



- FreeRTOS

Sistemas Embarcados: (ELF74)

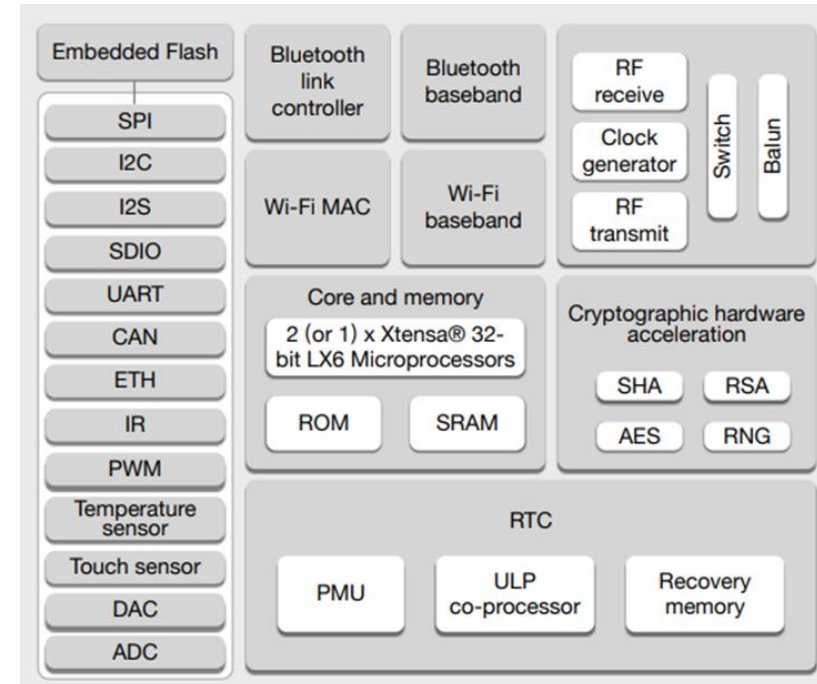
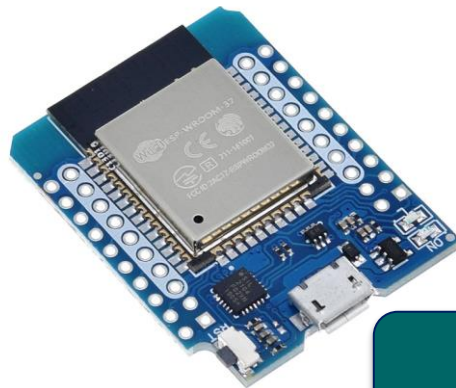
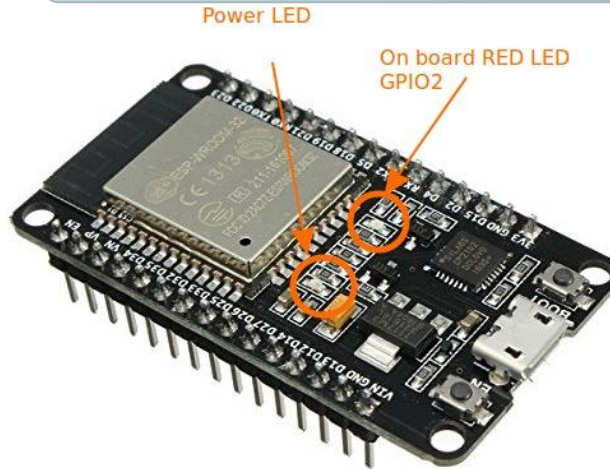
Prof: DaLuz



- FreeRTOS

FreeRTOS (Mini Curso)

Hardware:



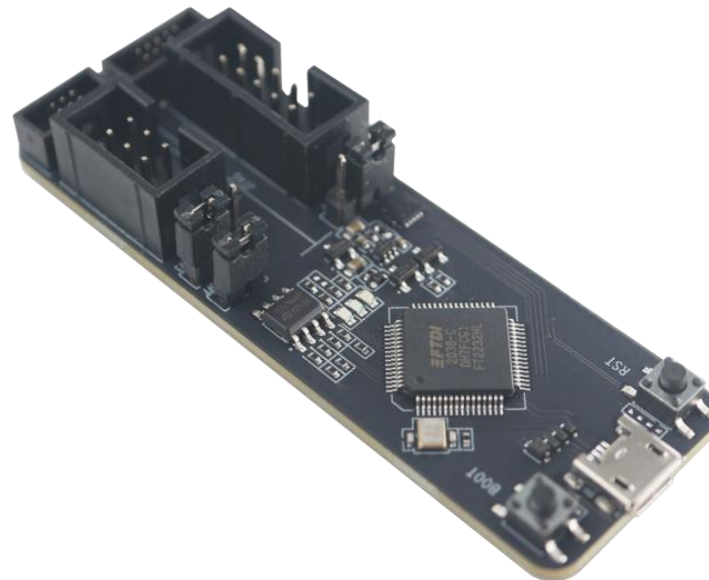
<https://www.espressif.com/en/products/devkits>
<https://idf.espressif.com/>
<https://docs.espressif.com/projects/esp-idf/en/stable/esp32/index.html>



- FreeRTOS

FreeRTOS (Mini Curso)

Hardware Debugger (ESP-Prog):



https://docs.espressif.com/projects/espressif-esp-dev-kits/en/latest/other/esp-prog/user_guide.html



- FreeRTOS

FreeRTOS (Mini Curso)

Hardware:



Company type	Subsidiary
Industry	Semiconductor intellectual property core
Founded	1997
Fate	Acquired by Cadence Design Systems in 2013
Headquarters	San Jose, California
Key people	Chris Rowen, Jack Guedj
Products	Microprocessors, HiFi audio, DSP cores
Website	ip.cadence.com

- Nome: **Tensile** - meaning capable of being extended + **Silica** from silicon
- 1997, Tensilica - Chris Rowen.
- 2002, Tensilica - flexible length instruction encodings, **FLIX**.
- 82 RISC ISA a 32-bit ALU, **16x GPRs** 32-bits, 1x **EPR**.
- 2013, **Cadence Design Systems** comprou 100% da Tensilica.
- AMD TrueAudio, PlayStation 4, "Kaveri" desktop APUs, Microsoft HoloLens (24 Tensilica DSP cores), **Espressif ESP8266 / ESP32**, ...

<https://en.wikipedia.org/wiki/Tensilica>

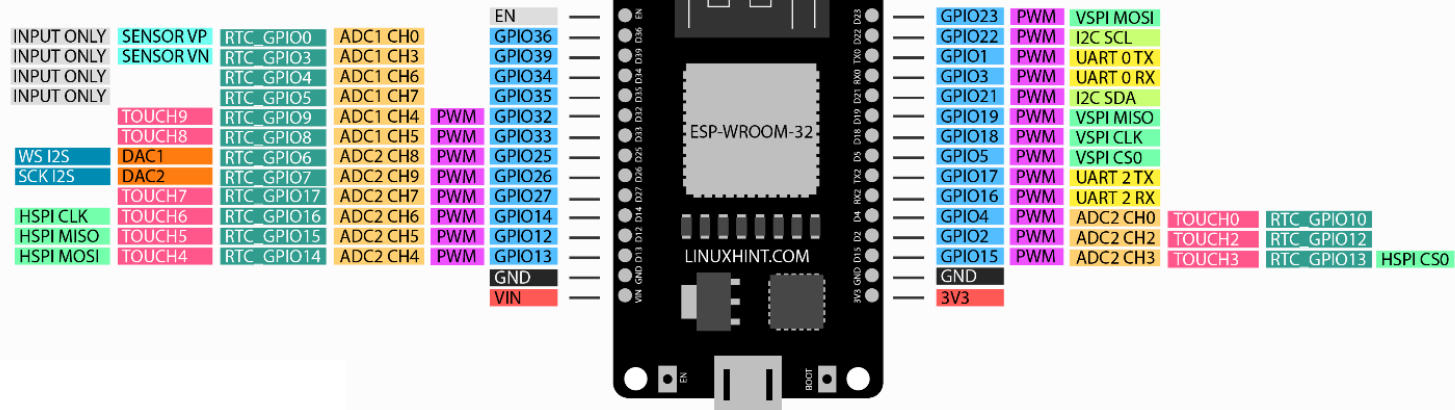


- FreeRTOS

FreeRTOS (Mini Curso)

Hardware (Kit ESP32-Doit 30Pin):

