Amsterdam University of Applied Sciences

FACULTY OF DIGITAL MEDIA AND CREATIVE INDUSTRIES
HBO – Information and Technology

TOYCAR

Embedded Systems 2

Jack Zwuup

November 22, 2023

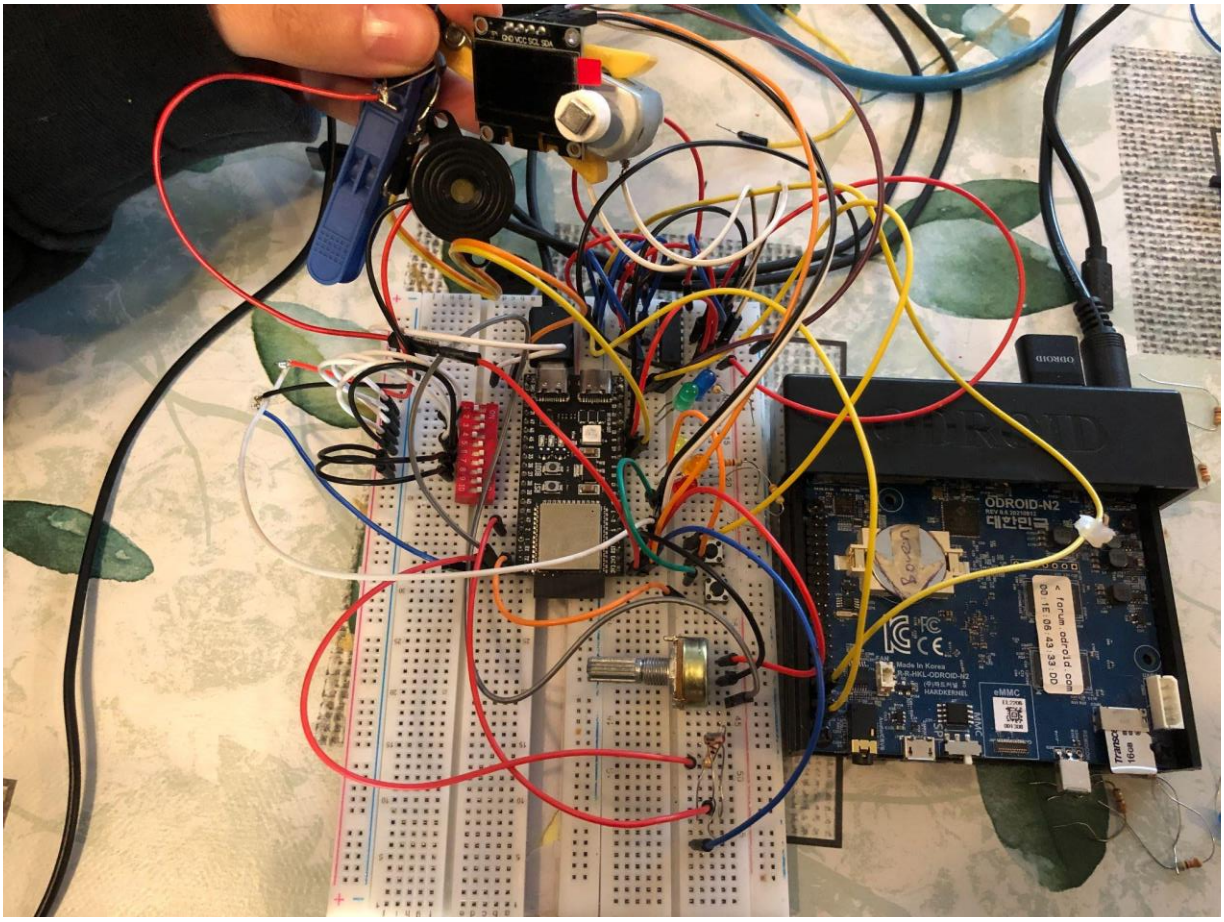


Figure 1: Photograph of the setup. The hall effect sensor is near the tip of my thumb nail.

0 Introduction

I have been instructed to make a simulation of a toy car by my teacher. I got 7 exercises where I have to work on specific parts of the car.

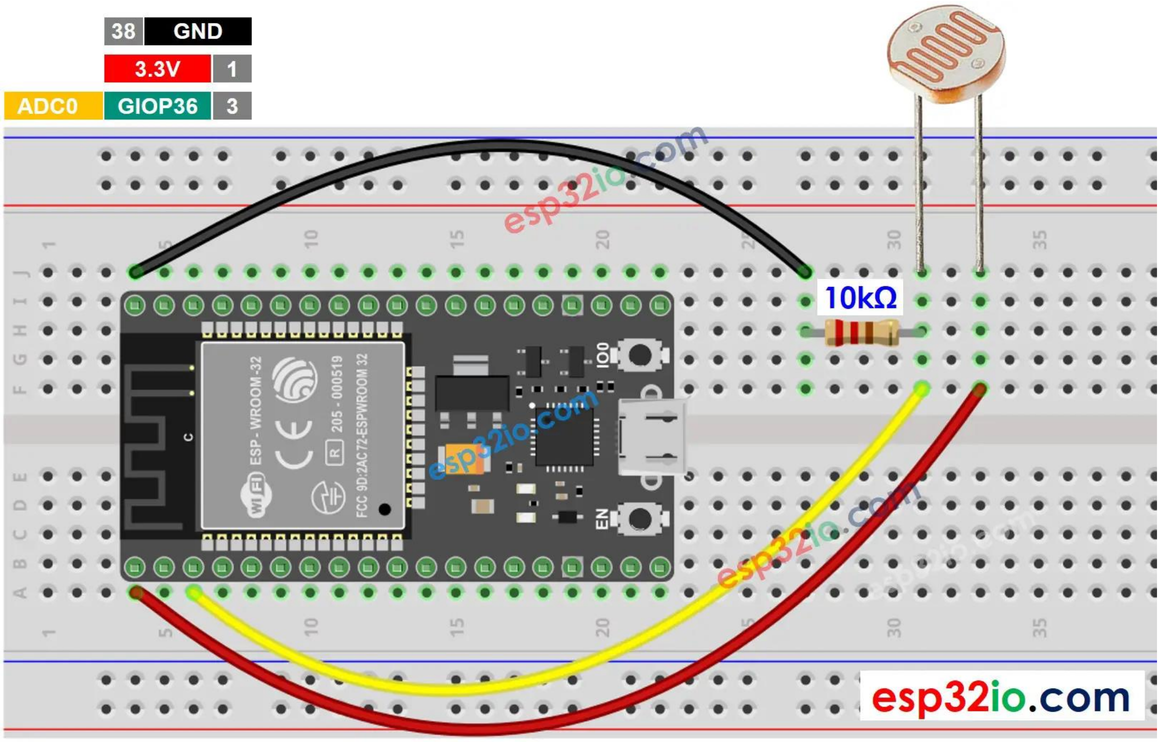
I have to use:

1. A light–dependent resistor (LDR) in exercise 1.
2. A DIP switch (DIP) in , a DC motor (MOT), an L293D motor driver (DRV), a potentiometer (POT) and auxiliary LEDs (LED) in exercise 2.
3. An I2C SSD1306 OLED display (DSP) in exercise 3.
4. Auxiliary LEDs and DIP switch (DIP) exercise 4.
5. Auxiliary push button (BTN) in exercise 5.
6. Auxiliary push button (BTN) and a buzzer (BZR) in exercise 6.
7. A Hall effect sensor (HAL) and a cube magnet (MGN) in exercise 7.

Disclaimer, when I use a picture to show the connection scheme, the gpio numbers will not be exactly the same as in the picture, but the overall principle will be the same.

1 LDR

You will use the light–dependent resistor (LDR), seen in Figure 3 (a), to simulate the ignition key of the car. You have to define a threshold indicating the level of light deciding whether the key is present or absent. As expected, the car only works when the key is present; otherwise, the car is OFF.

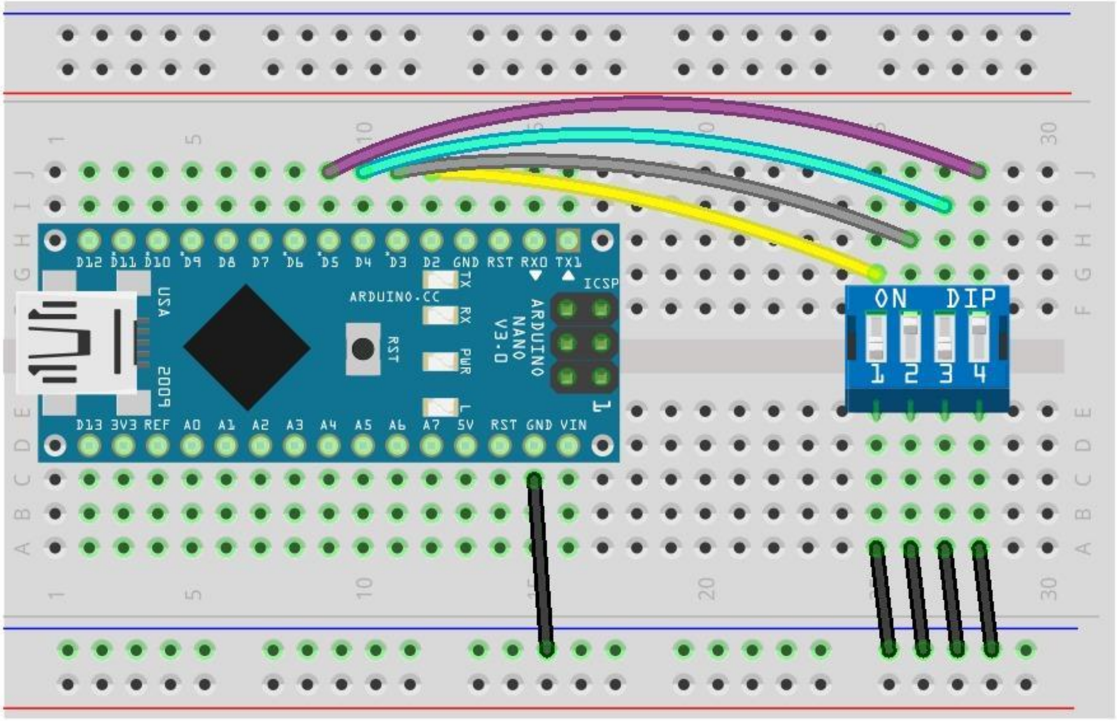


To do this, firstly I connected the ldr to a 3.3 V pin, a ground pin and a signal pin. Secondly I tested the limits of the ldr and determined a reasonable threshold based on it. Thirtly I found out a method to determine whether the key of the car is absent or not. Finally I have connected a led as indication for the absence of the key.

