MGA_Doxi

AUTORES
ANGIE TATIANA PEREZ
ANDRES FERNANDEZ RIOS
ELKIN ALEJANDRO LEDESMA

Versión 1.0.1

Tabla de contenidos

Table of contents
Índice de clases

Lista de clases

Lista de clases, estructuras, uniones e interfaces con breves descripciones:

Índice de archivos

Lista de archivos

Lista de todos los archivos con breves descripciones:

Documentación de clases

Referencia de la estructura ButtonEvent

Estructura para manejar eventos de botones.

Atributos públicos

- int **increment**
- char **name** [4]

Descripción detallada

Estructura para manejar eventos de botones.

Definición en la línea **144** del archivo **MGA.ino**.

Documentación de datos miembro

int ButtonEvent::increment

Incremento: +1 o -1.

Definición en la línea **145** del archivo **MGA.ino**.

char ButtonEvent::name[4]

Nombre del botón ("INC" o "DEC").

Definición en la línea **146** del archivo **MGA.ino**.

La documentación de esta estructura está generada del siguiente archivo:

- C:/Users/Andres Fernandez/Desktop/CODIGO_TEMPERATURA/MGA/MGA/**MGA.ino**

Referencia de la estructura SensorData

Estructura para almacenar datos de los sensores.

Atributos públicos

- float **temperature**
- float **humidity**
- int **light**
- uint8_t **errorFlags**

Descripción detallada

Estructura para almacenar datos de los sensores.

Definición en la línea **130** del archivo **MGA.ino**.

Documentación de datos miembro

uint8_t SensorData::errorFlags

Indicador de errores en la lectura (0: sin error).

Definición en la línea **134** del archivo **MGA.ino**.

float SensorData::humidity

Humedad medida.

Definición en la línea **132** del archivo **MGA.ino**.

int SensorData::light

Valor analógico del sensor de luz.

Definición en la línea **133** del archivo **MGA.ino**.

float SensorData::temperature

Temperatura medida.

Definición en la línea **131** del archivo **MGA.ino**.

La documentación de esta estructura está generada del siguiente archivo:

- C:/Users/Andres Fernandez/Desktop/CODIGO_TEMPERATURA/MGA/MGA/**MGA.ino**

Documentación de archivos

Referencia del archivo C:/Users/Andres Fernandez/Desktop/CODIGO_TEMPERATURA/MGA/MGA/MGA.ino

