TULU-C-IDE for Microcontroller Projects

ESP8266__RTOS__SDK Custom project setup with Localvimrc plugin, Vim-Clang plugin, and Syntastic plugin.

Objectives:

- Loading of Vimrc from a local directory
- Autocompletion
- Code Checking with Syntastic

Localvimrc is undeniably indispensable in extraordinary circumstances where cross-compilers (e.g., GNU-ARM, GNU-AVR, ESP-IDF, Extensa-esp32-elf, PlatformIO, Arduino-CLI) are involved. System-wide Vim configuration can lead to countless problems in a few use-case scenarios because different compilers with similar header/include files can conflict. Use the folder '**LocalVimrc_project_sample**' as a base for your custom projects targetting compilers for different CPU/MCU architectures. Things won't go out of hand as long as the compiler for the target processor is GCC-based and Clang has support for that target.

Espressif ESP8266 (NodeMCU) SDK Setup Guide on Windows 10

We will be using the native SDK provided by Espressif Systems. C is the recommended programming language. A custom version of the MSYS2 terminal is needed to build, upload codes to the microcontroller, and monitor the output through the UART channel.

Directory Structure and Usage:

A brief overview of the steps to follow for programming an ESP8266 compatible development board:

Requirements:

1. MSYS2 package provided by Espressif Systems (includes a Python2.7 based UART monitor and a Python2.7 based custom build-tool).
2. Toolchain and Compiler for ESP8266 MCUs.
3. esp-open-rtos.

4. LLVM Clang (for that, I would suggest the regular MSYS2 with LLVM-Clang and GCC added to the system path.
5. TULU-C-IDE with a custom '.lvimrc'.
6. A Development Board such as ESP8266 NodeMCU.

Before we start, let's have a look at some core features of ESP8266 without being too meticulous. We will not discuss the detailed specifications.

Specifications:

1. ESP8266 is a 32-bit Xtensa Tensilica L106 32-bit RISC processor of a modified Harvard Architecture family.
2. Clock speed: 80MHz.
3. Memory: 32 KiB instruction, 80 KiB user data. That is:

- 32 KiB instruction RAM
- 32 KiB instruction cache RAM
- 80 KiB user-data RAM
- 16 KiB ETS system-data RAM
- External QSPI flash: up to 16 MiB is supported (512 KiB to 4 MiB typically included). Most clones come with 2MB External Flash chips.

4. 17 GPIO pins.
5. Buil-in WiFi (BlueTooth not available).
6. Software implemented I2C, provided via a library in the ESP8266_RTOS_SDK.
7. Serial Peripheral Interface (SPI).
8. I2S interfaces with DMA (sharing pins with GPIO).
9. UART pins.
10. One 10-bit ADC pin.
11. Micro-USB connector via CH340G USB-to-TTL converter IC. Development boards (such as NodeMCU) come with an onboard USB (Android phone micro-USB type connector), powered by an additional USB to Serial converter IC (CH340G on most knockoff boards).

> ESP8266 doesn't have any native USB or I2C interface, although software I2C has been implemented.

Steps:

Download MSYS2 for ESP8266 'esp32_win32_msys2_environment_and_toolchain-20181001.zip' from https://dl.espressif.com/dl/esp32_win32_msys2_environment_and_toolchain-20181001.zip

Instructions are available at https://docs.espressif.com/projects/esp8266-rtos-sdk/en/latest/get-started/windows-setup.html

Download the Toolchain and the SDK: https://github.com/espressif/ESP8266_RTOS_SDK

Toolchain:

Delete the old 'xtensa-esp32-elf' dir in `/msys32/opt`.

Extract 'xtensa-lx106-elf-gcc8_4_0-esp-2020r3-win32.zip' to `/msys32/opt` directory.

Rename the folder 'xtensa-lx106-elf' to 'xtensa-esp32-elf'. Now the path to the compiler should be `/msys32/opt/xtensa-esp32-elf`.

SDK:

Open 'mingw32.exe'.

In the MSYS2 (Espressif provided) Terminal type:

```
mkdir -p ~/esp
cd ~/esp
git clone --recursive https://github.com/espressif/ESP8266_RTOS_SDK.git

echo 'export IDF_PATH=~/esp/ESP8266_RTOS_SDK' >> ~/.profile
echo 'export SDK_PATH=~/esp/ESP8266_RTOS_SDK' >> ~/.profile
echo 'export BIN_PATH=~/esp/ESP8266_RTOS_SDK/bin' >> ~/.profile
```

Edit

`~/.bashrc`

with Vim,

```
vim ~/.bashrc
```

or,

```
notepad ~/.bashrc
```

Add the following lines:

```
export IDF_PATH=~/esp/ESP8266_RTOS_SDK
export SDK_PATH=~/esp/ESP8266_RTOS_SDK
export BIN_PATH=~/esp/ESP8266_RTOS_SDK/bin
```

Close your text editor. In the MSYS2 Terminal, type:

```
cp ~/esp/ESP8266_RTOS_SDK/requirements.txt /
```

Check the version number of Python:

```
python --version
```

The output will be somewhat like this:

```
==>> Python 2.7.15
```

Now, type the commands below:

```
python2.7 -m pip install --user -r $IDF_PATH/requirements.txt
python -m pip install esptool
python -m pip install setuptools
rm /requirements.txt
```

Clang support: Since the version of MSYS2 supplied by Espressif is way too old, LLVM-Clang cannot be downloaded for such an archaic version. Use your regular MSYS2 installation's Clang by adding the path to Clang executable to the system path.