2 DIP switch gear

You will use a section on the DIP switch (DIP), seen in Figure 3 (b), to simulate the gear box of your car. The car can be in 4 possible states:

- OFF: The car is OFF and nothing is working.
- PARK: The car is going to park. You must use 2 LEDs, seen in Figure 3 (f), that will blink (at your desired frequency) for 10 times and then they will go OFF.
- DRIVE: The car is moving forward. The motor (MOT), seen in Figure 3 (c), rotates clockwise at a certain speed, given by the potentiometer (POT), seen in Figure 3 (d). The more you increase POT, the higher the speed. Use the motor driver (DRV), seen in Figure 3 (j), to set the direction of the rotation.
- BACK: The motor (MOT) rotates counterclockwise at a single predefined speed. Use the motor driver (DRV) to set the direction of the rotation.


D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 GND RST RX0 TX1
ARDUINO.CC
ARDUINO NANO V3.0
ICSP
USA
2009
TX
RX
PWR
L
RST
D13 3V3 REF A0 A1 A2 A3 A4 A5 A6 A7 5V RST GND VIN
ON DIP
1 2 3 4

Firstly I have connected the dip switch to the signal pins, secondly I have connected the leds, thirdly I connected all the grounds and vcc's together of the driver. I have also connected the motor with the motor driver to the signal pins and I have used the Odroid as external power source. Fourthly, I have connected the potmeter.

I have thought about a switch case for the dip switch, because it sounds funny, so I thought of bit shifting to decide the states.

I figured out that off is of no matter if certain states are 1

BackState	DriveState	ParkState	OffState
0 or 1	0 or 1	0 or 1	0

If OffState has become OnState by making it 1 and one of these states is 1 that state is activated.

BackState	DriveState	ParkState	OffState
0	0	1	1

If OffState has become OnState by making it 1 and more than one of these states is 1 an error will occur.

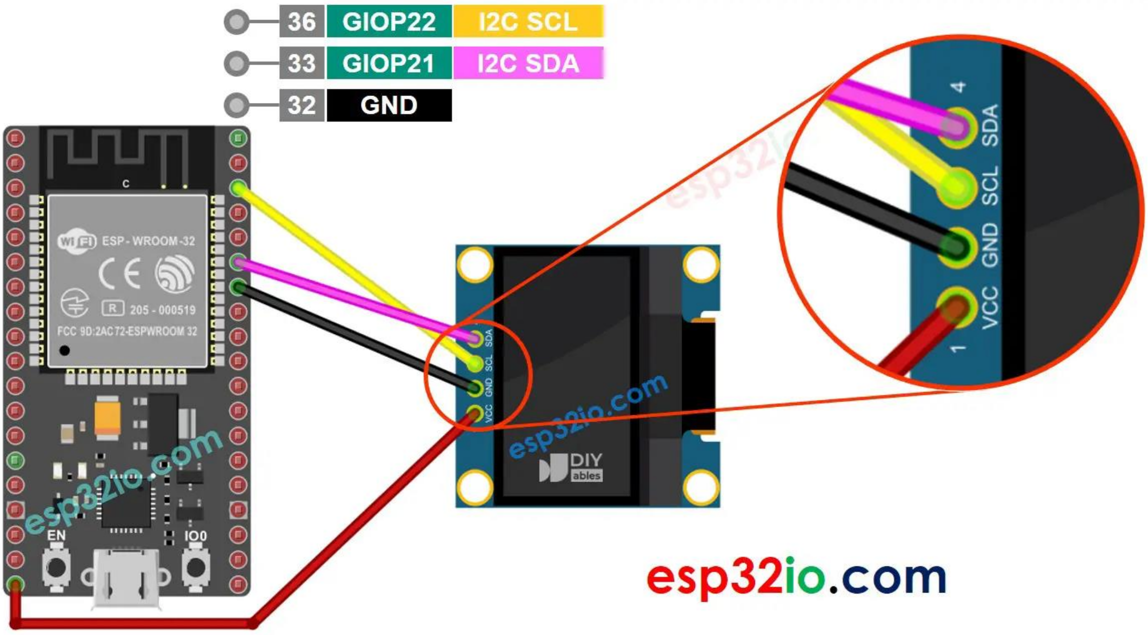
BackState	DriveState	ParkState	OffState	ERROR
0	1	1	1	

I was really stuck on going from parking state to off state, until I realized you only can park when you are driving beforehand, so the blinking count gets reset when you drive, so you can park.

The motor worked fine and spinned immediately CW and CCW after a few days of trying to connect it with the motor driver. Connecting a potentiometer directly to the motor didn't work, so I have worked around it and now it works fine.

3 DSP

You will use the screen (DSP), seen in Figure 3 (e), to show some important information of the car, for example: the state of the gearbox (DIP), the frequency and the speed of the motor rotation, additional state of the LEDs, or any other information that you may consider important.



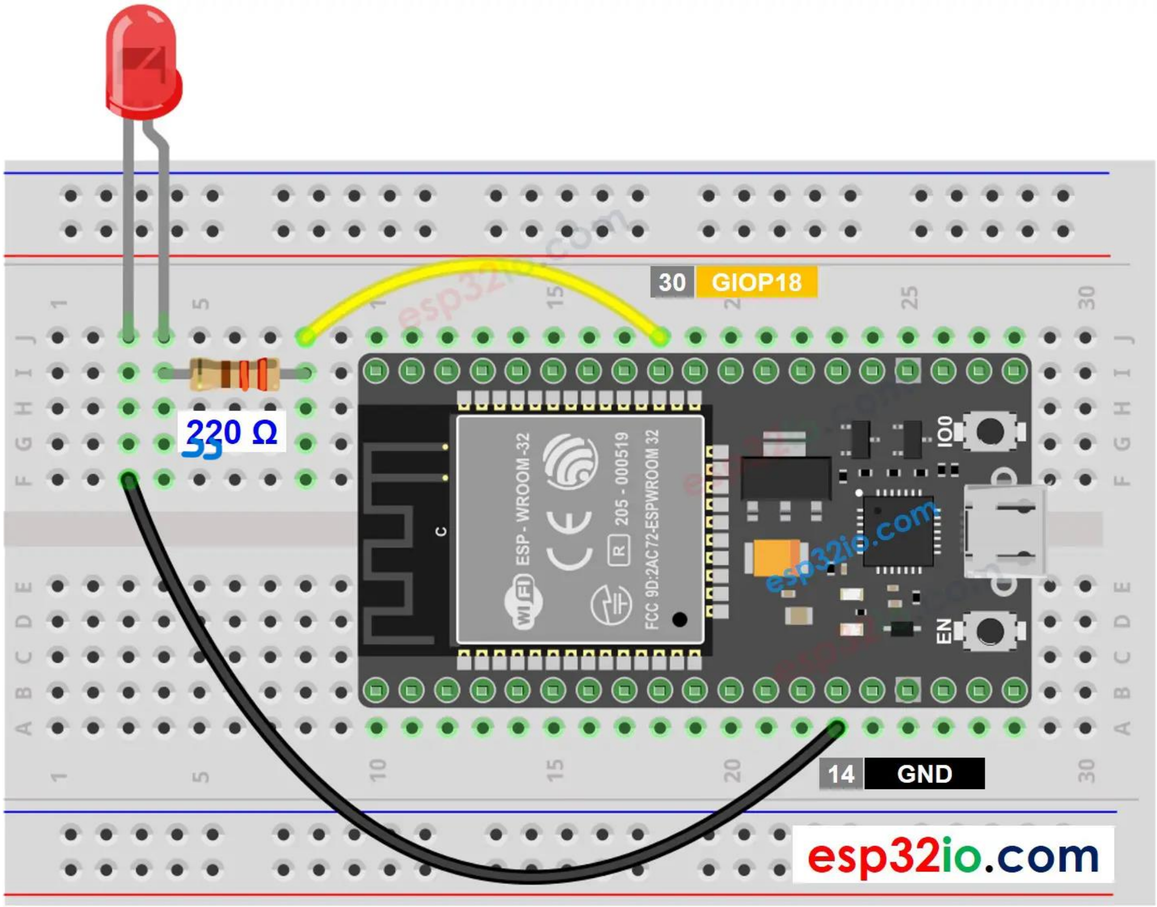
I connected the scl to the defined pin, same with sda. I connected the vcc pin to 3.3 volts and the gnd to the ground.

This was extremely hard. I tried several library's (lvgl, u8g2 and adafruit), but they didn't work. I tried the ssd1306.h library and this one did work. I implemented this when I already finished exercise 6. I decided to show the state of the dip switch, states of the car itself, and approximate speed of the motor.