ESP32 PINOUT - 30 PIN VERSION

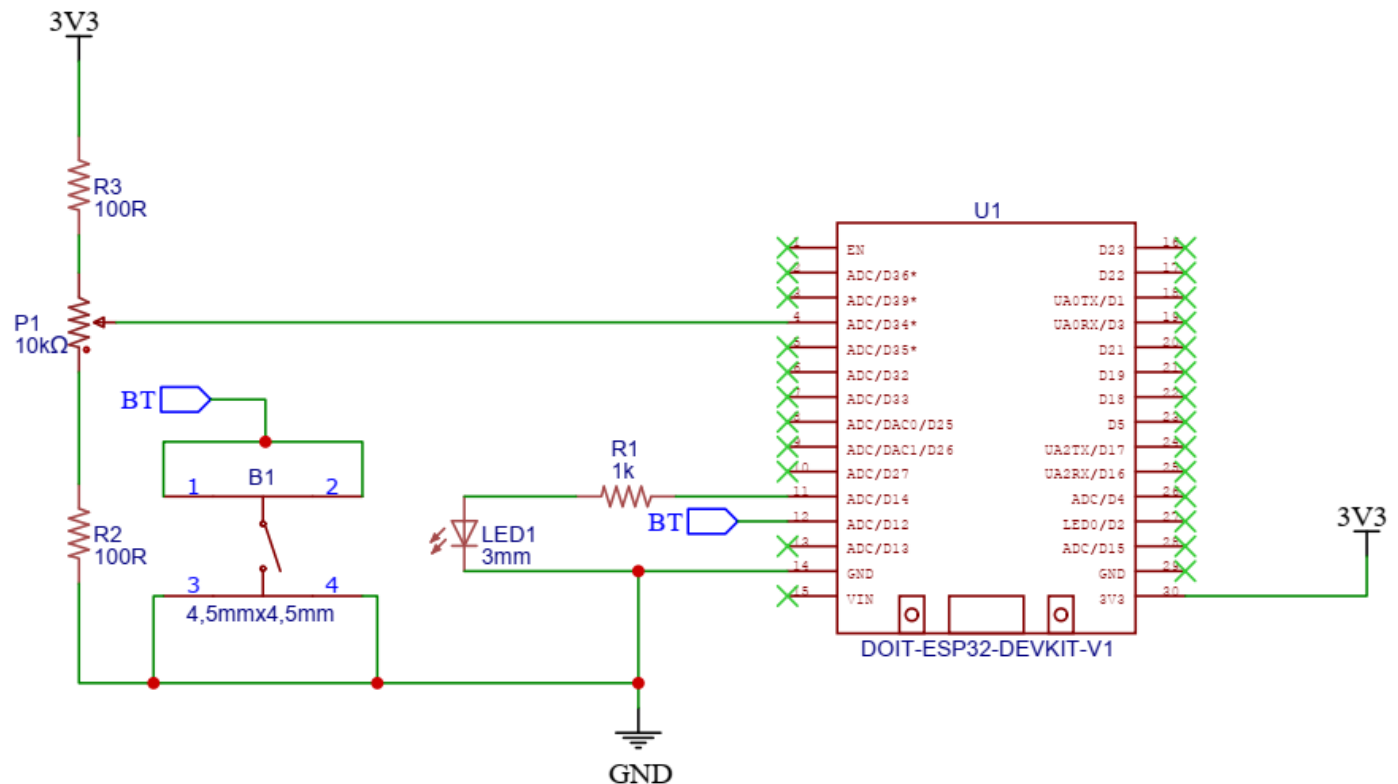




- FreeRTOS

FreeRTOS (Mini Curso)

Hardware (Esquemático):

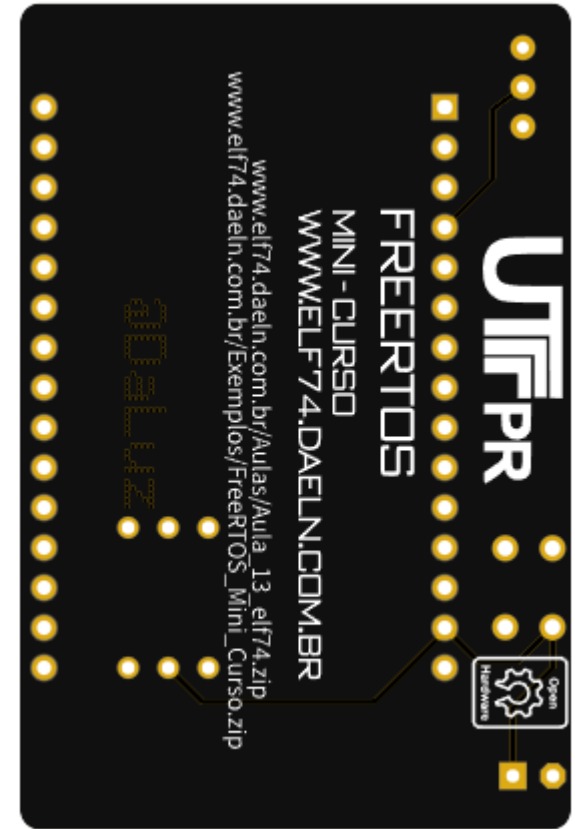
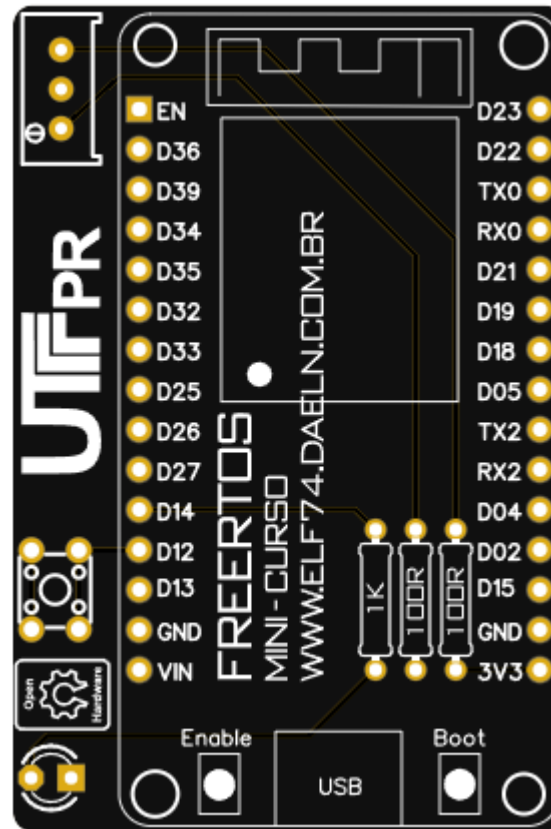




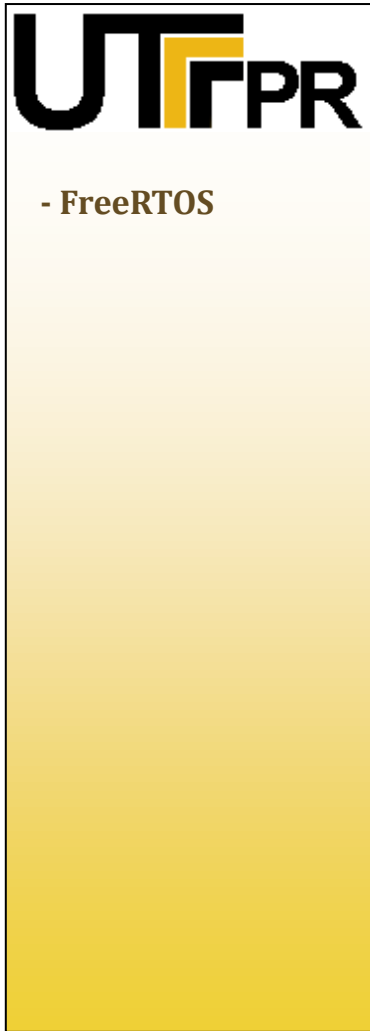
- FreeRTOS

FreeRTOS (Mini Curso)

