Getting Started with Ethernet on the STM32 Nucleo

Using STM32CubeMX with Light-Weight IP (LwIP) and System Workbench for STM32 (Eclipse)

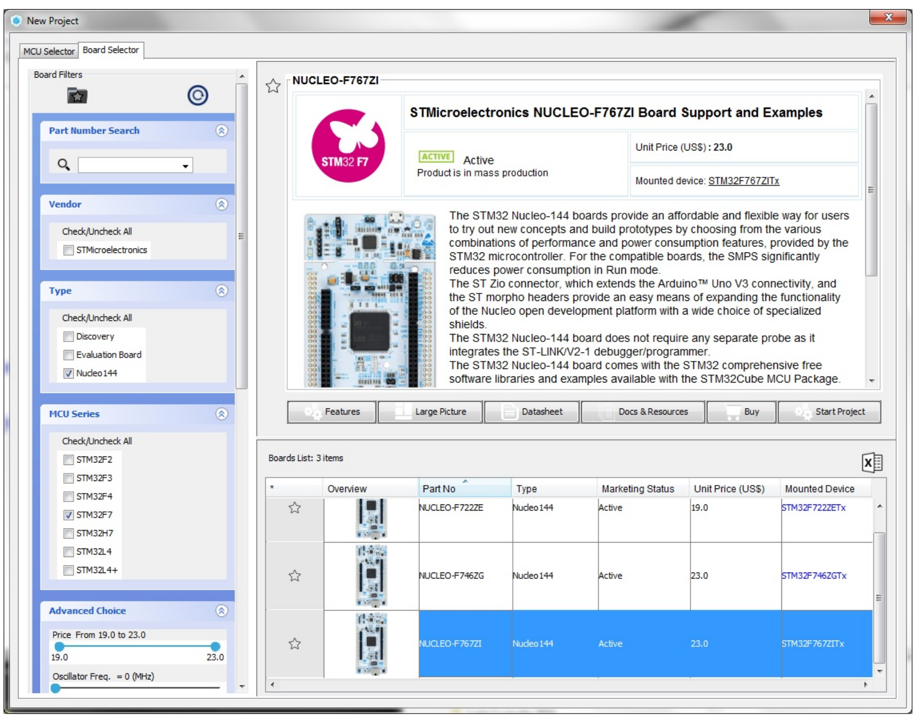
Daniel W Rickey
CancerCare Manitoba
drickey@cancercare.mb.ca
2018-03-07

Introduction

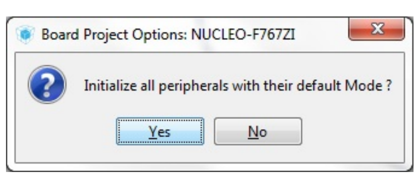
The Nucleo boards produced by ST Microelectronics are wonderfully powerful and cheap. It is fantastic that ST doesn't treat developers as just another revenue stream. On a whim I bought a Nucleo-F767ZI, which has a built-in ethernet connector (not Wifi). I found that setting up LwIP was a fairly involved process, thus I made these notes. I am assuming the reader, like myself, is new to these environments and will find these useful. To avoid complexities, I have not made use of RTOS.

CubeMX

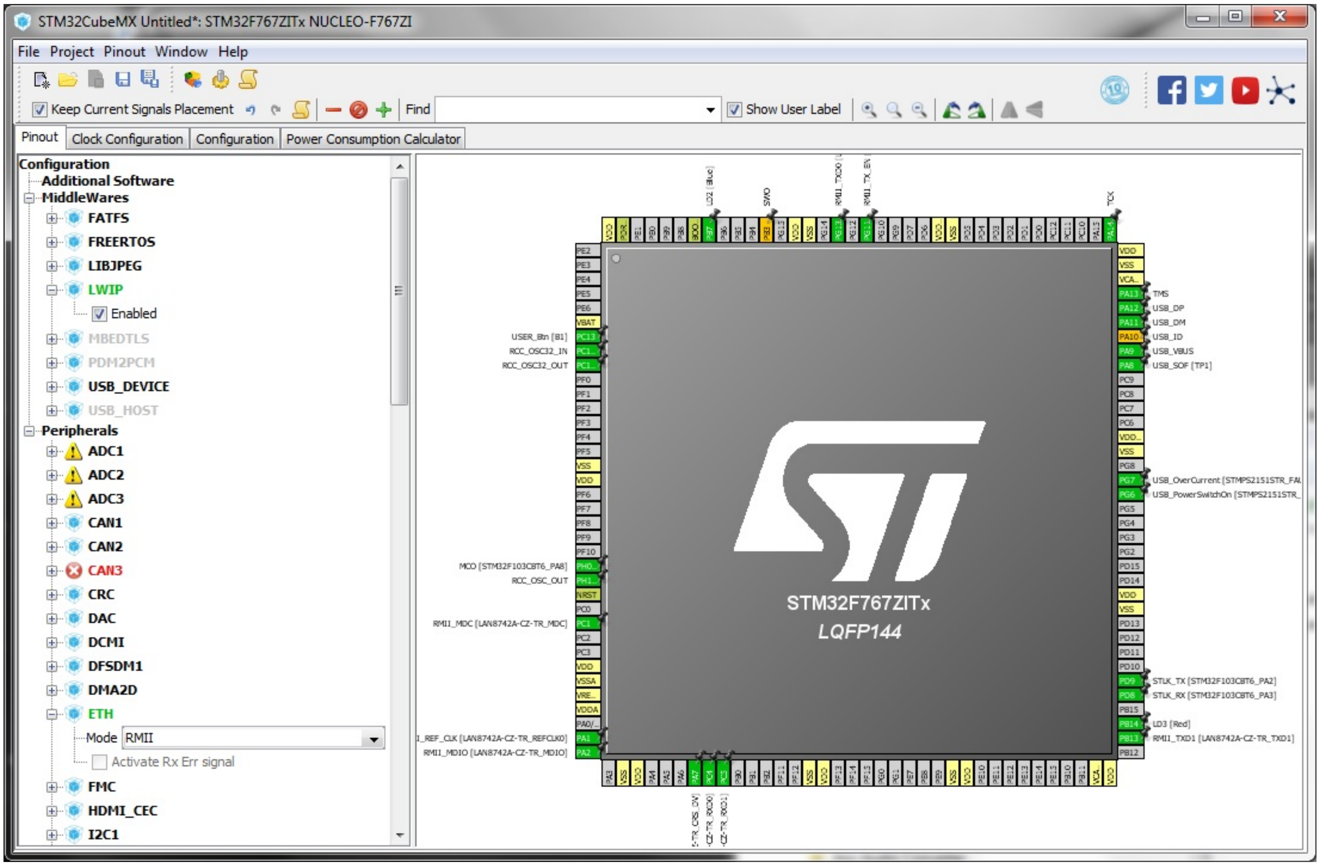
Fortunately CubeMX has improved a great deal since it was first introduced. It is used here to configure the peripherals and LwIP. The first step is to select the board from the list.



