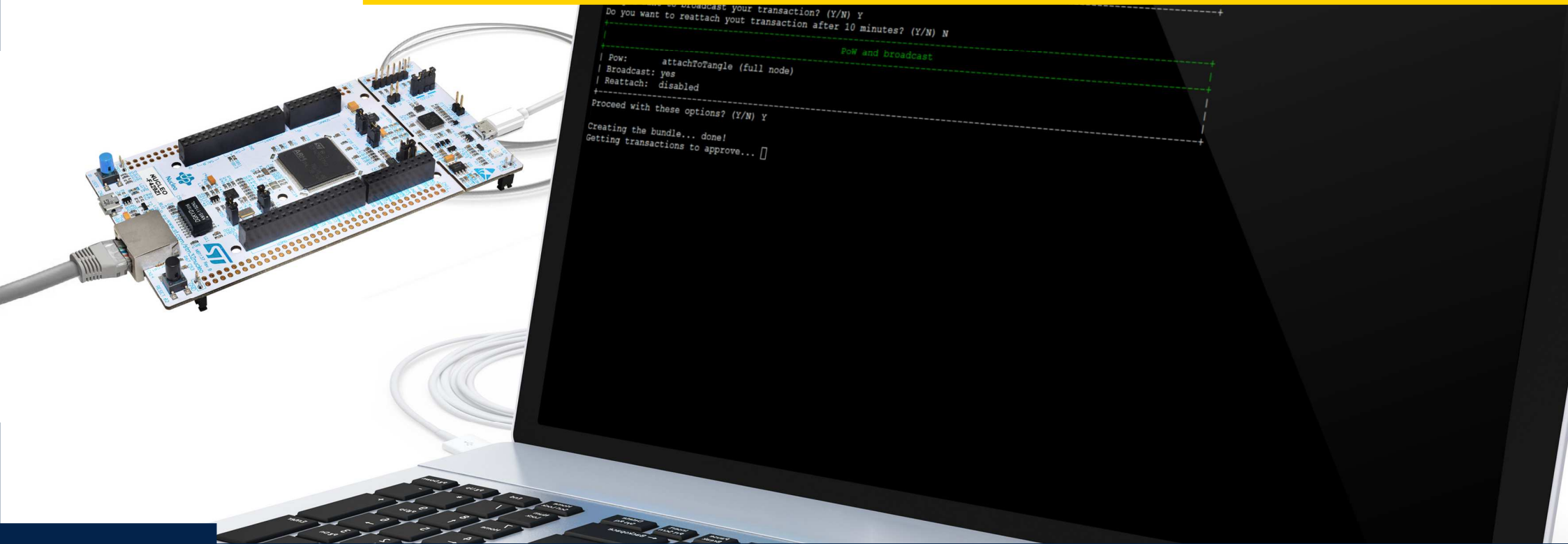


# Hands on

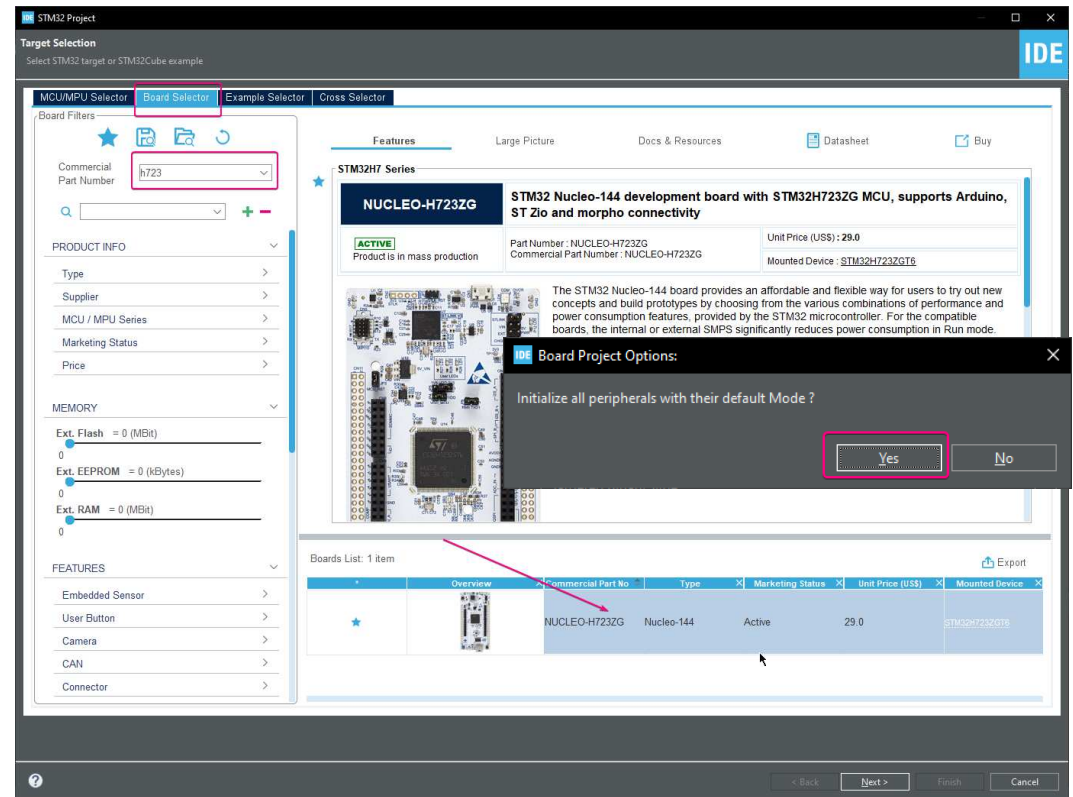


# Features applications

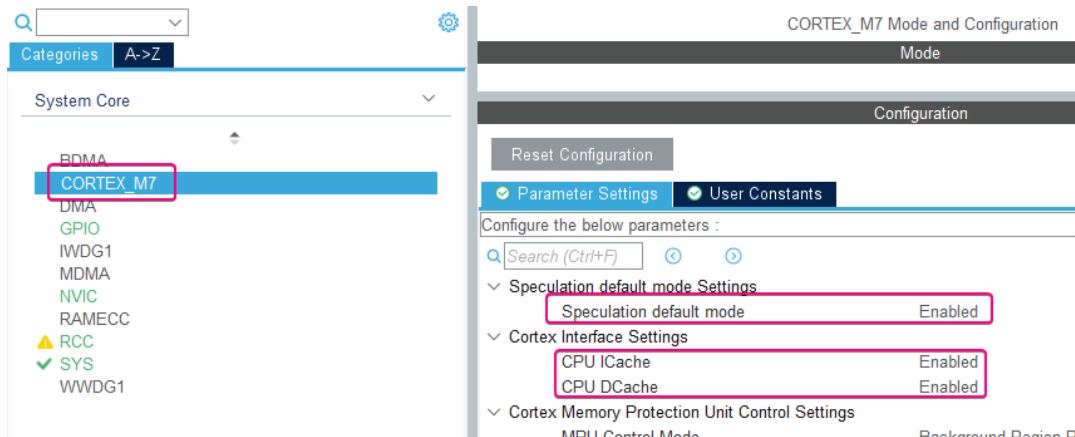
- Web server based on raw API:
- **GOAL**> Implement a web server based on the LwIP Socket API with FreeRTOS.
- Use Case: Connect to the STM32 MCU from a web client and to load HTML pages
- The web server application implements the following features:
  - URL parsing
  - CGI (Common Gateway Interface)
  - SSI (Server Side Includes)
  - Dynamic Header generation
  - HTTP Post request

# From Scratch

- Create a new Project using STM32CubeIDE
- Select the NUCLEO-H723ZG board
- Give it a name
- Initialize the peripherals



- Configure the MPU base settings and 2 regions to store the Ethernet descriptors



MPU Region	MPU Region Base Address	MPU Region Size	MPU SubRegion Disable	MPU TEX field level	MPU Access Permission	MPU Instruction Access	MPU Shareability Permission	MPU Cacheable Permission	MPU Bufferable Permission
<ul style="list-style-type: none"> <li>✓ Cortex Memory Protection Unit Region 1 Settings</li> </ul>	Enabled	0x30000000	256B	0x0	level 0	ALL ACCESS PERMITTED	ENABLE	DISABLE	DISABLE
<ul style="list-style-type: none"> <li>MPU Bufferable Permission</li> </ul>	ENABLE								
<ul style="list-style-type: none"> <li>✓ Cortex Memory Protection Unit Region 2 Settings</li> </ul>	Enabled	0x30004000	16KB	0x0	level 1	ALL ACCESS PERMITTED	ENABLE	DISABLE	DISABLE
<ul style="list-style-type: none"> <li>MPU Cacheable Permission</li> </ul>	DISABLE								
<ul style="list-style-type: none"> <li>MPU Bufferable Permission</li> </ul>	DISABLE								

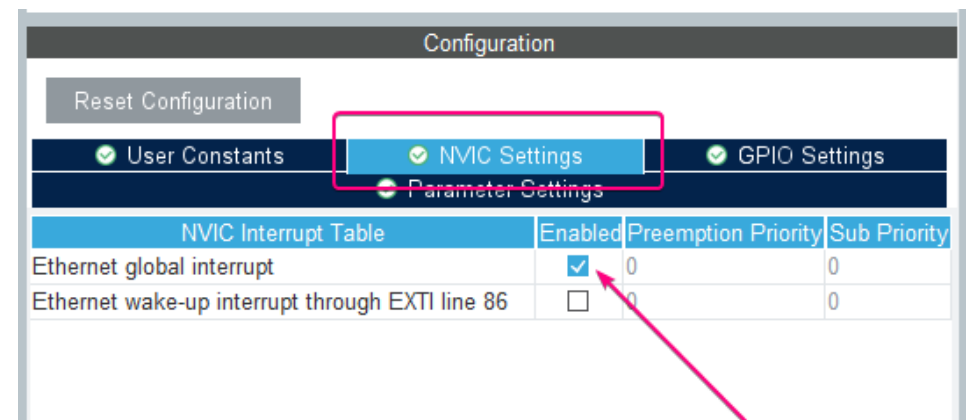
# Ethernet

- The First Tx Descriptor Addr:
  - 0x30000080
- The Rx Descriptor Addr:
  - 0x30000000
- Reduce Rx buffers length:
  - 1000

