Contents

[1. LWIP là gì 3](#_Toc108710542)

[1.1 Mô hình kết nối giữa vi điều khiển và server 3](#_Toc108710543)

[1.2 So sánh cách giao tiếp AT command và LWIP stack 4](#_Toc108710544)

[2. Porting 5](#_Toc108710545)

[2.1 Khởi tạo project 6](#_Toc108710546)

[2.2 Thêm thư viện LWIP vào project 7](#_Toc108710547)

[2.2 Port các file cần thiết cho nền tảng STM32 11](#_Toc108710548)

[a. lwipopts.h 11](#_Toc108710549)

[b. sys\_timer.c 14](#_Toc108710550)

[c. cc.h và cpu.h 15](#_Toc108710551)

[2.3 Porting các function cần thiết 16](#_Toc108710552)

[a. sys\_rand 16](#_Toc108710553)

[b. Các hàm liên quan đến việc log dữ liệu debug 17](#_Toc108710554)

[c. Hàm gửi dữ liệu qua cổng UART từ LWIP đến module 4G 17](#_Toc108710555)

[d. gsm\_hw\_pppos\_polling và sio\_read 17](#_Toc108710556)

[e. Hàm ngắt nhận dữ liệu UART 18](#_Toc108710557)

[3. Thiết kế chương trình 19](#_Toc108710558)

[3.2 Các hàm ứng dụng 22](#_Toc108710559)

[a. open\_ppp\_stack 22](#_Toc108710560)

[b. ppp\_link\_status\_cb 24](#_Toc108710561)

[c. ppp\_phase\_callback 28](#_Toc108710562)

[4. Thử nghiệm và ứng dụng 29](#_Toc108710563)

[4.1 Lấy IPv4 của thiết bị 29](#_Toc108710564)

[4.2 Thử nghiệm đồng bộ thời gian từ internet 30](#_Toc108710565)

[5. Kết nối HTTP 32](#_Toc108710566)

[6. Kết nối MQTT 36](#_Toc108710567)

[7. Source code 40](#_Toc108710568)

[Phần 2. Cấu hình và triển khai LWIP với ETH trên stm32. 41](#_Toc108710569)

[P2.1. Cấu hình trên cubeMX 41](#_Toc108710570)

[P2.2. Triển khai code và các hàm cần thiết 43](#_Toc108710571)

[P2.2.1. các file include 43](#_Toc108710572)

[P2.2.2 Triển khai DHCP 44](#_Toc108710573)

[P2.3. lwip và Quá trình truyền nhận bản tin qua Ethernet. 51](#_Toc108710574)

[P2.3.1. Cấu trúc khối kết nối Ethernet của STM32 51](#_Toc108710575)

[P2.3.2.1. SMI (Station management interface): Giao diện quản lý trạm 52](#_Toc108710576)

[2.3.2.3. RMII (Reduce Media-independent interface): 53](#_Toc108710577)

[P2.3.3. Giao tiếp với thanh ghi PHY. 54](#_Toc108710578)

[P2.3.2. Quá trình xử lý bản tin nhận được. 54](#_Toc108710579)

[P2.3.3. Quá trình truyền dữ liệu. 56](#_Toc108710580)

[P2.4. Http và lwip. 57](#_Toc108710581)

**Porting LWIP cho vi điều khiển STM32**

# LWIP là gì

* LWIP – Light weight IP là 1 bộ thư viện mã nguồn mở được thiết kế dành cho những hệ thống có tài nguyên tương đối hạn chế, phù hợp với các hệ thống nhúng.
* Hỗ trợ tương đối đầy đủ các giao thức mạnh trên nền TCP/IP
* Có thể hỗ trợ giao tiếp với vi điều khiển thông qua **Serial** hoặc **Ethernet**

Ngoài ra còn rất nhiều định nghĩa, các lí thuyết, các giao thức hỗ trợ, kiến trúc chi tiết của LWIP, bài viết này sẽ không đề cập đến mà tập trung vào việc porting. Phân tích kĩ hơn về LWIP sẽ dành cho các bài viết sau.

## Mô hình kết nối giữa vi điều khiển và server

LWIP hỗ trợ các cách kết nối vật lí chính

* Serial
* Ethernet
* Các chuẩn kết nối khác (WiFi…)

1. Sử dụng serial UART

Text

Description automatically generated

Ở mô hình kết nối này, vi điều khiển sẽ kết nối với modem mạng qua chuẩn giao tiếp nối tiếp UART, phù hợp với nhiều dòng VDK khác nhau, tuy nhiên tốc độ kết nối phụ thuộc nhiều vào tốc độ baudrate của VDK.

1. Sử dụng Ethernet

Text

Description automatically generated with medium confidence

* Với mô hình kết nối này, LWIP sẽ sử dụng Ethernet để kết nối lên server, tốc độ cao hơn Serial. Tuy nhiên VDK cần phải hỗ trợ Phy Ethernet hoặc module chuyển đổi SPI-Ethernet.

Bài viết sẽ tập trung vào cách porting LWIP qua chuẩn giao tiếp serial UART.

## 1.2 So sánh cách giao tiếp AT command và LWIP stack

Về cơ bản có 2 cách kết nối chính từ vi điều khiển lên cloud qua serial với module 4G như sau:

Letter

Description automatically generated with medium confidence

* Ở cách thứ 1 : vi điều khiển dựa hoàn toàn vào tập lệnh của nhà sản xuất module 4G để thực hiện kết nối internet.
* Ở cách số 2 : vi điều khiển dựa vào tập lênh của nhà sản xuất module 4G để khởi tạo, sau đó kết nối internet qua LWIP.
* So sánh ưu - nhược điểm của 2 phương pháp

|  |  |  |  |
| --- | --- | --- | --- |
| Phương pháp | Network stack | Ưu điểm | Nhược điểm |
| AT command (cách 1) | * Phần xử lí các application như TCP, UDP, HTTP… đều nằm trên module 4G. * Chip vi điều khiển chỉ nhận lấy các kết quả cuối cùng của các giao thức bằng các tập lệnh AT-Command | * Tiết kiệm tài nguyên cho vi điều khiển * Các ứng dụng mạng phụ thuộc vào firmware của nhà sản xuất module | * Khi thay đổi module thì cần phải thay đổi tập lệnh của các ứng dụng. * Tiêu tốn ít tài nguyên của vi điều khiển. * Độ ổn định của firmware module 4G cần test lại với mỗi module, mỗi nhà sản xuất khác nhau. * Nhiều giao thức mạng bị hạn chế bởi module của nhà sản xuất, và phụ thuộc và tính năng firmware module của nhà sản xuất. |
| LWIP stack (cách 2) | * Phần xử lí tầng application các giao thức mạng đều nằm hết trên vi điều khiển, module 4G không xử lí các giao thức mạng | * Khả năng port code giữa các nền tảng tương đối dễ dàng. * Kế thừa được nhiều framework base trên LWIP. * Khả năng “portable, reuseable” cao * Dễ dàng chuyển đổi module 4G giữa các nhà cung cấp khác nhau. | * Tiêu tốn tài nguyên của vi điều khiển nhiều hơn so với dùng AT command và network stack của nhà sản xuất. * Một số ứng dụng cần nhiều tài nguyên phần cứng (SSL..) thì đòi hỏi cấu hình RAM của vi điều khiển lớn. * Không chạy được đồng thời data mode và command mode. Phải chuyển qua lại giữa data mode và command mode. |

# Porting

Trong bài viết này, mình sử dụng kit có sẵn cho thuận tiện, với tài nguyên như sau

* + Vi điều khiển STM32L083RZ, cấu hình CPU 32Mhz, 20KB RAM, 192KB Flash. Lưu ý các bạn có thể dùng MCU có flash size nhỏ hơn, nhưng theo khuyến nghị của mình là 64KB Flash.
  + IDE : KeilC (MDK-ARM), optimize code level 0
  + Tool gen code STM32CubeMX
* Khởi tạo cơ bản phần cứng cơ bản
  + Khởi tạo clock
  + Khởi tạo khối USART
  + Generate project
* Mục tiêu demo được của bài viết
  + Port thành công LWIP cho vi điều khiển qua giao tiếp PPP overserial
  + Lấy được IPV4 của thiết bị
  + Thực hiện 1 lệnh đồng bộ thời gian qua internet
* Các bài viết tiếp theo
  + Kết nối MQTT
  + Kết nối HTTP
  + Optimize memory (ram, flash, speed)

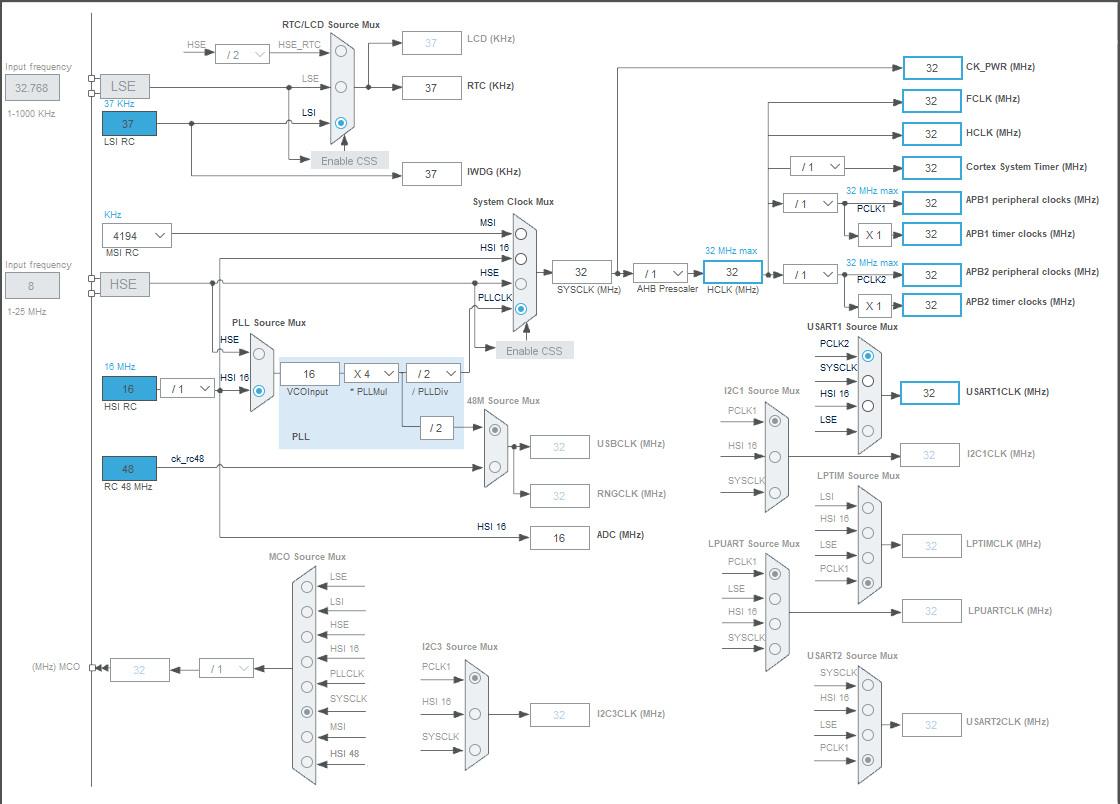
## 2.1 Khởi tạo project

* Trong bài viết này mình sẽ sử dụng bộ giao động nội 32Mhz clock, USART với DMA. Tuy nhiên mình sẽ không đi quá chi tiết vào cách thức khởi tạo và code driver.
* Pinout cơ bản

A picture containing waterfall chart

Description automatically generated

* Clock cơ bản



## 2.2 Thêm thư viện LWIP vào project

* Download source code của LWIP ở đường dẫn <http://download.savannah.nongnu.org/releases/lwip/>, phiên bản mình sử dụng là 2.1.2
* Giải nén vào project

A screenshot of a computer

Description automatically generated

* Cấu trúc folder cần import như sau

Diagram

Description automatically generated with low confidence

* Folder “**porting**” là phần lập trình viên cần code cho nền tảng Vi điều khiển STM32.
* Folder “**app**” chứa các thư viện liên quan đến tầng ứng dụng của LWIP như HTTP, MQTT, MDNS…
* Folder “**core**” và “**ipv4**” chứa thư viện lõi của LWIP.
* Đường dẫn đến header file cần include vào project

Graphical user interface, text, application

Description automatically generated

* Chi tiết các file cần import vào project

Diagram

Description automatically generated

Diagram

Description automatically generated

## Port các file cần thiết cho nền tảng STM32

Text, letter

Description automatically generated

### lwipopts.h

lwipopts.h chứa toàn bộ các cấu hình của LWIP stack.