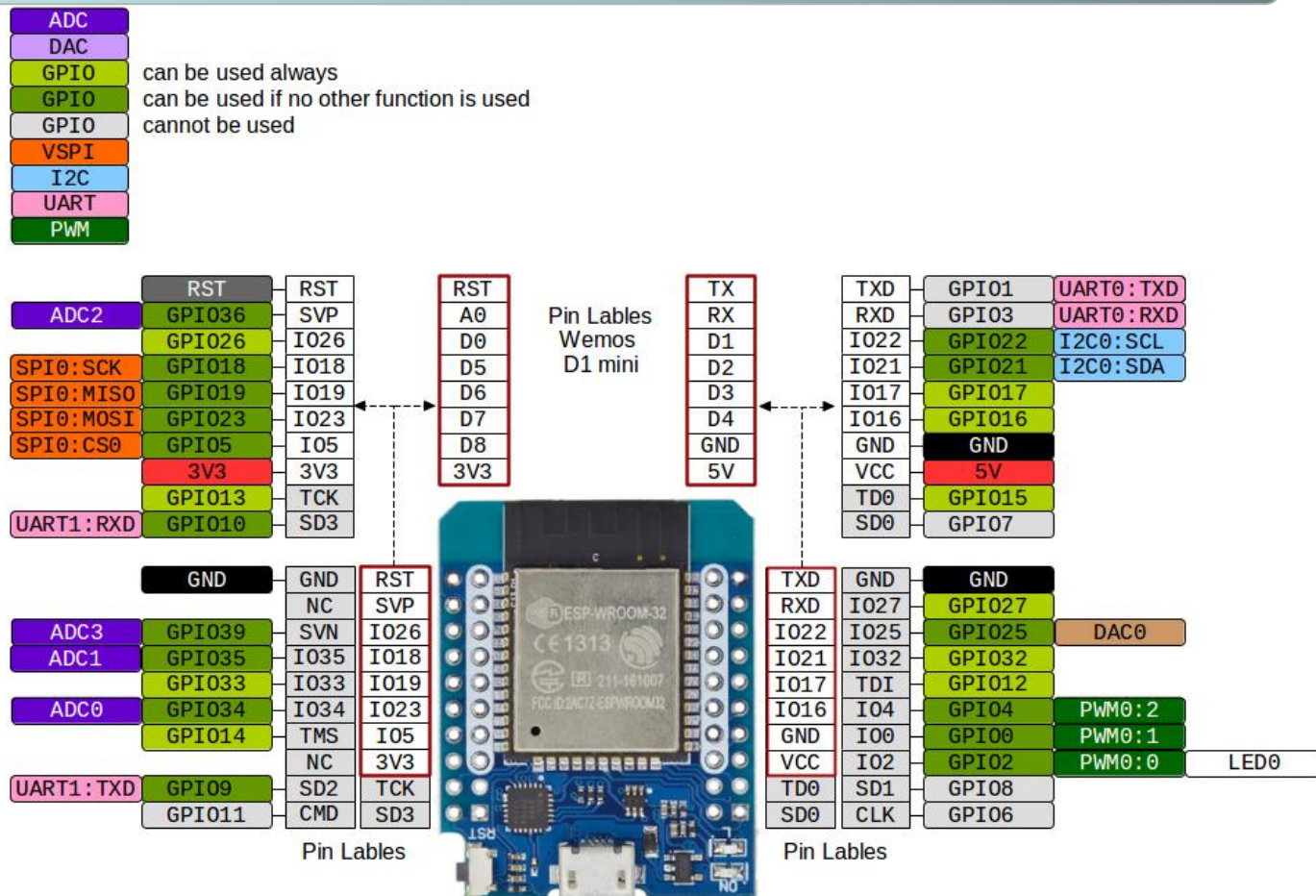
Hardware (PCB):



https://oshwlab.com/garcez/freertos_minipcb



Hardware (Kit ESP2 Wemos D1 Mini):

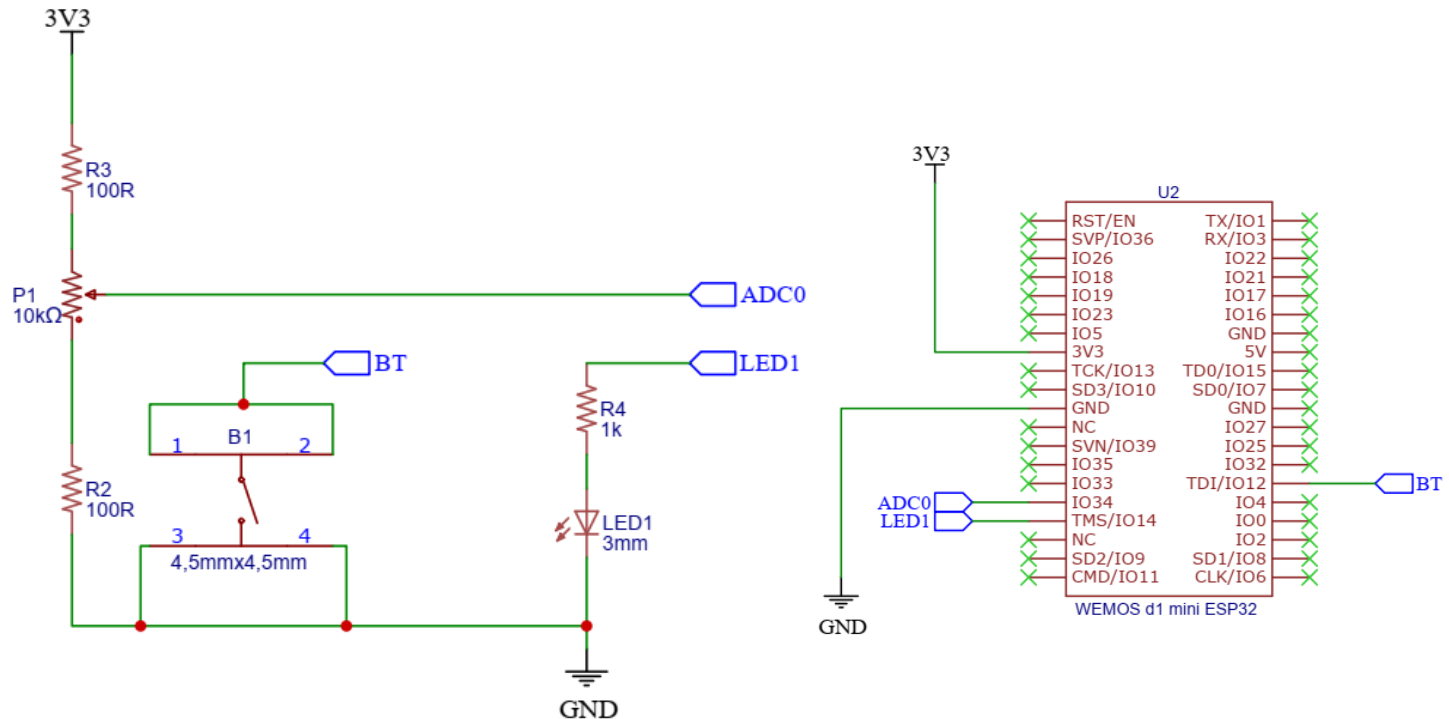




- FreeRTOS

FreeRTOS (Mini Curso)

Hardware (Esquemático):

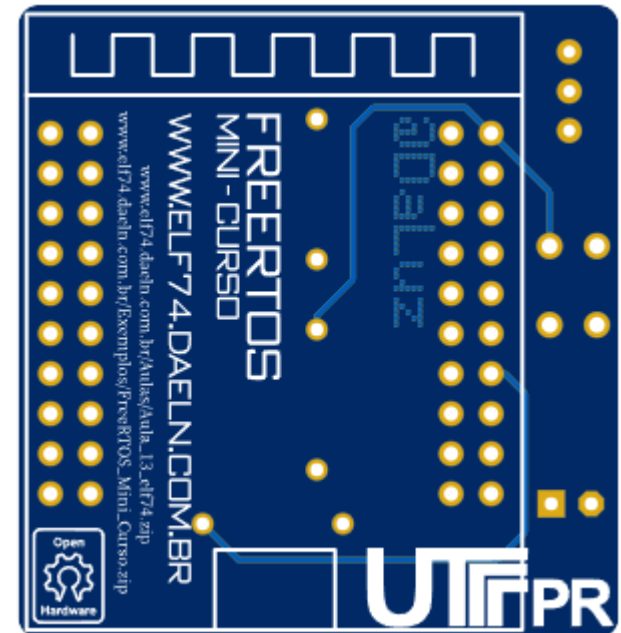
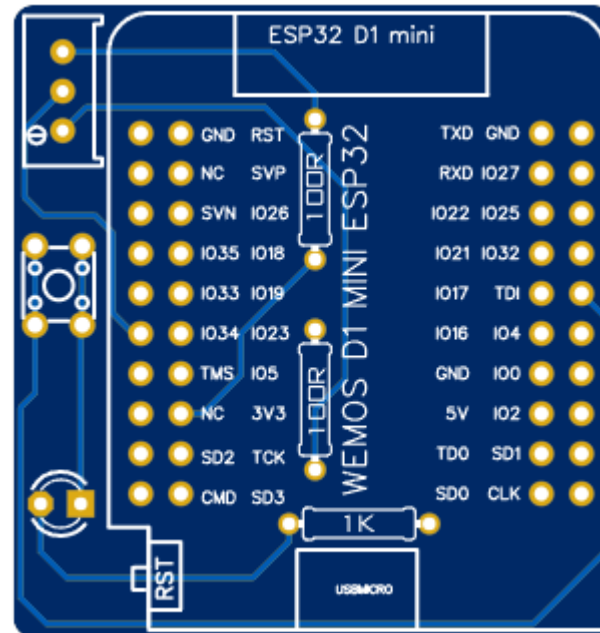