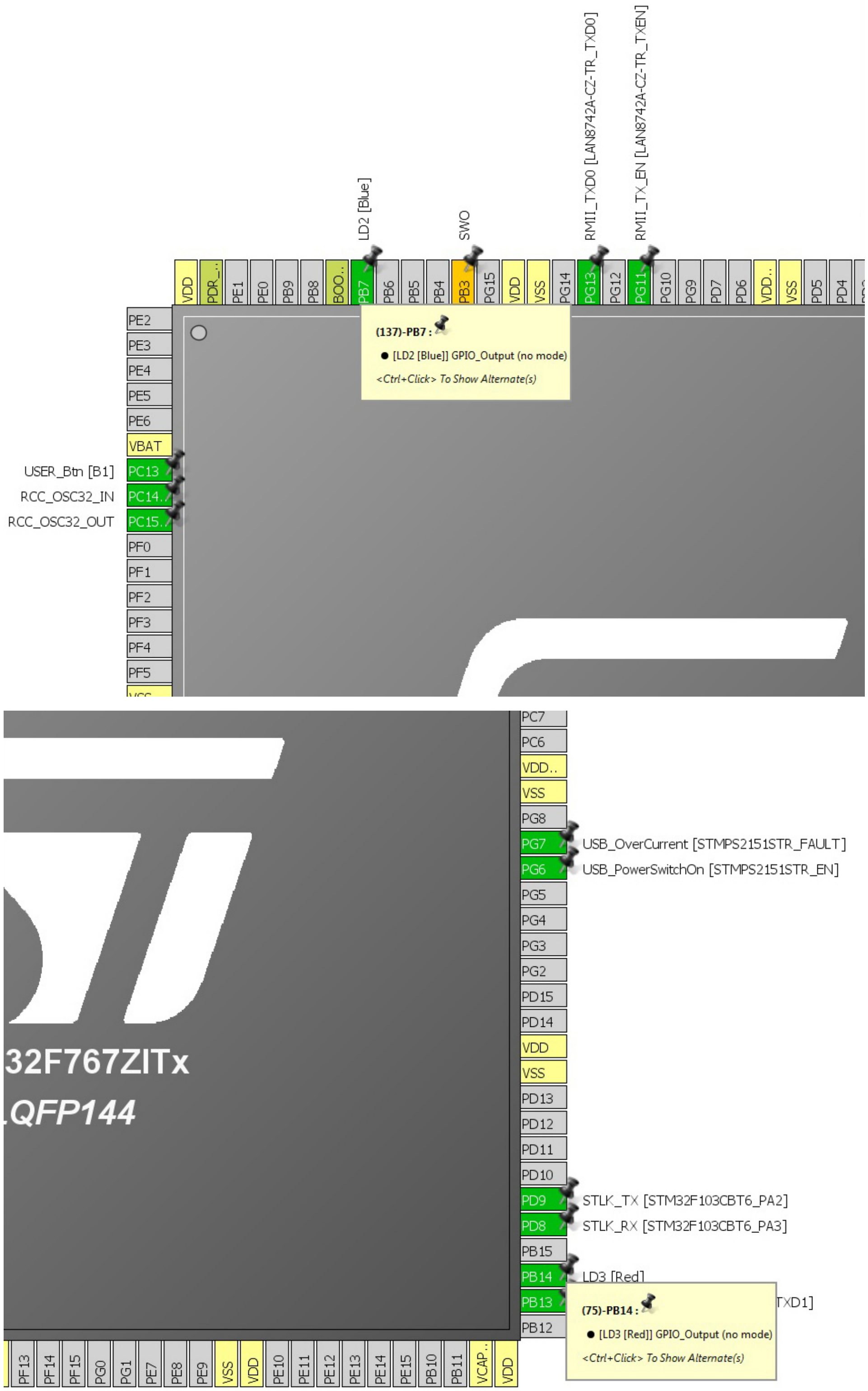
CubeMX will ask if all peripherals should be initialised to their default mode. For the Nucleo boards, I'm not sure this makes much difference as there are not very many peripherals. I selected "yes".



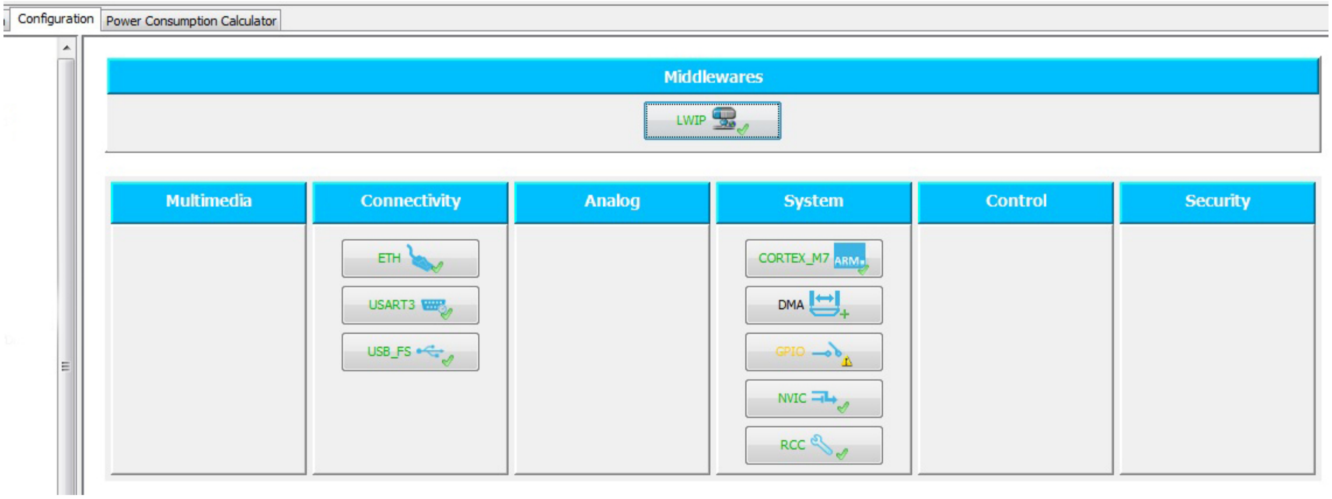
As shown below, enable LWIP, which is listed under “MiddleWares”. Check that the ethernet “Eth” is enabled.



The Nucleo boards contains LEDs that the user can turn on and off. These are used in this example. Note that ST failed here because different Nucleo boards use different names for the same LEDs. So much for code portability. This is a good time to note the names used by your board: here they are “LD2” and “LD3”.

