FACULTY OF DIGITAL MEDIA AND CREATIVE INDUSTRIES
HBO – Information and Communication Technologies

ESP32 Toy Car Assignment

Embedded Systems 2

Lars Grit

January 16, 2025

1 Introduction

The car should have the following features:

- The car should have a key that can be inserted and removed. The car should only work when the key is inserted.
- The car should have a gearbox with 4 different states: OFF, PARK, DRIVE, and BACK.
- The car should have a motor that can be controlled by the gearbox. The motor should be able to rotate clockwise and counterclockwise.
- The car should have a display that shows the state of the car.
- The car should have turn signals that can be activated by the gearbox.
- The car should have a horn that can be activated by a button.
- The car should have a brake that can be activated by a button.

For the assignment I had to use the IoT KIT given to me which included these parts:

- A light–dependent resistor (LDR).
- A DIP switch (DIP).
- A DC motor (MOT).
- An L293D motor driver (DRV).
- A potentiometer (POT).
- An I2C SSD1306 OLED display (DSP).
- A Hall effect sensor (HAL).
- A cube magnet (MGN).
- A buzzer (BZR).
- Auxiliary LEDs (LED).
- Auxiliary push buttons (BTN).

2 Assignments

1. 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.

2. You will use a section on the DIP switch (DIP), seen in Figure 3 (b), to simulate the gearbox 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.

3. 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.

4. 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.

5. 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 have to use an interrupt to simulate this process.

6. 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.

7. 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.

3 Report

3.1 Project Setup

To start this project I made sure to setup a proper project structure. This is the structure I used:

```
LaboratoryProject/       # PlatformIO project containing the ESP32 firmware
   src/                 # Source code for the project
   include/             # Header files
   lib/                 # Libraries used in the project
   test/                # Unit tests for the project
   platformio.ini       # PlatformIO configuration file
Latex Documentation/     # LaTeX files for the project report
   main.tex             # Main LaTeX file
   figures/             # Figures and diagrams used in the documentation
docs/                    # Project documentation in Markdown format
   index.md             # Main documentation file
   assets/              # Media assets for documentation
      schematics/      # Circuit diagrams and schematics
      design/          # CAD files and design-related visuals
      images/          # General images and screenshots
.gitignore               # Git ignore file
README.md                # This file
```

This structure was used to keep the project organized and to make it easier to navigate through the different parts of the project. The PlatformIO project contains the firmware for the ESP32, the LaTeX documentation contains the report, and the docs folder contains the project documentation in Markdown format.

PlatformIO is a cross-platform, cross-architecture, multiple framework, professional tool for embedded systems engineers and for software developers who write applications for embedded products. It is a powerful, open-source, and multi-platform IDE for IoT development.

3.2 Component List

The assignment was to use common components to create a toy car. The components used in this project are:

- ESP32-S3-Wroom-1
- A light–dependent resistor (LDR)
- A DIP switch (DIP)
- A DC motor (MOT)
- An L293D motor driver (DRV)
- A potentiometer (POT)
- An I2C SSD1306 OLED display (DSP)
- A Hall effect sensor (HAL)
- A cube magnet (MGN)
- A buzzer (BZR)
- Auxiliary LEDs (LED)
- Auxiliary push buttons (BTN)

3.3 Program

The program is written in C with espidf and platformIO.

3.4 Program Setup

The program is designed to handle various components of the toy car, each of which is controlled by a specific function. The setup begins with an LDR-based ignition system, followed by the DIP switch for gear control, motor controls, indicator lights, brake and horn buttons, Hall effect sensor for RPM measurement, and an OLED display to show the car's status.

3.4.1 LDR Ignition System

The LDR sensor is used to detect the presence of a key (represented by a finger) based on its light sensitivity. When the LDR detects a low light level (below a certain threshold), it triggers the ignition of the system. The function `key_check` reads the LDR value using ADC and returns `true` if a key is detected, signaling the car to power on and activate the DIP switch system. This key check is continually monitored in the main loop.

3.4.2 DIP Switch Gear Control

The DIP switch system is implemented using three switches to select the car's current mode. The `get_dip_switch_state` function checks the states of these switches and returns the selected mode (e.g., park, drive, reverse). The program uses the DIP switch inputs to control the car's motor direction and speed, with additional checks for brake and horn activation.

