TCP/IP Server & Client Demo

TCP/IP Server demosu NUCLEO-F429ZI kartı üzerinde kurulmuştur. Amaç TCP/IP Server cihaz üzerinden Client'a sürekli olarak belli aralıklarla dummy veri göndermektir.

1. Ethernet Konfigürasyonu

Bahsedilen konfigürasyonlar TCP/IP Server ve Client cihazlar için de geçerlidir.

a. Projenin Oluşturulması

Önce yeni bir proje oluşturulmalıdır. Sol üst köşedeki ***"File"*** kısmından ***"New"*** ve ***"STM32 Project"*** seçeneği seçilmelidir. Çıkan ekranda ***"Board Selector"*** kısmına gelinmeli ve ***"Commercial Part Number"*** kısmına ***"NUCLEO-F429ZI"*** yazılmalıdır (Figure 1).

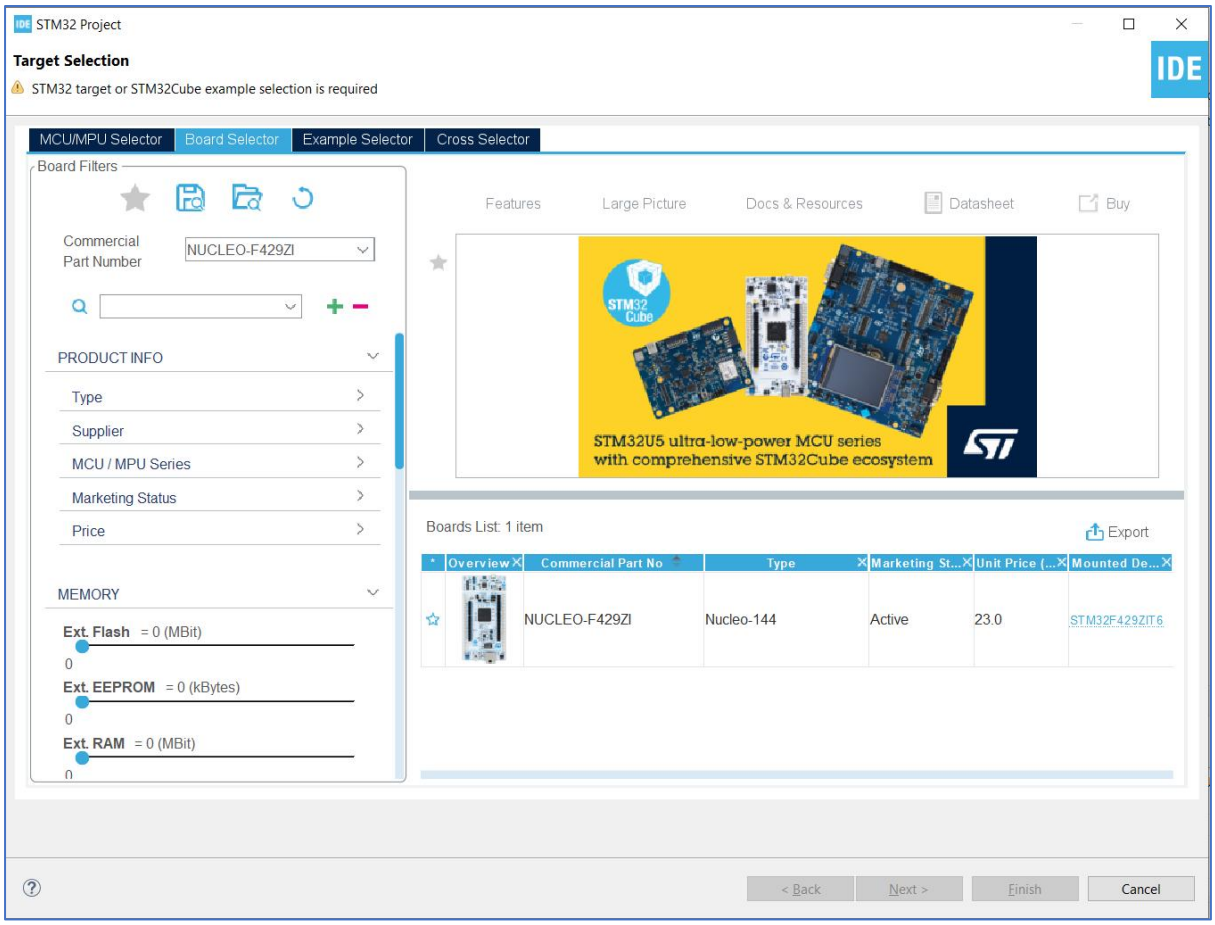

Figure 1 - Proje Kurulumu

"Boards List" kısmından kart seçilir, projeye isim verilir ve proje oluşturulur. Tercihen proje default ayarlarda oluşturulmamalıdır. Proje oluşturulduktan sonra projenin ioc dosyası açılır. Yine tercihen ioc dosyası açıldıktan sonra ***"Pinout"*** kısmından ***"Clear Pinout"*** seçeneği seçilerek mevcut seçili pinoutlar temizlenmelidir.

b. Clock Konfigürasyonu

Kartın clock ayarının yapılması için ***"System Core"*** kısmından ***"RCC"*** ekranı açılır. ***"High Speed Clock (HSE)"*** seçeneği ***"Crystal/Ceramic Resonator"*** olarak seçilmiştir. ***"Low Speed Clock (LSE)"*** seçeneği kapalı tutulmuştur (Figure 2). Demo oluşturulurken clock ayarları bu şekilde tercih edilmiştir. Farklı clock ayarları ile de proje çalışabilir.

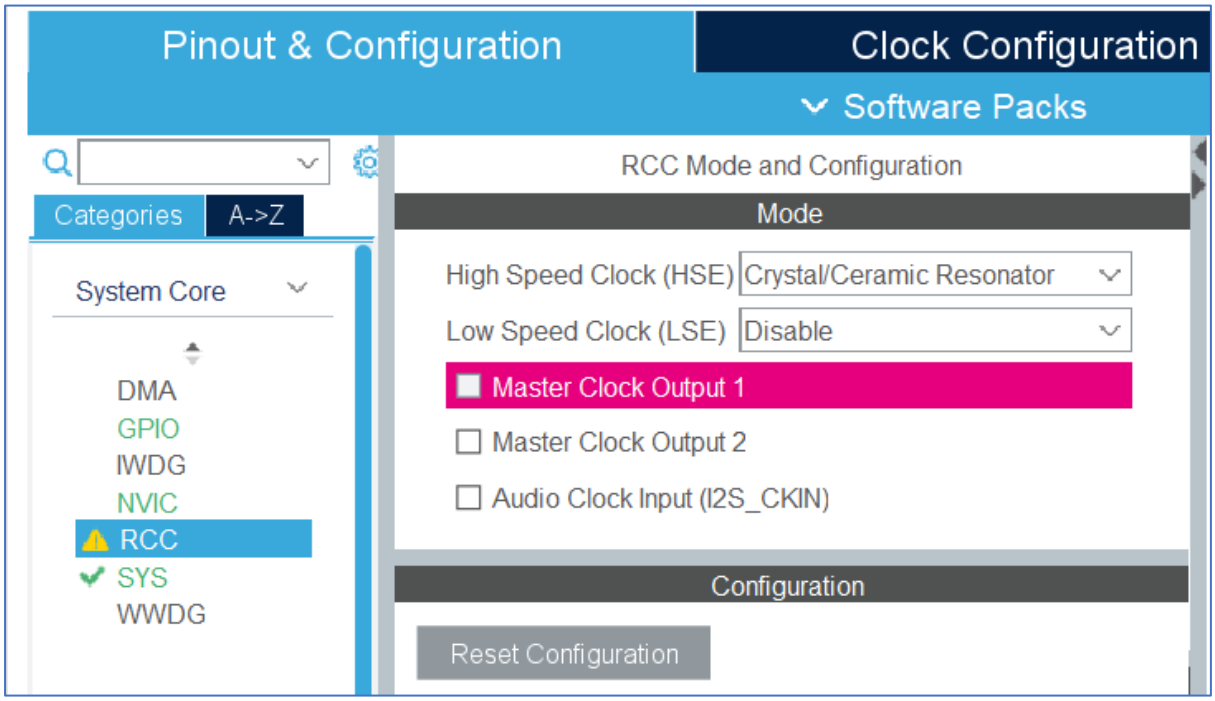

Figure 2 - Clock Seçimi

"Clock Configuration" kısmına gelinir. ***"HCLK"*** seçeneği maksimumda çalıştırılması tercih edilmiştir. 180 MHz olarak ayarlanmıştır. ***"Clock Configuration"*** kısmı sorunsuz bir şekilde ayarlandıktan sonra clock ayarları tamamlanmıştır.

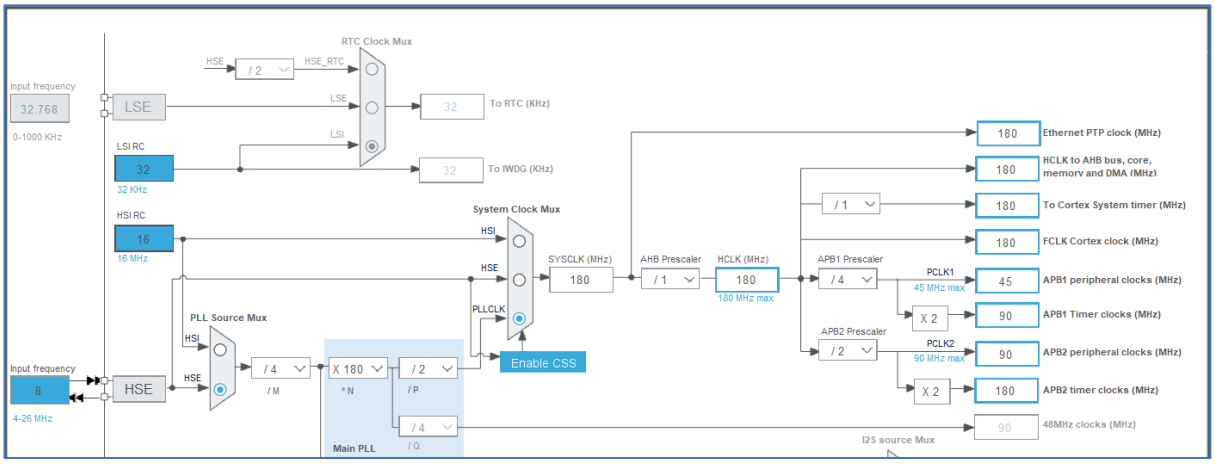

Figure 3 - Clock Konfigürasyonu

c. Timer Konfigürasyonu

"Timer" kategorisine gelinir ve ***"TIM1"*** seçilir. Tercihen ***"TIM1"*** kullanılmıştır. İstenirse farklı timer kullanılabilir. Bu noktada önemli olan seçilen timer'ın hangi APB hattına bağlı olduğudur. ***"TIM1"*** APB2 hattına bağlıdır. Eğer farklı bir timer kullanılacaksa STM32F429ZI Reference Manuel incelenerek seçilen timer'ın hangi APB hattına bağlı olduğu bulunabilir. Bunun için Reference Manuel'de ***"2.3 Memory Map"*** kısmına gelinir. Tabloda ***"Peripheral"*** kısmında TIM1 görüldüğü noktada yazan ***"Bus"*** seçeneği hangi APB hattına bağlı olduğunu belirtmektedir.

