МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **КНІТ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 5
З дисципліни: *"Програмування мікроконтролерів"*
На тему: "Робота з Ethernet на базі мікроконтролера STM32F407"

Лектор:
доц. каф. ПЗ
Марусенкова Т.А.

Виконав:
ст. гр. ПЗ-42
Бурець В.В.
Ільчук А.В.

Прийняв:
доц. каф. ПЗ
Марусенкова Т.А.

« ____ » _____ 2022 р.

∑= _____ _____.

Львів-2022

Тема роботи: Робота з Ethernet на базі мікроконтролера STM32F407.

Мета роботи: Навчитись створювати мережеве підключення з використанням Ethernet.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Теоретичні відомості Ethernet – найпопулярніший протокол кабельних комп'ютерних мереж, що працює на фізичному та канальному рівні мережевої моделі OSI. Саме цей протокол є найпоширенішим протоколом у сучасних локальних комп'ютерних мережах. Використання протоколу Ethernet повністю визначає фізичний рівень, а саме тип середовища по якому відбувається передача даних, тип роз'ємів для підключення провідника, рівні напруги і вид модуляції з допомогою якої передається інформація. Окрім, фізичного рівня стандарт (протокол) Ethernet визначає і канальний рівень. На початку трансляції пакета фізичний рівень генерує преамбулу у вигляді послідовності одиниць та нулів для синхронізації приймача і передавача сигналу. Після синхронізації генерується команд початку пакета (SFD). Як правило цю операцію виконує фізичний рівень про, яку користувач може нічого не знати. Однак, користувач повинен вказати MAC адресу призначення пакета та MAC адресу джерела пакета. Після MAC адресів надсилається 2 байти, які вказують довжину трансльованого пакету. Мінімальна довжина пакету складає 46 байт, при надсиланні даних меншого обсягу контролер STM32 доповнить це повідомлення пустими комірками. Для перевірки достовірності отриманої інформації використовується розрахунок контрольної суми за алгоритмом CRC32 ця операція виконується апаратно.

ЗАВДАННЯ

1. Використовуючи приклади коду, що представлені у методичних вказівках адаптувати надані коди для HAL бібліотеки.

Індивідуальне завдання: 4. Створити TCP сервер для роботи з N клієнтами; HAL

ХІД РОБОТИ

Ініціалізація Ethernet

Для ініціалізації Ethernet було використано мод RMII(Reduced Media Independent Interface). Це скорочений незалежний від середовища передачі інтерфейс. Який представляє собою стандартний інтерфейс, який використовує скорочений набір сигналів інтерфейса. При цьому розрядність шини RXD і TXD

скорочується в двоє у порівнянні з MII, але при цьому вдвічі піднімається частота синхронізації MAC.

