3/#comments)

Trong [**phần trước**](https://narodstream.ru/stm-urok-96-lan8720-lwip-tcp-client-chast-2/) của hướng dẫn, chúng ta đã tạo và cấu hình một dự án, viết một số chức năng và cố gắng kết nối với máy chủ TCP trong thực tế.

Bây giờ chúng tôi cần ngắt kết nối bằng cách nào đó và chúng tôi chưa viết bất kỳ mã nào. Trong dòng lệnh với netcat, chúng ta áp dụng tổ hợp phím **Ctrl + C** và máy chủ sẽ ngắt kết nối với máy khách

hình ảnh32

Hãy thử ngắt kết nối khỏi máy chủ của chúng tôi.

Để thực hiện việc này, hãy viết mã nội dung của điều kiện thứ hai trong net\_cmd.

if(usartprop.is\_tcp\_connect==2)//статус попытки разорвать соединение TCP с сервером

{

**ip\_extract(buf\_str,usartprop.usart\_cnt-1,ip);**

**port=port\_extract(buf\_str,usartprop.usart\_cnt-1);**

**usartprop.usart\_cnt=0;**

**usartprop.is\_tcp\_connect=0;**

**//проверим что IP правильный**

**if(!memcmp(ip,ipaddr\_dest,4))**

**{**

**//также проверим, что порт тоже правильный**

**if(port==port\_dest)**

**{**

**/\* close tcp connection \*/**

**tcp\_recv(client\_pcb, NULL);**

**tcp\_sent(client\_pcb, NULL);**

**tcp\_poll(client\_pcb, NULL,0);**

**tcp\_close(client\_pcb);**

**HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_15, *GPIO\_PIN\_SET*);**

**}**

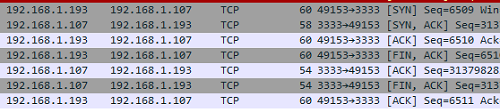
**}**

}

Ở đây, chúng tôi cũng trích xuất địa chỉ IP và số cổng của máy chủ từ chuỗi, nhập chúng vào các biến cục bộ, so sánh chúng với các biến toàn cầu và nếu nó hội tụ, chúng tôi ngắt kết nối khỏi máy chủ, trước tiên hãy hủy địa chỉ của các hàm gọi lại của sự liên quan.

Hãy lắp ráp mã, flash bộ điều khiển và kiểm tra xem kết nối với máy chủ có bị ngắt kết nối chính xác bằng lệnh thích hợp hay không.

Một lần nữa, theo cách tương tự, sử dụng lệnh, chúng tôi kết nối với máy chủ, sau đó sử dụng lệnh **“c:”** , chúng tôi yêu cầu máy khách ngắt kết nối



Chúng tôi thấy rằng việc ngắt kết nối diễn ra chính xác.

Tuyệt vời! Chúng tôi vẫn phải dạy khách hàng của mình nhận và truyền dữ liệu.

Trước tiên, hãy thêm tất cả các chức năng tiện ích của chúng tôi.

Để làm điều này, chúng tôi sẽ tạo thêm bốn hàm gọi lại bằng cách thêm chúng phía trên hàm khởi tạo

**//-----------------------------------------------**

**static err\_t tcp\_client\_recv(void \*arg, struct tcp\_pcb \*tpcb, struct pbuf \*p, err\_t err)**

**{**

**err\_t ret\_err;**

**return ret\_err;**

**}**

**//----------------------------------------------------------**

**static void tcp\_client\_send(struct tcp\_pcb \*tpcb, struct client\_struct \* es)**

**{**

**}**

**//----------------------------------------------------------**

**static err\_t tcp\_client\_sent(void \*arg, struct tcp\_pcb \*tpcb, u16\_t len)**

**{**

**return *ERR\_OK*;**

**}**

**//----------------------------------------------------------**

**static err\_t tcp\_client\_poll(void \*arg, struct tcp\_pcb \*tpcb)**

**{**

**err\_t ret\_err;**

**return ret\_err;**

**}**

**//----------------------------------------------------------**

Chúng ta sẽ làm quen với mục đích của các chức năng này sau.