The screenshot displays the STM32CubeMX configuration tool. On the left, the 'Pinout & Configuration' pane shows a tree view of components, with 'Connectivity' and 'ETH' highlighted. The main area shows the 'ETH Mode and Configuration' settings. The 'Mode' is set to 'RMII'. Under 'Configuration', the 'General: Ethernet Configuration' section is expanded, showing the following parameters:

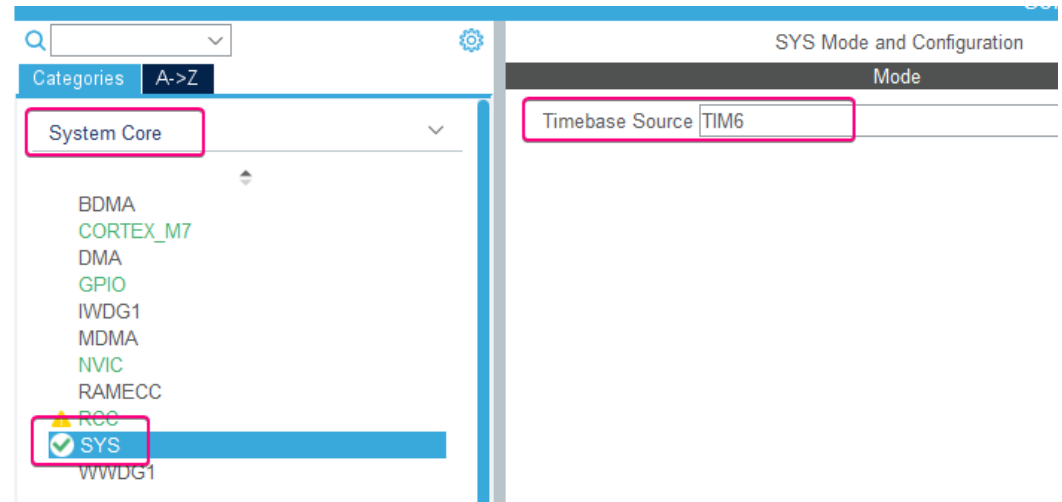
Parameter	Value
Ethernet MAC Address	00:80:E1:00:00:00
Tx Descriptor Length	4
First Tx Descriptor Address	0x30000080
Rx Descriptor Length	4
First Rx Descriptor Address	0x30000000
Rx Buffers Length	1000

- Since we are using an RTOS, enable the global eth IT in the NVIC



# System

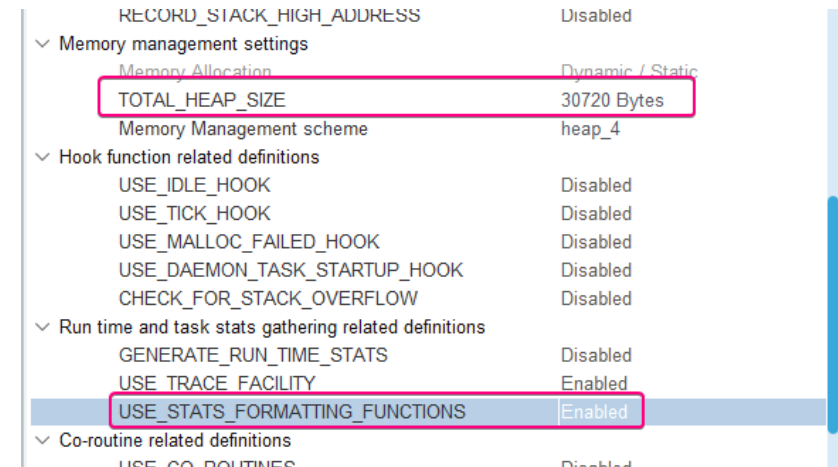
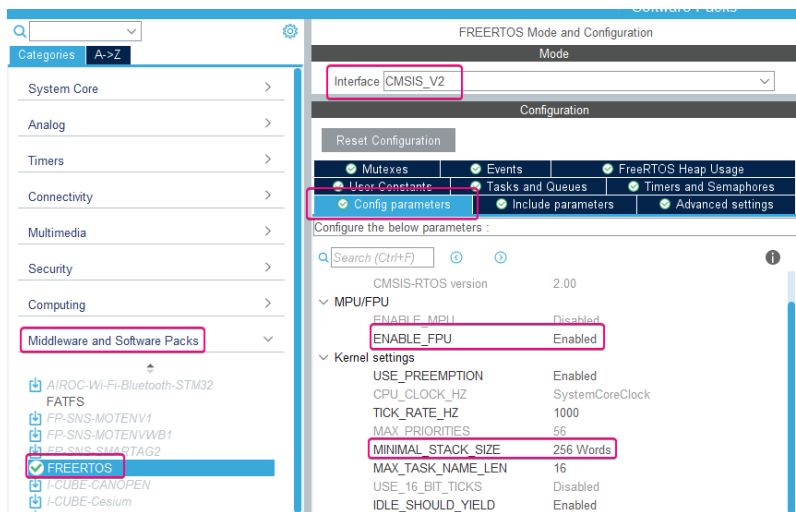
- Free up the SYSTICK for the RTOS by changing the HAL timebase
  - We can use TIM6



# FreeRTOS

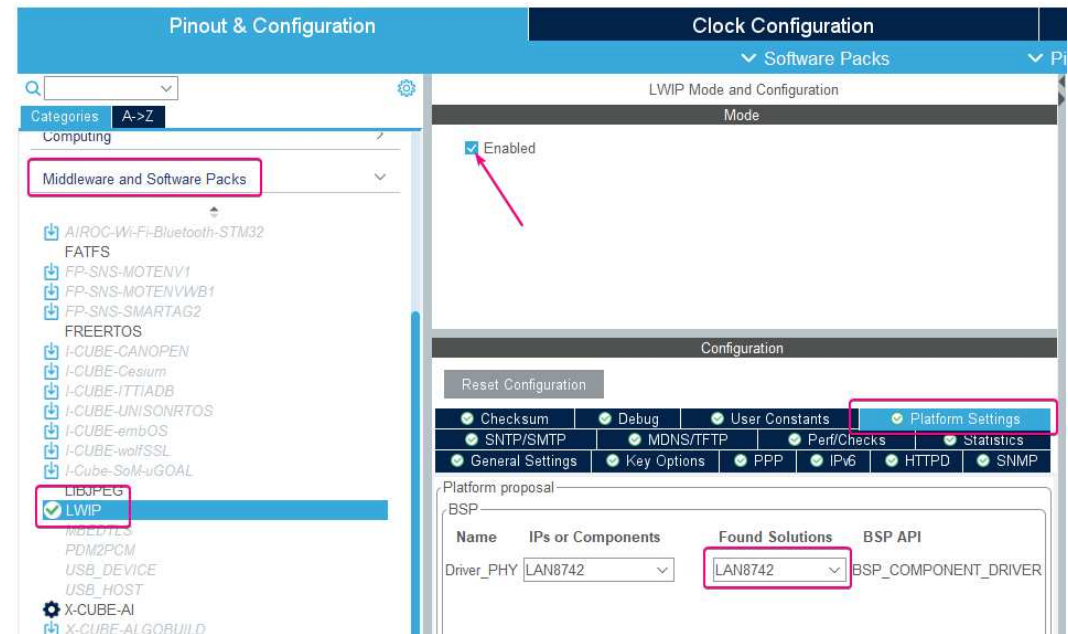
- In the middlewares tab, select FreeRTOS and choose CMSIS V2 interface.
- Increase stack size and enable FPU.

- Also increase the heap size and enable stats formatting functions.

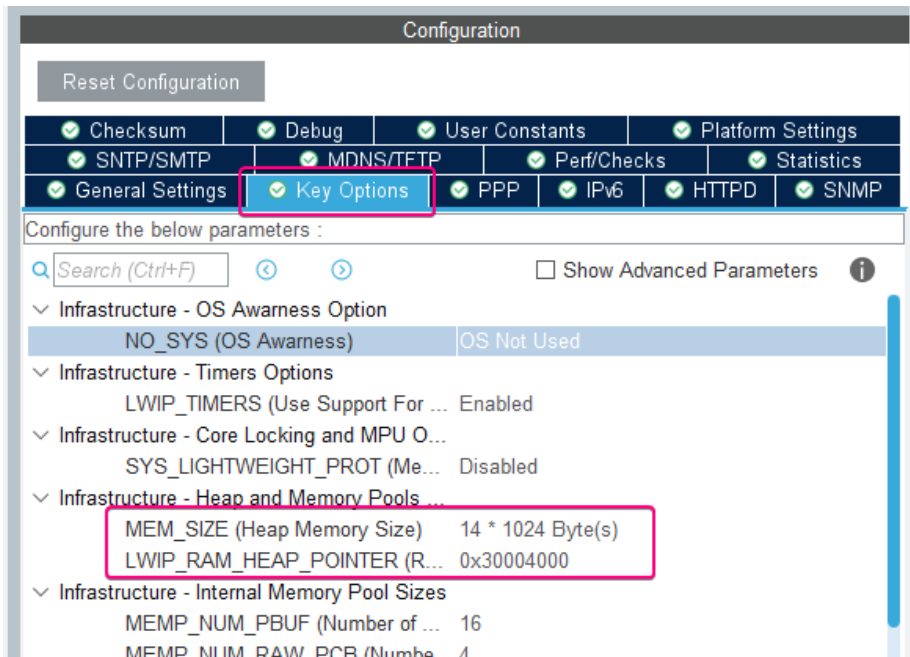




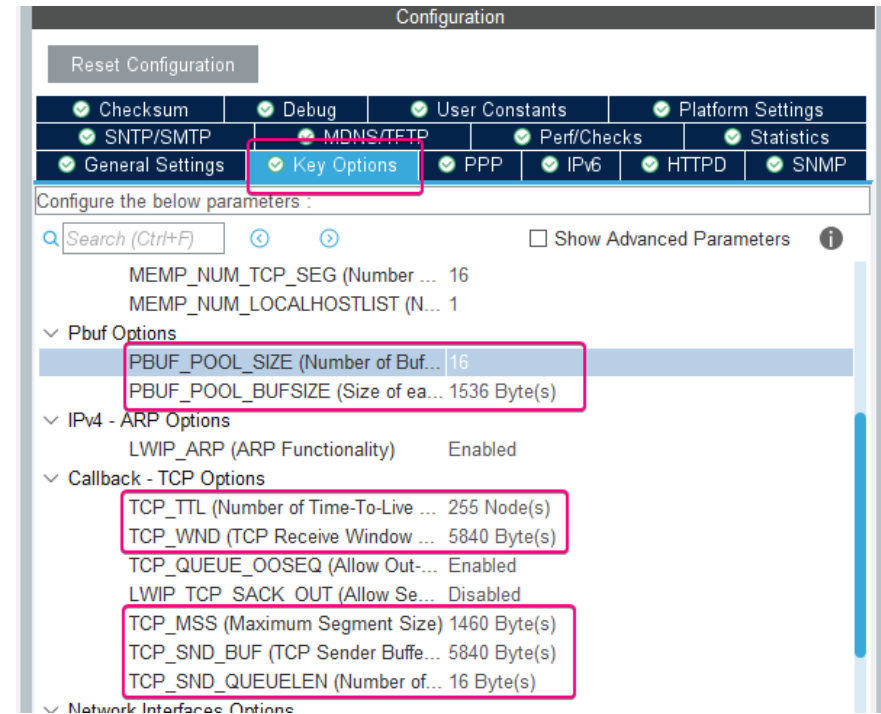
- Enable the LwIP middleware
- In the platform settings enable the phy driver for LAN8742