#ifndef \_\_LWIPOPTS\_H\_\_

#define \_\_LWIPOPTS\_H\_\_

#include <stdint.h>

#include "app\_debug.h"

#define NO\_SYS                      1

#define NO\_SYS\_NO\_TIMERS            0

//#define MEM\_LIBC\_MALLOC             1

//#define MEMP\_MEM\_MALLOC             1

#define MEM\_ALIGNMENT               4

#define MEM\_SIZE                    (1024)

#define PBUF\_POOL\_SIZE              6

#define LWIP\_ARP                    0

#define IP\_REASS\_MAX\_PBUFS          0

#define IP\_FRAG\_USES\_STATIC\_BUF     0

#define MEMP\_NUM\_RAW\_PCB            0

#define MEMP\_NUM\_UDP\_PCB            2

#define MEMP\_NUM\_TCP\_PCB            1

#define MEMP\_NUM\_TCP\_PCB\_LISTEN     1

#define MEMP\_NUM\_TCP\_SEG            8

#define MEMP\_NUM\_FRAG\_PBUF          0

#define MEMP\_NUM\_TCPIP\_MSG\_API      0

#define MEMP\_NUM\_TCPIP\_MSG\_INPKT    0

#define MEMP\_NUM\_SNMP\_NODE          0

#define MEMP\_NUM\_SNMP\_ROOTNODE      0

#define MEMP\_NUM\_SNMP\_VARBIND       0

#define MEMP\_NUM\_SNMP\_VALUE         0

#define IP\_DEFAULT\_TTL              255

#define IP\_SOF\_BROADCAST            0

#define IP\_SOF\_BROADCAST\_RECV       0

#define LWIP\_ICMP                   1

#define LWIP\_BROADCAST\_PING         0

#define LWIP\_MULTICAST\_PING         0

#define LWIP\_RAW                    0

#define TCP\_LISTEN\_BACKLOG          0

#define LWIP\_NETIF\_STATUS\_CALLBACK  1

#define LWIP\_NETIF\_LINK\_CALLBACK    1

#define LWIP\_NETIF\_HWADDRHINT       1

#define LWIP\_NETCONN                !NO\_SYS

#define LWIP\_SOCKET                 !NO\_SYS

#define LWIP\_STATS\_DISPLAY          0

#define MEM\_STATS                   0

#define SYS\_STATS                   0

#define MEMP\_STATS                  0

#define LINK\_STATS                  0

#define ETHARP\_TRUST\_IP\_MAC         0

#define ETH\_PAD\_SIZE                2

#define LWIP\_CHKSUM\_ALGORITHM       2

#define LWIP\_CHECKSUM\_ON\_COPY               1

#define LWIP\_ND6\_MAX\_MULTICAST\_SOLICIT      1

#define LWIP\_ND6\_MAX\_NEIGHBOR\_ADVERTISEMENT 1

#define LWIP\_ND6\_RETRANS\_TIMER              20000

#define LWIP\_ND6\_QUEUEING                   0

#define LWIP\_ND6\_NUM\_ROUTERS                0

#define LWIP\_ND6\_DELAY\_FIRST\_PROBE\_TIME     10000

#define LWIP\_TCP\_KEEPALIVE          1

#define LWIP\_ARP                    0

#define ARP\_TABLE\_SIZE              0

#define IP\_REASSEMBLY               0

#define IP\_FRAG                     0

#define LWIP\_TCP                    1

#define LWIP\_IPV6                   0

#define LWIP\_IPV4                   1

#define LWIP\_ICMP6                  0

#define LWIP\_IPV6\_REASS             0

#define LWIP\_ND6\_TCP\_REACHABILITY\_HINTS 0

#define LWIP\_IPV6\_MLD               0

#define LWIP\_STATS                  0

#define PPP\_IPV6\_SUPPORT            0

#define LWIP\_PPP\_API                0

#define PPP\_SUPPORT                 1

#define LWIP\_DNS                    1

#define LWIP\_SUPPORT\_CUSTOM\_PBUF    1

#define LWIP\_BTLE\_6LOWPAN           0

#define PPP\_NOTIFY\_PHASE            1

#define TCP\_TMR\_INTERVAL            500

#define LWIP\_CALLBACK\_API           1

// Keepalive values, compliant with RFC 1122. Don't change this unless you know what you're doing

#define TCP\_KEEPIDLE\_DEFAULT        10000UL // Default KEEPALIVE timer in milliseconds

#define TCP\_KEEPINTVL\_DEFAULT       2000UL  // Default Time between KEEPALIVE probes in milliseconds

#define TCP\_KEEPCNT\_DEFAULT         9U      // Default Counter for KEEPALIVE probes

#define TCP\_MSS                     536

#define TCP\_WND                     (2 \* TCP\_MSS)

#define TCP\_SND\_BUF                 (2 \* TCP\_MSS)

#define DNS\_TABLE\_SIZE              2

#define DNS\_MAX\_NAME\_LENGTH         128

#define SO\_REUSE                    1

//#define LWIP\_NOASSERT               1

//#define mem\_init(...)

//#define mem\_free(p)                 nrf\_free((p))

//#define mem\_malloc(sz)              nrf\_malloc((sz))

//#define mem\_trim(p,sz)              nrf\_realloc((p),(sz))

#define SNTP\_SERVER\_DNS             1

#define LWIP\_DEBUG                  LWIP\_DBG\_OFF

#define ETHARP\_DEBUG                LWIP\_DBG\_OFF

#define NETIF\_DEBUG                 LWIP\_DBG\_OFF

#define PBUF\_DEBUG                  LWIP\_DBG\_OFF

#define API\_LIB\_DEBUG               LWIP\_DBG\_OFF

#define API\_MSG\_DEBUG               LWIP\_DBG\_OFF

#define SOCKETS\_DEBUG               LWIP\_DBG\_OFF

#define ICMP\_DEBUG                  LWIP\_DBG\_OFF

#define INET\_DEBUG                  LWIP\_DBG\_OFF

#define IP\_DEBUG                    LWIP\_DBG\_OFF

#define IP\_REASS\_DEBUG              LWIP\_DBG\_OFF

#define RAW\_DEBUG                   LWIP\_DBG\_OFF

#define MEM\_DEBUG                   LWIP\_DBG\_OFF

#define MEMP\_DEBUG                  LWIP\_DBG\_OFF

#define SYS\_DEBUG                   LWIP\_DBG\_OFF

#define TCP\_DEBUG                   LWIP\_DBG\_OFF

#define TCP\_INPUT\_DEBUG             LWIP\_DBG\_OFF

#define TCP\_OUTPUT\_DEBUG            LWIP\_DBG\_OFF

#define TCP\_RTO\_DEBUG               LWIP\_DBG\_OFF

#define TCP\_CWND\_DEBUG              LWIP\_DBG\_OFF

#define TCP\_WND\_DEBUG               LWIP\_DBG\_OFF

#define TCP\_FR\_DEBUG                LWIP\_DBG\_OFF

#define TCP\_QLEN\_DEBUG              LWIP\_DBG\_OFF

#define TCP\_RST\_DEBUG               LWIP\_DBG\_OFF

#define UDP\_DEBUG                   LWIP\_DBG\_OFF

#define TCPIP\_DEBUG                 LWIP\_DBG\_OFF

#define PPP\_DEBUG                   LWIP\_DBG\_OFF

#define SLIP\_DEBUG                  LWIP\_DBG\_OFF

#define DHCP\_DEBUG                  LWIP\_DBG\_OFF

#define SYS\_LIGHTWEIGHT\_PROT        0

#define HTTPC\_DEBUG                 LWIP\_DBG\_OFF

#define HTTPC\_DEBUG\_REQUEST         LWIP\_DBG\_OFF

#define HTTPC\_CLIENT\_AGENT          "huytv"

extern void lwip\_sntp\_recv\_callback(uint32\_t sec);

#define SNTP\_SET\_SYSTEM\_TIME        lwip\_sntp\_recv\_callback

#endif /\* \_\_LWIPOPTS\_H\_\_ \*/

### sys\_timer.c

Có nhiệm vụ cung cấp các timer để định thời, timeout cho internal timer của LWIP. Trong đó bắt buộc cần có 2 hàm trả về thời gian hiện tại của hệ thống là “**sys\_now**” và “**sys\_jiffies**” với đơn vị millisecond, các bạn có thể thay thế hàm “**sys\_get\_tick\_ms**” bằng hàm “**HAL\_GetTick**” của STM32.

/\*\*

 \* @brief Override LWIP weak function

 \*/

uint32\_t sys\_now(void)

{

    return sys\_get\_tick\_ms();

}

/\*\*

 \* @brief Override LWIP weak function

 \*/

uint32\_t sys\_jiffies(void)

{

    return sys\_get\_tick\_ms();

}

uint32\_t sys\_get\_tick\_ms(void)

{

    return HAL\_GetTick();

}

### cc.h và cpu.h

Chứa các định nghĩa về kiến trúc MCU đang sử dụng.

#ifndef \_\_CC\_H\_\_

#define \_\_CC\_H\_\_

#include "cpu.h"

#include <stdlib.h>

#include <stdio.h>

#include "app\_debug.h"

typedef int32\_t sys\_prot\_t;

#define LWIP\_PROVIDE\_ERRNO

#if defined (\_\_GNUC\_\_) & !defined (\_\_CC\_ARM)

#define LWIP\_TIMEVAL\_PRIVATE 0

#include <sys/time.h>

#endif

/\* define compiler specific symbols \*/

#if defined (\_\_ICCARM\_\_)

#define PACK\_STRUCT\_BEGIN

#define PACK\_STRUCT\_STRUCT

#define PACK\_STRUCT\_END

#define PACK\_STRUCT\_FIELD(x) x

#define PACK\_STRUCT\_USE\_INCLUDES

#elif defined (\_\_GNUC\_\_)

#define PACK\_STRUCT\_BEGIN

#define PACK\_STRUCT\_STRUCT \_\_attribute\_\_ ((\_\_packed\_\_))

#define PACK\_STRUCT\_END

#define PACK\_STRUCT\_FIELD(x) x

#elif defined (\_\_CC\_ARM)

#define PACK\_STRUCT\_BEGIN \_\_packed

#define PACK\_STRUCT\_STRUCT

#define PACK\_STRUCT\_END

#define PACK\_STRUCT\_FIELD(x) x

#elif defined (\_\_TASKING\_\_)

#define PACK\_STRUCT\_BEGIN

#define PACK\_STRUCT\_STRUCT

#define PACK\_STRUCT\_END

#define PACK\_STRUCT\_FIELD(x) x

#endif

#ifndef LWIP\_PLATFORM\_ASSERT

#define LWIP\_PLATFORM\_ASSERT(x) do {DEBUG\_RAW("Assertion \"%s\" failed at line %d in %s\r\n", \

                                     x, \_\_LINE\_\_, \_\_FILE\_\_); } while(0)

#define LWIP\_PLATFORM\_DIAG(x) do {DEBUG\_RAW x;} while(0)

#endif /\* LWIP\_PLATFORM\_ASSERT \*/

/\* Define random number generator function \*/

extern uint32\_t sys\_rand(void);

#define LWIP\_RAND() ((u32\_t)sys\_rand())

#endif /\* \_\_CC\_H\_\_ \*/

## Porting các function cần thiết

### sys\_rand

Dùng để ramdom 1 số uint32\_t, để cho đơn giản mình sẽ lấy systick value của vi điều khiển lỗi ARM

uint32\_t sys\_rand()

{

    return SysTick->VAL;

}

### Các hàm liên quan đến việc log dữ liệu debug

Để phục vụ in dữ liệu qua màn hình, các bạn có thể **retarget printf** qua **UART**, tuy nhiên ở ví dụ này, mình sử dụng bộ thư việc “**Segger\_RTT**”, ưu điểm của bộ thư viện này là chỉ cần 2 dây debug “SWD” để in dữ liệu. Các bạn có thể tìm hiểu thêm về bộ thư viện này ở đây “**https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/”,** tuy nhiên ở bài viết này mình sẽ không đề cập chi tiết.

