REFERENCE MANUEL

FOR ETHERNET PROTOCOL

AUTHOR : ORHAN AĞIRMAN

CREATED ON : 07.09.2021

ETHERNET COMMUNICATION PROTOCOL USING STM32-NUCLEO BOARDS

The Ethernet peripheral enables the STM32 Nucleo Board to transmit and receive data over Ethernet in compliance with the IEEE 802.3-2002 standard. It supports two industry-standard interfaces to the external physical layer (PHY): the default media independent interface (MII) defined in the IEEE 802.3 specifications and the reduced media independent interface (RMII).

SMI (Station management interface)

The station management interface (SMI) allows the application to access any PHY registers through a 2-wire clock and data lines. The interface supports accessing up to 32 PHYs.

Both the MDC clock line and the MDIO data line are implemented as alternate function I/O in the microcontroller:

MDC: a periodic clock that provides the timing reference for the data transfer at the maximum frequency of 2.5 MHz.

MDIO: data input/output bitstream to transfer status information to/from the PHY device synchronously with the MDC clock signal

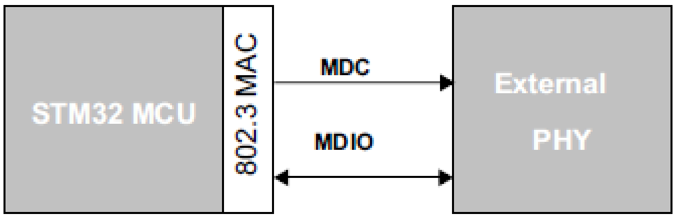


Figure-1: SMI interface signals

SMI frame format:

Table 1: Management frame format:

	Management frame fields							
	Preamble (32 bits)	Start	Operation	PADDR	RADDR	TA	Data (16 bits)	Idle
Read	1... 1	01	10	ppppp	rrrrr	Z0	dddddddddddddddd	Z
Write	1... 1	01	01	ppppp	rrrrr	10	dddddddddddddddd	Z

MII (Media-independent interface)

The media-independent interface (MII) defines the interconnection between the MAC sublayer and the PHY for data transfer at 10 Mbit/s and 100 Mbit/s.

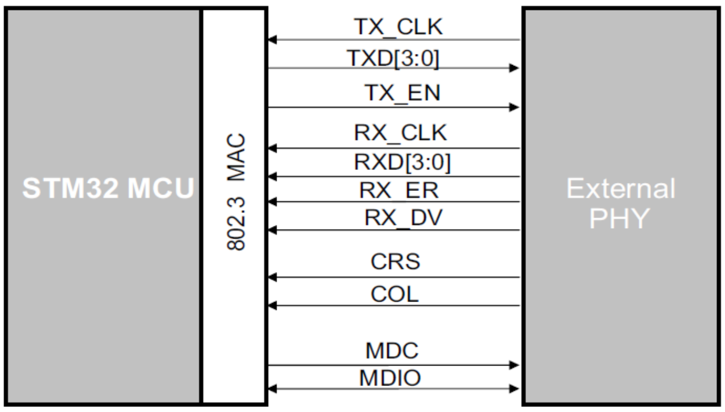


Figure 2: Media independent interface signals

Table 2: MII Signal Description:

MII_TX_CLK	Continuous clock that provides the timing reference for the TX data transfer.
MII_RX_CLK	Continuous clock that provides the timing reference for the RX data transfer.
MII_TX_EN	Transmission enable indicates that the MAC is presenting nibbles on the MII for transmission.
MII_TXD[3:0]	Transmit data is a bundle of 4 data signals driven synchronously by the MAC sublayer and qualified (valid data) on the assertion of the MII_TX_EN signal.
MII_CRS	Carrier sense is asserted by the PHY when either the transmit or receive medium is non idle. It shall be deasserted by the PHY when both the transmit and receive media are idle.
MII_COL	Collision detection must be asserted by the PHY upon detection of a collision on the medium and must remain asserted while the collision condition persists. This signal is not required to transition synchronously with respect to the TX and RX clocks.
MII_RXD[3:0]	Reception data is a bundle of 4 data signals driven synchronously by the PHY and qualified (valid data) on the assertion of the MII_RX_DV signal.
MII_RX_DV	Receive data valid indicates that the PHY is presenting recovered and decoded nibbles on the MII for reception.
MII_RX_ER	Receive error must be asserted for one or more clock periods (MII_RX_CLK) to indicate to the MAC sublayer that an error was detected somewhere in the frame.

RMII (Reduced media-independent interface)

The reduced media-independent interface (RMII) specification reduces the pin count between the microcontroller Ethernet peripheral and the external Ethernet in 10/100 Mbit/s.

According to the IEEE 802.3u standard, an MII contains 16 pins for data and control. The RMII specification is dedicated to reduce the pin count to 7 pins (a 62.5% decrease in pin count).

The RMII is instantiated between the MAC and the PHY. This helps translation of the MAC's MII into the RMII. The RMII block has the following characteristics:

- It supports 10-Mbit/s and 100-Mbit/s operating rates
- The clock reference must be doubled to 50 MHz
- The same clock reference must be sourced externally to both MAC and external Ethernet PHY
- It provides independent 2-bit wide (dibit) transmit and receive data paths

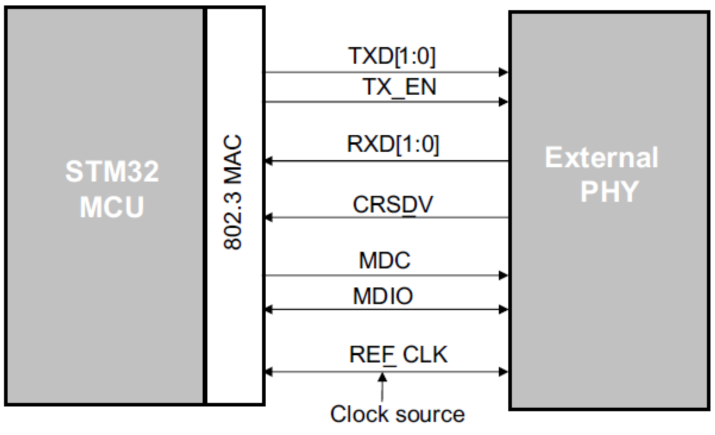


Figure 2: Reduced Media independent interface signals

LwIP (Lightweight Internet Protocol)

The lightweight Internet Protocol (lwIP) is a small independent implementation of the network protocol suite that has been initially developed by Adam Dunkels.

The focus of the lwIP network stack implementation is to reduce memory resource usage while still having a full-scale TCP. This makes lwIP suitable for use in embedded systems with tens of kilobytes of free RAM and room for around 40 kilobytes of code ROM.

lwIP supports the following protocols:

- ARP (Address Resolution Protocol)
- IP (Internet Protocol) v4 and v6
- TCP (Transmission Control Protocol)
- UDP (User Datagram Protocol)
- DNS (Domain Name Server)
- SNMP (Simple Network Management Protocol)
- DHCP (Dynamic Host Configuration Protocol)
- ICMP (Internet Control Message Protocol) for network maintenance and debugging
- IGMP (Internet Group Management Protocol) for multicast traffic management
- PPP (Point to Point Protocol)
- PPPoE (Point to Point Protocol over Ethernet)

LwIP Architecture

LwIP complies with the TCP/IP model architecture which specifies how data should be formatted, transmitted, routed, and received to provide end-to-end communications.