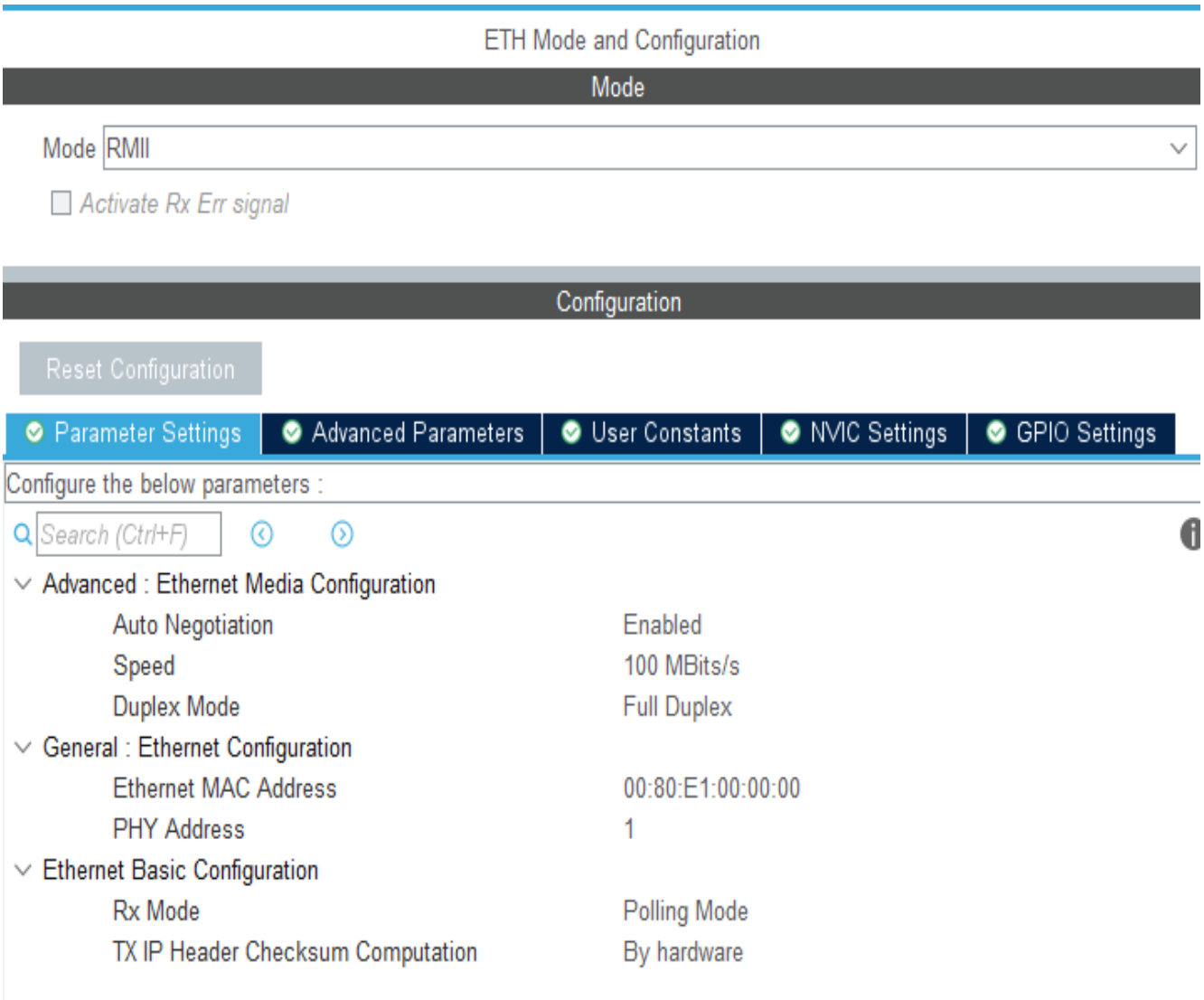

Рис. 1. Ініціалізація Ethernet

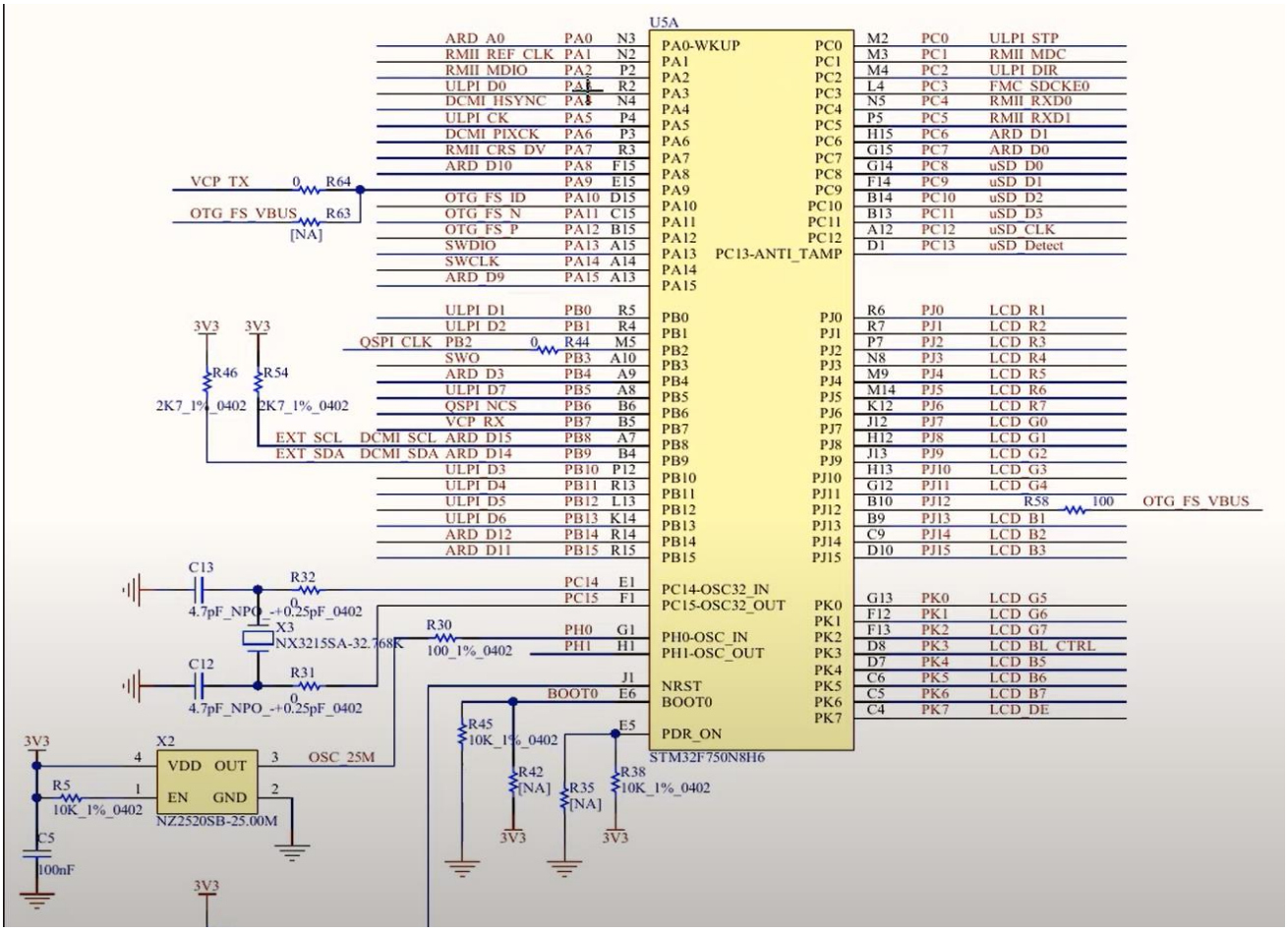

Рис. 2. Схема пінів з технічної документації

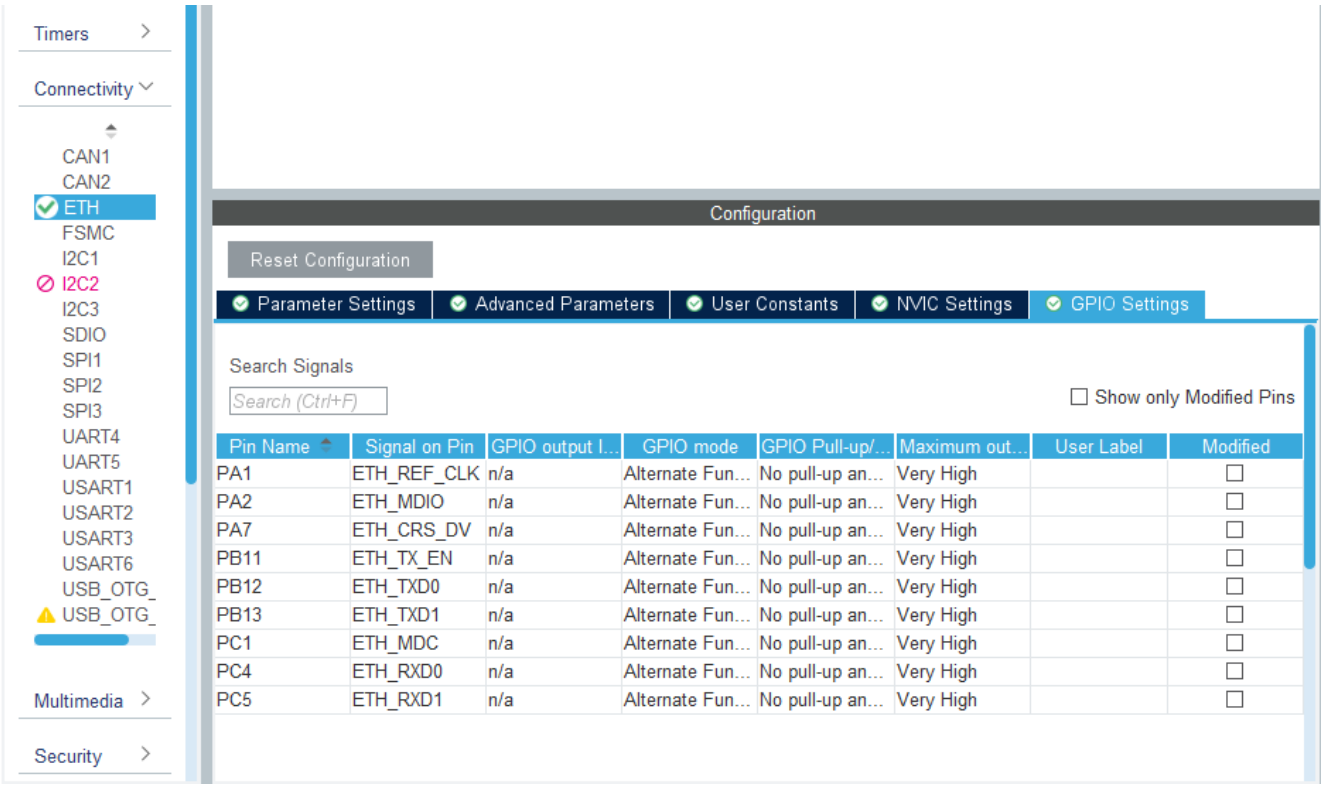

Рис. 3. Налаштування пінів у проекті

Ініціалізація LWIP