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
#include <debounce.h>
#include <RtcDS1302.h>
#include "esp_sleep.h"
```

Clases

struct **SensorData***Estructura para almacenar datos de los sensores.*

struct **ButtonEvent***Estructura para manejar eventos de botones.*

defines

- #define **DHTPIN** 5
  Pin de conexión para el sensor DHT.

- #define **DHTTYPE** DHT22
  Tipo de sensor DHT utilizado.

- #define **LDRPIN** 32
  Pin analógico para el sensor de luz (LDR).

- #define **LED_BLUE** 26
  Pin de salida para el LED azul.

- #define **BUTTON_DEC** 34
  Pin para el botón de decremento.

- #define **BUTTON_INC** 33
  Pin para el botón de incremento.

- #define **BUZZER_PIN** 14
  Pin de salida para el buzzer.

- #define **TEMP_THRESHOLD** 26
  Umbral de temperatura para activar la alarma.

- #define **HUM_THRESHOLD** 80
  Umbral de humedad para activar la alarma.

- #define **LIGHT_THRESHOLD** 700

Umbral de luz para activar la alarma.

- #define **uS_TO_S_FACTOR** 1000000
  Factor de conversión de microsegundos a segundos.

- #define **TIME_TO_SLEEP** 10
  Tiempo en segundos antes de entrar en deep sleep.

- #define **ERROR_NONE** 0x00
  Sin error en la lectura de sensores.

- #define **ERROR_TEMP** 0x01
  Error en la lectura de temperatura.

- #define **ERROR_HUM** 0x02
  Error en la lectura de humedad.

- #define **ERROR_LIGHT** 0x04
  Error en la lectura de luz.

- #define **countof**(a)
  Macro para calcular el tamaño de un arreglo.

Funciones

- ThreeWire **myWire** (4, 15, 2)
  Inicialización del bus de tres hilos para el RTC.

- RtcDS1302< ThreeWire > **Rtc** (**myWire**)
- DHT **dht** (**DHTPIN**, **DHTTYPE**)
- void **logWithTimestamp** (const char *label, float value, const char *unit)
  Imprime un mensaje con la fecha y hora actual del RTC.

- void **logError** (const char *errorMsg)
  Imprime mensajes de error con la fecha y hora actual del RTC.

- void **print_wakeup_reason** ()
  Imprime la razón por la cual el ESP32 despertó del deep sleep.

- void IRAM_ATTR **incISR** ()
  Rutina de servicio de interrupción para el botón de incremento.

- void IRAM_ATTR **decISR** ()
  Rutina de servicio de interrupción para el botón de decremento.

- void **readTemperatureTask** (void *pvParameters)
  Tarea de lectura de temperatura con timestamp.

- void **readHumidityTask** (void *pvParameters)
  Tarea de lectura de humedad con timestamp.

- void **readLightTask** (void *pvParameters)
  Tarea de lectura de luz con timestamp.

- void **aggregatorTask** (void *pvParameters)
  Tarea de agregación de datos de sensores.

- void **controlLedTask** (void *pvParameters)
  Tarea para el control del LED (alarma) y el buzzer.

- void **buttonEventTask** (void *pvParameters)
  Tarea para manejar eventos de botones (contador).

- void **deepSleepTask** (void *pvParameters)
  Tarea para iniciar Deep Sleep con timer y botón como despertador.

- void **setup** ()
  Función de configuración.

Variables

- QueueHandle_t **sensorQueue**
- QueueHandle_t **buttonQueue**
- volatile float **g_temperature** = 0.0f
- volatile float **g_humidity** = 0.0f
- volatile int **g_light** = 0
- volatile int **counter** = 0
- volatile unsigned long **lastIncTime** = 0
- volatile unsigned long **lastDecTime** = 0
- const unsigned long **debounceDelay** = 500000

Documentación de «define»

#define BUTTON_DEC 34

Pin para el botón de decremento.

Definición en la línea **56** del archivo **MGA.ino**.

#define BUTTON_INC 33

Pin para el botón de incremento.

Definición en la línea **61** del archivo **MGA.ino**.

#define BUZZER_PIN 14

Pin de salida para el buzzer.

Definición en la línea **66** del archivo **MGA.ino**.

#define countof(a)

Valor:

```
(sizeof(a) / sizeof(a[0]))
```

Macro para calcular el tamaño de un arreglo.

Definición en la línea **168** del archivo **MGA.ino**.

#define DHTPIN 5

Pin de conexión para el sensor DHT.

Definición en la línea **36** del archivo **MGA.ino**.

#define DHTTYPE DHT22

Tipo de sensor DHT utilizado.

Definición en la línea **41** del archivo **MGA.ino**.

#define ERROR_HUM 0x02

Error en la lectura de humedad.

Definición en la línea **109** del archivo **MGA.ino**.

#define ERROR_LIGHT 0x04

Error en la lectura de luz.

Definición en la línea **114** del archivo **MGA.ino**.

#define ERROR_NONE 0x00

Sin error en la lectura de sensores.

Definición en la línea **99** del archivo **MGA.ino**.

#define ERROR_TEMP 0x01

Error en la lectura de temperatura.

Definición en la línea **104** del archivo **MGA.ino**.

#define HUM_THRESHOLD 80

Umbral de humedad para activar la alarma.

Definición en la línea **77** del archivo **MGA.ino**.

#define LDRPIN 32

Pin analógico para el sensor de luz (LDR).

Definición en la línea **46** del archivo **MGA.ino**.

#define LED_BLUE 26

Pin de salida para el LED azul.

Definición en la línea **51** del archivo **MGA.ino**.

#define LIGHT_THRESHOLD 700

Umbral de luz para activar la alarma.

Definición en la línea **82** del archivo **MGA.ino**.

#define TEMP_THRESHOLD 26

Umbral de temperatura para activar la alarma.

Definición en la línea **72** del archivo **MGA.ino**.

#define TIME_TO_SLEEP 10

Tiempo en segundos antes de entrar en deep sleep.

Definición en la línea **93** del archivo **MGA.ino**.

#define uS_TO_S_FACTOR 1000000

Factor de conversión de microsegundos a segundos.

Definición en la línea **88** del archivo **MGA.ino**.

Documentación de funciones

void aggregatorTask (void * pvParameters)

Tarea de agregación de datos de sensores.

Recopila las lecturas de temperatura, humedad y luz, verifica errores y las envía a una cola.

Parámetros

pvParameters	Parámetro de entrada (no utilizado).

Definición en la línea **362** del archivo **MGA.ino**.

void buttonEventTask (void * pvParameters)

Tarea para manejar eventos de botones (contador).

Espera eventos en la cola de botones y actualiza un contador global.

Parámetros

pvParameters	Parámetro de entrada (no utilizado).

Definición en la línea **440** del archivo **MGA.ino**.

void controlLedTask (void * pvParameters)

Tarea para el control del LED (alarma) y el buzzer.

Recibe datos de la cola y activa o desactiva las alarmas según los umbrales y errores.

Parámetros

pvParameters	Parámetro de entrada (no utilizado).

Definición en la línea **401** del archivo **MGA.ino**.

void IRAM_ATTR decISR ()

Rutina de servicio de interrupción para el botón de decremento.

Implementa un debounce para evitar múltiples activaciones.

Definición en la línea **272** del archivo **MGA.ino**.

void deepSleepTask (void * pvParameters)

Tarea para iniciar Deep Sleep con timer y botón como despertador.

Configura el deep sleep tras esperar el tiempo definido.

Parámetros

pvParameters	Parámetro de entrada (no utilizado).

Definición en la línea **462** del archivo **MGA.ino**.

DHT dht (DHTPIN , DHTTYPE)

Objeto para el sensor DHT.

void IRAM_ATTR incISR ()

Rutina de servicio de interrupción para el botón de incremento.

Implementa un debounce para evitar múltiples activaciones.

Definición en la línea **252** del archivo **MGA.ino**.

void logError (const char * errorMsg)

Imprime mensajes de error con la fecha y hora actual del RTC.

Parámetros

errorMsg	Mensaje de error a imprimir.

Definición en la línea **207** del archivo **MGA.ino**.

void logWithTimestamp (const char * label, float value, const char * unit)

Imprime un mensaje con la fecha y hora actual del RTC.

Parámetros

label	Etiqueta o descripción del dato.
value	Valor a imprimir.
unit	Unidad del valor.

Definición en la línea **178** del archivo **MGA.ino**.

ThreeWire myWire (4 , 15 , 2)

Inicialización del bus de tres hilos para el RTC.

Se define el bus ThreeWire usando los pines DATOS, CLK y RST.

void print_wakeup_reason ()

Imprime la razón por la cual el ESP32 despertó del deep sleep.

Definición en la línea **230** del archivo **MGA.ino**.

void readHumidityTask (void * pvParameters)

Tarea de lectura de humedad con timestamp.

Lee la humedad del sensor DHT y la imprime con timestamp. Nota: Se ha incrementado el retardo a 3000 ms.

Parámetros

pvParameters	Parámetro de entrada (no utilizado).

Definición en la línea **320** del archivo **MGA.ino**.

void readLightTask (void * pvParameters)

Tarea de lectura de luz con timestamp.

Lee el valor analógico del sensor LDR y lo imprime con timestamp.

Parámetros

pvParameters	Parámetro de entrada (no utilizado).

Definición en la línea **344** del archivo **MGA.ino**.

void readTemperatureTask (void * pvParameters)

Tarea de lectura de temperatura con timestamp.

Lee la temperatura del sensor DHT y la imprime con timestamp.

Parámetros

pvParameters	Parámetro de entrada (no utilizado).

Definición en la línea **295** del archivo **MGA.ino**.

RtcDS1302< ThreeWire > Rtc (myWire)

void setup ()

Función de configuración.

Inicializa puertos, colas, interrupciones y tareas de FreeRTOS.

Definición en la línea **484** del archivo **MGA.ino**.

Documentación de variables

QueueHandle_t buttonQueue

Cola para eventos de botones.

Definición en la línea **149** del archivo **MGA.ino**.

volatile int counter = 0

Contador global modificado por botones.

Definición en la línea **155** del archivo **MGA.ino**.

const unsigned long debounceDelay = 500000

Retardo de debounce en microsegundos (500 ms).

Definición en la línea **162** del archivo **MGA.ino**.

volatile float g_humidity = 0.0f

Variable global para la humedad.

Definición en la línea **153** del archivo **MGA.ino**.

volatile int g_light = 0

Variable global para la lectura de luz.

Definición en la línea **154** del archivo **MGA.ino**.

volatile float g_temperature = 0.0f

Variable global para la temperatura.

Definición en la línea **152** del archivo **MGA.ino**.

volatile unsigned long lastDecTime = 0

Tiempo de la última interrupción para BUTTON_DEC.

Definición en la línea **161** del archivo **MGA.ino**.

volatile unsigned long lastIncTime = 0

Tiempo de la última interrupción para BUTTON_INC.

Definición en la línea **160** del archivo **MGA.ino**.

QueueHandle_t sensorQueue

Cola para enviar datos de sensores a tareas.

Definición en la línea **137** del archivo **MGA.ino**.

MGA.ino

Ir a la documentación de este archivo.