Hãy cũng thêm các nguyên mẫu cho các chức năng này trong tệp hiện tại

static err\_t tcp\_client\_connected(void \*arg, struct tcp\_pcb \*tpcb, err\_t err);

**static err\_t tcp\_client\_recv(void \*arg, struct tcp\_pcb \*tpcb, struct pbuf \*p, err\_t err);**

**static void tcp\_client\_send(struct tcp\_pcb \*tpcb, struct client\_struct \* es);**

**static err\_t tcp\_client\_sent(void \*arg, struct tcp\_pcb \*tpcb, u16\_t len);**

**static err\_t tcp\_client\_poll(void \*arg, struct tcp\_pcb \*tpcb);**

Tạo một enum cho các trạng thái kết nối

struct tcp\_pcb \*client\_pcb;

**//-----------------------------------------------**

**enum client\_states**

**{**

***ES\_NOT\_CONNECTED* = 0,**

***ES\_CONNECTED*,**

***ES\_RECEIVED*,**

***ES\_CLOSING*,**

**};**

**//-------------------------------------------------------**

Sau đó, một cấu trúc khác cho các thuộc tính của kết nối máy khách

**//-------------------------------------------------------**

**struct client\_struct**

**{**

**enum client\_states state; /\* connection status \*/**

**struct tcp\_pcb \*pcb; /\* pointer on the current tcp\_pcb \*/**

**struct pbuf \*p\_tx; /\* pointer on pbuf to be transmitted \*/**

**};**

**struct client\_struct \*cs;**

**//-----------------------------------------------**

Bài tập lĩnh vực được giải thích trong các ý kiến.

Bây giờ, hãy thêm một chức năng để ngắt kết nối với máy chủ một cách chính xác sau chức năng **tcp\_client\_connected**

**//----------------------------------------------------------**

**static void tcp\_client\_connection\_close(struct tcp\_pcb \*tpcb, struct client\_struct \* es)**

**{**

**/\* remove callbacks \*/**

**tcp\_recv(tpcb, NULL);**

**tcp\_sent(tpcb, NULL);**

**tcp\_poll(tpcb, NULL,0);**

**if (es != NULL)**

**{**

**mem\_free(es);**

**}**

**/\* close tcp connection \*/**

**tcp\_close(tpcb);**

**}**

**//----------------------------------------------------------**

Trong phần thân của chức năng này, chúng tôi hủy địa chỉ của các chức năng gọi lại, đồng thời giải phóng bộ nhớ được phân bổ cho các trường trong cấu trúc của chúng tôi và ngắt kết nối khỏi máy chủ.

Hãy cũng thêm một nguyên mẫu cho chức năng này trong tệp hiện tại

static err\_t tcp\_client\_connected(void \*arg, struct tcp\_pcb \*tpcb, err\_t err);

**static void tcp\_client\_connection\_close(struct tcp\_pcb \*tpcb, struct client\_struct \* es);**

static err\_t tcp\_client\_recv(void \*arg, struct tcp\_pcb \*tpcb, struct pbuf \*p, err\_t err);

Bây giờ chúng ta hãy bắt đầu viết phần thân của hàm **tcp\_client\_connected** mà chúng ta đã bỏ lỡ trong một thời gian dài mà không có phần thân

static err\_t tcp\_client\_connected(void \*arg, struct tcp\_pcb \*tpcb, err\_t err)

{

**struct client\_struct \*es = NULL;**

**if (err == *ERR\_OK*)**

**{**

**}**

**else**

**{**

**tcp\_client\_connection\_close(tpcb, es);**

**}**

  return err;

}

Chúng tôi tạo một cấu trúc kết nối, sau đó viết một điều kiện là chúng tôi đã vào chức năng này mà không gặp lỗi và nếu có lỗi, thì chúng tôi ngắt kết nối khỏi máy chủ.

