

**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).

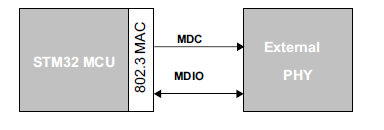
***SMI (Station management interface)***

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.

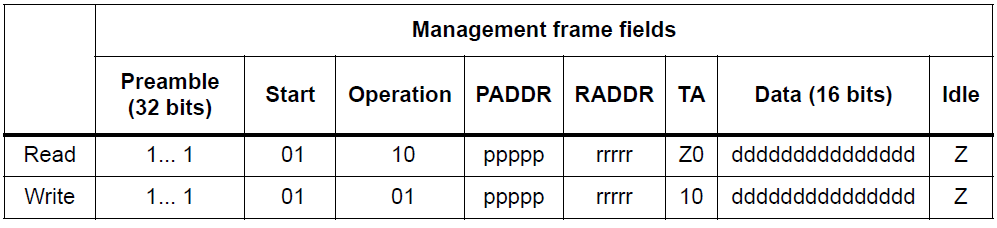
**MDIO:** data input/output bitstream to transfer status information to/from the PHY device synchronously with the MDC clock signal



*Figure-1: SMI interface signals*

**SMI frame format:**

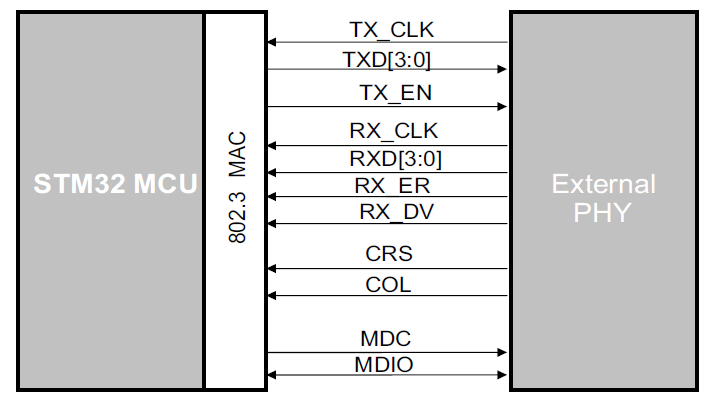
*Table 1: Management frame format:*



***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. |

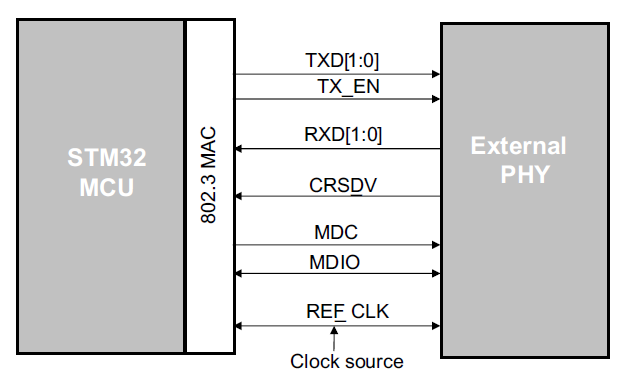
***RMII (Reduced media-independent interface)***

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.

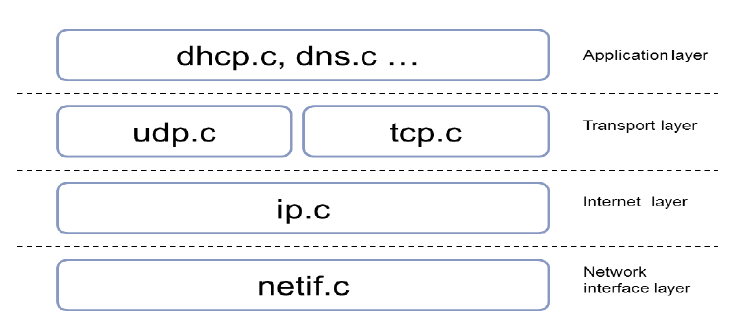
lwIP supports the following protocols:

* 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*

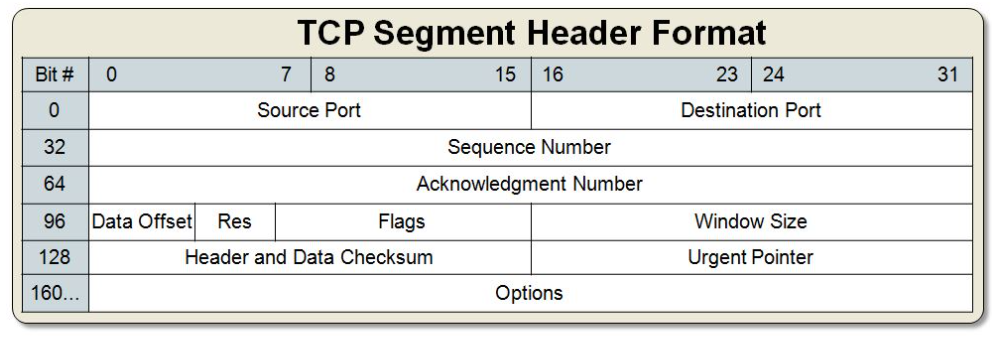
**TCP (Transmission Control Protocol)**

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.

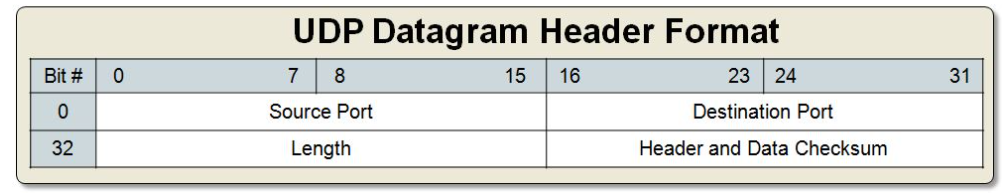
**UDP (User Datagram Protocol)**

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.

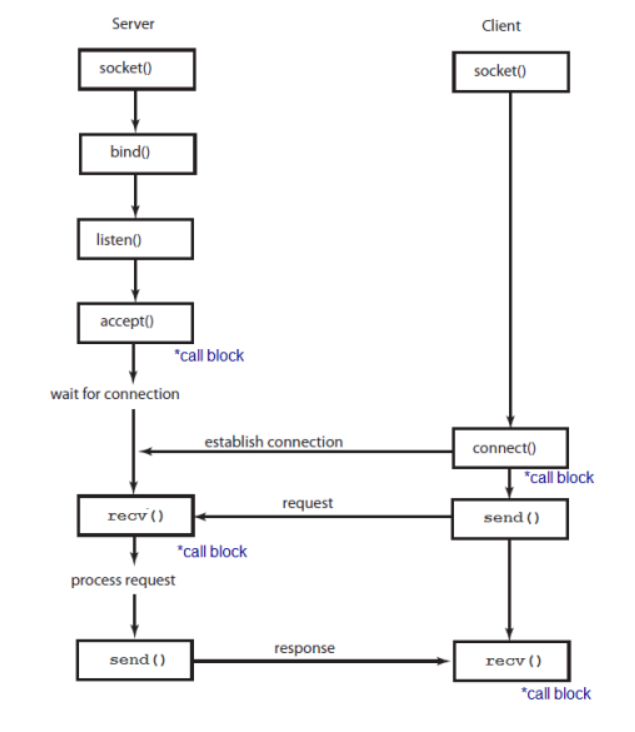
If we mail several letters, we can't be sure that they arrive in the same order, or even that they reached their destination at all.