### Hàm gửi dữ liệu qua cổng UART từ LWIP đến module 4G

“**ppp\_output\_callback**” gửi data qua cổng “USART1”, với 2 tham số cơ bản là “**data**” và “**len**”, lần lượt là dữ liệu và độ dài dữ liệu LWIP muốn đẩy tới module 4G.

Text

Description automatically generated

### gsm\_hw\_pppos\_polling và sio\_read

* Định kì nhận dữ liệu từ cổng UART của module sim, truyền vào LWIP stack. Đồng thời polling các timer của LWIP, hàm này cần gọi liên tục vô hạn trong vòng lặp while(1) của hệ thống

uint32\_t sio\_read(sio\_fd\_t fd, u8\_t \*data, u32\_t len)

{

    return gsm\_hardware\_layer\_copy\_ppp\_buffer(data, len);

}

static uint8\_t m\_ppp\_rx\_buffer[512];

void gsm\_hw\_pppos\_polling(void)

{

    uint32\_t sio\_size;

    sys\_check\_timeouts();

    sio\_size = sio\_read(0, m\_ppp\_rx\_buffer, 512);

    if (sio\_size > 0)

    {

        // Bypass data into ppp stack

        pppos\_input(gsm\_data\_layer\_get\_ppp\_control\_block(), m\_ppp\_rx\_buffer, sio\_size);

    }

}

* Khi chuyển sang chế độ PPP mode, mọi dữ liệu từ cổng UART sẽ chuyển vào PPP buffer, chúng ta cần phân biệt uart rx data đó dành cho AT command hay PPP stack, ý tưởng của mình đơn giản như sau:

A picture containing text, businesscard

Description automatically generated

### Hàm ngắt nhận dữ liệu UART

Hàm nhận dữ lịêu ngắt uart rồi lưu vào PPP buffer, hoặc AT command buffer. Đây là 1 cách implement đơn giản của mình.

uint32\_t prev\_index = 0;

// Ham ngat nhan du lieu tu cong UART

void gsm\_hw\_layer\_uart\_fill\_rx(uint8\_t \*data, uint32\_t length)

{

    if (length)

    {

        m\_new\_uart\_data = true;

        // If device in is data mode =>> bypass data into PPP stack

        if (gsm\_is\_in\_ppp\_mode())

        {

            for (uint32\_t i = 0; i < length; i++)

            {

                m\_gsm\_modem\_buffer.buffer[m\_gsm\_modem\_buffer.idx\_in++] = data[i];

                if (m\_gsm\_modem\_buffer.idx\_in >= GSM\_PPP\_MODEM\_BUFFER\_SIZE)

                {

                    m\_gsm\_modem\_buffer.idx\_in = 0;

                    DEBUG\_ERROR("GSM PPP RX overflow\r\n");

                }

                m\_gsm\_modem\_buffer.buffer[m\_gsm\_modem\_buffer.idx\_in] = 0;

            }

        }

        Else // Command mode, bypass data into AT command buffer

        {

            prev\_index = m\_gsm\_atc.atc.recv\_buff.index;

            for (uint32\_t i = 0; i < length; i++)

            {

                m\_gsm\_atc.atc.recv\_buff.buffer[m\_gsm\_atc.atc.recv\_buff.index++] = data[i];

                if (m\_gsm\_atc.atc.recv\_buff.index >= sizeof(((gsm\_atc\_buffer\_t\*)0)->buffer))

                {

                    DEBUG\_ERROR("GSM ATC RX overflow\r\n");

                    m\_gsm\_atc.atc.recv\_buff.index = 0;

                    m\_gsm\_atc.atc.recv\_buff.buffer[0] = 0;

                    return;

                }

            }

        }

    }

}

Trong đó **“uint32\_t sio\_read(sio\_fd\_t fd, u8\_t \*data, u32\_t len)**”trả về số byte UART-RX mà vi điều khiển nhận về từ module 4G, cần truyền lượng data này vào lwip stack, với tham số truyền vào là buffer data và độ dài dữ liệu mong muốn của LWIP. Dữ liệu được đưa tiếp vào LWIP stack bằng hàm “**pppos\_input**”.

Vậy là các bạn đã có cơ bản các hàm driver cần thiết cho LWIP, chúng ta sẽ đi vào thiết kế chương trình và porting ứng dụng cho LWIP.

# Thiết kế chương trình

Về cơ bản chu trình sẽ như sau

* Khởi tạo phần cứng
* Khởi tạo GSM, đăng kí cho module SIM register được vào nhà mạng Viettel…
* Active kết nối

Diagram

Description automatically generated3.1 Main

Mình sẽ khởi tạo LWIP và DNS qua hàm “**dns\_initialize()”** và “**lwip\_init()”.** Trong đó dns\_initialize dành để phân giả tên miền ra thành địa chỉ IP, **lwip\_init()** dùng để khởi tạo core của LWIP.

int main(void)

{

  /\* USER CODE BEGIN 1 \*/

  /\* USER CODE END 1 \*/

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

  /\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

  HAL\_Init();

  /\* USER CODE BEGIN Init \*/

  /\* USER CODE END Init \*/

  /\* Configure the system clock \*/

  SystemClock\_Config();

  /\* USER CODE BEGIN SysInit \*/

  /\* USER CODE END SysInit \*/

  /\* Initialize all configured peripherals \*/

  MX\_GPIO\_Init();

  MX\_DMA\_Init();

  MX\_USART1\_UART\_Init();

  /\* USER CODE BEGIN 2 \*/

    dns\_initialize();

    lwip\_init();

    DEBUG\_INFO("Application started\r\n");

    gsm\_init\_hw();

  /\* USER CODE END 2 \*/

  /\* Infinite loop \*/

  /\* USER CODE BEGIN WHILE \*/

  while (1)

  {

    /\* USER CODE END WHILE \*/

    /\* USER CODE BEGIN 3 \*/

    gsm\_mnr\_task(NULL);

  }

  /\* USER CODE END 3 \*/

}

static void dns\_initialize(void)

{

// 8.8.8.8 và 1.1.1.1

    ip\_addr\_t dns\_server\_0 = IPADDR4\_INIT\_BYTES(8, 8, 8, 8);

    ip\_addr\_t dns\_server\_1 = IPADDR4\_INIT\_BYTES(1, 1, 1, 1);

    dns\_setserver(0, &dns\_server\_0);

    dns\_setserver(1, &dns\_server\_1);

    dns\_init();

}

* Vòng lặp quét các trạng thái GSM

void gsm\_mnr\_task(void \*arg)

{

    gsm\_hw\_layer\_run(); // Hỏi vòng phần hardward uart, at command, rx data…

    gsm\_state\_machine\_polling(); // Hỏi vòng các state machine của GSM

    gsm\_hw\_pppos\_polling(); // Hỏi vòng các công việc liên quan đến timer và serial data của LWIP

}

## Các hàm ứng dụng

Sau khi khởi tạo xong module 4G chúng ta cần chuyển chế độ hoạt động của module từ “**command mode”** sang “**data mode**” bằng cách gọi hàm “**open\_ppp\_stack**”, lệnh cần quan tâm chính để đưa vào chế độ data mode là “**ATD\*99\*\*\*1#\r\n**” và chờ phản hồi “**CONNECT**” từ module SIM. Về cơ bản tập lệnh này giống như giữ các nhà sản xuất module SIM.

### open\_ppp\_stack

gsm\_hw\_send\_at\_cmd("ATV1\r\n", "OK\r\n", "", 1000, 3, open\_ppp\_stack);

static void open\_ppp\_stack(gsm\_response\_event\_t event, void \*response\_buffer, void \*data)

{

    DEBUG\_INFO("Open PPP stack\r\n");

    static uint8\_t retry\_count = 0;

    switch (m\_gsm\_manager.step)

    {

    case 1:

    {

        gsm\_hw\_send\_at\_cmd(AT\_CSQ, AT\_OK, AT\_NULL, 1000, 2, open\_ppp\_stack);

        m\_gsm\_manager.step = 2;

    }

    break;

    case 2:

    {

        //Check SIM inserted, if removed -> RESET module NOW!

        gsm\_hw\_send\_at\_cmd(AT\_CPIN, AT\_OK, AT\_NULL, 1000, 5, open\_ppp\_stack);

        m\_gsm\_manager.step = 3;

    }

    break;

    case 3:

    {

        if (event == GSM\_EVENT\_OK)

        {

            if (strstr(response\_buffer, GSM\_SIM\_NOT\_INSERT))

            {

                DEBUG\_INFO("Sim card not inserted\r\n");

                gsm\_change\_state(GSM\_STATE\_RESET);

                return;

            }

// Chuyển từ command mode sang data mode

            gsm\_hw\_send\_at\_cmd("ATD\*99\*\*\*1#\r\n",

                               "CONNECT",

                               AT\_NULL,

                               1000,

                               10,

                               open\_ppp\_stack);

            m\_gsm\_manager.step = 4;

        }

        else

        {

            DEBUG\_INFO("Open ppp stack failed\r\n");

            gsm\_change\_state(GSM\_STATE\_RESET);

            return;

        }

    }

    break;

    case 4:

    {

        DEBUG\_INFO("PPP state: %s\r\n", (event == GSM\_EVENT\_OK) ? "[OK]" : "[FAIL]");

// Đã chuyển sang ppp mode thành công, từ đây mọi data nhận từ module 4G sẽ truyền thằng vào LWIP //stack

        m\_gsm\_manager.mode = GSM\_PPP\_MODE;

        if (event != GSM\_EVENT\_OK)

        {

            retry\_count++;

            if (retry\_count > 4)

            {

                retry\_count = 0;

                ppp\_close(m\_ppp\_control\_block, 0);

                /\* Reset GSM \*/

                gsm\_change\_state(GSM\_STATE\_RESET);

            }

            else

            {

                m\_gsm\_manager.step = 3;

                ppp\_close(m\_ppp\_control\_block, 0);

                gsm\_hw\_send\_at\_cmd(ATV1, AT\_OK, AT\_NULL, 1000, 5, open\_ppp\_stack);

            }

        }

        else

        {

            gsm\_change\_state(GSM\_OK);

            //Create PPP connection

            m\_ppp\_control\_block = pppos\_create(&m\_ppp\_netif, ppp\_output\_callback, ppp\_link\_status\_cb, NULL);

            if (m\_ppp\_control\_block == NULL)

            {

                DEBUG\_ERROR("Create PPP interface ERR!\r\n");

                //assert(0);

NVIC\_SystemReset();

            }

            /\* Set this interface as default route \*/

            ppp\_set\_default(m\_ppp\_control\_block);

            //ppp\_set\_auth(m\_ppp\_control\_block, PPPAUTHTYPE\_CHAP, "", "");

            ppp\_set\_notify\_phase\_callback(m\_ppp\_control\_block, ppp\_notify\_phase\_cb);

            ppp\_connect(m\_ppp\_control\_block, 0);

        }

    }

    break;

default:

break;

    }

}

* Khi khởi tạo PPP connection, cần lưu ý cấp phát PPP control block

    // Create PPP connection

    m\_ppp\_control\_block = pppos\_create(&m\_ppp\_netif, ppp\_output\_callback, ppp\_link\_status\_cb, NULL);

    if (m\_ppp\_control\_block == NULL)

    {

     DEBUG\_ERROR("Create PPP interface ERR!\r\n");

     assert(0);

    }

    /\* Set this interface as default route \*/

    ppp\_set\_default(m\_ppp\_control\_block);

    //ppp\_set\_auth(m\_ppp\_control\_block, PPPAUTHTYPE\_CHAP, "", "");

    ppp\_set\_notify\_phase\_callback(m\_ppp\_control\_block, ppp\_notify\_phase\_cb);

    ppp\_connect(m\_ppp\_control\_block, 0);

### ppp\_link\_status\_cb

Dùng để nhận các trạng thái status, pha của PPP

/\*\*

 \* PPP status callback

 \* ===================

 \*

 \* PPP status callback is called on PPP status change (up, down, ...) from lwIP core thread

 \*/

static void ppp\_link\_status\_cb(ppp\_pcb \*pcb, int err\_code, void \*ctx)

{

    struct netif \*pppif = ppp\_netif(pcb);

    LWIP\_UNUSED\_ARG(ctx);

    switch (err\_code)

    {

        case PPPERR\_NONE:

        {

    #if LWIP\_DNS

        const ip\_addr\_t \*ns;

    #endif /\* LWIP\_DNS \*/

            DEBUG\_INFO("PPP Connected\r\n");