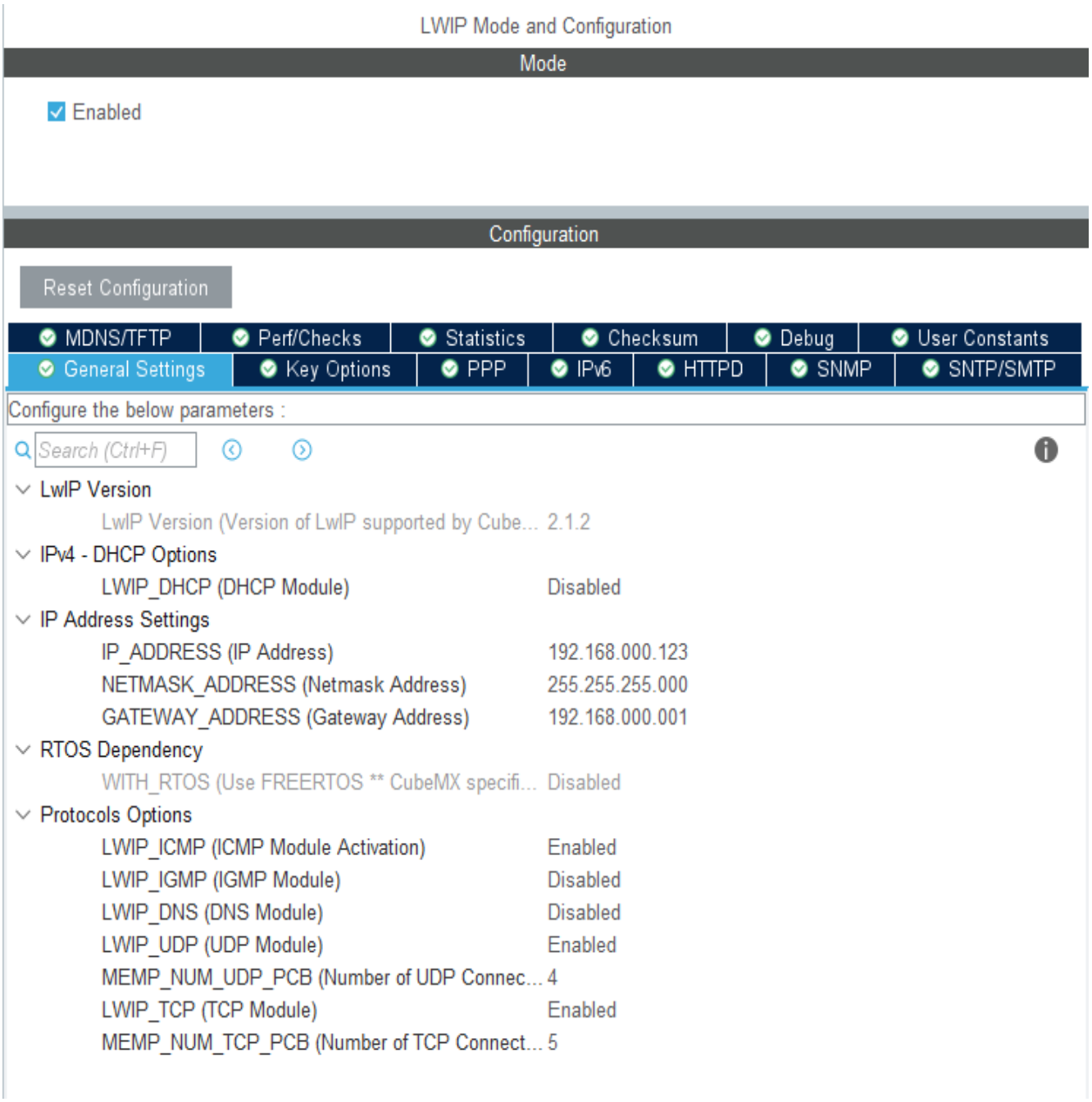

Рис. 4. Налаштування LwIP

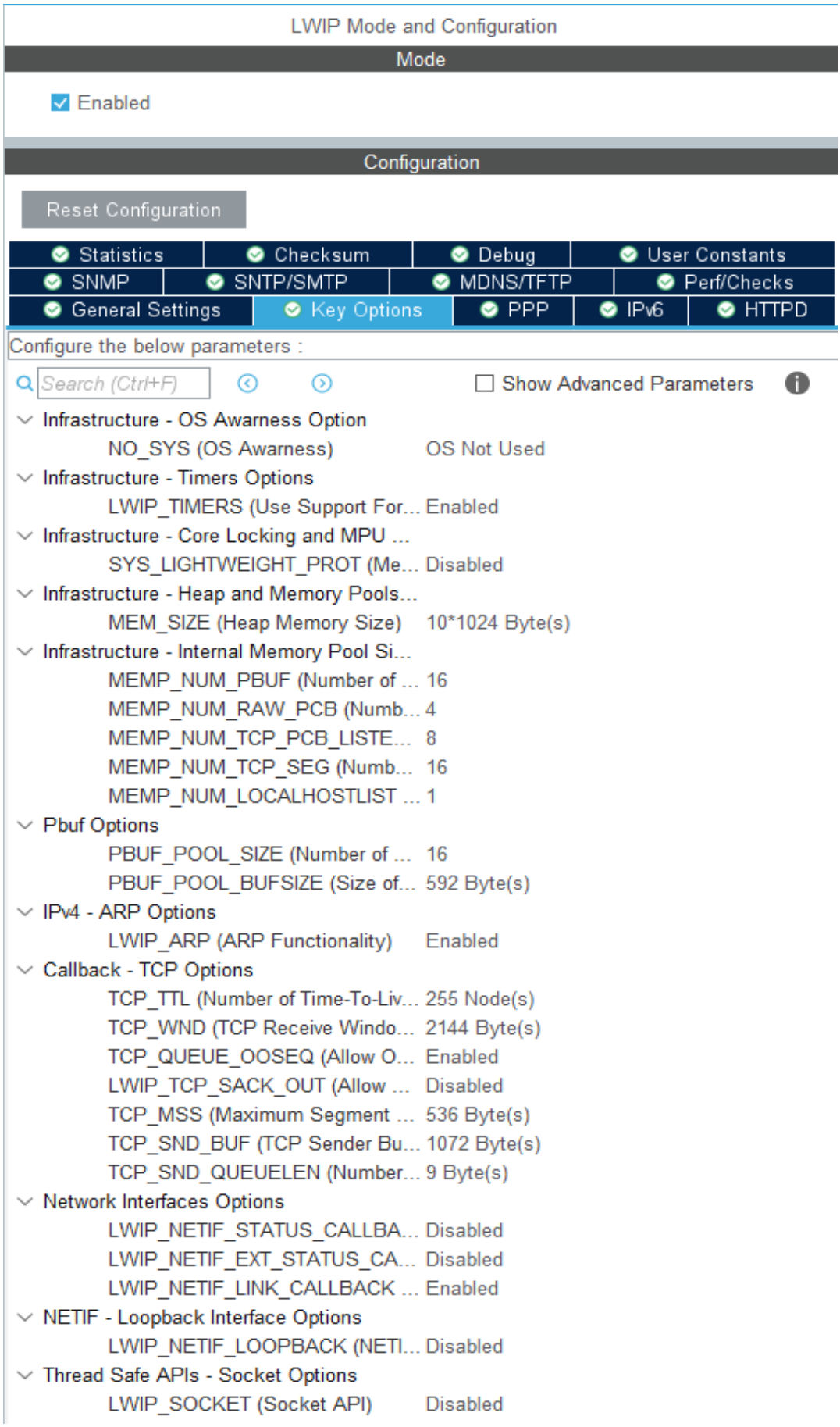

Рис. 4. Налаштування LwIP (Key options)

Файл Main.c

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "lwip.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "tcpServerRAW.h"
/* USER CODE END Includes */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
extern struct netif gnetif;
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_LWIP_Init();
  /* USER CODE BEGIN 2 */
  tcp_server_init();
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
        ethernetif_input(&gnetif);

        sys_check_timeouts();
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
```

```
  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 23;
  RCC_OscInitStruct.PLL.PLLN = 168;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
```