Close MSYS (mingw32.exe) and Exit.

Relaunch MSYS (mingw32.exe).

Copy their 'get-started/hello_world' program to somewhere else. I copied it to ~/esp. It could be /n/codes/esp/ or /d/code_test/esp_projs etc., but make sure you copy the contents of your 'LocalVimrc_project_sample' into that location as well.

```
cp -r $IDF_PATH/examples/get-started/hello_world ~/esp
```

Build the First Project:

```
cd ~/esp/hello_world
make menuconfig
```

'Serial flasher config' requires a change depending on the COM port connected to the development board. Do not change any other parameter in the menuconfig utility wizard. For other projects, you may need to change other parameters.

'Serial flasher config' -> 'Default serial port'

```
<ARROW> <ARROW> <ARROW> <ARROW> <ARROW> <ARROW> <TAB> <TAB> <TAB> <TAB> <TAB> <TAB>
<ENTER>

< Save > then -> < Exit >
```

```
make
```

Or, upload to the ESP8266 board:

```
make flash
```

Monitor the output:

```
make monitor
```

Done.

Erase flash:

```
make erase_flash
```

Some Make commands (to avoid fiddling with the menuconfig utility over and again):

```
make monitor ESPPORT=COM5
```

```
make flash ESPBAUD=9600
```

```
make monitor MONITORBAUD=9600
```

References:

https://docs.espressif.com/projects/esp8266-rtos-sdk/en/latest/get-started/index.html#install-the-required-python-packages

https://github.com/espressif/ESP8266_RTOS_SDK

https://www.freecodecamp.org/news/how-to-get-started-with-freertos-and-esp8266-7a16035ddd71/

https://www.youtube.com/watch?v=pWo-ErpVZC4

https://github.com/esp8266/esp8266-wiki

https://github.com/pfalcon/esp-open-sdk

Creating a Bash Alias: https://davidwalsh.name/alias-bash

`notepad ~/.bash_profile`

Relaunch MSYS2

Create a Start Menu Shortcut:

Go to Windows Explorer Address Bar:

%programdata%\Microsoft\Windows\Start Menu\Programs

R-Click. Copy mingw32.exe.

R-Click. Paste shortcut Here.

Rename the shortcut to esp8266.

R-Click. Copy esp8266.

Paste esp8266 to

%programdata%\Microsoft\Windows\Start Menu\Programs.

Copy N:\esp32_win32_msys2_environment_and_toolchain-20181001\msys32\home\AppuRaja\esp\ESP8266_RTOS_SDK

Paste shortcut there.

R-Click. Copy ESP8266_RTOS_SDK shortcut.

Go to Windows Explorer Address Bar again and type:

%programdata%\Microsoft\Windows\Start Menu\Programs

R-Click and Paste that copied shortcut.

Create Projects in other directories:

```
cp -r $IDF_PATH/examples/peripherals/adc /q/codes/esp
```

CH340G Windows Driver:

https://www.drivereasy.com/knowledge/ch340g-driver-download-and-update-in-windows/

https://www.arduined.eu/tag/ch340g/

Vim, Autocompletion, Code Checking:

Follow the instructions. Build the first project, 'get_started'.

In my case, it is "N:\esp32_win32_msys2_environment_and_toolchain-20181001\msys32\home\XxxxXxxx\esp" where I keep my ESP8266 projects. Copy the entire folder ESP8266_RTOS_SDK to C:\ drive. In C:\ESP8266_RTOS_SDK, keep only the following files and folders for your reference and delete everything else.

- components (dir)
- .gitignore (file)
- LICENSE (file)
- README.md (file)

The directory 'components' will be used by Vim-Clang for autocompletion and Syntastic for code checking.

Create a Windows Registry file 'openCMDhere.reg' with the following contents:

```
Windows Registry Editor Version 5.00

; Open CMD Here
; 'Open Terminal Here' MS equivalent

; https://github.com/microsoft/terminal/issues/1060
; https://stackoverflow.com/questions/27632612/comment-in-reg-file
; https://docs.microsoft.com/en-us/previous-
versions/windows/embedded/gg469889(v=winembedded.80)?redirectedfrom=MSDN

[HKEY_CURRENT_USER\Software\Classes\Directory\Background\shell\Open CMD Here\command]
@="C:\\Windows\\system32\\cmd.exe"
```

Now you can R-Click and select 'Open CMD Here'.

Navigate to the 'components' directory through CMD (or R-Click and select 'Open CMD Here' inside the 'components' directory) and list all the sub-directories there. Redirect the output to a text file.

https://stackoverflow.com/questions/35560540/batch-file-to-list-directories-recursively-in-windows-as-in-linux

```
dir /a:d /s /b /o:n > list.txt
```

You'll get an output quite similar to:

```
C:\ESP8266_RTOS_SDK\components\app_update
C:\ESP8266_RTOS_SDK\components\bootloader

...
...
...
...

C:\ESP8266_RTOS_SDK\components\wpa_supplicant\src\utils
C:\ESP8266_RTOS_SDK\components\wpa_supplicant\src\wps
```

Change the paths in the necessary configuration files.

Since you'll keep your future ESP8266 projects in a specific directory (for example, "N:\esp32_win32_msys2_environment_and_toolchain-20181001\msys32\home\XxxxXxxx\esp" in my case), you should keep your `.lvimrc`, `.clang.ow`, and `LocalVimConfig` in that folder. TULU-C-IDE will take care of the rest.