Library 52- Ethernet peripheral on STM32F4xx

BY TILZ0R · PUBLISHED FEBRUARY 28, 2015 · UPDATED MARCH 18, 2015

One of the greatest features on STM32F4xx for me was to get ethernet to work properly as server and client. I got it working pretty quickly and I was investigating how to make a library to be very useful and easy to use.

Ethernet library is built on LwIP TCP/IP stack version 1.4.1.

Library is pretty hard to “install” for first time, so I decided to provide you source files (on my Github) for Keil uVision and Coocox.

Examples for Keil uVision and Coocox are finally available on my Github.

Library

Features

- Supports RMII or MII interface with PHY and STM32F4 ETH interface
- Built on LwIP TCP/IP stack
- Support for TCP client and TCP server
    - 4 TCP client connections at a time
- Built-in support for opening files from SD card or any other source
- Support to disable server access to IP address
- DNS support for receive IP from domain
- DHCP support for receive IP in your network
    - Support for custom device name on router
- Support for callbacks with full control on your work
    - Callbacks for client (when connected, disconnected, data available, create headers)

        - Callbacks for server (new client connected to server, disconnected)
        - Callbacks for DNS (DNS has started, DNS has found IP)
        - Callbacks for DHCP (DHCP has IP for us, DHCP error)
        - Callbacks for link status (Link down, link up)
        - Callbacks for POST requests
    - Support for dynamic MAC address
    - Support for STATIC/DYNAMIC IP address
    - Support for monitoring how many bytes we sent/received over client/server mode
    - Enable/Disable server mode
    - User selectable PORT for server
    - Support for SSI tags and CGI handlers
        - SSI: You can display variables from MCU (RTC clock for example, LEDs status, etc)
        - CGI: You can control MCU from web (control leds)
    - Support for POST request
    - Support for TCP communication between 2 STM devices without router using crossover cable

Dependencies

- CMSIS
    - STM32F4xx
    - STM32F4xx RCC
    - STM32F4xx GPIO
    - STM32F4xx ETH (included in library)
    - STM32F4xx SYSCFG
- TM
    - DELAY

      TM STM32F4 Delay Library 45348 downloads 0.00 KB **Download**
    - GPIO

TM STM32F4 GPIO Library 80645 downloads 0.00 KB
Download

- defines.h

defines.h configuration example 158347 downloads 0.00 KB
Download

- attributes.h

attributes.h 26743 downloads 0.85 KB
Download

- LwIP TCP/IP stack (Included in library)

Pinout

Ethernet works with ETH peripheral. For this purpose, you can't use any STM32F4xx device (F401, F405, F411, F415 don't have ethernet). As of version 1.1, MII or RMII connection is supported. In table below is RMII pinout with 2 possible pinspacks. Look a little bit below to see table for MII connection.

	STM32F4xx		
ETHERNET PHY	**PINSPACK 1**	**PINSPACK 2**	**Description**
MDIO	PA2	PA2	Management Data I/O pin
MDC	PC1	PC1	Management Interface clock input
REF_CLK	PA1	PA1	50 MHz reference clock from PHY to STM3
CRS	PA7	PA7	Carrier Receive/Sense Data Valid
RX0	PC4	PC4	Receive data line 0
RX1	PC5	PC5	Receive data line 1
TX_EN	PB11	PG11	Enable transmitted
TX0	PB12	PG13	Transmit data line 0
TX1	PB13	PG14	Transmit data line 1

PA8	PA8	Possible bug, check below

Default pinout is the left one (PINSPACK1) but if you want for some reason right one, use define below:

```
/* defines.h file */
/* Enable PinsPack 2 for RMII ethernet */
#define ETHERNET_RMII_PINSPACK_2
```

Example works without problems on STM32F4-Discovery with pinspack1. To get it working on STM32F429-Discovery, you must “damage” your board. You must remove gyro and so on. Not nice to try it there. I have 2 boards F429, so I take apart one to get it working.

I believe, this is a bug, but if you can ensure me that is not, please report that to me:

- *PA8 HAVE to be declared as MCO alternate function, even if not used for ethernet, or ETHERNET will not work.*
- *So keep in mind that if you don’t use this pin for ethernet source clock you shouldn’t connect anything to this pin.*
    - *Some devices (like KSZ8081RNA can acceps 25MHz input clock and with internal PLL convert to 50MHz which is needed for RMII)*

MII table

ETHERNET PHY	STM32F4xx	Description
MDIO	PA2	Management Data I/O
MDC	PC1	Management Interface clock input
REF_CLK	PA1	25 MHz reference clock from PHY to STM32F4xx
CRS	PH2	Carrier Receive/Sense Data Valid
COL	PH3	Collision
PPS_OUT	PB5	
RXD0	PC4	Receive data 0

RXD1	PC5	Receive data 1
RXD2	PH6	Receive data 2
RXD3	PH7	Receive data 3
RX_ERR	PI10	Receive error
RX_DV	PA7	RX Data valid
RX_CLK	PC3	RX clock
TX_EN	PG11	Transmit enable
TXD0	PG13	Transmit data 0
TXD1	PG14	Transmit data 1
TXD2	PC2	Transmit data 2
TXD3	PB8	Transmit data 3
	PA8	Possible bug, check above

After you design your board for MII mode, you have to enable it with my lib. Open defines.h file and add define:

```
/* Enable MII connection mode */
#define ETHERNET_MII_MODE
```

ETHERNET PHY

Library was built in using DP83848 Ethernet PHY. I have 2 DP modules for testing on both discovery boards at the same time and works well on both.

You can get DP83848 module here.
PHY is connected to STM using RMII mode. Basically, every PHY should work with this configuration because they have standard registers locations inside. Of course, every PHY has something different but they should work from start.

According to the datasheet, I've made 3 supported PHY-s. To select your PHY, open defines.h file and make a define below:

```
/*
* 0: DP83848
* 1: LAN8720A
* 2: KSZ8081RNA
*/
/* If you want to use LAN8720A PHY, do this in defines.h file: */
#define ETHERNET_PHY    1
```

I also have some functions for ethernet PHYs, which can be used for set custom pinouts, which will overwrite default pinouts and function for setting custom PHY settings. In example below, LEDs configuration in DP83848 will be changed.

```
/**
* Called when ethernet peripheral is ready.
* In this function, user can do other stuff, depending on PHY which is used in project.
*
* Example:
*    - Set LED mode for DP83848 PHY, or something else
*
* Parameters:
*    - uint32_t PHYAddress:
*        Phy address when using ETH_ReadPHYRegister and ETH_WritePHYRegister functions
*
* With __weak parameter to prevent link errors if not defined by user
*
* No return
*/
extern __weak void TM_ETHERNETPHY_CustomOptions(uint32_t PHYAddress);

/**
* Called on ethernet GPIO initialization
* In this function, user can initialize custom pins, which will be used for communication with PHY.
* Don't implement this function in case you want default pinout
*
* With __weak parameter to prevent link errors if not defined by user
*
* Must return 1 in case user has made own pins, or 0 in case we want default pins
*/
extern __weak uint8_t TM_ETHERNET_InitPinsCallback(void);
```

```
/* PHY based callbacks */
void TM_ETHERNETPHY_CustomOptions(uint32_t PHYAddress) {
  /* Called when initialization for ethernet is done */
  /* User can here do custom initialization settings, based on custom PHY */

  /* Change LED mode on DP83848 for example */
  uint32_t Reg_Val;

  //Read register 0x19 on DP83848
  Reg_Val = ETH_ReadPHYRegister(PHYAddress, 0x19);

  //Clear both LED bits, SET led mode to 2, read DP83848 datasheet for more info
  Reg_Val &= ~(1 << 5 | 1 << 6);

  //Write new value to register
  ETH_WritePHYRegister(PHYAddress, 0x19, Reg_Val);
}
```

COMMON LIBRARY SETTINGS

In common I will set things which HAVE to be used, no matter which configuration you use. One of them is Initialize function, which is used to initialize LwIP TCP/IP stack and prepare Ethernet PHY in working state.

```
/**
 * Initialize ethernet device and prepare device to work.
 *
 * This function prepares LwIP stack to work and DP83848 PHY,
 * but does not enable server functionality. You have separate
 * function which enables server functionality in case you need it.
 *
 * Parameters:
 *   - uint8_t* mac_addr:
 *       Pointer to 6 bytes long array, if you want to select custom
 *       MAC address, instead of one in in defines.
 *       Useful if you have more than one device in one network
 *       and you want to separate them by custom MAC (with their unique ID number)
 *       Set to NULL for default value
 *   - uint8_t* ip_addr:
 *       Pointer to 4 bytes long array, if you want to select custom
 *       IP address in case STATIC is used. This will also be used,
 *       if DHCP can't get IP address
```

```
*       Set to NULL for default value
*    - uint8_t* gateway:
*       Pointer to 4 bytes long array, if you want to select custom    gateway.
*       Set to NULL for default value
*    - uint8_t* netmask:
*       Pointer to 4 bytes long array, if you want to select custom netmask
*       Set to NULL for default value
*
* Member of TM_ETHERNET_Result_t is returned
*/
extern TM_ETHERNET_Result_t TM_ETHERNET_Init(uint8_t* mac_addr, uint8_t* ip_addr, uint8_t* gateway, uint8_t*
netmask);
```

Function will return something from **result** enumeration. It's structure is below:

```
/**
* Result enumeration used in ethernet library.
*
* Parameters:
*    - TM_ETHERNET_Result_Ok:
*       Everything is OK
*    - TM_ETHERNET_Result_Error:
*       An error occured
*    - TM_ETHERNET_Result_IPIsNotSetYet:
*       We don't have set IP
*    - TM_ETHERNET_Result_LinkIsDown:
*       Link is down
*    - TM_ETHERNET_Result_NeedHardReset:
*       We need hardware reset
*    - TM_ETHERNET_Result_LibraryNotInitialized:
*       Library is not initialized
*/
typedef enum {
  TM_ETHERNET_Result_Ok = 0,
  TM_ETHERNET_Result_Error,
  TM_ETHERNET_Result_IPIsNotSetYet,
  TM_ETHERNET_Result_LinkIsDown,
  TM_ETHERNET_Result_NeedHardReset,
  TM_ETHERNET_Result_LibraryNotInitialized
} TM_ETHERNET_Result_t;
```

To be able to get ethernet properly in working state, you also have several functions which HAVE TO be called.

First function is **Ethernet update** which is used to update LwIP stack, and second is **Time update** which is used to update "local time" for LwIP stack. They are below:

```
/**
 * Update ethernet LwIP stack
 *
 * This function should be called periodically, as fast as possible.
 *
 * Member of TM_ETHERNET_Result_t is returned
 */
extern TM_ETHERNET_Result_t TM_ETHERNET_Update(void);

/**
 * This function should be called in specific time period for LwIP stack
 *
 * Parameters:
 *  - uint16_t millis:
 *      Number of milliseconds, that will be added to LwIP stack.
 *      This value should be the same as period for your interrupt (Systick) timer is
 *
 * Member of TM_ETHERNET_Result_t is returned
 */
extern TM_ETHERNET_Result_t TM_ETHERNET_TimeUpdate(uint16_t millis);
```

Both function must be called periodically. Time update must be called in specific time. According to periodic time you call it, you have parameter for millis. If you call this function every 10ms, then pass into function parameter “10” which will increase local time for 10ms.

Update function should be called as fast as possible. It’s not necessary to call it at specific time intervals.