```
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Файл tcpServerRAW.c

```
#include "tcpserverRAW.h"

#include "lwip/tcp.h"

#define N 10

enum tcp_server_states
{
  ES_NONE = 0,
  ES_ACCEPTED,
  ES_RECEIVED,
  ES_CLOSING
};

/* structure for maintaining connection infos to be passed as argument
   to LwIP callbacks*/
struct tcp_server_struct
{
  u8_t state;             /* current connection state */
  u8_t retries;
  struct tcp_pcb *pcb;    /* pointer on the current tcp_pcb */
  struct pbuf *p;         /* pointer on the received/to be transmitted pbuf */
};


static err_t tcp_server_accept(void *arg, struct tcp_pcb *newpcb, err_t err);
static err_t tcp_server_recv(void *arg, struct tcp_pcb *tpcb, struct pbuf *p, err_t err);
static void tcp_server_error(void *arg, err_t err);
static err_t tcp_server_poll(void *arg, struct tcp_pcb *tpcb);
static err_t tcp_server_sent(void *arg, struct tcp_pcb *tpcb, u16_t len);
static void tcp_server_send(struct tcp_pcb *tpcb, struct tcp_server_struct *es);
static void tcp_server_connection_close(struct tcp_pcb *tpcb, struct tcp_server_struct *es);

static void tcp_server_handle (struct tcp_pcb *tpcb, struct tcp_server_struct *es);

int connections_count = 0;

void tcp_server_init(void)
{
        /* 1. create new tcp pcb */
        struct tcp_pcb *tpcb;

        tpcb = tcp_new();

        err_t err;
```

```
        /* 2. bind _pcb to port 10 ( protocol) */
        ip_addr_t myIPADDR;
        IP_ADDR4(&myIPADDR, 192, 168, 0, 123);
        err = tcp_bind(tpcb, &myIPADDR, 10);

        connections_count = 0;

        if (err == ERR_OK)
        {
                /* 3. start tcp listening for _pcb */
                tpcb = tcp_listen(tpcb);

                /* 4. initialize LwIP tcp_accept callback function */
                tcp_accept(tpcb, tcp_server_accept);
        }
        else
        {
                /* deallocate the pcb */
                memp_free(MEMP_TCP_PCB, tpcb);
        }
}

/**
  * This function is the implementation of tcp_accept LwIP callback
  */
static err_t tcp_server_accept(void *arg, struct tcp_pcb *newpcb, err_t err)
{
  err_t ret_err;
  struct tcp_server_struct *es;

  LWIP_UNUSED_ARG(arg);
  LWIP_UNUSED_ARG(err);

  /* set priority for the newly accepted tcp connection newpcb */
  tcp_setprio(newpcb, TCP_PRIO_MIN);

  /* allocate structure es to maintain tcp connection information */
  es = (struct tcp_server_struct *)mem_malloc(sizeof(struct tcp_server_struct));

  if(connections_count >= N)
  {
        tcp_server_connection_close(newpcb, es);

        ret_err = ERR_MEM;

        return ret_err;
  }

  if (es != NULL)
  {
        connections_count += 1;

    es->state = ES_ACCEPTED;
    es->pcb = newpcb;
    es->retries = 0;
    es->p = NULL;

    /* pass newly allocated es structure as argument to newpcb */
    tcp_arg(newpcb, es);

    /* initialize lwip tcp_recv callback function for newpcb  */
    tcp_recv(newpcb, tcp_server_recv);

    /* initialize lwip tcp_err callback function for newpcb  */
    tcp_err(newpcb, tcp_server_error);
```

```
    /* initialize lwip tcp_poll callback function for newpcb */
    tcp_poll(newpcb, tcp_server_poll, 0);

    ret_err = ERR_OK;
  }
  else
  {
    /*  close tcp connection */
    tcp_server_connection_close(newpcb, es);
    /* return memory error */
    ret_err = ERR_MEM;
  }
  return ret_err;
}


/**
  *  This function is the implementation for tcp_recv LwIP callback
  */
static err_t tcp_server_recv(void *arg, struct tcp_pcb *tpcb, struct pbuf *p, err_t err)
{
  struct tcp_server_struct *es;
  err_t ret_err;

  LWIP_ASSERT("arg != NULL",arg != NULL);

  es = (struct tcp_server_struct *)arg;

  /* if we receive an empty tcp frame from client => close connection */
  if (p == NULL)
  {
    /* remote host closed connection */
    es->state = ES_CLOSING;
    if(es->p == NULL)
    {
       /* we're done sending, close connection */
       tcp_server_connection_close(tpcb, es);
    }
    else
    {
      /* we're not done yet */
      /* acknowledge received packet */
      tcp_sent(tpcb, tcp_server_sent);

      /* send remaining data*/
      tcp_server_send(tpcb, es);
    }
    ret_err = ERR_OK;
  }
  /* else : a non empty frame was received from client but for some reason err != ERR_OK */
  else if(err != ERR_OK)
  {
    /* free received pbuf*/
    if (p != NULL)
    {
      es->p = NULL;
      pbuf_free(p);
    }
    ret_err = err;
  }
  else if(es->state == ES_ACCEPTED)
  {
    /* first data chunk in p->payload */
    es->state = ES_RECEIVED;
```

```
    /* store reference to incoming pbuf (chain) */
    es->p = p;

    /* initialize LwIP tcp_sent callback function */
    tcp_sent(tpcb, tcp_server_sent);

    /* handle the received data */
    tcp_server_handle(tpcb, es);

    ret_err = ERR_OK;
  }
  else if (es->state == ES_RECEIVED)
  {
    /* more data received from client and previous data has been already sent*/