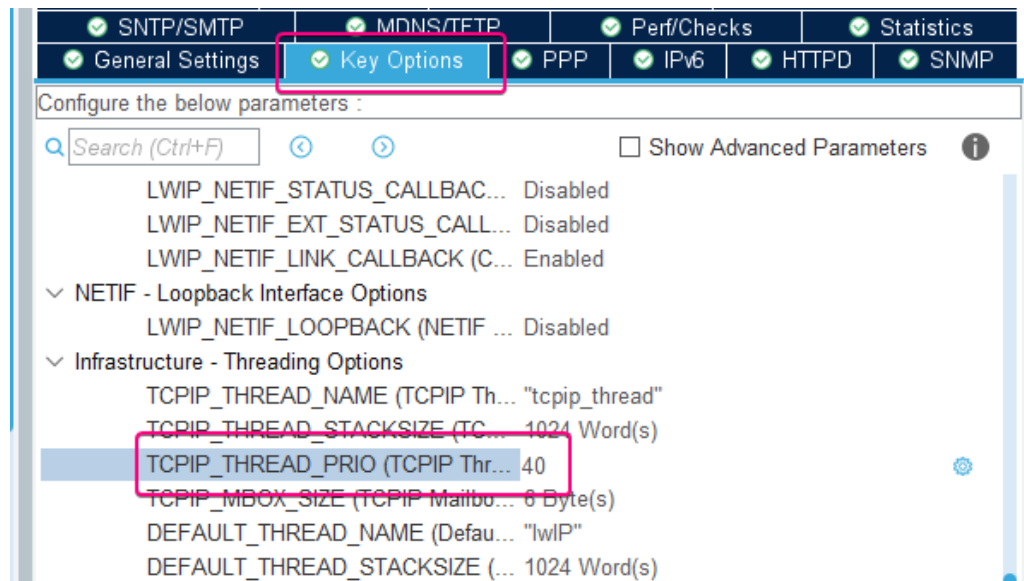
- Increase the heap size and change it's address



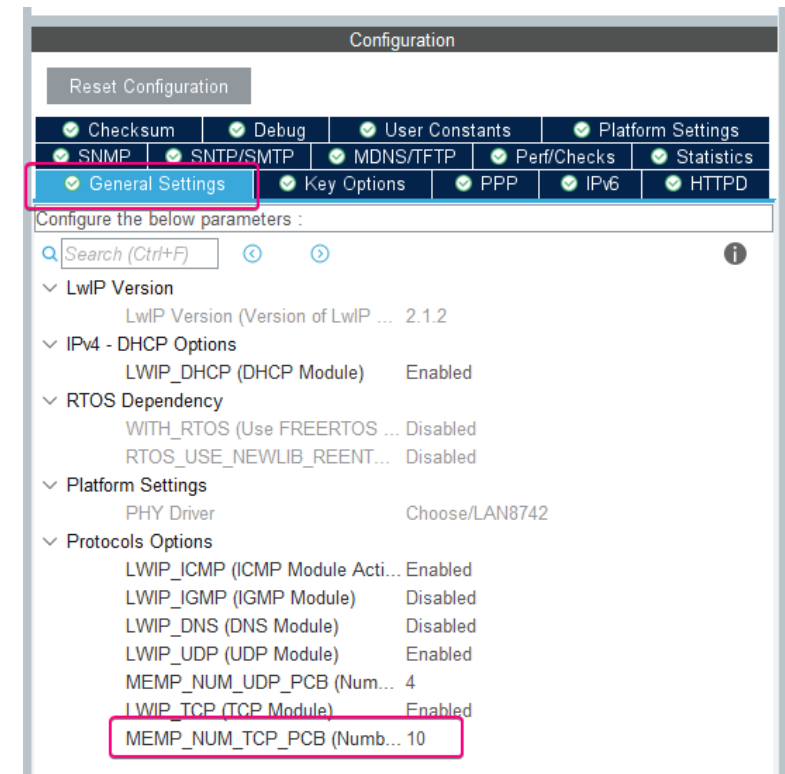
- Increase a few Pbuf and TCP parameters



- Change the TCPIP thread priority to 40

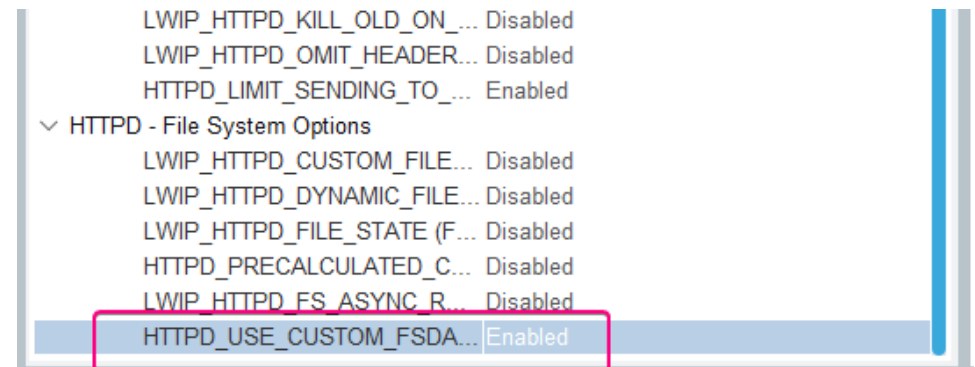
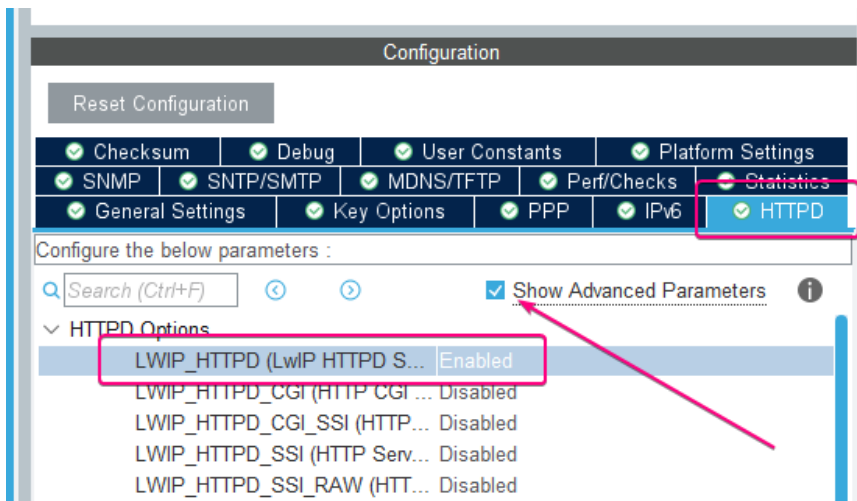


- Also increase the number of TCP PCBs



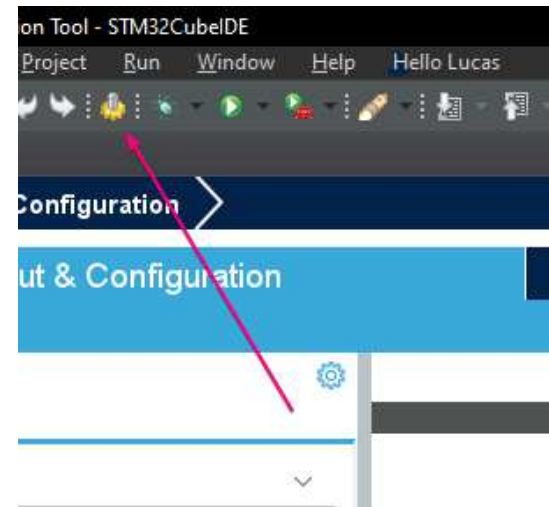
- In the HTTPD, enable it and show the advanced parameters

- Scroll all the way down and enable the custom data setting



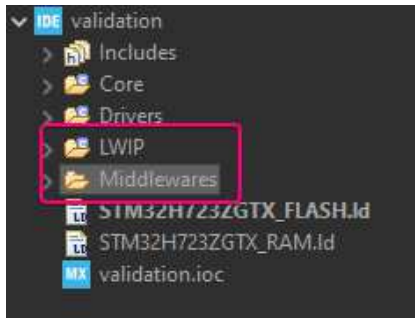
# Check Point - Configuration wrap up

- Generate the code
- Don't build it yet 😊

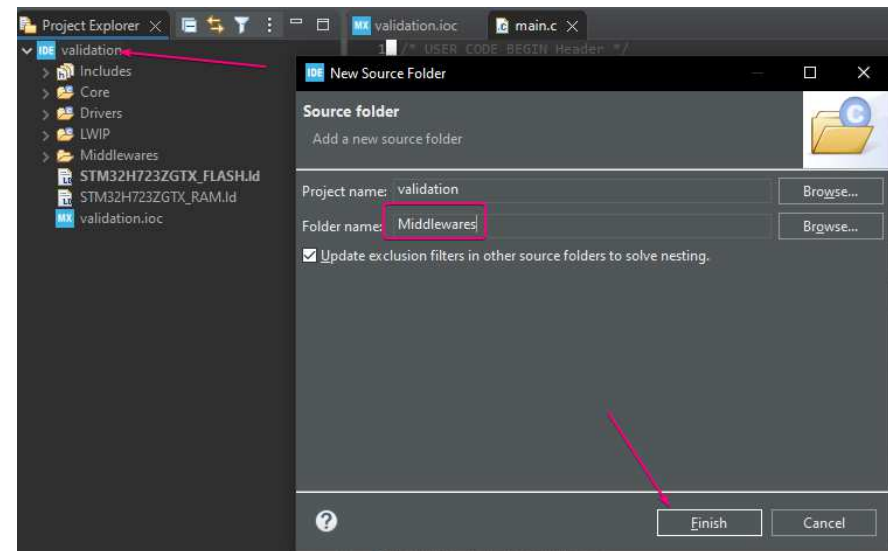


# STM32CubeIDE 1.15.0 bug found

- There's currently a bug in STM32CubeIDE 1.15.0 where the Middleware folder may not be a source folder