You might also use other common functions, which are described below:

```
/**
 * This function can be used to test if you are ready to work as server or client.
 *
 * Returns:
 *  - TM_ETHERNET_Result_Ok: In case link is up and ip is set (static or DHCP)
 *  - TM_ETHERNET_Result_Error: On other cases
 */
extern TM_ETHERNET_Result_t TM_ETHERNET_TestReady(void);

/**
```

```
11 * Check if device has static IP
12 *
13 * Returns 1 if it is static or 0 if not (DHCP)
14 */
15#define TM_ETHERNET_IsIPStatic()          (TM_ETHERNET.staticip)
16
17/**
18 * Check if device has 100Mbit network connection
19 *
20 * Returns 1 if it has, or 0 if not
21 */
22#define TM_ETHERNET_Is100M()              (TM_ETHERNET.speed_100m)
23
24/**
25 * Check if device is in full duplex mode
26 *
27 * Returns 1 if it is or 0 if not
28 */
29#define TM_ETHERNET_IsFullDuplex()          (TM_ETHERNET.full_duplex)
30
31/**
32 * Get local IP address
33 *
34 * Parameters:
35 *     - uint8_t x:
36 *         IP section, 0 to 3 are allowed. 0 is MSB and 3 is LSB
37 *
38 * IP address is returned
39 */
40#define TM_ETHERNET_GetLocalIP(x)          (((x) >= 0 && (x) < 4) ? TM_ETHERNET.ip_addr[(x)] : 0)
41
42/**
43 * Get MAC address
44 *
45 * Parameters:
46 *     - uint8_t x:
47 *         MAC section, 0 to 5 are allowed. 0 is MSB and 5 is LSB
48 *
49 * MAC address is returned
50 */
51#define TM_ETHERNET_GetMACAddr(x)          (((x) >= 0 && (x) < 6) ? TM_ETHERNET.mac_addr[(x)] : 0)
52
53/**
54 * Get gateway address
55 *
56 * Parameters:
57 *     - uint8_t x:
58 *         Gateway section, 0 to 3 are allowed. 0 is MSB and 3 is LSB
59 *
60 * Gateway is returned
61 */
62#define TM_ETHERNET_GetGateway(x)          (((x) >= 0 && (x) < 4) ? TM_ETHERNET.gateway[(x)] : 0)
63
64/**
65 * Get netmask address
66 *
67 * Parameters:
```

```
*     - uint8_t x:
*         Netmask section, 0 to 4 are allowed. 0 is MSB and 3 is LSB
*
* Net mask is returned
*/
#define TM_ETHERNET_GetNetmask(x)          (((x) >= 0 && (x) < 4) ? TM_ETHERNET.netmask[(x)] : 0)
```

DEFAULT NETWORK SETTINGS

Each device in your network has own MAC address and IP address, default gateway and netmask address.

My library allows you to set “dynamic” or “static” MAC address and IP.

Dynamic MAC means, that you apply MAC address when you initialize library (useful in case you have more than just one device in one network and you need different MAC addresses, you can use unique ID inside STM32F4). In case of STATIC MAC, you can set it using defines which are in my lib. In case you want to overwrite it, you can do this (defines.h file):

```
/* MAC ADDRESS: MAC_ADDR0:MAC_ADDR1:MAC_ADDR2:MAC_ADDR3:MAC_ADDR4:MAC_ADDR5 */
/* In case you want to use custom MAC, use parameter in init function */
#ifndef MAC_ADDR0
#define MAC_ADDR0                    0x06
#define MAC_ADDR1                    0x05
#define MAC_ADDR2                    0x04
#define MAC_ADDR3                    0x03
#define MAC_ADDR4                    0x02
#define MAC_ADDR5                    0x01
#endif
```

So, the same thing you can apply for other 3 settings too. All settings are below:

```
/* Static IP ADDRESS: IP_ADDR0.IP_ADDR1.IP_ADDR2.IP_ADDR3 */
/* Used in case DHCP is not used or response failed */
/* In case you want to use custom IP, use parameter in init function */
#ifndef IP_ADDR0
```

```
#define IP_ADDR0		192
#define IP_ADDR1		168
#define IP_ADDR2		0
#define IP_ADDR3		120
#endif

/* NETMASK */
/* In case you want to use custom netmask, use parameter in init function */
#ifndef NETMASK_ADDR0
#define NETMASK_ADDR0		255
#define NETMASK_ADDR1		255
#define NETMASK_ADDR2		255
#define NETMASK_ADDR3		0
#endif

/* Gateway Address */
/* In case you want to use custom gateway, use parameter in init function */
#ifndef GW_ADDR0
#define GW_ADDR0		192
#define GW_ADDR1		168
#define GW_ADDR2		0
#define GW_ADDR3		1
#endif
```

DHCP

DHCP is a great protocol, which can be used to assign device IP from your router. This basically means, that you (STM device) sends packet to router with “I want IP address” and then router sends you IP address and question “Is this IP OK?”. If it is OK, STM returns “Ok” and IP is assigned.

By default, DHCP assignment is disabled in my library, so IP which is set on defines (chapter above) or passed on initialization function is used. If you want to activate DHCP features, open defines.h file and add line below:

```
/* Enable DHCP IP assignment */
#define ETHERNET_USE_DHCP
```

One feature, you can use with DHCP, is that when you request a IP with DHCP, then you are displayed in control panel of your router. You can see device name and it’s IP and MAC address.

My library allows you, that you choose device name, which will be displayed in your router DHCP client list:

```
/* Device name which will be seen in router when requesting IP via DHCP */
#define ETHERNET_HOSTNAME_DHCP     "F4-Discovery"
```

DHCP client list on TP-Link router

DHCP is also a part of my callback functions.

If DHCP is enabled, when it will start with IP assignment, callback will be called.

```
/**
 * DHCP start callback
 *
 * This function is called when DHCP starts with IP address assigning for device.
 *
 * With __weak parameter to prevent link errors if not defined by user
 *
 * No return
 */
extern __weak void TM_ETHERNET_DHCPStartCallback(void);
```

DHCP makes 4 tries to get IP. If it fails, or not, a IP is set callback is called, which can be defined by user.

```
/**
 * IP is set callback
 * This function is called when IP is assigned to device.
 *
 * It is called in case DHCP is used or static ip is used.
 * It can be used to store IP address, if you use DHCP dynamic IP address.
 *
```

```
* Parameters:
*    - uint8_t ip_addr1:
*    - uint8_t ip_addr2:
*    - uint8_t ip_addr3:
*    - uint8_t ip_addr4:
*        IP addresses. ip_addr1 is MSB, ip_addr4 is LSB
*    - uint8_t dhcp:
*        This is set to 1 in case DHCP has been used for IP assign or 0 if static is used in case of user select or error
*
* With __weak parameter to prevent link errors if not defined by user
*
* No return
*/
extern __weak void TM_ETHERNET_IPIsSetCallback(uint8_t ip_addr1, uint8_t ip_addr2, uint8_t ip_addr3, uint8_t ip_addr4, uint8_t dhcp);
```

DNS

Another great feature in my lib is DNS. **D**omain **N**ame **S**erver allows you to get IP address from given doman name. For example, if you want to get IP address for “stm32f4-discovery.net” domain, you call DNS function and packet will be sent to DNS servers, with IP response (84.255.255.84).
To start DNS request, call my function:

```
/**
* DNS Get host by name
*
* This function can be used to get IP address by passing it's domain name to it.
*
* Different DNS callback functions will be called, depending on what will happen with DNS.
*
* Parameters:
*    - char* host_name:
*        Domain name for which you need IP address.
*
* Member of TM_ETHERNET_Result_t is returned
*/
extern TM_ETHERNET_Result_t TM_ETHERNETDNS_GetHostByName(char* host_name);
```

Function uses 2 callbacks in case DNS status. One callback is called when DNS found IP address, second is to told user that we have error:

- IP address is not found
- Invalid data passed to DNS function

```
/**
 * DNS Found callback
 *
 * This function will be called when DNS server has valid IP address for your domain request.
 * It can be useful to make first connection in this function when you know your IP address where you must connect to.
 *
 * Parameters:
 *    - char* host_name:
 *        The same host name as you use for DNS requesting, can't be larger than ETHERNET_MAX_CONNECTION_NAME define
 *    - uint8_t ip_addr1:
 *        IP address 1st byte, MSB
 *    - uint8_t ip_addr2:
 *        IP address 2nd byte
 *    - uint8_t ip_addr3:
 *        IP address 3rd byte
 *    - uint8_t ip_addr4:
 *        IP address 4th byte, LSB
 *
 * With __weak parameter to prevent link errors if not defined by user
 *
 * No return
 */
extern __weak void TM_ETHERNETDNS_FoundCallback(char* host_name, uint8_t ip_addr1, uint8_t ip_addr2, uint8_t ip_addr3, uint8_t ip_addr4);

/**
 * DNS error callback
 *
 * This function will be called when DNS error occured.
 * Possible errors are:
 *    - Requested domain does not exists,
 *    - You have bad parameters in DNS function call
 *
 * Parameters:
 *  - char* host_name:
 *     Host name where an error occured
 *
 * No return
 */
extern __weak void TM_ETHERNETDNS_ErrorCallback(char* host_name);
```

CLIENT

Client mode is one of 2 main features of ethernet. With client mode, you can request and receive data from another server, for example, you can make

“data logger” which will make GET request method to another server where you want to collect and store your data.

So when you are ready (IP is set) you can start with connection to another server at desired IP and port. **Library allows you to make 4 connections at the same time**. So you can call Connect function 4 times before any connection is closed.
To connect, call function below:

```
/**
* Make TCP CLIENT Connection to external server.
*
* After this call, different client callbacks will be called depending on what will happen.
*
* Parameters:
*    - char* conn_name:
*       Connection name. This value will be then passed to all client based handlers
*    - uint8_t ip1:
*    - uint8_t ip2:
*    - uint8_t ip3:
*    - uint8_t ip4:
*       IP addresses. ip1 is MSB, ip4 is LSB
*    - uint16_t port:
*       Port to which we want connect to server
*    - void* user_parameters:
*       Pointer to user data to be passed to this function.
*       This data will be used in all client based callbacks inside TM_TCPCLIENT_t pointer variable
*       Use NULL in case you don't want to pass any data.
*
* Member of TM_ETHERNET_Result_t is returned
*/
extern TM_ETHERNET_Result_t TM_ETHERNETCLIENT_Connect(char* conn_name, uint8_t ip1, uint8_t ip2, uint8_t ip3, uint8_t ip4, uint16_t port, void* user_parameters);
```

Function will start a TCP connection to specific IP address and PORT. Remember, this function must return TM_ETHERNET_Result_Ok which means that function succedded. In case it returns anything else, connect won’t work.

Another thing here are client callbacks. Because connecting to some server may take a while, but STM is too fast for waiting callbacks are used here. I’ve

made several callbacks. Here are just defines, how to use it you will see on examples.

```
/**
* Create headers callback
* This function is called when connection is prepared to send some headers data to server which is waiting for
response.
*
* If function is not implemented, then default headers are sent.
*
* Default headers:
*  GET / HTTP/1.1\r\n
*  Host: TM_ETHERNETClient\r\n
*  Connection: close\r\n
*  \r\n
*
* Parameters for function:
*     - TM_TCPCLIENT_t* connection:
*         Pointer to TM_TCPCLIENT_t which contains current connection specifications.
*  - char* buffer:
*      Char memory buffer where to store your header data
*  - uint16_t buffer_length:
*      Maximal memory length for your header data
*
* With __weak parameter to prevent link errors if not defined by user
*
* Function must return number of characters written in buffer.
* If you return 0 (zero) then connection will be closed.
* Buffer length can't be larger than ETHERNET_MAX_HEADER_SIZE define
*/
extern __weak uint16_t TM_ETHERNETCLIENT_CreateHeadersCallback(TM_TCPCLIENT_t* connection, char* buffer,
uint16_t buffer_length);

/**
* Receive data callback
*
* This function is called when we have data available to be received.
* It might happen that it will be called multiple times for one request, depends on server's response and length of
data.
*
* Parameters:
*     - TM_TCPCLIENT_t* connection:
*         Pointer to TM_TCPCLIENT_t which contanins current connection's specifications.
*     - uint8_t* buffer:
*         Memory buffer
*     - uint16_t buffer_length:
*         Number of characters in buffer
*    - uint16_t total_length
*        Number of unread data. If this value is the same as buffer_length, then no remaining data is available in packet
*
```

```
48 * With __weak parameter to prevent link errors if not defined by user
49 *
50 * No return
51 */
52 extern __weak void TM_ETHERNETCLIENT_ReceiveDataCallback(TM_TCPCLIENT_t* connection, uint8_t* buffer,
53 uint16_t buffer_length, uint16_t total_length);
54
55 /**
56 * Connected callback
57 *
58 * This function is called when device is connected to specific server.
59 *
60 * Parameters:
61 *    - TM_TCPCLIENT_t* connection:
62 *        Pointer to TM_TCPCLIENT_t which contains current connection's specifications
63 *
64 * With __weak parameter to prevent link errors if not defined by user
65 *
66 * No return
67 */
68 extern __weak void TM_ETHERNETCLIENT_ConnectedCallback(TM_TCPCLIENT_t* connection);
69
70 /**
71 * Connection closed callback
72 *
73 * This function is called when connection is closed and you are ready to make a new one.
74 * You can also detect, if connection was closed after success or error.
75 *
76 * Parameters:
77 *    - TM_TCPCLIENT_t* connection:
78 *        Pointer to TM_TCPCLIENT_t which contanins current connection specifications
79 *    - uint8_t success:
80 *        It is set to 1 if connection was closed after successfull transmission
81 *        or 0 if because of failure
82 *
83 * With __weak parameter to prevent link errors if not defined by user
84 *
85 * No return
86 */
87 extern __weak void TM_ETHERNETCLIENT_ConnectionClosedCallback(TM_TCPCLIENT_t* connection, uint8_t
88 success);
89
90 /**
91 * Error callback
92 *
93 * This function is called when an error occured.
94 * It will be called in case when IP address is not available
95 *
96 * Parameters:
97 *    - TM_TCPCLIENT_t* connection:
98 *        Pointer to TM_TCPCLIENT_t which contanins current connection specifications
99 *
100 * With __weak parameter to prevent link errors if not defined by user
101 *
102 * No return
103 */
104 extern __weak void TM_ETHERNETCLIENT_ErrorCallback(TM_TCPCLIENT_t* connection);
```

```

/**
* Connection started callback
*
* This function will be called when connection to server has started
*
* Parameters:
*    - TM_TCPCLIENT_t* connection:
*       Pointer to TM_TCPCLIENT_t which contanins current connection specifications
*
* With __weak parameter to prevent link errors if not defined by user
*
* No return
*/
extern __weak void TM_ETHERNETCLIENT_ConnectionStartedCallback(TM_TCPCLIENT_t* connection);
```