This model includes four abstraction layers which are used to sort all related protocols according to the scope of networking involved.

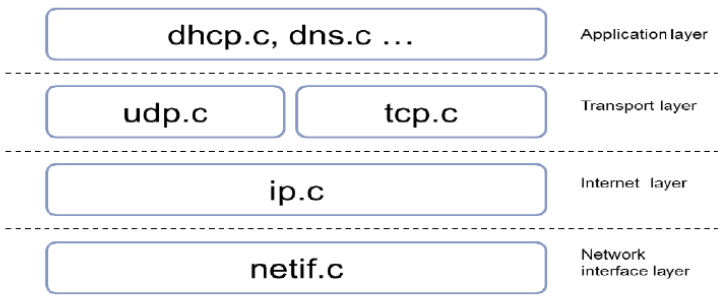


Figure 3: LwIP Architecture

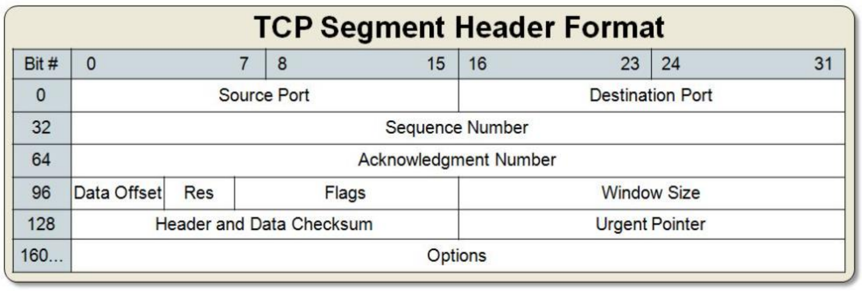
TCP (Transmission Control Protocol)

Transmission Control Protocol (TCP), which exists on the transport layer of the Open Systems Interconnection (OSI) model. The data is usually transmitted in packets. TCP is designed so that the packets of data will arrive without errors and in sequence. It provides reliable two-way communication similar to when we call someone on the phone. One side initiates the connection to the other, and after the connection is established, either side can communicate to the other. In addition, there is immediate confirmation that what we said actually reached its destination.

UDP (User Datagram Protocol)

User Datagram Protocol (UDP), is not a real connection, just a basic method for sending data from one point to another. Communicating with a UDP is more like mailing a letter than making a phone call. The connection is one-way only and unreliable.

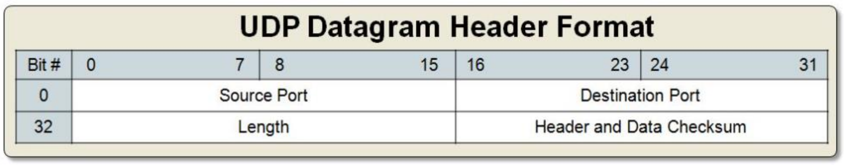
If we mail several letters, we can't be sure that they arrive in the same order, or even that they reached their destination at all.



TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset / Res	Flags	Window Size	
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

Figure 4: TCP Format



UDP Datagram Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

Figure 5: UDP Format

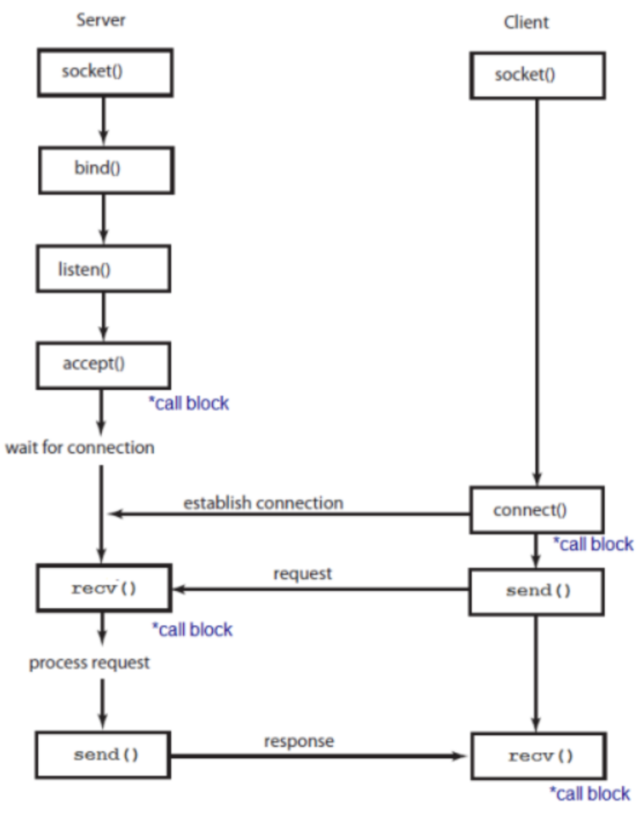


Figure 6: TCP Communication

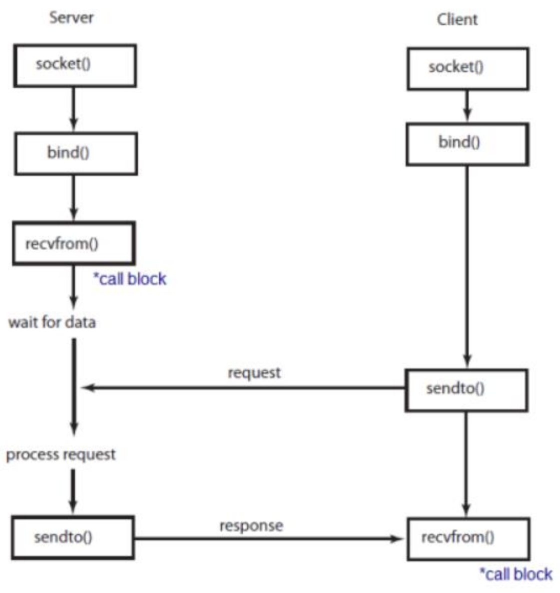


Figure 7: UDP Communication

lwIP offers three different APIs designed for different purposes:

- **Raw API** is the core API of lwIP. This API aims at providing the best performances while using a minimal code size. One drawback of this API is that it handles asynchronous events using callbacks which complexify the application design.
- **Netconn API** is a sequential API built on top of the Raw API. It allows multi-threaded operation and therefore requires an operating system. It is easier to use than the Raw API at the expense of lower performances and increased memory footprint.
- **BSD Socket API** is a Berkeley like Socket implementation (Posix/BSD) built on top of the Netconn API. Its interest is portability. It shares the same drawback than the Netconn API.

