Explanation

HARDWARE

Development Configuration

I used the EasyMx PRO v7 for STM32 ARM board with a STM32F407VG board plugged into the microcontroller socket. Three USB cables connect the programming interface, the USB interface, and the USB UART B interface, to the application computer. The programming interface is required to load the code compiled with MikroC PRO for ARM into the STM32F407 flash. The USB interface is used to print out diagnostic messages and returned messages from the WiFi ESP module. Unfortunately, the HID driver implemented by MikroE is limited to 64 bytes per line. The optional serial interface (USB UART B) can be used for longer messages. All other switch settings are described in the module My Code, Main.c.

WiFi ESP Click Board

This click board carries the ESP-WROOM-02 module that integrates ESP8266EX. Its receiver is very sensitive judging from the number of access points it detects in my neighbourhood. It uses a serial interface to attach to a microcontroller and incorporates a custom version of the AT command set to perform specific tasks. The click board is plugged in to slot 1. The Documents folder contains details of the AT commands with examples.

AT Commands

Looking at the MikroE example, you would think that it would be a simple matter to implement a device with WiFi. As you shall see, it takes a lot of code to interact with WiFi ESP and I have only scratched the surface in this example.

MikroC PRO for ARM v6.1.0 IDE Settings

Edit Project Settings for Main Project

For this project, the MCU Name should be STM32F407VG and MCU Clock Frequency [MHz] should be set to 120.000000. You should save the file Vals_STM32F407VG_25MHz_External_to_120MHz_from_PLL.cfgsch from the **Setups** folder

to your MikroC PRO for ARM, **Schemes** folder. Then, in Edit Project load the scheme above. It should set the following:
HIS oscillator OFF
HSE oscillator ON
HSE oscillator not bypassed
Clock detector OFF
PLL ON
PLLM=15
PLLN=144
PLLP=2
HSE oscillator clock selected as PLL and PLL2S clock entry
PLLQ=5
PLL selected as system clock
SYSCLK not divided
HCLK divided by 2
Etc.

Edit Project Settings for TFT Project

For this project, the MCU Name should be STM32F407VG and MCU Clock Frequency [MHz] should be set to 120.000000. You should save the file EasyMx_PRO_v7_for_STM32F407VG_ARM_9A.XML from the **Setups** folder to your MikroC PRO for ARM, **Board Defs** folder. Then, in Edit Project select the VTFT settings tab. In the Hardware Patterns box select the EasyMx_PRO_v7_for STMF407VG_ARM_9A.XML file. In the General project settings tab, load the scheme Vals_STM32F407VG_25MHz_External_to_120MFz_from_PLL.cfgsch file.
The parameters should be the same as in the main project above.

You can see from this exercise how much effort it takes to set up your environment for MikroC.

Explanation

SOFTWARE

FreeRTOS Source Folder

This folder contains a copy of FreeRTOS. I find it convenient to have it included in the same package when distributing my code. The FreeRTOSConfig.h file is in the My Code folder. With so many tasks, I had to double the configTOTAL_HEAP_SIZE to 16384 to accommodate them all.

TFT Project Folder

This is a separate project where you can develop your TFT screens. Leave all TFT files generated in this folder. You can come back and modify your screens without affecting the WiFi code.

My Code Folder

This folded contains all files for the main project which includes the driver for the WiFi ESP click board. The code is self contained in several files. Keeping the code separated in this way makes it easy to debug or change.

Buffers.c and Buffers.h Files
These files contain the code for implementing a bunch of buffers that can be used for many purposes. I have used this code in many of my posts, so it should be familiar by now. You can change the number of buffers and the size of each buffer in the #define statements of Buffers.h file.

HTTP.c and HTTP.h Files
These files contain the html code for a web page I borrowed from MikroE Weather Station project. It demonstrates how a long text file must be broken up into segments to be transmitted by the WiFi ESP module. It resides in code memory and must be accessed in a slightly different manner than code in RAM. Obviously, you can substitute your own web page html code here.

Explanation

init_task.c and init_task.h Files
These files contain the task initialization code for all other tasks. Some unused tasks are commented out. The initialization task deletes itself when done initializing all the other tasks. This code is taken directly from a MikroE example.

Main.c and Main.h Files
This is where the code starts and is a good place to include some notes as to the hardware configuration and other setups necessary.

MyTimers.c and MyTimers.h Files
I have included these files, but they are not used in this project.

RTC.c and RTC.h Files
This is code for the real-time clock inside the microcontroller.

SD_driver.c and SD_driver.h Files
These files are not used in the project.

TFT_main.c File
This is a copy of TFT_main.c file in the TFT project but modified for FreeRTOS.

USART2_driver.c and USART2_driver.h Files
This is the driver for the connector labelled USB UART B on the EasyMx board.

USART3_driver.c and USART3_driver.h Files
This is the driver for the interface between the micro and the WiFi ESP board. It operates in 2 modes. The “normal” mode is used for text characters where special ASCII characters are used for control. e.g. – return and new line. Transparent mode is used when the full 256-character set (binary) is being transferred.

USB_driver3.c, USB_driver3.h, and USBdsc.c Files
These files implement the native USB driver for the micro. It is primarily used to display diagnostic messages during debug.

WiFi_ESP_Driver.c and WiFi_ESP_Driver.h Files
These files implement the routines that work with WiFi ESP module according to the AT command set. Initially, when you get your WiFi ESP click board, the flash may be empty. It

should be programmed with some defaults. When you load the code, it will check if the flash has been programmed and display a message if the flash is blank. Once the flash has been set up, WiFi ESP will attempt to connect automatically to your router using the SSID and Password you provided in the code.

Common.h File
In this file you can turn off diagnostic messages from printing on USB by setting #define WIFI_DEBUG to 0.

NOTE: In MikroC Tools, Options, Output Tab, Output Settings – make sure that you set the check mark in the Compiler box labelled Case sensitive. This is required by FreeRTOS. When you try to compile some other code, you may get errors showing up that were not there previously. This is due to wrong case being used for functions or variables.

Documents Folder

Here you will find other useful information as well as this document.

Utilities Folder

This folder contains some useful programs for debugging your code.

UsbUtility.exe and HID_Library__.dll Program
This program displays messages using the native USB interface. You can send a command with a single alpha character to invoke services. See USB_driver.c for the services supported. These can be easily changed to suit your needs.

SerialUtility.exe Program
This program is similar to USBUtility.exe but works with the USB UART B interface. After powering up the EasyMx board and loading the code into it, start the SerialUtility program. The default COM port is COM1, but your computer may assign a different COM port to that interface. The SerialUtility program checks only once on start-up for COM ports. Start the program without the cable plugged in and then with the cable plugged in to see the difference.

UdpTcpIpc.exe Program
You can use this program to test TUDP and TCP code functions.

Explanation

Setups Folder

Here you will find a picture of the Paths for the project. Also, the 2 files mentioned previously are located here. The EasyMx_PRO_v7_for_STM32F407VG_ARM_9A.XML file should be copied from the **Setups** folder to your MikroC PRO for ARM, **Board Defs** folder. The other file, Vals_STM32F407VG_25MHz_External_to_120MHz_from_PLL.cfgsch should be copied from the **Setups** folder to your MikroC PRO for ARM, **Schemes** folder.

If you copy these files first, starting ether the TFT project or the main project should pick up these files and set up the parameters correctly. If not, follow the manual procedure.