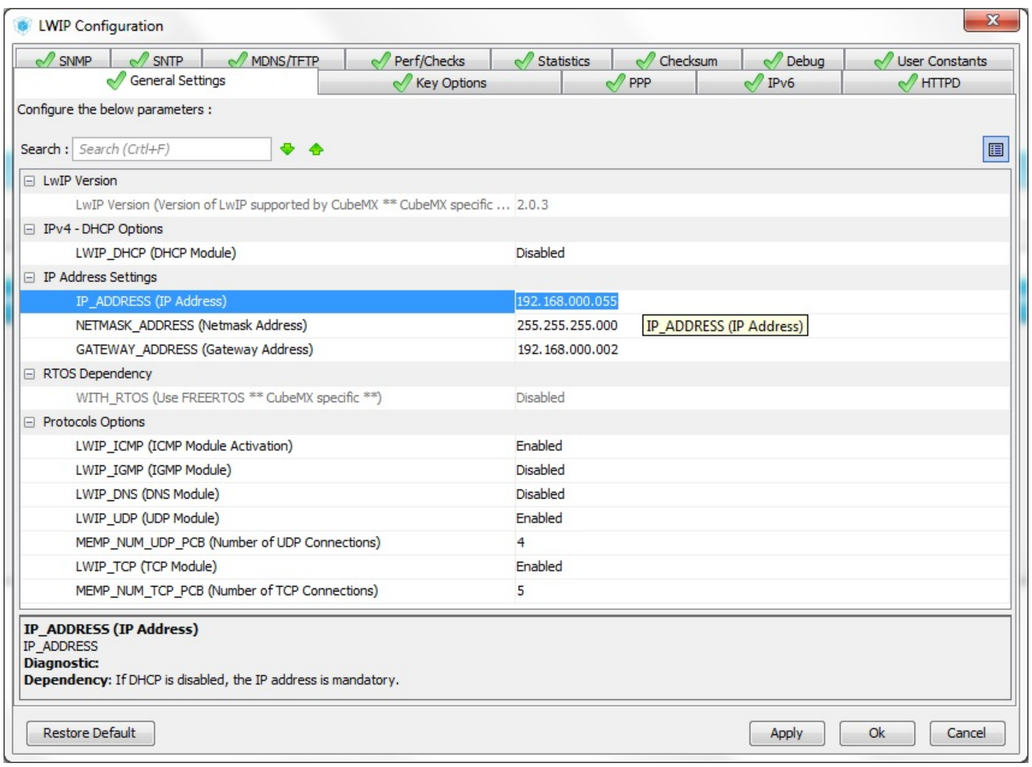


The next step is to configure the various options for LwIP. These are accessed through the "LWIP" button on the "Configuration" tab.



At this point development is a whole lot easier if you hardwire the IP address of the board. That is, disable DHCP (LWIP_DHCP). I plugged my board into the same switch as my development computer so it was easy to reach. Pick an address that's reachable but not already used by another device on your network: you could ping the proposed address to make sure it's free.

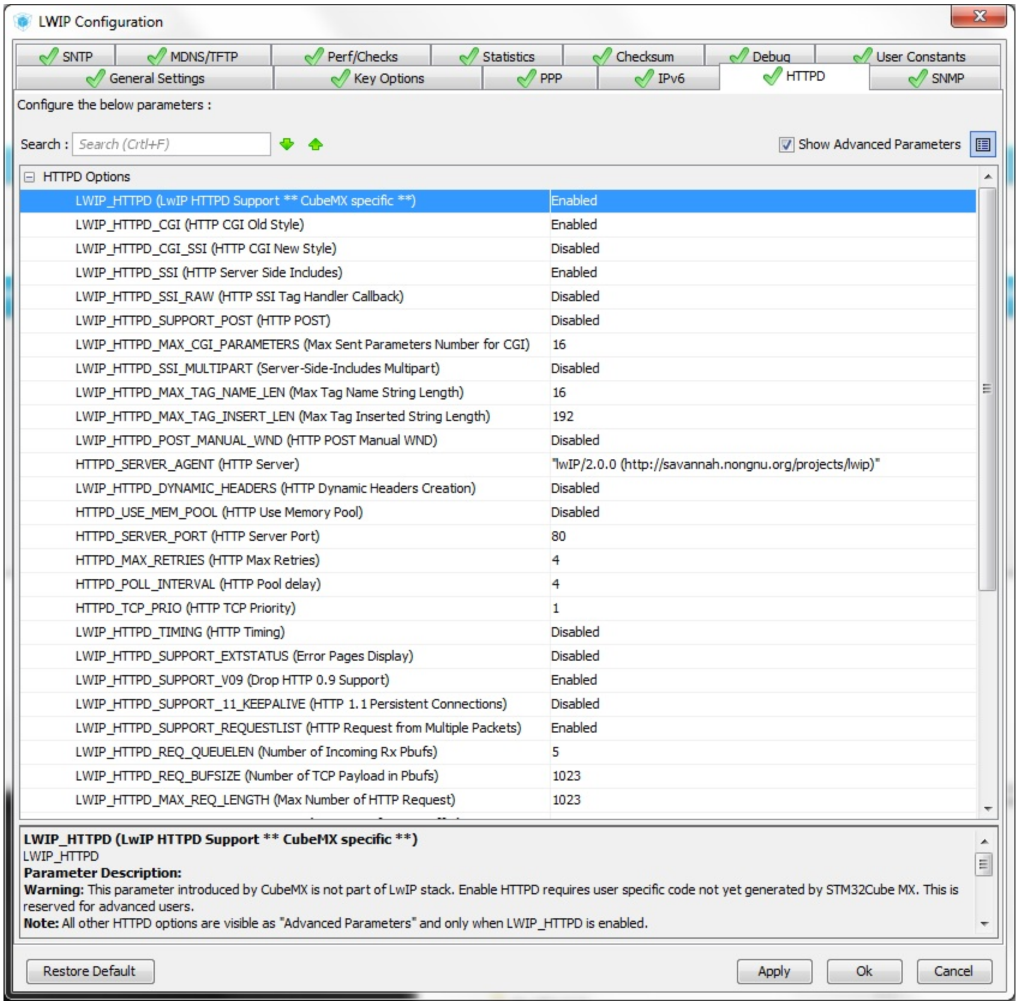
Also note that RTOS is disabled. A simple poling loop will be used instead.



