Hello,
I tryed to create new project in STM32CubeMx with Ethernet support based on DV83848.
Last time, when I did it, everything going well and my project works fine. But, after updating (even reinstalling) STM32CubeMx and HAL drivers to the latest version it stops to work. I meant, if I migrate in STM32CubeMx, my TCP TelNet Server (based on the RAW API) don't working, but ICMP still works fine.
So, I decide to make a new project. But it's didn't help me to make TCP RAW API work normaly it still working as not predictable - only ICMP. The same library that works on the old oproject don't work on the new generated on the last version STM32CubeMX.
I tryed to place breakpoints in debug mode and check does stm32 use the tcp_server_accept() and tcp_server_recv() functions the result - no, it don't stops in this functions and don't call them.
Also I used Wireshark to analyze and understand why it's not worknig, the result image attached. Farther I placed this project on GItHub LCC_GATE_0 it's for MDK5.
Does any one can help me to solve this issue? I read ST manuals but not found the solution.

Best regards,
Sergii Kirichok

Attachments are bellow .

main.c (For reducing size I deleted the comments and few not nescessary functions)

```
...

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f1xx_hal.h"
#include "lwip.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/
UART_HandleTypeDef huart2;
UART_HandleTypeDef huart3;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_USART3_UART_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/
extern void TCP_server_init(void);
/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

int main(void)
{

/* USER CODE BEGIN 1 */
```

```
/* USER CODE END 1 */

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

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

/* USER CODE BEGIN Init */

/* USER CODE END Init */

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

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_LWIP_Init();
MX_USART2_UART_Init();
MX_USART3_UART_Init();

/* USER CODE BEGIN 2 */
TCP_server_init();
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
MX_LWIP_Process();
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

}
/* USER CODE END 3 */

}

/** System Clock Configuration
*/
void SystemClock_Config(void)
{

RCC_OscInitTypeDef RCC_OscInitStruct;
RCC_ClkInitTypeDef RCC_ClkInitStruct;

/**Initializes the CPU, AHB and APB busses clocks
*/
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
RCC_OscInitStruct.HSIState = RCC_HSI_ON;
RCC_OscInitStruct.Prediv1Source = RCC_PREDIV1_SOURCE_HSE;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
```

```
RCC_OscInitStruct.PLL2.PLL2State = RCC_PLL_NONE;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
_Error_Handler(__FILE__, __LINE__);
}

/**Initializes the CPU, AHB and APB busses 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_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
{
_Error_Handler(__FILE__, __LINE__);
}

HAL_RCC_MCOConfig(RCC_MCO, RCC_MCO1SOURCE_HSE, RCC_MCODIV_1);

/**Configure the Systick interrupt time
*/
HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

/**Configure the Systick
*/
HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

/**Configure the Systick interrupt time
*/
__HAL_RCC_PLLI2S_ENABLE();

/* SysTick_IRQn interrupt configuration */
HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* USART2 init function */
static void MX_USART2_UART_Init(void)
{

huart2.Instance = USART2;
huart2.Init.BaudRate = 9600;
huart2.Init.WordLength = UART_WORDLENGTH_8B;
huart2.Init.StopBits = UART_STOPBITS_1;
huart2.Init.Parity = UART_PARITY_NONE;
huart2.Init.Mode = UART_MODE_TX_RX;
huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart2.Init.OverSampling = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart2) != HAL_OK)
{
_Error_Handler(__FILE__, __LINE__);
}

}

/* USART3 init function */
static void MX_USART3_UART_Init(void)
{
```

```
huart3.Instance = USART3;
huart3.Init.BaudRate = 115200;
huart3.Init.WordLength = UART_WORDLENGTH_8B;
huart3.Init.StopBits = UART_STOPBITS_1;
huart3.Init.Parity = UART_PARITY_NONE;
huart3.Init.Mode = UART_MODE_TX_RX;
huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart3.Init.OverSampling = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart3) != HAL_OK)
{
_Error_Handler(__FILE__, __LINE__);
}

}

/** Configure pins as
* Analog
* Input
* Output
* EVENT_OUT
* EXTI
PA8 ------> RCC_MCO
*/
static void MX_GPIO_Init(void)
{

GPIO_InitTypeDef GPIO_InitStruct;

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

/*Configure GPIO pin : PA8 */
GPIO_InitStruct.Pin = GPIO_PIN_8;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
* @brief This function is executed in case of error occurrence.
* @param None
* @retval None
*/

...
```