#if PPP\_IPV4\_SUPPORT

            DEBUG\_INFO("our\_ipaddr  = %s\r\n", ipaddr\_ntoa(&pppif->ip\_addr));

            DEBUG\_INFO("his\_ipaddr  = %s\r\n", ipaddr\_ntoa(&pppif->gw));

            DEBUG\_INFO("netmask    = %s\r\n", ipaddr\_ntoa(&pppif->netmask));

#if LWIP\_DNS

                    ns = dns\_getserver(0);

                    DEBUG\_INFO("\tdns1        = %s\r\n", ipaddr\_ntoa(ns));

                    ns = dns\_getserver(1);

                    DEBUG\_INFO("\tdns2        = %s\r\n", ipaddr\_ntoa(ns));

#endif /\* LWIP\_DNS \*/

#endif /\* PPP\_IPV4\_SUPPORT \*/

#if PPP\_IPV6\_SUPPORT

            DEBUG\_INFO("\r   our6\_ipaddr = %s\n", ip6addr\_ntoa(netif\_ip6\_addr(pppif, 0)));

#endif /\* PPP\_IPV6\_SUPPORT \*/

            break;

        }

        case PPPERR\_PARAM:

        {

            DEBUG\_INFO("status\_cb: Invalid parameter\r\n");

            break;

        }

        case PPPERR\_OPEN:

        {

            DEBUG\_INFO("status\_cb: Unable to open PPP session\r\n");

            break;

        }

        case PPPERR\_DEVICE:

        {

            DEBUG\_INFO("status\_cb: Invalid I/O device for PPP\r\n");

            break;

        }

        case PPPERR\_ALLOC:

        {

            DEBUG\_INFO("status\_cb: Unable to allocate resources\r\n");

            break;

        }

        case PPPERR\_USER: /\* 5 \*/

        {

            /\* ppp\_close() was previously called, reconnect \*/

            DEBUG\_INFO("status\_cb: ppp is closed by user OK! Try to re-open...\r\n");

            /\* ppp\_free(); -- can be called here \*/

            ppp\_free(m\_ppp\_control\_block);

            gsm\_change\_state(GSM\_REOPEN\_PPP);

            break;

        }

        case PPPERR\_CONNECT: /\* 6 \*/

        {

            DEBUG\_INFO("status\_cb: Connection lost\r\n");

            m\_ppp\_connected = false;

            ppp\_close(m\_ppp\_control\_block, 1);

            break;

        }

        case PPPERR\_AUTHFAIL:

        {

            DEBUG\_INFO("status\_cb: Failed authentication challenge\r\n");

            break;

        }

        case PPPERR\_PROTOCOL:

        {

            DEBUG\_INFO("status\_cb: Failed to meet protocol\n");

            break;

        }

        case PPPERR\_PEERDEAD:

        {

            DEBUG\_INFO("status\_cb: Connection timeout\r\n");

            break;

        }

        case PPPERR\_IDLETIMEOUT:

        {

            DEBUG\_INFO("status\_cb: Idle Timeout\r\n");

            break;

        }

        case PPPERR\_CONNECTTIME:

        {

            DEBUG\_INFO("status\_cb: Max connect time reached\r\n");

            break;

        }

        case PPPERR\_LOOPBACK:

        {

            DEBUG\_INFO("status\_cb: Loopback detected\r\n");

            break;

        }

        default:

        {

            DEBUG\_INFO("status\_cb: Unknown error code %d\r\n", err\_code);

            break;

        }

    }

    /\*

    \* This should be in the switch case, this is put outside of the switch

    \* case for example readability.

    \*/

    if (err\_code == PPPERR\_NONE)

    {

        DEBUG\_INFO("PPP is opened OK\r\n!");

        return;

    }

    //  /\* ppp\_close() was previously called, don't reconnect \*/

    //  if (err\_code == PPPERR\_USER) {

    //    /\* ppp\_free(); -- can be called here \*/

    //   m\_ppp\_connected = false;

    //   ppp\_free(m\_ppp\_control\_block);

    //   DEBUG\_INFO("\r PPP opened ERR!");

    //    return;

    //  }

    /\*

   \* Try to reconnect in 30 seconds, if you need a modem chatscript you have

   \* to do a much better signaling here ;-)

   \*/

    //  ppp\_connect(pcb, 30);

    /\* OR ppp\_listen(pcb); \*/

}

Trong trường hợp kết nối thành công, status của PPP callback sẽ nhảy vào trạng thái “**PPPERR\_NONE**”, đến bước này bạn đã **kết nối thành công và sẵn sàng kết nối internet.**

        case PPPERR\_NONE:

        {

    #if LWIP\_DNS

        const ip\_addr\_t \*ns;

    #endif /\* LWIP\_DNS \*/

            DEBUG\_INFO("PPP Connected\r\n");

#if PPP\_IPV4\_SUPPORT

// In địa chỉ IP của thiết bị

            DEBUG\_INFO("our\_ipaddr  = %s\r\n", ipaddr\_ntoa(&pppif->ip\_addr));

            DEBUG\_INFO("his\_ipaddr  = %s\r\n", ipaddr\_ntoa(&pppif->gw));

            DEBUG\_INFO("netmask    = %s\r\n", ipaddr\_ntoa(&pppif->netmask));

#if LWIP\_DNS

                    ns = dns\_getserver(0);

                    DEBUG\_INFO("\tdns1        = %s\r\n", ipaddr\_ntoa(ns));

                    ns = dns\_getserver(1);

                    DEBUG\_INFO("\tdns2        = %s\r\n", ipaddr\_ntoa(ns));

#endif /\* LWIP\_DNS \*/

#endif /\* PPP\_IPV4\_SUPPORT \*/

#if PPP\_IPV6\_SUPPORT

            DEBUG\_INFO("\r   our6\_ipaddr = %s\n", ip6addr\_ntoa(netif\_ip6\_addr(pppif, 0)));

#endif /\* PPP\_IPV6\_SUPPORT \*/

            break;

        }

### ppp\_phase\_callback

Nhận về pha của PPP và xử lí các trạng thái reconnect

static void ppp\_notify\_phase\_cb(ppp\_pcb \*pcb, u8\_t phase, void \*ctx)

{

    switch (phase)

    {

        /\* Session is down (either permanently or briefly) \*/

        case PPP\_PHASE\_DEAD:

            DEBUG\_INFO("PPP\_PHASE\_DEAD\r\n");

            m\_gsm\_manager.ppp\_phase = PPP\_PHASE\_DEAD;

            break;

        /\* We are between two sessions \*/

        case PPP\_PHASE\_HOLDOFF:

            DEBUG\_INFO("PPP\_PHASE\_HOLDOFF\r\n");

            m\_gsm\_manager.ppp\_phase = PPP\_PHASE\_HOLDOFF;

            break;

        /\* Session just started \*/

        case PPP\_PHASE\_INITIALIZE:

            DEBUG\_INFO("PPP\_PHASE\_INITIALIZE\r\n");

            m\_gsm\_manager.ppp\_phase = PPP\_PHASE\_INITIALIZE;

            break;

        case PPP\_PHASE\_NETWORK:

            DEBUG\_INFO("PPP\_PHASE\_NETWORK\r\n");

            break;

        case PPP\_PHASE\_ESTABLISH:

            DEBUG\_INFO("PPP\_PHASE\_ESTABLISH\r\n");

            break;

        /\* Session is running \*/

        case PPP\_PHASE\_RUNNING:

            DEBUG\_INFO("PPP\_PHASE\_RUNNING\r\n");

            m\_gsm\_manager.ppp\_phase = PPP\_PHASE\_RUNNING;

            m\_ppp\_connected = true;

            break;

        case PPP\_PHASE\_TERMINATE:

            DEBUG\_INFO("PPP\_PHASE\_TERMINATE\r\n");

            break;

        case PPP\_PHASE\_DISCONNECT:

            DEBUG\_INFO("PPP\_PHASE\_DISCONNECT\r\n");

            break;

        default:

            DEBUG\_INFO("Unknown PPP phase %d\r\n", phase);

            break;

    }

}

# Thử nghiệm và ứng dụng

## 4.1 Lấy IPv4 của thiết bị

* Kết quả thử nghiệm, thiết bị đã kết nối thành công vào internet và lấy được IPv4

Text

Description automatically generated

## 4.2 Thử nghiệm đồng bộ thời gian từ internet

Khởi tạo dịch vụ đồng bộ thời gian, ở đây mình sẽ lấy internet từ NTP server “**pool.ntp.org**"

static void initialize\_stnp(void)

{

    static bool sntp\_start = false;

    if (sntp\_start == false)

    {

        sntp\_start = true;

        DEBUG\_INFO("Initialize stnp\r\n");

        sntp\_setoperatingmode(SNTP\_OPMODE\_POLL);

        sntp\_setservername(0, "pool.ntp.org");

        sntp\_init();

    }

}

Khai báo hàm callback đồng bộ thời gian trong file “**lwipopts.h**”

Graphical user interface, text

Description automatically generated

* Xử lí callback data của dịch vụ SNTP

void lwip\_sntp\_recv\_callback(uint32\_t time)

{

    if (time == 0)

    {

        DEBUG\_WARN("NTP: Error, server not responding or bad response\r\n");

    }

    else

    {

        DEBUG\_INFO("NTP: %u seconds elapsed since 1.1.1970\r\n", time);

    }

}

* Kết quả

Text

Description automatically generated

# Kết nối HTTP

Ở phần tiếp theo này, mình sẽ thử nghiệm thực hiện 1 lệnh http get đến đường dẫn <http://httpbin.org/get>

* Thiết kế chương trình

Ý tưởng của mình như sau:

* + - Chờ cho PPP đã được kết nối.
    - Nếu đã kết nối thì truyền các thông số của HTTP URL vào hàm cấu hình.
    - Đăng kí các hàm callback nhận dữ liệu.
    - Thực hiện lệnh HTTP Get.

Lưu đồ thuật toán

Diagram

Description automatically generated with low confidence

* Trong đó http\_connection\_t chứa các thông tin mà server sẽ kết nối đến, chứa các thông tin

|  |  |  |
| --- | --- | --- |
| http\_connection\_t | proxy\_addr | Địa chỉ của server, có kiểu số nguyên dương 32 bit |
| proxy\_port | HTTP port (80,443…) |
| use\_proxy | 1 hoặc 0 |
| httpc\_result\_fn | Hàm nhận dữ liệu trả về của http |
| httpc\_headers\_done\_fn | Hàm callback khi socket nhận đủ dữ liệu của header |
| http\_state\_t |  | Chưa toàn bộ internal state-machine, timeout, bộ đệm… của http |

* Implement hàm xử lí khi nhận đủ header trong HTTP

err\_t httpc\_headers\_done\_callback(httpc\_state\_t \*connection, void \*arg, struct pbuf \*hdr, u16\_t hdr\_len, u32\_t content\_len)

{

    DEBUG\_INFO("httpc\_headers\_callback, content length %d\r\n", content\_len);

    if (content\_len == 0xFFFFFFFF)

    {

        /\*

            Content length không hợp lệ, có thể có những lí do sau

            - Trong header server trả về không có trường "Content-Length"

            - Server trả về dạng stream data, dữ liệu kết thúc khi server đóng kết nối

            - Các lí do khác

        \*/

        DEBUG\_INFO("Invalid content length\r\n");

    }

    else

    {

        DEBUG\_INFO("HTTP content length %u bytes\r\n", content\_len);

    }

    return ERR\_OK;

}

* Hàm nhận status của kết nối HTTP

static void httpc\_result\_callback(void \*arg, httpc\_result\_t httpc\_result, u32\_t rx\_content\_len, u32\_t srv\_res, err\_t err)

{

    DEBUG\_INFO("result: %d, content len: %d, status code: %d\r\n", httpc\_result, rx\_content\_len, srv\_res);

    switch (err)

    {

        case HTTPC\_RESULT\_OK: /\*\* File successfully received \*/

        {

            DEBUG\_INFO("HTTPC\_RESULT\_OK\r\n");

        }

            break;

        case HTTPC\_RESULT\_ERR\_UNKNOWN:     /\*\* Unknown error \*/

                                           //break;

        case HTTPC\_RESULT\_ERR\_CONNECT:     /\*\* Connection to server failed \*/

                                           //break;

        case HTTPC\_RESULT\_ERR\_HOSTNAME:    /\*\* Failed to resolve server hostname \*/