Memory and bus architecture **RM0090**

Table 1. STM32F4xx register boundary addresses (continued)

Boundary address	Peripheral	Bus	Register map
0x4001 4800 - 0x4001 4BFF	TIM11	APB2	Section 19.5.12: TIM10/11/13/14 register map on page 694
0x4001 4400 - 0x4001 47FF	TIM10		
0x4001 4000 - 0x4001 43FF	TIM9		Section 19.4.13: TIM9/12 register map on page 684
0x4001 3C00 - 0x4001 3FFF	EXTI		Section 12.3.7: EXTI register map on page 387
0x4001 3800 - 0x4001 3BFF	SYSCFG		Section 9.2.8: SYSCFG register maps for STM32F405xx/07xx and STM32F415xx/17xx on page 294 and Section 9.3.8: SYSCFG register maps for STM32F42xxx and STM32F43xxx on page 301
0x4001 3400 - 0x4001 37FF	SPI4	APB2	Section 28.5.10: SPI register map on page 925
0x4001 3000 - 0x4001 33FF	SPI1	APB2	Section 28.5.10: SPI register map on page 925
0x4001 2C00 - 0x4001 2FFF	SDIO		Section 31.9.16: SDIO register map on page 1074
0x4001 2000 - 0x4001 23FF	ADC1 - ADC2 - ADC3		Section 13.13.18: ADC register map on page 430
0x4001 1400 - 0x4001 17FF	USART6		Section 30.6.8: USART register map on page 1018
0x4001 1000 - 0x4001 13FF	USART1		
0x4001 0400 - 0x4001 07FF	TIM8		Section 17.4.21: TIM1 and TIM8 register map on page 587
0x4001 0000 - 0x4001 03FF	TIM1		
0x4000 7C00 - 0x4000 7FFF	UART8	APB1	Section 30.6.8: USART register map on page 1018
0x4000 7800 - 0x4000 7BFF	UART7		

Figure 4 - TIM1 BUS

"TIM1" seçeneği seçildikten sonra ***"Mode"*** kısmından ***"Clock Source"*** seçeneği ***"Internal Clock"*** olarak seçilir.

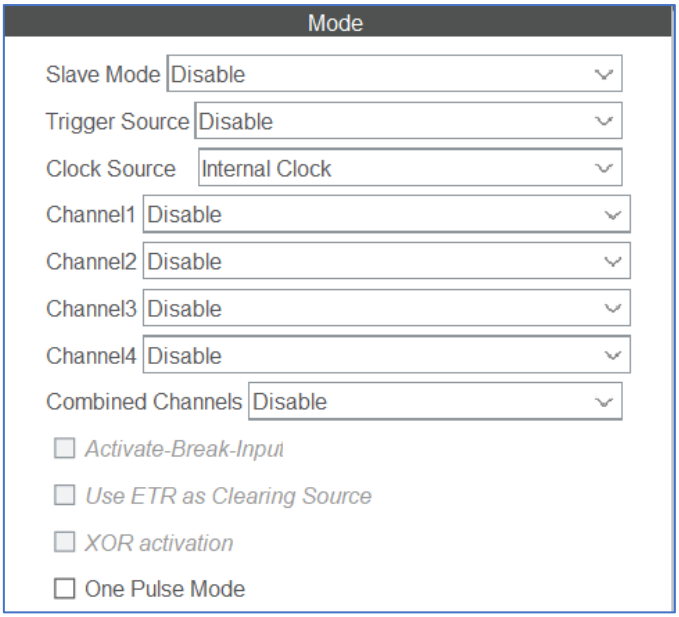

Figure 5 - TIM1 Mode

Timer'ın 1 saniye aralıklarla çalışması istenmektedir. ***"Configuration"*** kısmına gelinerek ***"Parameter Settings"*** kısmında ***"Prescaler"*** ve ***"Counter Period"*** 1 saniye için ayarlanmalıdır. Timer'ın çalışacağı süre aralığı, ilgili APB timer clock değeri ***"Prescaler"*** değerine bölünür. Çıkan sonuç da ***"Counter Period"*** değerine bölünür. Çıkan sonuç eğer 1 olursa timer süresi 1 saniye olacaktır. Bu yüzden ***"Prescaler"*** değeri ***"18000-1"*** ve ***"Counter Period"*** değeri ***"10000-1"*** olarak girilmiştir.

$$\mathrm{TIMupdateFreq(HZ)} = \mathrm{Clock}/((\mathrm{PSC}-1)*(\mathrm{Period}-1))$$

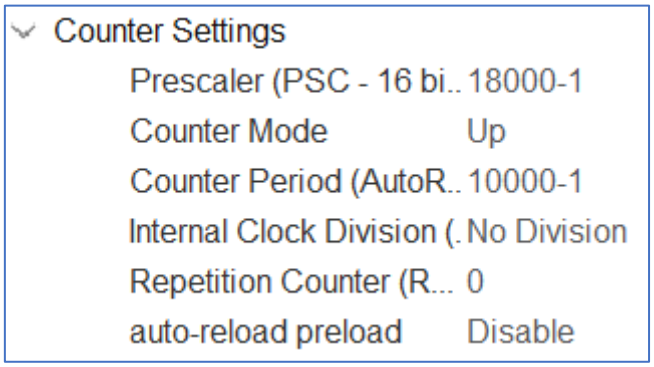

Figure 6 - Timer PSC ve CNT Konfigürasyonu

"Configuration" kısmında ***"NVIC Settings"*** içinde ***"TIM1 update interrupt and TIM10 global interrupt"*** seçeneği active edilmelidir. Bu ayarın tamamlanması ardından timer konfigürasyonu tamamlanacaktır.

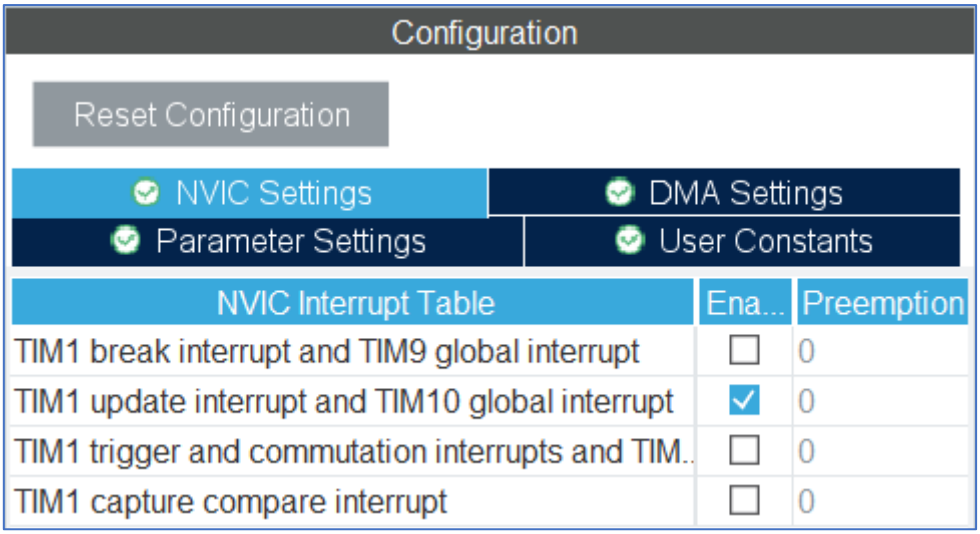

Figure 7 - Timer NVIC Konfigürasyonu

d. Ethernet Konfigürasyonu

"Categories" kısmından **"Connectivity"** altındaki **"ETH"** aktif edilmelidir. Bu noktada kullanılacak olan kartın MII'mı yoksa RMII'mı desteklediği bilinmelidir. Kullanılan kart örneğin yazılımsal olarak RMII destekliyor ama donanımsal olarak desteklemiyor olabilir. Bu bilginin kontrolü için ilgili kartın şematik dosyası incelenebilir. NUCLEO-F429ZI kartı için şematik dosyasına ulaşmak adına şu adımlar izlenebilir. Tarayıcı üzerinde **"NUCLEO-F429ZI schematic"** şeklinde arama yapılır. **"NUCLEO-F429ZI – STMicroelectronics"** isimli arama sonucuna girilir. **"CAD Resources"** sekmesinde **"Schematic Pack"** başlığı altında **"STM Nucleo (144 Pins) schematics"** isimli dosya indirilir.