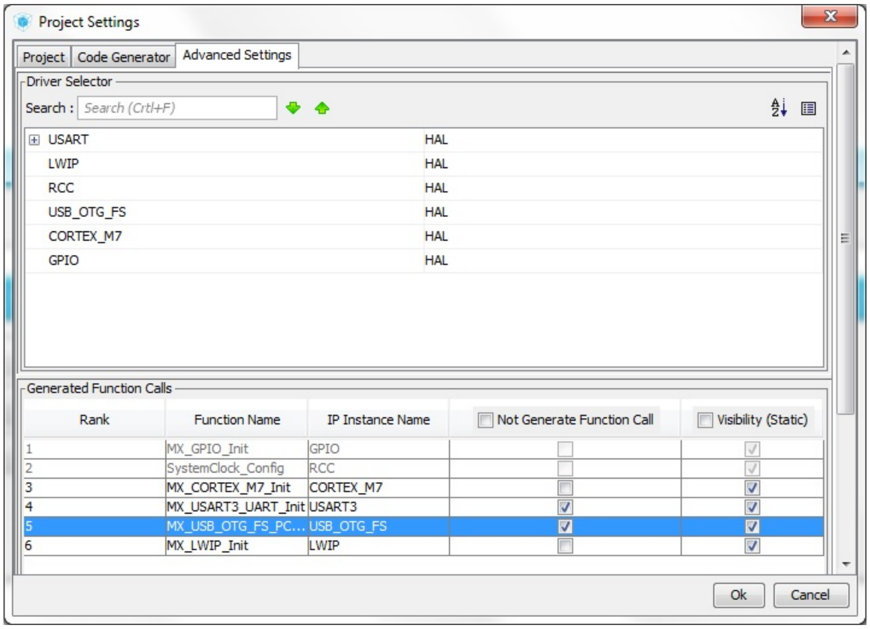
Most of the settings were left at their defaults. However, to enable the webserver, the following options were enabled: LWIP_HTTPD, LWIP_HTTPD_CGI and LWIP_HTTPD_SSI. Because the Nucleo board has bucket loads of memory, the maximum tag length (LWIP_HTTPD_MAX_TAG_NAME_LEN) was increased from 8 to 16 characters.

I found out that it is important to set these options before importing the project into Eclipse. I encountered bizarre behaviour when I used CubeMX to enable an option after the project had been imported.

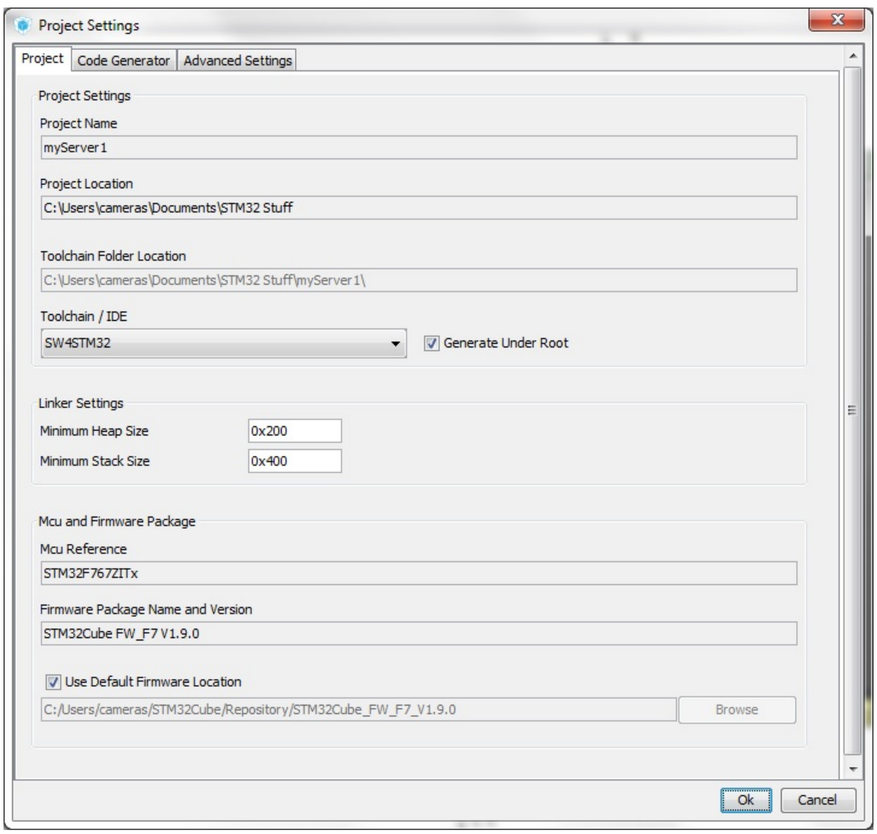
Note that I didn't bother with "HTTP CGI NEW STYLE" as I could not find much in the way of documentation or examples.



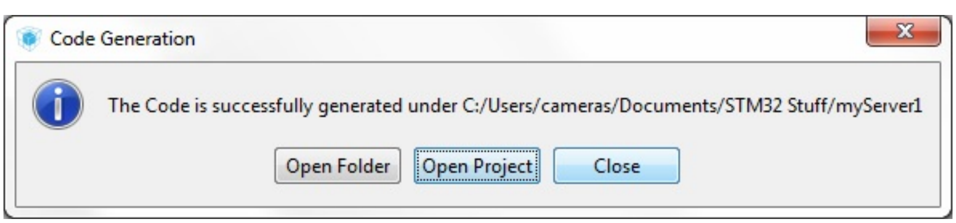
Under "Advanced Settings", it is possible to disable initialisation of the peripherals. Here, I disabled the UART and USB initialisation as they are not needed for this example.



Export the project using the toolchain "SW4STM32" (System Workbench for STM32). I think you want "Generate Under Root" enabled.

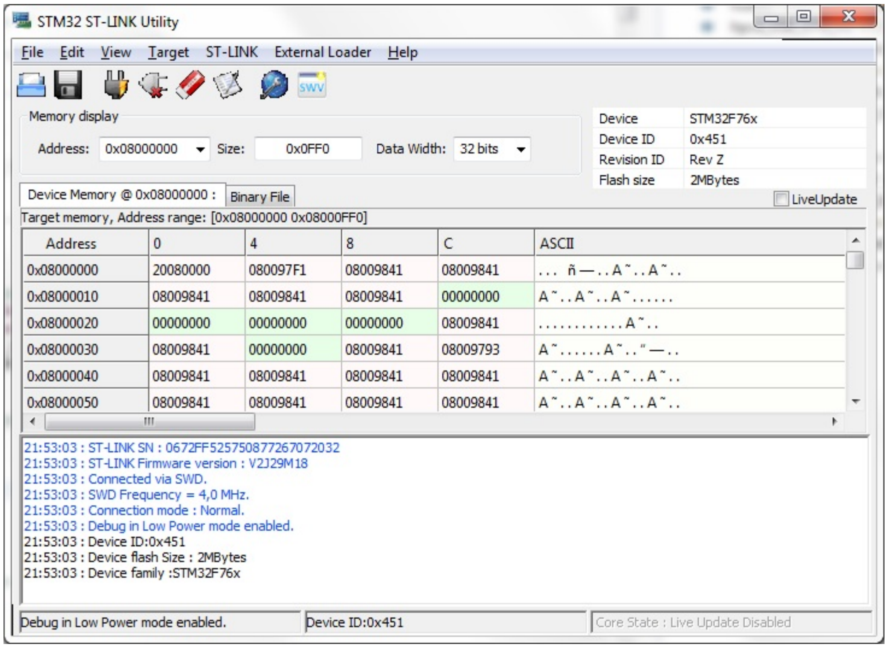


The following dialogue box is presented. I just selected "Close".