- FreeRTOS

FreeRTOS (Mini Curso)

Hardware (PCB):



https://oshwlab.com/garcez/freertos_minipcb_wemos

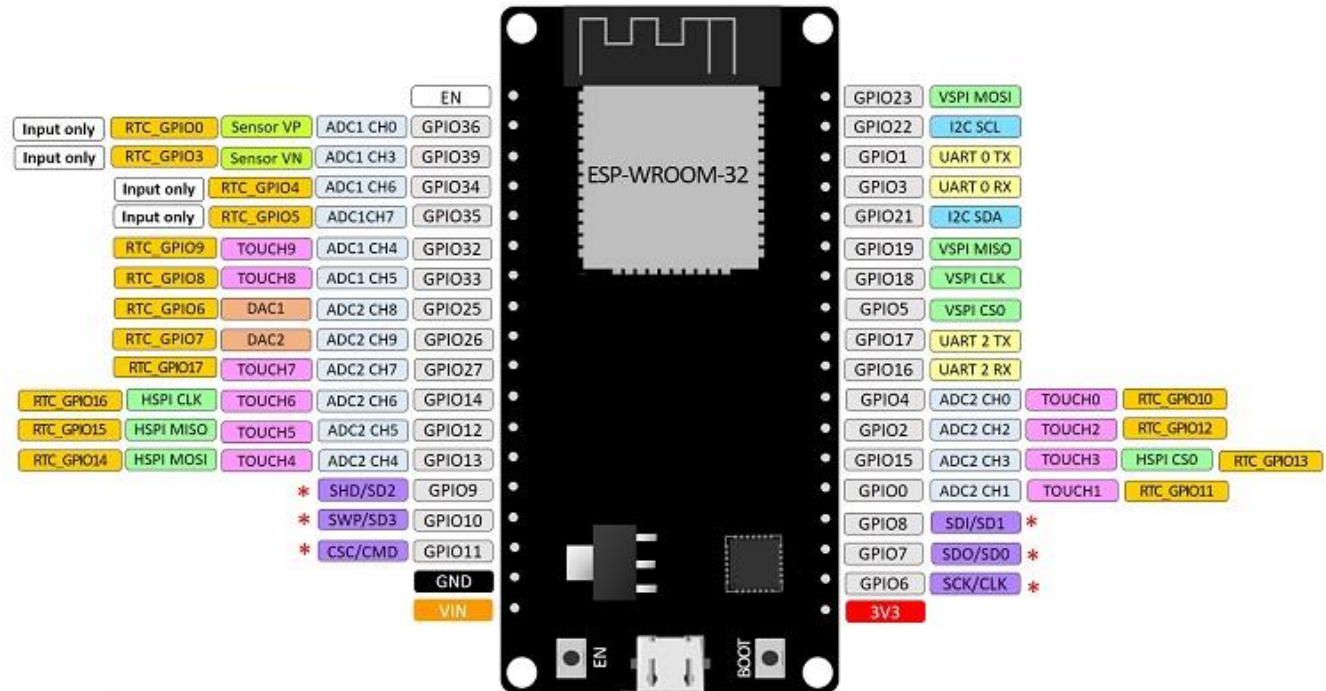


- FreeRTOS

FreeRTOS (Mini Curso)

Hardware (Kit SP32-Doit 36Pin):

ESP32 DEVKIT V1 - DOIT
version with 36 GPIOs



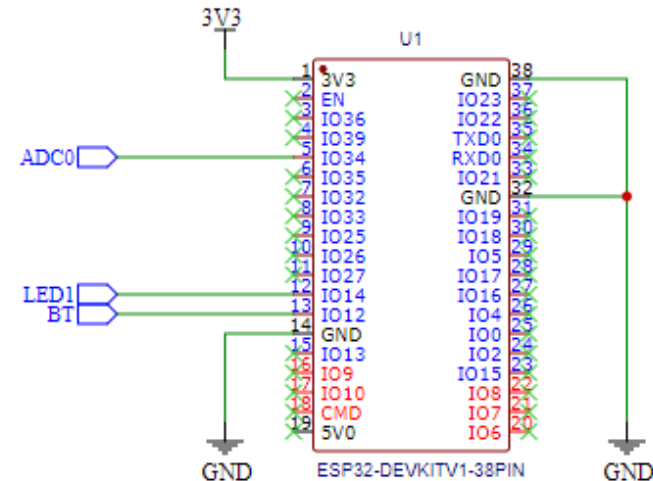
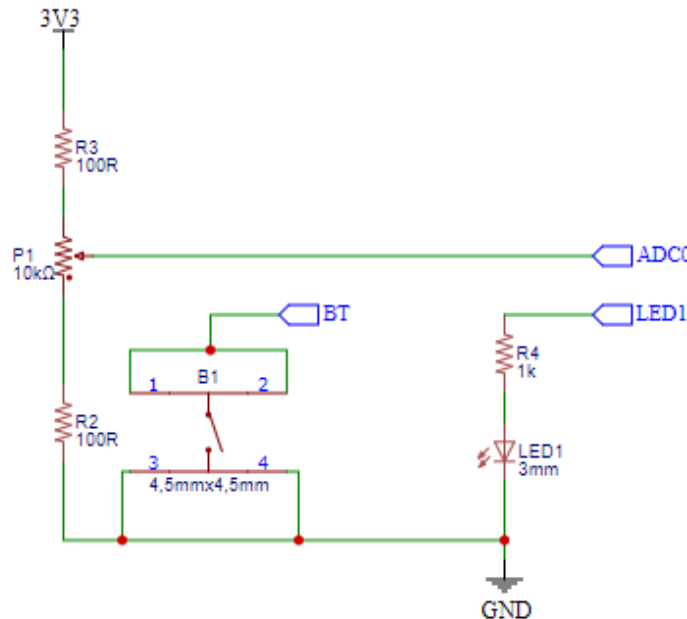
* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and CSC/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.



- FreeRTOS

FreeRTOS (Mini Curso)

Hardware (Esquemático):

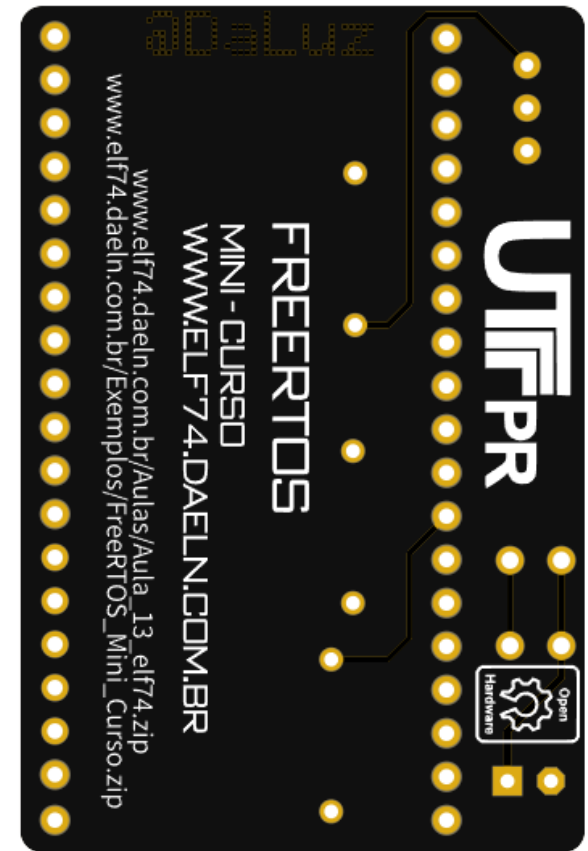
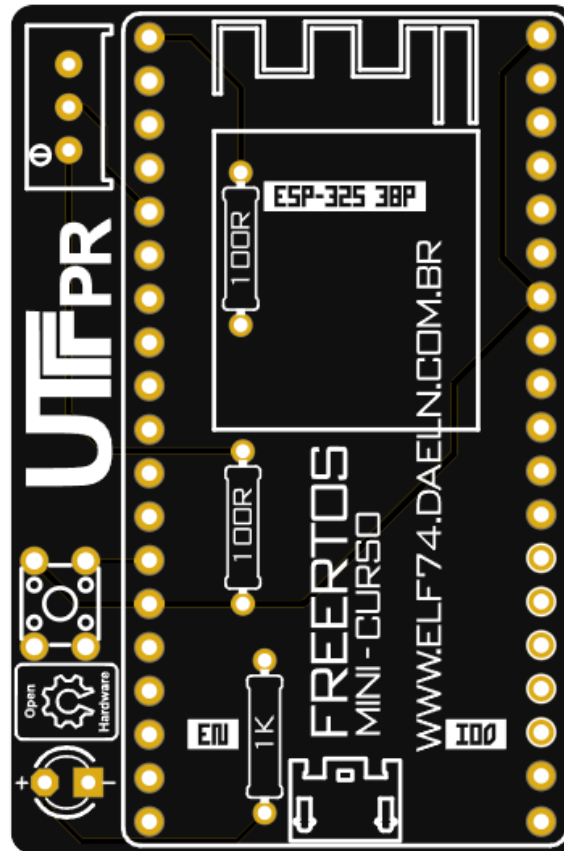




