ANIOT. Práctica 5.

Patricia Barrios
Daniel Calatayud

Modos de bajo consumo en ESP32

---

1. Crear una partición

Gracias a ESP-IDF vamos a crear una partición de tipo *data* y subtipo *fat* para poder montar un sistema de archivos FAT de un tamaño 1M. Declaramos las particiones en el archivo *partitions.csv*.

```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x6000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  app,  factory, 0x10000, 1M,
storage,  data, fat,     ,        1M,
```

Listing 1: Archivo partitions.csv

Además, es necesario incluir nuestro archivo en platform.ini el nombre y configurar desde *menuconfig* una partición personalizada como se muestra en la figura 1.

```
board_build.partitions = partitions.csv
```

```
(Top) → Partition Table
        Espressif IoT Development Framework Configuration
    Partition Table (Custom partition table CSV)  --->
(partitions.csv) Custom partition CSV file
(0x8000) Offset of partition table
[*] Generate an MD5 checksum for the partition table
```

Figura 1: Configuración de particiones con menuconfig

2. Redirigir los logs

Se opta por leer los valores recogidos por el *sensor hall* de nuestra ESP32 cada 10 segundos utilizando un *timer* periódico de alta resolución, mostrando el valor mediante el método

ESP_LOGI. Sin embargo, hemos redirigido la escritura de los *ESP_LOG** utilizando la función esp_log_set_vprintf, en la que tomaremos el mensaje y lo escribiremos en un archivo /spiflash/log.txt.[a]l que podremos acceder gracias a la partición anterior. Debemos tener en cuenta que no se pueden usar métodos de la familia *ESP_LOG** dentro de esta función ya que entraríamos en una recursión infinita.

También es importante aclarar que tras cada ejecución del programa el contenido del archivo *log.txt* se va resetear, gracias a los métodos de escritura 'wb' (*write and binary*) y 'a' (*append*).

Ayudándonos de otro *timer* auxiliar pero esta vez '*one-shot*', tras transcurrir 1 minuto, se parará la lectura de valores del *sensor hall*, se mostrarán en la terminal las primeras 5 líneas del archivo *log.txt* y se terminara la ejecución del desmontando la partición.

```
/**
 * Redirect logs to file
 */
int redirect_log(const char * message, va_list args) {
    FILE *f;
    printf("Redirecting log value \n");
    if(is_first_time) {
        is_first_time = false;
        f = fopen("/spiflash/log.txt", WRITE_FILE);
    } else {
        f = fopen("/spiflash/log.txt", APPEND_FILE);
    }
    if (f == NULL) {
        printf("Error openning file \n");
        return -1;
    }
    vfprintf(f, message, args);
    fclose(f);

  return 0;
}

    esp_log_set_vprintf(&redirect_log);
```

Listing 2: Redirección de logs

2.1. Ejemplo de la traza de la lectura de fichero de los *logs*

Sobre este sistema de ficheros vamos a registrar

```
Status triggered: write_log
Redirecting log value

Status triggered: write_log
Redirecting log value

Status triggered: write_log
Redirecting log value
```

```
Status triggered: write_log
Redirecting log value

Status triggered: write_log
Redirecting log value

Status triggered: write_log
Redirecting log value

Status triggered: read_log
Reading values from file
Read value: I (10000) P6: 22
Read value: I (20000) P6: 30
Read value: I (30000) P6: 31
Read value: I (40000) P6: 35
Read value: I (50000) P6: 29
```