- To fix this, left click the Project and create a new source folder called Middlewares



# main.c – USER CODE BEGIN and PFP

- Add 2 includes and the putchar implementation

```
/* USER CODE BEGIN Includes */
#include "lwip/tcpip.h"
#include "stdio.h"
/* USER CODE END Includes */
```

```
/* USER CODE BEGIN PFP */
int __io_putchar(int ch) {
    HAL_UART_Transmit(&huart3, (uint8_t*) &ch, 1, 0xFFFF);
    return ch;
}
/* USER CODE END PFP */
```

```
23
24
25 /* Private includes -----*/
26 #include "lwip/tcpip.h"
27 #include "stdio.h"
28
29
30 /* Private typedef -----*/
31 /* USER CODE BEGIN PTD */
32
33 /* USER CODE END PTD */
34
35 /* Private define -----*/
36 /* USER CODE BEGIN PD */
37
38 /* USER CODE END PD */
39
40 /* Private macro -----*/
41 /* USER CODE BEGIN PM */
42
43 /* USER CODE END PM */
44
45 /* Private variables -----*/
46
47 UART_HandleTypeDef huart3;
48
49 /* Definitions for defaultTask */
50 osThreadId_t defaultTaskHandle;
51 const osThreadAttr_t defaultTask_attributes = {
52     .name = "defaultTask",
53     .stack_size = 256 * 4,
54     .priority = (osPriority_t) osPriorityNormal,
55 };
56 /* USER CODE BEGIN PV */
57
58 /* USER CODE END PV */
59
60 /* Private function prototypes -----*/
61 void SystemClock_Config(void);
62 static void MPU_Config(void);
63 static void MX_GPIO_Init(void);
64 static void MX_USART3_UART_Init(void);
65 static void MX_USB_OTG_HS_USB_Init(void);
66 void StartDefaultTask(void *argument);
67
68 /* USER CODE BEGIN PFP */
69 int __io_putchar(int ch) {
70     /* Place your implementation of fputc here */
71     /* e.g. write a character to the USART3 and Loop until the end of transmission */
72     HAL_UART_Transmit(&huart3, (uint8_t*) &ch, 1, 0xFFFF);
73     return ch;
74 }
75 /* USER CODE END PFP */
76
```

# main.c – USER CODE BEGIN 2 and 5

- In the USER CODE 2 section, add a few application start prints

```
/* USER CODE BEGIN 2 */
printf("LwIP_HTTP_Server_Socket_RTOS\n");
printf("NUCLEO-H723ZG board\n");
printf("State: Ethernet Initialization ...\n");
/* USER CODE END 2 */
```

```
/* USER CODE BEGIN 2 */
printf("LwIP_HTTP_Server_Socket_RTOS\n");
printf("NUCLEO-H723ZG board\n");
printf("State: Ethernet Initialization ...\n");
/* USER CODE END 2 */
```

- Use the default task to start the http server, then kill the thread.

```
/* USER CODE END Header_StartDefaultTask */
void StartDefaultTask(void *argument)
{
    /* Init code for LWIP */
    MX_LWIP_Init();
    /* USER CODE BEGIN 5 */

    http_server_socket_init();
    /* Infinite loop */
    for(;;)
    {
        osThreadTerminate(defaultTaskHandle);
    }
    /* USER CODE END 5 */
}
```

```
/* USER CODE BEGIN 5 */

http_server_socket_init();
/* Infinite loop */
for(;;)
{
    osThreadTerminate(defaultTaskHandle);
}
/* USER CODE END 5 */
```



# main.h – USER CODE BEGIN EC

- In the exported constants section, add the static IP gateway and netmask addresses.

```
/* USER CODE BEGIN EC */
#define IP_ADDR0 ((uint8_t)192U)
#define IP_ADDR1 ((uint8_t)168U)
#define IP_ADDR2 ((uint8_t)0U)
#define IP_ADDR3 ((uint8_t)10U)

/*NETMASK*/
#define NETMASK_ADDR0 ((uint8_t)255U)
#define NETMASK_ADDR1 ((uint8_t)255U)
#define NETMASK_ADDR2 ((uint8_t)255U)
#define NETMASK_ADDR3 ((uint8_t)0U)

/*Gateway Address*/
#define GW_ADDR0 ((uint8_t)192U)
#define GW_ADDR1 ((uint8_t)168U)
#define GW_ADDR2 ((uint8_t)0U)
#define GW_ADDR3 ((uint8_t)1U)
/* USER CODE END EC */
```

```
42 /* EXPORTED CONSTANTS
43 /* USER CODE BEGIN EC */
44 #define IP_ADDR0 ((uint8_t)192U)
45 #define IP_ADDR1 ((uint8_t)168U)
46 #define IP_ADDR2 ((uint8_t)0U)
47 #define IP_ADDR3 ((uint8_t)10U)
48
49 /*NETMASK*/
50 #define NETMASK_ADDR0 ((uint8_t)255U)
51 #define NETMASK_ADDR1 ((uint8_t)255U)
52 #define NETMASK_ADDR2 ((uint8_t)255U)
53 #define NETMASK_ADDR3 ((uint8_t)0U)
54
55 /*Gateway Address*/
56 #define GW_ADDR0 ((uint8_t)192U)
57 #define GW_ADDR1 ((uint8_t)168U)
58 #define GW_ADDR2 ((uint8_t)0U)
59 #define GW_ADDR3 ((uint8_t)1U)
60 /* USER CODE END EC */
61
```

# Iwip.h – USER CODE BEGIN 0

- On Iwip.h add the dhcp states, thread and the 2 application threads

```
/* USER CODE BEGIN 0 */
/* DHCP process states */
#define DHCP_OFF                (uint8_t) 0
#define DHCP_START              (uint8_t) 1
#define DHCP_WAIT_ADDRESS      (uint8_t) 2
#define DHCP_ADDRESS_ASSIGNED  (uint8_t) 3
#define DHCP_TIMEOUT            (uint8_t) 4
#define DHCP_LINK_DOWN         (uint8_t) 5
/* Exported functions ----- */
#if LWIP_DHCP
void DHCP_Thread(void *argument);
#endif
void http_server_socket_init(void);
void DynWebPage(int conn);
/* USER CODE END 0 */
```

```
/* USER CODE BEGIN 0 */
/* DHCP process states */
#define DHCP_OFF                (uint8_t) 0
#define DHCP_START              (uint8_t) 1
#define DHCP_WAIT_ADDRESS      (uint8_t) 2
#define DHCP_ADDRESS_ASSIGNED  (uint8_t) 3
#define DHCP_TIMEOUT            (uint8_t) 4
#define DHCP_LINK_DOWN         (uint8_t) 5
/* Exported functions ----- */
#if LWIP_DHCP
void DHCP_Thread(void *argument);
#endif
void http_server_socket_init(void);
void DynWebPage(int conn);
/* USER CODE END 0 */
```

# lwip.c – USER CODE BEGIN 0

- Add the necessary includes, the dhcp variables and a page hits Variable.

```
/* USER CODE BEGIN 0 */
#include "lwip/opt.h"
#include "main.h"
#if LWIP_DHCP
#include "lwip/dhcp.h"
#endif
#include "lwip/api.h"
#include "lwip/inet.h"
#include "lwip/sockets.h"
#include "lwip/apps/fs.h"
#include "FreeRTOS.h"
#include "task.h"
#include <stdio.h>

#if LWIP_DHCP
#define MAX_DHCP_TRIES 4
__IO uint8_t DHCP_state = DHCP_OFF;
#endif

u32_t nPageHits = 0;
```

```
30
31 /* USER CODE BEGIN 0 */
32 #include "lwip/opt.h"
33 #include "main.h"
34 #if LWIP_DHCP
35 #include "lwip/dhcp.h"
36 #endif
37 #include "lwip/api.h"
38 #include "lwip/inet.h"
39 #include "lwip/sockets.h"
40 #include "lwip/apps/fs.h"
41 #include "FreeRTOS.h"
42 #include "task.h"
43 #include <stdio.h>
44
45 #if LWIP_DHCP
46 #define MAX_DHCP_TRIES 4
47 __IO uint8_t DHCP_state = DHCP_OFF;
48 #endif
49
50 u32_t nPageHits = 0;
```

# Iwip.c – USER CODE BEGIN 0

- Add the page header const
  - From provided material
- Add the server prototype
- Add the priority define

```
#define WEBSERVER_THREAD_PRIO ( osPriorityAboveNormal )  
void http_server_serve(int conn);
```

```
146 0x73,0x74,0x79,0x6c,0x65,0x3d,0x22,0x66,0x6f,0x6e,0x74,0x2d,0x77,0x65,0x69,0x67,  
147 0x68,0x74,0x3a,0x20,0x62,0x6f,0x6c,0x64,0x3b,0x22,0x3e,0x3c,0x2f,0x73,0x70,0x61,  
148 0x6e,0x3e,0x3c,0x73,0x6d,0x61,0x6c,0x6c,0x3e,0x3c,0x73,0x70,0x61,0x6e,0x20,0x73,  
149 0x74,0x79,0x6c,0x65,0x3d,0x22,0x66,0x6f,0x6e,0x74,0x2d,0x66,0x61,0x6d,0x69,0x6c,  
150 0x79,0x3a,0x20,0x56,0x65,0x72,0x64,0x61,0x6e,0x61,0x3b,0x22,0x3e,0x4e,0x75,0x6d,  
151 0x62,0x65,0x72,0x20,0x6f,0x66,0x20,0x70,0x61,0x67,0x65,0x20,0x68,0x69,0x74,0x73,  
152 0x3a,0x26,0x6e,0x62,0x73,0x70,0x3b,0x0d,0x0a,0x0d,0x0a,0x3c,0x2f,0x73,0x70,0x61,  
153 0x6e,0x3e,0x3c,0x2f,0x73,0x6d,0x61,0x6c,0x6c,0x3e,0x0d,0x0a,0x3c,0x2f,0x62,0x6f,  
154 0x64,0x79,0x3e,0x3c,0x2f,0x68,0x74,0x6d,0x6c,0x3e, 0  
155 };  
156 #define WEBSERVER_THREAD_PRIO ( osPriorityAboveNormal )  
157  
158 void http_server_serve(int conn);  
159 /* USER CODE END 0 */
```