```
00001
00016
00017 #include <Arduino.h>
00018 #include <Wire.h>
00019 #include <Adafruit_Sensor.h>
00020 #include <DHT.h>
00021 #include <DHT U.h>
00022 #include <debounce.h>
00023 #include <RtcDS1302.h>
00024 #include "esp_sleep.h"  // Necesario para deep sleep
00025
00026 #if CONFIG_FREERTOS_UNICORE
00027 static const BaseType_t app_cpu = 0;
00028 #else
00029 static const BaseType_t app_cpu = 1;
00030 #endif
00031
00032 // ====================== CONFIGURACIÓN HARDWARE ======================
00036 #define DHTPIN      5
00037
00041 #define DHTTYPE     DHT22
00042
00046 #define LDRPIN      32
00047
00051 #define LED_BLUE    26
00052
00056 #define BUTTON_DEC  34
00057
00061 #define BUTTON_INC  33
00062
00066 #define BUZZER_PIN  14
00067
00068 // ====================== UMBRALES ======================
00072 #define TEMP_THRESHOLD   26
00073
00077 #define HUM_THRESHOLD    80
00078
00082 #define LIGHT_THRESHOLD  700
00083
00084 // ====================== CONFIGURACIÓN PARA DEEP SLEEP ======================
00088 #define uS_TO_S_FACTOR 1000000
00089
00093 #define TIME_TO_SLEEP  10
00094
00095 // ====================== DEFINICIÓN DE ERRORES ======================
00099 #define ERROR_NONE      0x00
00100
00104 #define ERROR_TEMP      0x01
00105
00109 #define ERROR_HUM       0x02
00110
00114 #define ERROR_LIGHT     0x04
00115
00116 // ====================== RTC DS1302 ======================
00122 ThreeWire myWire(4, 15, 2); // DATOS, CLK, RST
00123 RtcDS1302<ThreeWire> Rtc(myWire);
00124
00125 // ====================== ESTRUCTURA DE DATOS ======================
00130 struct SensorData {
00131   float temperature;
00132   float humidity;
00133   int   light;
00134   uint8_t errorFlags;
00135 };
00136
00137 QueueHandle_t sensorQueue;
00138
00139 // -------------------- ESTRUCTURA PARA EVENTOS DE BOTONES ----------------------
00144 struct ButtonEvent {
00145   int increment;
```

```
00146   char name[4];
00147 };
00148
00149 QueueHandle_t buttonQueue;
00150
00151 // --------------------- VARIABLES GLOBALES ---------------------
00152 volatile float g_temperature = 0.0f;
00153 volatile float g_humidity    = 0.0f;
00154 volatile int   g_light       = 0;
00155 volatile int   counter       = 0;
00156
00157 DHT dht(DHTPIN, DHTTYPE);
00158
00159 // --------------------- VARIABLES PARA DEBOUNCE ---------------------
00160 volatile unsigned long lastIncTime = 0;
00161 volatile unsigned long lastDecTime = 0;
00162 const unsigned long debounceDelay = 500000;
00163
00164 // --------------------- MACRO PARA EL TAMAÑO DE UN ARREGLO ---------------------
00168 #define countof(a) (sizeof(a) / sizeof(a[0]))
00169
00170 // --------------------- FUNCIÓN PARA IMPRIMIR CON TIMESTAMP ---------------------
00178 void logWithTimestamp(const char* label, float value, const char* unit)
00179 {
00180   RtcDateTime now = Rtc.GetDateTime();
00181   char datestring[26];
00182   snprintf_P(datestring,
00183              countof(datestring),
00184              PSTR("%02u/%02u/%04u %02u:%02u:%02u"),
00185              now.Month(),
00186              now.Day(),
00187              now.Year(),
00188              now.Hour(),
00189              now.Minute(),
00190              now.Second());
00191   Serial.print("[");
00192   Serial.print(datestring);
00193   Serial.print("] ");
00194   Serial.print(label);
00195   Serial.print(": ");
00196   Serial.print(value);
00197   Serial.print(" ");
00198   Serial.println(unit);
00199 }
00200
00201 // --------------------- FUNCIÓN PARA IMPRIMIR ERRORES CON TIMESTAMP
---------------------
00207 void logError(const char* errorMsg)
00208 {
00209   RtcDateTime now = Rtc.GetDateTime();
00210   char datestring[26];
00211   snprintf_P(datestring,
00212              countof(datestring),
00213              PSTR("%02u/%02u/%04u %02u:%02u:%02u"),
00214              now.Month(),
00215              now.Day(),
00216              now.Year(),
00217              now.Hour(),
00218              now.Minute(),
00219              now.Second());
00220   Serial.print("[");
00221   Serial.print(datestring);
00222   Serial.print("] [ERROR] ");
00223   Serial.println(errorMsg);
00224 }
00225
00226 // --------------------- FUNCIONES DE WAKEUP ---------------------
00230 void print_wakeup_reason(){
00231   esp_sleep_wakeup_cause_t wakeup_reason = esp_sleep_get_wakeup_cause();
00232   switch(wakeup_reason)
00233   {
00234     case ESP_SLEEP_WAKEUP_EXT0:
00235       Serial.println("Wakeup caused by external signal (ext0)");
00236       break;
00237     case ESP_SLEEP_WAKEUP_TIMER:
00238       Serial.println("Wakeup caused by timer");
00239       break;
```

