Visual Studio PlatformIO Installation

Dec. 27, 2020

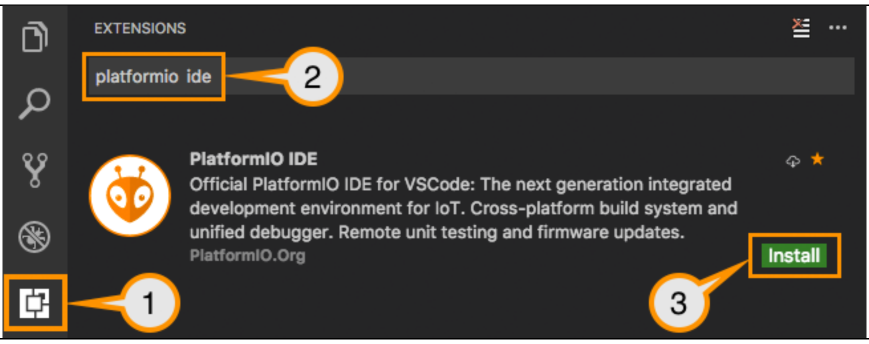
https://docs.platformio.org/en/latest/boards/espressif32/featheresp32.html

1. Download and install official Microsoft's Visual Studio Code, PlatformIO IDE is built on top of it

https://code.visualstudio.com/docs/?dv=win

2. Quickstart Platform IO

https://docs.platformio.org/en/latest/integration/ide/vscode.html#quick-start



Official PlatformIO IDE for VSCode: The next generation integrated development environment for IoT. Cross-platform build system and unified debugger. Remote unit testing and firmware updates.
PlatformIO.Org

2.1 **Open** VSCode Package Manager (#1)

2.2 select PlatformIO IDE 2.2.1 (#2)

2.3 select install (#3)

2.4 select reload VS window

3. Click on “New Project”, select a board and create new PlatformIO Project

3.1 Board: espressif32: adafruit ESP32 Feather, framework Arduino, default location, finish

(wait for tools to be installed)

4. Build (check mark on bottom of tool bar), upload (right arrow on bottom of tool bar)

(Led should blink).

5. Install ESP-IDF framework

5.1. Click on “New Project”, select a board and create new PlatformIO Project

5.2 Board: espressif32: adafruit ESP32 Feather, Framework: Espressif IoT Development Framework, default location, finish

(wait for tools to be installed)

Configuration File\r\n\n\nError: Processing featheresp32 (platform: espressif32; board: featheresp32; framework: espidf)\r\n—

Error: Detected a whitespace character in project paths.\r\n

Make sure Name: does not contain space or will crash

WiFi node example:

https://docs.platformio.org/en/latest/tutorials/espressif32/espidf_debugging_unit_testing_analysis.html#tutorial-espressif32-espidf-debugging-unit-testing-analysis

5.3 Make sure this new file main.c is registered as source file using idf_component_register function in src/CMakeLists.txt file:

```
idf_component_register(SRCS "main.c")
```

5.4 allow vscode to install CMake Tools.

6. Hello World example

Examples are located in:
`~/home/.platformio/packages/framework-espidf/examples`

File-> Open Folder -> examples/**espidf-hello-world**
(also can be downloaded from github:
https://github.com/platformio/platform-espressif32/blob/master/examples/espidf-hello-world/src/hello_world_main.c
6.1 build (takes about 6 minutes)

6.2 check baud rate in platformio.ini: (monitor_speed = 115200)

6.3 upload

6.4 works in putty terminal

update the default monitor speed to `115200` in `platformio.ini` file:

```
[env:featheresp32]
platform = espressif32
framework = espidf
board = featheresp32
monitor_speed = 115200
```

7. SkeletonHuzzah32

7.1 git clone https://github.com/ucb-ee192/SkeletonHuzzah32

7.2 build in platformio (first time will also build libraries for eps-idf framework)

7.3 flash, and monitor in either built-in terminal or putty terminal

8. Other Examples

Application programming interface (API) description can be found here:

`https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/index.html`

Examples setup for platformio:

https://github.com/platformio/platform-espressif32/tree/master/examples

~/home/.platformio/platforms/espressif32/examples 🡐 already has platformio.ini

Other example code for esp32 can be found here (missing platformio.ini, which needs to be added):

https://github.com/espressif/esp-idf/tree/master/examples

~/home/.platformio/packages/framework-espidf/examples/

8.1 Adding files to project folder (VS: file -> open file)

```
CMakeLists.txt: idf_component_register(SRCS "hello_world_main.c" 
“other1.c”   INCLUDE_DIRS "")
```

note that platformio.ini should be copied as above for featheresp32

9. Basic text IO

printf works.