# Iwip.c – USER CODE BEGIN 1

- In the USER CODE 1 section create 2 thread handles and their attributes

```
/* USER CODE BEGIN 1 */
osThreadId_t EthLinkHandle;
osThreadId_t DHCPHandle;
const osThreadAttr_t EthLinkThread_attributes = {
    .name = "EthLink",
    .stack_size = 256 * 4,
    .priority = osPriorityNormal
};

const osThreadAttr_t DHCPThread_attributes = {
    .name = "DHCP",
    .stack_size = 256 * 4,
    .priority = osPriorityBelowNormal
};
/* USER CODE END 1 */
```

```
165 /* USER CODE BEGIN 1 */
166 osThreadId_t EthLinkHandle;
167 osThreadId_t DHCPHandle;
168 const osThreadAttr_t EthLinkThread_attributes = {
169     .name = "EthLink",
170     .stack_size = 256 * 4,
171     .priority = osPriorityNormal
172 };
173
174 const osThreadAttr_t DHCPThread_attributes = {
175     .name = "DHCP",
176     .stack_size = 256 * 4,
177     .priority = osPriorityBelowNormal
178 };
179 /* USER CODE END 1 */
180
```

- Add the main application code, starting by the DHCP process thread

```

/* USER CODE BEGIN 2 */
#if LWIP_DHCP
void DHCP_Thread(void *argument)
{
  struct netif *netif = (struct netif *) argument;
  ip_addr_t ipaddr;
  ip_addr_t netmask;
  ip_addr_t gw;
  struct dhcp *dhcp;
  uint8_t iptxt[20];
  for (;;)
  {
    switch (DHCP_state)
    {
      case DHCP_START:
      {
        printf("State: Looking for DHCP server ...\n");
        HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_RESET);
        ip_addr_set_zero_ip4(&netif->ip_addr);
        ip_addr_set_zero_ip4(&netif->netmask);
        ip_addr_set_zero_ip4(&netif->gw);
        dhcp_start(netif);
        DHCP_state = DHCP_WAIT_ADDRESS;
      }
      break;
    }
  }
}

```

## lwip.c – USER CODE 2

```

191 /* USER CODE BEGIN 2 */
192 #if LWIP_DHCP
193 /**
194  * @brief DHCP Process
195  * @param argument: network interface
196  * @param None
197  */
198 void DHCP_Thread(void *argument)
199 {
200   struct netif *netif = (struct netif *) argument;
201   ip_addr_t ipaddr;
202   ip_addr_t netmask;
203   ip_addr_t gw;
204   struct dhcp *dhcp;
205   uint8_t iptxt[20];
206
207   for (;;)
208   {
209     switch (DHCP_state)
210     {
211       case DHCP_START:
212       {
213         printf("State: Looking for DHCP server ...\n");
214         HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_RESET);
215         HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_RESET);
216         ip_addr_set_zero_ip4(&netif->ip_addr);
217         ip_addr_set_zero_ip4(&netif->netmask);
218         ip_addr_set_zero_ip4(&netif->gw);
219         dhcp_start(netif);
220         DHCP_state = DHCP_WAIT_ADDRESS;
221       }
222       break;
223       case DHCP_WAIT_ADDRESS:
224       {
225         if (dhcp_supplied_address(netif))
226         {
227           DHCP_state = DHCP_ADDRESS_ASSIGNED;
228           sprintf((char *)iptxt, "%s", ip4addr_ntoa(netif_ip4_addr(netif)));
229           printf("IP address assigned by a DHCP server: %s\n", iptxt);
230           HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_SET);
231           HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_RESET);
232         }
233         else
234         {
235           dhcp = (struct dhcp *)netif_get_client_data(netif, LWIP_NETIF_CLIENT_DATA_INDEX_DHCP);
236

```

# Iwip.c – USER CODE BEGIN 2

- Cont.

```
case DHCP_WAIT_ADDRESS:
```

```
{
  if (dhcp_supplied_address(netif))
  {
    DHCP_state = DHCP_ADDRESS_ASSIGNED;
    sprintf((char *)iptxt, "%s", ip4addr_ntoa(netif_ip4_addr(netif)));
    printf("IP address assigned by a DHCP server: %s\n", iptxt);
    HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_RESET);
  }
  else
  {
    dhcp = (struct dhcp *)netif_get_client_data(netif, LWIP_NETIF_CLIENT_DATA_INDEX_DHCP);
```

```
191 /* USER CODE BEGIN 2 */
192 #if LWIP_DHCP
193 /**
194  * @brief DHCP Process
195  * @param argument: network interface
196  * @param None
197  */
198 void DHCP_Thread(void *argument)
199 {
200     struct netif *netif = (struct netif *) argument;
201     ip_addr_t ipaddr;
202     ip_addr_t netmask;
203     ip_addr_t gw;
204     struct dhcp *dhcp;
205     uint8_t iptxt[20];
206
207     for (;;)
208     {
209         switch (DHCP_state)
210         {
211             case DHCP_START:
212                 {
213                     printf("State: Looking for DHCP server ...\n");
214                     HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_RESET);
215                     HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_RESET);
216                     ip_addr_set_zero_ip4(&netif->ip_addr);
217                     ip_addr_set_zero_ip4(&netif->netmask);
218                     ip_addr_set_zero_ip4(&netif->gw);
219                     dhcp_start(netif);
220                     DHCP_state = DHCP_WAIT_ADDRESS;
221                 }
222                 break;
223             case DHCP_WAIT_ADDRESS:
224                 {
225                     if (dhcp_supplied_address(netif))
226                     {
227                         DHCP_state = DHCP_ADDRESS_ASSIGNED;
228                         sprintf((char *)iptxt, "%s", ip4addr_ntoa(netif_ip4_addr(netif)));
229                         printf("IP address assigned by a DHCP server: %s\n", iptxt);
230                         HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_SET);
231                         HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_RESET);
232                     }
233                     else
234                     {
235                         dhcp = (struct dhcp *)netif_get_client_data(netif, LWIP_NETIF_CLIENT_DATA_INDEX_DHCP);
```

- Cont.

```

/* DHCP timeout */
if (dhcp->tries > MAX_DHCP_TRIES)
{
    DHCP_state = DHCP_TIMEOUT;
    /* Static address used */
    IP_ADDR4(&ipaddr, IP_ADDR0, IP_ADDR1, IP_ADDR2, IP_ADDR3);
    IP_ADDR4(&netmask, NETMASK_ADDR0, NETMASK_ADDR1, NETMASK_ADDR2,
NETMASK_ADDR3);
    IP_ADDR4(&gw, GW_ADDR0, GW_ADDR1, GW_ADDR2, GW_ADDR3);
    netif_set_addr(netif, ip_2_ip4(&ipaddr), ip_2_ip4(&netmask), ip_2_ip4(&gw));
    sprintf((char *)iptxt, "%s", ip4addr_ntoa(netif_ip4_addr(netif)));
    printf("DHCP Timeout !! \n");
    printf("Static IP address: %s\n", iptxt);
    HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_RESET);
}
}
break;
case DHCP_LINK_DOWN:
{
    DHCP_state = DHCP_OFF;
    HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_SET);
}
break;
default: break;
}
/* wait 500 ms */
osDelay(500);
}
}
#endif /* LWIP_DHCP */

```

## lwip.c – USER CODE 2

```

236
237 /* DHCP timeout */
238 if (dhcp->tries > MAX_DHCP_TRIES)
239 {
240     DHCP_state = DHCP_TIMEOUT;
241
242     /* Static address used */
243     IP_ADDR4(&ipaddr, IP_ADDR0, IP_ADDR1, IP_ADDR2, IP_ADDR3);
244     IP_ADDR4(&netmask, NETMASK_ADDR0, NETMASK_ADDR1, NETMASK_ADDR2, NETMASK_ADDR3);
245     IP_ADDR4(&gw, GW_ADDR0, GW_ADDR1, GW_ADDR2, GW_ADDR3);
246     netif_set_addr(netif, ip_2_ip4(&ipaddr), ip_2_ip4(&netmask), ip_2_ip4(&gw));
247     sprintf((char *)iptxt, "%s", ip4addr_ntoa(netif_ip4_addr(netif)));
248     printf("DHCP Timeout !! \n");
249     printf("Static IP address: %s\n", iptxt);
250     HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_SET);
251     HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_RESET);
252 }
253 }
254 }
255 break;
256 case DHCP_LINK_DOWN:
257 {
258     DHCP_state = DHCP_OFF;
259     HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_RESET);
260     HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_SET);
261 }
262 break;
263 default: break;
264 }
265 }
266 /* wait 500 ms */
267 osDelay(500);
268 }
269 }
270 #endif /* LWIP_DHCP */

```



- Next is the server function, where the assets will be accessed and served

```

void http_server_serve(int conn)
{
    int buflen = 1500;
    int ret;
    struct fs_file file;
    unsigned char recv_buffer[1500];

    /* Read in the request */
    ret = read(conn, recv_buffer, buflen);
    if(ret < 0) return;

    /* Check if request to get ST.gif */
    if (strncmp((char *)recv_buffer,"GET /STM32H7xx_files/ST.gif",27)==0)
    {
        fs_open(&file, "/STM32H7xx_files/ST.gif");
        write(conn, (const unsigned char*)(file.data), (size_t)file.len);
        fs_close(&file);
    }
    /* Check if request to get stm32.jpeg */
    else if (strncmp((char *)recv_buffer,"GET /STM32H7xx_files/stm32.jpg",30)==0)
    {
        fs_open(&file, "/STM32H7xx_files/stm32.jpg");
        write(conn, (const unsigned char*)(file.data), (size_t)file.len);
        fs_close(&file);
    }
}