Temer sẽ bắt đầu viết phần nội dung của tình trạng trong trường hợp có kết quả khả quan

if (err == *ERR\_OK*)

{

**es = (struct client\_struct \*)mem\_malloc(sizeof(struct client\_struct));**

**if (es != NULL)**

**{**

**}**

**else**

**{**

**tcp\_client\_connection\_close(tpcb, es);**

**return *ERR\_MEM*;**

**}**

}

Ở đây chúng tôi cấp phát bộ nhớ cho cấu trúc của mình và nếu không thể cấp phát bộ nhớ, chúng tôi sẽ ngắt kết nối với máy chủ và để lại chức năng bị lỗi bộ nhớ.

Bây giờ chúng ta viết mã cho phần thân của điều kiện cấp phát bộ nhớ trong trường hợp kết quả đúng

if (es != NULL)

{

**es->state = *ES\_CONNECTED*;**

**es->pcb = tpcb;**

**es->p\_tx = pbuf\_alloc(*PBUF\_TRANSPORT*, strlen((char\*)data) , *PBUF\_POOL*);**

}

Chúng tôi gán các giá trị thích hợp cho các trường trong cấu trúc của chúng tôi - trạng thái của kết nối thành công, cũng là địa chỉ của cấu trúc kết nối và chúng tôi cũng gán địa chỉ của mảng cho dữ liệu dưới dạng bộ đệm.

Không rời khỏi điều kiện, chúng tôi viết mã thêm

es->p\_tx = pbuf\_alloc(*PBUF\_TRANSPORT*, strlen((char\*)data) , *PBUF\_POOL*);

**if (es->p\_tx)**

**{**

**/\* copy data to pbuf \*/**

**pbuf\_take(es->p\_tx, (char\*)data, strlen((char\*)data));**

**/\* pass newly allocated es structure as argument to tpcb \*/**

**tcp\_arg(tpcb, es);**

**/\* initialize LwIP tcp\_recv callback function \*/**

**tcp\_recv(tpcb, tcp\_client\_recv);**

**/\* initialize LwIP tcp\_sent callback function \*/**

**tcp\_sent(tpcb, tcp\_client\_sent);**

**/\* initialize LwIP tcp\_poll callback function \*/**

**tcp\_poll(tpcb, tcp\_client\_poll, 1);**

**/\* send data \*/**

**tcp\_client\_send(tpcb,es);**

**return *ERR\_OK*;**

**}**

Nếu cấp phát bộ nhớ cho bộ đệm thành công, chúng tôi sao chép dữ liệu từ nó, sau đó đưa các đối số vào cấu trúc của chúng tôi, sau đó gán các hàm gọi lại mà chúng tôi đã thêm làm hàm cho các điều kiện khác nhau của kết nối TCP và chuyển bộ đệm nếu không trống. Đối với chức năng **tcp\_client\_poll** , chúng tôi cũng chuyển một tham số - một. Đây là một giá trị tính bằng giây, cho biết tần suất hàm này sẽ được gọi.

Bây giờ hãy bắt đầu viết phần thân của hàm gọi lại để nhận các gói TCP.

Chúng tôi cũng thêm biến cấu trúc của chúng tôi

static err\_t tcp\_client\_recv(void \*arg, struct tcp\_pcb \*tpcb, struct pbuf \*p, err\_t err)

{

**struct client\_struct \*es;**

  err\_t ret\_err;

Chúng tôi chuyển cho nó địa chỉ của các đối số, có cấu trúc tương tự

err\_t ret\_err;

**es = (struct client\_struct \*)arg;**

Hãy thêm mã với một số điều kiện cho những dịp khác nhau

es = (struct client\_struct \*)arg;

**if (p == NULL)**

**{**

**}**

**else if(err != *ERR\_OK*)**

**{**

**}**

**else if(es->state == *ES\_CONNECTED*)**

**{**

**}**

**else if (es->state == *ES\_RECEIVED*)**

**{**

**}**

**else**

**{**

**}**

return ret\_err;

Hãy xử lý điều kiện đầu tiên - nhận một gói không có dữ liệu với trạng thái ngắt kết nối. Đây thường là một gói có cờ FIN từ máy chủ

if (p == NULL)

{

**es->state = *ES\_CLOSING*;**

**if(es->p\_tx == NULL)**

**{**

**tcp\_client\_connection\_close(tpcb, es);**

**}**

**ret\_err = *ERR\_OK* ;**

}

Trong trường hợp này, chúng tôi ngắt kết nối khỏi máy chủ.