Every client callback, has first parameter **TM_TCPCLIENT_t* connection**. This is special structure, which is used for client to easily work with multiple connections at a time. It's structure is below:

```
/**
* Client structure, passed as first parameter in all client based callback functions
*
* Parameters:
*    - char name[ETHERNET_MAX_CONNECTION_NAME]:
*       Connection name, we choose it when we call TM_ETHERNETCLIENT_Connect function
*    - uint8_t active:
*       Connection is active
*    - uint8_t ip_addr[4]:
*       IP address for our external connection
*    - uint16_t port:
*       Port for our external connection
*    - uint8_t headers_done:
*       This is user defined option.
*       When you connect to some website and receive data back, you will also get HTTP headers and your data.
*       When receive data handler will be first called, this parameter will be set to 0.
*       If you detect when headers are done ("\r\n\r\n" string) then you can set this parameter to 1,
*       and it will stay 1 until connection is closed. This can be used, if you don't want to print headers to user.
*    - uint8_t* active_connections_count:
*       Pointer to number of active connections this time.
*    - void* user_parameters:
*       Pointer to user parameters for connection which are passed on "Connect" function
*       This can be used to pass special data for your connection, which you can then easily be used in headers callback
*       to format proper request string.
*/
typedef struct {
  char name[ETHERNET_MAX_CONNECTION_NAME];
  uint8_t active;
  uint8_t ip_addr[4];
  uint16_t port;
  uint8_t headers_done;
```

```
  uint8_t* active_connections_count;
  void* user_parameters;
  /* Private use */
  struct tcp_pcb* pcb;
  client_states state;
  struct pbuf *p_tx;
} TM_TCPCLIENT_t;
```

You might want to know some time how many data you sent/receive as a client over ethernet and how many connections you've made as a client. I've made some macro functions, which can be used in your project:

```
/**
 * Get total number of sent bytes when you are client
 *
 * Returns number of transmitted bytes as a client
 */
#define TM_ETHERNETCLIENT_GetTXBytes()                (TM_ETHERNET.Client_TX_Bytes)

/**
 * Get total number of received bytes when you are client
 *
 * Returns number of received bytes as a client
 */
#define TM_ETHERNETCLIENT_GetRXBytes()                (TM_ETHERNET.Client_RX_Bytes)

/**
 * Get number of all connections done as a client
 *
 * Returns number of all connections done as a client
 */
#define TM_ETHERNETCLIENT_GetConnectionsCount()            (TM_ETHERNET.ClientConnections)

/**
 * Get number of all successfull connections done as a client
 *
 * Returns number of all successfull connections done as a client
 */
#define TM_ETHERNETCLIENT_GetSuccessfullConnectionsCount()    (TM_ETHERNET.ClientSuccessfullConnections)
```

SERVER

Server mode is a little bit difficult for explain it's configuration, but I will try to be clear.

First, when you initialize library, server is disabled. Because, you don't want that someone just come to your device if you don't want. if you want to use server mode, then you have to enable it and tell on which PORT you will use your server.

```
/**
* Enable and prepare LwIP stack for server functionality.
* Before you can use this function, you have to initialize LwIP stack
* with TM_ETHERNET_Init() function.
*
* Parameters:
*   - uint16_t server_port:
*       Select your port on which you want to operate.
*       Set this variable to 0 and last used PORT will be selected
*       If you call this function multiple times, only first time will work this parameter
*
* Member of TM_ETHERNET_Result_t is returned
*/
extern TM_ETHERNET_Result_t TM_ETHERNETSERVER_Enable(uint16_t server_port);

/**
* Disable and stop LwIP stack from server functionality.
*
* Member of TM_ETHERNET_Result_t is returned
*/
extern TM_ETHERNET_Result_t TM_ETHERNETSERVER_Disable(void);
```

Enable function is a little bit weird I would say. So only first time parameter will work. It means, that if you call this function multiple times, PORT will be used the one, which was used on first function call. Future calls don't have affect of this parameter.

Ok, our server is enabled, and is accessible on our local IP. You can get local IP address using **TM_ETHERNET_GetLocalIP** function. But, it has no effect, if we don't show anything to user.
In file **fsdata.c** are 3 files, which are displayed to user in case you don't make your own. They are just for demonstration and I suggest you to remove them, or at least short data to minimum to free memory in flash.
The useful feature become when you are able to open custom files (from SD card or USB flash). This library allows you this also. It is designed to allow 10

opened files at a time by default. If you want to change this, open defines.h file and change:

```
/* Maximal number of opened files at a time as server */
#define ETHERNET_MAX_OPEN_FILES          10
```

```
/**
 * Called when user based file should be open when new client is connected
 * If you want to use server on your SDcard or USB flash or whatever,
 * here you can open file and do the stuff.
 *
 * Note: You must set file->len parameter to your file size number!
 *
 * If You don't open file (return 0) then default file in flash will be used,
 * that will say "Hello world from default server file", stored in fsdata.c.
 *
 * Only one file at a time will be opened, so if you need to mount card,
 * you can mount it in this function also.
 *
 * Parameters:
 *  - struct fs_file *file:
 *     Pointer to fs_file struct
 *  - const char *name:
 *     File name that should be opened
 *
 * With __weak parameter to prevent link errors if not defined by user
 *
 * Return 0 in case you can't/don't want to open file, or 1 if file is successfully opened
 */
extern __weak int TM_ETHERNETSERVER_OpenFileCallback(struct fs_file* file, const char *name);

/**
 * Called when reading file is finished and file can be closed.
 *
 * If you need to unmount your SD card, you can do it here.
 *
 * Parameters:
 *    - struct fs_file *file:
 *       Pointer to fs_file struct
 *
 * With __weak parameter to prevent link errors if not defined by user
 *
 * No return
 */
extern __weak void TM_ETHERNETSERVER_CloseFileCallback(struct fs_file* file);
```

```

/**
 * Called when file should be read.
 * This function may be called more than just one time, depends on file length.
 *
 * Parameters:
 *    - struct fs_file *file:
 *        Pointer to fs_file struct
 *    - char *buffer:
 *        Pointer to buffer, where you should save your data
 *    - int count:
 *        Number of data that should be read.
 *        You can read less than this number. This is max number you can read this time.
 *
 * With __weak parameter to prevent link errors if not defined by user
 *
 * Returns number of read data, or -1 (EOF) if file has end.
 */
extern __weak int TM_ETHERNETSERVER_ReadFileCallback(struct fs_file* file, char* buffer, int count);
```

In the examples below, you will see how to implement server with FatFs library from Chan. Which file should be opened will LwIP tell to you, you just have to make sure that you open file (if existing) that is in parameter.

All 3 functions have special file pointer parameter. It's structure is below:

```
struct fs_file {
  const char *data;
  int len;
  int index;
  void* pextension;
  uint8_t id;
  uint16_t* opened_files_count;
  u8_t http_header_included;
  u8_t is_custom_file;
  char file_name[ETHERNET_SERVER_MAX_CUSTOM_FILENAME];
};
```

Like client, server has also some other callbacks to work with connected clients. One of them is to know, when client is connected or disconnected. It allows you to block access to client if you detect bad remote IP address. Callbacks below:

```
/**
 * Called when new client is connected to server
 * This function can be used, to block IP address.
 * IP address, from where user access to your server is stored in *pcb variable
 *
 * remote_ip = pcb->remote_ip.addr
 *
 * You can check here for your blocked IP addresses
 *
 * Parameters:
 *    - struct tcp_pcb *pcb:
 *       Pointer to active LwIP PCB structure
 *
 * With __weak parameter to prevent link errors if not defined by user
 *
 * This function must return 1 if you want to allow connection,
 * or 0 if you want to deny access to your web.
 */
extern __weak uint8_t TM_ETHERNETSERVER_ClientConnectedCallback(struct tcp_pcb *pcb);

/**
 * Called when new client is diconnected from server (Or connection closed)
 *
 * With __weak parameter to prevent link errors if not defined by user
 *
 * No return
 */
extern __weak void TM_ETHERNETSERVER_ClientDisconnectedCallback(void);
```

Other statistical functions which correspond to server are below:

```
/**
 * Check if server mode is enabled.
 *
 * Returns 1 if active, or 0 if not
 */
#define TM_ETHERNETSERVER_Enabled()             (TM_ETHERNET.server_active)

/**
 * Get port number on which server is active
 *
 * Returns port number on which server is active
 */
#define TM_ETHERNETSERVER_GetPortNumber()       (TM_ETHERNET.server_port)

```

```
/**
 * Get number of all sent bytes from server to client
 *
 * Returns number of sent bytes from server to client
 */
#define TM_ETHERNETSERVER_GetTXBytes()          (TM_ETHERNET.Server_TX_Bytes)

/**
 * Get number of all received bytes from client to server
 *
 * Returns number of received bytes from client to server
 */
#define TM_ETHERNETSERVER_GetRXBytes()          (TM_ETHERNET.Server_RX_Bytes)

/**
 * Get number of all connections that have been made to server
 *
 * Returns number of connections made to server
 */
#define TM_ETHERNETSERVER_GetConnectionsCount()    (TM_ETHERNET.Server_Connections)
```

Our server works, web site is displayed to user.

Two most asked questions now probably are:

- How to display useful data to user?
- How to control device over ethernet?

Answers:

- To display useful data to user (temperature, whatever…) you **SSI tags**
- To control device (leds, PWM, whatever…), use **CGI handlers**

SSI TAGS

Let's say, that you have website and when user access to your server, you want to display current temperature to him. You will have to assign new SSI tag and pass it into SetSSITags function, which will be described below.

Then, in your HTML code, you need to add SSI tag also, which will then be replaced by LwIP to useful data.

For example, you define SSI tag in your code, named "**temperature**".
Then, in your HTML code, where you want to display it, you have to do this:
<!–#temperature–>

Important is part in bold text. Other is just normal HTML which you wanna use.

