Universidad Autónoma de Baja California

Facultad de Ciencias Químicas e Ingeniería



**SISTEMAS EMBEBIDOS**

**Práctica 1: GPIOs**

**Docente: Evangelina Lara Camacho**

**Alumno:** Gómez Cárdenas Emmanuel Alberto

**Matricula:** 01261509

## Objetivo

El alumno se familiarizará con el uso del GPIOs usando el sistema embebido ESP32 DevKit v1 para desarrollar aplicaciones para sistemas basados en microcontrolador para aplicarlos en la resolución de problemas de cómputo, de una manera eficaz y responsable.

## Equipo

Computadora personal con conexión a internet.

## Teoría

### Describa a detalle la función *gpio\_dump\_io\_configuration* para desplegar la configuración actual de GPIOs y el formato de su salida. Incluya un ejemplo.

El propósito de la función ***gpio\_dump\_io\_configuration*** es mostrar el estado y la configuración de cada pin, esto incluye la siguiente información:

* Numero de pin: Identificador del pin en el hardware
* Nombre del pin.
* Estado de resistencias: Configuración de las resistencias internas (Pull-up, pull-down, ninguna)
* Configuración actual del pin: Entrada, salida o modo alterno)
* Función alterna: Si aplica (I2C, UART, etc.)
* Nivel de voltaje.

La función se llama de esta forma **“gpio\_dump\_io\_configuration();” y c**uenta con dos parámetros

### out\_stream: Un puntero a tipo FILE en el cual se escribirá toda la información (Normalmente suele usarse stdout para imprimir en la salida estándar.

### io\_bit\_mask: un entero sin signo de 64 bits que es utilizado como mascara, cada bit es mapeado a un pin de entrada/salida.

### Ejemplos

* gpio\_dump\_io\_configuration(stdout, (1ULL << 4) | (1ULL << 18) | (1ULL << 26));
  + Imprime la configuración de los pines 4, 16 y 26 en la salida estandar
* gpio\_dump\_io\_configuration(stdout, GPIO\_NUM\_2);
  + Imprime solo la configuración del pin 2

# Desarrollo

### Entrega en el Laboratorio

Agregue un debouncer al ejemplo de uso de interrupciones de GPIO visto en clase

static void IRAM\_ATTR buttonInterruptHandler(void \*args) {

   uint32\_t buttonActioned = (uint32\_t)args;

   currentTime = xTaskGetTickCount() \* portTICK\_PERIOD\_MS;

   if (currentTime - lastStateChange < BUTTON\_BOUNCE\_TIME) {

      return;

   }

   lastStateChange = currentTime;

   xQueueSendFromISR(buttonQueueHandler, &buttonActioned, NULL);

}

### Entrega en las dos siguientes sesiones

Implemente en el **ESP32 ESP-IDF** una aplicación que representa un sistema de pago de uso de estacionamiento. La implementación debe ser eficiente en el uso de recursos de cómputo (procesador, memoria y periféricos).

## Parte 1

Una plaza comercial cobra 15 pesos por el uso de su estacionamiento. Cuando un conductor quiere retirar su vehículo, paga ese monto por medio de una máquina expendedora, ésta le provee el recibo de pago que el conductor después ingresa en la barrera de control vehicular para poder salir de la plaza. La máquina expendedora solo permite el pago con monedas de 1, 5, 10 y 20.

Diseñe e implemente el comportamiento de la máquina expendedora de recibos por medio de una **maquina de estados**. Algunos estados en lo que podría estar la máquina expendedora son

* Estado inicial donde todavía no se han insertado monedas.
* Estado donde se insertan monedas de 1 peso.
* Estado donde se insertan monedas de 5 pesos.
* Estado donde se insertan monedas de 10 pesos.
* Estado donde se insertan monedas de 20 pesos.
* Estado donde ya se tiene le pago completo y se expide el recibo.
* Estado donde se devuelve el cambio, esto cuando el conductor ingresa más de 15 pesos

Conecte cuatro botones LEDs al ESP32, cada botón representa el ingreso de monedas de 1, 5, 10 o 20 pesos. Un LED representa la expedición del recibo de pago.