tcp_server.c

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

```
#include "lwip/udp.h"
#include "lwip/tcp.h"
#include <string.h>
#include <stdio.h>

//#include "usbd_cdc_if.h"


/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
#ifndef TCP_PORT
#define TCP_PORT 23 /* define the TCP connection port */
#endif

#define MAX_NAME_SIZE 64

struct name
{
int length;
char bytes[MAX_NAME_SIZE];
};

/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/

/* Private function prototypes -----------------------------------------------*/
void udp_server_callback(void *arg, struct udp_pcb *upcb, struct pbuf *p, struct ip_addr *addr,
u16_t port);
err_t tcp_server_accept(void *arg, struct tcp_pcb *pcb, err_t err);
static err_t tcp_server_recv(void *arg, struct tcp_pcb *pcb, struct pbuf *p, err_t err);

/* Private functions ---------------------------------------------------------*/


void TCP_server_init(void)
{
struct tcp_pcb *pcb;
ip_addr_t *addr;// == NULL;

/* Create a new TCP control block */
pcb = tcp_new();

if(pcb !=NULL) {
err_t err;

/* Assign to the new pcb a local IP address and a port number */
err = tcp_bind(pcb, addr, TCP_PORT);

if(err != ERR_USE){
/* Set the connection to the LISTEN state */
pcb = tcp_listen(pcb);

/* Specify the function to be called when a connection is established */
tcp_accept(pcb, tcp_server_accept);

}else{
/* We enter here if a conection to the addr IP address already exists */
/* so we don't need to establish a new one */
tcp_close(pcb);
```

```
}
}
}


/**
* @brief This funtion is called when a TCP connection has been established on the port
TCP_PORT.
* @param arg user supplied argument
* @param pcb the tcp_pcb which accepted the connection
* @param err error value
* @retval ERR_OK
*/
err_t tcp_server_accept(void *arg, struct tcp_pcb *pcb, err_t err)
{
/* Tell LwIP to associate this structure with this connection. */
tcp_arg(pcb, mem_calloc(sizeof(struct name), 1));

/* Specify the function that should be called when the TCP connection receives data */
tcp_recv(pcb, tcp_server_recv);

/* Send out the first message */
tcp_write(pcb, "LCC GATE\r\n", 10, 1);
tcp_write(pcb, "FirmWare ver.0.0.0.a", 13, 1); //FWVersion
tcp_write(pcb, "\r\n", 2, 1); //FWVersion

tcp_write(pcb, ">", 1, 1);
//tcp_write(pcb, "SilmpleSolutions\r\n", 18, 1);

return ERR_OK;
}

/**
* @brief This function is called when a data is received over the TCP_PORT.
* The received data contains the number of the led to be toggled.
* @param arg user supplied argument
* @param pcb the tcp_pcb which accepted the connection
* @param p the packet buffer that was received
* @param err error value
* @retval ERR_OK
*/
static err_t tcp_server_recv(void *arg, struct tcp_pcb *pcb, struct pbuf *p, err_t err)
{
struct pbuf *q;
struct name *name = (struct name *)arg;
int done;
char *data;
int i;
//char answer[32] = "";
//int answerNumb = 0;

if (p != NULL)
{
/* We call this function to tell the LwIp that we have processed the data */
/* This lets the stack advertise a larger window, so more data can be received*/
tcp_recved(pcb, p->tot_len);

/* Check the name if NULL, no data passed, return withh illegal argument error */
if(!name)
```

```
{
pbuf_free(p);
return ERR_ARG;
}

done = 0;
for(q=p; q != NULL; q = q->next)
{
data = q->payload;
for(i=0; i<q->len && !done; i++)
{
done = ((data[i] == '\r') || (data[i] == '\n')); //Here we are got the end of string

if(name->length < MAX_NAME_SIZE)
{
name->bytes[name->length++] = data[i];
}
}
}


if(done) //it means that we got the packet with \r or \n on the end
{
if(name->bytes[name->length-2] != '\r' || name->bytes[name->length-1] != '\n') //checking that it
ends with /r/n
{
if((name->bytes[name->length-1] == '\r' || name->bytes[name->length-1] == '\n') && (name-
>length+1 <= MAX_NAME_SIZE))
{
name->length += 1;
}
else if(name->length+2 <= MAX_NAME_SIZE)
{
name->length += 2;
}
else
{
name->length = MAX_NAME_SIZE;
}

name->bytes[name->length-2] = '\r';
name->bytes[name->length-1] = '\n';
}


if((strncmp(name->bytes, "?", 1) == 0) || (strncmp(name->bytes, "help", 4) == 0)) // ------- ? -------
-----//
{
//showHelpMessage(pcb);
tcp_write(pcb, "It Works\r\n", 10, 1);
}

else if(strncmp(name->bytes, "exit", 4) == 0)
{
//pbuf_free(p);
mem_free(name);
tcp_close(pcb);
}
```

```
if(strncmp(name->bytes, "exit", 4) != 0) //It's no exit command, we could to add some data to the
string
{
tcp_write(pcb, ">", 1, 1);
}

name->length = 0;
}

/* End of processing, we free the pbuf */
pbuf_free(p);

}
else if (err == ERR_OK) //Pbuff NULL!
{
/* When the pbuf is NULL and the err is ERR_OK, the remote end is closing the connection. */
/* We free the allocated memory and we close the connection */
mem_free(name);
return tcp_close(pcb);
}

return ERR_OK;
}

...
```