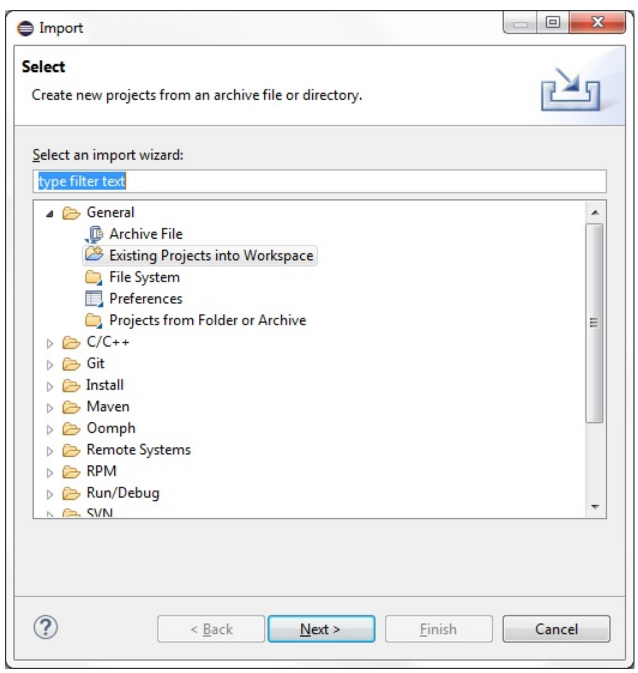
ST-Link

It is a good idea to confirm that the ST-Link debugger has the latest firmware on it. Use the ST-Link utility to do this. This also ensures that the required device drivers have been installed.

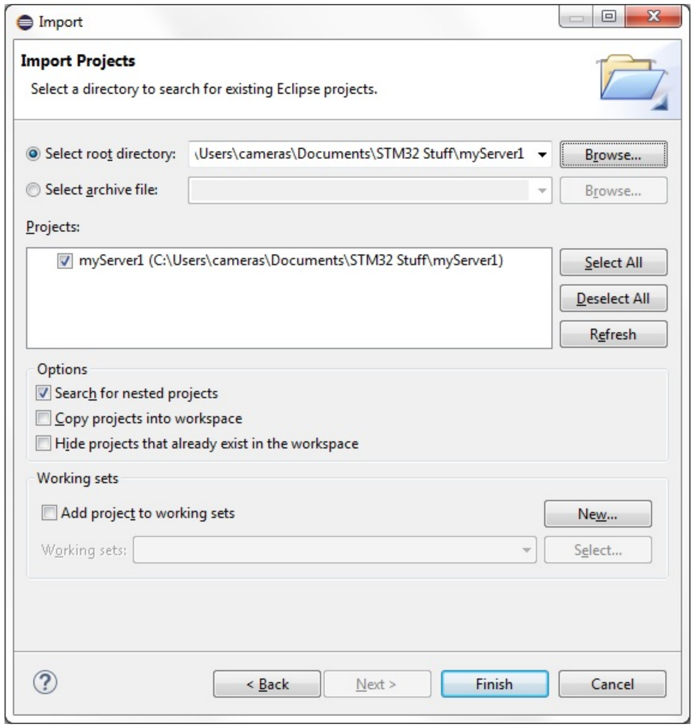


Importing into Eclipse

Importing into Eclipse requires a couple of steps but works well. Import through the dropdown menu file:import as “Existing Projects into Workspace”.

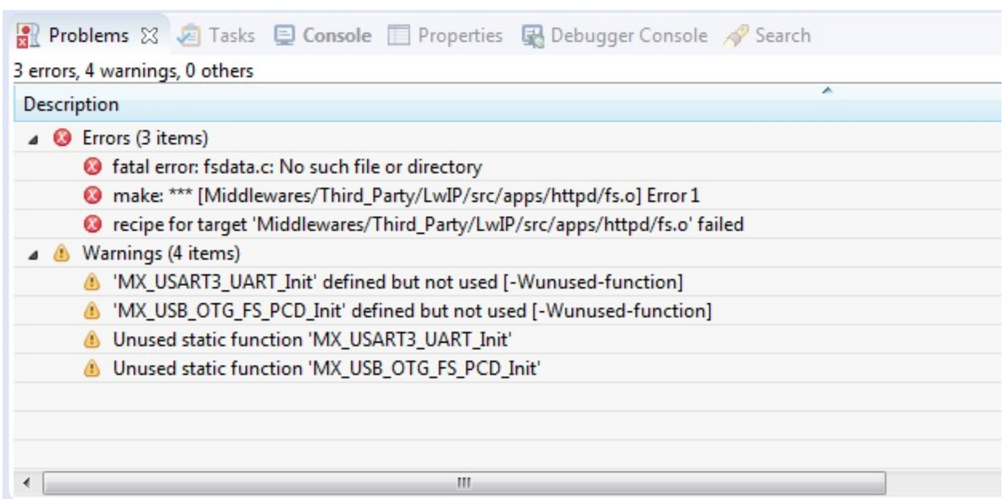


Browse to the folder holding the project. There should only be one project visible, and tell it to finish. If the project imports properly, you should be able to few the various files.



Missing bits

If you try to build the project it will fail miserably as shown in the following figure. The problem is that there is a very important file missing: fsdata.c. This is the file that holds the content of your website and is needed by HTTPD.