- FreeRTOS

FreeRTOS (Mini Curso)

Hardware (PCB):



https://oshwlab.com/garcez/freertos_minipcb_devkit

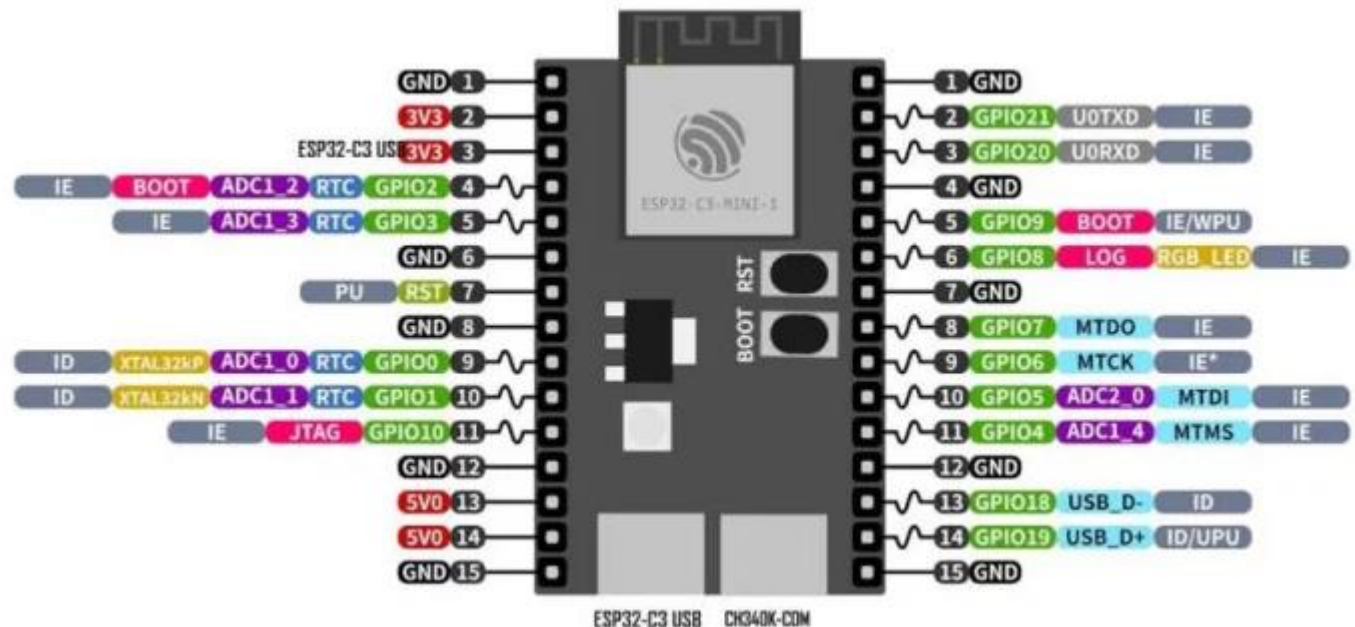


- FreeRTOS

FreeRTOS (Mini Curso)

Hardware (Kit ESP32-C3 RISC-V):

ESP32 C3 Mini Dev Board





- FreeRTOS

FreeRTOS (Mini Curso)

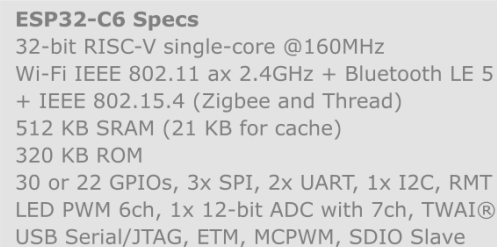
Hardware (Esquemático):

Hardware (PCB):

 Em Produção ...

 ESP32 C3 Mini Dev Board

Hardware (Kit ESP32-C6 RISC-V):



-
- PWM Capable Pin**
- Fast SPI Functions
 - General Purpose Input /Output
 - Low-Power UART Functions
 - Low-Power I2C Functions
 - SDIO Functions
 - Other Related Functions
 - Strapping Pin Functions
 - JTAG for Debugging and/or USB
 - Analogue-to-Digital Converter
 - Serial for Debug/Programming
 - Ground Plane
 - Power Rails (3V3 and 5V)
 - Low-Power GPIO Functions



- FreeRTOS

FreeRTOS (Mini Curso)

Hardware (Esquemático):

Hardware (PCB):

 Em Produção ...



- FreeRTOS

FreeRTOS (Mini Curso)

Software:



DownLoad:

<https://code.visualstudio.com/download>



DownLoad: (Obs: Usar Extensão do VSCode)

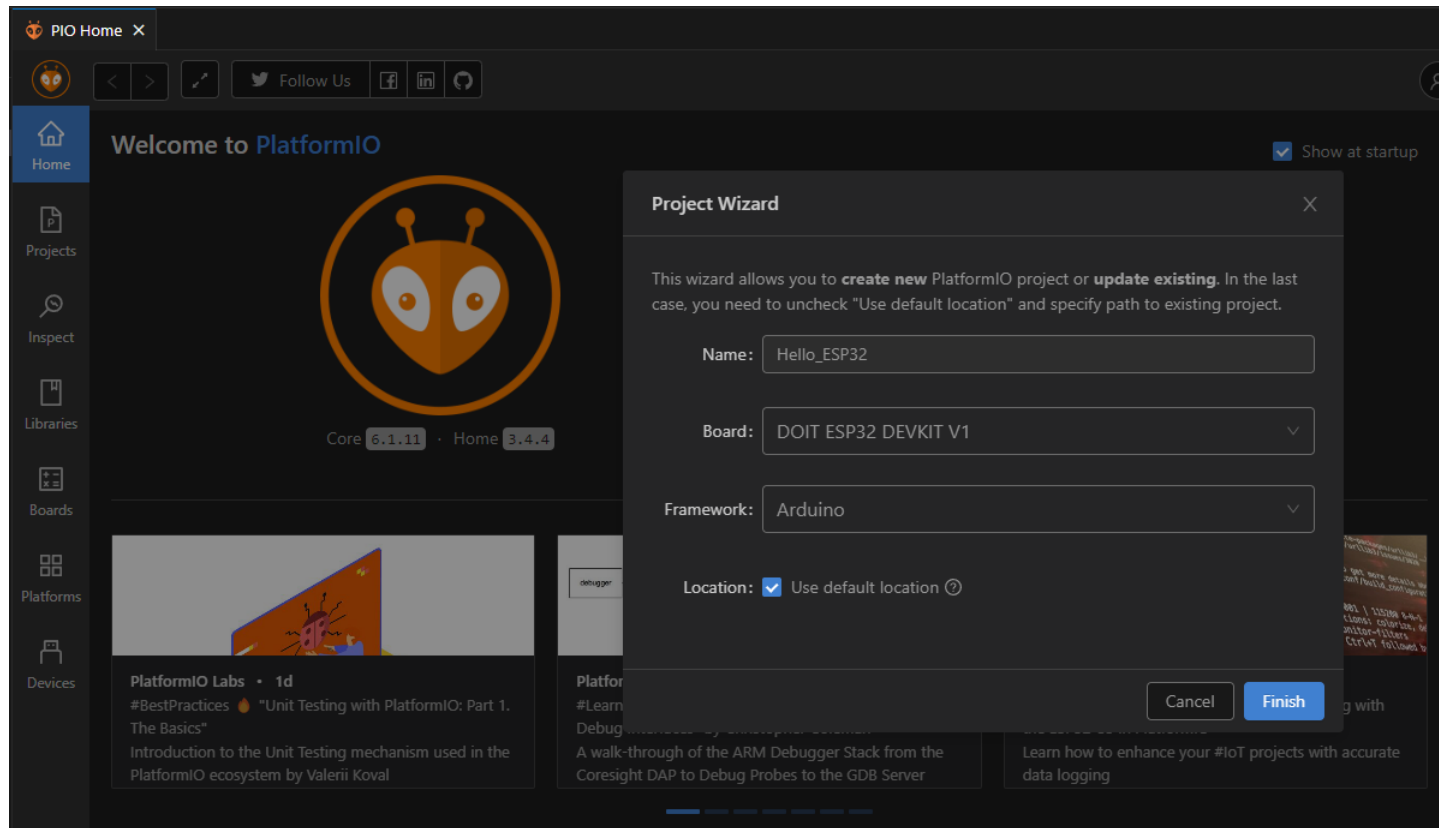
<https://platformio.org/>



- FreeRTOS

FreeRTOS (Mini Curso)