                                           //break;

        case HTTPC\_RESULT\_ERR\_CLOSED:      /\*\* Connection unexpectedly closed by remote server \*/

                                           //break;

        case HTTPC\_RESULT\_ERR\_TIMEOUT:     /\*\* Connection timed out (server didn't respond in time) \*/

                                           //break;

        case HTTPC\_RESULT\_ERR\_SVR\_RESP:    /\*\* Server responded with an error code \*/

                                           //break;

        case HTTPC\_RESULT\_ERR\_MEM:         /\*\* Local memory error \*/

                                           //break;

        case HTTPC\_RESULT\_LOCAL\_ABORT:     /\*\* Local abort \*/

                                           //break;

        case HTTPC\_RESULT\_ERR\_CONTENT\_LEN: /\*\* Content length mismatch \*/

            DEBUG\_ERROR("Error content length\r\n");

            break;

        default:

            DEBUG\_INFO("httpc\_result\_callback error %d\r\n", err);

            break;

    }

}

* Hàm cuối cùng chúng ta cần implent để nhận toàn độ data từ HTTP, hàm này chỉ được gọi khi thiết bị kết nối thành công và bắt đầu nhận dữ liệu. Khi nhận data và xử lí xong, hệ thống cần giải phóng memory cho http bằng hàm “**pbuf\_free(p)”.** Trong trường hợp có lỗi xảy xa, cần phải đóng kết nối bằng cách gọi hàm “**tcp\_close**”
* Dữ liệu nhận về có độ dài “**q->len**”, nằm tại địa chỉ “**q->payload**”, ở demo này mình sẽ in dữ liệu nhận về qua cổng debug

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

{

    // Nếu có dữ liệu đến, kiểm tra buffer

    // Nếu buffer không hợp lệ (NULL) thì đóng kết nối TCP

    if (p)

    {

        struct pbuf \*q;

        for (q = p; q; q = q->next)

        {

            // Đây là dữ liệu nhận về, có độ dài bằng q->len, và data là q->payload

            DEBUG\_INFO("HTTP data %.\*s\r\n", q->len, q->payload);

        }

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

        pbuf\_free(p);

    }

    else

    {

        DEBUG\_WARN("tcp\_close\r\n");

        tcp\_close(tpcb);

        return ERR\_ABRT;

    }

    return ERR\_OK;

}

Vậy là các hàm callback xử lí dữ liệu đã xong, chúng ta khởi tạo tham số và tiến hành kết nối. Giả sử bạn cần kết nối đến “<http://httpbin.org/get>” thì các tham số truyền vào **httpc\_get\_file\_dns** như sau

* Server name “**httpbin.org**”
* Port **80**
* File **“/get”**

    /\* Init Http connection params \*/

    m\_conn\_settings\_try.use\_proxy = 0;

    m\_conn\_settings\_try.headers\_done\_fn = httpc\_headers\_done\_callback;

    m\_conn\_settings\_try.result\_fn = httpc\_result\_callback;

    DEBUG\_INFO("HTTP url %s%s, port %d\r\n", m\_http\_cfg.url, m\_http\_cfg.file, m\_http\_cfg.port);

    // Kết nối HTTP

    err\_t error = httpc\_get\_file\_dns((const char\*)m\_http\_cfg.url,

                                    m\_http\_cfg.port,

                                    m\_http\_cfg.file,

                                    &m\_conn\_settings\_try,

                                    httpc\_file\_recv\_callback,

                                    NULL,

                                    &m\_http\_connection\_state);

    m\_conn\_settings\_try.headers\_done\_fn = httpc\_headers\_done\_callback;

    m\_conn\_settings\_try.result\_fn = httpc\_result\_callback;

    if (error != ERR\_OK)

    {

        DEBUG\_INFO("Cannot connect HTTP server, error %d\r\n", error);

        return false;

    }

- Kết quả thử nghiệm

A screenshot of a computer

Description automatically generated with low confidence

# Kết nối MQTT

Design state machine

Diagram

Description automatically generated

- Ý tưởng của mình như sau

* Kiểm tra xem PPP stack đã được kết nối hay chưa.
* Nếu PPP đã kết nối thành công thì khởi tạo các tham số của mqtt như broker address, port…
* Thực hiện phân giải domain name của mqtt broker sang địa chỉ IP.
* Sau khi phân giải IP thành công thì thực hiện kết nối MQTT, gán trạng thái mqtt thành “**Connecting**”.
* Nếu kết nối thành công thì gán trạng thái MQTT thành “**Connected**”, sau đó tiến hành subscribe vào các topic.
* Nếu trạng thái MQTT bằng “**Connected**” thì định kì gửi bản tin ping, định kì resubcribe lại các topic cần thiết với chu kì đã biết trước.

- Các hàm callback của mqtt

* Hàm phân giải DNS **dns\_gethostbyname** với 2 tham số quan trọng truyền vào là broker name và hàm xử lí callback “**mqtt\_dns\_found**”

err\_t err = dns\_gethostbyname(m\_cfg.broker\_addr,

                                &m\_mqtt\_server\_address,

                                mqtt\_dns\_found,

                                NULL);

if (err == ERR\_INPROGRESS)

{

    /\* DNS request sent, wait for sntp\_dns\_found being called \*/

    DEBUG\_INFO("sntp\_request: %d - Waiting for server %s address to be resolved\r\n",

                err,

                m\_cfg.broker\_addr);

}

else if (err == ERR\_OK)

{

    DEBUG\_INFO("DNS resolved aready, host %s, mqtt\_ipaddr = %s\r\n",

                m\_cfg.broker\_addr,

                ipaddr\_ntoa(&m\_mqtt\_server\_address));

    m\_is\_dns\_resolved = 1;

}

- Hàm callback phân giải dns (thành công hoặc thất bại), với kết quả nằm tại con trỏ “**ipaddr**”. Trong trường hợp con trỏ **ipaddr**” khác **null** thì thiết bị đã phân giải dns thành công.

/\*\*

 \* @brief DNS found callback when using DNS names as server address.

 \*/

static void mqtt\_dns\_found(const char \*hostname, const ip\_addr\_t \*ipaddr, void \*arg)

{

    DEBUG\_INFO("mqtt\_dns\_found: %s\r\n", hostname);

    LWIP\_UNUSED\_ARG(hostname);

    LWIP\_UNUSED\_ARG(arg);

    if (ipaddr != NULL)

    {

        /\* Address resolved, send request \*/

        m\_mqtt\_server\_address.addr = ipaddr->addr;

        DEBUG\_INFO("Server address resolved = %s\r\n", ipaddr\_ntoa(&m\_mqtt\_server\_address));

        m\_is\_dns\_resolved = 1;

        m\_mqtt\_process\_now = true;

    }

    else

    {

        /\* DNS resolving failed -> try another server \*/

        DEBUG\_INFO("mqtt\_dns\_found: Failed to resolve server address resolved, trying next server\r\n");

        m\_is\_dns\_resolved = 0;

    }

}

- Hàm subcribe 1 topic, với các tham số chính truyền vào là topic name, QoS và callback

    /\* Subscribe to a topic named "qrm/imei/st\_data" with QoS level 1, call mqtt\_sub\_request\_cb with result \*/

    err\_t err = mqtt\_subscribe(&m\_mqtt\_static\_client,

                                m\_mqtt\_subscribe\_topic\_name,

                                SUB\_QoS,

                                mqtt\_sub\_request\_cb, NULL);

    DEBUG\_INFO("%s: topic %s\r\n", \_\_FUNCTION\_\_, m\_mqtt\_subscribe\_topic\_name);

- Hàm publish data lên mqtt broker và callback

    err\_t err = mqtt\_publish(&m\_mqtt\_static\_client,

                             m\_mqtt\_publish\_topic\_name,

                             m\_mqtt\_tx\_buffer,

                             strlen(m\_mqtt\_tx\_buffer),

                             PUB\_QoS,

                             PUB\_RETAIN,

                             mqtt\_pub\_request\_cb,

                             NULL);

- Hàm nhận dữ liệu từ mqtt broker

static void mqtt\_incoming\_data\_cb(void \*arg, const u8\_t \*data, u16\_t len, u8\_t flags)

{

    DEBUG\_INFO("MQTT data cb, length %d, flags %u\r\n", len, (unsigned int)flags);

    if (flags & MQTT\_DATA\_FLAG\_LAST)

    {

        /\* Last fragment of payload received (or whole part if payload fits receive buffer

              See MQTT\_VAR\_HEADER\_BUFFER\_LEN)  \*/

        DEBUG\_INFO("Payload data: %s\r\n", (const char \*)data);

        if (m\_is\_valid\_sub\_topic == 1)

        {

            m\_is\_valid\_sub\_topic = 0;

            /\* Update firmware message  \*/

            if (strstr((char \*)data, "UDFW,"))

            {

                DEBUG\_INFO("Update firmware\r\n");

            }

            else if (strstr((char \*)data, "PLEASE RESET"))

            {

                NVIC\_SystemReset();

            }

        }

        // Clear received buffer of client -> du lieu nhan lan sau khong bi thua cua lan truoc,

        // neu lan truoc gui length > MQTT\_VAR\_HEADER\_BUFFER\_LEN

        memset(m\_mqtt\_static\_client.rx\_buffer, 0, MQTT\_VAR\_HEADER\_BUFFER\_LEN);

    }

    else

    {

        /\* Handle fragmented payload, store in buffer, write to file or whatever \*/

    }

}

# Source code

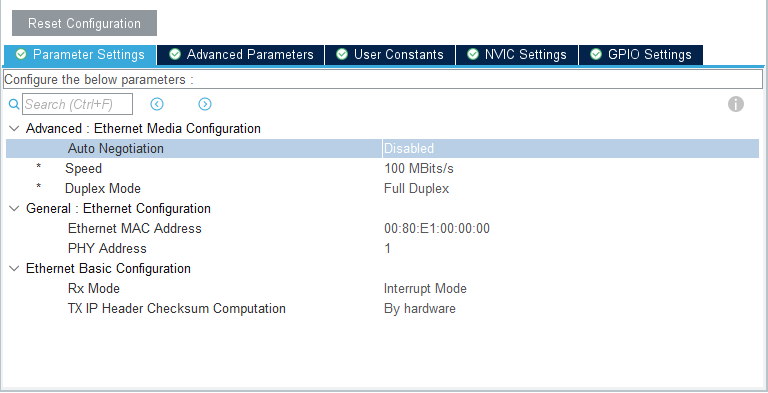
Ở demo này, mình sử dụng 2 thiết bị là 1 kit stm32l083 và 1 kit stm32f4. Các bạn có thể tham khảo ở repository sau.

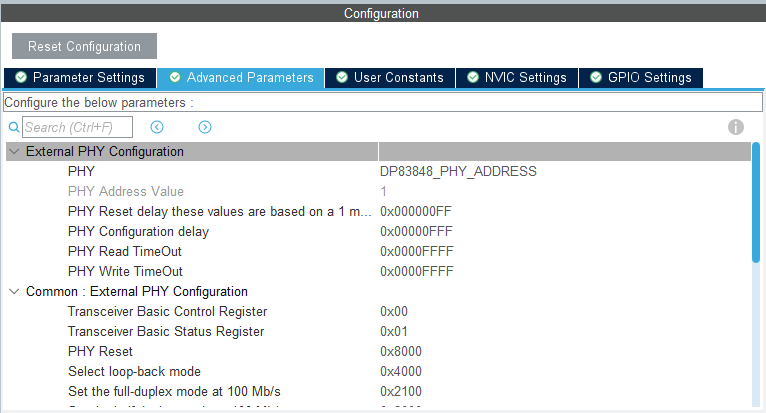
<https://github.com/huybk213/lwip_porting>

Phần 2. Cấu hình và triển khai LWIP với ETH trên stm32.  
Trong phạm vi bài viết này, ta sẽ thiết lập kết nối internet qua Ethernet và truyền nhận tin qua http.