Note:

- *SSI tags only works if your file name ends with .shtml, .shtm or .ssi*
- *SSI tags can't be unlimited long*
- *SSI response text can't be unlimited long*

```
/* Maximal length for SSI tag used in HTML */
#ifndef ETHERNET_SSI_MAX_TAG_LEN
#define ETHERNET_SSI_MAX_TAG_LEN          32
#endif

/* Maximal buffer length for SSI tag insert text */
/* Content of buffer will be replaced with SSI tag */
#ifndef ETHERNET_SSI_MAX_TAG_INSERT_LEN
#define ETHERNET_SSI_MAX_TAG_INSERT_LEN       512
#endif
```

Now, you have tags defines and place into code. Next step is to set some data when this tag is found by LwIP in code. When SSI tag should be placed into code, my function for SSI will be called. This is fixed name function and is only one function for all SSI tags in your code.

```
/**
* Set SSI tags, which will be used in your HTML files for displaying variables on website.
*
* For more about SSI tags, you can check chapter 4.2 on link below:
* http://www.state-machine.com/lwip/AN_QP_and_lwIP.pdf
*
* Note: SSI tags only work if your file name ends with .shtm, .shtml, or .ssi!
*
* This function can be called only once, future calls will be denied!
*
* Parameters:
*    - TM_ETHERNET_SSI_t* tags:
*        Pointer to array of pointers for tags that are used in your html files
*    - uint16_t number_of_tags:
```

```
*       Number of tags in your array
*
* Member of TM_ETHERNET_Result_t is returned
*/
extern TM_ETHERNET_Result_t TM_ETHERNETSERVER_SetSSITags(TM_ETHERNET_SSI_t* tags, uint16_t
number_of_tags);

/**
* SSI callbacks for ethernet server
*
* This function is called, when ethernet server wants to serve tag you pass in TM_ETHERNETSERVER_SetSSITags()
function.
*
* Parameters:
*     - int iIndex:
*       Tag index number. They are specified in order you pass to TM_ETHERNETSERVER_SetSSITags function
*     - char *pcInsert:
*       Pointer where you should set your content that will be displayed where your tag is specified in files
*     - int iInsertLen:
*       Max string length
*
* With __weak parameter to prevent link errors if not defined by user
*
* Must return number of characters in your pcInsert pointer.
*/
extern __weak uint16_t TM_ETHERNETSERVER_SSICallback(int iIndex, char *pcInsert, int iInsertLen);
```

In examples below, you will see how to use tags in practise.

CGI HANDLERS

CGI handlers are a simple way to control your STM device over ethernet. They are a way on how to handle GET method with url link this: **/my_url.cgi?ledon=1&ledoff=3&ledtoggle=4**
You can use more than just one CGI handler. CGI handler is called when you access to a URL which ends with **.cgi**. In our case we have “my_url.cgi” and let’s assign function which will be called on CGI call.
It will look something like that:

```
/* LED/RELAY CGI handlers */
const char * LEDS_CGI_Handler(int iIndex, int iNumParams, char *pcParam[], char *pcValue[]);
const char * RELAY_CGI_Handler(int iIndex, int iNumParams, char *pcParam[], char *pcValue[]);

/* CGI call table, 2 CGIs used */
TM_ETHERNET_CGI_t CGI_Handlers[] = {
```

```
  {"/ledaction.cgi", LEDS_CGI_Handler}, /* LEDS_CGI_Handler will be called when user connects to "/ledaction.cgi" URL */
  {"/relayaction.cgi", LEDS_CGI_Handler}, /* RELAY_CGI_Handler will be called when user connects to "/relayaction.cgi" URL */
};
```

More about that will be clear when I show you an example

POST HANDLER

My library can also handle POST requests but I don't suggest to use this. It is not 100% stable for now, and I will not describe how to use it yet.

Functions and enumerations

Here are described all functions and enumerations. It's long part so they are hidden by default.

Ethernet functions and enumerations

Example 1

- STM configured as server mode on port 80.
- For debug purpose is used PC6 (USART6) @ 115200baud where you will be able to see initialization settings.
- How it works:
    - Data for user are stored in fsdata.c file,
    - There are SSI tags which are replaced with some settings
    - There is one CGI handlers which handles LEDs actions
    - DHCP is disabled, **192.168.0.120** IP is used

```
/**
*   Keil project template for ethernet server.
*   Data to be shown to user by default is stored in file fsdata.c where you can edit it if you want
*
*   Before you start, select your target, on the right of the "Load" button
*
*   @author      Tilen Majerle
*   @email       tilen@majerle.eu
```

```
9  *   @website   http://stm32f4-discovery.net
10 *   @ide       Keil uVision 5
11 *   @conf      PLL parameters are set in "Options for Target" -> "C/C++" -> "Defines"
12 *   @packs     STM32F4xx Keil packs version 2.2.0 or greater required
13 *   @stdperiph   STM32F4xx Standard peripheral drivers version 1.4.0 or greater required
14 */
15 /* Include core modules */
16 #include "stm32f4xx.h"
17 /* Include my libraries here */
18 #include "defines.h"
19 #include "tm_stm32f4_delay.h"
20 #include "tm_stm32f4_disco.h"
21 #include "tm_stm32f4_usart.h"
22 #include "tm_stm32f4_ethernet.h"
23 #include "tm_stm32f4_rtc.h"
24 #include "tm_stm32f4_watchdog.h"
25 #include <stdio.h>
26 #include <stdlib.h>
27
28 /* RTC data variable */
29 TM_RTC_t RTC_Data;
30
31 /* Set SSI tags for handling variables */
32 static TM_ETHERNET_SSI_t SSI_Tags[] = {
33   "led1_s", /* Tag 0 = led1 status */
34   "led2_s", /* Tag 1 = led2 status */
35   "led3_s", /* Tag 2 = led3 status */
36   "led4_s", /* Tag 3 = led4 status */
37   "srv_adr",/* Tag 4 = server address */
38   "clt_a_c",/* Tag 5 = client all connections */
39   "clt_s_c",/* Tag 6 = client successfull connections */
40   "clt_per",/* Tag 7 = client percentage */
41   "clt_tx", /* Tag 8 = client TX bytes */
42   "clt_rx", /* Tag 9 = client RX bytes */
43   "srv_c",  /* Tag 10 = server all connections */
44   "srv_tx", /* Tag 11 = server TX bytes */
45   "srv_rx", /* Tag 12 = server RX bytes */
46   "mac_adr",/* Tag 13 = MAC address */
47   "gateway",/* Tag 14 = gateway */
48   "netmask",/* Tag 15 = netmask */
49   "link",   /* Tag 16 = link status */
50   "duplex", /* Tag 17 = duplex status */
51   "hardware",/* Tag 18 = hardware where code is running */
52   "rtc_time",/* Tag 19 = current RTC time */
53   "compiled",/* Tag 20 = compiled date and time */
54 };
55
56 /* LED CGI handler */
57 const char * LEDS_CGI_Handler(int iIndex, int iNumParams, char *pcParam[], char *pcValue[]);
58
59 /* CGI call table, only one CGI used */
60 TM_ETHERNET_CGI_t CGI_Handlers[] = {
61   {"/ledaction.cgi", LEDS_CGI_Handler}, /* LEDS_CGI_Handler will be called when user connects to "/ledaction.cgi"
62 URL */
63 };
64
65 int main(void) {
```

```
66  /* Initialize system */
67  SystemInit();
68
69  /* Init USART6, TX: PC6 for debug */
70  TM_USART_Init(USART6, TM_USART_PinsPack_1, 115200);
71
72  /* Enable watchdog, 4 seconds before timeout */
73  if (TM_WATCHDOG_Init(TM_WATCHDOG_Timeout_4s)) {
74     /* Report to user */
75     printf("Reset occured because of Watchdog\n");
76  }
77
78  /* Initialize delay */
79  TM_DELAY_Init();
80
81  /* Initialize leds on board */
82  TM_DISCO_LedInit();
83
84  /* Initialize button */
85  TM_DISCO_ButtonInit();
86
87  /* Display to user */
88  printf("Program starting..\n");
89
90  /* Initialize RTC with internal clock if not already */
91  if (!TM_RTC_Init(TM_RTC_ClockSource_Internal)) {
92     /* Set default time for RTC */
93
94     /* Set date and time if RTC is not initialized already */
95     TM_RTC_SetDateTimeString("28.02.15.6;23:35:30");
96  };
97
98  /* Initialize ethernet peripheral */
99  /* All parameters NULL, default options for MAC, static IP, gateway and netmask will be used */
100 /* They are defined in tm_stm32f4_ethernet.h file */
101 if (TM_ETHERNET_Init(NULL, NULL, NULL, NULL) == TM_ETHERNET_Result_Ok) {
102    /* Successfully initialized */
103    TM_DISCO_LedOn(LED_GREEN);
104 } else {
105    /* Unsuccessfull communication */
106    TM_DISCO_LedOn(LED_RED);
107 }
108
109 /* Reset watchdog */
110 TM_WATCHDOG_Reset();
111
112 /* Initialize ethernet server if you want use it, server port 80 */
113 TM_ETHERNETSERVER_Enable(80);
114
115 /* Set SSI tags, we have 21 SSI tags */
116 TM_ETHERNETSERVER_SetSSITags(SSI_Tags, 21);
117
118 /* Set CGI tags, we have 1 CGI handler, for leds only */
119 TM_ETHERNETSERVER_SetCGIHandlers(CGI_Handlers, 1);
120
121 /* Read RTC clock */
122 TM_RTC_GetDateTime(&RTC_Data, TM_RTC_Format_BIN);
```

```
123
124   /* Print current time to USART */
125   printf("Current date: %02d:%02d:%02d\n", RTC_Data.hours, RTC_Data.minutes, RTC_Data.seconds);
126
127   /* Reset watchdog */
128   TM_WATCHDOG_Reset();
129
130   while (1) {
131     /* Update ethernet, call this as fast as possible */
132     TM_ETHERNET_Update();
133
134     /* Reset watchdog */
135     TM_WATCHDOG_Reset();
136   }
137}
138
139/* Delay 1ms handler */
140void TM_DELAY_1msHandler(void) {
141   /* Time update for ethernet, 1ms */
142   /* Add 1ms time for ethernet */
143   TM_ETHERNET_TimeUpdate(1);
144}
145
146/* Handle CGI request for LEDS */
147const char* LEDS_CGI_Handler(int iIndex, int iNumParams, char *pcParam[], char *pcValue[]) {
148   uint8_t i;
149
150   /* This function handles request like one below: */