Software:





- FreeRTOS

FreeRTOS (Mini Curso)

Software:

A screenshot of the Visual Studio Code interface. The Explorer panel on the left shows a workspace named 'FREERTOS_ESP32_WS (WORKSPACE)' with folders for 'Vscode' and 'Hello_ESP32'. The 'Hello_ESP32' folder contains subfolders '.pio', '.vscode', 'include', 'lib', and 'src'. The 'src' folder contains files 'main.cpp', 'test', '.gitignore', and 'platformio.ini'. The 'platformio.ini' file is selected and its content is displayed in the main editor. The content of 'platformio.ini' is as follows:

```
Hello_ESP32 > platformio.ini
1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter
4 ; Upload options: custom upload port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ; Advanced options: extra scripting
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/page/projectconf.html
10
11 [env:esp32doit-devkit-v1]
12 platform = espressif32
13 board = esp32doit-devkit-v1
14 framework = arduino
15
16 monitor_speed=115200
17
```




- FreeRTOS

FreeRTOS (Mini Curso)

Software:

```
main.cpp x PIO Home platformio.ini
Hello_ESP32 > src > main.cpp > ...
1  /*#####
2  # Exemplo Hello World - ESP32 DOIT
3  # Pisca o LED da placa em intervalos de 1 segundo (500ms ON / 500ms OFF)
4  # Exemplo para entender o fluxo de criação de um projeto no Platformio com VSCode
5  # Por: Prof. Dr. Paulo Denis Garcez da Luz
6  # http://www.elf74.daeln.com.br
7  #####*/
8
9  //Inclusão de biblioteca Arduino
10 #include <Arduino.h>
11
12 /*mapeamento de pinos*/
13 #define LEDBUILTIN 2
14
15 //Função setup
16 void setup()
17 {
18     pinMode(LEDBUILTIN,OUTPUT);           //configura pino do LED como saída
19     Serial.begin(115200);                 //configura comunicação serial com baudarate de 115200 bps
20 }
21
22 //Função Loop
23 void loop()
24 {
25     digitalWrite(LEDBUILTIN,HIGH);        //liga LED
26     delay(500);                           //espera 0,5 s
27     digitalWrite(LEDBUILTIN,LOW);         //desliga LED
28     delay(500);                           //espera 0,5 s
29     Serial.println("Hello World ESP32-DOIT"); // imprime mensagem na serial
30 }
```



- FreeRTOS

FreeRTOS (Mini Curso)

Convenções:

- ▣ Variáveis *unsigned* começam com “u”.
- ▣ Variáveis do tipo *char* (8 bits) começam com “c”.
- ▣ Variáveis do tipo *short* (16 bits) começam com “s”.
- ▣ Variáveis do tipo *long* (32 bits) começam com “l”.
- ▣ Ponteiros começam com “p”.
- ▣ Funções privadas em um arquivo começam com “prv”.
- ▣ As funções da API são prefixadas com seu tipo de retorno, conforme a convenção definida para variáveis, com a adição do prefixo “v” de “void”.
- ▣ Os nomes das funções da API começam com o nome do arquivo no qual estão definidos. Por exemplo “vTaskDelete” é definido em “tasks.c” e possui um tipo de retorno “void”.



- FreeRTOS

FreeRTOS (Mini Curso)

Convenções:

- Indentação:
 - TAB** é usado para indentação. Um **TAB** é igual a 4 espaços.
- Comentários:
 - Os comentários nunca passam da coluna 80, a menos que sigam e descrevam um parâmetro.
 - Comentários de barra dupla no estilo C++ (//) não são usados.
- Layout:
 - O *layout* do código-fonte do **FreeRTOS** foi projetado para ser o mais fácil de visualizar e ler quanto possível.

```
/* Booleans use the microcontroller architecture's most efficient type, which
is defined in the kernel's portable layer as BaseType_t. */
BaseType_t xSchedulerRunning = pdFALSE;

/* Function names are always written on a single line, including the return
type. As always, there is no space before the opening parenthesis. There
is a space after an opening parenthesis. There is a space before a closing
parenthesis. There is a space after each comma. Parameters are given
verbose, descriptive names (unlike this example!). The opening and closing
curly brackets appear on their own lines, lined up underneath each other. */
void vAnExampleFunction( uint32_t ulParameter1, uint16_t usParameter2 )
{
/* Variable declarations are not indented, and use stdint.h defined types. */
uint8_t ucByte;

/* Code is indented. Curly brackets are always on their own lines
and lined up underneath each other. */
for( ucByte = 0U; ucByte < fileBUFFER_LENGTH; ucByte++ )
{
/* Indent again. */
}

/* For, while, do and if constructs follow a similar pattern. There is no
space before the opening parenthesis. There is a space after an opening
parenthesis. There is a space before a closing parenthesis. There is a
space after each semicolon (if there are any). There are spaces before and
after each operator. No reliance is placed on operator precedence -
parenthesis are always used to make precedence explicit. Magic numbers,
other than zero, are always replaced with a constant or #defined constant.
The opening and closing curly brackets appear on their own lines. */
for( ucByte = 0U; ucByte < fileBUFFER_LENGTH; ucByte++ )
{

```

<https://www.freertos.org/FreeRTOS-Coding-Standard-and-Style-Guide.html#NamingConventions>



- FreeRTOS

FreeRTOS (Mini Curso)

FreeRTOS API:

Task Creation

xTaskCreate
xTaskCreateStatic
vTaskDelete
xTaskGetStaticBuffers

Task Control

vTaskDelay
vTaskDelayUntil
xTaskDelayUntil
uxTaskPriorityGet
uxTaskPriorityGetFromISR
uxTaskBasePriorityGet
uxTaskBasePriorityGetFromISR
vTaskPrioritySet
vTaskSuspend
vTaskResume
xTaskResumeFromISR
xTaskAbortDelay



- FreeRTOS

FreeRTOS (Mini Curso)

FreeRTOS API:

Task Utilities

- uxTaskGetSystemState
- vTaskGetInfo
- xTaskGetCurrentTaskHandle
- xTaskGetIdleTaskHandle
- uxTaskGetStackHighWaterMark
- eTaskGetState
- pcTaskGetName
- xTaskGetHandle
- xTaskGetTickCount
- xTaskGetTickCountFromISR
- xTaskGetSchedulerState
- uxTaskGetNumberOfTasks
- vTaskList
- vTaskListTasks
- vTaskStartTrace
- ulTaskEndTrace
- vTaskGetRunTimeStats
- vTaskGetRunTimeStatistics
- vTaskGetIdleRunTimeCounter
- ulTaskGetRunTimeCounter
- ulTaskGetRunTimePercent
- ulTaskGetIdleRunTimeCounter
- ulTaskGetIdleRunTimePercent
- vTaskSetApplicationTaskTag
- xTaskGetApplicationTaskTag
- xTaskCallApplicationTaskHook
- pvTaskGetThreadLocalStoragePointer
- vTaskSetThreadLocalStoragePointer
- vTaskSetTimeOutState
- xTaskCheckForTimeOut



- FreeRTOS

FreeRTOS (Mini Curso)

FreeRTOS API:

Kernel Control

taskYIELD
taskENTER_CRITICAL
taskEXIT_CRITICAL
taskENTER_CRITICAL_FROM_ISR
taskEXIT_CRITICAL_FROM_ISR
taskDISABLE_INTERRUPTS
taskENABLE_INTERRUPTS
vTaskStartScheduler
vTaskEndScheduler
vTaskSuspendAll
xTaskResumeAll
vTaskStepTick



- FreeRTOS

FreeRTOS (Mini Curso)

FreeRTOS API:

Queue Management

- xQueueCreate
- xQueueCreateStatic
- vQueueDelete
- xQueueSend
- xQueueSendFromISR
- xQueueSendToBack
- xQueueSendToBackFromISR
- xQueueSendToFront
- xQueueSendToFrontFromISR
- xQueueReceive
- xQueueReceiveFromISR
- uxQueueMessagesWaiting
- uxQueueMessagesWaitingFromISR
- uxQueueSpacesAvailable
- xQueueReset
- xQueuePeek
- xQueuePeekFromISR
- vQueueAddToRegistry
- pcQueueGetName
- vQueueUnregisterQueue
- xQueueIsQueueEmptyFromISR
- xQueueIsQueueFullFromISR
- xQueueOverwrite
- xQueueOverwriteFromISR
- xQueueGetStaticBuffers



- FreeRTOS

FreeRTOS (Mini Curso)

FreeRTOS API:

Semaphores

- xSemaphoreCreateBinary
- xSemaphoreCreateBinaryStatic
- vSemaphoreCreateBinary [use xSemaphoreCreateBinary() for new designs]
- xSemaphoreCreateCounting
- xSemaphoreCreateCountingStatic
- xSemaphoreCreateMutex
- xSemaphoreCreateMutexStatic
- xSemaphoreCreateRecursiveMutex
- xSemaphoreCreateRecursiveMutexStatic
- vSemaphoreDelete
- xSemaphoreGetMutexHolder
- xSemaphoreTake
- xSemaphoreTakeFromISR
- xSemaphoreTakeRecursive
- xSemaphoreGive
- xSemaphoreGiveRecursive
- xSemaphoreGiveFromISR
- uxSemaphoreGetCount
- xSemaphoreGetStaticBuffer



- FreeRTOS

FreeRTOS (Mini Curso)

FreeRTOS API:

Software Timers

xTimerCreate
xTimerCreateStatic
xTimerIsTimerActive
pvTimerGetTimerID
pcTimerGetName
vTimerSetReloadMode
xTimerStart
xTimerStop
xTimerChangePeriod
xTimerDelete
xTimerReset
xTimerStartFromISR
xTimerStopFromISR
xTimerChangePeriodFromISR
xTimerResetFromISR
pvTimerGetTimerID
vTimerSetTimerID
xTimerGetTimerDaemonTaskHandle
xTimerPendFunctionCall
xTimerPendFunctionCallFromISR
pcTimerGetName
xTimerGetPeriod
xTimerGetExpiryTime
xTimerGetReloadMode



- FreeRTOS

FreeRTOS (Mini Curso)

FreeRTOS Docs:



Getting Started:

<https://www.freertos.org/a00104.html#getting-started>

API Reference:

<https://www.freertos.org/a00106.html>

FreeRTOS Documentation:

https://www.freertos.org/Documentation/RTOS_book.html



- FreeRTOS

FreeRTOS (Mini Curso)

Tarefas:

Protótipo de uma Tarefa:

```
void vTaskCode( void * pvParameters );
```

Corpo de uma Tarefa:

```
/* Task to be created. */  
void vTaskCode( void * pvParameters )  
{  
    for( ;; )  
    {  
        /* Task code goes here. */  
    }  
}
```

task.h

```
BaseType_t xTaskCreate(TaskFunction_t pvTaskCode,  
                        const char * const pcName,  
                        configSTACK_DEPTH_TYPE usStackDepth,  
                        void *pvParameters,  
                        UBaseType_t uxPriority,  
                        TaskHandle_t *pxCreatedTask);
```





- FreeRTOS

FreeRTOS (Mini Curso)

EXEMPLOS:

Download:

http://www.elf74.daeln.com.br/Exemplos/FreeRTOS_Mini_Curso.zip



- FreeRTOS

FreeRTOS (Mini Curso)

EX01:

Task1:	Led Blink de 200ms
Task2:	Imprime contador de tempo no monitor serial a cada 1s.
Funções utilizadas:	API Reference:
<code>void vTaskDelay(const TickType_t xTicksToDelay);</code>	Link URL
Macro: <code>pdMS_TO_TICKS(xTimeInMs);</code>	<code>#define pdMS_TO_TICKS(xTimeInMs) ((TickType_t) (((TickType_t) (xTimeInMs) * (TickType_t) configTICK_RATE_HZ) / (TickType_t) 1000))</code>



- FreeRTOS

FreeRTOS (Mini Curso)

EX02:

Task1:	Led Blink de 200ms
Task2:	Imprime contador de tempo no monitor serial a cada 1s. Quando o contador chegar em 11 deletamos a Task1. E quando o contador chegar em 17 deletamos a Task2.
Funções utilizadas:	API Reference:
<code>void vTaskDelete(TaskHandle_t xTask);</code>	Link URL



- FreeRTOS

FreeRTOS (Mini Curso)

EX03:

Task1:	Led Blink de 200ms
Task2:	Imprime contador de tempo no monitor serial a cada 1s. Quando o contador chegar em 11 suspendemos a Task1. E quando o contador chegar em 17 reiniciamos a Task1.
Funções utilizadas:	API Reference:
<code>void vTaskSuspend(TaskHandle_t xTaskToSuspend);</code>	Link URL
<code>void vTaskResume(TaskHandle_t xTaskToResume);</code>	Link URL



- FreeRTOS

FreeRTOS (Mini Curso)

EX04:

Task1:	Led Blink de 200ms com passagem de parâmetro (pino) de inicialização da task1.
Task2:	Imprime contador de tempo no monitor serial a cada 1s. O valor inicial de contagem é passado como parâmetro na inicialização da task2.



- FreeRTOS

FreeRTOS (Mini Curso)

ESP32-IDF -> FrameWork -> Arduino

C:\Users\xxxxxxx\.platformio\packages\framework-arduinoespressif32\cores\main.cpp

```
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_task_wdt.h"
#include "Arduino.h"

TaskHandle_t loopTaskHandle = NULL;

bool loopTaskWDTEnabled;

void loopTask(void *pvParameters)
{
    setup();
    for (;;)
    {
        if (loopTaskWDTEnabled) esp_task_wdt_reset();
        loop();
        if (serialEventRun) serialEventRun();
    }
}

extern "C" void app_main()
{
    loopTaskWDTEnabled = false;
    initArduino();
    xTaskCreateUniversal(loopTask, "loopTask", getArduinoLoopTaskStackSize(), NULL, 1,
                        &loopTaskHandle, ARDUINO_RUNNING_CORE);
}
```



- FreeRTOS

FreeRTOS (Mini Curso)

EX05:

Task1:	Led Blink de 200ms com passagem de parâmetro (pino led_built_in) de inicialização da task1. Utilizando o core principal.
Task2:	Imprime o contador de tempo no monitor serial a cada 1s. O valor inicial de contagem é passado como parâmetro na inicialização da task2. Utilizando o core secundário.
Task3:	Led Blink de 500ms com passagem de parâmetro (pino led_pcb_ext1) de inicialização da task1. Utilizando o core principal.
Funções utilizadas:	API Reference:
<code>void xTaskCreatePinnedToCore (...);</code>	Função fornecida pelo fabricante, não é da API do FreeRTOS



- FreeRTOS

FreeRTOS (Mini Curso)

EX06:

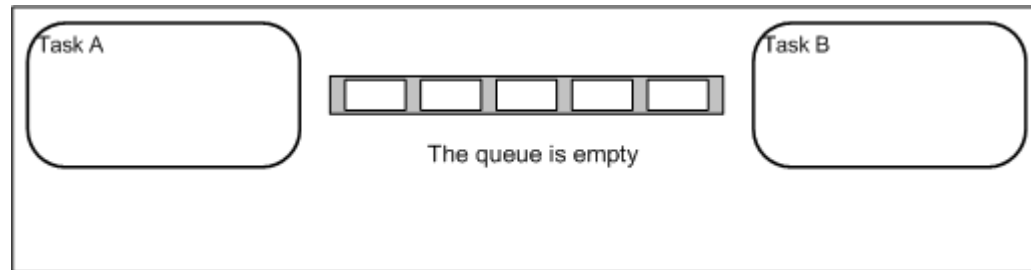
Task1:	Led Blink de 200ms com passagem de parâmetro (pino led_built_in) de inicialização da task1. Utilizando o core principal.
Task2:	Imprime o contador de tempo no monitor serial a cada 1s e o tamanho de sua Stack. O valor inicial de contagem é passado como parâmetro na inicialização da task2. Utilizando o core secundário.
Task3:	Led Blink de 500ms com passagem de parâmetro (pino led_pcb_ext1) de inicialização da task1. Utilizando o core principal.
Funções utilizadas:	API Reference:
<pre>UBaseType_t uxTaskGetStackHighWaterMark (TaskHandle_t xTask);</pre>	Link URL



- FreeRTOS

FreeRTOS (Mini Curso)

EX07:



Task1:	Coloca inteiros na Fila a cada 500ms até a o contador chegar em 10. Quando o contador chega em 10 a tarefa entra em espera de 5s.
Task2:	Retira os inteiros da fila e mostra no monitor serial. Quando a tarefa1 está em espera exibe um Timeout.