Hãy viết mã cơ thể của điều kiện sau - một gói không trống có trạng thái lỗi đã đến

else if(err != *ERR\_OK*)

{

**if (p != NULL)**

**{**

**pbuf\_free(p);**

**}**

**ret\_err = err;**

}

Trong trường hợp này, chúng tôi giải phóng bộ nhớ đệm và rời khỏi thân vòng lặp, đặt mã lỗi vào biến trạng thái trả về.

Chúng tôi xử lý điều kiện sau - một gói đến với kết nối bình thường

else if(es->state == *ES\_CONNECTED*)

{

**message\_count++;**

**tcp\_recved(tpcb, p->tot\_len);**

**HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_15, *GPIO\_PIN\_SET*);**

**es->p\_tx = p;**

**strncpy(str1,es->p\_tx->payload,es->p\_tx->len);**

**str1[es->p\_tx->len] = '\0';**

**HAL\_UART\_Transmit(&huart6, (uint8\_t\*)str1,strlen(str1),0x1000);**

**ret\_err = *ERR\_OK*;**

}

Chúng tôi tăng bộ đếm tin nhắn, chấp nhận gói của chúng tôi vào bộ đệm, gán con trỏ tới bộ đệm cho cấu trúc của chúng tôi, sao chép nội dung của bộ đệm vào một mảng chuỗi, kết thúc chuỗi bằng 0, chuyển nó đến chương trình đầu cuối để xem và rời khỏi phần thân điều kiện, đặt thành công vào biến trạng thái trả về.

Chúng tôi viết mã cho phần thân của điều kiện tiếp theo - nhận dữ liệu thành công

else if (es->state == *ES\_RECEIVED*)

{

**HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_13, *GPIO\_PIN\_SET*);**

**ret\_err = *ERR\_OK*;**

}

Trong trường hợp này, chúng tôi chỉ cần bật đèn LED và rời đi với trạng thái thành công.

Và cơ thể cuối cùng là một trường hợp không phù hợp với tất cả những người khác - theo quy định, đây là xác nhận đã nhận dữ liệu

else

{

**/\* Acknowledge data reception \*/**

**tcp\_recved(tpcb, p->tot\_len);**

**/\* free pbuf and do nothing \*/**

**pbuf\_free(p);**

**ret\_err = *ERR\_OK*;**

}

return ret\_err;

Chúng tôi viết mã cho chức năng gọi lại sau, được gọi khi gửi dữ liệu đến máy chủ

static void tcp\_client\_send(struct tcp\_pcb \*tpcb, struct client\_struct \* es)

{

**struct pbuf \*ptr;**

**err\_t wr\_err = *ERR\_OK*;**

**while ((wr\_err == *ERR\_OK*) &&**

**(es->p\_tx != NULL) &&**

**(es->p\_tx->len <= tcp\_sndbuf(tpcb)))**

**{**

**}**

}

Chúng tôi tạo một con trỏ tới bộ đệm, tạo một biến có trạng thái và tạo một vòng lặp trong đó chúng tôi sẽ ở đó nếu ba điều kiện được đáp ứng cùng một lúc - chúng tôi sẽ không gặp lỗi, sẽ có một con trỏ khác 0 tới bộ đệm và độ dài của bộ đệm sẽ nhỏ hơn hoặc bằng lượng dữ liệu được truyền.

Bây giờ chúng ta viết phần thân của vòng lặp

while ((wr\_err == *ERR\_OK*) &&

(es->p\_tx != NULL) &&

(es->p\_tx->len <= tcp\_sndbuf(tpcb)))

{

**ptr = es->p\_tx;**

**wr\_err = tcp\_write(tpcb, ptr->payload, ptr->len, 1);**

Chúng ta sẽ lấy một con trỏ tới bộ đệm và gọi hàm gửi tới bộ đệm nội dung cần gửi. Cái cuối cùng là lá cờ. Nếu là một, thì gói sẽ được gửi mà không có cờ PSH và nếu là hai, thì với nó.