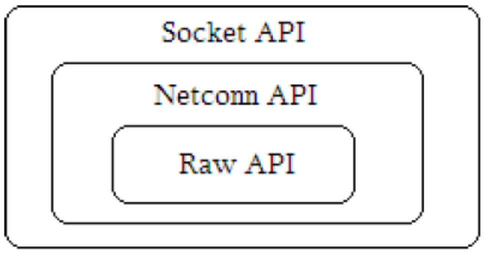


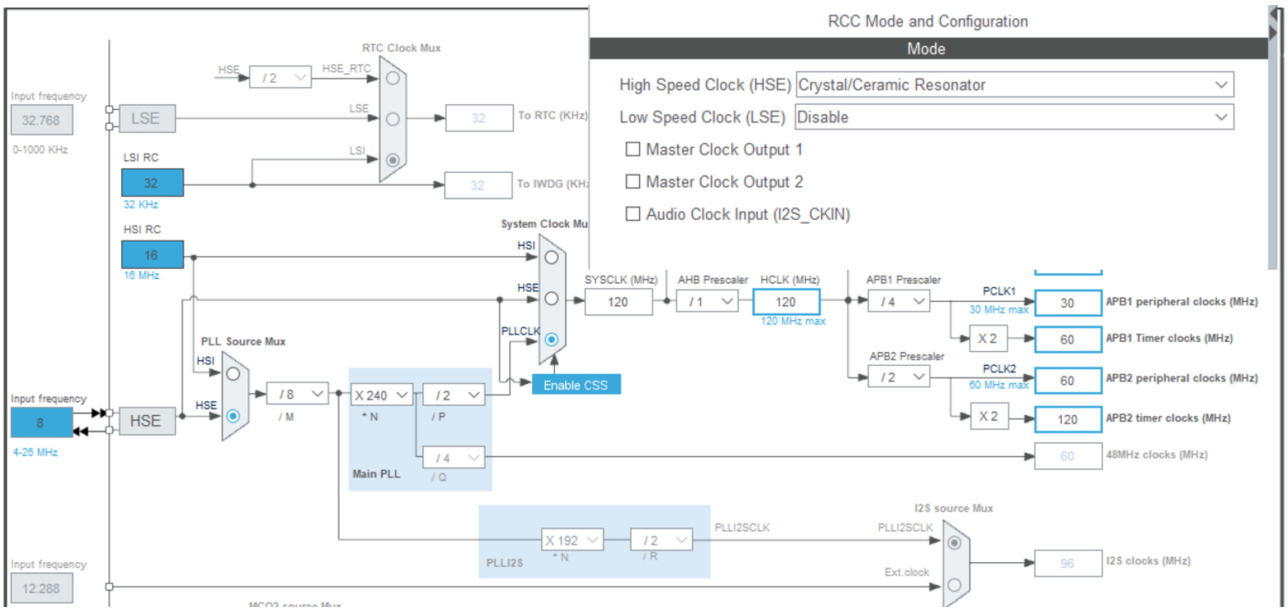
Figure 8: LwIP API

Table 3: Socket API functions:

API functions	Description
socket	Creates a new socket.
bind	Binds a socket to an IP address and port.
listen	Listens for socket connections.
connect	Connects a socket to a remote host IP address and port.
accept	Accepts a new connection on a socket.
read	Reads data from a socket.
write	Writes data on a socket.
close	Closes a socket (socket is deleted).

CONFIGURATION OF STM32CUBEMX

First of all, the clock configuration of STM32 Nucleo-F207ZG board should be done as follows.


RCC Mode and Configuration

Mode

High Speed Clock (HSE) Crystal/Ceramic Resonator

Low Speed Clock (LSE) Disable

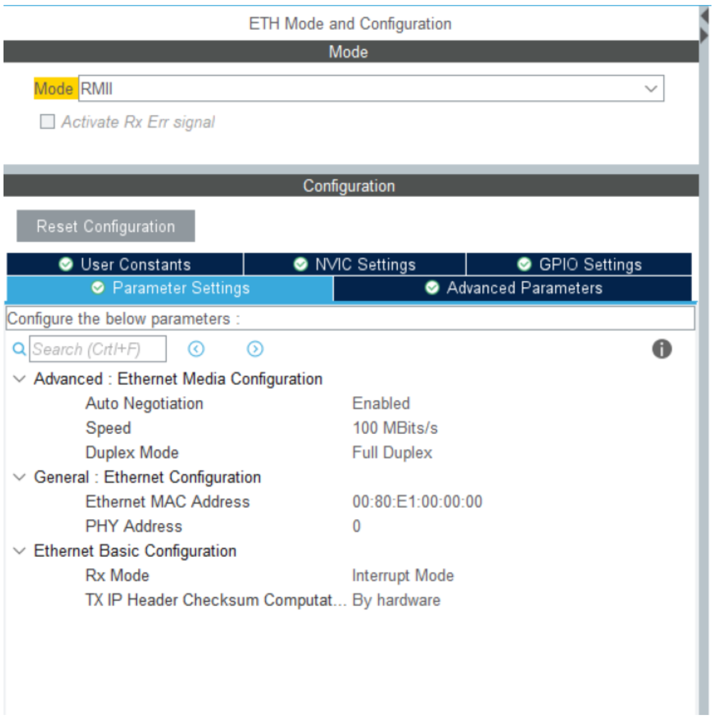
☐ Master Clock Output 1

☐ Master Clock Output 2

☐ Audio Clock Input (I2S_CKIN)

Figure 9: The clock configuration of STM32 Nucleo-F207ZG

Ethernet must be activated from the Connectivity tab. RMII (Reduced media-independent interface) should be selected as the mode. The PHY address should be set to 0 or 1 depending on the board used. For the STM32 Nucleo-F207ZG board, this value is 0. The appropriate value can be checked from the schematic of the relevant board. If desired, the MAC address can be changed.



ETH Mode and Configuration

Mode

Mode RMII

Activate Rx Err signal

Configuration

Reset Configuration

User Constants | NVIC Settings | GPIO Settings

Parameter Settings | Advanced Parameters

Configure the below parameters :

Advanced : Ethernet Media Configuration	
Auto Negotiation	Enabled
Speed	100 MBits/s
Duplex Mode	Full Duplex
General : Ethernet Configuration	
Ethernet MAC Address	00:80:E1:00:00:00
PHY Address	0
Ethernet Basic Configuration	
Rx Mode	Interrupt Mode
TX IP Header Checksum Computat...	By hardware

Figure 10: Ethernet Mode and Configuration

Ethernet pins should be rearranged according to the diagram below. According to the diagram, the ETH_TXD0 and ETH_TX_EN pins should be changed, the other pins are automatically set correctly. The PHY address value is also set by checking the RXER/PHYAD0 pin in the diagram, we set the PHY address as 0 because the relevant pin is connected to the ground in the diagram.