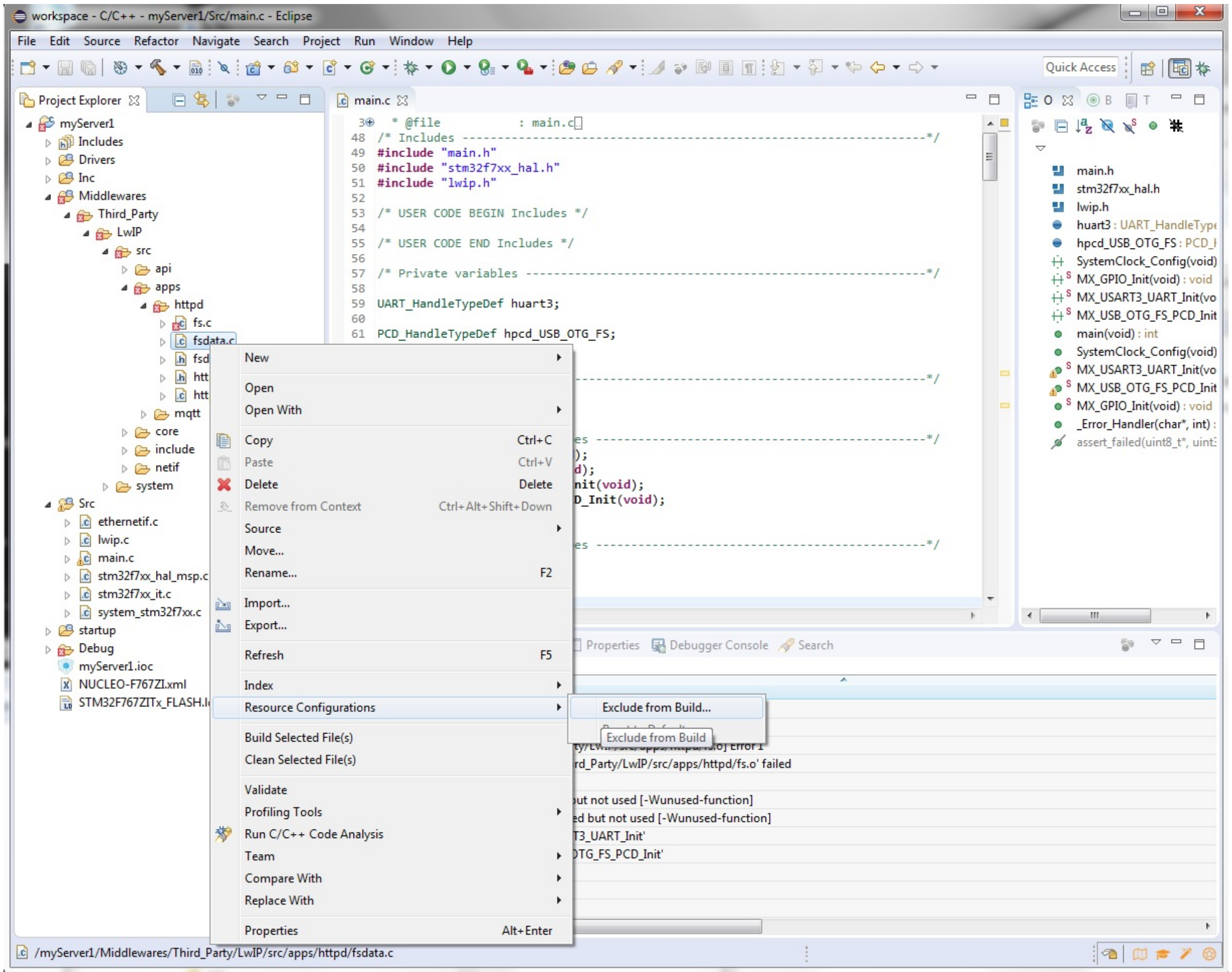
To quickly try out the HTTP server, you can make use of an existing fsdata.c that is included as an example with CubeMX. You may need to search your drive for it although one is located at:

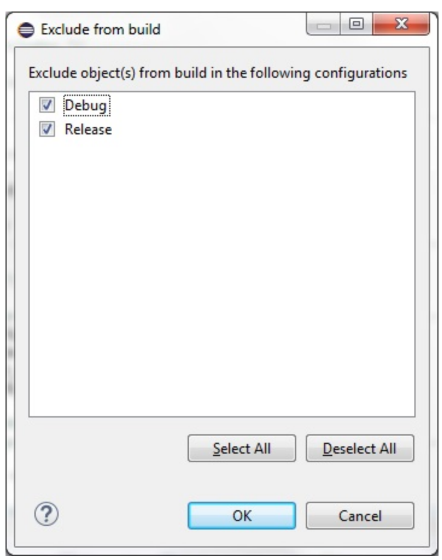
...\STM32Cube\Repository\STM32Cube_FW_F7_V1.9.0\Middlewares\Third_Party\LwIP\src\apps\httpd

Annoyingly LwIP has hard coded the path so you will need to copy fsdata.c to the following folder:

...\myServer1\Middlewares\Third_Party\LwIP\src\apps\httpd

Note that although fsdata.c appears to be a C source file, it must not be compiled because truly bizarre error messages will result. Select fsdata.c in the project explorer and exclude it from the build as shown in the following two figures.



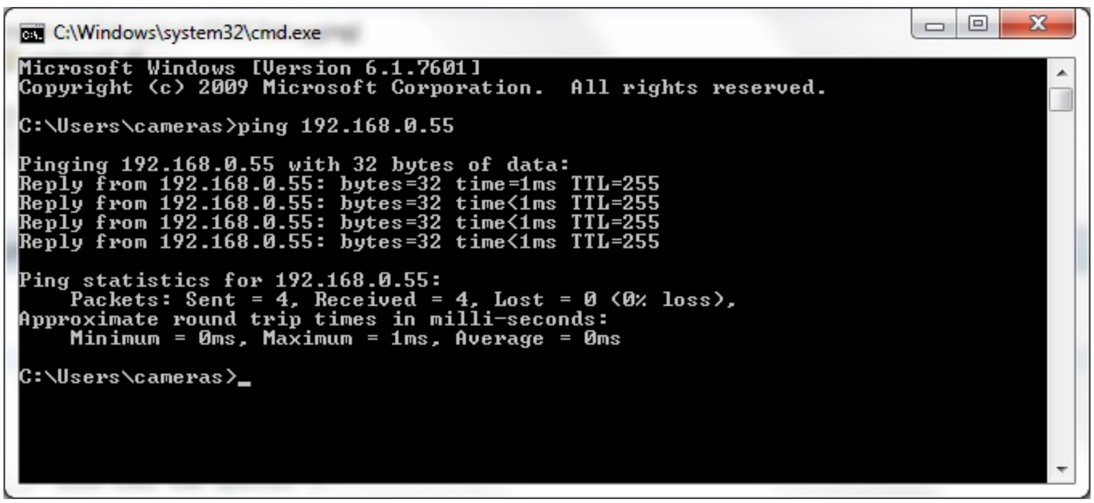


Ping

At this point the code should build but it won't work properly. The function call *MX_LWIP_Process()* needs to be added to the while loop in main.

```
/* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
   {
   //Read a received packet from the Ethernet buffers and Send it to the lwIP stack for handling
     MX_LWIP_Process();
     /* USER CODE END WHILE */
     /* USER CODE BEGIN 3 */
     }
   /* USER CODE END 3 */
```

The code should be able to build. Then you should be able to ping your board as shown in the following figure.



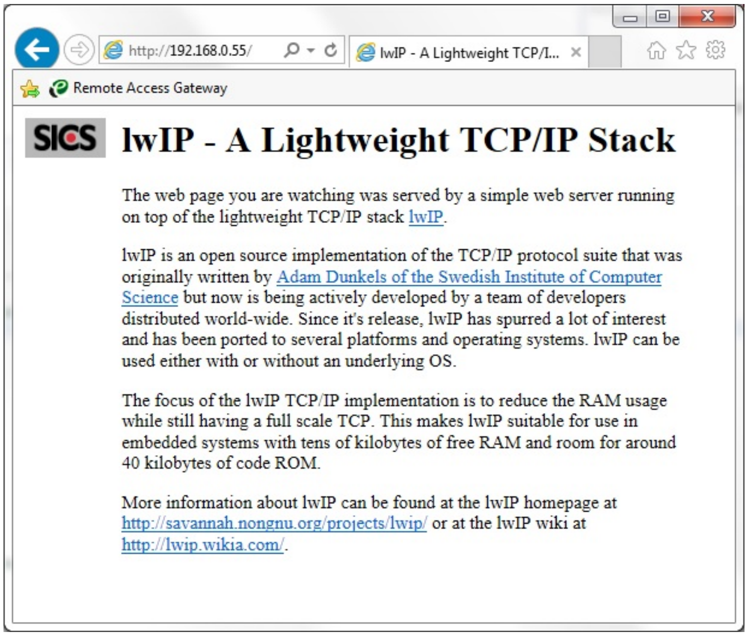
Viewing a webpage

The next step is to start the webserver by adding *httpd_init()* to main. Be sure to add the corresponding include statement for httpd.h.

```
/* USER CODE BEGIN Includes */
#include "lwip/apps/httpd.h"

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

/* USER CODE BEGIN 2 */
//start the web server
httpd_init();
```

```
/* USER CODE END 2 */
```

When the code is running on the Nucleo board, you should be able to access a webpage by typing http://192.168.0.55 into a web browser. At this point you should be able to view the webpage in a browser.