```

## lwip.c – USER CODE 2

```

268 void http_server_serve(int conn)
269 {
270     int buflen = 1500;
271     int ret;
272     struct fs_file file;
273     unsigned char recv_buffer[1500];
274
275     /* Read in the request */
276     ret = read(conn, recv_buffer, buflen);
277     if(ret < 0) return;
278
279     /* Check if request to get ST.gif */
280     if (strncmp((char *)recv_buffer,"GET /STM32H7xx_files/ST.gif",27)==0)
281     {
282         fs_open(&file, "/STM32H7xx_files/ST.gif");
283         write(conn, (const unsigned char*)(file.data), (size_t)file.len);
284         fs_close(&file);
285     }
286     /* Check if request to get stm32.jpeg */
287     else if (strncmp((char *)recv_buffer,"GET /STM32H7xx_files/stm32.jpg",30)==0)
288     {
289         fs_open(&file, "/STM32H7xx_files/stm32.jpg");
290         write(conn, (const unsigned char*)(file.data), (size_t)file.len);
291         fs_close(&file);
292     }
293     /* Check if request to get ST logo.jpeg */
294     else if (strncmp((char *)recv_buffer,"GET /STM32H7xx_files/logo.jpg", 29) == 0)
295     {
296         fs_open(&file, "/STM32H7xx_files/logo.jpg");
297         write(conn, (const unsigned char*)(file.data), (size_t)file.len);
298         fs_close(&file);
299     }
300     else if(strncmp((char *)recv_buffer, "GET /STM32H7xxTASKS.html", 24) == 0)
301     {
302         /* Load dynamic page */
303         DynWebPage(conn);
304     }
305     else if((strncmp((char *)recv_buffer, "GET /STM32H7xx.html", 19) == 0)|| (strncmp((char *)recv_buffer, "GET /", 6) == 0))
306     {
307         /* Load STM32H7xxpage */
308         fs_open(&file, "/STM32H7xx.html");
309         write(conn, (const unsigned char*)(file.data), (size_t)file.len);
310         fs_close(&file);
311     }
312     else
313     {
314         /* Load 404 page */
315         fs_open(&file, "/404.html");
316         write(conn, (const unsigned char*)(file.data), (size_t)file.len);
317         fs_close(&file);
318     }
319     /* Close connection socket */
320     close(conn);
321 }

```

- Cont.

# Iwip.c – USER CODE BEGIN 2

```

/* Check if request to get ST logo.jpeg */
else if (strncmp((char *)recv_buffer,"GET /STM32H7xx_files/logo.jpg", 29) == 0)
{
    fs_open(&file, "/STM32H7xx_files/logo.jpg");
    write(conn, (const unsigned char*)(file.data), (size_t)file.len);
    fs_close(&file);
}
else if (strncmp((char *)recv_buffer, "GET /STM32H7xxTASKS.html", 24) == 0)
{
    /* Load dynamic page */
    DynWebPage(conn);
}
else if ((strncmp((char *)recv_buffer, "GET /STM32H7xx.html", 19) == 0) || (strncmp((char
*)recv_buffer, "GET /", 6) == 0))
{
    /* Load STM32H7xxpage */
    fs_open(&file, "/STM32H7xx.html");
    write(conn, (const unsigned char*)(file.data), (size_t)file.len);
    fs_close(&file);
}
else
{
    /* Load 404 page */
    fs_open(&file, "/404.html");
    write(conn, (const unsigned char*)(file.data), (size_t)file.len);
    fs_close(&file);
}
/* Close connection socket */
close(conn);
}

```

```

268 void http_server_serve(int conn)
269 {
270     int buflen = 1500;
271     int ret;
272     struct fs_file file;
273     unsigned char recv_buffer[1500];
274
275     /* Read in the request */
276     ret = read(conn, recv_buffer, buflen);
277     if (ret < 0) return;
278
279     /* Check if request to get ST gif */
280     if (strncmp((char *)recv_buffer, "GET /STM32H7xx_files/ST.gif", 27) == 0)
281     {
282         fs_open(&file, "/STM32H7xx_files/ST.gif");
283         write(conn, (const unsigned char*)(file.data), (size_t)file.len);
284         fs_close(&file);
285     }
286     /* Check if request to get stm32_img */
287     else if (strncmp((char *)recv_buffer, "GET /STM32H7xx_files/stm32_img", 30) == 0)
288     {
289         fs_open(&file, "/STM32H7xx_files/stm32_img");
290         write(conn, (const unsigned char*)(file.data), (size_t)file.len);
291         fs_close(&file);
292     }
293     /* Check if request to get ST logo.jpeg */
294     else if (strncmp((char *)recv_buffer, "GET /STM32H7xx_files/logo.jpg", 29) == 0)
295     {
296         fs_open(&file, "/STM32H7xx_files/logo.jpg");
297         write(conn, (const unsigned char*)(file.data), (size_t)file.len);
298         fs_close(&file);
299     }
300     else if (strncmp((char *)recv_buffer, "GET /STM32H7xxTASKS.html", 24) == 0)
301     {
302         /* Load dynamic page */
303         DynWebPage(conn);
304     }
305     else if ((strncmp((char *)recv_buffer, "GET /STM32H7xx.html", 19) == 0) || (strncmp((char *)recv_buffer, "GET /", 6) == 0))
306     {
307         /* Load STM32H7xxpage */
308         fs_open(&file, "/STM32H7xx.html");
309         write(conn, (const unsigned char*)(file.data), (size_t)file.len);
310         fs_close(&file);
311     }
312     else
313     {
314         /* Load 404 page */
315         fs_open(&file, "/404.html");
316         write(conn, (const unsigned char*)(file.data), (size_t)file.len);
317         fs_close(&file);
318     }
319     /* Close connection socket */
320     close(conn);
321 }

```

- The server socket thread will handle the TCP connection, binding to port 80 and listening to port 5

```
static void http_server_socket_thread(void *arg)
{
    int sock, newconn, size;
    struct sockaddr_in address, remotehost;

    /* create a TCP socket */
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        return;
    }
    /* bind to port 80 at any interface */
    address.sin_family = AF_INET;
    address.sin_port = htons(80);
    address.sin_addr.s_addr = INADDR_ANY;
    if (bind(sock, (struct sockaddr *)&address, sizeof (address)) < 0)
    {
        return;
    }
    /* listen for incoming connections (TCP listen backlog = 5) */
    listen(sock, 5);
    size = sizeof(remotehost);
    while (1)
    {
        newconn = accept(sock, (struct sockaddr *)&remotehost, (socklen_t *)&size);
        http_server_serve(newconn);
    }
}
```

## lwip.c – USER CODE 2

```
323 static void http_server_socket_thread(void *arg)
324 {
325     int sock, newconn, size;
326     struct sockaddr_in address, remotehost;
327
328
329     /* create a TCP socket */
330     if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
331     {
332         return;
333     }
334
335     /* bind to port 80 at any interface */
336     address.sin_family = AF_INET;
337     address.sin_port = htons(80);
338     address.sin_addr.s_addr = INADDR_ANY;
339
340     if (bind(sock, (struct sockaddr *)&address, sizeof (address)) < 0)
341     {
342         return;
343     }
344
345     /* listen for incoming connections (TCP listen backlog = 5) */
346     listen(sock, 5);
347
348     size = sizeof(remotehost);
349
350     while (1)
351     {
352         newconn = accept(sock, (struct sockaddr *)&remotehost, (socklen_t *)&size);
353         http_server_serve(newconn);
354     }
355 }
```

# lwip.c – USER CODE 2

```
void http_server_socket_init()
{
    sys_thread_new("HTTP", http_server_socket_thread, NULL,
DEFAULT_THREAD_STACKSIZE * 4, WEBSERVER_THREAD_PRIO);
}

void DynWebPage(int conn)
{
    portCHAR PAGE_BODY[512];
    portCHAR pagehits[10] = {0};

    memset(PAGE_BODY, 0,512);

    /* Update the hit count */
    nPageHits++;
    sprintf( pagehits, "%d", (int)nPageHits );
    strcat(PAGE_BODY, pagehits);
    strcat((char *) PAGE_BODY, "<pre><br>Name      State Priority Stack Num" );
    strcat((char *) PAGE_BODY, "<br>-----<br>");

    /* The list of tasks and their status */
    vTaskList((char *) (PAGE_BODY + strlen(PAGE_BODY)));
    strcat((char *) PAGE_BODY, "<br><br>-----");
    strcat((char *) PAGE_BODY, "<br>B : Blocked, R : Ready, D : Deleted, S :
Suspended<br>");

    /* Send the dynamically generated page */
    write(conn, PAGE_START, strlen((char*)PAGE_START));
    write(conn, PAGE_BODY, strlen(PAGE_BODY));
}
/* USER CODE END 2 */
```

- The final functions are responsible for the HTTP thread init and the dynamic webpage

```
357 void http_server_socket_init()
358 {
359     sys_thread_new("HTTP", http_server_socket_thread, NULL, DEFAULT_THREAD_STACKSIZE * 4, WEBSERVER_THREAD_PRIO);
360 }
361
362 void DynWebPage(int conn)
363 {
364     portCHAR PAGE_BODY[512];
365     portCHAR pagehits[10] = {0};
366
367     memset(PAGE_BODY, 0,512);
368
369     /* Update the hit count */
370     nPageHits++;
371     sprintf( pagehits, "%d", (int)nPageHits );
372     strcat(PAGE_BODY, pagehits);
373     strcat((char *) PAGE_BODY, "<pre><br>Name      State Priority Stack Num" );
374     strcat((char *) PAGE_BODY, "<br>-----<br>");
375
376     /* The list of tasks and their status */
377     vTaskList((char *) (PAGE_BODY + strlen(PAGE_BODY)));
378     strcat((char *) PAGE_BODY, "<br><br>-----");
379     strcat((char *) PAGE_BODY, "<br>B : Blocked, R : Ready, D : Deleted, S : Suspended<br>");
380
381     /* Send the dynamically generated page */
382     write(conn, PAGE_START, strlen((char*)PAGE_START));
383     write(conn, PAGE_BODY, strlen(PAGE_BODY));
384 }
```