151   /* /ledaction.cgi?ledtoggle=1&ledoff=2 */
152   /* This will toggle LED 1 and turn off LED 2 */
153
154   /* Callback */
155   if (iIndex == 0) {
156     /* Go through all params */
157     for (i = 0; i < iNumParams; i++) {
158       /* If current pair = ledtoggle=someled */
159       if (strstr(pcParam[i], "ledtoggle")) {
160         /* Switch first character */
161         switch (pcValue[i][0]) {
162           case '1':
163             TM_DISCO_LedToggle(LED_GREEN);
164             break;
165           case '2':
166             TM_DISCO_LedToggle(LED_ORANGE);
167             break;
168           case '3':
169             TM_DISCO_LedToggle(LED_RED);
170             break;
171           case '4':
172             TM_DISCO_LedToggle(LED_BLUE);
173             break;
174           default:
175             break;
176         }
177       } else if (strstr(pcParam[i], "ledon")) {
178         switch (pcValue[i][0]) {
179           case '1':
```

```
180                 TM_DISCO_LedOn(LED_GREEN);
181                 break;
182               case '2':
183                 TM_DISCO_LedOn(LED_ORANGE);
184                 break;
185               case '3':
186                 TM_DISCO_LedOn(LED_RED);
187                 break;
188               case '4':
189                 TM_DISCO_LedOn(LED_BLUE);
190                 break;
191               default:
192                 break;
193             }
194           } else if (strstr(pcParam[i], "ledoff")) {
195             switch (pcValue[i][0]) {
196               case '1':
197                 TM_DISCO_LedOff(LED_GREEN);
198                 break;
199               case '2':
200                 TM_DISCO_LedOff(LED_ORANGE);
201                 break;
202               case '3':
203                 TM_DISCO_LedOff(LED_RED);
204                 break;
205               case '4':
206                 TM_DISCO_LedOff(LED_BLUE);
207                 break;
208               default:
209                 break;
210             }
211           }
212         }
213       }
214
215       /* Return URL to be used after call */
216       return "/index.shtml";
217     }
218
219     /* SSI server callback, always is called this callback */
220     uint16_t TM_ETHERNETSERVER_SSICallback(int iIndex, char *pcInsert, int iInsertLen) {
221       uint8_t status;
222
223       /* Return number of characters written */
224       if (iIndex < 4) {
225         /* First 4 tags are leds */
226         /* Get led status */
227         switch (iIndex) {
228           case 0:
229             /* Green LED */
230             status = TM_DISCO_LedIsOn(LED_GREEN);
231             break;
232           case 1:
233             /* Orange LED */
234             status = TM_DISCO_LedIsOn(LED_ORANGE);
235             break;
236           case 2:
```

```
237             /* Red LED */
238             status = TM_DISCO_LedIsOn(LED_RED);
239             break;
240         case 3:
241             /* Blue LED */
242             status = TM_DISCO_LedIsOn(LED_BLUE);
243             break;
244         default:
245             return 0;
246     }
247
248     /* Set string according to status */
249     if (status) {
250         /* Led is ON */
251         sprintf(pcInsert, "<span class=\"green\">On</span>");
252     } else {
253         /* Led is OFF */
254         sprintf(pcInsert, "<span class=\"red\">Off</span>");
255     }
256   } else if (iIndex == 4) {
257     /* #serv_adr tag is requested */
258     sprintf(pcInsert, "%d.%d.%d.%d", TM_ETHERNET_GetLocalIP(0), TM_ETHERNET_GetLocalIP(1),
259TM_ETHERNET_GetLocalIP(2), TM_ETHERNET_GetLocalIP(3));
260   } else if (iIndex == 5) {
261     /* #clt_a_c tag */
262     sprintf(pcInsert, "%u", TM_ETHERNETCLIENT_GetConnectionsCount());
263   } else if (iIndex == 6) {
264     /* #clt_s_c tag */
265     sprintf(pcInsert, "%u", TM_ETHERNETCLIENT_GetSuccessfullConnectionsCount());
266   } else if (iIndex == 7) {
267     /* #clt_per tag */
268     if (TM_ETHERNETCLIENT_GetConnectionsCount() == 0) {
269         strcpy(pcInsert, "0 %");
270     } else {
271         sprintf(pcInsert, "%f %%", (float)TM_ETHERNETCLIENT_GetSuccessfullConnectionsCount() /
272(float)TM_ETHERNETCLIENT_GetConnectionsCount() * 100);
273     }
274   } else if (iIndex == 8) {
275     /* #clt_tx tag */
276     sprintf(pcInsert, "%llu", TM_ETHERNETCLIENT_GetTXBytes());
277   } else if (iIndex == 9) {
278     /* #clt_rx tag */
279     sprintf(pcInsert, "%llu", TM_ETHERNETCLIENT_GetRXBytes());
280   } else if (iIndex == 10) {
281     /* #srv_c tag */
282     sprintf(pcInsert, "%u", TM_ETHERNETSERVER_GetConnectionsCount());
283   } else if (iIndex == 11) {
284     /* #srv_tx tag */
285     sprintf(pcInsert, "%llu", TM_ETHERNETSERVER_GetTXBytes());
286   } else if (iIndex == 12) {
287     /* #srv_rx tag */
288     sprintf(pcInsert, "%llu", TM_ETHERNETSERVER_GetRXBytes());
289   } else if (iIndex == 13) {
290     /* #mac_adr */
291     sprintf(pcInsert, "%02X-%02X-%02X-%02X-%02X-%02X",
292         TM_ETHERNET_GetMACAddr(0),
293         TM_ETHERNET_GetMACAddr(1),
```

```
294         TM_ETHERNET_GetMACAddr(2),
295         TM_ETHERNET_GetMACAddr(3),
296         TM_ETHERNET_GetMACAddr(4),
297         TM_ETHERNET_GetMACAddr(5)
298     );
299 } else if (iIndex == 14) {
300     /* #gateway */
301     sprintf(pcInsert, "%d.%d.%d.%d",
302         TM_ETHERNET_GetGateway(0),
303         TM_ETHERNET_GetGateway(1),
304         TM_ETHERNET_GetGateway(2),
305         TM_ETHERNET_GetGateway(3)
306     );
307 } else if (iIndex == 15) {
308     /* #netmask */
309     sprintf(pcInsert, "%d.%d.%d.%d",
310         TM_ETHERNET_GetNetmask(0),
311         TM_ETHERNET_GetNetmask(1),
312         TM_ETHERNET_GetNetmask(2),
313         TM_ETHERNET_GetNetmask(3)
314     );
315 } else if (iIndex == 16) {
316     /* #link */
317     if (TM_ETHERNET_Is100M()) {
318         strcpy(pcInsert, "100Mbit");
319     } else {
320         strcpy(pcInsert, "10Mbit");
321     }
322 } else if (iIndex == 17) {
323     /* #duplex */
324     if (TM_ETHERNET_IsFullDuplex()) {
325         strcpy(pcInsert, "Full");
326     } else {
327         strcpy(pcInsert, "Half");
328     }
329 } else if (iIndex == 18) {
330     /* #hardware */
331     strcpy(pcInsert, "STM32F4-Discovery");
332 } else if (iIndex == 19) {
333     /* #rtc_time */
334     TM_RTC_GetDateTime(&RTC_Data, TM_RTC_Format_BIN);
335     sprintf(pcInsert, "%04d-%02d-%02d %02d:%02d:%02d",
336         RTC_Data.year + 2000,
337         RTC_Data.month,
338         RTC_Data.date,
339         RTC_Data.hours,
340         RTC_Data.minutes,
341         RTC_Data.seconds
342     );
343 } else if (iIndex == 20) {
344     /* #compiled */
345     strcpy(pcInsert, __DATE__ " at " __TIME__);
346 } else {
347     /* No valid tag */
348     return 0;
349 }
350
```

```
351   /* Return number of characters written in buffer */
352   return strlen(pcInsert);
353}
354
355void TM_ETHERNET_IPIsSetCallback(uint8_t ip_addr1, uint8_t ip_addr2, uint8_t ip_addr3, uint8_t ip_addr4, uint8_t
356dhcp) {
357   /* Called when we have valid IP, it might be static or DHCP */
358
359   if (dhcp) {
360      /* IP set with DHCP */
361      printf("IP: %d.%d.%d.%d assigned by DHCP server\n", ip_addr1, ip_addr2, ip_addr3, ip_addr4);
362   } else {
363      /* Static IP */
364      printf("IP: %d.%d.%d.%d; STATIC IP used\n", ip_addr1, ip_addr2, ip_addr3, ip_addr4);
365   }
366
367   /* Print MAC address to user */
368   printf("MAC: %02X-%02X-%02X-%02X-%02X-%02X\n",
369      TM_ETHERNET_GetMACAddr(0),
370      TM_ETHERNET_GetMACAddr(1),
371      TM_ETHERNET_GetMACAddr(2),
372      TM_ETHERNET_GetMACAddr(3),
373      TM_ETHERNET_GetMACAddr(4),
374      TM_ETHERNET_GetMACAddr(5)
375   );
376   /* Print netmask to user */
377   printf("Netmask: %d.%d.%d.%d\n",
378      TM_ETHERNET_GetGateway(0),
379      TM_ETHERNET_GetGateway(1),
380      TM_ETHERNET_GetGateway(2),
381      TM_ETHERNET_GetGateway(3)
382   );
383   /* Print gateway to user */
384   printf("Gateway: %d.%d.%d.%d\n",
385      TM_ETHERNET_GetNetmask(0),
386      TM_ETHERNET_GetNetmask(1),
387      TM_ETHERNET_GetNetmask(2),
388      TM_ETHERNET_GetNetmask(3)
389   );
390   /* Print 100M link status, 1 = 100M, 0 = 10M */
391   printf("Link 100M: %d\n", TM_ETHERNET.speed_100m);
392   /* Print duplex status: 1 = Full, 0 = Half */
393   printf("Full duplex: %d\n", TM_ETHERNET.full_duplex);
394}
395
396void TM_ETHERNET_LinkIsDownCallback(void) {
397   /* This function will be called when ethernet cable will not be plugged */
398   /* It will also be called on initialization if connection is not detected */
399
400   /* Print to user */
401   printf("Link is down, do you have connected to your modem/router?");
402}
403void TM_ETHERNET_LinkIsUpCallback(void) {
404   /* Cable has been plugged in back, link is detected */
405   /* I suggest you that you reboot MCU here */
406   /* Do important stuff before */
407
```

```
  /* Print to user */
  printf("Link is up back\n");
}

/* For printf function */
int fputc(int ch, FILE *f) {
  /* Send over usart */
  TM_USART_Putc(USART6, ch);

  /* Return character back */
  return ch;
}
```