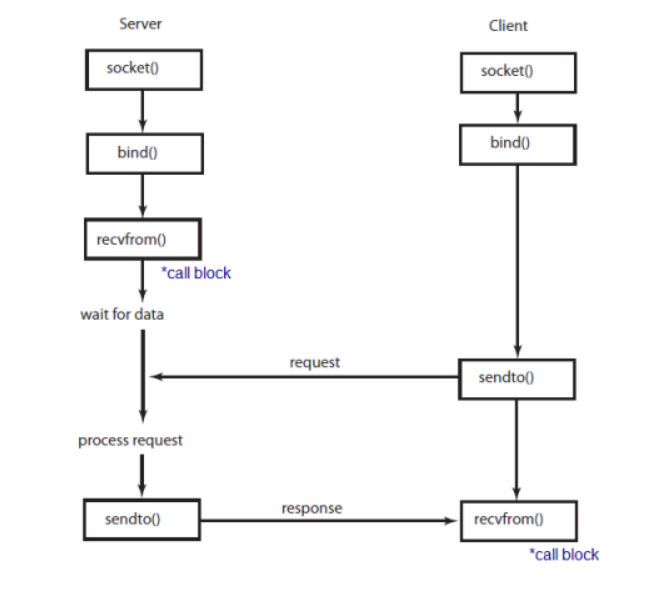
*Figure 4: TCP Format*



*Figure 5: UDP Format*



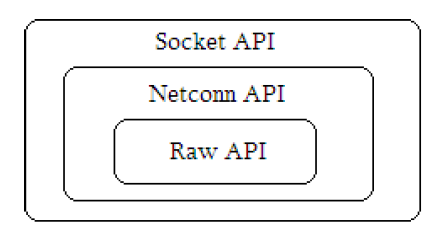
*Figure 6: TCP Communication*



*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.



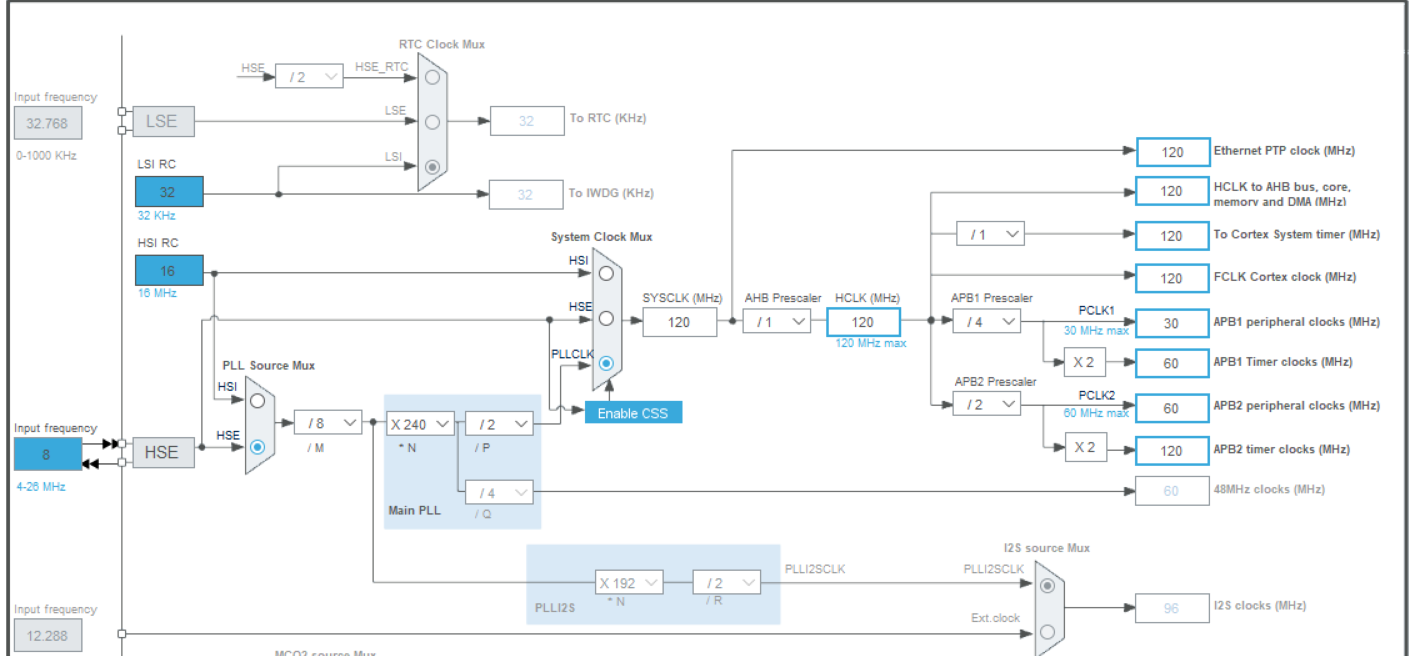
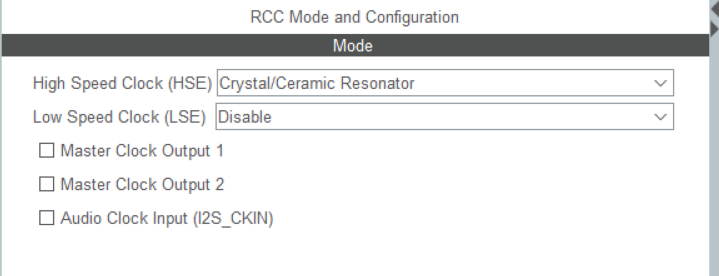
*Figure 8: LwIP API*

*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.