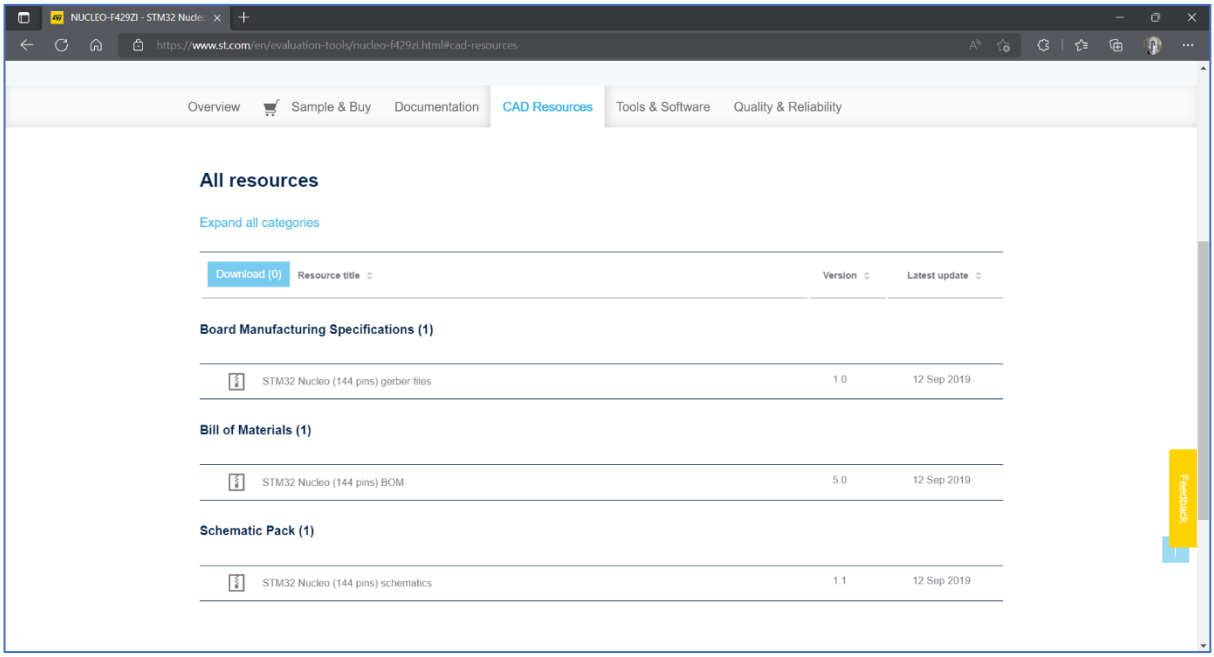

Figure 8 - NUCLEO-F29ZI Şematik Dosyalarına Erişim

İndirilen zip'in içinde hem tek tek Altium dosyaları hem de tüm şematiğin olduğu **"MB1137"** bir pdf dosyası bulunmaktadır. Pdf dosyasında sayfa 5'te **"LAN8742A-CZ-TR"** çipi ve çevre birimleri görülebilir.

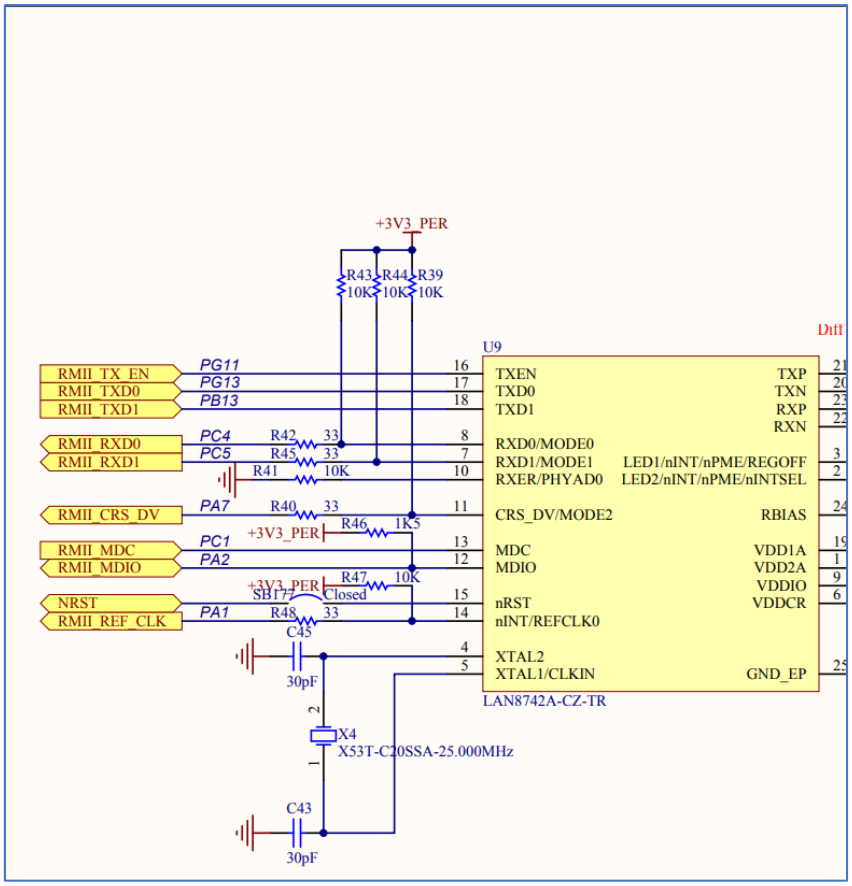

Figure 9 - LAN8742 Bağlantıları

Figür 9'da çipin bağlantılarının RMII ile yapıldığı görülebilir. Bu yüzden **"ETH"** konfigürasyonu sayfasında **"Mode"** seçeneği **"RMII"** seçilebilir.

"Parameter Settings" kısmında bir not düşüldüğü görülebilir. **"PHY Driver must be configured from the LwIP 'Platform Settings' top right tab"** şeklinde bir not düşülmüştür. İlgili ayar, raporun **"LwIP Konfigürasyonu"** kısmında yapılmaktadır.

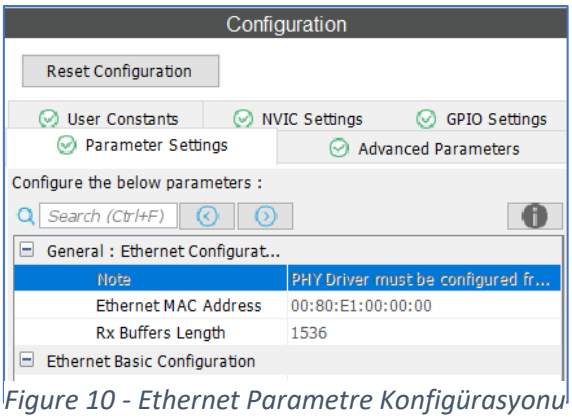

Figure 10 - Ethernet Parametre Konfigürasyonu

“Advanced Parameters” sekmesinde **“PHY”** ayarının **“LAN8742A_PHY_ADDRESS”** olarak seçili olduğundan emin olunması gerekmektedir.

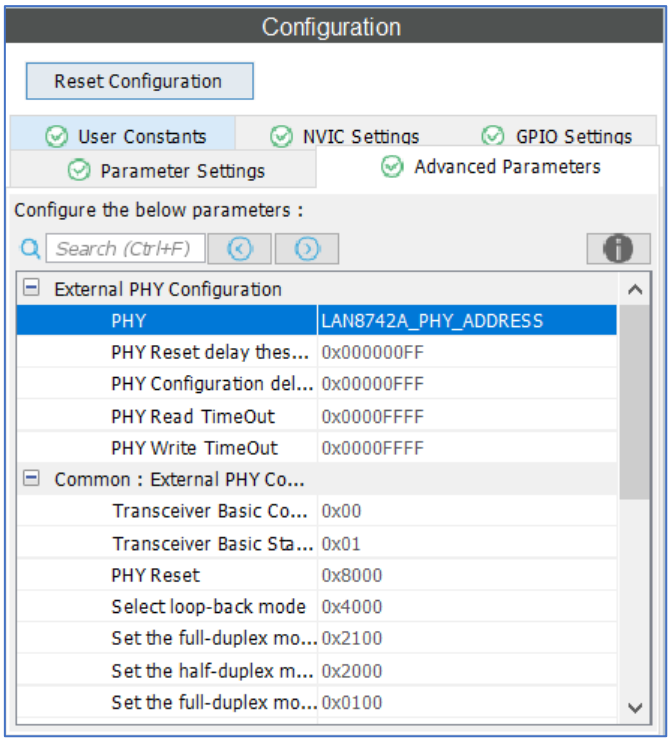

Figure 11 - Ethernet PHY Konfigürasyonu

“GPIO Settings” sekmesinde ilgili sinyallerin hangi pinlere bağlı olduğu görülmektedir. CubeIde default olarak bu pinleri bazen yanlış konfigüre edebilmektedir. Pinlerin doğru olduğundan emin olunması için kontrol edilmesi gerekmektedir. Figür 9’da pin konfigürasyonun nasıl olması gerektiği gösterilmiştir. Eğer düzeltimesi gereken bir pin olması durumunda **“Pinout View”** sekmesinde çipin ilgili pinine sağ tıklayarak, o pine atanabilecek olan sinyaller görülebilir. Pinoutla ilgili bir hata olması durumunda bu şekilde ilgili sinyaller ilgili pinlere atanabilir.

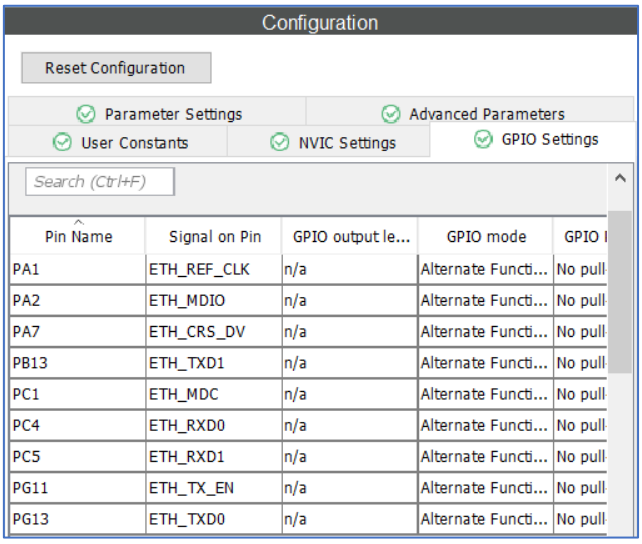

Figure 12 - Ethernet Pin Konfigürasyonu

“NVIC Settings” sekmesinde **“Ethernet Global Interrupt”** ayarının aktif edilmesi gerekmektedir.