## Parte 2

Diseñe e implemente el comportamiento de la barrera de control vehicular por medio de una maquina de estados. Las acciones que debe realizar son:

* Si hay un carro en espera de cruzar la barrera de control vehicular, ésta tiene que elevar la aguja hasta que esté en la posición de arriba.
* Una vez arriba, debe permanecer así hasta que el carro hay cruzado la barrera.
* Después de que el carro ha cruzado, la barrera de control vehicular tiene que bajar la aguja hasta la posición de abajo.

La aguja se desplaza en tres posiciones: abajo, en medio y arriba. De forma que cuando se eleva pasa por las posiciones de abajo, después en medio y finalmente, arriba. Cuando la aguja desciende, pasa por arriba, en medio y abajo.

Algunos estados en los que podría estar la barrera de control son:

* Estado inicial donde esta en espera de un vehículo.
* Estados donde se está elevando la aguja.
* Estados donde la aguja ya está en la posición de arriba y la barrera de control vehicular está en espera de que el carro pase.
* Estados donde está bajando la aguja

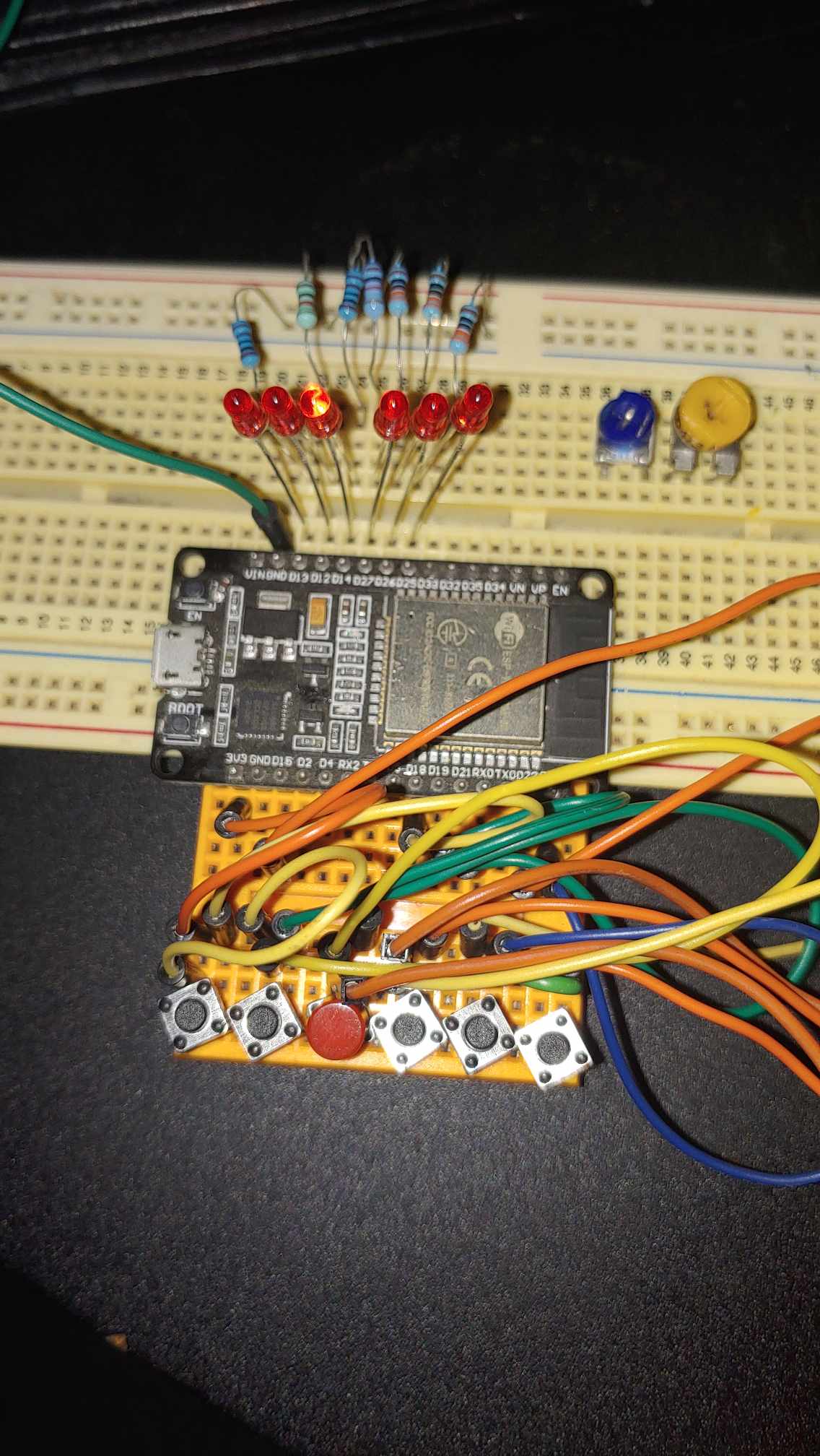
En su diseño, puede agregar o modificar los estados anteriores según lo considere necesario. **Incluya el diagrama de estados a su reporte.**