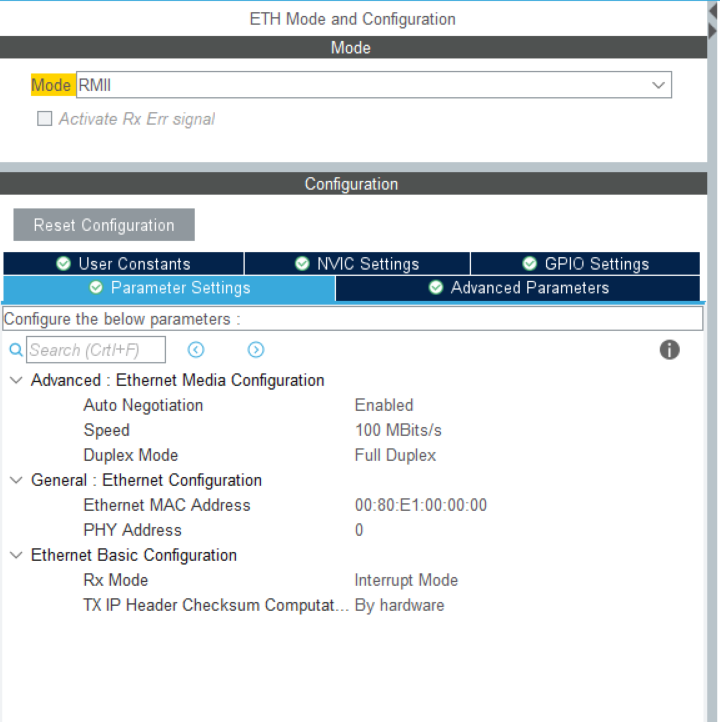




*Figure 9: The clock configuration of STM32 Nucleo-F207ZG*

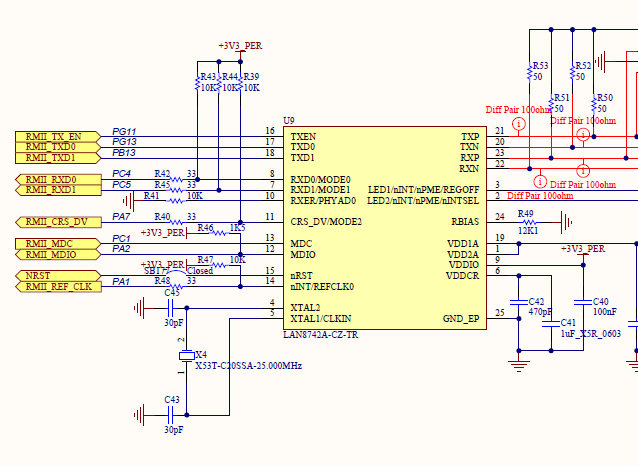


Ethernet must be activated from the Connectivity tab. RMII (Reduced media-independent interface) should be selected as the mode. The PHY address should be set to 0 or 1 depending on the board used. For the STM32 Nucleo-F207ZG board, this value is 0. The appropriate value can be checked from the schematic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



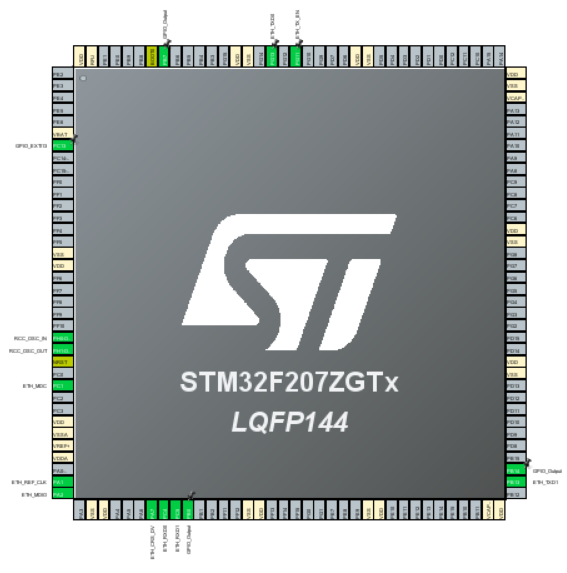
*Figure 10: Ethernet Mode and Configuration*