Funções utilizadas:

```
QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength,
                           UBaseType_t uxItemSize );
BaseType_t xQueueReceive(
    QueueHandle_t xQueue,
    void *pvBuffer,
    TickType_t xTicksToWait );
```

API Reference:

[Link URL](#)

[Link URL](#)



- FreeRTOS

FreeRTOS (Mini Curso)

EX08:

Task0: (loop)	Executa o Led Blinky a 333ms.
Task1:	Retira da fila quantas vezes o botão foi pressionado e envia pela serial o valor.
ISR:	Interrupção que pega o pressionamento do botão, sem tratamento do repique. Quando em execução coloca o número de vezes que o botão foi pressionado na fila.

Funções utilizadas:	API Reference:
<pre>BaseType_t xQueueSendFromISR (QueueHandle_t xQueue, const void *pvItemToQueue, BaseType_t *pxHigherPriorityTaskWoken);</pre>	Link URL



- FreeRTOS

FreeRTOS (Mini Curso)

EX09:



Task0: (loop)

Inverte o estado do Led a cada 2,5s e libera o semáforo binário neste mesmo intervalo.

Task1:

Quando adquire o semáforo binário, executa uma leitura analógica do potenciômetro e envia esse dado pela serial, voltando a aguardar a liberação do semáforo binário.

Funções utilizadas:

API Reference:

SemaphoreHandle_t xSemaphoreCreateBinary(void);

[Link URL](#)

xSemaphoreGive(SemaphoreHandle_t xSemaphore);

[Link URL](#)

xSemaphoreTake(SemaphoreHandle_t xSemaphore,
TickType_t xTicksToWait);

[Link URL](#)



- FreeRTOS

FreeRTOS (Mini Curso)

EX10:



Task0: (loop)	Pisca o Led (100ms=ON e 900ms=OFF)
Task1:	Aguarda o semáforo binário ser liberado e informa o número de vezes que o botão foi pressionado via ISR do botão.
ISR:	Interrupção que pega o pressionamento do botão, sem tratamento do repique. Quando em execução libera o semáforo e força a troca de contexto.
Funções utilizadas:	API Reference:
xSemaphoreGiveFromISR(SemaphoreHandle_t xSemaphore, signed BaseType_t *pxHigherPriorityTaskWoken)	Link URL
xTaskNotifyAndQueryFromISR / xTaskNotifyAndQueryIndexedFromISR	Link URL



- FreeRTOS

FreeRTOS (Mini Curso)

EX11:



Task0: (loop)	Pisca o Led (200ms=ON e 800ms=OFF)
Task1:	Aguarda o semáforo contador ter eventos e informa o número de vezes que o botão foi pressionado via ISR do botão, tendo um delay(500ms) de espera entre os tratamentos.
ISR:	Interrupção que pega o pressionamento do botão, sem tratamento do repique. Quando em execução libera o semáforo contador e força a troca de contexto.

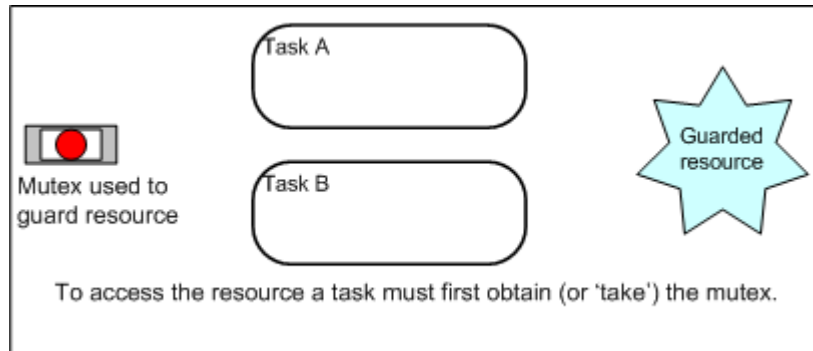
Funções utilizadas:	API Reference:
SemaphoreHandle_t xSemaphoreCreateCounting(UBaseType_t uxMaxCount, UBaseType_t uxInitialCount);	Link URL



- FreeRTOS

FreeRTOS (Mini Curso)

EX12:



Task0: (loop)	Pisca o Led (300ms=ON e 700ms=OFF)
Task1 (prioridade 1):	Aguarda pelo Mutex e informa o número da Task pela serial através de uma função. Congela por 2s a Task e libera para troca de contexto (100ms).
Task2 (prioridade 4):	Aguarda pelo Mutex e informa o número da Task pela serial através de uma função. Congela por 500ms a Task e libera para troca de contexto (100ms).
<p><i>*Obs: recompilar com a linha do <code>vTaskDelay(pdMS_TO_TICKS(100));</code> comentada e verificar o que acontece ...</i></p>	
Funções utilizadas:	API Reference:
SemaphoreHandle_t xSemaphoreCreateMutex(void)	Link URL



- FreeRTOS

FreeRTOS (Mini Curso)

EX13:



Task0: (loop)	Idle ...
Task1:	Faz <i>pooling</i> no botão com tratamento do repique, caso o botão seja pressionado ativa o timer2, desativa o timer1 e liga o Led da pcb.
vFTimer1:	Função de <i>callback</i> do S.T.1: Led <i>built-in blinky</i> de 750ms. (<i>auto-reload</i>)
vFTimer2:	Função de <i>callback</i> do S.T.2: reativa o S.T.1 e desliga o Led da pcb. (<i>one-shot</i>)

Funções utilizadas:	API Reference:
<pre>TimerHandle_t xTimerCreate (const char * const pcTimerName, const TickType_t xTimerPeriod, const UBaseType_t uxAutoReload, void * const pvTimerID, TimerCallbackFunction_t pxCallbackFunction);</pre>	Link URL
<pre>BaseType_t xTimerStart(TimerHandle_t xTimer, TickType_t xBlockTime);</pre>	Link URL
<pre>BaseType_t xTimerStop(TimerHandle_t xTimer, TickType_t xBlockTime);</pre>	Link URL



- FreeRTOS

FreeRTOS (Mini Curso)

EX14:



Task0: (loop)	Pisca o Led (300ms=ON e 700ms=OFF)
Task1:	Aguarda sinalização da FLAG do grupo de eventos no bit 0, enviando pela serial o evento, após o S.T.1 ativar a FLAG.
Task2:	Aguarda sinalização da FLAG do grupo de eventos no bit 1, enviando pela serial o evento, após o S.T.1 ativar a FLAG.
vFTimer1:	Ativa as FLAGS ... Com 3s, 6s e 9s.

Funções utilizadas:	API Reference:
EventGroupHandle_t xEventGroupCreate(void);	Link URL
EventBits_t xEventGroupWaitBits (const EventGroupHandle_t xEventGroup, const EventBits_t uxBitsToWaitFor, const BaseType_t xClearOnExit, const BaseType_t xWaitForAllBits, TickType_t xTicksToWait);	Link URL
EventBits_t xEventGroupSetBits(EventGroupHandle_t xEventGroup, const EventBits_t uxBitsToSet);	Link URL



- FreeRTOS

FreeRTOS (Mini Curso)

EX15:



Task0: (loop)	Led Blinky (250ms=ON e 250ms=OFF)
Task1:	Aguarda uma notificação da ISR para mudar o estado do Led e enviar pela serial a notificação de tratamento.
ISR:	ISR do pino do PushButton (não faz tratamento do repique), usa Task Notification como sinalizador de eventos do botão para a Task1

Funções utilizadas:	API Reference:
<pre>void vTaskNotifyGiveFromISR(TaskHandle_t xTaskToNotify, BaseType_t *pxHigherPriorityTaskWoken);</pre>	Link URL
<pre>void vTaskNotifyGiveIndexedFromISR(TaskHandle_t xTaskHandle, UBaseType_t uxIndexToNotify, BaseType_t *pxHigherPriorityTaskWoken);</pre>	



- FreeRTOS

FreeRTOS (Mini Curso)

EX16:

Task0: (loop)	Idle ...
Task1:	Led Blinky de 1s com passagem de struct como parâmetro durante a criação da Task1
Task2:	Led Blinky de 200ms com passagem de struct como parâmetro durante a criação da Task2



- FreeRTOS

Referências:

Continuação dos Labs ...



* Refs ↔ freertos.org, microncontrollerslab.com, jblopen.com, microsoft.com, udemy.com, embarcados.com.br.