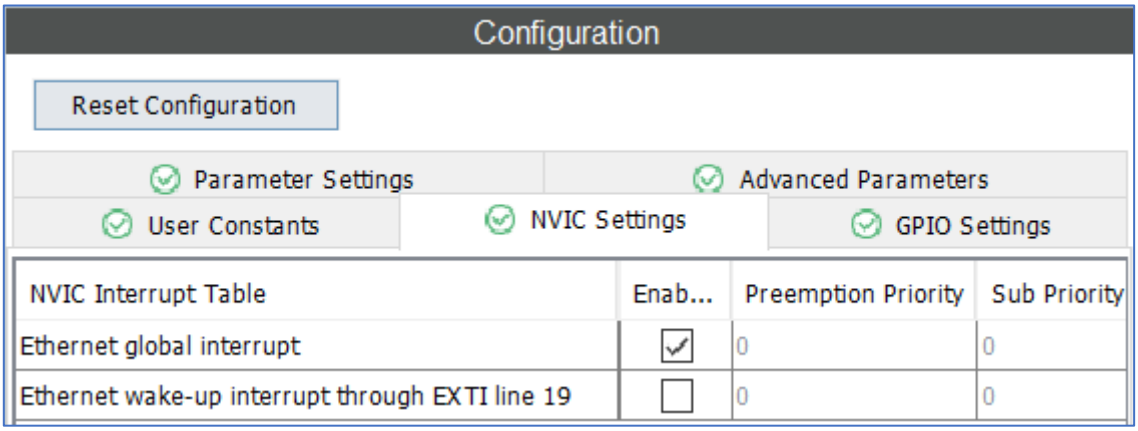

Figure 13 - NVIC Konfigürasyonu

e. LWIP Konfigürasyonu

“Categories” kısmından ***“Middleware”*** altında ***“LWIP”*** seçeneği active edilmelidir. ***“LWIP”*** seçeneği altındaki ***“Mode”*** kısmından ***“Enabled”*** kutucuğu aktive edilmelidir. Bu kutucuk aktive edildikten sonra ***“Configuration”*** kısmı aktif olacaktır.

“Platform Settings” kısmında ***“Driver_PHY”*** seçeneği olarak ***“LAN8742”*** seçilmelidir. Bu seçenek kart üzerindeki ethernet çipini seçmektedir. Eğer seçenek seçilmezse ethernet çalışmayacaktır. Aynı zamanda kod derlendiğinde de hata alınmaz. O yüzden ayarı gözden kaçırmamak gerekmektedir.

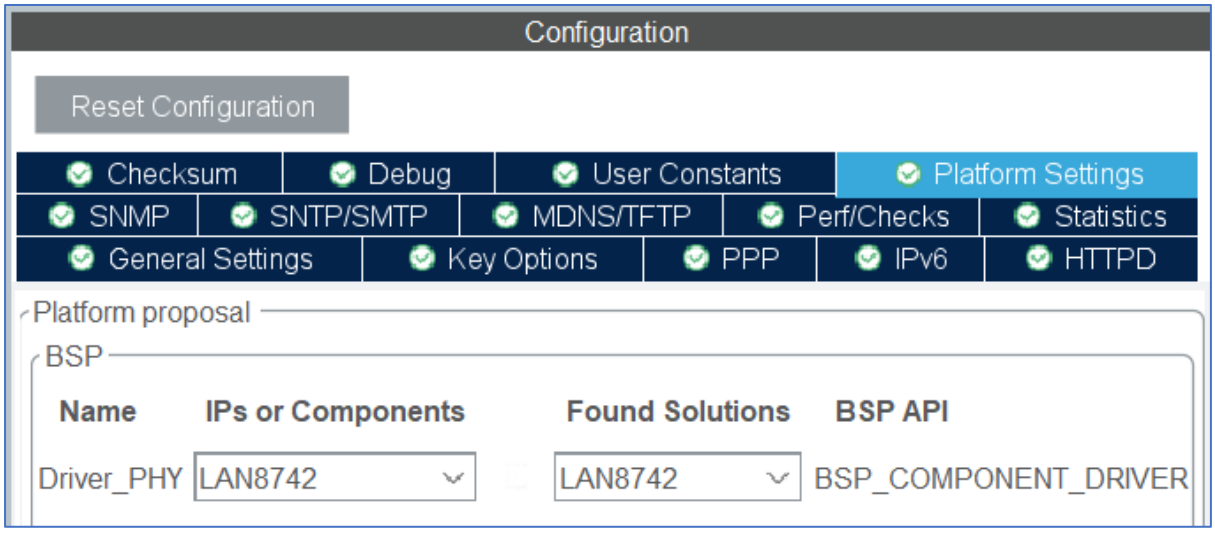

Figure 14 - Platform Settings

“Checksum” kısmında ***“CHECKSUM_BY_HARDWARE”*** seçeneği ***“Enabled”*** olmalıdır.

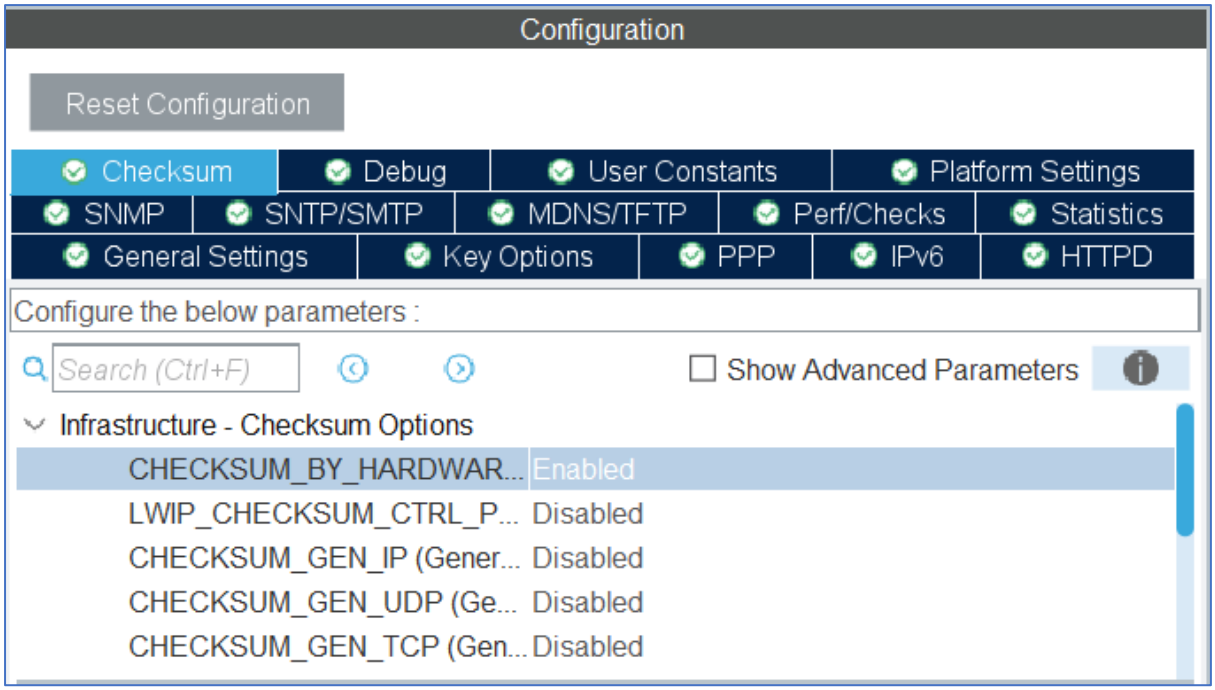

Figure 15 - Checksum Konfigürasyonu

"General Settings" kısmında ***"LWIP_DHCP"*** ayarı ***"Disabled"*** olmalıdır. Bu ayarın ***"Disabled"*** olması ethernet server'ı için manuel ip verebilmemizi sağlamaktadır. ***"LWIP_DHCP"*** ayarı ***"Disabled"*** edildikten sonra ***"IP_ADDRESS"***, ***"NETMASK_ADDRESS"*** ve ***"GATEWAY_ADDRESS"*** ayarları yapılmalıdır. Buradaki ayarlar istendiği gibi konfigüre edilebilir. Fakat ***"IP_ADDRESS"*** değeri sonrasında ethernet bağlantılarında kullanılacağı için unutulmamalıdır.

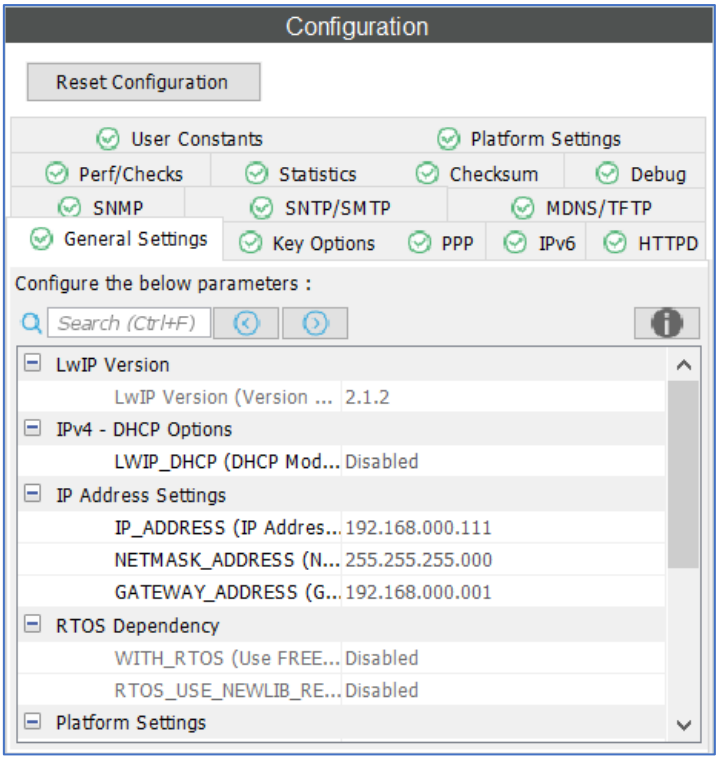

Figure 16 - IP Konfigürasyonu

"Key Options" kısmında ***"MEM_SIZE"*** default olarak bırakılabilir. Örnek projede *10*1024 Byte* olarak tanımlanmıştır.