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*

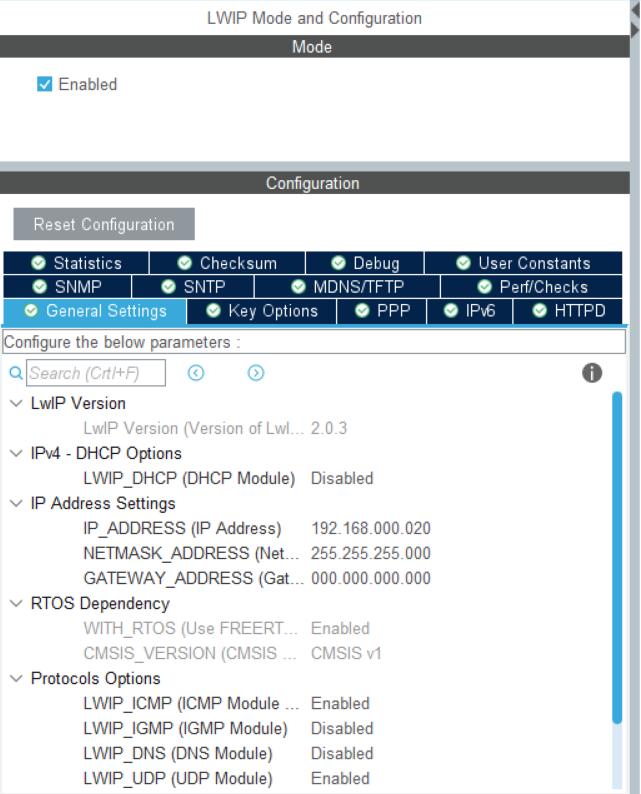
Since we will use 3 user leds and 1 user button of the nucleo card in our project, we define the relevant leds as input and output. All of our pin definitions are as in figure-12 below.



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

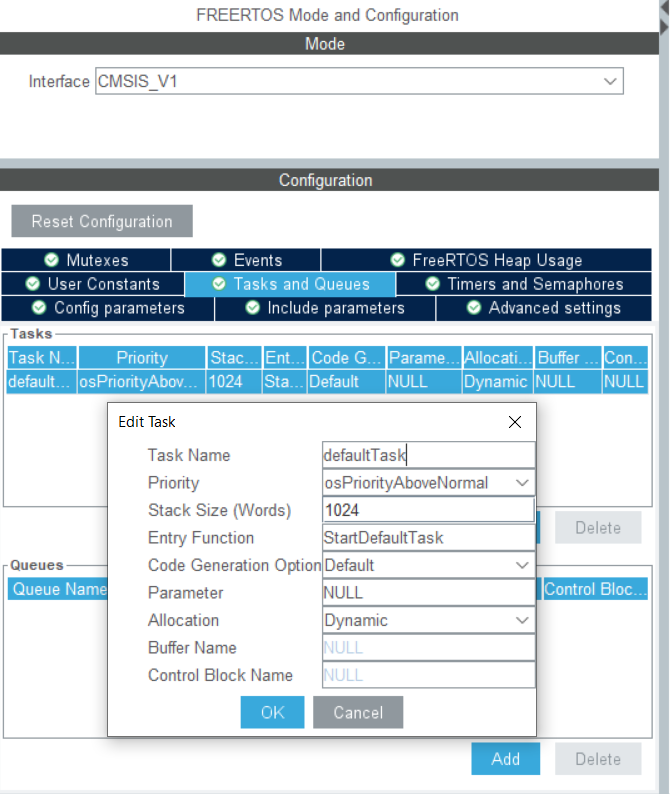
Since we will use the Socket API in our project, we need to use FREERTOS. Also, since we need to use LwIP, additional settings must be made for the use of FREERTOS in LwIP. However, if we activate LwIP first and then activate FREERTOS, STM32CUBEMX will make the necessary adjustments for FREERTOS in LwIP.

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.