Making a Webpage

You will need to make your own webpage. Because the microcontroller doesn't (usually) have a file system, the webpages are converted to a single file fsdata.c that is included at compile time. To do this use the command line utility htmlgen.exe in the DOS command line or makefsdata in unix. Rather annoyingly a binary for DOS of this utility is not included with LWIP but there is one on the web ...somewhere.

For example, if the html files are contained in the folder "leds", the command is *htmlgen leds -f:fsdata.c.* The folder should contain an index.html file and a 404.html file at a minimum. The 404.html file is helpful if you make a mistake and tell the server to load a non-existent webpage. Otherwise, it would just sit there like a bump and you won't know what is wrong.

Note that you should do a clean rebuild after changing fsdata.c.

Simple CGI handler

Implementing a CGI handler isn't too difficult. Start with a simple web page that has two checkboxes on it. The web page used here is really short.

```
<!DOCTYPE html>
<html><head><title>LED Test</title>


<p>This allows you to control the LEDs: LED1 and LED2.  You have to click on "Send" button
to change the LEDs</p>
```

```
<form method="get" action="/leds.cgi">
<input value="1" name="led" type="checkbox">LED1<br>
<input value="2" name="led" type="checkbox">LED2<br>
<br>
<input value="Send" type="submit"> </form>
</html>
```

The idea behind this webpage is it allows the user to turn on the LEDs on the Nucleo board. So you will need to enable these LEDs with code similar to the following

```
/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
static GPIO_InitTypeDef  GPIO_InitStruct;

/* USER CODE BEGIN 2 */
//setup the blue LED
//note that different Nucleo boards use different names for the same LEDs
GPIO_InitStruct.Pin = LD2_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FAST;
HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

//setup the red LED
GPIO_InitStruct.Pin = LD3_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FAST;
HAL_GPIO_Init(LD3_GPIO_Port, &GPIO_InitStruct);
```

The CGI handler for turning on the LEDs is the following. The function header is defined in httpd.h as type tCGIHandler.

```
/* USER CODE BEGIN 0 */

/****  CGI handler for controlling the LEDs  ****/
// the function pointer for a CGI script handler is defined in httpd.h as tCGIHandler
const char * LedCGIhandler(int iIndex, int iNumParams, char *pcParam[], char *pcValue[])
{
 uint32_t i=0;

 // index of the CGI within the theCGItable array passed to http_set_cgi_handlers
 // Given how this example is structured, this may be a redundant check.
// Here there is only one handler iIndex == 0
 if (iIndex == 0)
   {
   // turn off the LEDs
```

```
    HAL_GPIO_WritePin(LD3_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, GPIO_PIN_RESET);

    // Check the cgi parameters, e.g., GET /leds.cgi?led=1&led=2
    for (i=0; i<iNumParams; i++)
     {
     //if pcParmeter contains "led", then one of the LED check boxes has been set on
     if (strcmp(pcParam[i], "led") == 0)
      {
      //see if checkbox for LED 1 has been set
      if(strcmp(pcValue[i], "1") == 0)
       {
       // switch led 1 ON if 1
       HAL_GPIO_WritePin(LD3_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
       }

       //see if checkbox for LED 2 has been set
      else if(strcmp(pcValue[i], "2") == 0)
       {
       // switch led 2 ON if 2
       HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, GPIO_PIN_SET);
       }
      } //if
     } //for
    } //if

  //uniform resource identifier to send after CGI call, i.e., path and filename of the
 response
  return "/index.html";
 } //LedCGIhandler
```

We need to tell the HTTPD code to use this handler. To do this, the LedCGIhandler is added to a list. First a structure is created that shows that leds.cgi corresponds to the LedCGIhandler.

```
/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
// prototype CGI handler for the LED control
const char * LedCGIhandler(int iIndex, int iNumParams, char *pcParam[], char *pcValue[]);

// this structure contains the name of the LED CGI and corresponding handler for the LEDs
const tCGI LedCGI={"/leds.cgi", LedCGIhandler};

//table of the CGI names and handlers
tCGI theCGItable[1];
```

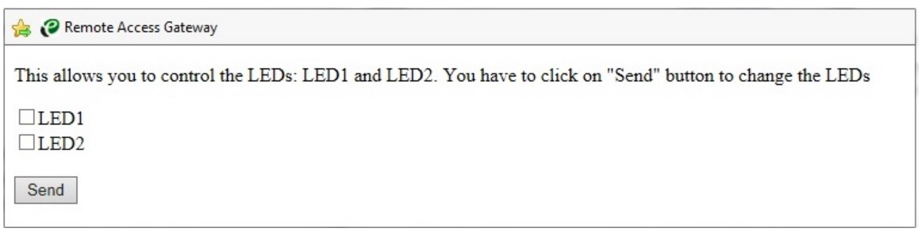
Also add the following function to /* USER CODE BEGIN 0 */

```
// Initialize the CGI handlers
void myCGIinit(void)
{
//add LED control CGI to the table
theCGItable[0] = LedCGI;

//give the table to the HTTP server
http_set_cgi_handlers(theCGItable, 1);
} //myCGIinit
```

Then in the main code, call myCGIinit.

```
//start the web server
httpd_init();
//initialise the CGI handlers
myCGIinit();
```

The code should compile and run. With some luck the you will be able to set the LEDs from the webpage.