```
00240     default:
00241       Serial.printf("Wakeup was not caused by deep sleep: %d\n", wakeup_reason);
00242       break;
00243   }
00244 }
00245
00246 // ---------------------- ISR PARA BOTÓN DE INCREMENTO ----------------------
00252 void IRAM_ATTR incISR()
00253 {
00254   unsigned long nowMicros = micros();
00255   if(nowMicros - lastIncTime < debounceDelay) return; // Evitar rebotes
00256   lastIncTime = nowMicros;
00257
00258   ButtonEvent evt = { +1, "INC" };
00259   BaseType_t xHigherPriorityTaskWoken = pdFALSE;
00260   xQueueSendFromISR(buttonQueue, &evt, &xHigherPriorityTaskWoken);
00261   if (xHigherPriorityTaskWoken) {
00262     portYIELD_FROM_ISR();
00263   }
00264 }
00265
00266 // ---------------------- ISR PARA BOTÓN DE DECREMENTO ----------------------
00272 void IRAM_ATTR decISR()
00273 {
00274   unsigned long nowMicros = micros();
00275   if(nowMicros - lastDecTime < debounceDelay) return; // Evitar rebotes
00276   lastDecTime = nowMicros;
00277
00278   ButtonEvent evt = { -1, "DEC" };
00279   BaseType_t xHigherPriorityTaskWoken = pdFALSE;
00280   xQueueSendFromISR(buttonQueue, &evt, &xHigherPriorityTaskWoken);
00281   if (xHigherPriorityTaskWoken) {
00282     portYIELD_FROM_ISR();
00283   }
00284 }
00285
00286 // ====================== TAREAS ======================
00287
00295 void readTemperatureTask(void *pvParameters)
00296 {
00297   while (1)
00298   {
00299     float temp = dht.readTemperature();
00300     if (!isnan(temp))
00301     {
00302       g_temperature = temp;
00303       logWithTimestamp("Temperature", temp, "°C");
00304     }
00305     else {
00306       logError("Fallo en lectura de temperatura.");
00307     }
00308     vTaskDelay(pdMS_TO_TICKS(2000));
00309   }
00310 }
00311
00320 void readHumidityTask(void *pvParameters)
00321 {
00322   while (1)
00323   {
00324     float hum = dht.readHumidity();
00325     if (!isnan(hum))
00326     {
00327       g_humidity = hum;
00328       logWithTimestamp("Humidity", hum, "%");
00329     }
00330     else {
00331       logError("Fallo en lectura de humedad.");
00332     }
00333     vTaskDelay(pdMS_TO_TICKS(3000));
00334   }
00335 }
00336
00344 void readLightTask(void *pvParameters)
00345 {
00346   while (1)
00347   {
00348     int lightVal = analogRead(LDRPIN);
```