## Diagrama de estados (ambas partes mezcladas)

A computer screen shot of a flowchart

Description automatically generated

## Cableado



## Video en funcionamiento

A circuit board with wires and lights

Description automatically generated

[**Enlace al video en Drive**](https://drive.google.com/file/d/1eaWOV6_Hw9B_Jb3IOSf4hm4U7ztg_v8I/view?usp=sharing)

## Conclusiones y Comentarios

Aprender a utilizar correctamente lo que son las tareas, GPIOs y las interrupciones es una buena forma de empezar a trabajar con el ESP32 ya que esta práctica servirá como base para las demás.

## Dificultades en el Desarrollo

Entender correctamente el funcionamiento de las tareas y las interrupciones fue la parte más complicada de la práctica. Una vez que entendí como funcionan ya no tuve tantos bloqueos.

## Referencias

<https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/index.html>

Código  
El código fuente puede ser encontrado en el [Repositorio de GitHub “Sistemas Embebidos”](https://github.com/AlbGmx/SistemasEmbebidos)

#include "driver/gpio.h"

#include "esp\_log.h"

#include "freertos/FreeRTOS.h"

#include "freertos/queue.h"

#include "freertos/task.h"

#define TAG "ParkingMachine"

#define PARKING\_PRICE 15

#define LED\_DELAY\_ON 500

#define LED\_DELAY\_OFF LED\_DELAY\_ON / 2

#define PRINTING\_DELAY 2000

#define BUTTON\_BOUNCE\_TIME 150

#define RELEASED 0

#define PRESSED 1

#define MINIMUM\_DELAY\_MS 10

// LOGGING

#define LOG\_DELAY 5       // Time Between Logs in Seconds

#define LOG\_STATES false  // Set to false to disable state logging

// LEDS

#define LED\_CHANGE\_1\_PESO GPIO\_NUM\_13

#define LED\_CHANGE\_5\_PESOS GPIO\_NUM\_12

#define LED\_CHANGE\_10\_PESOS GPIO\_NUM\_14

#define LED\_PRINTING\_RECEIPT GPIO\_NUM\_2

#define LED\_NEEDLE\_DOWN GPIO\_NUM\_27

#define LED\_NEEDLE\_MIDDLE GPIO\_NUM\_26

#define LED\_NEEDLE\_UP GPIO\_NUM\_25

// BUTTONS

#define BUTTON\_1\_PESO GPIO\_NUM\_18

#define BUTTON\_5\_PESOS GPIO\_NUM\_19

#define BUTTON\_10\_PESOS GPIO\_NUM\_21

#define BUTTON\_20\_PESOS GPIO\_NUM\_22

#define BUTTON\_CAR\_CROSSED GPIO\_NUM\_23

#define BUTTON\_CAR\_PRESENT GPIO\_NUM\_15

typedef enum {

   STATE\_INITIAL = 0,

   STATE\_WAITING\_FOR\_CAR,

   STATE\_RECEIVING\_MONEY,

   STATE\_RETURNING\_CHANGE,

   STATE\_WAITING\_CHANGE,

   STATE\_RECEIPT,

   STATE\_ELEVATING\_NEEDLE,

   STATE\_CAR\_CROSSING,

   STATE\_DESCENDING\_NEEDLE

} states\_t;