3. Código

```
#include "customTypes.h"

esp_timer_handle_t periodic_timer_sensor_hall;
esp_timer_handle_t periodic_timer_reader;
int value_sensor = 0;
bool is_first_time = true;

static wl_handle_t s_wl_handle = WL_INVALID_HANDLE;
char *base_path = "/spiflash";    //Base path for FAT
const esp_vfs_fat_mount_config_t mount_config = {
    .max_files = 2,
    .format_if_mount_failed = true,
    .allocation_unit_size = CONFIG_WL_SECTOR_SIZE
    };

QueueHandle_t queueOut;

/**
 * Redirect logs to file
 */
int redirect_log(const char * message, va_list args) {
    FILE *f;
    printf("Redirecting log value \n");
    if(is_first_time) {
        is_first_time = false;
        f = fopen("/spiflash/log.txt", WRITE_FILE);
    } else {
        f = fopen("/spiflash/log.txt", APPEND_FILE);
    }
    if (f == NULL) {
        printf("Error openning file \n");
```

```
        return -1;
    }
    vfprintf(f, message, args);
    fclose(f);

  return 0;
}


/**
 * Send machine status
 */
static void send_machine_state(void *args)
{
    enum machine_status status = *(enum machine_status *)args;
    if (xQueueSendFromISR(queueOut, &status, 200) != pdTRUE) {
        ESP_LOGE(TAG, "ERROR: Could not put item on delay queue.");
    }
}

/**
 * Sensor hall handler
 */
static void sensor_hall_timer_callback(void *args)
{
    // Reading voltage on ADC1 channel 0 (GPIO 36):
    adc1_config_width(ADC_WIDTH_BIT_12);
    value_sensor = hall_sensor_read();
    enum machine_status status = WRITE_LOG;
    send_machine_state(&status);
}

/**
 * Reader handler
 */
static void reader_timer_callback(void *args)
{
    enum machine_status status = READ_LOG;
    send_machine_state(&status);
}

static void status_handler_task(void *args)
{
    FILE *f;
    const int bufferLength = 255;

    ESP_ERROR_CHECK(esp_vfs_fat_spiflash_mount(base_path, "storage", &
   mount_config, &s_wl_handle));
    esp_log_set_vprintf(&redirect_log);

    // handle events for different status
    enum machine_status status;
    while(1) {
        if(xQueueReceive(queueOut, &(status) , (TickType_t)  300) ==
```

```
    pdTRUE) {
85              printf("Status triggered: %s \n", machine_status_string[status
    ]);
86              switch (status)
87              {
88                  case WRITE_LOG:
89                  ESP_LOGI(TAG, "%d", value_sensor);
90                  break;
91
92                  case READ_LOG:
93                  printf("Reading values from file \n");
94                  //stop timers
95                  esp_timer_stop(periodic_timer_sensor_hall);
96                  esp_timer_stop(periodic_timer_reader);
97                  //read from logs.txt file
98                  f = fopen("/spiflash/log.txt", READ_FILE);
99                  if (f == NULL) {
100                     printf("Failed to open file for reading");
101                     return;
102                 }
103
104                 int i = 0;
105                 char bufferRead[bufferLength];
106                 while (fgets(bufferRead, bufferLength, f) && i <
    NUM_LOGS_READ) {
107                     printf("Read value: %s", bufferRead);
108                     i++;
109                 }
110                 fclose(f);
111
112                 break;
113             }
114         }
115         vTaskDelay(100 / portTICK_PERIOD_MS);
116     }
117     vTaskDelete(NULL);
118
119     ESP_ERROR_CHECK(esp_vfs_fat_spiflash_unmount(base_path, s_wl_handle));
120 }
121
122 void app_main(void)
123 {
124     /* ---------------------  STATUS MACHINE TASK
    --------------------------*/
125     /**
126      * 'machine_status' is an enum type defining all different states
    managed
127      */
128     queueOut = xQueueCreate( 10, sizeof( enum machine_status ));
129     xTaskCreate(&status_handler_task, "status_machine_handler_task", 3072,
     NULL, PRIORITY_STATUS_HANDLER_TASK, NULL);
130
131
132     /* -------------------------  SENSOR HALL
```

```
    ------------------------------*/
133     const esp_timer_create_args_t periodic_hall_timer_args = {
134         .callback = &sensor_hall_timer_callback,
135         .name = "periodic"};
136     esp_timer_create(&periodic_hall_timer_args, &
    periodic_timer_sensor_hall);
137     esp_timer_start_periodic(periodic_timer_sensor_hall,
    MICROS_PERIODIC_SENSOR_HALL);
138 
139     /* -------------------------  READER  ------------------------------
    */
140     const esp_timer_create_args_t one_shot_reader_timer_args = {
141         .callback = &reader_timer_callback,
142         .name = "one-shot"};
143     esp_timer_create(&one_shot_reader_timer_args, &periodic_timer_reader);
144     esp_timer_start_periodic(periodic_timer_reader, MICROS_CHRONO_READER);
145 }
```

Listing 3: main.c

```
1  #ifndef _custom_types_h
2  #define _custom_types_h
3  
4  #include <stdlib.h>
5  #include <stdio.h>
6  #include <time.h>
7  #include "freertos/FreeRTOS.h"
8  #include "esp_heap_task_info.h"
9  #include "esp_log.h"
10 #include "freertos/task.h"
11 #include "freertos/queue.h"
12 #include "driver/adc.h"
13 
14 #include "esp_vfs.h"
15 #include "esp_vfs_fat.h"
16 #include "esp_system.h"
17 
18 #define PRIORITY_STATUS_HANDLER_TASK 4          // min -->0, max --> 9
19 
20 #define MICROS_PERIODIC_SENSOR_HALL 10000000    // 10 seconds
21 #define MICROS_CHRONO_READER 60000000           // 1 minute
22 #define READ_FILE "rb"
23 #define WRITE_FILE "wb"
24 #define APPEND_FILE "a"
25 #define NUM_LOGS_READ 5
26 #define TAG "P6"
27 
28 enum machine_status{WRITE_LOG, READ_LOG};
29 static const char *machine_status_string[] = {
30     "write_log", "read_log"
31 };
32 
33 #endif
```

Listing 4: customTypes.h