4 DIP switch other section

You will use another section on the DIP switch (DIP), seen in Figure 3 (b), to simulate the controls of the directional lights. The lights can be in three different options:

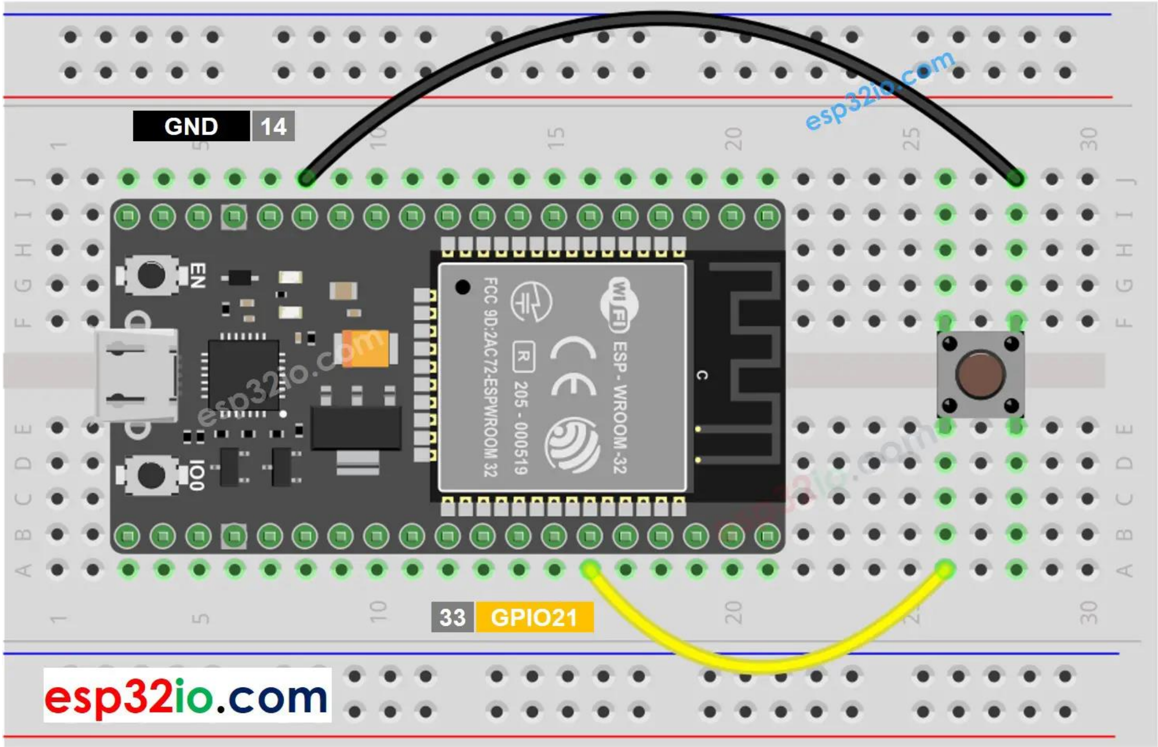
- LEFT: The car is turning to the left. You should blink (at your desired frequency) the corresponding directional LED.
- RIGHT: The car is turning to the right. You should blink (at your desired frequency) the corresponding directional LED.
- NONE: The car is not turning. You should not blink any directional LED.



This was almost the same as exercise 2, but easier, because you only have to use leds. I only had to realize that this is probably separate from the other states of exercise 2, except for off state, because off is off.

5 BTN

You will use a push button (BTN), seen in Figure 3 (i), to simulate the brake pedal. When the pedal is pressed, the car will stop. You must use an interrupt to simulate this process.

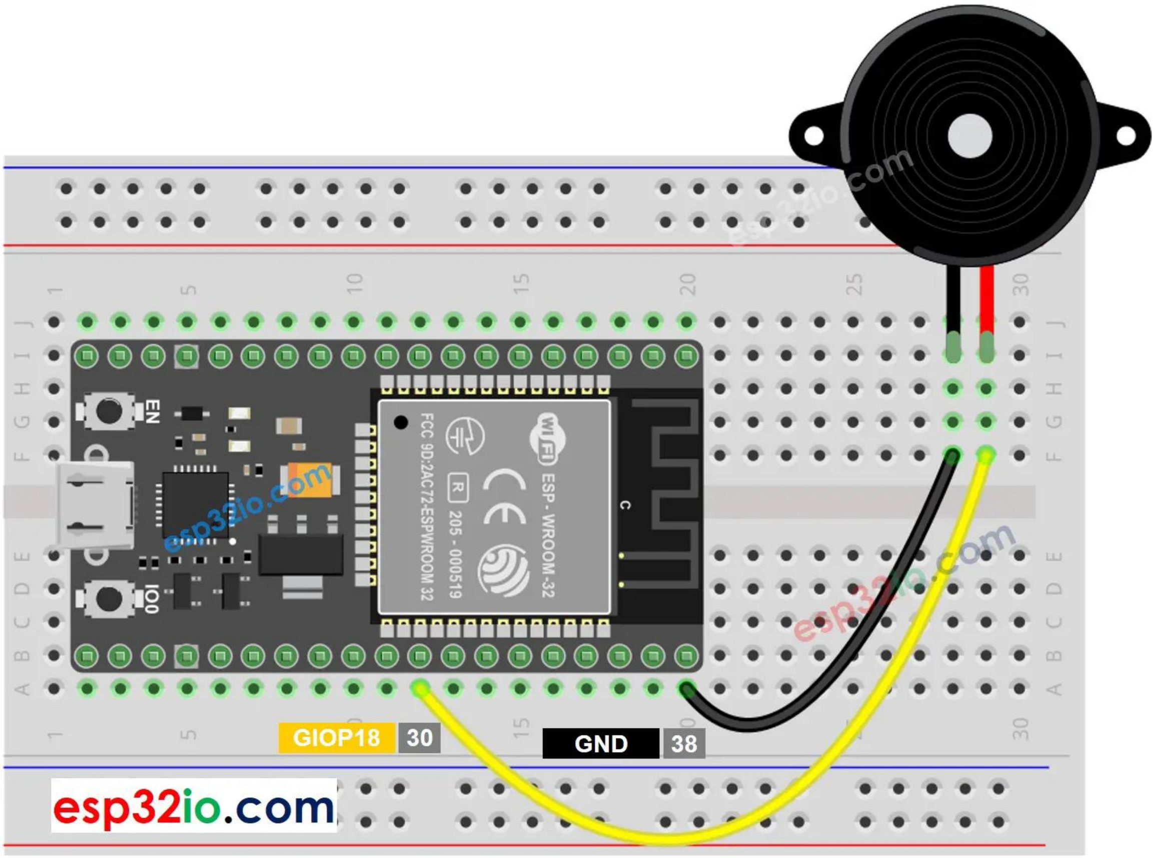


I connected the button to a signal pin and the other side to a grd pin.

I have searched the internet randomly to make an interrupt. It was quite easy. When it was working, I removed the garbage code, because it wasn't necessary to use.

6 BTN BZR

You will use another push button (BTN) to activate/deactivate the horn of the car. You will use a buzzer (BZR), seen in Figure 3 (h), as the car horn. You have to use an interrupt to simulate this process.



I connected the buzzer with the red wire to the signal pin and the black wire to the ground. I also connected another button to a signal pin and the other side to a grd pin.

Anonymous said to me that the buzzer is very loud and sounds like a fire alarm. He was right, it's quite loud and doesn't sound like a buzzer at all. The working of interrupts is the same as in exercise 5, but instead of breaking (stopping the motor) when you press the button you now hear an annoying sound, which makes my dog upset.

7 Hall effect and stuff

The frequency and speed of the motor rotation can be estimated using the Hall effect sensor, seen in Figure 3 (g). You have to assemble the magnet holder, seen in Figure 3 (l), into the motor shaft, and then attach the cube magnet, seen in Figure 3 (k), into the holder. Figure 2 shows an example of the mounting schema.

I connected the longest leg of the halsensor to the 3.3V, the middle leg to the ground and the shortest leg to the signal pin.

The cube didn't fit in the magnet holder, so I used a nutcracker to get it in and I put it on the motor. I used the connected hall sensor in my code.

Experiments

I have done some experiments with every exercise, because it doesn't take one try to get it right, you have to try multiple times and experiment.

exercise 1: I experimented with different resistors and noticed that you get different ranges. I also noticed that my LDR sensor is more sensitive, ranging from 0 to 4095 easily, while another one I used has a more consistent range from 1000.

exercise 2: I connected the dip switch first with 100 ohm resistors, but I afterwards just did it without and just with wires.

exercise 3: I experimented with the position and sizes of the letters of the display.

exercise 4: I figured out that combining these states with those from exercise 2 doesn't give good results.

exercise 5: I experimented with different interrupt functions.

exercise 6: The same with 5.

exercise 7: I experimented with the connection scheme and how you can read data from the hall effect sensor.

Conclusion

The toy car simulation is mostly finished in every capacity. The simulation works practically in combination with the code provided. The code is structured in a way that is readable clearly with comments. The physical parts of the toy car work well in that it's steady, clear and stuck.