Chúng tôi viết phần thân của chu kỳ hơn nữa. Hãy tạo ra một số điều kiện

wr\_err = tcp\_write(tpcb, ptr->payload, ptr->len, 1);

**if (wr\_err == *ERR\_OK*)**

**{**

**}**

**else if(wr\_err == *ERR\_MEM*)**

**{**

**}**

**else**

**{**

**/\* other problem ?? \*/**

**}**

Hãy viết mã cơ thể của điều kiện đầu tiên - nếu chức năng ghi vào bộ đệm trả về cho chúng ta một kết quả bình thường

if (wr\_err == *ERR\_OK*)

{

**es->p\_tx = ptr->next;**

**if(es->p\_tx != NULL)**

**{**

**pbuf\_ref(es->p\_tx);**

**}**

**pbuf\_free(ptr);**

}

Ở đây chúng ta đi đến chuỗi dữ liệu tiếp theo, nếu có, sau đó tăng số lượng tham chiếu, sau đó giải phóng bộ đệm.

Hãy xử lý điều kiện sau - nếu lỗi bộ nhớ.

else if(wr\_err == *ERR\_MEM*)

{

**es->p\_tx = ptr;**

}

Chúng tôi gán một con trỏ tới bộ đệm cho trường tương ứng của cấu trúc, tức là chúng tôi hoãn chuyển. Đó là, bộ đệm của chúng tôi vẫn đang bận.

Chà, bây giờ chúng ta hãy để trống trường hợp ngược lại.

Chúng tôi làm việc với phần thân của chức năng gọi lại tiếp theo - **tcp\_client\_sent** , được gọi sau khi bộ đệm được chuyển đến máy chủ.

static err\_t tcp\_client\_sent(void \*arg, struct tcp\_pcb \*tpcb, u16\_t len)

{

**struct client\_struct \*es;**

**LWIP\_UNUSED\_ARG(len);**

**es = (struct client\_struct \*)arg;**

**if(es->p\_tx != NULL)**

**{**

**tcp\_client\_send(tpcb, es);**

**}**

  return *ERR\_OK*;

Chúng tôi cũng tạo một con trỏ tới cấu trúc, truyền các đối số cho nó và nếu còn thứ gì đó trong bộ đệm, chúng tôi sẽ chuyển nó tới máy chủ.

Và hàm gọi lại cuối cùng, phần thân mà chúng ta sẽ bắt đầu viết, là hàm **tcp\_client\_poll** , được gọi liên tục và giám sát mạng, ngay cả khi không có trao đổi

err\_t ret\_err;

**struct client\_struct \*es;**

**es = (struct client\_struct\*)arg;**

**HAL\_GPIO\_TogglePin(GPIOD, GPIO\_PIN\_14);**

Chúng tôi cũng sẽ tạo một con trỏ tới cấu trúc, truyền đối số cho nó và thay đổi trạng thái của đèn LED.

Chúng tôi viết mã của phần thân của hàm hơn nữa

HAL\_GPIO\_TogglePin(GPIOD, GPIO\_PIN\_14);

**if (es != NULL)**

**{**

**}**

**else**

**{**

**}**

return ret\_err;

Chúng tôi đã tạo một điều kiện sẽ kiểm tra xem con trỏ tới cấu trúc kết nối có tồn tại hay không.

Trước tiên chúng ta hãy viết phần thân của kết quả tích cực của điều kiện

if (es != NULL)

{

**if (es->p\_tx != NULL)**

**{**

**}**

**else**

**{**

**if(es->state == *ES\_CLOSING*)**

**{**

**tcp\_client\_connection\_close(tpcb, es);**

**}**

**}**

**ret\_err = *ERR\_OK*;**

}

Nếu chúng ta có nội dung khác không của bộ đệm truyền, thì chúng ta không làm gì cả, vì điều này chúng ta có một chức năng khác và nếu trạng thái của kết nối bị đóng, thì chúng ta sẽ đóng nó và thoát khỏi tình trạng có trạng thái tốt.

