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.

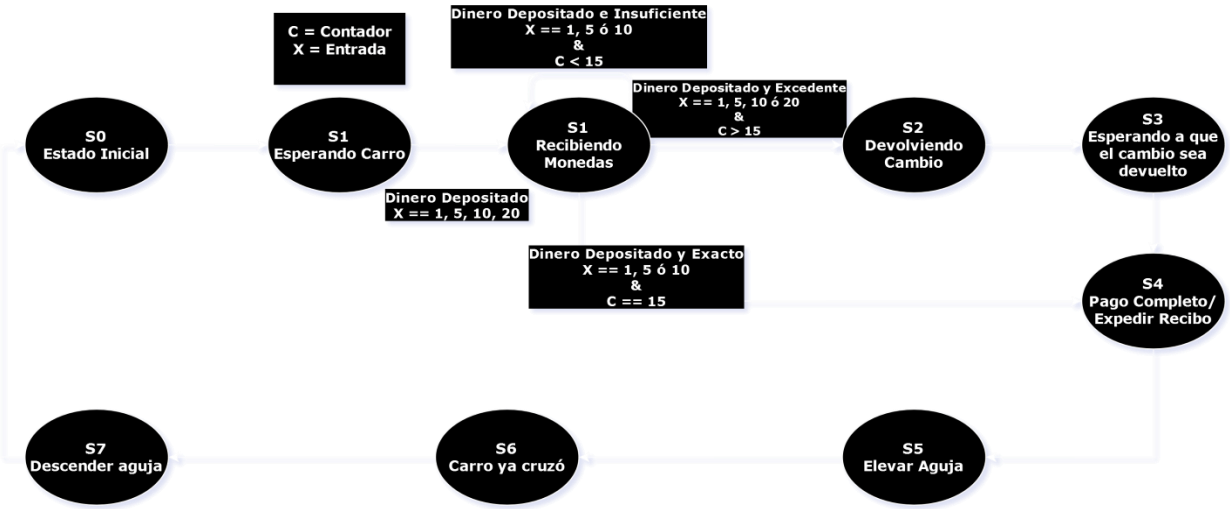
La aguja se desplaza en tres posiciones: abajo, en medio y arriba. De forma que cuando se eleva pasa por las posiciones de abajo, después en medio y finalmente, arriba. Cuando la aguja desciende, pasa por arriba, en medio y abajo.

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

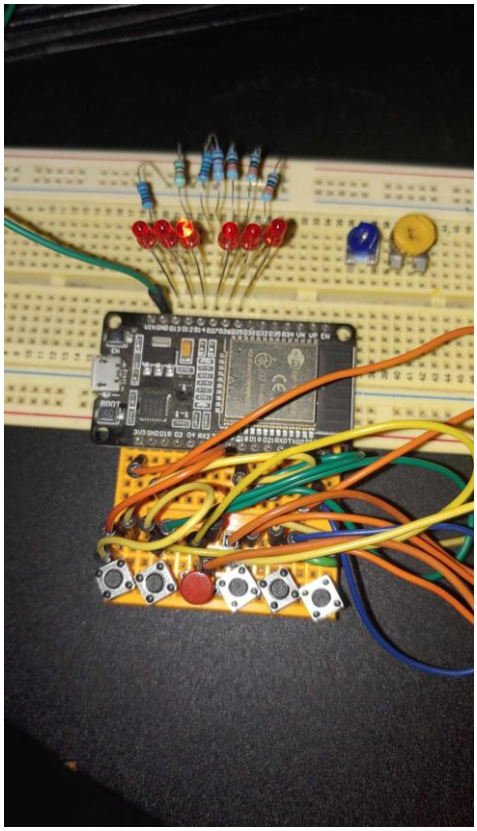
- Estado inicial donde esta en espera de un vehículo.
- Estados donde se está elevando la aguja.
- Estados donde la aguja ya está en la posición de arriba y la barrera de control vehicular está en espera de que el carro pase.
- Estados donde está bajando la aguja

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

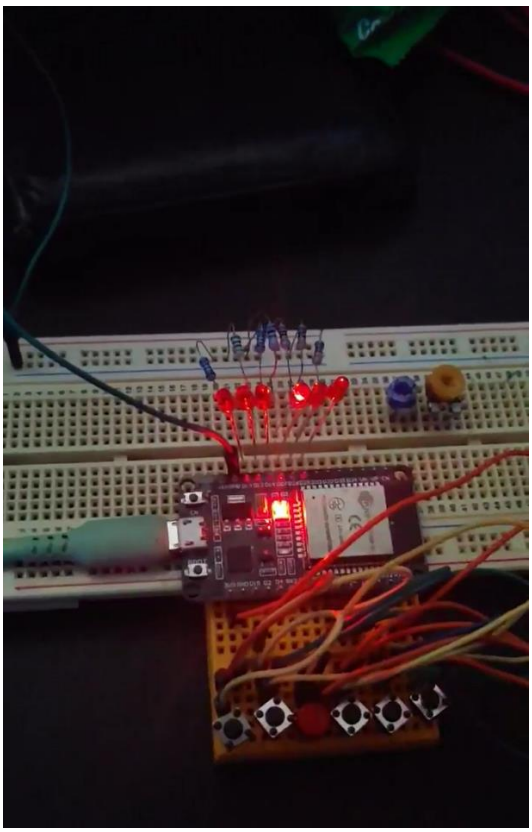
Diagrama de estados (ambas partes mezcladas)



Cableado



Video en funcionamiento



Enlace al video en Drive

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”]

```
#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;
   }
}
```