    if(es->p == NULL)
    {
      es->p = p;

      /* handle the received data */
      tcp_server_handle(tpcb, es);
    }
    else
    {
      struct pbuf *ptr;

      /* chain pbufs to the end of what we recv'ed previously  */
      ptr = es->p;
      pbuf_chain(ptr,p);
    }
    ret_err = ERR_OK;
  }
  else if(es->state == ES_CLOSING)
  {
    /* odd case, remote side closing twice, trash data */
    tcp_recved(tpcb, p->tot_len);
    es->p = NULL;
    pbuf_free(p);
    ret_err = ERR_OK;
  }
  else
  {
    /* unknown es->state, trash data  */
    tcp_recved(tpcb, p->tot_len);
    es->p = NULL;
    pbuf_free(p);
    ret_err = ERR_OK;
  }
  return ret_err;
}

/**
  * This function implements the tcp_err callback function (called when a fatal tcp_connection error occurs.)
  */
static void tcp_server_error(void *arg, err_t err)
{
  struct tcp_server_struct *es;

  LWIP_UNUSED_ARG(err);

  es = (struct tcp_server_struct *)arg;
  if (es != NULL)
  {
    /*  free es structure */
    mem_free(es);
  }
}
```

```
/**
  * This function implements the tcp_poll LwIP callback function
  */
static err_t tcp_server_poll(void *arg, struct tcp_pcb *tpcb)
{
  err_t ret_err;
  struct tcp_server_struct *es;

  es = (struct tcp_server_struct *)arg;
  if (es != NULL)
  {
    if (es->p != NULL)
    {
      tcp_sent(tpcb, tcp_server_sent);
      /* there is a remaining pbuf (chain) , try to send data */
      tcp_server_send(tpcb, es);
    }
    else
    {
      /* no remaining pbuf (chain)  */
      if(es->state == ES_CLOSING)
      {
        /*  close tcp connection */
        tcp_server_connection_close(tpcb, es);
      }
    }
    ret_err = ERR_OK;
  }
  else
  {
    /* nothing to be done */
    tcp_abort(tpcb);
    ret_err = ERR_ABRT;
  }
  return ret_err;
}

/**
  * This function implements the tcp_sent LwIP callback (called when ACK is received from remote host for sent
data)
  */
static err_t tcp_server_sent(void *arg, struct tcp_pcb *tpcb, u16_t len)
{
  struct tcp_server_struct *es;

  LWIP_UNUSED_ARG(len);

  es = (struct tcp_server_struct *)arg;
  es->retries = 0;

  if(es->p != NULL)
  {
    /* still got pbufs to send */
    tcp_sent(tpcb, tcp_server_sent);
    tcp_server_send(tpcb, es);
  }
  else
  {
    /* if no more data to send and client closed connection*/
    if(es->state == ES_CLOSING)
      tcp_server_connection_close(tpcb, es);
  }
  return ERR_OK;
}
```

```
/**
  * This function is used to send data for tcp connection
  */
static void tcp_server_send(struct tcp_pcb *tpcb, struct tcp_server_struct *es)
{
  struct pbuf *ptr;
  err_t wr_err = ERR_OK;

  while ((wr_err == ERR_OK) &&
         (es->p != NULL) &&
         (es->p->len <= tcp_sndbuf(tpcb)))
  {

    /* get pointer on pbuf from es structure */
    ptr = es->p;

    /* enqueue data for transmission */
    wr_err = tcp_write(tpcb, ptr->payload, ptr->len, 1);

    if (wr_err == ERR_OK)
    {
      u16_t plen;
      u8_t freed;

      plen = ptr->len;

      /* continue with next pbuf in chain (if any) */
      es->p = ptr->next;

      if(es->p != NULL)
      {
        /* increment reference count for es->p */
        pbuf_ref(es->p);
      }

     /* chop first pbuf from chain */
      do
      {
        /* try hard to free pbuf */
        freed = pbuf_free(ptr);
      }
      while(freed == 0);
     /* we can read more data now */
     tcp_recved(tpcb, plen);
   }
   else if(wr_err == ERR_MEM)
   {
      /* we are low on memory, try later / harder, defer to poll */
     es->p = ptr;
   }
   else
   {
     /* other problem ?? */
   }
  }
}

/**
  * This functions closes the tcp connection
  */
static void tcp_server_connection_close(struct tcp_pcb *tpcb, struct tcp_server_struct *es)
{

  /* remove all callbacks */
  tcp_arg(tpcb, NULL);
```

```
  tcp_sent(tpcb, NULL);
  tcp_recv(tpcb, NULL);
  tcp_err(tpcb, NULL);
  tcp_poll(tpcb, NULL, 0);

  /* delete es structure */
  if (es != NULL)
  {
    mem_free(es);
  }

  /* close tcp connection */
  tcp_close(tpcb);
}

/* Handle the incoming TCP Data */
static void tcp_server_handle (struct tcp_pcb *tpcb, struct tcp_server_struct *es)
{
        struct tcp_server_struct *esTx = 0;

        /* get the Remote IP */
        ip4_addr_t inIP = tpcb->remote_ip;
        uint16_t inPort = tpcb->remote_port;

        /* Extract the IP */
        char *remIP = ipaddr_ntoa(&inIP);

        esTx->state = es->state;
        esTx->pcb = es->pcb;
        esTx->p = es->p;

        char buf[100];
        memset (buf, '\0', 100);

        strncpy(buf, (char *)es->p->payload, es->p->tot_len);
        strcat (buf, " | Valentyn has received your message.");


        esTx->p->payload = (void *)buf;
        esTx->p->tot_len = (es->p->tot_len - es->p->len) + strlen (buf);
        esTx->p->len = strlen (buf);

        tcp_server_send(tpcb, esTx);

        pbuf_free(es->p);

}
```