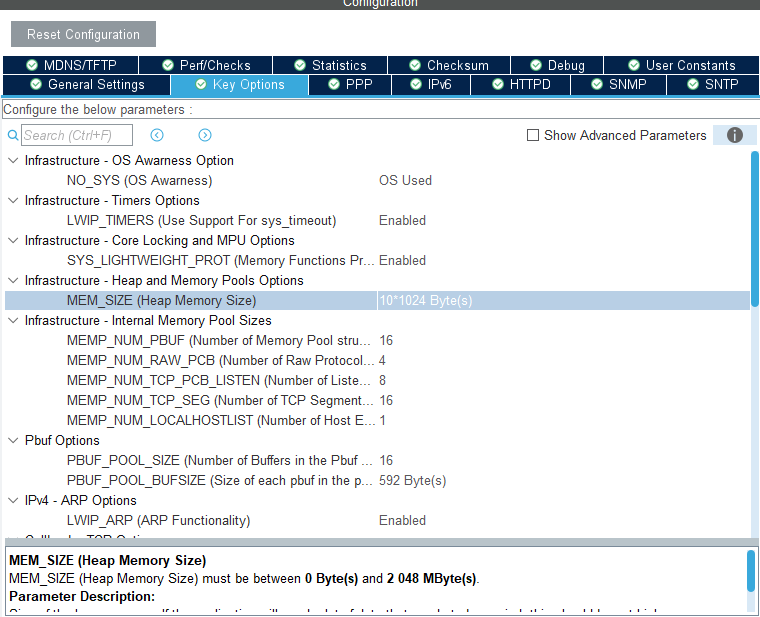
## P2.1. Cấu hình trên cubeMX

Trong phần connectivity, enable phần ETH chọn mode RMII (cho tiết kiệm chân). Trong mục pẩmeter settings các bạn chọn như hình:



Tắt chức năng auto negotiation khi bạn không biết địa chỉ bạn sẽ kết nối đến có cấu hình chức năng này không (chức năng chỉ hoạt dông khi 2 thiết bị kết nối với nhau đều bật chức năng auto negotiation này). Rx mode chọn Interrupt Mode vì ta sẽ dùng free – rtos.  
trong phần advance parameters cấu hình như sau:  


PHY chọn theo phần cứng bạn sử dụng. các thông số khác để nguyên. Đối với 1 số vi xử lý, ta cần điều chỉnh lại địa chỉ chứa phần xử lý ethernet (xem datasheet để biết cấu trúc bộ nhớ).   
Tại phần Middleware bật chức năng free-rtos và LWIP.  
Phần RTOS cấu hình tuỳ theo ứng dụng nhưng phải đảm bảo heap size đủ lớn.

Phần LWIP các cấu hình cần chú ý như sau:  
Enable chức năng LWIP\_DHCP, cài giá trị MEM\_SIZE, ở đây mình cài giá trị là 10KB, tuỳ chọn các bật chức năng HTTP hay SNTP tuỳ vào ứng dụng.  


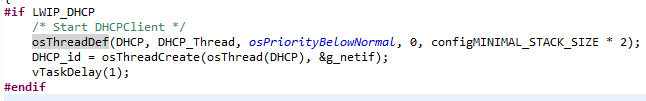
Sau đó, cấu hình cho clock và tiến hành gen code, trong phần cấu hình code generator chọn option “Generate peripheral initialization á a pair of .c/.h file” để dễ dàng quản lí các file và function hơn.  
Ở đây t sẽ dùng CubeIDE để code.

## P2.2. Triển khai code và các hàm cần thiết

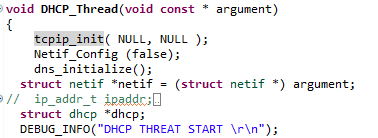
### P2.2.1. các file include

Trong file chứa task xử lý mạng, cần include các file sau:

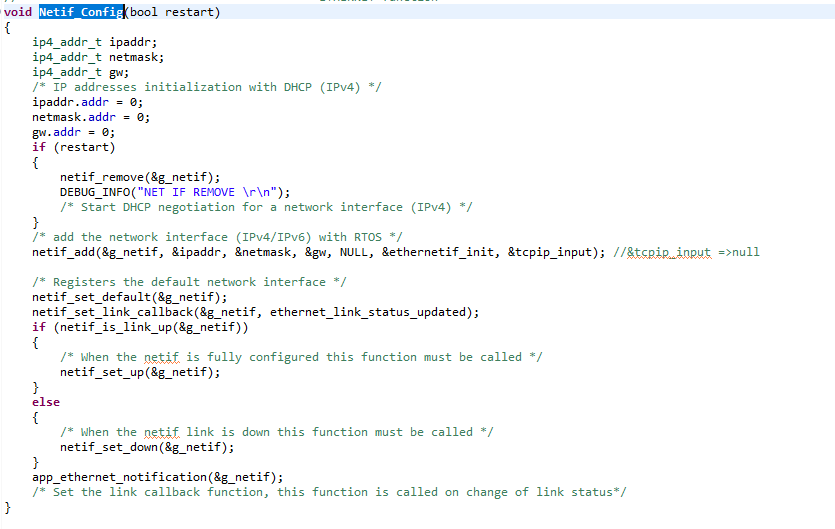


P2.2.2 Triển khai DHCP  
Để kết nối internet, cần lấy được Ip của thiết bị, ta cần tạo 1 task để xử lý tác vụ DHCP.  


Trong DHCP\_Thread sẽ gọi các hàm khác để xử lý. Trước hết ta sẽ cấu hình tcpip và đăng ký giao diện mạng (Network interface).



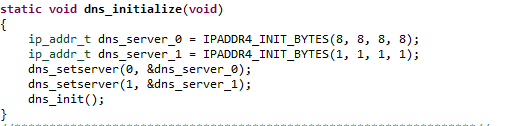
Hàm tcpip\_init được cung cấp sẵn trong thư viện LWIP. Hàm Netif\_Config( bool restart) được viết như sau:



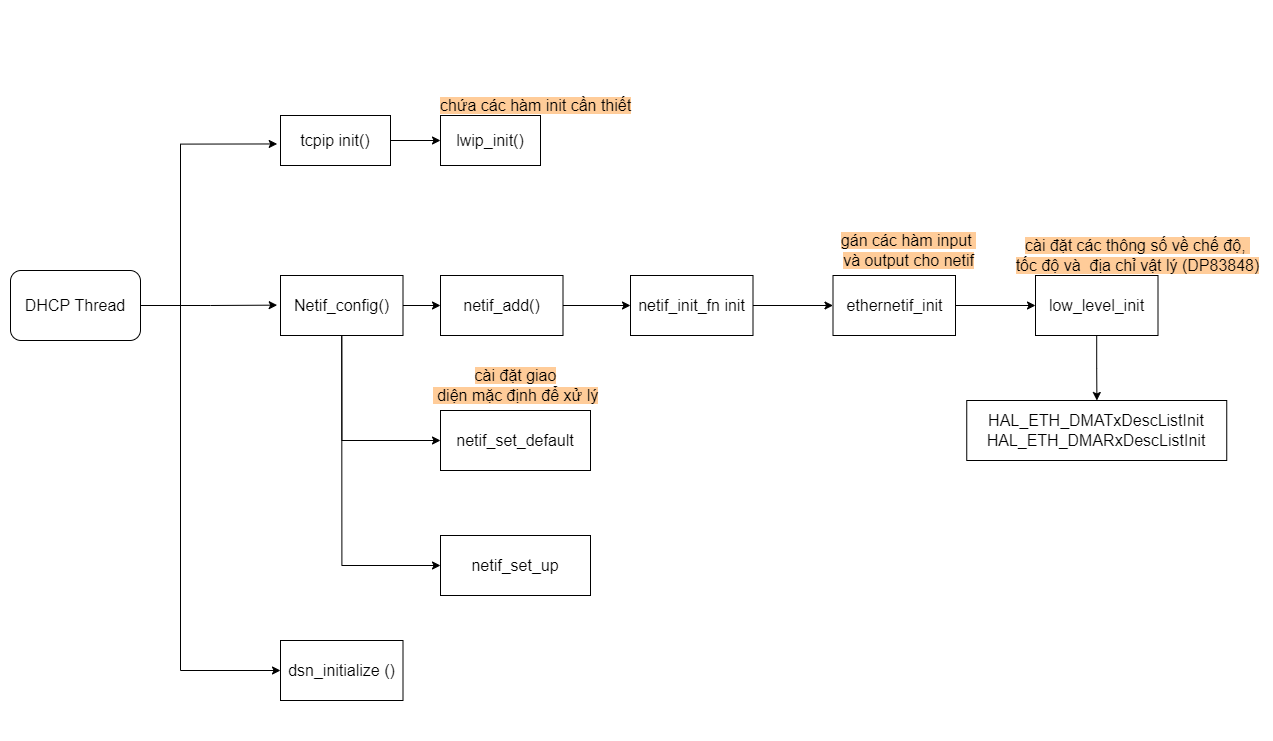
Giải thích code: 1 biến struct netif được khai báo chứa các thông tin về net interface. Sau khi reset các thông số hàm netif\_add đăng kí 1 giao diện cho biến vừa khai báo, các tác vụ mạng sau này sẽ thao tác trên biến này. Hàm netif\_set\_default đăng kí biến g\_netif trên là giao diện mặc định. Hàm netif\_set\_link\_callback đăng ký callback được gọi khi thay đổi trạng thái mạng.

Giá trị restart để xác định thời điểm hàm này được gọi, nếu restart = true thì nghĩa là trước đó đã mất mạng ta cần remove trước khi config lại.

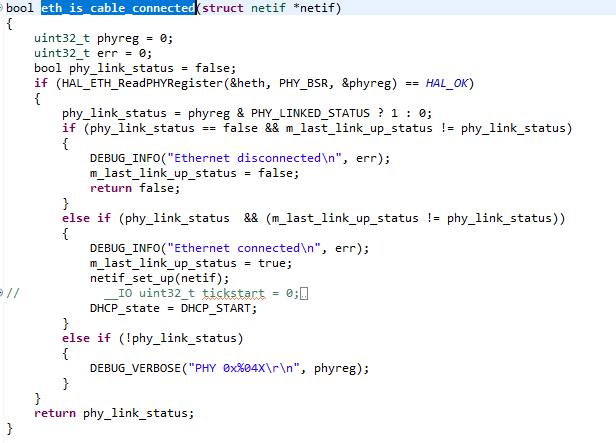
Hàm đăng kí DNS Sever:



Đăng kí DNS server để có thể truy cập theo tên miền.

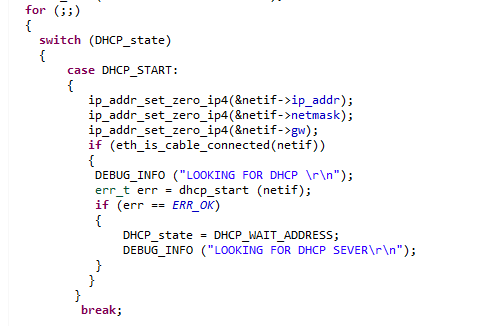
Sơ đồ các bước cài đặt mạng:  


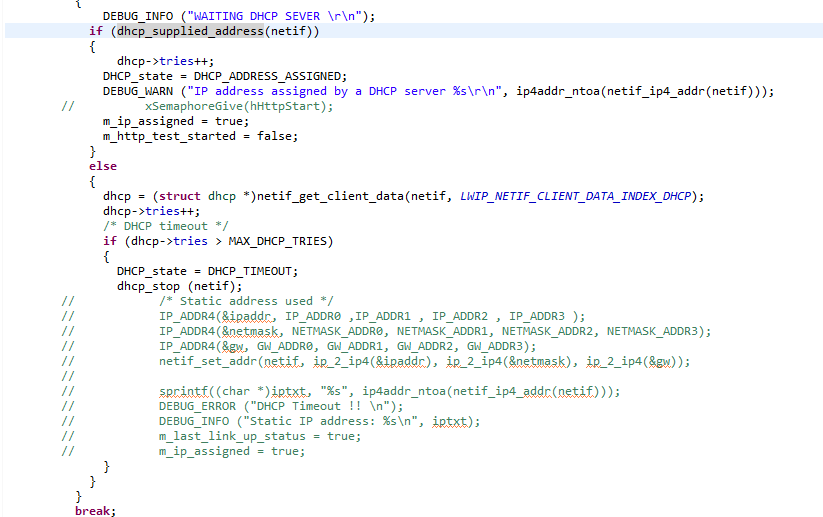
Hàm kiểm tra trạng thái mạng:



Đọc về giá trị thanh ghi PHY\_BSR để biết trạng thái mạng.