// Global variables

states\_t currentState = STATE\_INITIAL;

uint8\_t change = 0;

uint8\_t totalAmount = (uint8\_t)-1;

int32\_t lastStateChange = 0;

QueueHandle\_t buttonQueueHandler;

uint32\_t currentTime = 0;

bool areInterruptsAttached = false;

bool giveChange = false;

bool changeGiven = false;

bool isPressed = false;

void delayMillis(int millis) { vTaskDelay(millis / portTICK\_PERIOD\_MS); }

void configGPIOs() {

   gpio\_config\_t io\_conf;

   io\_conf.intr\_type = GPIO\_INTR\_DISABLE;

   io\_conf.mode = GPIO\_MODE\_OUTPUT;

   io\_conf.pin\_bit\_mask = (1 << LED\_CHANGE\_1\_PESO) | (1 << LED\_CHANGE\_5\_PESOS)

| (1 << LED\_CHANGE\_10\_PESOS) | (1 << LED\_NEEDLE\_DOWN)

| (1 << LED\_NEEDLE\_MIDDLE) | (1 << LED\_NEEDLE\_UP)

| (1 << LED\_PRINTING\_RECEIPT);

   io\_conf.pull\_down\_en = 0;

   io\_conf.pull\_up\_en = 0;

   gpio\_config(&io\_conf);

   io\_conf.intr\_type = GPIO\_INTR\_NEGEDGE;

   io\_conf.mode = GPIO\_MODE\_INPUT;

   io\_conf.pin\_bit\_mask = (1 << BUTTON\_1\_PESO) | (1 << BUTTON\_5\_PESOS)

| (1 << BUTTON\_10\_PESOS) | (1 << BUTTON\_20\_PESOS)

| (1 << BUTTON\_CAR\_CROSSED) | (1 << BUTTON\_CAR\_PRESENT);

   io\_conf.pull\_down\_en = 1;

   io\_conf.pull\_up\_en = 0;

   gpio\_config(&io\_conf);

}

static void IRAM\_ATTR buttonInterruptHandler(void \*args) {

   uint32\_t buttonActioned = (uint32\_t)args;

   currentTime = xTaskGetTickCount() \* portTICK\_PERIOD\_MS;

   if (currentTime - lastStateChange < BUTTON\_BOUNCE\_TIME) {

      return;

   }

   lastStateChange = currentTime;

   xQueueSendFromISR(buttonQueueHandler, &buttonActioned, NULL);

}

void parkingMachineTask(void \*args) {

   int pinNumber;

   while (true) {

      if (xQueueReceive(buttonQueueHandler, &pinNumber, portMAX\_DELAY)) {

         if (gpio\_get\_level(pinNumber) == RELEASED) {

            switch (currentState) {

               case STATE\_WAITING\_FOR\_CAR:

                  if (pinNumber == BUTTON\_CAR\_PRESENT) {

                     currentState = STATE\_RECEIVING\_MONEY;

                     ESP\_LOGW(TAG, "Automovil detectado, esperando dinero\n");

                  } else {

                     ESP\_LOGE(TAG, "No se puede recibir dinero hasta que se detecte un automovil\n");

                  }

                  break;

               case STATE\_RECEIVING\_MONEY:

                  switch (pinNumber) {

                     case BUTTON\_1\_PESO:

                        totalAmount += 1;

                        ESP\_LOGW(TAG, "Se han depositado 1 peso, total: %d\n", totalAmount);

                        break;

                     case BUTTON\_5\_PESOS:

                        totalAmount += 5;

                        ESP\_LOGW(TAG, "Se han depositado 5 pesos, total: %d\n", totalAmount);

                        break;

                     case BUTTON\_10\_PESOS:

                        totalAmount += 10;

                        ESP\_LOGW(TAG, "Se han depositado 10 pesos, total: %d\n", totalAmount);

                        break;

                     case BUTTON\_20\_PESOS:

                        totalAmount += 20;

                        ESP\_LOGW(TAG, "Se han depositado 20 pesos, total: %d\n", totalAmount);

                        break;

                     default:

                        ESP\_LOGE(TAG, "Operación no autorizada\n");

                        break;

                  }

                  if (totalAmount >= PARKING\_PRICE) {

                     currentState = STATE\_RETURNING\_CHANGE;

                     ESP\_LOGW(TAG, "Dinero suficiente, devolviendo %d de cambio\n", totalAmount - PARKING\_PRICE);

                  }

                  break;

case STATE\_CAR\_CROSSING:

                  if (pinNumber == BUTTON\_CAR\_CROSSED) {

                     currentState = STATE\_DESCENDING\_NEEDLE;

                     ESP\_LOGW(TAG, "Automovil cruzó, bajando aguja\n");

                  }

                  break;

               default:

                  ESP\_LOGE(TAG, "Operación no autorizada\n");

                  break;

            }

         }

      }

      delayMillis(MINIMUM\_DELAY\_MS);

   }

}

void configInterruptions() {

   buttonQueueHandler = xQueueCreate(10, sizeof(uint32\_t));

   xTaskCreate(parkingMachineTask, "ParkingMachineTask", 2048, NULL, 1, NULL);

   gpio\_install\_isr\_service(0);

   gpio\_isr\_handler\_add(BUTTON\_1\_PESO, buttonInterruptHandler,

(void \*)BUTTON\_1\_PESO);

   gpio\_isr\_handler\_add(BUTTON\_5\_PESOS, buttonInterruptHandler,

(void \*)BUTTON\_5\_PESOS);

   gpio\_isr\_handler\_add(BUTTON\_10\_PESOS, buttonInterruptHandler,

(void \*)BUTTON\_10\_PESOS);

   gpio\_isr\_handler\_add(BUTTON\_20\_PESOS, buttonInterruptHandler,

(void \*)BUTTON\_20\_PESOS);

   gpio\_isr\_handler\_add(BUTTON\_CAR\_CROSSED, buttonInterruptHandler,

(void \*)BUTTON\_CAR\_CROSSED);

   gpio\_isr\_handler\_add(BUTTON\_CAR\_PRESENT, buttonInterruptHandler,

(void \*)BUTTON\_CAR\_PRESENT);

}

void indicateReturnWithLedsTask(void \*param) {

   while (true) {

      if (giveChange == true) {

         gpio\_set\_level(LED\_CHANGE\_1\_PESO, 0);

         gpio\_set\_level(LED\_CHANGE\_5\_PESOS, 0);

         gpio\_set\_level(LED\_CHANGE\_10\_PESOS, 0);

         gpio\_set\_level(LED\_PRINTING\_RECEIPT, 0);

         while (change > 0) {

            if (change >= 10) {

               gpio\_set\_level(LED\_CHANGE\_10\_PESOS, 1);

               delayMillis(LED\_DELAY\_ON \* 2);

               gpio\_set\_level(LED\_CHANGE\_10\_PESOS, 0);

               delayMillis(LED\_DELAY\_OFF \* 2);

               change -= 10;

            } else if (change >= 5) {

               gpio\_set\_level(LED\_CHANGE\_5\_PESOS, 1);

               delayMillis(LED\_DELAY\_ON \* 2);

               gpio\_set\_level(LED\_CHANGE\_5\_PESOS, 0);

               delayMillis(LED\_DELAY\_OFF \* 2);

               change -= 5;

            } else if (change >= 1) {

               gpio\_set\_level(LED\_CHANGE\_1\_PESO, 1);

               delayMillis(LED\_DELAY\_ON \* 2);

               gpio\_set\_level(LED\_CHANGE\_1\_PESO, 0);

               delayMillis(LED\_DELAY\_OFF \* 2);

               change -= 1;

            }

         }

         ESP\_LOGW(TAG, "Cambio devuelto\n");

         giveChange = false;

         changeGiven = true;

      }

      delayMillis(MINIMUM\_DELAY\_MS);

   }

}

void printReceivedReceipt() {

   gpio\_set\_level(LED\_PRINTING\_RECEIPT, 1);

   delayMillis(LED\_DELAY\_ON);

   gpio\_set\_level(LED\_PRINTING\_RECEIPT, 0);

   delayMillis(LED\_DELAY\_OFF);