Функція для ініціалізації Ethernet

```
void HAL_ETH_MspInit(ETH_HandleTypeDef* ethHandle)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(ethHandle->Instance==ETH)
  {
    /* Enable Peripheral clock */
    __HAL_RCC_ETH_CLK_ENABLE();

    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    /**ETH GPIO Configuration
    PC1     ------> ETH_MDC
    PA1     ------> ETH_REF_CLK
    PA2     ------> ETH_MDIO
    PA7     ------> ETH_CRS_DV
    PC4     ------> ETH_RXD0
```

```
    PC5     ------> ETH_RXD1
    PB11     ------> ETH_TX_EN
    PB12     ------> ETH_TXD0
    PB13     ------> ETH_TXD1
    */
    GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_4|GPIO_PIN_5;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF11_ETH;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_7;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF11_ETH;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF11_ETH;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  }
}
```

ВИСНОВКИ

Цю лабораторну роботу було виконано у команді, до команди входили Бурець Валентин та Ільчук Анатолій.

Бурець Валентин виконав ініціалізацію портів Ethernet та IwIP.

Ільчук Анатолій займався розробкою TCP сервера для N клієнтів.

На даній лабораторній роботі було створено мережеве підключення з використанням Ethernet. Відповідно до індивідуального завдання було створено TCP сервер для роботи з N клієнтами з використанням бібліотеки HAL.

У ході виконання лабораторної роботи було підключено до створюваного проекту відповідної бібліотеки. Потім було написано ініціалізацію Ethernet. І згідно індивідуального завдання було реалізовано відповідний функціонал.

Для виконання лабораторної роботи використовувалася бібліотека LwIP(light weight IP). Це широко використовуваний TCP/IP-стек з відкритим кодом, призначений для використання у вбудованих системах.