Scanf does not work.

fgetc(stdin) and fputc(stdin) work

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/console.html

ESP-IDF provides `console` component, which includes building blocks needed to develop an interactive console over serial port. This component includes following facilities:

10. Configuring Options

Exit VS so there is not a conflict with sdkconfig. Make a backup copy of sdkconfig just in case

To edit sdkconfig use:

```
.platformio/penv/Scripts/pio.exe run -t menuconfig
```

(if pio.exe not in search path, add)

Use slash to find symbol if option is hard to find

Save, then exit.

(do not edit sdkconfig manually, it will be overwritten)

After restarting VS, allow VS to automatically remake CMakelist

Would you like to configure project 'SkeletonHuzzah32'? YES

If getting strange results, the sdkconfig may have been corrupted. Start over with a fresh version, such as from examples directory.

May be better to close and open folder in VS to read sdkconfig.

If get something like:

```
Error: Couldn't find target config target-__idf_cbor-
a7ffc6f8bf1ed76651c1.json
```

Then do `clean' from toolbar in platformio.

Useful `menuconfig` things to change:

1. CONFIG_FREERTOS_HZ
   `Component config > FreeRTOS ( -> Tick rate (Hz)`
   Tick rate (Hz) set to 200 Hz (can update steering servo at 5 ms interval)

2. CONFIG_FREERTOS_GENERATE_RUN_TIME_STATS

`Component config > FreeRTOS -> Enable FreeRTOS to collect run time stats`
vTaskGetRunTimeStats() will display the run time of each task as a % of the total run time of all CPUs (task run time / no of CPUs) / (total run time / 100)

3. CONFIG_ESP_COREDUMP_TO_FLASH_OR_UART
 Component config → Core dump (22 down) → Data destination -> UART
Select place to store core dump: flash, uart or none (to disable core dumps generation).

4. CONFIG_ESP_SYSTEM_PANIC_PRINT_REBOOT   or
ESP_SYSTEM_PANIC_PRINT_HALT
Component config → ESP32-specific (8 down)→ Panic handler behavior -> Print Registers and halt
Outputs the relevant registers over the serial port and halt the processor. Needs a manual reset to restart.

WARNING: `idf.py menuconfig` is incompatible with platformio and will mess up build.

However, the documentation describes all the options:

`https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/kconfig.html#component-config`

11. FreeRTOS

ESP-IDF FreeRTOS is based on the Xtensa port of FreeRTOS v8.2.0 with significant modifications for SMP compatibility (see ESP-IDF FreeRTOS SMP Changes). SMP= Symmetric Multi Processing

Furthermore, `float` cannot be used in interrupt service routines. (probably due to saving state of FPU registers?) Be careful on this. Double is ok, and it may be possible to save FPU state.

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html

https://github.com/espressif/esp-idf/tree/master/components/freertos

Level 1 so ok to use FPU (it will be interrupted by higher priority process)

- Timer 0 (int 6, level 1) (FREERTOS_CORETIMER_0) Select this to use timer 0

Enable FreeRTOS static allocation API: avoids problems with alloc/fragmentation. Probably safer for production code

The function vTaskGetRunTimeStats() will also be available if FREERTOS_USE_STATS_FORMATTING_FUNCTIONS and FREERTOS_USE_TRACE_FACILITY are enabled. vTaskGetRunTimeStats() will display the run time of each task as a % of the total run time of all CPUs (task run time / no of CPUs) / (total run time / 100)

Component config > FreeRTOS -> Allow use of float inside Level 1 ISR (EXPERIMENTAL)

When enabled, the usage of float type is allowed inside Level 1 ISRs.

11.1 Getting System Time

System time is read from an interrupt and made available to other processes through a queue. The queue avoids the problem of reading a 64 bit value in the middle of an update, giving an inconsistent time value.

Timer can also be read directly (this is a safe 64 bit read, protected by a spinlock.

```
timer_get_counter_value(timer_group_t group_num, timer_idx_t
timer_num, uint64_t *timer_val)
```

xQueuePeek: reads queue without emptying

xQueueOverwriteFromISR: overwriting data that is already held in the queue.

xQueueOverwriteFromISR() is intended for use with queues that have a length of one,

meaning the queue is either empty or full.

Time stamp is used for delay measurement such as velocity estimate, and for debugging to keep track of events

12. Debugging Tools

Useful `menuconfig` things to change:

1. CONFIG_COMPILER_OPTIMIZATION
   Compiler options -> Optimization Level
   Optimization level: “None” with -O0 produces compiled code without optimization.
   (sometimes needed if scanning a value which is changed by an external routine such as an IO line or an interrupt.)