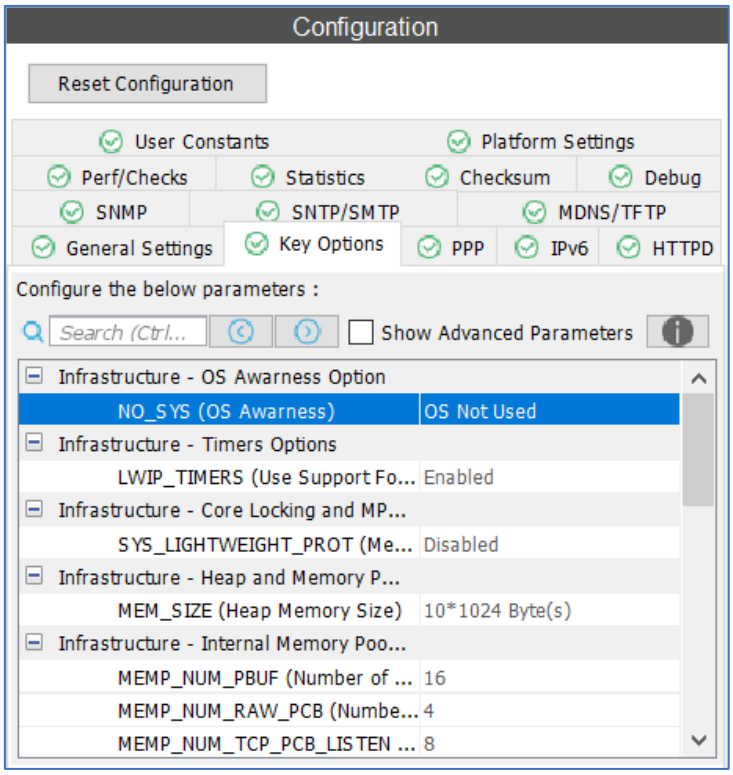

Figure 17 - Heap Memory Size Konfigürasyonu

Kalan ayarlar default olarak bırakılmıştır.

2. Yazılım Konfigürasyonu

a. TCP/IP Server

Örnek proje dosyaları içerisinde ***"tcpServerRAW.c"*** ve ***"tcpServerRAW.h"*** dosyaları bulunmaktadır. Bu dosyalar TCP ethernet server için kullanılacak olan temel fonksiyonları içermektedir. Fonksiyonlar ST tarafından yazılmış olup, dosyalar ControllersTech tarafından düzenlenmiştir. Koda istenilen bazı eklemeler yapılarak örnek projedeki son halini almıştır.

Main kodda ***"Private Includes"*** kısmında ***"tcpServerRAW.h"*** eklenmelidir.

```
/* Includes ------------------
#include "main.h"
#include "lwip.h"

/* Private includes ----------
/* USER CODE BEGIN Includes */

#include "tcpServerRAW.h"

/* USER CODE END Includes */
```

Figure 18 - tcpServerRAW Headerının Eklenmesi

"Private Define" kısmına ***"extern struct netif gnetif"*** kodu eklenmelidir. Ethernet ile ilgili konfigürasyonları barıdındıran struct yapısı bu şekilde main kodda çağrılabilir olmaktadır.

```
/* Private define ----------
/* USER CODE BEGIN PD */

extern struct netif gnetif;

/* USER CODE END PD */
```

Figure 19 - netif Structının Çağrılması

Main fonksiyonu içerisinde TCP server ve TIM1 init edilmelidir. Timerın TCP server'ın init edilmesinden sonra init edilmesi daha sağlıklı olacaktır.

```
/* USER CODE BEGIN 2 */

tcp_server_init();
HAL_TIM_Base_Start_IT(&htim1);
/* USER CODE END 2 */
```

Figure 20 - TCP Server ve Timer Init Edilmesi

Main fonksiyonu içerisindeki while döngüsü içinde ***"ethernetif_input(&gnetif)"*** ve ***"sys_check_timeouts"*** fonksiyonları çağrılmalıdır. ***"ethernetif_input(&gnetif)"*** fonksiyonu ***"gnetif"*** ile ethernet konfigürasyonlarını girdi olarak alarak ethernet portunu dinlemeye almaktadır. ***"sys_check_timeouts"*** fonksiyonu ise timeout expire olduğunda timeout handlerını çağırmaktadır.

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)

  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

      ethernetif_input(&gnetif);
//    MX_LWIP_Process();
      sys_check_timeouts();

  }
  /* USER CODE END 3 */
}
```

Figure 21 - While(1) döngüsü

Main kod içerisindeki düzenlemeler bu noktada tamamlanmıştır. Fonskiyonları daha iyi anlamak için ***"tcpServerRAW.c"*** dosyası incelenebilir. Giriş kısmında includelar ve yapılar tanımlanmıştır. ***"tcp_server_states"*** enum yapısı ethernet protokolünün mevcut durum tanımlamalarının tutulduğu yapıdır. ***"tcp_server_struct"*** yapısı direkt olarak mevcut bağlantı bilgilerinin tutulduğu yapıdır.

```
#include "tcpserverRAW.h"
#include "lwip/tcp.h"

/*  protocol states */
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 */
};
```

Figure 22 - tcpServerRAW.c Includelar ve Yapılar

Kodun devamında tanımlanan fonksiyonların prototipleri bulunmaktadır.

```
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);
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);
```

Figure 23 - tcpServerRAW.c Fonksiyon Prototipleri

Timer callback fonksiyonun tanımlaması ***"tcpServerRAW.c"*** dosyası içerisinde yapılmıştır. Fonksiyon erişimleri dosya içerisinde tutulduğu için böyle bir yöntem izlenmiştir. Timer saniyede 1 callback fonksiyonuna girmektedir. Ethernet portuna veri basma callback fonksiyonu içerisinde yapılacaktır. Fonksiyonun başında 100'lük bir char array tanımlanmıştır. Sprintf kullanılarak tanımlanan char array'e string atanmıştır ve string uzunluğu ***"len"*** isimli değişkene return edilmiştir.

Counter değerinin 0 olmadığı durumlarda pBuf string uzunluğu kadar bir boyutta allocate edilmiştir. ***"pbuf_take"*** fonksiyonu ile allocate edilen bölgeye stringi içeren buffer atanmıştır. Sonrasında ***"tcp_server_send"*** fonksiyon çağıralarak veri ethernet portuna gönderilmiş ve sonrasında pbuf ***"pbuf_free"*** fonksiyonu ile pBuf serbest bırakılmıştır.

```
int counter = 0;

struct tcp_server_struct *esTx = 0;
struct tcp_pcb *pcbTx = 0;

extern TIM_HandleTypeDef htim1;

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    char buf[100];

    /* Prepare the first message to send to the server */
    int len = sprintf (buf, "Sending TCPclient Message %d\n", counter);

    if (counter !=0)
    {
        /* allocate pbuf */
        esTx->p = pbuf_alloc(PBUF_TRANSPORT, len , PBUF_POOL);

        /* copy data to pbuf */
        pbuf_take(esTx->p, (char*)buf, len);

        tcp_server_send(pcbTx, esTx);

        pbuf_free(esTx->p);
    }
}
```

Figure 24 - TIM Callback Fonksiyonu

"tcp_server_init" fonksiyonu incelenebilir. Fonksiyonun başlangıcında ***"tcp_pcb"*** yapısında ***"tpcb"*** isminde bir pointer obje oluşturulmuştur. Bu obje ***"tcp_new"*** fonksiyonu ile init edilmitşir. ***"err_t"*** tipi error kodları saklamaktadır. Bu tipte bir değişken oluşturulmasının sebebi budur. Typedef ile tanımlanmıştır. ***"ip_addr_t"*** tipinde ***"myIPADDR"*** isimli bir obje tanımlanmıştır. IP adres bilgilerini bu obje tutmaktadır. Sonraki satırda ***"IP_ADDR4"*** fonksiyonu ile istenilen IP adres değerleri objeye tanımlanmıştır. Burada verilen IP adres değeri ile ethernet konfigürasyonu kısmında ayarlanan IP değeri aynı olmalıdır. ***"tcp_bind"*** fonksiyonu birlikte verilen port ve IP bilgileri ile bağlantı kurulmuştur. Return edilen error kodu ***"err"*** değişkenine atanmıştır. Eğer return edilen error kodu ***"ERR_OK"*** ise pcb için TCP dinlemeye sokulmuş ve ***"tcp_accept"*** fonksiyonu çağırılarak LwIP initialize edilmiştir. Eğer error kodu ***"ERR_OK"*** değilse allocate edilen memory serbest bırakılmıştır.

```c
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, 111);
    err = tcp_bind(tpcb, &myIPADDR, 10);

    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);
    }
}
```

Figure 25 - tcp_server_init Fonksiyonu

"tcp_server_accept" fonksiyonu incelenebilir. ***"err_t"*** tipinde, error durumunu tutan ***"ret_err"*** objesi ve ***"tcp_server_struct"*** tipinde ethernet konfigürasyonlarını tutan ***"es"*** pointeri tanımlanmıştır. Kullanılmayan objelerin derleme sırasında sorun çıkarmaması adına ***"LWIP_UNUSED_ARG"*** fonksiyonu kullanılmıştır. Ethernet konfigürasyonlarının init edilmesi ve prioritylerin atanması için ***"tcp_setprio"*** fonksiyonu çağrılmıştır. RAM'de ***"tcp_server_struct"*** kadar yerin ayrılması adına ***"mem_malloc"*** fonksiyonu kullanılmıştır.

```c
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));
```

Figure 26 - tcp_server_accept Fonksiyonu Tanımlamalar

Eğer ***"es"*** objesi ***"NULL"*** değilse ve gerekli yer RAM'de ayrılabildiyse kod if bloğunun içine girer. Diğer durumda else bloğunun içine girecektir. If bloğunun içine girmesi durumunda ***"es"***

objesine gerekli atamalar yapılır ve sırasıyla ***"tcp_arg"***, ***"tcp_recv"***, ***"tcp_err"*** ve ***"tcp_poll"*** fonksiyonları çağrılır. Fonksiyonların amaçlarının ne olduğu raporun ilgili yerlerinde anlatılmıştır. Bloğun sonunda herhangi bir sorunla karşılaşılmadığına dair ***"ret_err"*** değişkenine atama yapılır. Else bloğuna girilmesi durumunda tcp bağlantısı ***"tcp_server_connection_close"*** fonksiyonu çağrılarak sonlandırılır. Son olarak ***"ret_err"*** değişkenine ***"ERR_MEM"*** ataması yapılır. Bunun anlamı memory allocation konusunda bir sıkıntı olduğu ve bağlantının sonlandırıldığıdır.

```
  if (es != NULL)
  {
    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;
}
```

Figure 27 - tcp_server_accept Fonksiyonu If Else Bloğu

"tcp_server_recv" fonksiyonu incelenebilir. Önceki fonksiyonlarda olduğu gibi ***"es"*** ve ***"ret_err"*** objeleri oluşturulur. Eğer ***"arg != NULL"*** ise ***"LWIP_ASSERT"*** fonksiyonu kullanılarak ***"arg != NULL"*** yazısı ***"printf"*** ile yazdırılır.