```
00349     g_light = lightVal;
00350     logWithTimestamp("Light", (float)lightVal, "");
00351     vTaskDelay(pdMS_TO_TICKS(2000));
00352   }
00353 }
00354
00362 void aggregatorTask(void *pvParameters)
00363 {
00364   while (1)
00365   {
00366     SensorData data;
00367     data.temperature = g_temperature;
00368     data.humidity    = g_humidity;
00369     data.light       = g_light;
00370     data.errorFlags  = ERROR_NONE;
00371
00372     // Verifica errores en cada sensor
00373     if (isnan(data.temperature))
00374       data.errorFlags |= ERROR_TEMP;
00375     if (isnan(data.humidity))
00376       data.errorFlags |= ERROR_HUM;
00377     // Suponiendo que la lectura del LDR debe estar en el rango 0-4095 (para ESP32)
00378     if (data.light < 0 || data.light > 4095)
00379       data.errorFlags |= ERROR_LIGHT;
00380
00381     // Se informa el error con la hora del RTC si se detecta alguna falla
00382     if (data.errorFlags != ERROR_NONE)
00383     {
00384       char errorMsg[64];
00385       snprintf(errorMsg, sizeof(errorMsg), "Fallo en lectura de sensores. Flags:
%02X", data.errorFlags);
00386       logError(errorMsg);
00387     }
00388
00389     xQueueSend(sensorQueue, &data, portMAX_DELAY);
00390     vTaskDelay(pdMS_TO_TICKS(2000));
00391   }
00392 }
00393
00401 void controlLedTask(void *pvParameters)
00402 {
00403   SensorData data;
00404   while (1)
00405   {
00406     if (xQueueReceive(sensorQueue, &data, portMAX_DELAY))
00407     {
00408       // Si se detecta algún error en la trama, se evita la activación de la alarma
00409       if (data.errorFlags != ERROR_NONE)
00410       {
00411         logError("Trama con error. No se activa alarma.");
00412         digitalWrite(LED_BLUE, LOW);
00413         digitalWrite(BUZZER_PIN, LOW);
00414       }
00415       // Evaluación de umbrales en condiciones normales
00416       else if (data.temperature > TEMP_THRESHOLD ||
00417                data.humidity    > HUM_THRESHOLD  ||
00418                data.light       > LIGHT_THRESHOLD)
00419       {
00420         digitalWrite(LED_BLUE, HIGH);  // Enciende el LED
00421         digitalWrite(BUZZER_PIN, HIGH); // Activa el buzzer
00422         Serial.println("[ALERTA] Umbral superado: LED y Buzzer ACTIVADOS");
00423       }
00424       else
00425       {
00426         digitalWrite(LED_BLUE, LOW);   // Apaga el LED
00427         digitalWrite(BUZZER_PIN, LOW);   // Apaga el buzzer
00428       }
00429     }
00430   }
00431 }
00432
00440 void buttonEventTask(void *pvParameters)
00441 {
00442   ButtonEvent evt;
00443   while (1)
00444   {
00445     if (xQueueReceive(buttonQueue, &evt, portMAX_DELAY))
```

```
00446     {
00447       counter += evt.increment;
00448       char msg[64];
00449       snprintf(msg, sizeof(msg), "Button %s pressed, counter: %d", evt.name,
counter);
00450       logWithTimestamp(msg, 0, "");
00451     }
00452   }
00453 }
00454
00462 void deepSleepTask(void *pvParameters)
00463 {
00464   // Espera el tiempo definido antes de entrar en deep sleep
00465   vTaskDelay(pdMS_TO_TICKS(TIME_TO_SLEEP * 1000));
00466   Serial.println("Preparando para Deep Sleep...");
00467   Serial.flush();
00468
00469   // Configura el despertador por timer (TIME_TO_SLEEP segundos)
00470   esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
00471   // Configura ext0 para que, al detectar que BUTTON_INC pasa a LOW, despierte el
ESP32.
00472   esp_sleep_enable_ext0_wakeup((gpio_num_t)BUTTON_INC, 0);
00473
00474   // Entra en Deep Sleep
00475   esp_deep_sleep_start();
00476 }
00477
00478 // ====================== SETUP ======================
00484 void setup()
00485 {
00486   Serial.begin(115200);
00487   Wire.begin();
00488   dht.begin();
00489
00490   // Inicialización del RTC DS1302
00491   Rtc.Begin();
00492   Serial.print("compiled: ");
00493   Serial.print(_DATE_);
00494   Serial.println(_TIME_);
00495
00496   // Imprime la razón del wakeup
00497   print_wakeup_reason();
00498
00499   // Configuración de pines
00500   pinMode(LED_BLUE, OUTPUT);
00501   // Configuramos BUTTON_INC para que funcione con ext0: en_
```