Sau khi cấu hình ta tiến hành lấy IP qua DHCP. Việc lấy ip được triển khai state machine bao gồm các state:

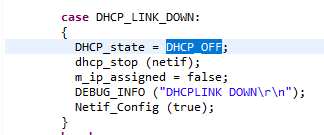
DHCP\_START : Trạng thái ban đầu, reset các giá trị ip\_addr, netmask và gw của netif. Kiểm tra nếu có mạng thì chuyển tới trạng thái tiếp theo.  


DHCP\_WAIT\_ADDRESS: Trạng thái đợi dhcp sever trả về ip được cấp.  


Hàm dhcp\_supplied\_address sẽ trả về true nếu ip đã được cấp.  
DHCP\_ADDRESS\_ASSIGNED: trạng thái sau khi đã lấy được ip. Xử lý tuỳ ứng dụng, hiện chưa xử lý state này.

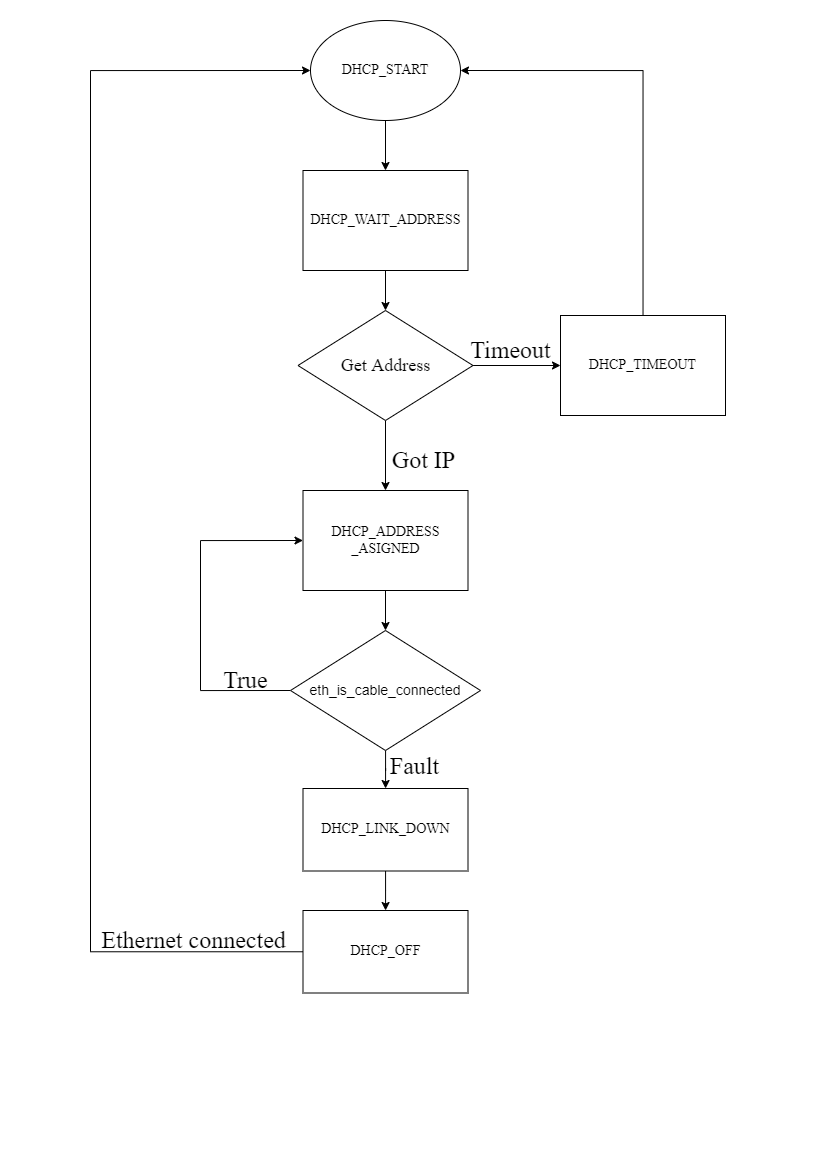
DHCP\_TIMEOUT: sever không trả về IP. Chuyển về trạng thái DHCP\_START để thử lại.

DHCP\_LINKDOWN: Trạng thái mất mạng. Tiến hành dừng tác vụ dhcp và Config lại interface.



Hàm kiểm tra trạng thái mạng được gọi mỗi 500ms để chuyển state nếu cần.

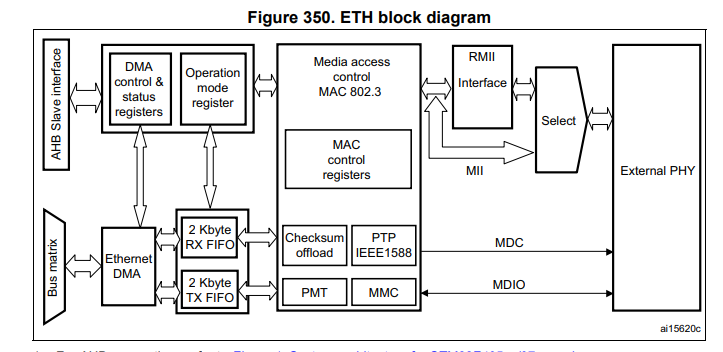
\* Lưu đồ DHCP:



## P2.3. lwip và Quá trình truyền nhận bản tin qua Ethernet.

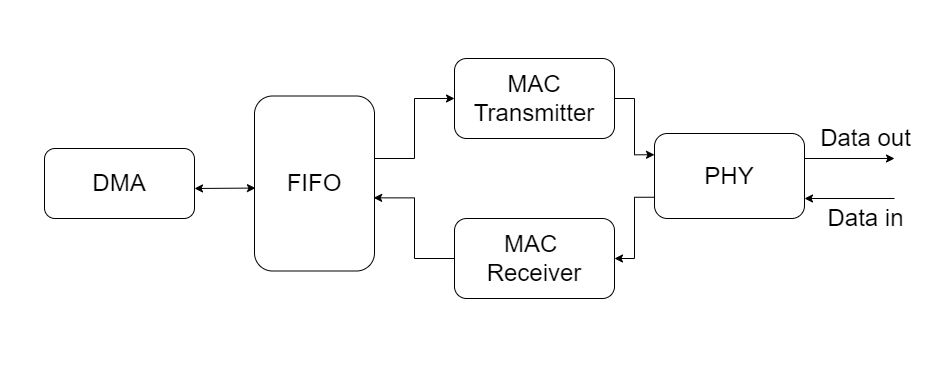
## P2.3.1. Cấu trúc khối kết nối Ethernet của STM32

Khối kết nối Ethernet STM32 được thể hiện trong sơ đồ sau:



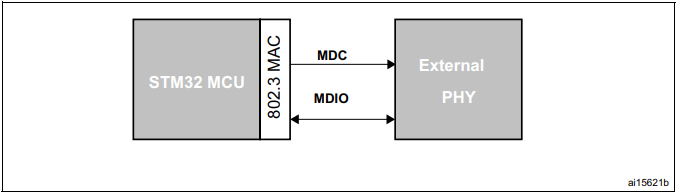
Có 3 interface được hỗ trợ để kết nối internet là MII và RMII.

Sơ đồ trên có thể đơn giản hoá như sau:

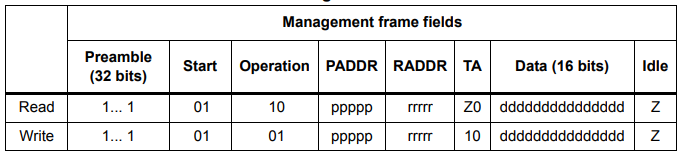


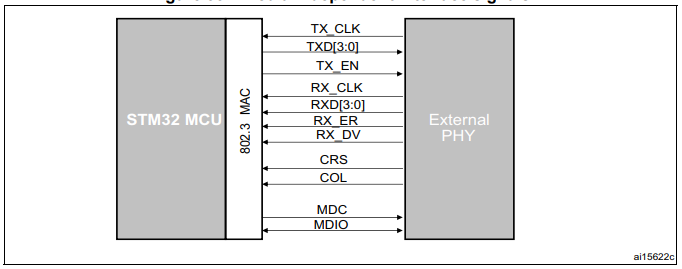
Dữ liệu sau khi được DMA truyền/nhận, STM32 sẽ bật cờ để báo hiệu dữ liệu đã được truyền nhận thành công. Hàm xử lý ngắt **ETH\_IRQHandler**P2.3.2. Giới thiệu các ETH interface

#### P2.3.2.1. SMI (Station management interface): Giao diện quản lý trạm

SMI cho phép truy cập thanh ghi PHY thông qua 2 đường clock và data. Giao diện này hỗ trợ có thể truy cập lên tới 32 PHYs.  
MDC: cấp xung đồng hồ phục vụ cho việc truyền data.  
MDIO: Dữ liệu vào /ra để truyền thoong tin trạng thái đến/ từ PHY và được đồng bộ với MDC clock.  
  


Tín hiệu SMI interface

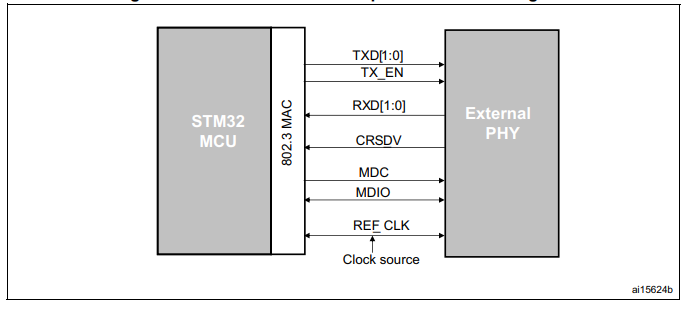
\* Cấu trúc khung bản tin SMI  


Preamble: Gồm 32 bit 1 liền kề với nhiệm vụ chính là thiết lậpdồng bộ hoá với thiết bị PHY (Ví dụ: DP83848).  
Start: bắt đầu gói tin được định nghĩa trc là 2 bit <01>.  
Operation: Định nghĩa chiều truyền dữ liệu (Đọc hoặc ghi)  
PADDR: địa chỉ PHY cho phép phân biệt 32 địa chỉ PHY. MSB là bit đầu tiên được truyền nhận.   
RADDR: Địa chỉ thanh ghi, tương tụ địa chỉ PHY.  
TA: Bit phân biệt trong quá trình đọc để tránh nhầm lẫn giữa địa chỉ và data. Khi đọc thì Bit đầu tên sẽ ở trạng thái cao trở.  
2.3.2.2. MII (Media-independent interface):  
Giao diện độc lập với phương tiện (MII) xác định kết nối giữa lớp con MAC và PHY để truyền dữ liệu ở tốc độ 10 Mbit / s và 100 Mbit / s.  
  


Đối với MII có thêm các đường tín hiệu riêng cho việc điều khiển và truyền nhận dữ liệu dễ dàng hơn.  
\* MII clock:  
Để có thể tạo ra cả clock truyền và nhận thì giao diện vật lý bên ngoài (DP83848) cần có thêm 1 nguồn clock ngoại 25 MHz.

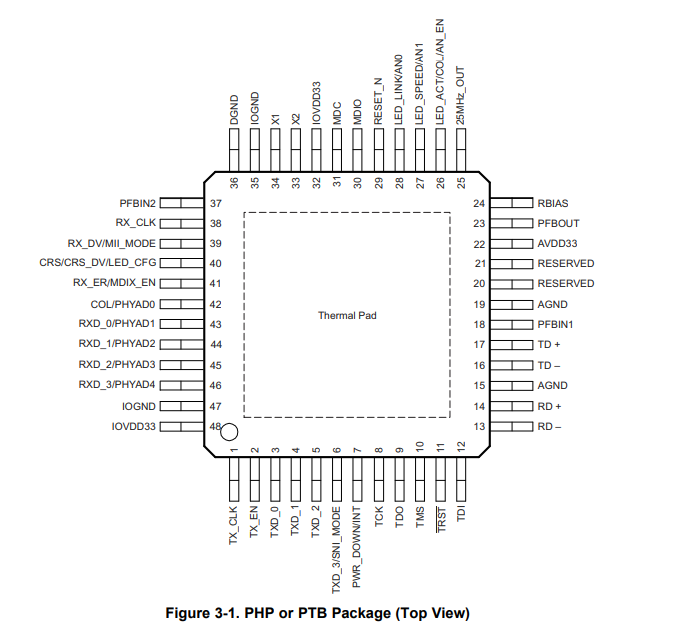
#### 2.3.2.3. RMII (Reduce Media-independent interface):

RMII tương tự như MII nhưng số chân cần sử dụng giảm đi đáng kể. Tuy nhiên, clock sử dung phải tăng thành 50MHz.