```
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;
```

Figure 28 - tcp_server_recv Fonksiyonu Tanımlamalar

Yorum satırında da bahsedildiği gibi if bloğu incelendiğinde, eğer ***"p"*** pointer'ı ***"NULL"*** ise boş frame alınmıştır. Bu durumda tcp durumu ***"ES_CLOSING"*** yapılır. Eğer ***"es -> p"*** değeri de ***"NULL"*** ise ***"tcp_server_connection_close"*** fonksiyonu çağrılarak TCP bağlantısı sonlandırılır. Diğer

durumda paket alınmış demektir ve kalan paket gönderilir. ***"ret_err"*** objesine ***"ERR_OK"*** ataması yapılır.

```
/* 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;
}
```

Figure 29 - tcp_server_recv Fonksiyonu If Bloğu

Yorum satırında da bahsedildiği gibi eğer gelen frame boş değilse ama bir şekilde ***"err"*** ***"ERR_OK"*** olarak gelmediyse bu bloğa girilir. ***"p"*** ***"NULL"*** değilse if bloğuna girer ve ***"NULL"*** ataması ile birlikte ***"pbuf"*** serbest bırakılır. **"err"** objesinin mevcut durumu **"ret_err"** objesine de atanır.

"es" objesinin mevcut durumu ***"ES_ACCEPTED"*** olduğunda data alınmıştır. ***"es"*** objesinin durumu ***"ES_RECEIVED"*** olarak güncellenir. ***"p"*** objesine gelen veri ***"es -> p"*** objesine atanır. ***"tcp_set"*** ve ***"tcp_server_handle"*** fonksiyonları sırası ile çağrılır. ***"ret_err"*** objesi ***"ERR_OK"*** olarak güncellenir.

```
/* 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;
}
```

Figure 30 - tcp_server_recv Fonksiyonu ERR_OK ve ES_ACCEPTED Blokları

"ES_RECEIVED" bloğu incelenebilir. Yorum satırında da anlatıldığı gibi eğer client tarafından data gelmeye devam ediyor ve önceki data gönderilmişse if bloğuna girer. Bu durumda ***"es -> p"*** objesine ***"p"*** ataması yapılır. Yani mevcut veri struct yapısına alınır ve ***"tcp_server_handle"*** fonksiyonu çağrılır. Diğer durumda **"pbuf"** tipinde bir pointer tanımlanır ve bu pointera ***"es -> p"*** atanır. ***"pbuf_chain"*** fonksiyonu kullanılarak gelen veri ***"pbuf"*** zincirine eklenir. Son durumda ***"ret_err"*** objesine ***"ERR_OK"*** ataması yapılır.

"ES_CLOSING" bloğu incelenebilir. ***"tcp_recved"*** fonksiyonu çağrılarak daha büyük bir frame ihtiyacı olduğu belirlenir. ***"es -> p"*** objesine ***"NULL"*** ataması yapılır ve ***"p"*** objesi serbest bırakılır. Son olarak ***"ret_err"*** objesine ***"ERR_OK"*** ataması yapılır.

```
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;
}
```

Figure 31 - tcp_server_recv Fonksiyonu ES_RECEIVED ve ES_CLOSING Blokları

Son olarak else bloğunda ***"es -> state"*** bilinmemektedir. Bu yüzden ***"tcp_recved"*** fonksiyonu daha önce olduğu gibi çağrılır. ***"es -> p"*** objesine ***"NULL"*** ataması yapılır ve ***"p"*** objesi ***"pbuf_free"*** fonksiyonu ile serbest bırakılır. Son olarak ***"ret_err"*** objesine ***"ERR_OK"*** ataması yapılır. Fonksiyonun sonunda ***"ret_err"*** değeri return edilir.

```
  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;
}
```

Figure 32 - tcp_server_recv Fonksiyonu Else Bloğu

"tcp_server_error" fonksiyonu incelenebilir. Önceki fonksiyonlarda olduğu gibi ***"es"*** objesi oluşturulmaktadır. Derleme sırasında sorun olmaması için kullanılmayan argumanlar ***"LWIP_UNUSED_ARG"*** fonkisyonu ile gönderilmiştir. ***"tcp_server_error"*** fonksiyonuna verilen ***"arg"***, ***"es"*** objesine atanmıştır. Eğer ***"es != NULL"*** ise if bloğuna girilmektedir. Bu durumda memory serbest bırakılmaktadır.

```
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);
  }
}
```

Figure 33 - tcp_server_error Fonksiyonu

"tcp_server_poll" fonksiyonu incelenebilir. Daha önceki fonksiyonlarda olduğu gibi ***"ret_err"*** ve ***"es"*** objeleri tanımlanmıştır. Fonkisyona gönderilen ***"arg"*** cast edilerek ***"es"*** objesine atanmıştır. Eğer ***"es"*** objesi boş değilse if bloğuna girilir. Eğer ***"es -> p"*** objesi boş değilse gönderilecek veri vardır. ***"tcp_sent"*** ve ***"tcp_server_send"*** fonksiyonları sırası ile çağrılır. Eğer **"es -> p" "NULL"** ise ve ***"es -> state"*** objesine mevcut durumda ***"ES_CLOSING"*** atanmışsa daha fazla ***"pbuf"*** zinciri kalmamıştır ve ***"tcp_server_connection_close"*** fonksiyonu çağrılır. Son olarak ***"ret_err"*** objesine ***"ERR_OK"*** ataması yapılır. Eğer ***"es"*** objesi ***"NULL"*** ise, yapılacak bir şey kalmamıştır ve ***"tcp_abort"*** fonksiyonu çağrılır. Son olarak ***"ret_err"*** objesine ***"ERR_ABRT"*** ataması yapılır ve ***"ret_err"*** return edilir.

```
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;
}
```

Figure 34 - tcp_server_poll Fonksiyonu

"tcp_server_sent" fonksiyonu incelenebilir. Önceki fonksiyonlarda olduğu gibi ***"es"*** objesi tanımlanmaktadır. Derleme sırasında sorun olmaması için kullanılmayan ***"len"*** değişkeni ***"LWIP_UNUSED_ARG"*** fonksiyonuna gönderilmiştir. Fonksiyona gönderilen ***"arg"*** objesi ***"es"*** objesine atanmıştır. Ayrıca ***"es -> retries"*** objesi de sıfırlanmıştır. Eğer ***"es -> p"*** objesi ***"NULL"*** değilse if bloğuna girmektedir. Bu gönderilecek ***"pbuf"*** olduğunu belirtmektedir. Sırası ile ***"tcp_sent"*** ve ***"tcp_server_send"*** fonksiyonları çağırılmıştır. Eğer değilse ve ***"es -> state"*** objesi ***"ES_CLOSING"*** olarak atalıysa ***"tcp_server_connection_close"*** fonksiyonu çağrılarak bağlantı sonlandırılır. Son olarak ***"ERR_OK"*** değeri return edilir.

```
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;
}
```

Figure 35 - tcp_server_sent Fonksiyonu

"tcp_server_send" fonksiyonu incelenebilir. Başlangıç olarak ***"pbuf"*** tipinde ***"ptr"*** isminde bir pointer struct tanımlanmıştır. Aynı zamanda ***"err_t"*** tipinde ***"wr_err"*** isminde bir değişken tanımlanmış ve ***"ERR_OK"*** atanmıştır. ***"tcp_sndbuf"*** fonksiyonu uygun olan buffer size'ını döndürmektedir. Kod while döngüsüne sokulmuştur. Koşul olarak, ***"wr_err" "ERR_OK"*** olduğu, ***"es -> p" "NULL"*** olmadığı ve ***"es -> p -> len"*** objesi uygun olan buffer size'ından küçük olduğu sürece while dönmektedir. ***"es -> p"*** pointer'ı başlangıçta tanımlanan ***"ptr"*** pointerına atanmıştır. Mevcut data ***"tcp_write"*** ile yazıldıktan sonra return değeri ***"wr_err"*** değişkenine atanmıştır.

```
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);
```

Figure 36 - tcp_server_send Fonksiyonu

"wr_err" değişkeni ***"ERR_OK"*** olduğu durumda if bloğuna girilmiştir. ***"plen"*** ve ***"freed"*** değişkenleri tanımlandıktan sonra, ***"plen"*** değişkenine ***"ptr -> len"*** ataması yapılmıştır. ***"es -> p"*** pointerına ***"ptr -> next"*** ataması yapılarak sıradaki ***"pbuf"*** atanır. Eğer ***"es -> p" "NULL"*** değilse ***"pbuf_ref"*** fonksiyonu çağrılarak referans counteri artırılır. Do while bloğunun do bloğu içerisinde ***"pbuf_free"*** fonksiyonu çağırılarak, return değeri ***"freed"*** değişkenine atanmıştır. ***"freed"*** değişkeni 0'a eşit olana kadar kod while içerisinde boş dönmüştür. Son durumda daha fazla data okunulabilir olmuştur ve ***"tcp_recved"*** fonksiyonu çağırılmıştır.

```
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);
}
```

Figure 37 - tcp_server_send Fonksiyonu If Bloğu

Eğer ***"wr_err"*** değişkeni ***"ERR_MEM"*** değerine eşitse else if bloğuna girer ve ***"ptr"*** pointeri ***"es -> p"*** pointerına atanır. Bu durumda memory azalmıştır ve sonra denemek üzere devam edilir. Farklı problemlerde yapılacakların yazılması için else bloğu boş bırakılmıştır.

```
else if(wr_err == ERR_MEM)
{
   /* we are low on memory, try later / harder, defer to poll */
  es->p = ptr;
}
else
{
  /* other problem ?? */
}
}
```

Figure 38 - tcp_server_send Fonksiyonu Else If ve Else Bloğu

"tcp_server_connection_close" fonksiyonu incelenebilir. Sırasıyla ***"tcp_arg"***, ***"tcp_sent"***, ***"tcp_recv"***, ***"tcp_err"*** ve ***"tcp_poll"*** fonksiyonları çağırılarak ***"tpcb"*** objesi boşaltılır. Fonksiyonlar çağırılırken argüman olarak ***"tpcb"*** ve ***"NULL"*** değeleri verilmiştir. ***"es"*** structure yapısının da memoryde tuttuğu alan serbest bırakılır. Son olarak TCP bağlantısı ***"tcp_close"*** fonksiyonu kullanılarak sonlandırılır.