Example 2

- STM configured as server mode on port 80.
- For debug purpose is used PC6 (USART6) @ 115200baud where you will be able to see initialization settings.
- How it works:
    - Data for user are stored on SD card (how you will implement low layer is on you, but you can use my FatFs library as well)
    - There are SSI tags which are replaced with some settings
    - There is one CGI handlers which handles LEDs actions
    - DHCP is disabled, **192.168.0.120** IP is used

Response on PC6 pin

```
/**
*   Keil project template for ethernet server.
*   File to be shown to user is stored on SD card
*
*   Before you start, select your target, on the right of the "Load" button
*
*   @author       Tilen Majerle
*   @email       tilen@majerle.eu
*   @website    http://stm32f4-discovery.net
*   @ide       Keil uVision 5
*   @conf       PLL parameters are set in "Options for Target" -> "C/C++" -> "Defines"
*   @packs       STM32F4xx Keil packs version 2.2.0 or greater required
*   @stdperiph    STM32F4xx Standard peripheral drivers version 1.4.0 or greater required
*/
/* Include core modules */
#include "stm32f4xx.h"
/* Include my libraries here */
#include "defines.h"
#include "tm_stm32f4_delay.h"
#include "tm_stm32f4_disco.h"
#include "tm_stm32f4_usart.h"
#include "tm_stm32f4_ethernet.h"
#include "tm_stm32f4_rtc.h"
#include "tm_stm32f4_watchdog.h"
#include "tm_stm32f4_fatfs.h"
#include <stdio.h>
#include <stdlib.h>

/* File variable */
FIL fil[ETHERNET_MAX_OPEN_FILES];
/* Fatfs variable */
FATFS fs;

/* RTC data variable */
TM_RTC_t RTC_Data;

/* Set SSI tags for handling variables */
static TM_ETHERNET_SSI_t SSI_Tags[] = {
  "led1_s", /* Tag 0 = led1 status */
  "led2_s", /* Tag 1 = led2 status */
  "led3_s", /* Tag 2 = led3 status */
  "led4_s", /* Tag 3 = led4 status */
  "srv_adr",/* Tag 4 = server address */
  "clt_a_c",/* Tag 5 = client all connections */
  "clt_s_c",/* Tag 6 = client successfull connections */
  "clt_per",/* Tag 7 = client percentage */
  "clt_tx", /* Tag 8 = client TX bytes */
  "clt_rx", /* Tag 9 = client RX bytes */
  "srv_c",  /* Tag 10 = server all connections */
  "srv_tx", /* Tag 11 = server TX bytes */
  "srv_rx", /* Tag 12 = server RX bytes */
```

```
52    "mac_adr",/* Tag 13 = MAC address */
53    "gateway",/* Tag 14 = gateway */
54    "netmask",/* Tag 15 = netmask */
55    "link",   /* Tag 16 = link status */
56    "duplex", /* Tag 17 = duplex status */
57    "hardware",/* Tag 18 = hardware where code is running */
58    "rtc_time",/* Tag 19 = current RTC time */
59    "compiled",/* Tag 20 = compiled date and time */
60 };
61
62 /* LED CGI handler */
63 const char * LEDS_CGI_Handler(int iIndex, int iNumParams, char *pcParam[], char *pcValue[]);
64
65 /* CGI call table, only one CGI used */
66 TM_ETHERNET_CGI_t CGI_Handlers[] = {
67    {"/ledaction.cgi", LEDS_CGI_Handler}, /* LEDS_CGI_Handler will be called when user connects to "/ledaction.cgi"
68 URL */
69 };
70
71 int main(void) {
72    /* Initialize system */
73    SystemInit();
74
75    /* Init USART6, TX: PC6 for debug */
76    TM_USART_Init(USART6, TM_USART_PinsPack_1, 115200);
77
78    /* Enable watchdog, 4 seconds before timeout */
79    if (TM_WATCHDOG_Init(TM_WATCHDOG_Timeout_4s)) {
80       /* Report to user */
81       printf("Reset occured because of Watchdog\n");
82    }
83
84    /* Initialize delay */
85    TM_DELAY_Init();
86
87    /* Initialize leds on board */
88    TM_DISCO_LedInit();
89
90    /* Initialize button */
91    TM_DISCO_ButtonInit();
92
93    /* Display to user */
94    printf("Program starting..\n");
95
96    /* Initialize RTC with internal clock if not already */
97    if (!TM_RTC_Init(TM_RTC_ClockSource_Internal)) {
98       /* Set default time for RTC */
99
100      /* Set date and time if RTC is not initialized already */
101      TM_RTC_SetDateTimeString("28.02.15.6;23:35:30");
102   };
103
104   /* Initialize ethernet peripheral */
105   /* All parameters NULL, default options for MAC, static IP, gateway and netmask will be used */
106   /* They are defined in tm_stm32f4_ethernet.h file */
107   if (TM_ETHERNET_Init(NULL, NULL, NULL, NULL) == TM_ETHERNET_Result_Ok) {
108      /* Successfully initialized */
```

```
    TM_DISCO_LedOn(LED_GREEN);
  } else {
    /* Unsuccessfull communication */
    TM_DISCO_LedOn(LED_RED);
  }

  /* Reset watchdog */
  TM_WATCHDOG_Reset();

  /* Initialize ethernet server if you want use it, server port 80 */
  TM_ETHERNETSERVER_Enable(80);

  /* Set SSI tags, we have 21 SSI tags */
  TM_ETHERNETSERVER_SetSSITags(SSI_Tags, 21);

  /* Set CGI tags, we have 1 CGI handler, for leds only */
  TM_ETHERNETSERVER_SetCGIHandlers(CGI_Handlers, 1);

  /* Read RTC clock */
  TM_RTC_GetDateTime(&RTC_Data, TM_RTC_Format_BIN);

  /* Print current time to USART */
  printf("Current date: %02d:%02d:%02d\n", RTC_Data.hours, RTC_Data.minutes, RTC_Data.seconds);

  /* Reset watchdog */
  TM_WATCHDOG_Reset();

  while (1) {
    /* Update ethernet, call this as fast as possible */
    TM_ETHERNET_Update();

    /* If button pressed, toggle server status */
    if (TM_DISCO_ButtonOnPressed()) {
      /* If server is enabled */
      if (TM_ETHERNETSERVER_Enabled()) {
        /* Disable it */
        TM_ETHERNETSERVER_Disable();
        /* Print to user */
        printf("Server disabled\n");
      } else {
        /* Enable it */
        TM_ETHERNETSERVER_Enable(80);
        /* Print to user */
        printf("Server enabled\n");
      }
    }

    /* Reset watchdog */
    TM_WATCHDOG_Reset();
  }
}

/* Delay 1ms handler */
void TM_DELAY_1msHandler(void) {
  /* Time update for ethernet, 1ms */
  /* Add 1ms time for ethernet */
  TM_ETHERNET_TimeUpdate(1);
```

```
166}
167
168/* Handle CGI request for LEDS */
169const char* LEDS_CGI_Handler(int iIndex, int iNumParams, char *pcParam[], char *pcValue[]) {
170   uint8_t i;
171
172   /* This function handles request like one below: */
173   /* /ledaction.cgi?ledtoggle=1&ledoff=2 */
174   /* This will toggle LED 1 and turn off LED 2 */
175
176   /* Callback */
177   if (iIndex == 0) {
178      /* Go through all params */
179      for (i = 0; i < iNumParams; i++) {
180         /* If current pair = ledtoggle=someled */
181         if (strstr(pcParam[i], "ledtoggle")) {
182            /* Switch first character */
183            switch (pcValue[i][0]) {
184               case '1':
185                  TM_DISCO_LedToggle(LED_GREEN);
186                  break;
187               case '2':
188                  TM_DISCO_LedToggle(LED_ORANGE);
189                  break;
190               case '3':
191                  TM_DISCO_LedToggle(LED_RED);
192                  break;
193               case '4':
194                  TM_DISCO_LedToggle(LED_BLUE);
195                  break;
196               default:
197                  break;
198            }
199         } else if (strstr(pcParam[i], "ledon")) {
200            switch (pcValue[i][0]) {
201               case '1':
202                  TM_DISCO_LedOn(LED_GREEN);
203                  break;
204               case '2':
205                  TM_DISCO_LedOn(LED_ORANGE);
206                  break;
207               case '3':
208                  TM_DISCO_LedOn(LED_RED);
209                  break;
210               case '4':
211                  TM_DISCO_LedOn(LED_BLUE);
212                  break;
213               default:
214                  break;
215            }
216         } else if (strstr(pcParam[i], "ledoff")) {
217            switch (pcValue[i][0]) {
218               case '1':
219                  TM_DISCO_LedOff(LED_GREEN);
220                  break;
221               case '2':
222                  TM_DISCO_LedOff(LED_ORANGE);
```

```
223                 break;
224             case '3':
225                 TM_DISCO_LedOff(LED_RED);
226                 break;
227             case '4':
228                 TM_DISCO_LedOff(LED_BLUE);
229                 break;
230             default:
231                 break;
232             }
233         }
234     }
235 }
236
237 /* Return URL to be used after call */
238 return "/index.shtml";
239 }
240
241 /* SSI server callback, always is called this callback */
242 uint16_t TM_ETHERNETSERVER_SSICallback(int iIndex, char *pcInsert, int iInsertLen) {
243     uint8_t status;
244
245     /* Return number of characters written */
246     if (iIndex < 4) {
247         /* First 4 tags are leds */
248         /* Get led status */
249         switch (iIndex) {
250             case 0:
251                 /* Green LED */
252                 status = TM_DISCO_LedIsOn(LED_GREEN);
253                 break;
254             case 1:
255                 /* Orange LED */
256                 status = TM_DISCO_LedIsOn(LED_ORANGE);
257                 break;
258             case 2:
259                 /* Red LED */
260                 status = TM_DISCO_LedIsOn(LED_RED);
261                 break;
262             case 3:
263                 /* Blue LED */
264                 status = TM_DISCO_LedIsOn(LED_BLUE);
265                 break;
266             default:
267                 return 0;
268         }
269
270         /* Set string according to status */
271         if (status) {
272             /* Led is ON */
273             sprintf(pcInsert, "<span class=\"green\">On</span>");
274         } else {
275             /* Led is OFF */
276             sprintf(pcInsert, "<span class=\"red\">Off</span>");
277         }
278     } else if (iIndex == 4) {
279         /* #serv_adr tag is requested */
```

```
280     sprintf(pcInsert, "%d.%d.%d.%d", TM_ETHERNET_GetLocalIP(0), TM_ETHERNET_GetLocalIP(1),
281TM_ETHERNET_GetLocalIP(2), TM_ETHERNET_GetLocalIP(3));
282   } else if (iIndex == 5) {
283     /* #clt_a_c tag */
284     sprintf(pcInsert, "%u", TM_ETHERNETCLIENT_GetConnectionsCount());
285   } else if (iIndex == 6) {
286     /* #clt_s_c tag */
287     sprintf(pcInsert, "%u", TM_ETHERNETCLIENT_GetSuccessfullConnectionsCount());
288   } else if (iIndex == 7) {
289     /* #clt_per tag */
290     if (TM_ETHERNETCLIENT_GetConnectionsCount() == 0) {
291       strcpy(pcInsert, "0 %");
292     } else {
293       sprintf(pcInsert, "%f %%", (float)TM_ETHERNETCLIENT_GetSuccessfullConnectionsCount() /
294(float)TM_ETHERNETCLIENT_GetConnectionsCount() * 100);
295     }
296   } else if (iIndex == 8) {
297     /* #clt_tx tag */
298     sprintf(pcInsert, "%llu", TM_ETHERNETCLIENT_GetTXBytes());
299   } else if (iIndex == 9) {
300     /* #clt_rx tag */
301     sprintf(pcInsert, "%llu", TM_ETHERNETCLIENT_GetRXBytes());
302   } else if (iIndex == 10) {
303     /* #srv_c tag */
304     sprintf(pcInsert, "%u", TM_ETHERNETSERVER_GetConnectionsCount());
305   } else if (iIndex == 11) {
306     /* #srv_tx tag */
307     sprintf(pcInsert, "%llu", TM_ETHERNETSERVER_GetTXBytes());
308   } else if (iIndex == 12) {
309     /* #srv_rx tag */
310     sprintf(pcInsert, "%llu", TM_ETHERNETSERVER_GetRXBytes());
311   } else if (iIndex == 13) {
312     /* #mac_adr */
313     sprintf(pcInsert, "%02X-%02X-%02X-%02X-%02X-%02X",
314       TM_ETHERNET_GetMACAddr(0),
315       TM_ETHERNET_GetMACAddr(1),
316       TM_ETHERNET_GetMACAddr(2),
317       TM_ETHERNET_GetMACAddr(3),
318       TM_ETHERNET_GetMACAddr(4),
319       TM_ETHERNET_GetMACAddr(5)
320     );
321   } else if (iIndex == 14) {
322     /* #gateway */
323     sprintf(pcInsert, "%d.%d.%d.%d",
324       TM_ETHERNET_GetGateway(0),
325       TM_ETHERNET_GetGateway(1),
326       TM_ETHERNET_GetGateway(2),
327       TM_ETHERNET_GetGateway(3)
328     );
329   } else if (iIndex == 15) {
330     /* #netmask */
331     sprintf(pcInsert, "%d.%d.%d.%d",
332       TM_ETHERNET_GetNetmask(0),
333       TM_ETHERNET_GetNetmask(1),
334       TM_ETHERNET_GetNetmask(2),
335       TM_ETHERNET_GetNetmask(3)
336     );
```

```
337   } else if (iIndex == 16) {
338     /* #link */
339     if (TM_ETHERNET_Is100M()) {
340       strcpy(pcInsert, "100Mbit");
341     } else {
342       strcpy(pcInsert, "10Mbit");
343     }
344   } else if (iIndex == 17) {
345     /* #duplex */
346     if (TM_ETHERNET_IsFullDuplex()) {
347       strcpy(pcInsert, "Full");
348     } else {
349       strcpy(pcInsert, "Half");
350     }
351   } else if (iIndex == 18) {
352     /* #hardware */
353     strcpy(pcInsert, "STM32F4-Discovery");
354   } else if (iIndex == 19) {
355     /* #rtc_time */
356     TM_RTC_GetDateTime(&RTC_Data, TM_RTC_Format_BIN);
357     sprintf(pcInsert, "%04d-%02d-%02d %02d:%02d:%02d",
358       RTC_Data.year + 2000,
359       RTC_Data.month,
360       RTC_Data.date,
361       RTC_Data.hours,
362       RTC_Data.minutes,
363       RTC_Data.seconds
364     );
365   } else if (iIndex == 20) {
366     /* #compiled */
367     strcpy(pcInsert, __DATE__ " at " __TIME__);
368   } else {
369     /* No valid tag */
370     return 0;
371   }
372
373   /* Return number of characters written in buffer */
374   return strlen(pcInsert);
375 }
376
377 void TM_ETHERNET_IPIsSetCallback(uint8_t ip_addr1, uint8_t ip_addr2, uint8_t ip_addr3, uint8_t ip_addr4, uint8_t
378 dhcp) {
379   /* Called when we have valid IP, it might be static or DHCP */
380
381   if (dhcp) {
382     /* IP set with DHCP */
383     printf("IP: %d.%d.%d.%d assigned by DHCP server\n", ip_addr1, ip_addr2, ip_addr3, ip_addr4);
384   } else {
385     /* Static IP */
386     printf("IP: %d.%d.%d.%d; STATIC IP used\n", ip_addr1, ip_addr2, ip_addr3, ip_addr4);
387   }
388
389   /* Print MAC address to user */
390   printf("MAC: %02X-%02X-%02X-%02X-%02X-%02X\n",
391     TM_ETHERNET_GetMACAddr(0),
392     TM_ETHERNET_GetMACAddr(1),
393     TM_ETHERNET_GetMACAddr(2),
```

```
394     TM_ETHERNET_GetMACAddr(3),
395     TM_ETHERNET_GetMACAddr(4),
396     TM_ETHERNET_GetMACAddr(5)
397   );
398   /* Print netmask to user */
399   printf("Netmask: %d.%d.%d.%d\n",
400     TM_ETHERNET_GetGateway(0),
401     TM_ETHERNET_GetGateway(1),
402     TM_ETHERNET_GetGateway(2),
403     TM_ETHERNET_GetGateway(3)
404   );
405   /* Print gateway to user */
406   printf("Gateway: %d.%d.%d.%d\n",
407     TM_ETHERNET_GetNetmask(0),
408     TM_ETHERNET_GetNetmask(1),
409     TM_ETHERNET_GetNetmask(2),
410     TM_ETHERNET_GetNetmask(3)
411   );
412   /* Print 100M link status, 1 = 100M, 0 = 10M */
413   printf("Link 100M: %d\n", TM_ETHERNET.speed_100m);
414   /* Print duplex status: 1 = Full, 0 = Half */
415   printf("Full duplex: %d\n", TM_ETHERNET.full_duplex);
416 }
417
418 void TM_ETHERNET_DHCPStartCallback(void) {
419   /* Called when has DHCP started with getting IP address */
420   printf("Trying to get IP address via DHCP\n");
421 }
422
423 void TM_ETHERNET_LinkIsDownCallback(void) {
424   /* This function will be called when ethernet cable will not be plugged */
425   /* It will also be called on initialization if connection is not detected */
426
427   /* Print to user */
428   printf("Link is down, do you have connected to your modem/router?");
429 }
430
431 void TM_ETHERNET_LinkIsUpCallback(void) {
432   /* Cable has been plugged in back, link is detected */
433   /* I suggest you that you reboot MCU here */
434   /* Do important stuff before */
435
436   /* Print to user */
437   printf("Link is up back\n");
438 }
439
440 int TM_ETHERNETSERVER_OpenFileCallback(struct fs_file* file, const char* name) {
441   FRESULT fres;
442   char buffer[100];
443
444   /* Print which file you will try to open */
445   printf("Trying to open file %s\n", name);
446
447   /* Mount card, it will be mounted when needed */
448   if ((fres = f_mount(&fs, "", 1)) != FR_OK) {
449     printf("Mount error: %d\n", fres);
450   }
```

```
451
452   /* Format name, I have on subfolder everything on my SD card */
453   sprintf((char *)buffer, "/www%s", name);
454
455   /* Try to open */
456   fres = f_open(&fil[file->id], buffer, FA_OPEN_EXISTING | FA_READ | FA_WRITE);
457
458   /* If not opened OK */
459   if (fres != FR_OK) {
460      /* In case we are only opened file, but we didn't succedded */
461      if (*file->opened_files_count == 0) {
462         /* Unmount card, for safety reason */
463         f_mount(NULL, "", 1);
464      }
465
466      /* Return 0, opening error */
467      return 0;
468   }
469
470   /* !IMPORTANT; Set file size */
471   file->len = f_size(&fil[file->id]);
472
473   /* Return 1, file opened OK */
474   return 1;
475}
476
477int TM_ETHERNETSERVER_ReadFileCallback(struct fs_file* file, char* buffer, int count) {
478   uint32_t readed;
479
480   /* print debug */
481   printf("Trying to read %d bytes from file %s\n", count, file->file_name);
482
483   /* End of file? */
484   if (f_eof(&fil[file->id])) {
485      return -1;
486   }
487
488   /* Read max block */
489   if (count > 65535) {
490      count = 65535;
491   }
492
493   /* Read data */
494   f_read(&fil[file->id], buffer, count, &readed);
495
496   /* Return number of bytes read */
497   return readed;
498}
499
500void TM_ETHERNETSERVER_CloseFileCallback(struct fs_file* file) {
501   /* Close file */
502   f_close(&fil[file->id]);
503
504   /* Print to user */
505   printf("Closing file %s\n", file->file_name);
506
507   /* Unmount in case there is no opened files anymore */
```