P2.3.3. Giao tiếp với thanh ghi PHY.

Dùng lệnh HAL\_ETH\_ReadPHYRegister và HAL\_ETH\_WritePHYRegister để đọc và ghi lên thanh ghi. Câu lệnh sẽ đọc / ghi vào thanh ghi data của MAC: MACMIIDR theo địa chỉ của PHY và sẽ được nhận từ / truyền tới External PHY. Ở đây ta dùng DP83848 với RMII interface.



Mô tả các chân của chuẩn RMII:

|  |  |  |
| --- | --- | --- |
| Tên tín hiệu | Mô tả | Hướng đi |
| REF\_CLK | Đồng hồ tham chiếu 50 MHz liên tục | Đồng hồ tham chiếu có thể là đầu vào trên cả hai thiết bị từ nguồn xung nhịp MAC sang PHYcó thể được điều khiển từ PHY sang MAC |
| TXD0 | Truyền bit dữ liệu 0  (truyền trước) | MAC sang PHY |
| TXD1 | Truyền bit dữ liệu 1 | MAC sang PHY |
| TX\_EN | Khi ở mức cao, truyền dữ liệu trên TXD0 và TXD1 đến máy phát | MAC sang PHY |
| RXD0 | Nhận bit dữ liệu 0 (nhận trước) | PHY sang MAC |
| RXD1 | Nhận bit dữ liệu 1 | PHY sang MAC |
| CRS\_DV | Carrier Sense (CRS) và RX\_Data Valid (RX\_DV) được ghép lại dựa trên các chu kỳ đồng hồ thay thế. Ở chế độ 10 Mbit / s, nó luân phiên mỗi 10 chu kỳ đồng hồ. | PHY sang MAC |
| RX\_ER | Nhận về lỗi (tùy chọn trên công tắc) | PHY sang MAC |
| MDIO | Dữ liệu quản lý | Hai chiều |
| MDC | Đồng hồ dữ liệu quản lý. | MAC sang PHY |

### P2.3.2. Quá trình xử lý bản tin nhận được.

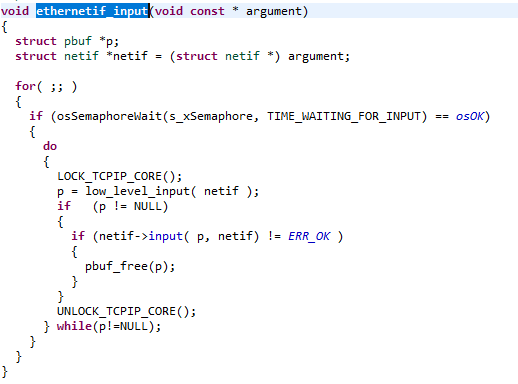
Khi có bản tin được truyền đến, 1 tín hiệu sẽ được phát đi để bắt đầu xử lý dữ liệu truyền đến. Cụ thể trong trường hợp dùng free-rtos sẽ gửi 1 semaphore để bắt đầu nhận dữ liệu sau đó dữ liệu sẽ được xử lí trong hàm tcpip\_input.

Đăng ký hàm xử lý input cho data: tcpip\_input



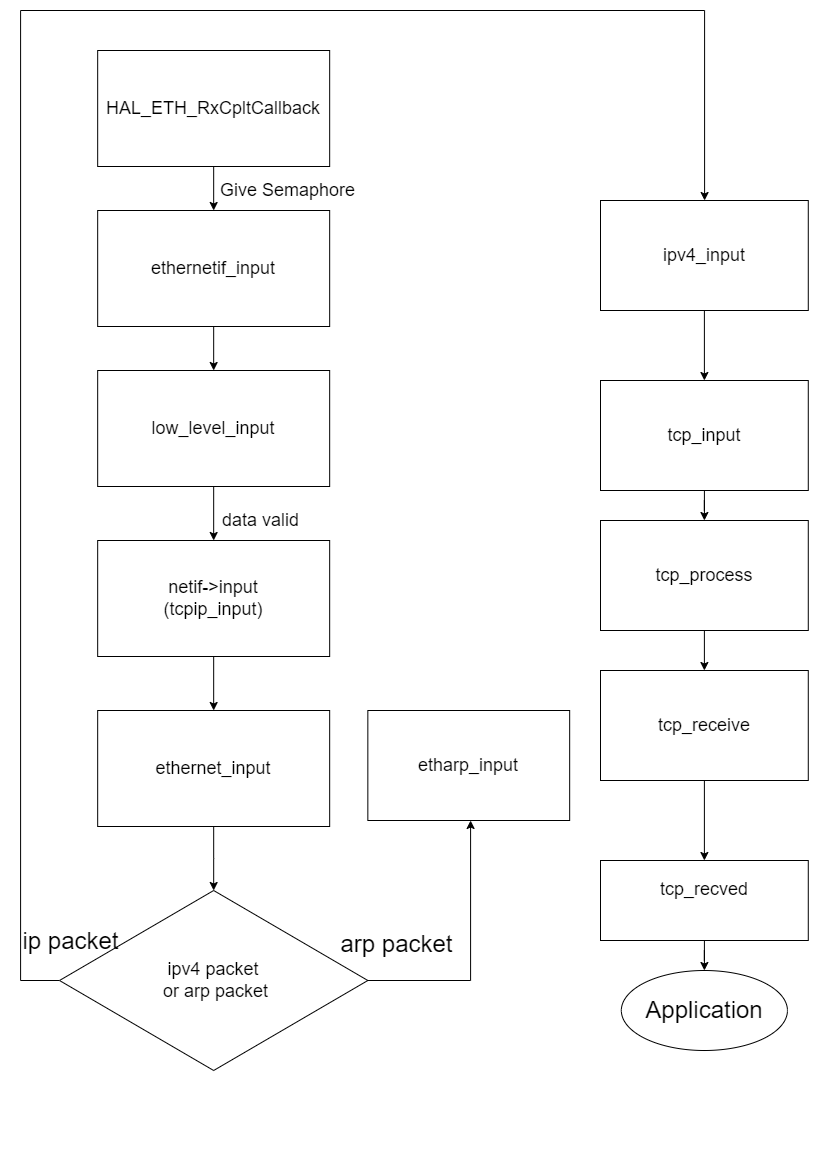
Tạo task xử lý data về: ethernet\_input





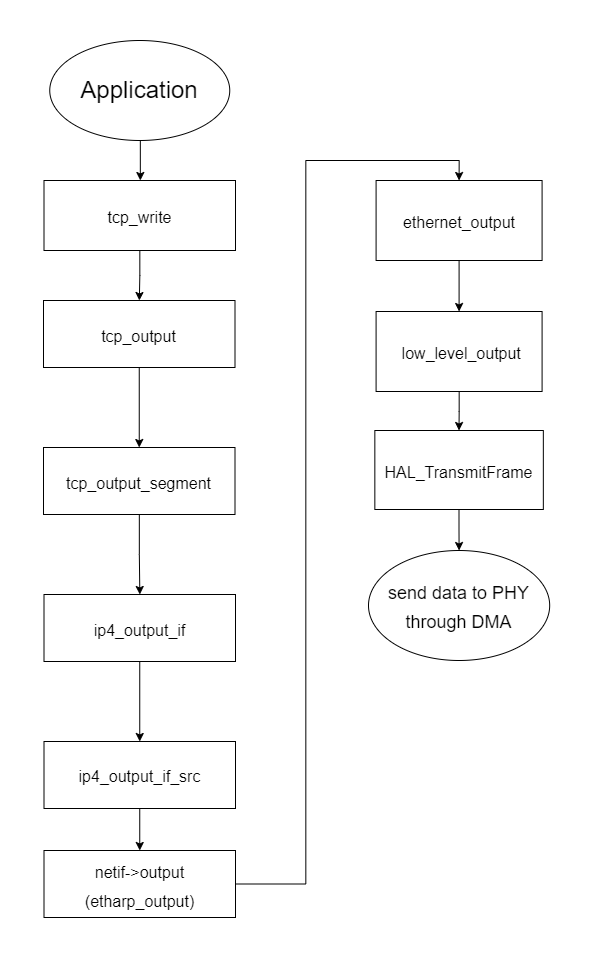
Task trên nhận đợi data truyền về, nhận data vào con trỏ p, sau đó xử lý trong hàm input đã đăng ký.

Quá trình nhận data và xử lý data nhận được trình bày cụ thể ở sơ đồ sau (Sơ đồ trình bày quá trình nhận bản tin từ khi dữ liệu đã được lưu vào bộ nhớ nhờ DMA):



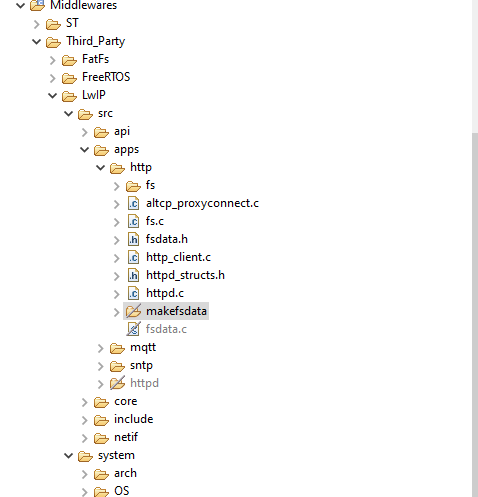
Quá trình nhận dữ liệu qua lwip

P2.3.3. Quá trình truyền dữ liệu.  
tương tự như quá trình nhận dữ liệu, qua trình truyền cũng đi qua các lớp từ ứng dụng xuống tới lớp vật lí để gửi dữ liệu. Sơ đồ sau thể hiện quá trình truyền dữ liệu của thiết bị.



Quá trình truyền dữ liệu qua lwip

P2.4. Http và lwip.  
Lưu ý trong file ta cần exclude thư mục “makefsdata” và file “fsdata”.

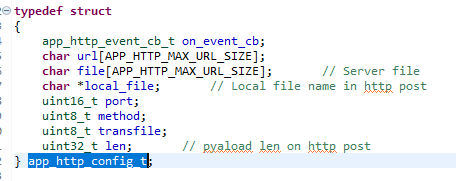


File “http\_client.c” sẽ chứa các hàm để kết nối với http server.

Các file “app\_http.c” chứa các hàm ứng dụng http để truy cập đến các lớp dưới.

Để sử dung các hàm này ta khai báo các biến app\_http\_config\_t chứa các trường để xử lý tác vụ http.  


Ta gán các giá trị cho các trường của biến app\_http\_config\_t



url: mảng chứa tên miền hoặc địa chỉ ip của server.  
file: đường dẫn sau tên miền server.  
port: cổng http thường bằng 80 và 8080.

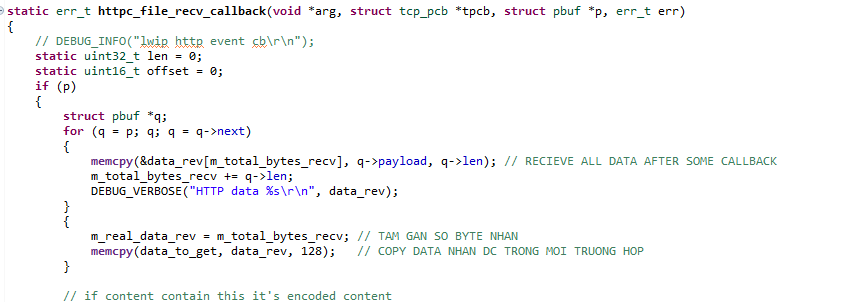
Các tuỳ chọn khác tuỳ vào ứng dụng có thể thay đổi.

Các callback sẽ được gọi:

Hàm nhận data.

Data được nhận qua hàm httpc\_tcp\_rev bằng cách gọi hàm tcp\_recved, sau đó gọi hàm sau để xử lý dữ liệu nhận được.

**httpc\_file\_recv\_callback**



Con trỏ pbuf \*p trở tới địa chỉ chứa data nhận về.

Hàm trả về kết quả kết nối HTTP



HTTPC\_RESULT\_OK: “gửi file thành công, status code trả về :200” /\*\* File successfully received \*/

Các kết quả khác cho về kết quả lỗi.