Bây giờ trường hợp ngược lại

else

{

**tcp\_abort(tpcb);**

**ret\_err = ERR\_ABRT;**

}

return ret\_err;

Trong trường hợp này, chúng tôi không làm gì cả và có một chức năng đặc biệt **tcp\_abort** cho this và thoát với trạng thái trả về đặc biệt.

Chúng ta đã hoàn thành các chức năng xử lý sự kiện.

Chúng tôi vẫn xử lý một dòng từ thiết bị đầu cuối không phải là lệnh.

Để làm điều này, chúng ta sẽ viết một hàm truyền chuỗi đến máy chủ thông qua  hàm phân tích chuỗi **string\_parse**

**//-----------------------------------------------**

**void sendstring(char\* buf\_str)**

**{**

**tcp\_sent(client\_pcb, tcp\_client\_sent);**

**tcp\_write(client\_pcb, (void\*)buf\_str, strlen(buf\_str), 1);**

**tcp\_output(client\_pcb);**

**usartprop.usart\_cnt=0;**

**usartprop.is\_text=0;**

**}**

**//-----------------------------------------------**

Một chuỗi (và bất kỳ dữ liệu nào nói chung) được truyền qua TCP như sau: đầu tiên, hàm xử lý được khởi tạo, sau đó dữ liệu được ghi vào bộ đệm bằng hàm thư viện đặc biệt, sau đó hàm truyền bộ đệm đã chuẩn bị được gọi. sau khi truyền, chúng tôi sẽ đặt lại bộ đếm và trạng thái của một chuỗi bình thường.

Tất nhiên, chức năng này sẽ không bao giờ hoạt động nếu nó không được gọi.

Hãy chuyển đến  hàm **string\_parse** và viết phần thân của điều kiện phân tích chuỗi

else

{

**usartprop.is\_text=1;**

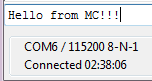
**sendstring(buf\_str);**

**HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_12|GPIO\_PIN\_13, *GPIO\_PIN\_RESET*);**

}

Đó dường như là tất cả với mã. Nó vẫn còn để xem kết quả lao động của chúng tôi.

Hãy biên dịch mã, flash bộ điều khiển, chạy netcat để nghe trên cổng, kết nối với nó bằng một lệnh đặc biệt và thử gửi một cái gì đó đến máy tính từ chương trình đầu cuối của chúng



Hãy xem dòng có đến bảng điều khiển netcat không

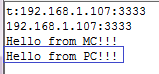
hình ảnh35

Mọi thứ đã đến! Tuyệt vời!

Bây giờ, ngược lại, chúng tôi sẽ chuyển chuỗi từ bảng điều khiển sang MK của chúng tôi

ảnh36

Hãy xem liệu một thông báo đã đến bộ điều khiển trong cửa sổ chương trình đầu cuối chưa

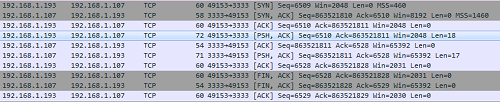


Mọi thứ đã đến.

Hãy gửi một lệnh ngắt kết nối.

Máy khách và máy chủ đã ngắt kết nối thành công.

Hãy xem trao đổi trong WireShark



Cuộc trao đổi đã thành công. Bạn cũng có thể thấy. cho dù tất cả các byte đã đến chưa, nhưng chúng tôi sẽ không làm điều này, vì tôi đã kiểm tra điều này nhiều lần và nếu có lỗi, tôi đã loại bỏ chúng khi viết mã.

Do đó, chúng tôi đã học cách tổ chức máy khách TCP đơn giản nhất bằng cách sử dụng ngăn xếp giao thức LWIP bằng chip **LAN8720** , chỉ bao gồm lớp vật lý.

Cám ơn vì sự quan tâm của bạn!