References

[1] https://www.youtube.com/watch?v=1O_WaySGjHI

[2] https://www.youtube.com/watch?v=VkCvKtRsunU&ab_channel=ControltheController

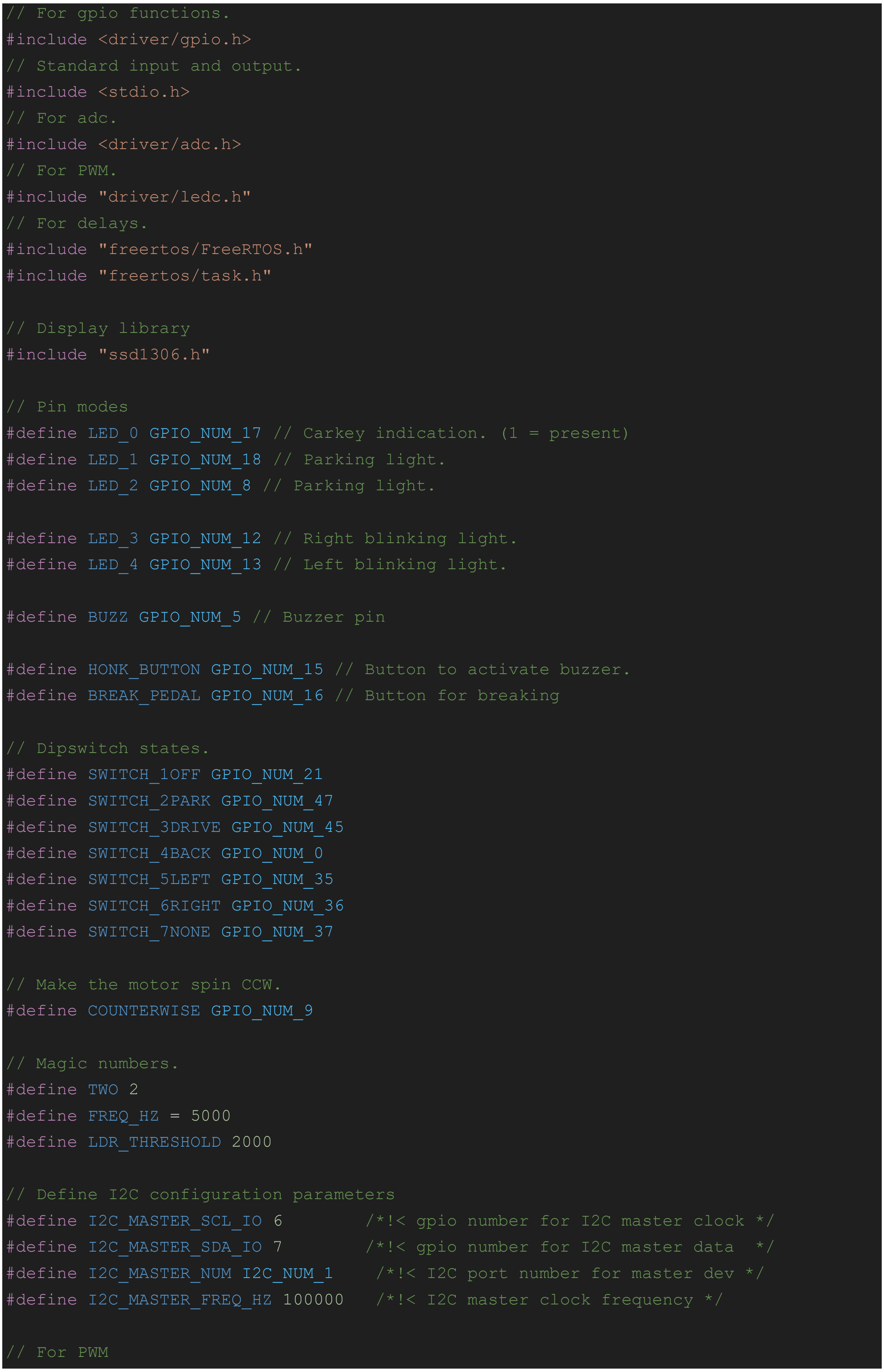
[3] https://components.espressif.com/components/espressif/ssd1306

[4] https://github.com/espressif/esp-idf/tree/e7879abbcce8566c32a5e0b86d46a1f6285951ff/examples/peripherals

[5] http://www.whatimade.today/reading-multiple-bits-using-one-arduino-pin/

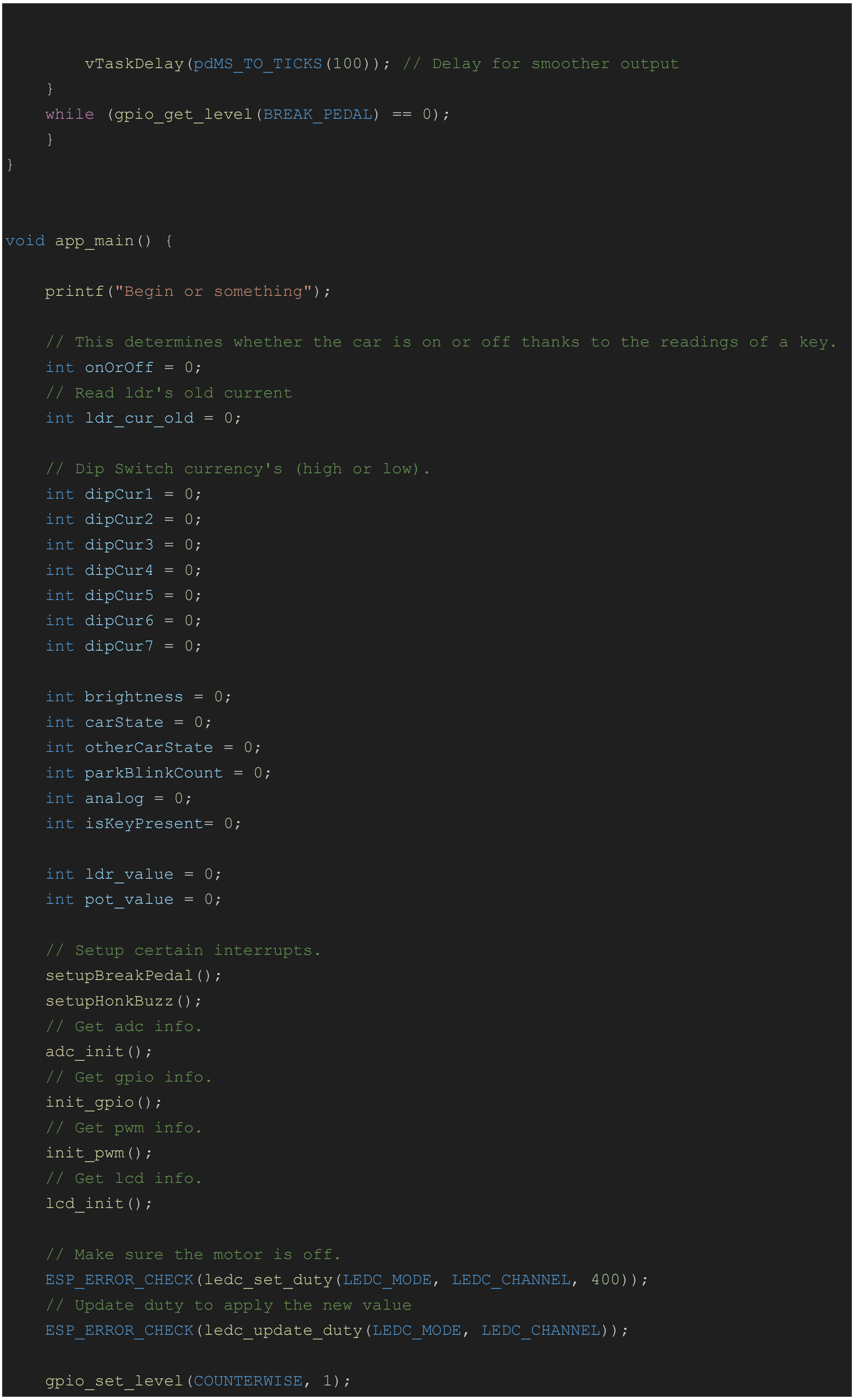
[6] https://esp32io.com/tutorials/esp32-button