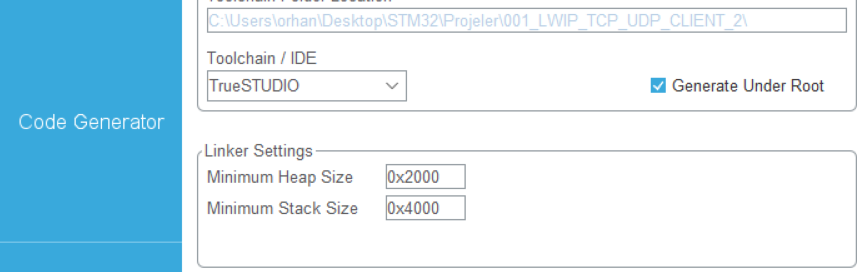
*Figure 13: LWIP Mode and Configuration*

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.



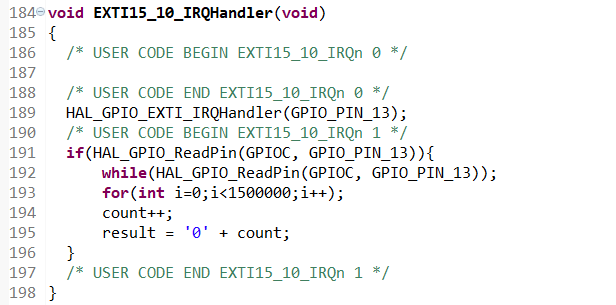
*Figure 14: FREERTOS Mode and Configuration*

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



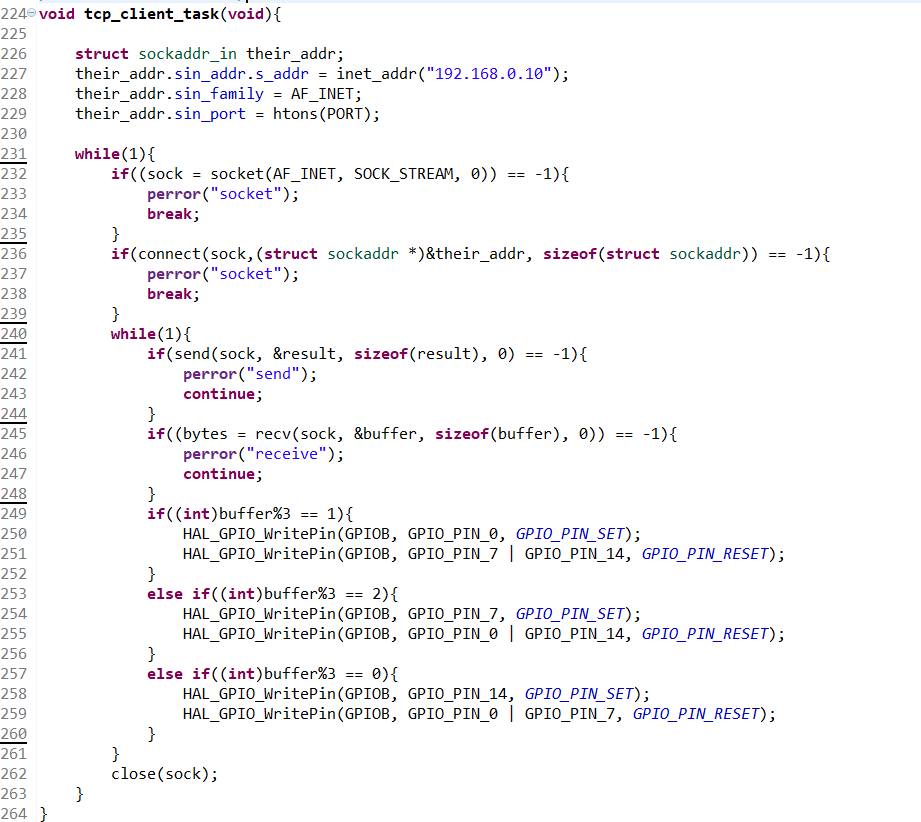
*Figure 15: Heap and Stack Size*

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.