# lwip.c – USER CODE BEGIN H7\_OS

- Initialise the DHCP thread in the LWIP\_Init function

```
#if LWIP_DHCP
/* Start DHCPClient */
DHCPHandle = osThreadNew(DHCP_Thread, &gnetif, &DHCPThread_attributes);
#endif
/* USER CODE END H7_OS_THREAD_NEW_CMSIS_RTOS_V2 */
```

```
409 /* Set the link callback function, this function is called on change of link status*/
410 netif_set_link_callback(&gnetif, ethernet_link_status_updated);
411
412 /* Create the Ethernet link handler thread */
413 /* USER CODE BEGIN H7_OS_THREAD_NEW_CMSIS_RTOS_V2 */
414 memset(&attributes, 0x0, sizeof(osThreadAttr_t));
415 attributes.name = "EthLink";
416 attributes.stack_size = INTERFACE_THREAD_STACK_SIZE;
417 attributes.priority = osPriorityBelowNormal;
418 osThreadNew(ethernet_link_thread, &gnetif, &attributes);
419 #if LWIP_DHCP
420 /* Start DHCPClient */
421 DHCPHandle = osThreadNew(DHCP_Thread, &gnetif, &DHCPThread_attributes);
422 #endif
423 /* USER CODE END H7_OS_THREAD_NEW_CMSIS_RTOS_V2 */
424
425 /* Start DHCP negotiation for a network interface (IPv4) */
426 dhcp_start(&gnetif);
427
428 /* USER CODE BEGIN 3 */
429
430 /* USER CODE END 3 */
431 }
432
```

# Iwip.c – USER CODE BEGIN 5

- Lastly, add the final DHCP state machine code on the link update function

```
/* USER CODE BEGIN 5 */
#if LWIP_DHCP
  /* Update DHCP state machine */
  DHCP_state = DHCP_START;
#endif /* LWIP_DHCP */
/* USER CODE END 5 */
}
else /* netif is down */
{
/* USER CODE BEGIN 6 */
#if LWIP_DHCP
  /* Update DHCP state machine */
  DHCP_state = DHCP_LINK_DOWN;
#endif /* LWIP_DHCP */
/* USER CODE END 6 */
}
```

```
444  /*
445  static void ethernet_link_status_updated(struct netif *netif)
446  {
447    if (netif_is_up(netif))
448    {
449      /* USER CODE BEGIN 5 */
450      #if LWIP_DHCP
451        /* Update DHCP state machine */
452        DHCP_state = DHCP_START;
453      #endif /* LWIP_DHCP */
454      /* USER CODE END 5 */
455    }
456    else /* netif is down */
457    {
458      /* USER CODE BEGIN 6 */
459      #if LWIP_DHCP
460        /* Update DHCP state machine */
461        DHCP_state = DHCP_LINK_DOWN;
462      #endif /* LWIP_DHCP */
463      /* USER CODE END 6 */
464    }
465  }
```



# ethernetif.c – USER CODE BEGIN 2

- Add semaphores for the packets and allocating the Rx section

```
/* USER CODE BEGIN 2 */
__attribute__((section(".Rx_PoolSection"))) extern u8_t
memp_memory_RX_POOL_base[];

osThreadId_t EthIfHandle;
const osSemaphoreAttr_t RxPktSemaphore_attributes = {
    .name = "RxPktSemaphore"
};
osThreadId_t EthIfThread; /* Handle of the interface thread */
const osThreadAttr_t EthIf_attributes = {
    .name = "EthIf",
    .priority = (osPriority_t) osPriorityRealtime,
    .stack_size = 6 * INTERFACE_THREAD_STACK_SIZE
};

const osSemaphoreAttr_t TxPktSemaphore_attributes = {
    .name = "TxPktSemaphore"
};
/* USER CODE END 2 */
```

```
121 /* USER CODE BEGIN 2 */
122 __attribute__((section(".Rx_PoolSection"))) extern u8_t memp_memory_RX_POOL_base[];
123
124 osThreadId_t EthIfHandle;
125 const osSemaphoreAttr_t RxPktSemaphore_attributes = {
126     .name = "RxPktSemaphore"
127 };
128 osThreadId_t EthIfThread; /* Handle of the interface thread */
129 const osThreadAttr_t EthIf_attributes = {
130     .name = "EthIf",
131     .priority = (osPriority_t) osPriorityRealtime,
132     .stack_size = 6 * INTERFACE_THREAD_STACK_SIZE
133 };
134
135 const osSemaphoreAttr_t TxPktSemaphore_attributes = {
136     .name = "TxPktSemaphore"
137 };
138 /* USER CODE END 2 */
```

# LINKER SCRIPT

- In the linker file add the descriptor sections on the RAM\_D2 memory

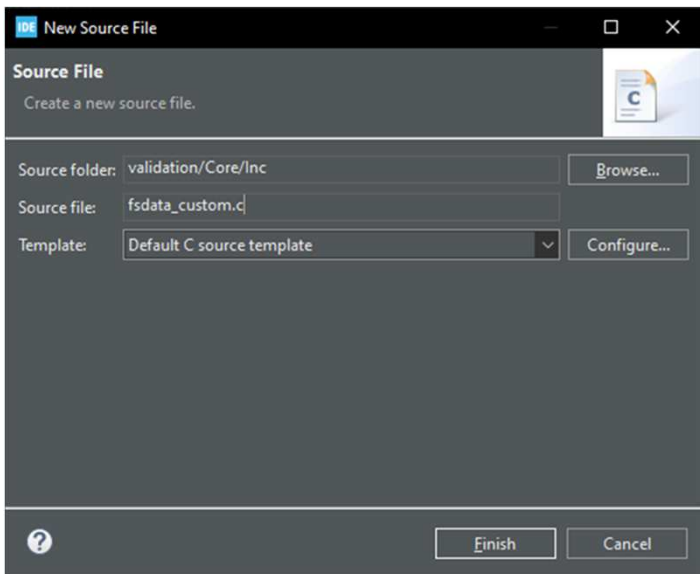
```
.lwip_sec (NOLOAD) : {  
  . = ABSOLUTE(0x30000000);  
  *(.RxDecripSection)  
  
  . = ABSOLUTE(0x30000080);  
  *(.TxDecripSection)  
  
  . = ABSOLUTE(0x30000100);  
  *(.Rx_PoolSection)  
} >RAM_D2 AT> FLASH
```

```
168  
169     .lwip_sec (NOLOAD) : {  
170     . = ABSOLUTE(0x30000000);  
171     *(.RxDecripSection)  
172  
173     . = ABSOLUTE(0x30000080);  
174     *(.TxDecripSection)  
175  
176     . = ABSOLUTE(0x30000100);  
177     *(.Rx_PoolSection)  
178     } >RAM_D2 AT> FLASH  
179
```



# fsdata\_custom.c

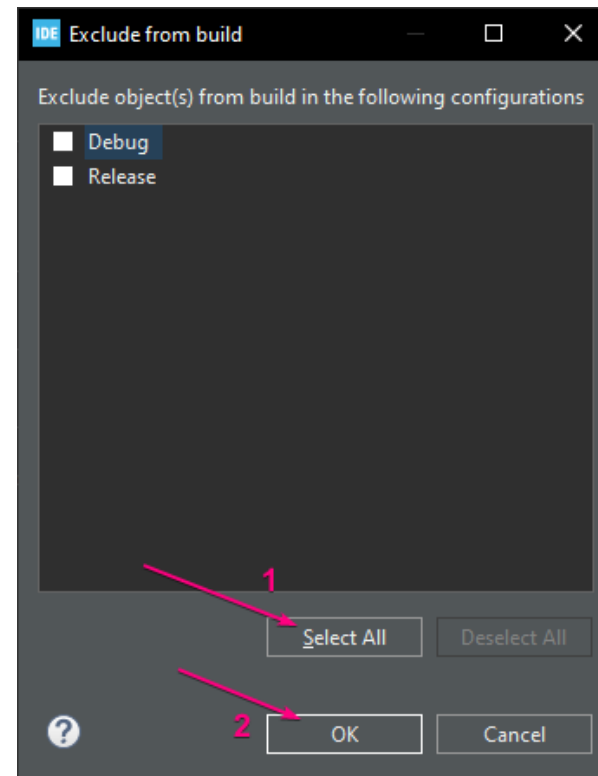
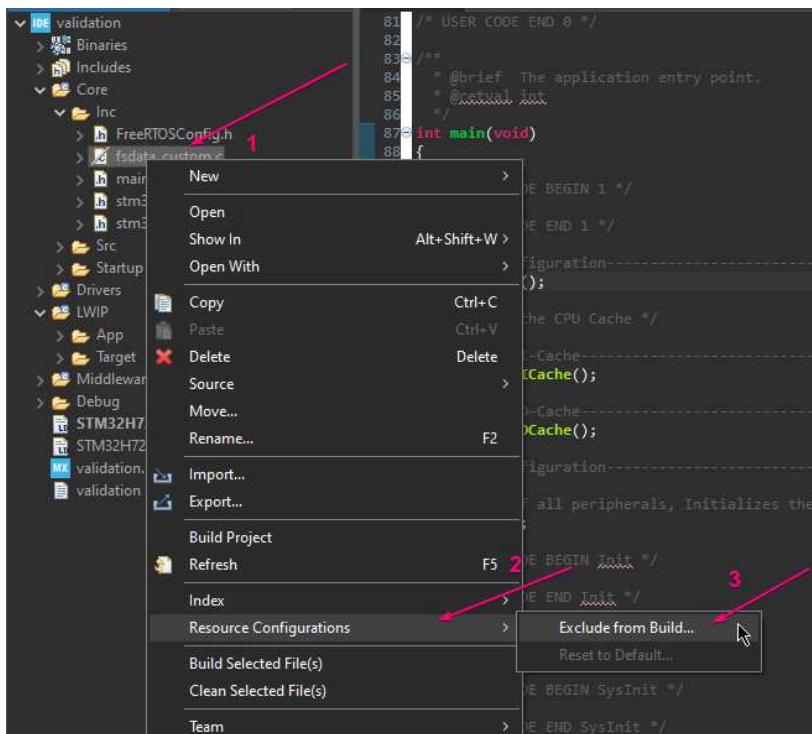
- A file named fsdata\_custom.c must be created inside the inc folder
- Copy the provided data from the txt file to this. These will be our webpages.



```
4233 const struct fsdata_file file_STM32H7xx_files_ST_gif[] = { {
4234 file_NULL,
4235 data_STM32H7xx_files_ST_gif,
4236 data_STM32H7xx_files_ST_gif + 24,
4237 sizeof(data_STM32H7xx_files_ST_gif) - 24,
4238 FS_FILE_FLAGS_HEADER_INCLUDED | FS_FILE_FLAGS_HEADER_PERSISTENT,
4239 }};
4240
4241 const struct fsdata_file file_STM32H7xx_files_logo_jpg[] = { {
4242 file_STM32H7xx_files_ST_gif,
4243 data_STM32H7xx_files_logo_jpg,
4244 data_STM32H7xx_files_logo_jpg + 28,
4245 sizeof(data_STM32H7xx_files_logo_jpg) - 28,
4246 FS_FILE_FLAGS_HEADER_INCLUDED | FS_FILE_FLAGS_HEADER_PERSISTENT,
4247 }};
4248
4249 const struct fsdata_file file_STM32H7xx_files_stm32_jpg[] = { {
4250 file_STM32H7xx_files_logo_jpg,
4251 data_STM32H7xx_files_stm32_jpg,
4252 data_STM32H7xx_files_stm32_jpg + 28,
4253 sizeof(data_STM32H7xx_files_stm32_jpg) - 28,
4254 FS_FILE_FLAGS_HEADER_INCLUDED | FS_FILE_FLAGS_HEADER_PERSISTENT,
4255 }};
4256
4257 const struct fsdata_file file_404_html[] = { {
4258 file_STM32H7xx_files_stm32_jpg,
4259 data_404_html,
4260 data_404_html + 12,
4261 sizeof(data_404_html) - 12,
4262 FS_FILE_FLAGS_HEADER_INCLUDED | FS_FILE_FLAGS_HEADER_PERSISTENT,
4263 }};
4264
4265 const struct fsdata_file file_STM32H7xx_html[] = { {
4266 file_404_html,
4267 data_STM32H7xx_html,
4268 data_STM32H7xx_html + 16,
4269 sizeof(data_STM32H7xx_html) - 16,
4270 FS_FILE_FLAGS_HEADER_INCLUDED | FS_FILE_FLAGS_HEADER_PERSISTENT,
4271 }};
```