My main.c code



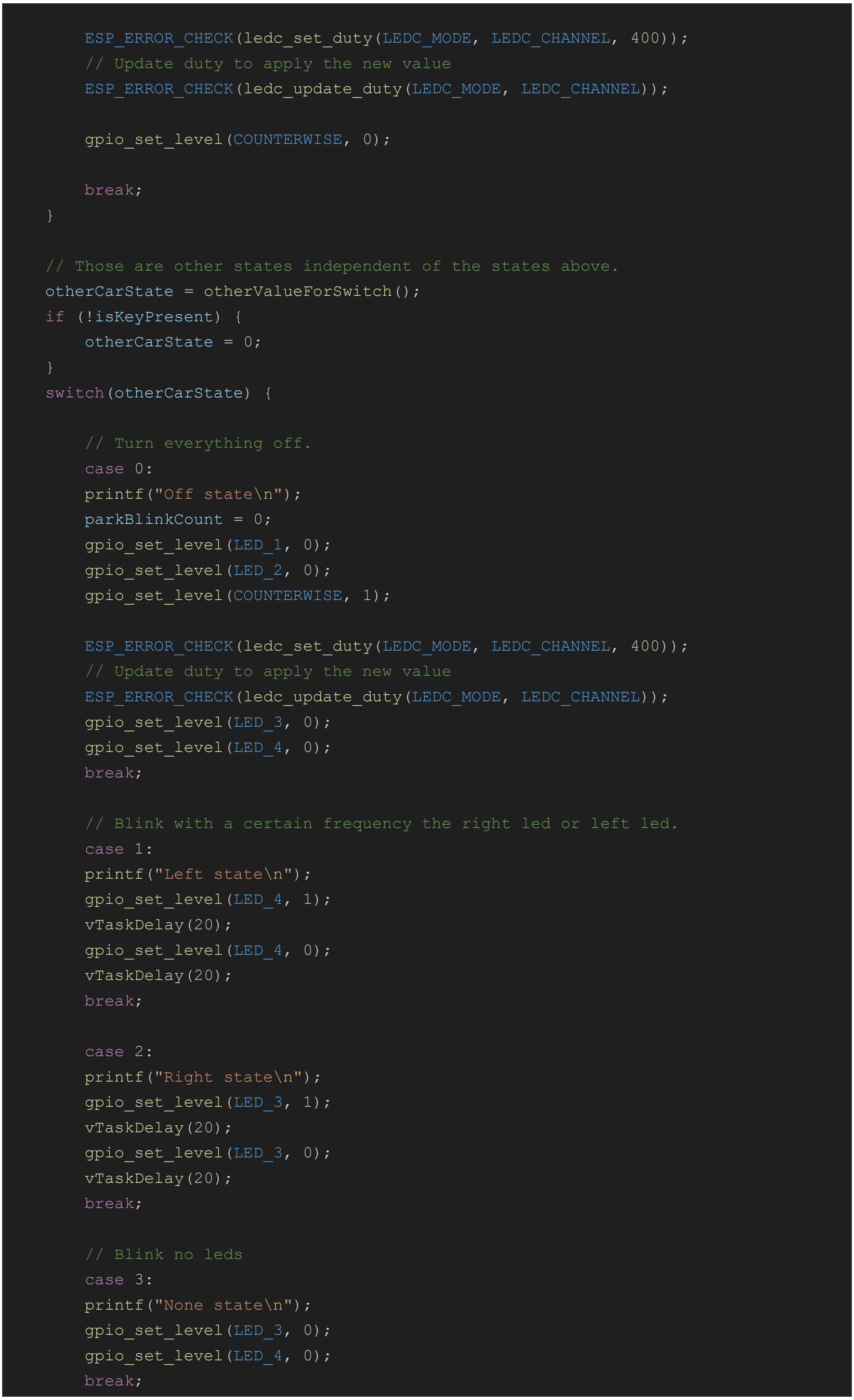
```c
// For gpio functions.
#include <driver/gpio.h>
// Standard input and output.
#include <stdio.h>
// For adc.
#include <driver/adc.h>
// For PWM.
#include "driver/ledc.h"
// For delays.
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

// Display library
#include "ssd1306.h"

// Pin modes
#define LED_0 GPIO_NUM_17 // Carkey indication. (1 = present)
#define LED_1 GPIO_NUM_18 // Parking light.
#define LED_2 GPIO_NUM_8 // Parking light.

#define LED_3 GPIO_NUM_12 // Right blinking light.
#define LED_4 GPIO_NUM_13 // Left blinking light.

#define BUZZ GPIO_NUM_5 // Buzzer pin

#define HONK_BUTTON GPIO_NUM_15 // Button to activate buzzer.
#define BREAK_PEDAL GPIO_NUM_16 // Button for breaking

// Dipswitch states.
#define SWITCH_1OFF GPIO_NUM_21
#define SWITCH_2PARK GPIO_NUM_47
#define SWITCH_3DRIVE GPIO_NUM_45
#define SWITCH_4BACK GPIO_NUM_0
#define SWITCH_5LEFT GPIO_NUM_35
#define SWITCH_6RIGHT GPIO_NUM_36
#define SWITCH_7NONE GPIO_NUM_37

// Make the motor spin CCW.
#define COUNTERWISE GPIO_NUM_9

// Magic numbers.
#define TWO 2
#define FREQ_HZ = 5000
#define LDR_THRESHOLD 2000

// Define I2C configuration parameters
#define I2C_MASTER_SCL_IO 6        /*!< gpio number for I2C master clock */
#define I2C_MASTER_SDA_IO 7        /*!< gpio number for I2C master data  */
#define I2C_MASTER_NUM I2C_NUM_1    /*!< I2C port number for master dev */
#define I2C_MASTER_FREQ_HZ 100000   /*!< I2C master clock frequency */

// For PWM
```

```
#define LEDC_TIMER              LEDC_TIMER_0
#define LEDC_MODE               LEDC_LOW_SPEED_MODE
#define LEDC_OUTPUT_IO          (10) // de output GPIO beschrijven
#define LEDC_CHANNEL            LEDC_CHANNEL_0
#define LEDC_DUTY_RES           LEDC_TIMER_13_BIT // Set duty resolution to 13 bits
#define LEDC_DUTY               (400) // Set duty to 400. ((2 ** 13) - 1) * 50% =
4095

// Declare a handle for the SSD1306 device
static ssd1306_handle_t ssd1306_dev = NULL;
volatile bool button_pressed = false;
volatile bool button_buzz = false;
bool carKeyValue(int onOrOff);
int valueForSwitch();
int otherValueForSwitch();
int getAdcForPwm();

/// @brief Determine the gpio directions.
///
/// This function determines what kind of function the gpio's have
///
/// @param
///
/// @return none
void init_gpio() {
    // For gpio_set_level set high or low.
    gpio_set_direction(LED_0, GPIO_MODE_OUTPUT);
    gpio_set_direction(LED_1, GPIO_MODE_OUTPUT);
    gpio_set_direction(LED_2, GPIO_MODE_OUTPUT);
    gpio_set_direction(LED_3, GPIO_MODE_OUTPUT);
    gpio_set_direction(LED_4, GPIO_MODE_OUTPUT);

    gpio_set_direction(COUNTERWISE, GPIO_MODE_OUTPUT);

    gpio_set_direction(BUZZ, GPIO_MODE_OUTPUT);

    // gpio_get_level iternal resistor.
    gpio_set_direction(HONK_BUTTON,GPIO_MODE_INPUT);
    gpio_set_pull_mode(HONK_BUTTON,GPIO_PULLUP_ONLY);

    gpio_set_direction(BREAK_PEDAL,GPIO_MODE_INPUT);
    gpio_set_pull_mode(BREAK_PEDAL,GPIO_PULLUP_ONLY);

    gpio_set_direction(SWITCH_1OFF,GPIO_MODE_INPUT);
    gpio_set_pull_mode(SWITCH_1OFF,GPIO_PULLUP_ONLY);

    gpio_set_direction(SWITCH_2PARK,GPIO_MODE_INPUT);
    gpio_set_pull_mode(SWITCH_2PARK,GPIO_PULLUP_ONLY);

    gpio_set_direction(SWITCH_3DRIVE,GPIO_MODE_INPUT);
    gpio_set_pull_mode(SWITCH_3DRIVE,GPIO_PULLUP_ONLY);

    gpio_set_direction(SWITCH_4BACK,GPIO_MODE_INPUT);
    gpio_set_pull_mode(SWITCH_4BACK,GPIO_PULLUP_ONLY);
    gpio_set_direction(SWITCH_5LEFT,GPIO_MODE_INPUT);
    gpio_set_pull_mode(SWITCH_5LEFT,GPIO_PULLUP_ONLY);

    gpio_set_direction(SWITCH_6RIGHT,GPIO_MODE_INPUT);
    gpio_set_pull_mode(SWITCH_6RIGHT,GPIO_PULLUP_ONLY);

    gpio_set_direction(SWITCH_7NONE,GPIO_MODE_INPUT);
    gpio_set_pull_mode(SWITCH_7NONE,GPIO_PULLUP_ONLY);
}

/// @brief Determine the pwm functions.
///
/// This function determines the ledc timer config and ledc channel_config.
///
/// @param
///
/// @return none
void init_pwm()
{
    ledc_timer_config_t timer_config;
    ledc_channel_config_t channel_config;
    timer_config.speed_mode = LEDC_LOW_SPEED_MODE;
    timer_config.timer_num = LEDC_TIMER_0;
    timer_config.duty_resolution = LEDC_TIMER_12_BIT;
    timer_config.freq_hz = 5000;
    timer_config.clk_cfg = LEDC_AUTO_CLK;

    channel_config.gpio_num = LEDC_OUTPUT_IO;
    channel_config.speed_mode = LEDC_LOW_SPEED_MODE;
    channel_config.channel = LEDC_CHANNEL_0;

    channel_config.timer_sel = LEDC_TIMER_0;
    channel_config.intr_type = LEDC_INTR_DISABLE;
    channel_config.duty = 0;
    channel_config.hpoint = 0;

    ledc_timer_config(&timer_config);
    ledc_channel_config(&channel_config);
}

/// @brief Determine the adc functions.
///
/// This function determines the adc1 config width and adc1 config attenuation
///
/// @param
///
/// @return none
void adc_init(){
  // We start with the configuration of the resolution.
  adc1_config_width(ADC_WIDTH_BIT_DEFAULT);

  // Then you set the range.

  // For LDR
  adc1_config_channel_atten(ADC1_CHANNEL_1, ADC_ATTEN_DB_11);
  // For potmeter.
  adc1_config_channel_atten(ADC1_CHANNEL_3, ADC_ATTEN_DB_11);

  // For Hall effect sensor
  adc1_config_channel_atten(ADC1_CHANNEL_2, ADC_ATTEN_DB_11);
}

/// @brief Determine the lcd display functions.
///
/// This function determines the i2c configuration.
///
/// @param
///
/// @return none
void lcd_init() {
    // Configure I2C settings
    i2c_config_t conf;
    conf.mode = I2C_MODE_MASTER;
    conf.sda_io_num = (gpio_num_t)I2C_MASTER_SDA_IO;
    conf.sda_pullup_en = GPIO_PULLUP_ENABLE;
    conf.scl_io_num = (gpio_num_t)I2C_MASTER_SCL_IO;
    conf.scl_pullup_en = GPIO_PULLUP_ENABLE;
    conf.master.clk_speed = I2C_MASTER_FREQ_HZ;
    conf.clk_flags = I2C_SCLK_SRC_FLAG_FOR_NOMAL;

    // Initialize and install the I2C driver
    i2c_param_config(I2C_MASTER_NUM, &conf);
    i2c_driver_install(I2C_MASTER_NUM, conf.mode, 0, 0, 0);

    // Create an instance of the SSD1306 device
    ssd1306_dev = ssd1306_create(I2C_MASTER_NUM, SSD1306_I2C_ADDRESS);
}

/// @brief Everything that need to be displayed in one function
///
/// This function is called to display certain words dependent on the context.
///
/// @param dipCur1to7
/// @param carState
/// @param otherCarState
/// @param speed
///
/// @return none

void whatNeedToBeDisplayed(int dipCur1, int dipCur2, int dipCur3, int dipCur4, int
dipCur5, int dipCur6, int dipCur7, int carState, int otherCarState, int speed, bool
carKey) {
        // Refresh the display and clear the screen
        ssd1306_refresh_gram(ssd1306_dev);
        ssd1306_clear_screen(ssd1306_dev, 0x00);

        // Display a string on the screen
        char data_str[30] = {0};
        sprintf(data_str, "gearbox %d %d %d %d %d %d", dipCur1, dipCur2, dipCur3,
dipCur4, dipCur5, dipCur6, dipCur7);
        ssd1306_draw_string(ssd1306_dev, 0, 0, (const uint8_t *)data_str, 12, 1);

        // Choices between different states.
        if (carState == 0 || otherCarState == 0) {
            sprintf(data_str, "OffState");
            ssd1306_draw_string(ssd1306_dev, 0, 14, (const uint8_t *)data_str, 12,
1);
        }
        else if (carState == -1) {
            sprintf(data_str, "ERROR");
            ssd1306_draw_string(ssd1306_dev, 0, 14, (const uint8_t *)data_str, 12,
1);
        }
        else if (carState == 1) {
            sprintf(data_str, "OnState");
            ssd1306_draw_string(ssd1306_dev, 0, 14, (const uint8_t *)data_str, 12,
1);
        }
        else if (carState == TWO) {
            sprintf(data_str, "ParkState");
            ssd1306_draw_string(ssd1306_dev, 0, 14, (const uint8_t *)data_str, 12,
1);
        }
        else if (carState == 3) {
            sprintf(data_str, "DriveState");
            ssd1306_draw_string(ssd1306_dev, 0, 14, (const uint8_t *)data_str, 12,
1);

            sprintf(data_str, "SPEED (2000 max): %d", speed - 2095);
            ssd1306_draw_string(ssd1306_dev, 0, 28, (const uint8_t *)data_str, 12,
1);


        }
        else if (carState == 4) {
            sprintf(data_str, "BackState");
            ssd1306_draw_string(ssd1306_dev, 0, 14, (const uint8_t *)data_str, 12,
1);
        }

        // Choices between other states.
        if (otherCarState == -1) {
            sprintf(data_str, "ERROR");
            ssd1306_draw_string(ssd1306_dev, 64, 14, (const uint8_t *)data_str, 12,
1);
        }

        else if (otherCarState == 1) {
            sprintf(data_str, "RightState");
            ssd1306_draw_string(ssd1306_dev, 64, 14, (const uint8_t *)data_str, 12,
1);
        }
        else if (otherCarState == 2) {
            sprintf(data_str, "LeftState");
            ssd1306_draw_string(ssd1306_dev, 64, 14, (const uint8_t *)data_str, 12,
1);
        }
        else if (otherCarState == 3) {
            sprintf(data_str, "NoneState");
            ssd1306_draw_string(ssd1306_dev, 64, 14, (const uint8_t *)data_str, 12,
1);
        }

        if (!carKey) {
            sprintf(data_str, "Key is absent");
            ssd1306_draw_string(ssd1306_dev, 0, 42, (const uint8_t *)data_str, 18,
1);

        }
        else {
            sprintf(data_str, "Key is present");
            ssd1306_draw_string(ssd1306_dev, 0, 56, (const uint8_t *)data_str, 18,
1);

        }



        // Refresh the display again
        ssd1306_refresh_gram(ssd1306_dev);
}

/// @brief Interrupt handlers for the honk button.
///
/// This function is called when the honk button is pressed,
/// and it sets the global variable `button_buzz` to true.
///
/// @param arg: Unused argument (can be NULL).
///
/// @return none
static void breakPedalInterrupt(void* arg) {
    button_pressed = true;
}

static void honkInterrupt(void* arg) {
    button_buzz = true;
}

/// @brief Setup function for configuring the honk or break button.
///
/// Configures the honk or break button as an input, sets interrupt type
/// to trigger on the falling edge (1 to 0 transition), and installs
/// the interrupt handler.
///
/// @return none
void setupBreakPedal() {
    // Configure the button as input.
    esp_rom_gpio_pad_select_gpio(BREAK_PEDAL);
    gpio_set_direction(BREAK_PEDAL, GPIO_MODE_INPUT);
    // From 1 to 0 in this button's case.
    gpio_set_intr_type(BREAK_PEDAL, GPIO_INTR_NEGEDGE);

    // Install the interrupt
    gpio_install_isr_service(0);
    gpio_isr_handler_add(BREAK_PEDAL, breakPedalInterrupt, NULL);
}

void setupHonkBuzz() {
    // Configure the button as input.
    esp_rom_gpio_pad_select_gpio(HONK_BUTTON);
    gpio_set_direction(HONK_BUTTON, GPIO_MODE_INPUT);
    // From 1 to 0 in this button's case.
    gpio_set_intr_type(HONK_BUTTON, GPIO_INTR_NEGEDGE);

    // Install the interrupt
    gpio_install_isr_service(0);
    gpio_isr_handler_add(HONK_BUTTON, honkInterrupt, NULL);
}

/// @brief Process function for the honk or break button actions.
///
/// Makes sure that the buzzer is beeping or the car is breaking,
/// when the button is pressed thans to the interrupt.
///
/// @return none
void processBuzzInterrupt() {
    if (button_buzz) {
        button_buzz = false;
    do {
        // Stopping the car.
        printf("Beep, beep (Honking sound)\n");
        gpio_set_level(BUZZ, 1);

        vTaskDelay(pdMS_TO_TICKS(100)); // Delay for smoother output
    }
    while (gpio_get_level(HONK_BUTTON) == 0);
    }
    else {
        gpio_set_level(BUZZ, 0);
    }
}


void processBreakInterrupt() {
    if (button_pressed) {
        button_pressed = false;
    do {
        // Stopping the car.
        printf("Skrrt, skrtt (Braking sounds)\n");
        gpio_set_level(COUNTERWISE, 1);

        ESP_ERROR_CHECK(ledc_set_duty(LEDC_MODE, LEDC_CHANNEL, 400));
        // Update duty to apply the new value
        ESP_ERROR_CHECK(ledc_update_duty(LEDC_MODE, LEDC_CHANNEL));
```