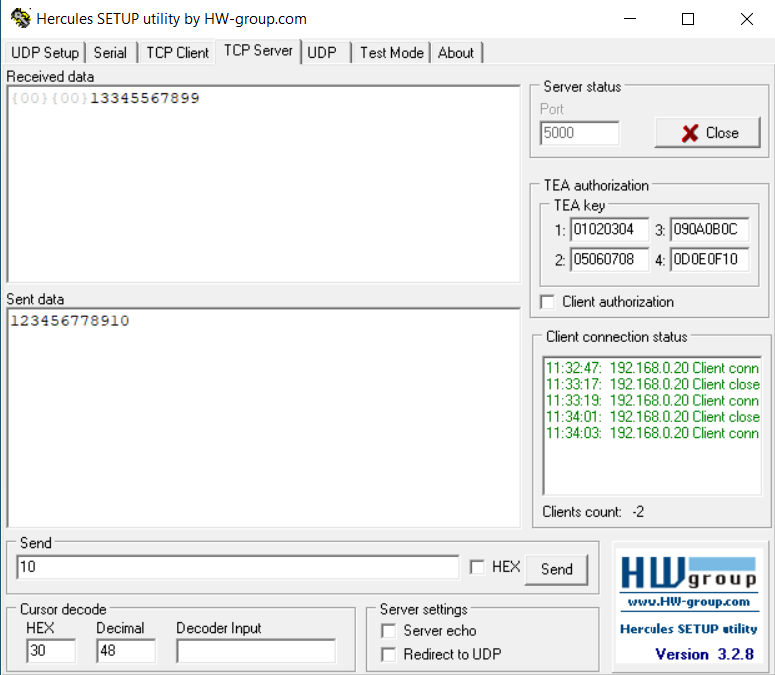
*Figure 16: External Interrupt function of User Button (PC13)*

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.



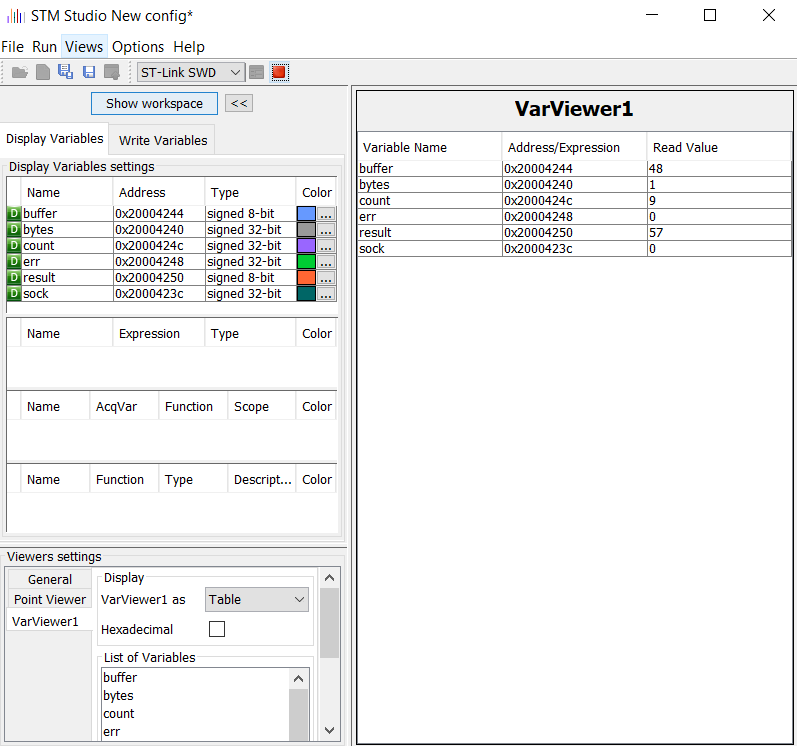
*Figure 17: TCP Client Function*

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.



*Figure 18: HERCULES as a TCP Server*

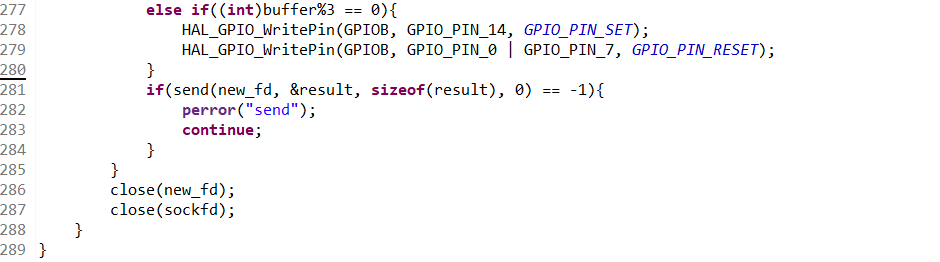
In STM Studio, the counter value sent to the PC by the Nucleo board and the value sent by the PC can be seen as a buffer in the figure 19 below.