Server side includes (SSI)

Server side includes let the microcontroller generate text that is displayed on the webpage. For example, this could be used to display the output of an analogue-to-digital converter . To do this, tags located in the HTML code are replaced by text when the webpage is served to the client. Note that the SSI function is called each time the HTTPD server detects a tag of the form <!--#name--> in a .shtml, .ssi or .shtm file. It won't work if the file has a .html extension. Start by adding a couple of lines to the previous webpage as shown below. Note that the filename must be index.shtml, and you will need to adjust this file name in the LedCGIhandler function. Use the command line utility to convert this to fsdata.c

```
<!DOCTYPE html>
<html><head><title>LED Test</title>

<body>
<p>This allows you to control the LEDs: LED1 and LED2.  You have to click on "Send" button
to change the LEDs</p>

<form method="get" action="/leds.cgi">
<input value="1" name="led" type="checkbox">LED1<br>
<input value="2" name="led" type="checkbox">LED2<br>
<br>
```

```
<p>text for tag1: <!--#tag1--></p>
<p>text for tag2: <!--#tag2--></p>

<input value="Send" type="submit"> </form>
</body></html>
```

The HTTPD functions need a list of the tags contained in the HTML code.

```
//array of tags for the SSI handler
//these are the tags <!--#tag1--> contained in the shtml file
#define numSSItags 2
char const *theSSItags[numSSItags] = {"tag1","tag2"};
```

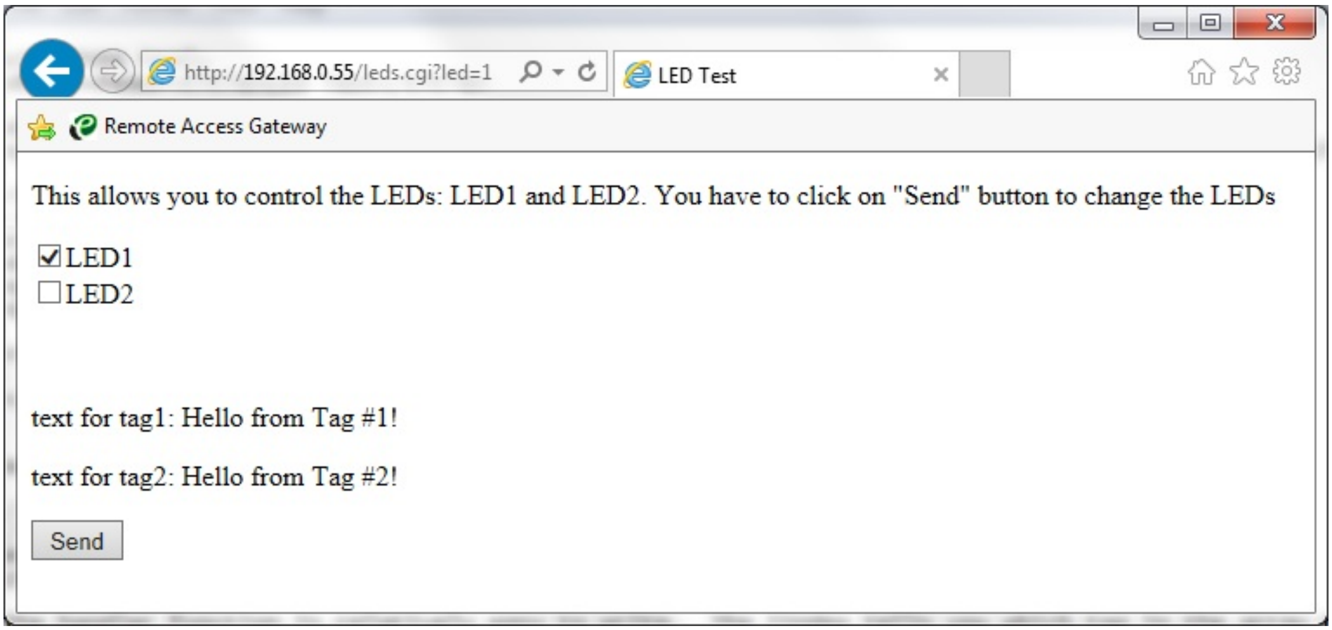
The handler function is relatively easy to write. The iIndex tells you which tag in the array to take care of. Place the text to be displayed into pcInsert and return the number of characters inserted.

```
/**** SSI handler ****/
// This function is called each time the HTTPD server detects a tag of the form
// <!--#name--> in a .shtml, .ssi or .shtm file
// It won't work if the file has a .html extension.
u16_t mySSIHandler(int iIndex, char *pcInsert, int iInsertLen)
{
 // see which tag in the array theSSItags to handle
 if (iIndex == 0)  //is “tag1”
   {
   char myStr1[] = "Hello from Tag #1!";  //string to be displayed on web page

   //copy the string to be displayed to pcInsert
   strcpy(pcInsert, myStr1);

   //return number of characters that need to be inserted in html
   return strlen(myStr1);
   }
   else if (iIndex == 1)  //is “tag2”
   {
   char myStr2[] = "Hello from Tag #2!";  //string to be displayed on web page

   //copy string to be displayed
   strcpy(pcInsert, myStr2);

   //return number of characters that need to be inserted in html
   return strlen(myStr2);
   }
   return 0;
} //mySSIHandler
```

Call the following function from main to initialise the SSI handler.

```
/**** Initialize SSI handlers ****/
void mySSIinit(void)
{
  //configure SSI handler function
  //theSSItags is an array of SSI tag strings to search for in SSI-enabled files
  http_set_ssi_handler(mySSIHandler, (char const **)theSSItags, numSSItags);
} //mySSIinit

/* USER CODE END 0 */
```

Then you should be able to compile and run the code. With some luck the text will be displayed on the webpage as shown below.



Sending email

The LWIP code does not include an email client. However, there is an SMTP client available as a separate download. See contrib at download.savannah.nongnu.org/releases/lwip/. There are two files smtp.c and smtp.h. Place these in the Src and Inc folders in your project, respectively.

Using the email routines is really easy. Simply set up the server address and authentication. Then sending an email is a single function call.

```
//this function is called when SMTP wants to tell us something
void mySMTPresult(void *arg, u8_t smtp_result, u16_t srv_err, err_t err)
{
printf("mail (%p) sent with results: 0x%02x, 0x%04x, 0x%08x\n", arg, smtp_result, srv_err,
err);
} //mySMTPresult

/**** send an email using SMTP ****/
static void sendAnEmail(void)
```

```
{
#define emailFrom       "yourEmail@yourISP.ca"
#define emailTo         "drickey@cancercare.mb.ca"

#define emailSubject    "annoying"
#define emailMessage    "this is an annoying message"

int * some_argument = 0;
//IP address or DNS name for your SMTP connection
//smtp_set_server_addr("mail.yourISP.ca"); //if using DNS
smtp_set_server_addr("xxx.xxx.128.128");    //if using IP address

// set both username and password as NULL if no authentication needed
smtp_set_auth("yourUserID", "yourPassowrd");
smtp_send_mail(emailFrom,    textTo,    emailSubject,    emailMessage,    mySMTPresult,
some_argument);
} //sendAnEmail
```

I believe the password is sent unencrypted, so be careful with your selection of email host. You can use this to send a text message to a cell phone via email. This could be used to let you know that your garage door is open or for annoying your friends.

Problems

1) If you encounter bizarre behaviour where a #define seems to change values, i.e., is “1” in one location and “0” in another, then try exporting the project from CubeMX under a new name. This will require you to reimport it into Eclipse and add the example code to main.c.

2) When using server side includes, the file must have a .shtml extension otherwise the tags won’t work.

3) Be sure to include a 404.html file in case you screw up and ask the server to show a non-existent webpage.

4) If the webpage displays but is really slow, try rebooting the whole works (computer and microcontroller).

5) Make sure fsdata.c is in the correct folder but is excluded from the build. Trying to compile fsdata.c will generate very odd errors.

6) Always do a clean rebuild after changing fsdata.c.

-- end --