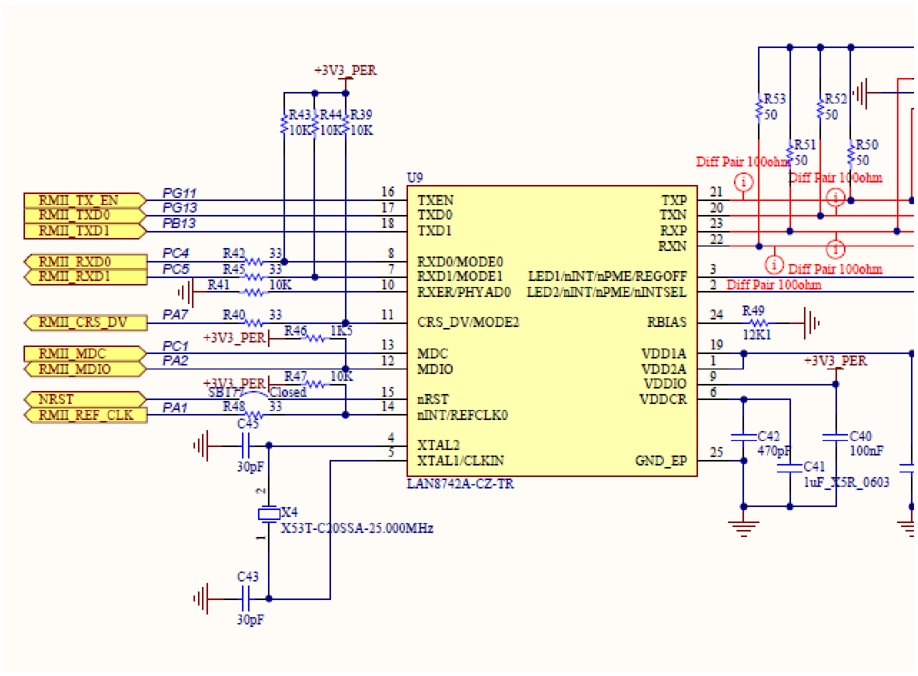


Figure 11: Part of the ethernet schematic of the STM32 Nucleo-F207ZG board

Since we will use 3 user leds and 1 user button of the nucleo card in our project, we define the relevant leds as input and output. All of our pin definitions are as in figure-12 below.

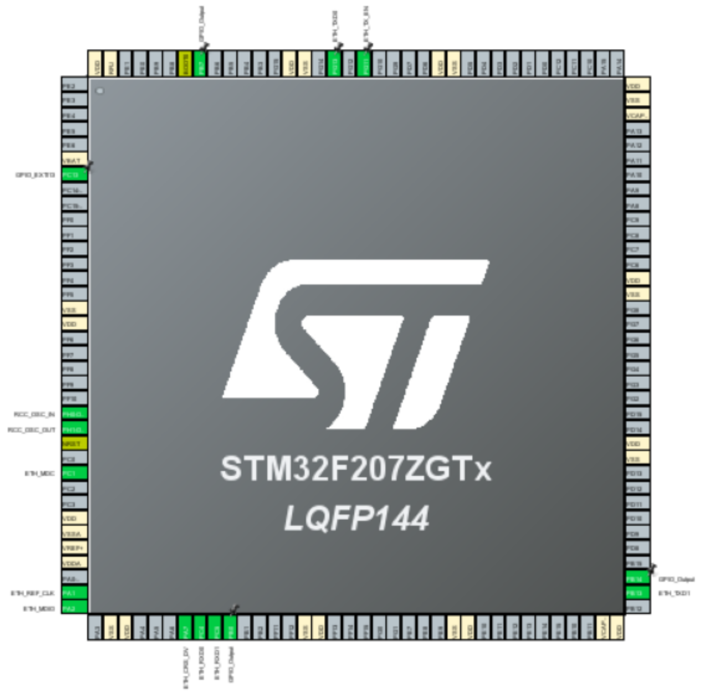
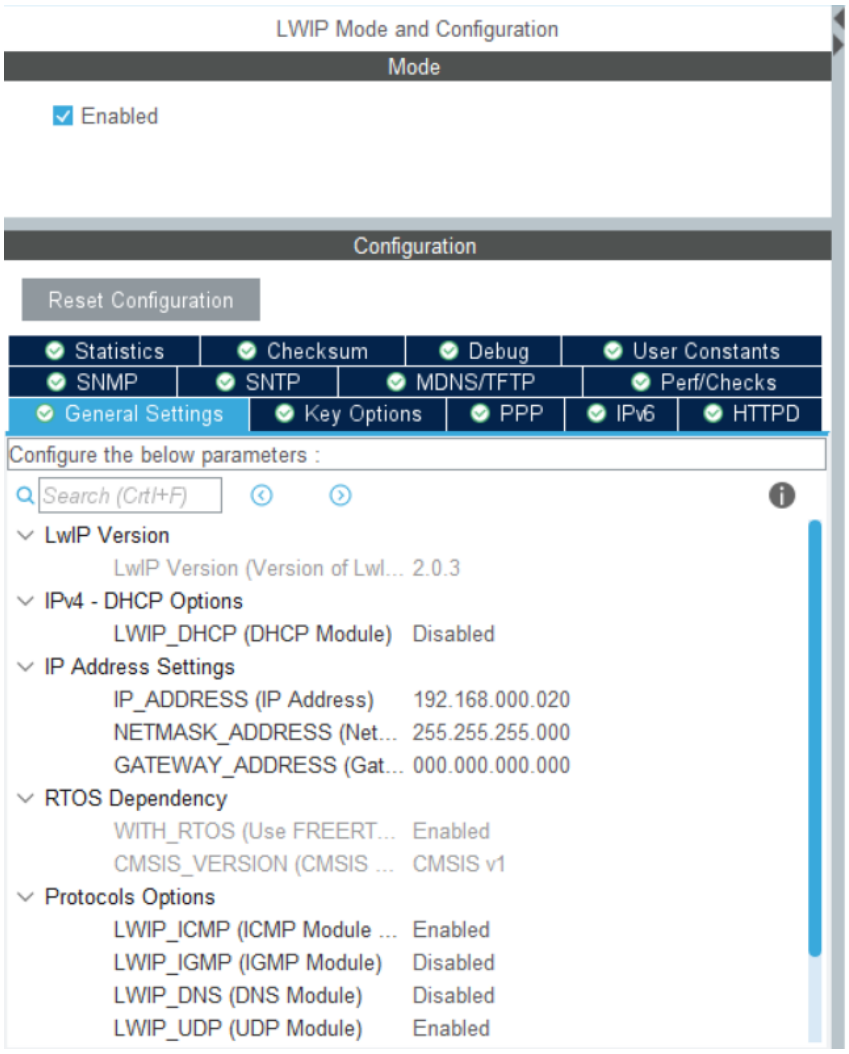


Figure 12: Pinout view of the STM32 Nucleo-F207ZG board

Since we will use the Socket API in our project, we need to use FREERTOS. Also, since we need to use LwIP, additional settings must be made for the use of FREERTOS in LwIP. However, if we activate LwIP first and then activate FREERTOS, STM32CUBEMX will make the necessary adjustments for FREERTOS in LwIP.

Therefore, LwIP is activated first and the settings are made as follows. We can manually assign IP to our bourd by disabling LWIP_DHCP (DHCP Module). In this example, we have assigned the IP of 192.168.0.20 to STM32 Nucleo-F207ZG.