*Figure 19: The Nucleo’s Variables in STM Studio*

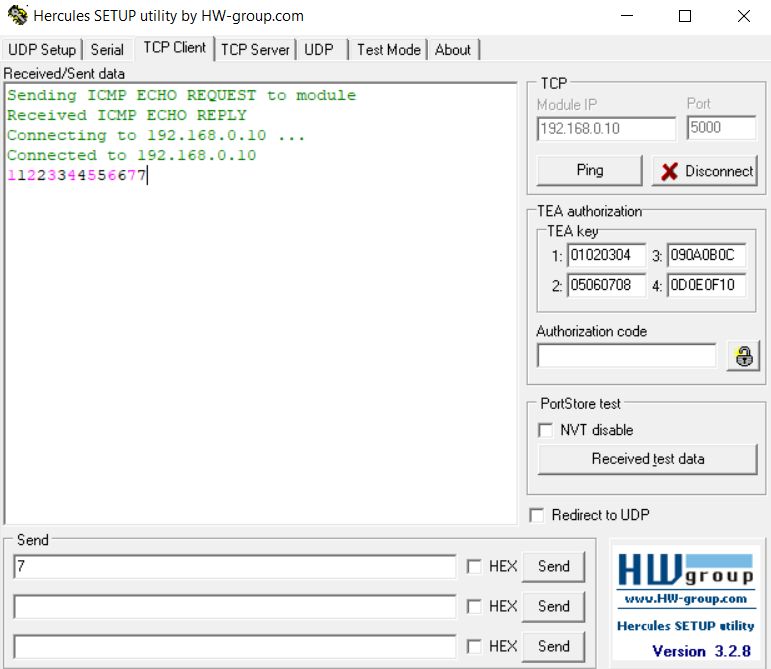
In the next stage of the project, the STM32 Nucleo-F207ZG card was used as a server and the PC side as a client, and communication was successfully achieved. TCP server code on Nucleo side is as in figure 20.





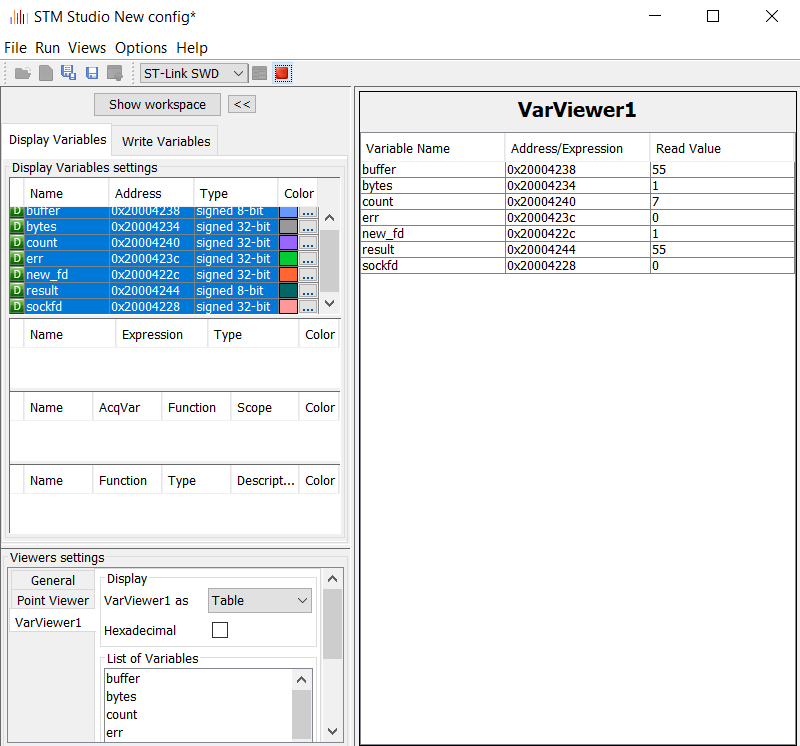
*Figure 20: TCP Server Function*

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.



*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.



*Figure 22: The Nucleo’s Variables in STM Studio*