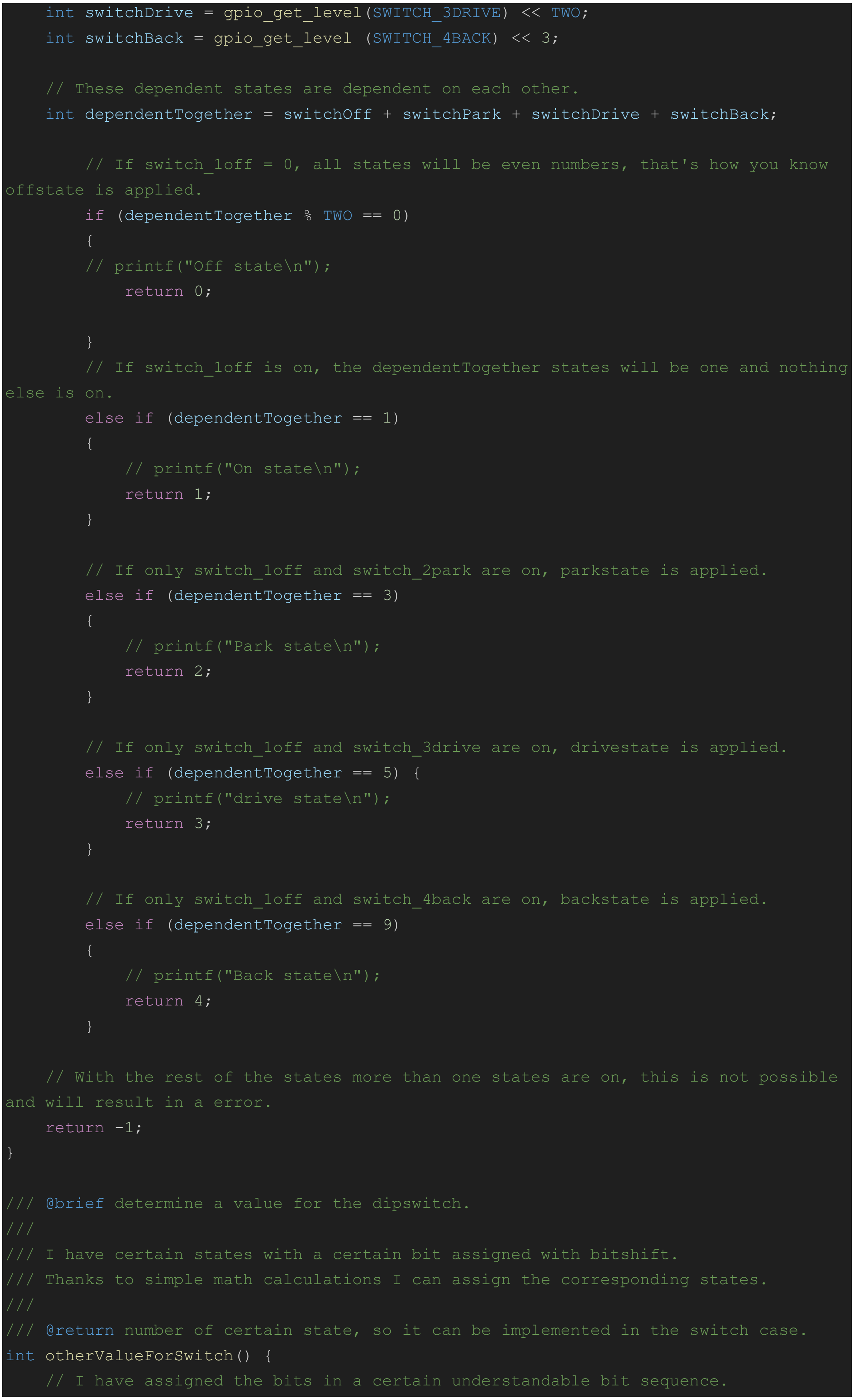


```
        vTaskDelay(pdMS_TO_TICKS(100)); // Delay for smoother output
    }
    while (gpio_get_level(BREAK_PEDAL) == 0);
    }
}


void app_main() {

    printf("Begin or something");

    // This determines whether the car is on or off thanks to the readings of a key.
    int onOrOff = 0;
    // Read ldr's old current
    int ldr_cur_old = 0;

    // Dip Switch currency's (high or low).
    int dipCur1 = 0;
    int dipCur2 = 0;
    int dipCur3 = 0;
    int dipCur4 = 0;
    int dipCur5 = 0;
    int dipCur6 = 0;
    int dipCur7 = 0;

    int brightness = 0;
    int carState = 0;
    int otherCarState = 0;
    int parkBlinkCount = 0;
    int analog = 0;
    int isKeyPresent= 0;

    int ldr_value = 0;
    int pot_value = 0;

    // Setup certain interrupts.
    setupBreakPedal();
    setupHonkBuzz();
    // Get adc info.
    adc_init();
    // Get gpio info.
    init_gpio();
    // Get pwm info.
    init_pwm();
    // Get lcd info.
    lcd_init();

    // Make sure the motor is off.
    ESP_ERROR_CHECK(ledc_set_duty(LEDC_MODE, LEDC_CHANNEL, 400));
    // Update duty to apply the new value
    ESP_ERROR_CHECK(ledc_update_duty(LEDC_MODE, LEDC_CHANNEL));

    gpio_set_level(COUNTERWISE, 1);
```