LWIP Mode and Configuration

Mode

☑ Enabled

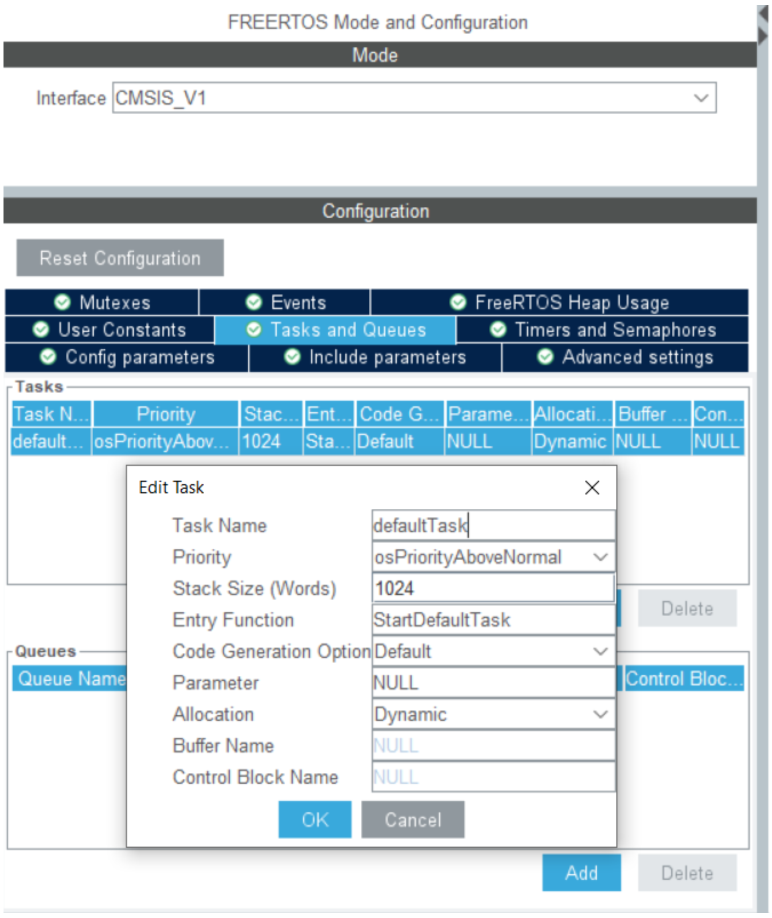
Configuration

Configure the below parameters :

LwIP Version	
LwIP Version (Version of Lwl...	2.0.3
IPv4 - DHCP Options	
LWIP_DHCP (DHCP Module)	Disabled
IP Address Settings	
IP_ADDRESS (IP Address)	192.168.000.020
NETMASK_ADDRESS (Net...	255.255.255.000
GATEWAY_ADDRESS (Gat...	000.000.000.000
RTOS Dependency	
WITH_RTOS (Use FREERT...	Enabled
CMSIS_VERSION (CMSIS ...	CMSIS v1
Protocols Options	
LWIP_ICMP (ICMP Module ...	Enabled
LWIP_IGMP (IGMP Module)	Disabled
LWIP_DNS (DNS Module)	Disabled
LWIP_UDP (UDP Module)	Enabled

Figure 13: LWIP Mode and Configuration

After LwIP is activated and necessary adjustments are made, FREERTOS is activated and the mode is selected as CMSIS_V1. A task is created and the Stack Size (Words) is set to 1024 and the Priority to osPriorityAboveNormal because the Socket API has a heavy load on the system.


FREERTOS Mode and Configuration

Mode

Interface: CMSIS_V1

Configuration

Tasks

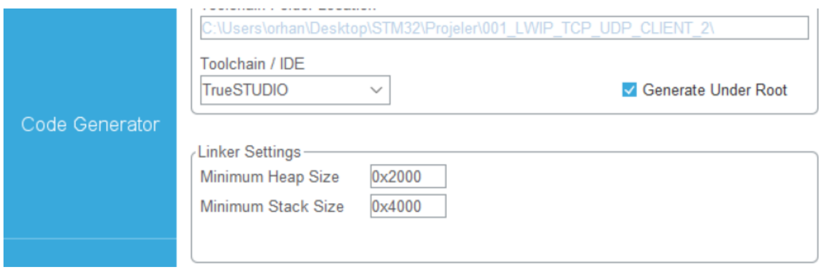
Task N...	Priority	Stac...	Ent...	Code G...	Parame...	Allocati...	Buffer ...	Con...
default...	osPriorityAbov...	1024	Sta...	Default	NULL	Dynamic	NULL	NULL

Edit Task

Task Name	defaultTask
Priority	osPriorityAboveNormal
Stack Size (Words)	1024
Entry Function	StartDefaultTask
Code Generation Option	Default
Parameter	NULL
Allocation	Dynamic
Buffer Name	NULL
Control Block Name	NULL

Figure 14: FREERTOS Mode and Configuration

Finally, Heap and Stack size are set to 0x2000 and 0x4000 respectively.