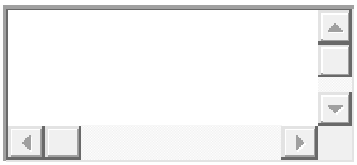
```
  if (!*file->opened_files_count) {
    /* Unmount, protect SD card */
    f_mount(NULL, "", 1);
  }
}

/* Client is connected */
uint8_t TM_ETHERNETSERVER_ClientConnectedCallback(struct tcp_pcb *pcb) {
  struct ip_addr ip;
  /* Fill bad IP */
  IP4_ADDR(&ip, 84, 12, 16, 46);

  /* Check IP address */
  if (pcb->remote_ip.addr == ip.addr) {
    /* Print to user */
    printf("User with bad IP was trying to access to website\n");
    /* Disable access, show error page */
    return 0;
  }
  /* Print to user */
  printf("Connection allowed\n");
  /* Allow access to others */
  return 1;
}

void TM_ETHERNETSERVER_ClientDisconnectedCallback(void) {
  /* Print to user */
  printf("Client disconnected\n");
}

/* For printf function */
int fputc(int ch, FILE *f) {
  /* Send over usart */
  TM_USART_Putc(USART6, ch);

  /* Return character back */
  return ch;
}
```

Example 3

- Server disabled in this example (but it can be enabled too without problems)
- Client example
- For debug purpose is used PC6 (USART6) @ 115200baud where you will be able to see initialization settings.
- How it works:
    - After initialization, DNS will try to get IP address for stm32f4-discovery.net

- On each button click, a new TCP connection will be established to IP which will be returned from DNS
- Response from server will be displayed on USART debug output
- DHCP is still disabled

Ethernet response client request



```
/**
*   Keil project template for ethernet client and DNS.
*
*   Before you start, select your target, on the right of the "Load" button
*
*   @author      Tilen Majerle
*   @email       tilen@majerle.eu
*   @website     http://stm32f4-discovery.net
*   @ide         Keil uVision 5
*   @conf        PLL parameters are set in "Options for Target" -> "C/C++" -> "Defines"
*   @packs       STM32F4xx Keil packs version 2.2.0 or greater required
*   @stdperiph   STM32F4xx Standard peripheral drivers version 1.4.0 or greater required
*/
/* Include core modules */
#include "stm32f4xx.h"
/* Include my libraries here */
#include "defines.h"
#include "tm_stm32f4_delay.h"
#include "tm_stm32f4_disco.h"
#include "tm_stm32f4_usart.h"
#include "tm_stm32f4_ethernet.h"
#include "tm_stm32f4_watchdog.h"
#include <stdio.h>
#include <stdlib.h>

/* Create simple typedef for DNS controlling */
typedef struct {
    uint8_t Working;
    uint8_t HaveIP;
    uint8_t ip[4];
} TM_DNS_t;
```

```
32 TM_DNS_t MyDNS;
33
34 uint16_t requests_count = 1;
35
36 int main(void) {
37   /* Initialize system */
38   SystemInit();
39
40   /* Init USART6, TX: PC6 for debug */
41   TM_USART_Init(USART6, TM_USART_PinsPack_1, 115200);
42
43   /* Enable watchdog, 4 seconds before timeout */
44   if (TM_WATCHDOG_Init(TM_WATCHDOG_Timeout_4s)) {
45     /* Report to user */
46     printf("Reset occured because of Watchdog\n");
47   }
48
49   /* Initialize delay */
50   TM_DELAY_Init();
51
52   /* Initialize leds on board */
53   TM_DISCO_LedInit();
54
55   /* Initialize button */
56   TM_DISCO_ButtonInit();
57
58   /* Display to user */
59   printf("Program starting..\n");
60
61   /* Initialize ethernet peripheral */
62   /* All parameters NULL, default options for MAC, static IP, gateway and netmask will be used */
63   /* They are defined in tm_stm32f4_ethernet.h file */
64   if (TM_ETHERNET_Init(NULL, NULL, NULL, NULL) == TM_ETHERNET_Result_Ok) {
65     /* Successfully initialized */
66     TM_DISCO_LedOn(LED_GREEN);
67   } else {
68     /* Unsuccessfull communication */
69     TM_DISCO_LedOn(LED_RED);
70   }
71
72   /* Reset watchdog */
73   TM_WATCHDOG_Reset();
74
75   while (1) {
76     /* Update ethernet, call this as fast as possible */
77     TM_ETHERNET_Update();
78
79     /* If DNS is not working and we don't have IP yet */
80     if (MyDNS.Working == 0 && MyDNS.HaveIP == 0) {
81       /* Try to start DNS */
82       /* It will return error in case Ethernet is not ready yet so you have to try more than one time */
83       if (TM_ETHERNETDNS_GetHostByName("stm32f4-discovery.net") == TM_ETHERNET_Result_Ok) {
84         /* We started with working */
85         MyDNS.Working = 1;
86       }
87     }
88
```

```
    /* On button pressed, make a new connection and if we have IP known */
    if (TM_DISCO_ButtonOnPressed() && MyDNS.HaveIP) {
      /* Try to make a new connection, port 80 */
      if (TM_ETHERNETCLIENT_Connect("stm32f4-discovery.net", MyDNS.ip[0], MyDNS.ip[1], MyDNS.ip[2],
MyDNS.ip[3], 80, &requests_count) != TM_ETHERNET_Result_Ok) {
        /* Print to user */
        printf("Can not make a new connection!\n");
      }
    }

    /* Reset watchdog */
    TM_WATCHDOG_Reset();
  }
}

/* Delay 1ms handler */
void TM_DELAY_1msHandler(void) {
  /* Time update for ethernet, 1ms */
  /* Add 1ms time for ethernet */
  TM_ETHERNET_TimeUpdate(1);
}

uint16_t TM_ETHERNETCLIENT_CreateHeadersCallback(TM_TCPCLIENT_t* connection, char* buffer, uint16_t
buffer_length) {
  /* Create request headers */
  sprintf(buffer, "GET /hello_world.php?number=%d HTTP/1.1\r\n", *(uint16_t *)connection->user_parameters);
  strcat(buffer, "Host: stm32f4-discovery.net\r\n");
  strcat(buffer, "Connection: close\r\n");
  strcat(buffer, "\r\n");

  /* Return number of bytes in buffer */
  return strlen(buffer);
}

void TM_ETHERNETCLIENT_ReceiveDataCallback(TM_TCPCLIENT_t* connection, uint8_t* buffer, uint16_t
buffer_length, uint16_t total_length) {
  uint16_t i = 0;

  /* We have available data for connection to receive */
  printf("Receiveing %d bytes of data from %s\n", buffer_length, connection->name);

  /* Go through entire buffer, remove response headers */
  if (connection->headers_done == 0) {
    for (i = 0; i < buffer_length; i++) {
      if (
        buffer[i] == '\r' &&
        buffer[i + 1] == '\n' &&
        buffer[i + 2] == '\r' &&
        buffer[i + 3] == '\n'
      ) {
        /* Headers done */
        connection->headers_done = 1;
        /* Increase i */
        i += 3;
        /* Break */
        break;
      }
```