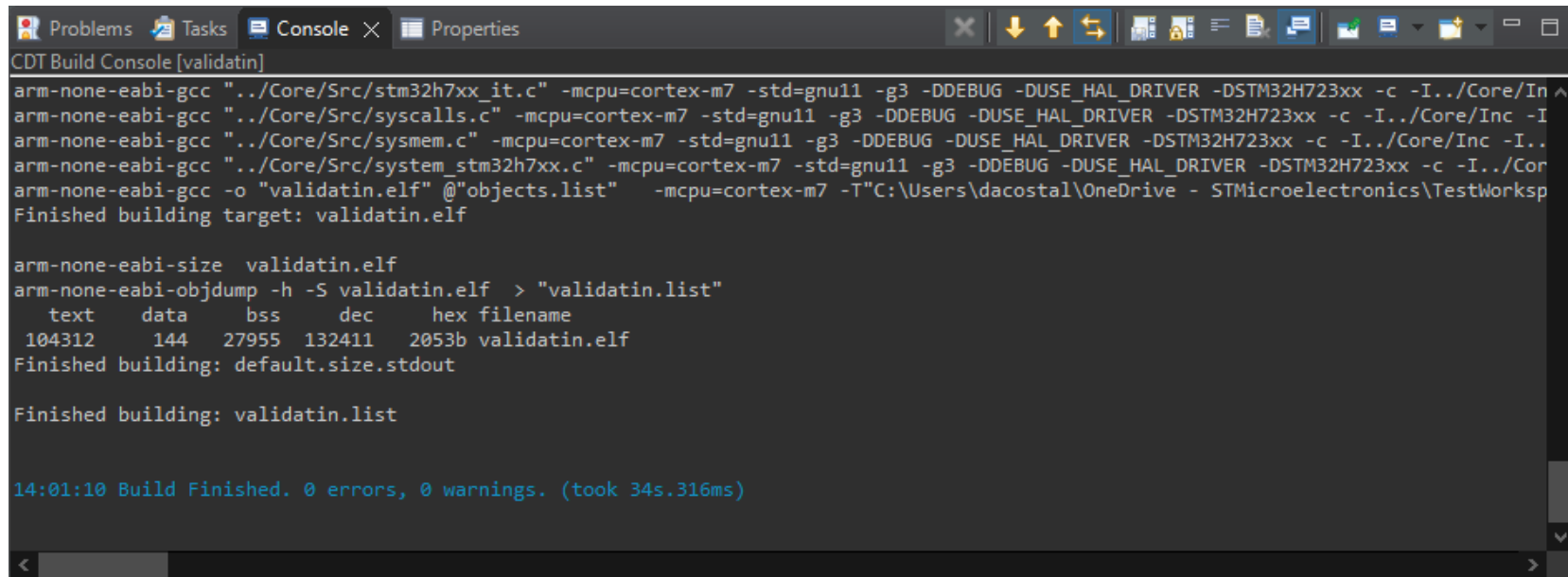
# fsdata\_custom.c

- The fsdata file must not be built with the project, only loaded. Left click it and follow the steps below



# The moment of truth

- Compile the project
  - We expect 0 errors and 0 warnings



```
CDT Build Console [validatin]
arm-none-eabi-gcc "../Core/Src/stm32h7xx_it.c" -mcpu=cortex-m7 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32H723xx -c -I../Core/In ^
arm-none-eabi-gcc "../Core/Src/syscalls.c" -mcpu=cortex-m7 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32H723xx -c -I../Core/Inc -I
arm-none-eabi-gcc "../Core/Src/systemem.c" -mcpu=cortex-m7 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32H723xx -c -I../Core/Inc -I..
arm-none-eabi-gcc "../Core/Src/system_stm32h7xx.c" -mcpu=cortex-m7 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32H723xx -c -I../Cor
arm-none-eabi-gcc -o "validatin.elf" @"objects.list" -mcpu=cortex-m7 -T"C:\Users\dacostal\OneDrive - STMicroelectronics\TestWorksp
Finished building target: validatin.elf

arm-none-eabi-size validatin.elf
arm-none-eabi-objdump -h -S validatin.elf > "validatin.list"
  text  data  bss  dec  hex filename
104312  144  27955  132411  2053b validatin.elf
Finished building: default.size.stdout

Finished building: validatin.list

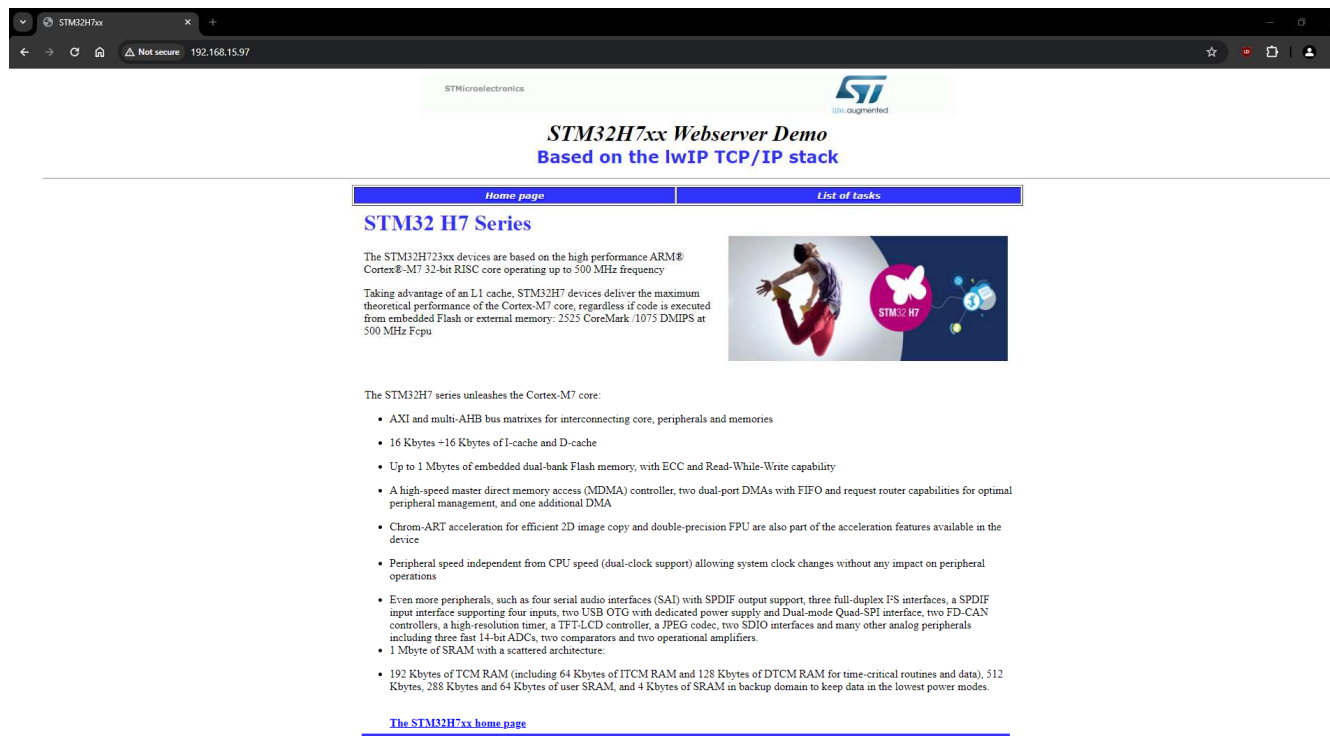
14:01:10 Build Finished. 0 errors, 0 warnings. (took 34s.316ms)
```

# How to test it?

- To test the web server application, follow these steps:
  - Build and program the code we've just made.
  - Open your preferred virtual com terminal software and connect to the micro's COM port
  - On the remote PC, open a web client (Mozilla Firefox. Google Chrome or Internet Explorer) and type the board IP address in a web browser.


# How to test it?

- The webserver home page is like this:



# How to test it?

- Clicking the 'list of tasks' button will take us to the dynamic page we created, containing our RTOS stats



The screenshot shows a web browser window with the address bar displaying "192.168.15.97/STM32H7xxTASKS.html". The page title is "STM32H7xx List of tasks and their status". Below the title, there are two navigation buttons: "Home page" and "List of tasks". The "List of tasks" button is highlighted. Below the buttons, the text "Number of page hits: 7" is displayed. A table follows, showing the status of various tasks. The table has five columns: Name, State, Priority, Stack, and Num. The tasks listed are HTTP, IDLE, tcpip\_thread, DHCP, EthLink, EthIf, and Tmr Svc. Below the table, a legend explains the state abbreviations: B : Blocked, R : Ready, D : Deleted, S : Suspended.

Name	State	Priority	Stack	Num
HTTP	X	32	239	8
IDLE	R	0	222	2
tcpip_thread	B	40	2	4
DHCP	B	16	93	7
EthLink	B	16	69	6
EthIf	B	48	1	5
Tmr Svc	B	2	471	3

-----  
B : Blocked, R : Ready, D : Deleted, S : Suspended