```
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);
}
```

Figure 39 - tcp_server_connection_close Fonksiyonu

"tcp_server_handle" fonksiyonu incelenebilir. Client cihazının IP adresi ve port bilgisi, tanımlanan ***"inIP"*** ve ***"inPort"*** değişkenlerine atanmıştır lakin bu değişkenler kullanılmayacaktır. Aynı şekilde ***"ipaddr_ntoa"*** fonksiyonu kullanılarak char pointer olan ***"remIP"*** pointerına IP bilgisi çıkartılmıştır. Aynı şekilde bu değişken de kullanılmamıştır. Mevcut olan ***"es"*** structure içindeki bilgiler ve ethernet konfigürasyonu bilgilerini saklayan ***"tpcb"***, ***"esTx"*** ve ***"pcbTx"*** yapılarına çıkarılmıştır. Bu değişkenler ***"HAL_TIM_PeriodElapsedCallback"*** fonksiyonunda kullanılmıştır. Son olarak ***"counter"*** değeri artırılmıştır. Bu durum, client tarafından her veri geldiğinde ***"counter"*** değeri artırılacaktır anlamına gelmektedir.

```
static void tcp_server_handle (struct tcp_pcb *tpcb, struct tcp_server_struct *es)
{
    /* 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 = es;
    pcbTx = tpcb;

    counter++;
}
```

Figure 40 - tcp_server_handle Fonksiyonu

b. TCP/IP Client

Örnek proje dosyaları içerisinde ***"tcpClientRAW.c"*** ve ***"tcpClientRAW.h"*** dosyaları bulunmaktadır. Bu dosyalar TCP ethernet client için kullanılacak olan temel fonksiyonları içermektedir. Fonksiyonlar ST tarafından yazılmış olup, dosyalar ControllersTech tarafından düzenlenmiştir. Koda istenilen bazı eklemeler yapılarak örnek projedeki son halini almıştır.

"TCP/IP Server" kısmında yazılım konfigürasyonu konusunda çok büyük bir kısmı benzerdir. ***Bu kısımda sadece farklı olan kısımlar anlatılacaktır.*** Eğer client bağlantısı yapılmak isteniyorsa önce ***"TCP/IP Server"*** kısmı incelenip sonra bu kısıma geçilebilir.

Main kodda ***"Private Includes"*** kısmında ***"tcpClientRAW.h"*** eklenmelidir.

```
/* Private includes ----------
/* USER CODE BEGIN Includes */

#include "tcpClientRAW.h"

/* USER CODE END Includes */
```

Figure 41 - tcpClientRaw Headerın Eklenmesi

Main fonksiyonu içerisinde TCP client ve TIM1 init edilmelidir. Timerın TCP client'ın init edilmesinden sonra init edilmesi daha sağlıklı olacaktır.

```
/* USER CODE BEGIN 2 */

tcp_client_init();
HAL_TIM_Base_Start_IT(&htim1);