C:\Users\orhan\Desktop\STM32\Projeler\001_LWIP_TCP_UDP_CLIENT_2\

Toolchain / IDE: TrueSTUDIO

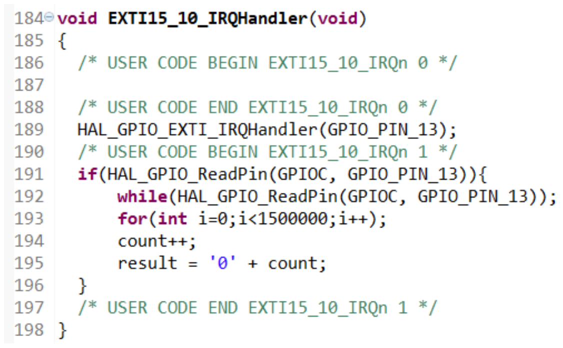
Generate Under Root ✓

Linker Settings

Minimum Heap Size	0x2000
Minimum Stack Size	0x4000

Figure 15: Heap and Stack Size

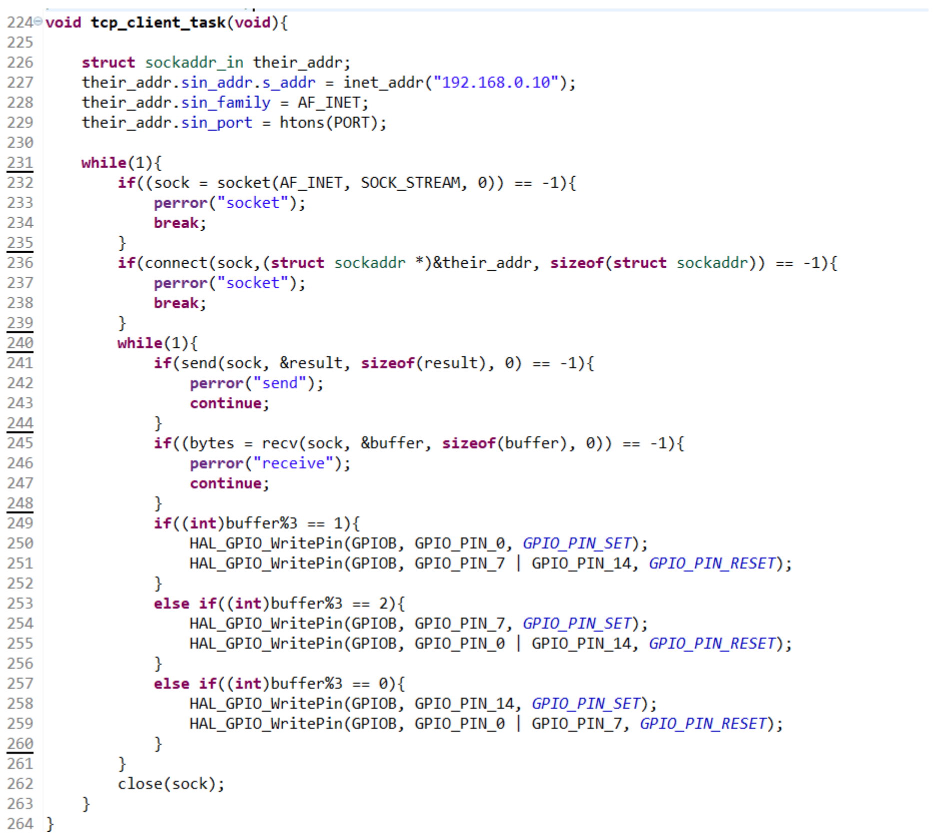
PC13 pin of STM32 Nucleo-F207ZG board is user button. In SGTM32CUBEMX, we defined this pin as external interrupt. Each time the button is pressed, the counter is incremented. The relevant code is as in figure 16 below. The counter value is summed with '0' and converted to char.



```c
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */

  /* USER CODE END EXTI15_10_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */
  if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13)){
      while(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13));
      for(int i=0;i<1500000;i++);
      count++;
      result = '0' + count;
  }
  /* USER CODE END EXTI15_10_IRQn 1 */
}
```

Figure 16: External Interrupt function of User Button (PC13)

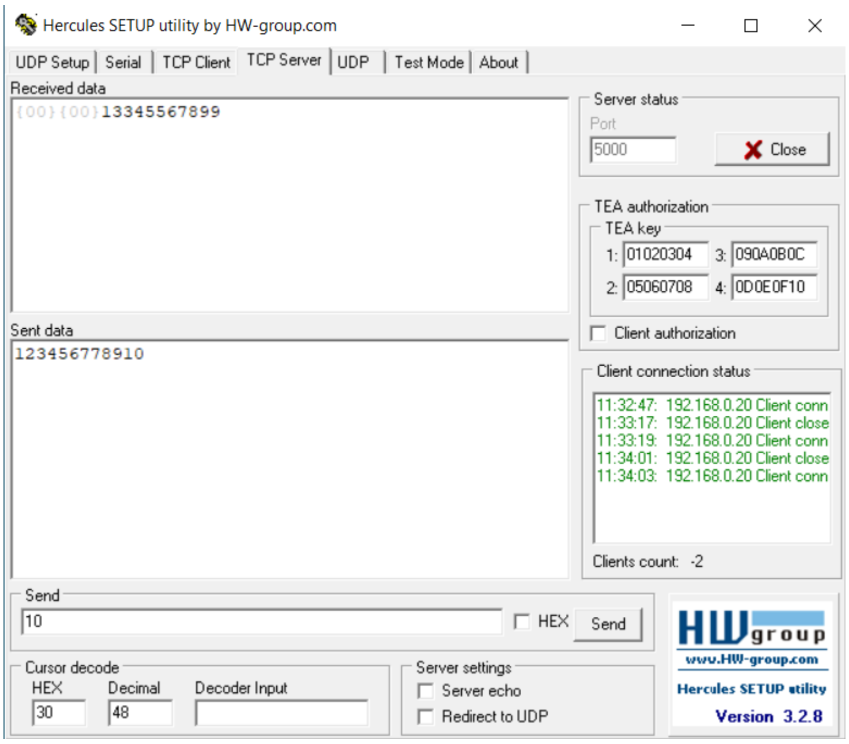
Firstly, STM32 Nucleo-F207ZG board was used as client and PC as server. As mentioned before, the HERCULES program was used on the PC. TCP client code on Nucleo side is as in figure 17.



```c
void tcp_client_task(void){

    struct sockaddr_in their_addr;
    their_addr.sin_addr.s_addr = inet_addr("192.168.0.10");
    their_addr.sin_family = AF_INET;
    their_addr.sin_port = htons(PORT);

    while(1){
        if((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1){
            perror("socket");
            break;
        }
        if(connect(sock,(struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1){
            perror("socket");
            break;
        }
        while(1){
            if(send(sock, &result, sizeof(result), 0) == -1){
                perror("send");
                continue;
            }
            if((bytes = recv(sock, &buffer, sizeof(buffer), 0)) == -1){
                perror("receive");
                continue;
            }
            if((int)buffer%3 == 1){
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7 | GPIO_PIN_14, GPIO_PIN_RESET);
            }
            else if((int)buffer%3 == 2){
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_SET);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0 | GPIO_PIN_14, GPIO_PIN_RESET);
            }
            else if((int)buffer%3 == 0){
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0 | GPIO_PIN_7, GPIO_PIN_RESET);
            }
        }
        close(sock);
    }
}
```

Figure 17: TCP Client Function

Our application has been tested with the HERCULES program on the PC side. Our Nucleo board is given an IP address of 192.168.0.20, and port number 5000. It can be seen in figure 18 below that we can access the Nucleo board when we listen to port 5000 with the HERCULES program. In the Receive data section, we can see the counter value every time we press the button. In addition, the user LEDs on the Nucleo board were successfully turned on by sending data via the HERCULES program.



Hercules SETUP utility by HW-group.com

UDP Setup | Serial | TCP Client | TCP Server | UDP | Test Mode | About

Received data

```
{00}{00}13345567899
```

Sent data

```
123456778910
```

Server status
Port: 5000
Close

TEA authorization
TEA key
1: 01020304 3: 090A0B0C
2: 05060708 4: 0D0E0F10
Client authorization

Client connection status

```
11:32:47:  192.168.0.20 Client conn
11:33:17:  192.168.0.20 Client close
11:33:19:  192.168.0.20 Client conn
11:34:01:  192.168.0.20 Client close
11:34:03:  192.168.0.20 Client conn
```