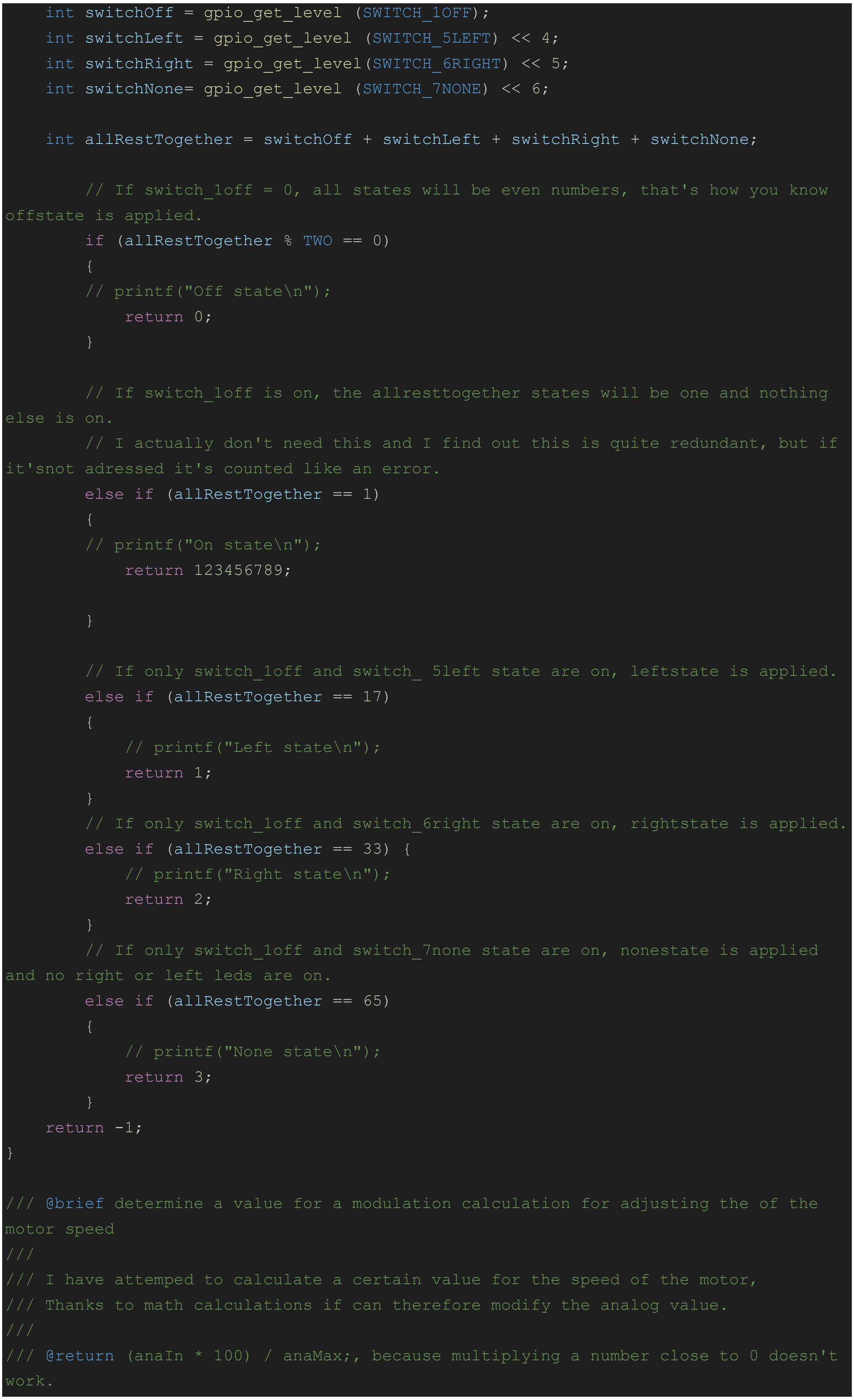
```c
while(1) {
    // get ldr adc value.
    ldr_value = adc1_get_raw(ADC1_CHANNEL_1);
    pot_value = adc1_get_raw(ADC1_CHANNEL_3);

    // get dipCur.
    dipCur1 = gpio_get_level (SWITCH_1OFF);
    dipCur2 = gpio_get_level (SWITCH_2PARK);
    dipCur3 = gpio_get_level (SWITCH_3DRIVE);
    dipCur4 = gpio_get_level (SWITCH_4BACK);
    dipCur5 = gpio_get_level (SWITCH_5LEFT);
    dipCur6 = gpio_get_level (SWITCH_6RIGHT);
    dipCur7 = gpio_get_level (SWITCH_7NONE);

    printf("\nswitch state: %d %d %d %d %d %d %d\n", dipCur1, dipCur2, dipCur3, dipCur4, dipCur5, dipCur6, dipCur7);
    // Display hall effect sensor value.
    printf("Hall effect adc value: %d\n", adc1_get_raw(ADC1_CHANNEL_2));
    printf("Button state = %d\n", gpio_get_level(BREAK_PEDAL));
    printf("carState = %d\n", carState);
    printf("LDR value: %d\n", ldr_value);
    printf("pot value: %d\n", pot_value);

    // Make sure the buttons are detected and break or buzz.
    processBreakInterrupt();
    processBuzzInterrupt();

    // Display things in the display.
    whatNeedToBeDisplayed(dipCur1, dipCur2, dipCur3, dipCur4, dipCur5, dipCur6, dipCur7, carState, otherCarState, brightness, isKeyPresent);

    // Get calculated adc value for pwm.
    analog = getAdcForPwm();

    // Retrieve carkey value.
    isKeyPresent = carKeyValue(onOrOff);

    // Get one of the 6 states the car could be in.
    carState = valueForSwitch();

    // Car is only on when the key is present.
    if (!isKeyPresent || parkBlinkCount == 10) {
        carState = 0;
    }

    switch(carState) {
        // Turn everything off what could be turned of.
        case 0:
        printf("Off state\n");
        gpio_set_level(LED_1, 0);
        gpio_set_level(LED_2, 0);
        gpio_set_level(LED_4, 0);
        gpio_set_level(LED_3, 0);
        gpio_set_level(COUNTERWISE, 1);

        ESP_ERROR_CHECK(ledc_set_duty(LEDC_MODE, LEDC_CHANNEL, 400));
        // Update duty to apply the new value
        ESP_ERROR_CHECK(ledc_update_duty(LEDC_MODE, LEDC_CHANNEL));
        break;

        // Just an indication that it's on.
        case 1:
        printf("On state\n");
        break;

        // Park the car, after ten blinks the car is parked.
        case 2:
            printf("Park state\n");
                if (parkBlinkCount < 10) {
                    gpio_set_level(LED_1, 1);
                    gpio_set_level(LED_2, 1);
                    vTaskDelay(50);
                    gpio_set_level(LED_1, 0);
                    gpio_set_level(LED_2, 0);
                    vTaskDelay(50);
                    parkBlinkCount++;
                }
            gpio_set_level(LED_1, 0);
            gpio_set_level(LED_2, 0);

            gpio_set_level(COUNTERWISE, 1);

            ESP_ERROR_CHECK(ledc_set_duty(LEDC_MODE, LEDC_CHANNEL, 400));
            // Update duty to apply the new value
            ESP_ERROR_CHECK(ledc_update_duty(LEDC_MODE, LEDC_CHANNEL));
        break;

        // Drive the car, wheels CW. Adjusted speed thanks to the potmeter. If the
car drives it can park.
        case 3:
        printf("Drive state\n");

        parkBlinkCount = 0;
        brightness = (analog * 40.95);
        // Set duty to 50%
        ESP_ERROR_CHECK(ledc_set_duty(LEDC_MODE, LEDC_CHANNEL, brightness));
        // Update duty to apply the new value
        ESP_ERROR_CHECK(ledc_update_duty(LEDC_MODE, LEDC_CHANNEL));

        gpio_set_level(COUNTERWISE, 1);

        break;

        // Wheels turn CCW, if the car drives it can park.
        case 4:
        printf("Back state\n");

        parkBlinkCount = 0;
```