/* USER CODE END 2 */
```

Figure 42 - TCP Server ve Timer Init Edilmesi

Main kod içerisindeki düzenlemeler bu noktada tamamlanmıştır. Fonskiyonları daha iyi anlamak için ***"tcpClientRAW.c"*** dosyası incelenebilir. "tcpClientRAW.c" dosyası ile "tcpServerRAW.c" dosyaları birbirine neredeyse tamamen benzemektedir. ***O yüzden bu kısımda "tcpClientRAW.c" dosyası incelenmeyecektir***. Eğer client bağlantısı yapılmak isteniyorsa önce ***"TCP/IP Server"*** sonra devam edilebilir.

3. Ethernet Bağlantısı

a. TCP/IP Server

"tcp_server_init" fonksiyonu içerisinde server cihazının IP adresi ve portu belirlenmişti. Raporun önceki kısımlarında görülebilir. ***"ioc"*** dosyası içerisinde de aynı şekilde tanımlı ise bağlantı yapılabilir. Bağlantı için ***"Hercules"*** program kullanılacaktır. Verilen link üzerinden uygulama indirilebilir. Program direkt olarak exe üzerinden çalışmaktadır ve kuruluma ihtiyaç yoktur. Program kurulduktan sonra ***"TCP Client"*** sekmesi açılır. Bu sekmede program TCP bağlantısında client gibi davranacaktır. ***"TCP"*** isimli bloğun içerisinde ***"Module IP"*** kısmına server cihazının IP adresi yazılır. ***"Port"*** kısmına haberleşilen port bilgisi yazılır.

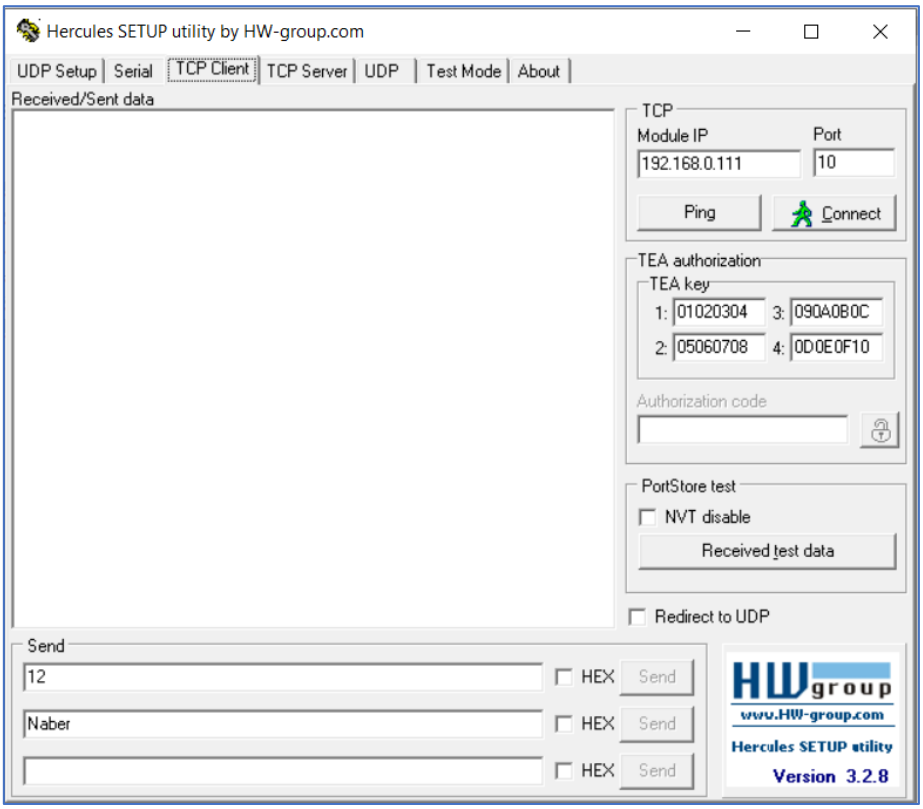

Figure 43 - Hercules TCP Client Konfigürasyonu

Ethernet için kablo bağlantısı yapıldıysa ve NUCLEO kartına program yüklenip, karta güç verildiyse bağlantı hazır demektir. ***"TCP"*** isimli bloğun altındaki ***"Ping"*** tuşuna basılırsa kart pinglenir. Eğer bağlantı başarılı ise ekranın ***"Received/Sent data"*** kısmında ***"Received ICMP ECHO REPLY"*** yazısının görülmesi gerekmektedir. Eğer ping başarılı ise ekrandaki ***"TCP"*** isimli bloğun altındaki ***"Connect"*** tuşuna basılarak bağlantı kurulabilir. Eğer bağlantı başarılı oldu ise ekranda ***"Connected to 192.168.0.111"*** yazısı görülmelidir. Yazı içindeki IP adresi server IP adresidir. Eğer server IP adresi farklı bir şekilde tanımlandıysa yazı içerisinde o adres görülecektir. Aynı zamanda **"Module IP"** kısmına, o IP adresi girilmedilir. Aksi takdirde zaten bağlantı başarılı olmayacaktır.

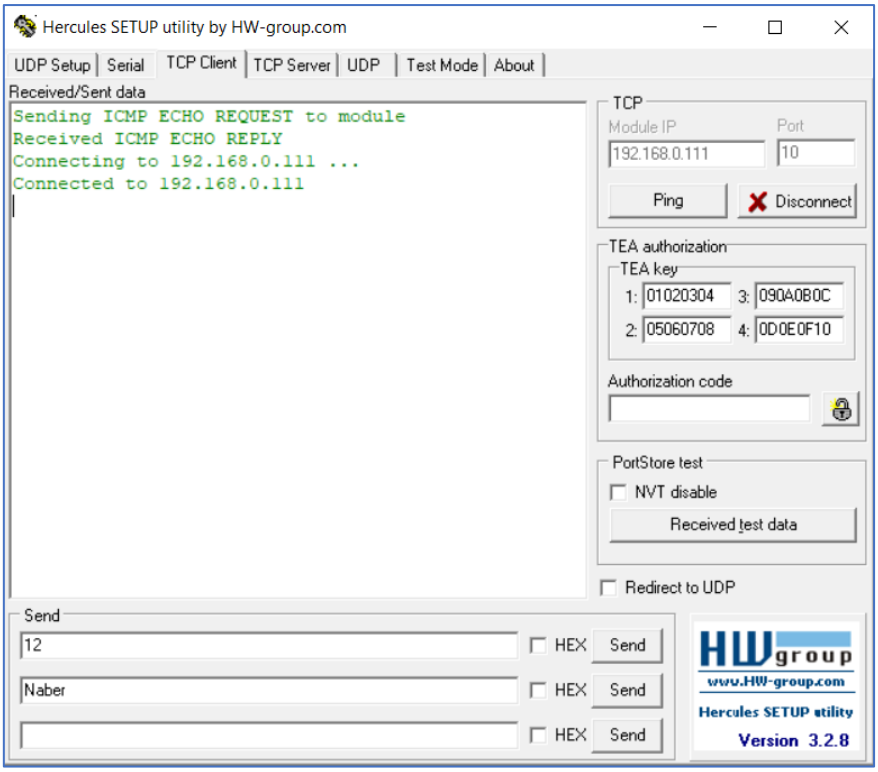

Figure 44 - TCP Server ve Client Arasındaki Bağlantının Başarılı Olması Durumu

Bağlantı başarılı bir şekilde kurulduysa, artık veri gönderilip alınabilir. NUCLEO kartına yüklenen mevcut yazılım konfigürasyonunda, server tarafından veri alınabilmesi için önce client tarafından veri gönderilmelidir. Herhangi, rastgele bir veri gönderilenebilir.

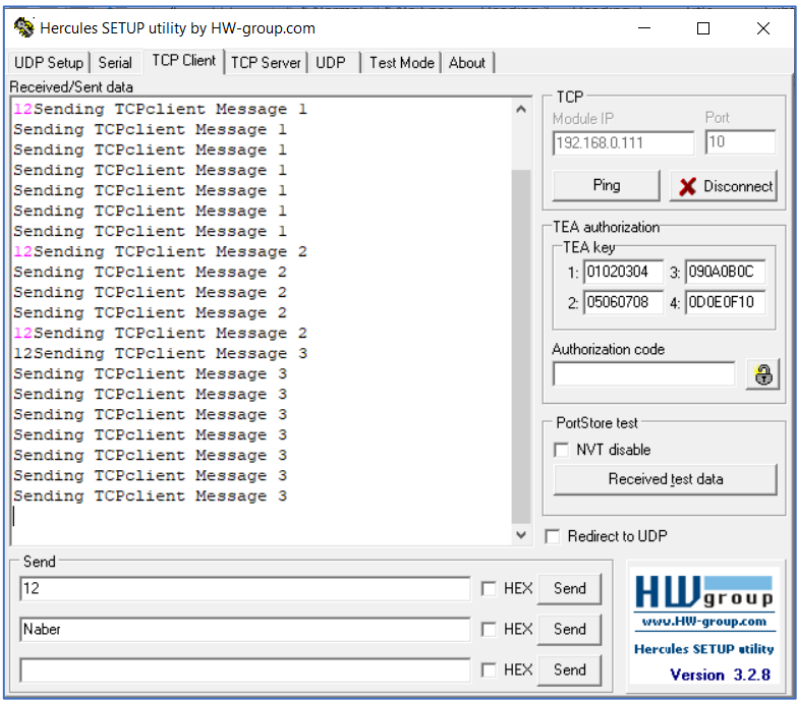

Figure 45 - Server ve Client Tarafından Gönderilen Veriler

Figür 43'te görülen ***"Received/Sent data"*** ekranın altındaki mesajlarda pembe renkli olan mesajlar client tarafından yani Hercules üzerinden bizim gönderdiğimiz mesajlar, siyah renkli olanlar ise server tarafından client'a gönderilen mesajlardır. Görüldüğü üzere client tarafından ilk veri gönderildikten sonra server 1 saniye aralıklarla veri göndermeye başlamıştır. Bir saniye aralıklarla olmasının sebebi ***"Clock Konfigürasyonu"*** kısmında anlatıldığı gibi timer süresinin 1 saniyeye göre ayarlanmış olmasıdır. Client tarafından her veri geldiğinde, server tarafından gönderilen mesajın sonundaki counter değeri artmaktadır. Bir nevi client tarafından gelen mesajları sayan bir system gibi çalışmaktadır kod.

b. TCP/IP Client

"tcp_client_init" fonksiyonu içerisinde client cihazının IP adresi ve portu belirlenmişti. Raporun önceki kısımlarında görülebilir. ***"ioc"*** dosyası içerisinde de aynı şekilde tanımlı ise bağlantı yapılabilir. Bağlantı için ***"Hercules"*** program kullanılacaktır. Verilen link üzerinden uygulama indirilebilir. Program direkt olarak exe üzerinden çalışmaktadır ve kuruluma ihtiyaç yoktur. Program kurulduktan sonra ***"TCP Server"*** sekmesi açılır. Bu sekmede program TCP bağlantısında server gibi davranacaktır. ***"Server Status"*** isimli bloğun içerisinde ***"Port"*** kısmına haberleşme için kullanılan ethernet portu yazılır.

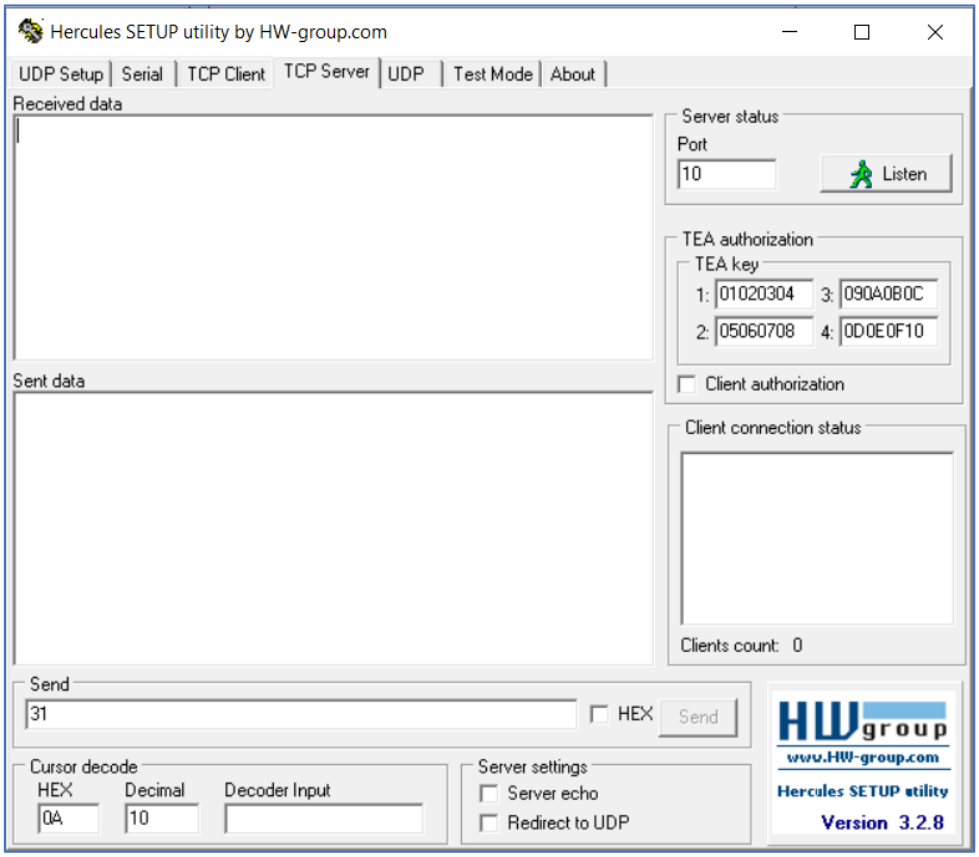

Figure 46 - Hercules Programında TCP Server Konfigürasyonu

Eğer port değeri doğru girildiyse ***"Server Status"*** bloğunun altındaki ***"Listen"*** tuşuna basıldığı durumda, ***"Client connection status"*** bloğunun altındaki ekranda ***"Client Connected"*** yazısı görülecektir. Bu NUCLEO kartının Hercules program üzerinde oluşturulan TCP Server'a bağlandığını göstermektedir. Bağlandıktan hemen sonra mevcut yazılım konfigürasyonuna göre ***"Received data"*** kısmında client tarafından gönderilen veriler gözükecektir. Server yazılımında yapılan konfigürasyonda olduğu gibi client yazılımında da karşı taraftan, burada karşı taraf server oluyor, veri geldikçe client tarafından gelen verinin sonundaki counter artacaktır. Figür 47'de görüldüğü gibi ***"Sent data"*** kısmında iki kez ***"30"*** verisi gönderilmiştir. ***"Received data"*** kısmında mesajın sonunda counter değerinin 2 kez arttığı görülebilir.

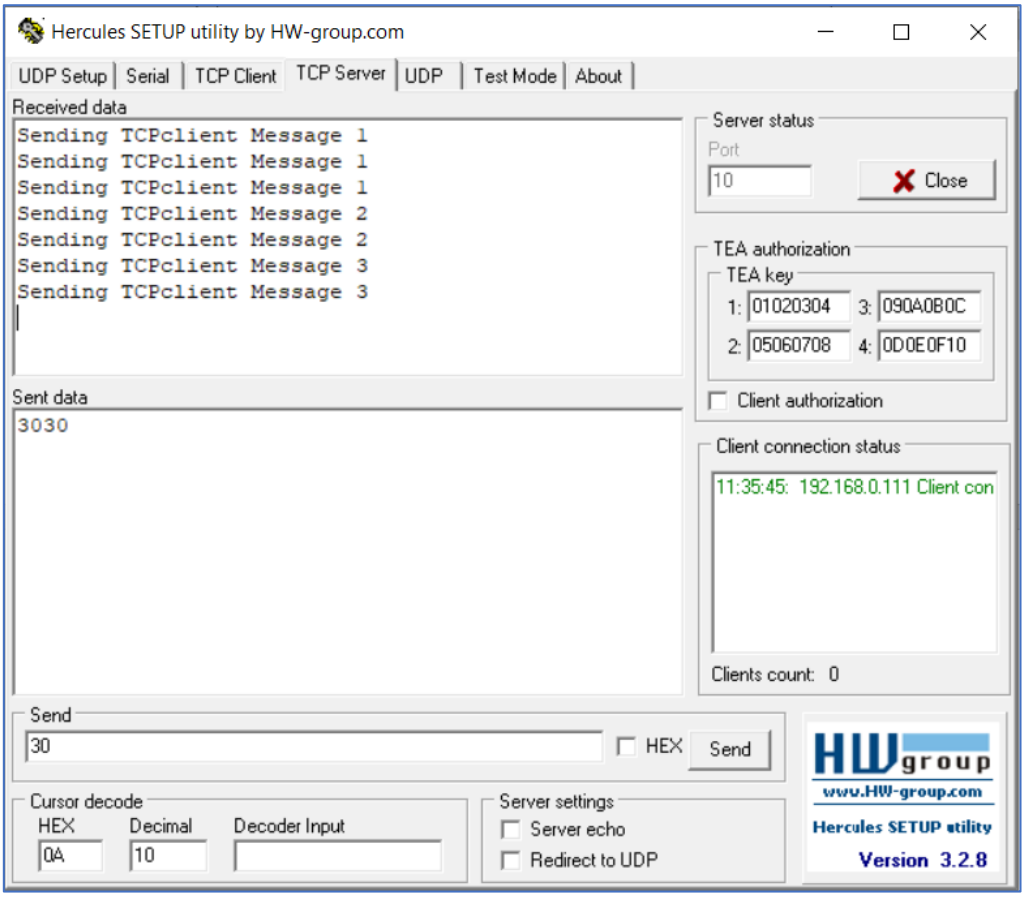

Figure 47 - TCP Server ve Client Tarafından Gönderilen Veriler

Client dinleme konusunda client'a bağlanırken sorun yaşanabilir. Bu durumda ***"Listen"*** tuşuna basıldıktan sonra NUCLEO kartına reset atılabilir.