Clients count: -2

Send: 10 HEX Send

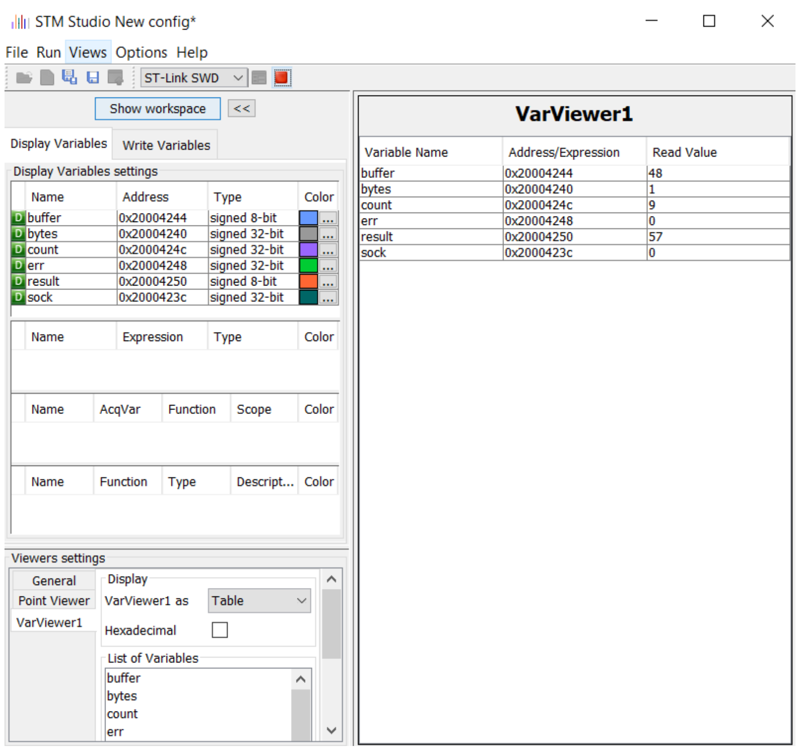
Cursor decode
HEX: 30 Decimal: 48 Decoder Input:

Server settings
Server echo
Redirect to UDP

HWgroup www.HW-group.com Hercules SETUP utility Version 3.2.8

Figure 18: HERCULES as a TCP Server

In STM Studio, the counter value sent to the PC by the Nucleo board and the value sent by the PC can be seen as a buffer in the figure 19 below.



STM Studio New config*

File Run Views Options Help

ST-Link SWD

Show workspace <<

Display Variables | Write Variables

Display Variables settings

Name	Address	Type	Color
buffer	0x20004244	signed 8-bit	🟦
bytes	0x20004240	signed 32-bit	⬜
count	0x2000424c	signed 32-bit	🟪
err	0x20004248	signed 32-bit	🟩
result	0x20004250	signed 8-bit	🟧
sock	0x2000423c	signed 32-bit	🟦

Name	Expression	Type	Color

Name	AcqVar	Function	Scope	Color

Name	Function	Type	Descript...	Color

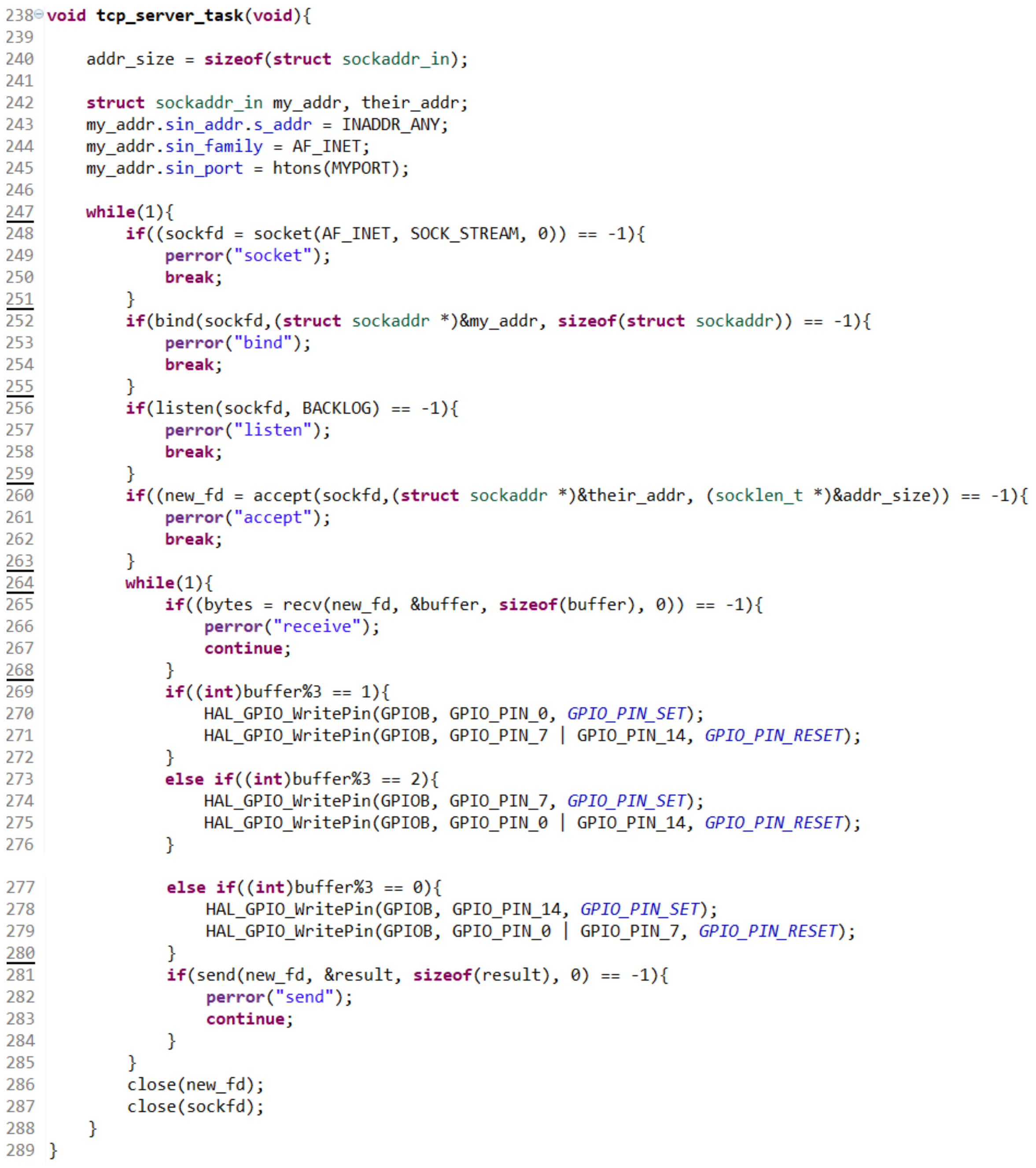
Viewers settings
General | Point Viewer | VarViewer1
Display
VarViewer1 as: Table
Hexadecimal
List of Variables: buffer, bytes, count, err

VarViewer1

Variable Name	Address/Expression	Read Value
buffer	0x20004244	48
bytes	0x20004240	1
count	0x2000424c	9
err	0x20004248	0
result	0x20004250	57
sock	0x2000423c	0