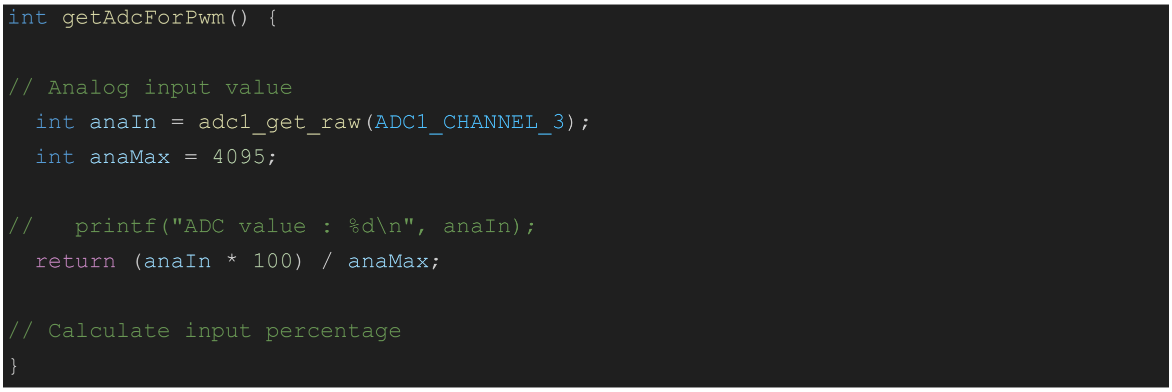
```
    ESP_ERROR_CHECK(ledc_set_duty(LEDC_MODE, LEDC_CHANNEL, 400));
    // Update duty to apply the new value
    ESP_ERROR_CHECK(ledc_update_duty(LEDC_MODE, LEDC_CHANNEL));

    gpio_set_level(COUNTERWISE, 0);

    break;
}

// Those are other states independent of the states above.
otherCarState = otherValueForSwitch();
if (!isKeyPresent) {
    otherCarState = 0;
}
switch(otherCarState) {

    // Turn everything off.
    case 0:
    printf("Off state\n");
    parkBlinkCount = 0;
    gpio_set_level(LED_1, 0);
    gpio_set_level(LED_2, 0);
    gpio_set_level(COUNTERWISE, 1);

    ESP_ERROR_CHECK(ledc_set_duty(LEDC_MODE, LEDC_CHANNEL, 400));
    // Update duty to apply the new value
    ESP_ERROR_CHECK(ledc_update_duty(LEDC_MODE, LEDC_CHANNEL));
    gpio_set_level(LED_3, 0);
    gpio_set_level(LED_4, 0);
    break;

    // Blink with a certain frequency the right led or left led.
    case 1:
    printf("Left state\n");
    gpio_set_level(LED_4, 1);
    vTaskDelay(20);
    gpio_set_level(LED_4, 0);
    vTaskDelay(20);
    break;

    case 2:
    printf("Right state\n");
    gpio_set_level(LED_3, 1);
    vTaskDelay(20);
    gpio_set_level(LED_3, 0);
    vTaskDelay(20);
    break;

    // Blink no leds
    case 3:
    printf("None state\n");
    gpio_set_level(LED_3, 0);
    gpio_set_level(LED_4, 0);
    break;
    }

    // If it's -1 it's error.
    if (valueForSwitch() == -1 || otherValueForSwitch() == -1) {
        printf("Error");
    }

    // This determines whether the threshold has been passed and

    // determines whether the key is present or not.
    if (ldr_value >= LDR_THRESHOLD && ldr_cur_old == 0) {
        // To add one to to onorof the function: bool carKeyValue(int onOrOff) can 
determine whether the key is absent or not (the car is on or off).
        onOrOff++;
    }
    // if ldr value is higher than ldr threshold ldr cur old = 1 if not it's 0.
    // Update the old currency recording to that.
    ldr_cur_old = (ldr_value > LDR_THRESHOLD) ? 1 : 0;

    vTaskDelay(100);
}
}

/// @brief This function determines whether the carkey is absent or not.
///
/// If it's an even number the car key is absent if it's a uneven number the car key 
is present.
/// @param onOrOff this number increases everytime the threshold is passed.
///
/// @return true or false. When key is present or not.

bool carKeyValue(int onOrOff) {
    // If threshold has been passed ones after car key is present it's absent.
    if (onOrOff % TWO == 0) {
        printf("Car key is absent\n");
        gpio_set_level(LED_0, 0);
        return false;

    // If threshold has been passed ones after car key is absent it's present.
    } else {
        printf("Car key is present\n");
        gpio_set_level(LED_0, 1);
        return true;
    }
}

/// @brief determine a value for the dipswitch.
///
/// I have certain states with a certain bit assigned with bitshift.
/// Thanks to simple math calculations I can assign the corresponding states.
///
/// @return number of certain state, so it can be implemented in the switch case.
int valueForSwitch() {
    int switchOff = gpio_get_level (SWITCH_1OFF);
    int switchPark = gpio_get_level (SWITCH_2PARK) << 1;
```



```
    int switchDrive = gpio_get_level(SWITCH_3DRIVE) << TWO;
    int switchBack = gpio_get_level (SWITCH_4BACK) << 3;

    // These dependent states are dependent on each other.
    int dependentTogether = switchOff + switchPark + switchDrive + switchBack;

        // If switch_1off = 0, all states will be even numbers, that's how you know offstate is applied.
        if (dependentTogether % TWO == 0)
        {
        // printf("Off state\n");
            return 0;

        }
        // If switch_1off is on, the dependentTogether states will be one and nothing else is on.
        else if (dependentTogether == 1)
        {
            // printf("On state\n");
            return 1;
        }

        // If only switch_1off and switch_2park are on, parkstate is applied.
        else if (dependentTogether == 3)
        {
            // printf("Park state\n");
            return 2;
        }

        // If only switch_1off and switch_3drive are on, drivestate is applied.
        else if (dependentTogether == 5) {
            // printf("drive state\n");
            return 3;
        }

        // If only switch_1off and switch_4back are on, backstate is applied.
        else if (dependentTogether == 9)
        {
            // printf("Back state\n");
            return 4;
        }

    // With the rest of the states more than one states are on, this is not possible and will result in a error.
    return -1;
}

/// @brief determine a value for the dipswitch.
///
/// I have certain states with a certain bit assigned with bitshift.
/// Thanks to simple math calculations I can assign the corresponding states.
///
/// @return number of certain state, so it can be implemented in the switch case.
int otherValueForSwitch() {
    // I have assigned the bits in a certain understandable bit sequence.
```



```
    int switchOff = gpio_get_level (SWITCH_1OFF);
    int switchLeft = gpio_get_level (SWITCH_5LEFT) << 4;
    int switchRight = gpio_get_level(SWITCH_6RIGHT) << 5;
    int switchNone= gpio_get_level (SWITCH_7NONE) << 6;

    int allRestTogether = switchOff + switchLeft + switchRight + switchNone;

        // If switch_1off = 0, all states will be even numbers, that's how you know
offstate is applied.
        if (allRestTogether % TWO == 0)
        {
        // printf("Off state\n");
            return 0;
        }

        // If switch_1off is on, the allresttogether states will be one and nothing
else is on.
        // I actually don't need this and I find out this is quite redundant, but if
it'snot adressed it's counted like an error.
        else if (allRestTogether == 1)
        {
        // printf("On state\n");
            return 123456789;

        }

        // If only switch_1off and switch_ 5left state are on, leftstate is applied.
        else if (allRestTogether == 17)
        {
            // printf("Left state\n");
            return 1;
        }
        // If only switch_1off and switch_6right state are on, rightstate is applied.
        else if (allRestTogether == 33) {
            // printf("Right state\n");
            return 2;
        }
        // If only switch_1off and switch_7none state are on, nonestate is applied
and no right or left leds are on.
        else if (allRestTogether == 65)
        {
            // printf("None state\n");
            return 3;
        }
    return -1;
}

/// @brief determine a value for a modulation calculation for adjusting the of the
motor speed
///
/// I have attemped to calculate a certain value for the speed of the motor,
/// Thanks to math calculations if can therefore modify the analog value.
///
/// @return (anaIn * 100) / anaMax;, because multiplying a number close to 0 doesn't
work.
```



```
int getAdcForPwm() {

// Analog input value
  int anaIn = adc1_get_raw(ADC1_CHANNEL_3);
  int anaMax = 4095;

//   printf("ADC value : %d\n", anaIn);
  return (anaIn * 100) / anaMax;

// Calculate input percentage
}
```