2. CONFIG_COMPILER_STACK_CHECK_MODE
   Compiler options -> Stack smashing protection mode
   Stack smashing protection mode, Emit extra code to check for buffer overflows, such as stack smashing attacks

3. CONFIG_ESP_MAIN_TASK_STACK_SIZE

Component config > Common ESP-related (11 from top) -> Main task stack size
check stack size here if run out, default is 3584 byes. ~450 doubles

12.1 Core dump

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/core_dump.html?highlight=assert

1. Copy and paste base64 encoded core dump to a text file.
2. Remove ================ CORE DUMP START ================
3. And ================ CORE DUMP END ==================
   From text file
4. `espcoredump.py info_corefile -t b64 -c </path/to/saved/base64/text> </path/to/program/elf/file>`

c:>python c:\Users\ronf\esp\esp-idf\components\espcoredump\espcoredump.py info_corefile -t b64 -c coredump.txt .pio\build\featheresp32\firmware.elf

Shows lots of information, not so easy to parse, but does show active tasks/threads. Perhaps dbgcorefile (using gdb) would be more useful.

12.2 Error Codes

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/esp_err.html

Convert error codes: esp_err_to_name() and esp_err_to_name_r()

12.3 Watchdog timers

Watchdgs are useful for detecting stuck code.

Use RTC watchdog in start code: causes watchdog if startup fails

Task Watchdog Timer API Reference

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/wdts.html?highlight=task%20watchdog%20timer%20api%20reference

The Interrupt Watchdog Timer and the Task Watchdog Timer (TWDT). The Interrupt Watchdog Timer and the TWDT can both be enabled using Project Configuration Menu, however the TWDT can also be enabled during runtime.

CONFIG_ESP_INT_WDT

Component config → Common ESP-related -> Interrupt Watchdog

Useful `menuconfig` things to change:

1. CONFIG_ESP_INT_WDT
   Component config → Common ESP-related -> Interrupt Watchdog
   This watchdog timer can detect if the FreeRTOS tick interrupt has not been called for a certain time, either because a task turned off interrupts and did not turn them on for a long time, or because an interrupt handler did not return. (300 ms default timeout). Make this higher than the FreeRTOS tick rate.
   The INT WDT timeout should always be longer than the period between FreeRTOS ticks (see CONFIG_FREERTOS_HZ).

2. CONFIG_ESP_TASK_WDT_TIMEOUT_S.
   Component config > Common ESP-related > Task Watchdog timeout period (seconds)
   Watch dog timer triggered (5 seconds by default) if a task does not yield.
   The Task Watchdog Timer can be used to make sure individual tasks are still running. Enabling this option will cause the Task Watchdog Timer to be initialized automatically at startup. The Task Watchdog timer can be initialized after startup as well (see Task Watchdog Timer API Reference)

   A task can then subscribe to the TWDT using `esp_task_wdt_add()` in order to be watched. Each subscribed task must periodically call `esp_task_wdt_reset()` to reset the TWDT. Failure by any subscribed tasks to periodically call `esp_task_wdt_reset()` indicates that one or more tasks have been starved of CPU time or are stuck in a loop somewhere.

13. Virtual file system:

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/storage/vfs.html

included by default in sdkconfig, This component allows C library functions, such as fopen and fprintf, to work with FS drivers.

14. Floating Point and Interrupts

From forum: https://esp32.com/viewtopic.php?f=19&t=1292&p=6078&hilit=FPU+state#p6078

Yes, it is solvable: it means messing with the asssembler in the coprocessor handler to add the code to store the FPU state when an interrupt happens and restoring it when the interrupt is done. I have it on my list, but it's fairly low-priority: you can expect a fix eventually, but I don't know the timeframe for that.

```
uint32_t timer0_int = 0;
float timer0_float = 0.0;
DRAM_ATTR float timer0_k = 1.111111;

uint32_t cp0_regs[18];
```

```
void IRAM_ATTR timer0_intr()  // Interrupt handler for timer 0
{

  // get FPU state
  uint32_t cp_state = xthal_get_cpenable();

  if(cp_state) {
    // Save FPU registers
    xthal_save_cp0(cp0_regs);
  } else {
    // enable FPU
    xthal_set_cpenable(1);
  }

  timer0_int++;
  timer0_float = timer0_int * timer0_k;

  if(cp_state) {
    // Restore FPU registers
    xthal_restore_cp0(cp0_regs);
  } else {
    // turn it back off
    xthal_set_cpenable(0);
  }

}
```

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/freertos-smp.html#floating-points

Likewise due to Lazy Context Switching, only interrupt service routines of lowest priority (that is it the Level 1) can use `float`, higher priority interrupts do not support FPU usage.

6. SSH/WiFi

https://libraries.io/platformio/LibSSH-ESP32

https://libraries.io/cargo/esp32-hal

https://github.com/espressif/esp-drone/blob/master/docs/en/rst/communication.rst

https://github.com/espressif/esp-idf/tree/master/examples/protocols/sockets

https://github.com/espressif/esp-idf/blob/master/examples/protocols/sockets/udp_client/example_test.py

https://freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_TCP/API/socket.html