Figure 19: The Nucleo's Variables in STM Studio

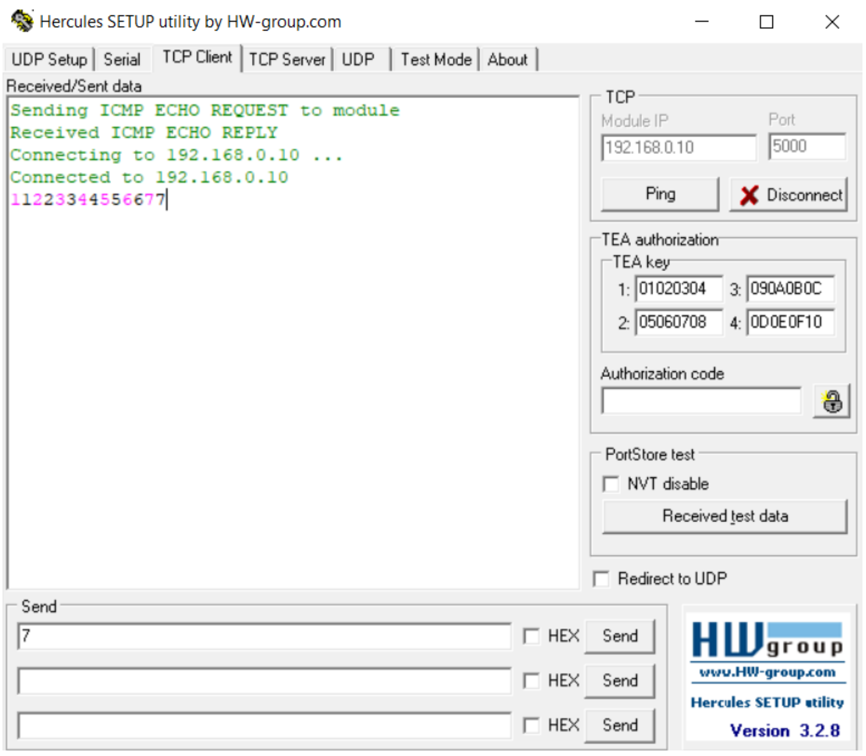
In the next stage of the project, the STM32 Nucleo-F207ZG card was used as a server and the PC side as a client, and communication was successfully achieved. TCP server code on Nucleo side is as in figure 20.



```
void tcp_server_task(void){

    addr_size = sizeof(struct sockaddr_in);

    struct sockaddr_in my_addr, their_addr;
    my_addr.sin_addr.s_addr = INADDR_ANY;
    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(MYPORT);

    while(1){
        if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1){
            perror("socket");
            break;
        }
        if(bind(sockfd,(struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1){
            perror("bind");
            break;
        }
        if(listen(sockfd, BACKLOG) == -1){
            perror("listen");
            break;
        }
        if((new_fd = accept(sockfd,(struct sockaddr *)&their_addr, (socklen_t *)&addr_size)) == -1){
            perror("accept");
            break;
        }
        while(1){
            if((bytes = recv(new_fd, &buffer, sizeof(buffer), 0)) == -1){
                perror("receive");
                continue;
            }
            if((int)buffer%3 == 1){
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7 | GPIO_PIN_14, GPIO_PIN_RESET);
            }
            else if((int)buffer%3 == 2){
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_SET);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0 | GPIO_PIN_14, GPIO_PIN_RESET);
            }

            else if((int)buffer%3 == 0){
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0 | GPIO_PIN_7, GPIO_PIN_RESET);
            }
            if(send(new_fd, &result, sizeof(result), 0) == -1){
                perror("send");
                continue;
            }
        }
        close(new_fd);
        close(sockfd);
    }
}
```

Figure 20: TCP Server Function

The Nucleo board has been given the IP address of 192.168.0.10 and the port numbered 5000. In figure 21 below, it can be seen that the PC side is used as a client in the HERCULES program and successfully connected with the Nucleo board. The pink-colored data represents the data sent to the Nucleo board using the HERCULES software, while the black-colored data represents the data sent and successfully received from the Nucleo board to the PC.



Hercules SETUP utility by HW-group.com

UDP Setup | Serial | TCP Client | TCP Server | UDP | Test Mode | About

Received/Sent data

```
Sending ICMP ECHO REQUEST to module
Received ICMP ECHO REPLY
Connecting to 192.168.0.10 ...
Connected to 192.168.0.10
1122334455667 7
```

TCP — Module IP: 192.168.0.10 Port: 5000

Ping | Disconnect

TEA authorization — TEA key 1: 01020304 3: 090A0B0C 2: 05060708 4: 0D0E0F10

Authorization code

PortStore test — NVT disable — Received test data

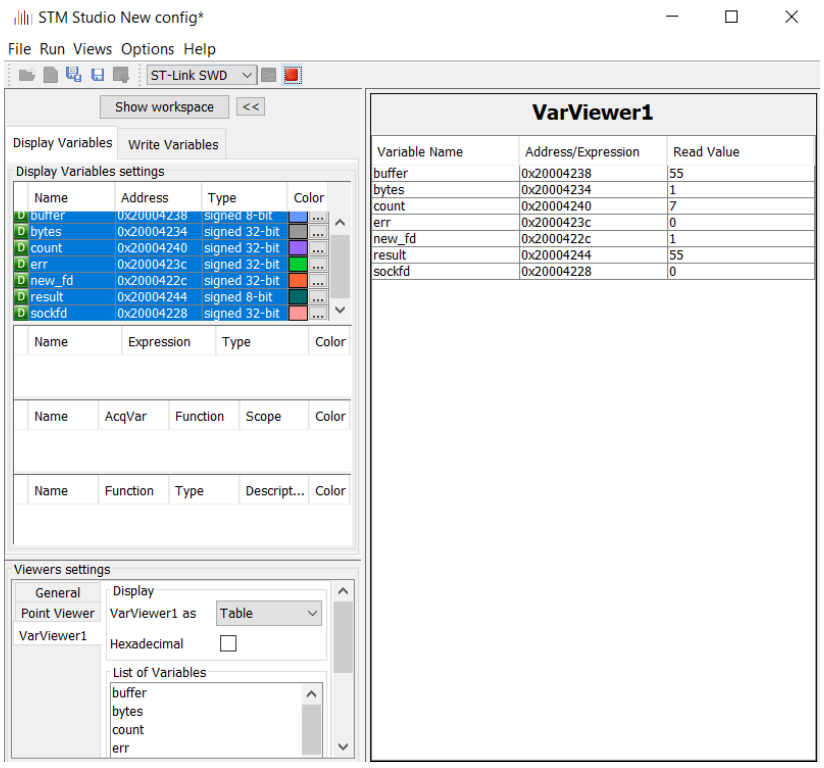
Redirect to UDP

Send: 7 — HEX — Send

Version 3.2.8

Figure 21: HERCULES as a TCP Client

In STM Studio, the counter value sent to the PC by the Nucleo board and the value sent by the PC can be seen as a buffer in the figure 22 below.



STM Studio New config*

File Run Views Options Help

ST-Link SWD

Show workspace <<

Display Variables | Write Variables

Display Variables settings

Name	Address	Type	Color
buffer	0x20004238	signed 8-bit	🟦
bytes	0x20004234	signed 32-bit	⬛
count	0x20004240	signed 32-bit	🟪
err	0x2000423c	signed 32-bit	🟩
new_fd	0x2000422c	signed 32-bit	🟧
result	0x20004244	signed 8-bit	🟦
sockfd	0x20004228	signed 32-bit	🟥

VarViewer1

Variable Name	Address/Expression	Read Value
buffer	0x20004238	55
bytes	0x20004234	1
count	0x20004240	7
err	0x2000423c	0
new_fd	0x2000422c	1
result	0x20004244	55
sockfd	0x20004228	0

Viewers settings: General | Point Viewer | VarViewer1

Display — VarViewer1 as: Table — Hexadecimal: ☐

List of Variables: buffer, bytes, count, err

Figure 22: The Nucleo's Variables in STM Studio