   ESP\_LOGW(TAG, "Recibo impreso, subiendo aguja\n");

}

void logCurrrentStateTask() {

   while (true) {

      switch (currentState) {

         case STATE\_INITIAL:

            ESP\_LOGI(TAG, "State %d -> Parking Machine Started\n", currentState);

            break;

         case STATE\_WAITING\_FOR\_CAR:

            ESP\_LOGI(TAG, "State %d -> Waiting for car\n", currentState);

            break;

         case STATE\_RECEIVING\_MONEY:

            ESP\_LOGI(TAG, "State %d -> Receiving money\n", currentState);

            break;

         case STATE\_RETURNING\_CHANGE:

            ESP\_LOGI(TAG, "State %d -> Returning change\n", currentState);

            break;

         case STATE\_WAITING\_CHANGE:

            ESP\_LOGI(TAG, "State %d -> Waiting for change\n", currentState);

            break;

         case STATE\_RECEIPT:

            ESP\_LOGI(TAG, "State %d -> Printing receipt\n", currentState);

            break;

         case STATE\_ELEVATING\_NEEDLE:

            ESP\_LOGI(TAG, "State %d -> Elevating needle\n", currentState);

            break;

         case STATE\_CAR\_CROSSING:

            ESP\_LOGI(TAG, "State %d -> Car crossing\n", currentState);

            break;

         case STATE\_DESCENDING\_NEEDLE:

            ESP\_LOGI(TAG, "State %d -> Descending needle\n", currentState);

            break;

         default:

            ESP\_LOGI(TAG, "State %d -> Unknown State :/\n", currentState);

            break;

      }

      delayMillis(LOG\_DELAY \* 1000);

   }

}

void app\_main() {

   configGPIOs();

   configInterruptions();

   xTaskCreate(indicateReturnWithLedsTask, "IndicateReturnWithLedsTask", 2048,

NULL, 1, NULL);

   if (LOG\_STATES) xTaskCreate(logCurrrentStateTask, "LogCurrrentStateTask",

2048, NULL, 1, NULL);

while (true) {

      switch (currentState) {

         case STATE\_INITIAL:

            if (totalAmount == (uint8\_t)-1) {

               totalAmount = 0;

               gpio\_set\_level(LED\_NEEDLE\_DOWN, 1);

               ESP\_LOGW(TAG, "Maquina inicializada\n");

            }

            currentState = STATE\_WAITING\_FOR\_CAR;

            break;

         case STATE\_RETURNING\_CHANGE:

            change = totalAmount - PARKING\_PRICE;

            giveChange = true;

            currentState = STATE\_WAITING\_CHANGE;

            break;

         case STATE\_WAITING\_CHANGE:

            if (changeGiven) {

               currentState = STATE\_RECEIPT;

               changeGiven = false;

            }

            break;

         case STATE\_RECEIPT:

            printReceivedReceipt();

            totalAmount = -1;

            currentState = STATE\_ELEVATING\_NEEDLE;

            break;

         case STATE\_ELEVATING\_NEEDLE:

            gpio\_set\_level(LED\_NEEDLE\_MIDDLE, 1);

            gpio\_set\_level(LED\_NEEDLE\_DOWN, 0);

            delayMillis(1000);

            gpio\_set\_level(LED\_NEEDLE\_UP, 1);

            gpio\_set\_level(LED\_NEEDLE\_MIDDLE, 0);

            currentState = STATE\_CAR\_CROSSING;

            break;

         case STATE\_DESCENDING\_NEEDLE:

            gpio\_set\_level(LED\_NEEDLE\_MIDDLE, 1);

            gpio\_set\_level(LED\_NEEDLE\_UP, 0);

            delayMillis(1000);

            gpio\_set\_level(LED\_NEEDLE\_DOWN, 1);

            gpio\_set\_level(LED\_NEEDLE\_MIDDLE, 0);

            currentState = STATE\_INITIAL;

            break;

         default:

            break;

      }

      delayMillis(MINIMUM\_DELAY\_MS);

      currentTime = xTaskGetTickCount() \* portTICK\_PERIOD\_MS;

   }

}