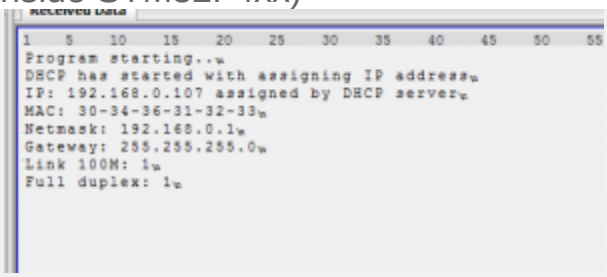
```
146         }
147     }
148 
149     /* Print data */
150     for (; i < buffer_length; i++) {
151         /* Print response */
152         printf("%c", buffer[i]);
153     }
154 }
155 
156 void TM_ETHERNETCLIENT_ConnectedCallback(TM_TCPCLIENT_t* connection) {
157     /* We are connected */
158     printf("We are connected to %s\n", connection->name);
159 }
160 
161 void TM_ETHERNETCLIENT_ConnectionClosedCallback(TM_TCPCLIENT_t* connection, uint8_t success) {
162     /* We are disconnected, done with connection */
163     if (success) {
164         printf("Connection %s was successfully closed. Number of active connections: %d\n", connection->name,
165 *connection->active_connections_count);
166     } else {
167         printf("Connection %s was closed because of error. Number of active connections: %d\n", connection->name,
168 *connection->active_connections_count);
169     }
170 
171     /* Increase number of requests */
172     requests_count++;
173 }
174 
175 void TM_ETHERNETCLIENT_ErrorCallback(TM_TCPCLIENT_t* connection) {
176     /* Print to user */
177     printf("An error occured on connection %s\n", connection->name);
178 }
179 
180 void TM_ETHERNETCLIENT_ConnectionStartedCallback(TM_TCPCLIENT_t* connection) {
181     /* Print to user */
182     printf("Connection %s has started\n", connection->name);
183 }
184 
185 /* DNS has IP */
186 void TM_ETHERNETDNS_FoundCallback(char* host_name, uint8_t ip_addr1, uint8_t ip_addr2, uint8_t ip_addr3,
187 uint8_t ip_addr4) {
188     /* If host name is stm32f4-discovery.net */
189     if (strstr(host_name, "stm32f4-discovery.net")) {
190         /* We have IP */
191         MyDNS.HaveIP = 1;
192         /* Save IP */
193         MyDNS.ip[0] = ip_addr1;
194         MyDNS.ip[1] = ip_addr2;
195         MyDNS.ip[2] = ip_addr3;
196         MyDNS.ip[3] = ip_addr4;
197         /* We are not working anymore */
198         MyDNS.Working = 0;
199 
200         /* Print to user */
201         printf("We have IP address for %s: %d.%d.%d.%d\n", host_name, ip_addr1, ip_addr2, ip_addr3, ip_addr4);
202     }
```

```
}

/* DNS error callback */
void TM_ETHERNETDNS_ErrorCallback(char* host_name) {
  /* If host name is stm32f4-discovery.net */
  if (strstr(host_name, "stm32f4-discovery.net")) {
    /* We have IP */
    MyDNS.HaveIP = 0;
    /* We are not working anymore */
    MyDNS.Working = 0;
  }
  /* Print to user */
  printf("DNS has failed to get IP address for %s\n", host_name);
}

void TM_ETHERNET_IPIsSetCallback(uint8_t ip_addr1, uint8_t ip_addr2, uint8_t ip_addr3, uint8_t ip_addr4, uint8_t
dhcp) {
  /* Called when we have valid IP, it might be static or DHCP */

  if (dhcp) {
    /* IP set with DHCP */
    printf("IP: %d.%d.%d.%d assigned by DHCP server\n", ip_addr1, ip_addr2, ip_addr3, ip_addr4);
  } else {
    /* Static IP */
    printf("IP: %d.%d.%d.%d; STATIC IP used\n", ip_addr1, ip_addr2, ip_addr3, ip_addr4);
  }

  /* Print MAC address to user */
  printf("MAC: %02X-%02X-%02X-%02X-%02X-%02X\n",
    TM_ETHERNET_GetMACAddr(0),
    TM_ETHERNET_GetMACAddr(1),
    TM_ETHERNET_GetMACAddr(2),
    TM_ETHERNET_GetMACAddr(3),
    TM_ETHERNET_GetMACAddr(4),
    TM_ETHERNET_GetMACAddr(5)
  );
  /* Print netmask to user */
  printf("Netmask: %d.%d.%d.%d\n",
    TM_ETHERNET_GetGateway(0),
    TM_ETHERNET_GetGateway(1),
    TM_ETHERNET_GetGateway(2),
    TM_ETHERNET_GetGateway(3)
  );
  /* Print gateway to user */
  printf("Gateway: %d.%d.%d.%d\n",
    TM_ETHERNET_GetNetmask(0),
    TM_ETHERNET_GetNetmask(1),
    TM_ETHERNET_GetNetmask(2),
    TM_ETHERNET_GetNetmask(3)
  );
  /* Print 100M link status, 1 = 100M, 0 = 10M */
  printf("Link 100M: %d\n", TM_ETHERNET.speed_100m);
  /* Print duplex status: 1 = Full, 0 = Half */
  printf("Full duplex: %d\n", TM_ETHERNET.full_duplex);
}

void TM_ETHERNET_LinkIsDownCallback(void) {
```

```
260   /* This function will be called when ethernet cable will not be plugged */
261   /* It will also be called on initialization if connection is not detected */
262
263   /* Print to user */
264   printf("Link is down, do you have connected to your modem/router?");
265}
266void TM_ETHERNET_LinkIsUpCallback(void) {
267   /* Cable has been plugged in back, link is detected */
268   /* I suggest you that you reboot MCU here */
269   /* Do important stuff before */
270
271   /* Print to user */
272   printf("Link is up back\n");
273}
274
275/* For printf function */
    int fputc(int ch, FILE *f) {
      /* Send over usart */
      TM_USART_Putc(USART6, ch);

      /* Return character back */
      return ch;
    }
```

Example 4

- Server disabled in this example (but it can be enabled too without problems)
- DHCP example
- For debug purpose is used PC6 (USART6) @ 115200baud where you will be able to see initialization settings.
- How it works:
    - After initialization, DHCP will try to assign IP address to router
    - “Dynamic” MAC address is set to device (from unique ID inside STM32F4xx)



Ethernet local IP assigned by DHCP

```
/**
 *  Defines for your entire project at one place
 *
 *   @author     Tilen Majerle
 *   @email      tilen@majerle.eu
 *   @website    http://stm32f4-discovery.net
 *   @version    v1.0
 *   @ide        Keil uVision 5
 *   @license    GNU GPL v3
 *
 * |----------------------------------------------------------------------
 * | Copyright (C) Tilen Majerle, 2014
 * |
 * | This program is free software: you can redistribute it and/or modify
 * | it under the terms of the GNU General Public License as published by
 * | the Free Software Foundation, either version 3 of the License, or
 * | any later version.
 * |
 * | This program is distributed in the hope that it will be useful,
 * | but WITHOUT ANY WARRANTY; without even the implied warranty of
 * | MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * | GNU General Public License for more details.
 * |
 * | You should have received a copy of the GNU General Public License
 * | along with this program.  If not, see <http://www.gnu.org/licenses/>.
 * |----------------------------------------------------------------------
 */
#ifndef TM_DEFINES_H
#define TM_DEFINES_H

/* Put your global defines for all libraries here used in your project */

/* Enable DHCP IP assignment */
#define ETHERNET_USE_DHCP

#endif
```

```
/**
 *   Keil project template for ethernet DHCP
 *
 *   Before you start, select your target, on the right of the "Load" button
 *
 *   @author       Tilen Majerle
 *   @email        tilen@majerle.eu
```

```
8   *   @website   http://stm32f4-discovery.net
9   *   @ide       Keil uVision 5
10  *   @conf      PLL parameters are set in "Options for Target" -> "C/C++" -> "Defines"
11  *   @packs     STM32F4xx Keil packs version 2.2.0 or greater required
12  *   @stdperiph   STM32F4xx Standard peripheral drivers version 1.4.0 or greater required
13  */
14 /* Include core modules */
15 #include "stm32f4xx.h"
16 /* Include my libraries here */
17 #include "defines.h"
18 #include "tm_stm32f4_delay.h"
19 #include "tm_stm32f4_disco.h"
20 #include "tm_stm32f4_usart.h"
21 #include "tm_stm32f4_ethernet.h"
22 #include "tm_stm32f4_watchdog.h"
23 #include "tm_stm32f4_id.h"
24 #include <stdio.h>
25 #include <stdlib.h>
26
27 int main(void) {
28     uint8_t i;
29     uint8_t mac_address[6];
30     uint8_t ip_address[] = {192, 168, 0, 150};
31
32     /* Initialize system */
33     SystemInit();
34
35     /* Init USART6, TX: PC6 for debug */
36     TM_USART_Init(USART6, TM_USART_PinsPack_1, 115200);
37
38     /* Enable watchdog, 4 seconds before timeout */
39     if (TM_WATCHDOG_Init(TM_WATCHDOG_Timeout_4s)) {
40         /* Report to user */
41         printf("Reset occured because of Watchdog\n");
42     }
43
44     /* Initialize delay */
45     TM_DELAY_Init();
46
47     /* Initialize leds on board */
48     TM_DISCO_LedInit();
49
50     /* Initialize button */
51     TM_DISCO_ButtonInit();
52
53     /* Display to user */
54     printf("Program starting..\n");
55
56     /* Set MAC address from unique ID */
57     for (i = 0; i < 6; i++) {
58         /* Set MAC addr */
59         mac_address[i] = TM_ID_GetUnique8(11 - i);
60     }
61
62     /* Initialize ethernet peripheral */
63     /* Set MAC address, set IP address which will be used in case DHCP can't get IP from router */
64     if (TM_ETHERNET_Init(mac_address, ip_address, NULL, NULL) == TM_ETHERNET_Result_Ok) {
```

```
65      /* Successfully initialized */
66      TM_DISCO_LedOn(LED_GREEN);
67    } else {
68      /* Unsuccessfull communication */
69      TM_DISCO_LedOn(LED_RED);
70    }
71
72    /* Reset watchdog */
73    TM_WATCHDOG_Reset();
74
75    while (1) {
76      /* Update ethernet, call this as fast as possible */
77      TM_ETHERNET_Update();
78
79      /* Reset watchdog */
80      TM_WATCHDOG_Reset();
81    }
82 }
83
84 /* Delay 1ms handler */
85 void TM_DELAY_1msHandler(void) {
86    /* Time update for ethernet, 1ms */
87    /* Add 1ms time for ethernet */
88    TM_ETHERNET_TimeUpdate(1);
89 }
90
91 void TM_ETHERNET_DHCPStartCallback(void) {
92    /* Print to user */
93    printf("DHCP has started with assigning IP address\n");
94 }
95
96 void TM_ETHERNET_IPIsSetCallback(uint8_t ip_addr1, uint8_t ip_addr2, uint8_t ip_addr3, uint8_t ip_addr4, uint8_t
97 dhcp) {
98    /* Called when we have valid IP, it might be static or DHCP */
99
100   if (dhcp) {
101     /* IP set with DHCP */
102     printf("IP: %d.%d.%d.%d assigned by DHCP server\n", ip_addr1, ip_addr2, ip_addr3, ip_addr4);
103   } else {
104     /* Static IP */
105     printf("IP: %d.%d.%d.%d; STATIC IP used\n", ip_addr1, ip_addr2, ip_addr3, ip_addr4);
106   }
107
108   /* Print MAC address to user */
109   printf("MAC: %02X-%02X-%02X-%02X-%02X-%02X\n",
110     TM_ETHERNET_GetMACAddr(0),
111     TM_ETHERNET_GetMACAddr(1),
112     TM_ETHERNET_GetMACAddr(2),
113     TM_ETHERNET_GetMACAddr(3),
114     TM_ETHERNET_GetMACAddr(4),
115     TM_ETHERNET_GetMACAddr(5)
116   );
117   /* Print netmask to user */
118   printf("Netmask: %d.%d.%d.%d\n",
119     TM_ETHERNET_GetGateway(0),
120     TM_ETHERNET_GetGateway(1),
121     TM_ETHERNET_GetGateway(2),
```

```
    TM_ETHERNET_GetGateway(3)
  );
  /* Print gateway to user */
  printf("Gateway: %d.%d.%d.%d\n",
    TM_ETHERNET_GetNetmask(0),
    TM_ETHERNET_GetNetmask(1),
    TM_ETHERNET_GetNetmask(2),
    TM_ETHERNET_GetNetmask(3)
  );
  /* Print 100M link status, 1 = 100M, 0 = 10M */
  printf("Link 100M: %d\n", TM_ETHERNET.speed_100m);
  /* Print duplex status: 1 = Full, 0 = Half */
  printf("Full duplex: %d\n", TM_ETHERNET.full_duplex);
}

void TM_ETHERNET_LinkIsDownCallback(void) {
  /* This function will be called when ethernet cable will not be plugged */
  /* It will also be called on initialization if connection is not detected */

  /* Print to user */
  printf("Link is down, do you have connected to your modem/router?\n");
}
void TM_ETHERNET_LinkIsUpCallback(void) {
  /* Cable has been plugged in back, link is detected */
  /* I suggest you that you reboot MCU here */
  /* Do important stuff before */

  /* Print to user */
  printf("Link is up back\n");
}

/* For printf function */
int fputc(int ch, FILE *f) {
  /* Send over usart */
  TM_USART_Putc(USART6, ch);

  /* Return character back */
  return ch;
}
```

All examples are available on my Github account, including Coocox examples. You can download library below.