3.4.3 Motor Control

Motor control is achieved through the L293D motor driver. The motor's direction is set based on the selected DIP switch state, and its speed is adjusted using the potentiometer value. The motor speed is dynamically adjusted by reading the potentiometer through ADC and mapping the values to a PWM duty cycle. This functionality is controlled by the `get_motor_speed` function, which reads the potentiometer's value and adjusts the motor speed accordingly.

3.4.4 Directional Lights and Indicator Task

To handle the indicator lights, a dedicated FreeRTOS task (`indicator_light_task`) continuously checks the DIP switch states for lights control. The lights are turned on or blink based on the active DIP switch setting. This task uses GPIOs to manage the LED behavior, providing visual feedback on the current car mode (e.g., park, drive, reverse).

3.4.5 Brake and Horn Buttons

The brake and horn buttons are configured with interrupts. When the brake button is pressed, the car stops, and the motor is deactivated. Similarly, when the horn button is pressed, the system activates the buzzer to simulate the horn sound. These actions are handled by the `handle_brake` and `handle_horn` functions. Button debouncing is handled using timestamps to avoid multiple triggers from a single press.

3.4.6 Hall Effect Sensor for RPM Measurement

The Hall effect sensor detects pulses generated by a rotating magnet, which is used to calculate the car's motor RPM. The sensor is configured with an interrupt handler (`hall_sensor_isr_handler`) to count the pulses. A separate task (`calculate_rpm_task`) calculates the RPM by measuring the pulse frequency over a defined period. This value is updated periodically and displayed on the OLED screen.

3.4.7 OLED Display

The OLED display is used to show the car's status (e.g., mode, motor speed). The display is controlled using the I2C interface, with functions like `oled_init`, `oled_clear`, and `oled_write_text` managing communication with the screen. The car's current state is displayed in real-time on the OLED screen, providing users with visual feedback on the car's operation.

3.4.8 System Integration and Flow

The system is integrated in a modular way, with each function responsible for a specific hardware component. The main program flow handles the logic of checking the key presence, reading the DIP switch state, controlling the motor and lights, and processing button presses and RPM calculations. The system is built step by step, starting from basic key detection, progressing through motor and light controls, and ending with the full system functionality being displayed on the OLED screen.

3.5 Hardware Setup

The hardware setup for this project involved using a variety of components to control the toy car's functionalities. All the components were connected on three breadboards, allowing for a clean and modular arrangement. The wiring was done with a combination of standard jumper wires and stripped wires for a more organized and aesthetically pleasing layout. This approach reduced the overall mess of wires while maintaining flexibility for testing and modifications.

3.5.1 Custom Wiring for Cleanliness

To make the wiring cleaner and more manageable, I used stripped wires for most of the connections. I had never used stripped wires before, but I found them to be very useful for creating custom lengths and keeping the wiring neat.

4 Experiments

4.1 Challenges and Learning Experiences

4.1.1 Writing My First Extensive ESP-IDF Project

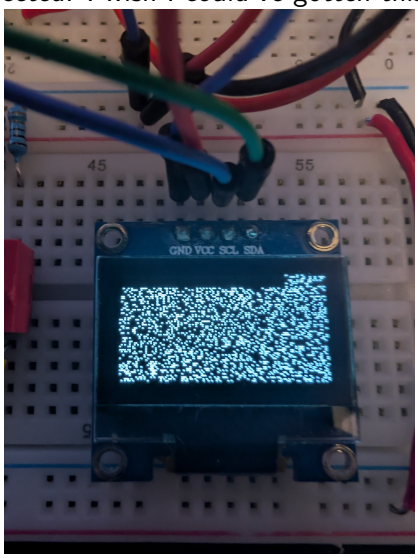
This project was my first experience writing such an extensive program using the ESP-IDF framework, and it was both challenging and rewarding. I had never worked with the ESP32 in such depth before, and the complexity of integrating various hardware components and managing their interactions made this project a great learning experience. Compared to using the Arduino framework where many things are abstracted and you can use libraries for most tasks.

4.1.2 Working with Interrupts

One of the most significant challenges I faced was working with interrupts. Interrupts were a fairly new concept to me in practice, and I had to ensure that each component was correctly handled through interrupt service routines (ISRs). These are critical for real-time response, especially for buttons and sensors, but understanding how to manage them effectively required careful study and implementation. Despite the steep learning curve, I now feel more comfortable with this aspect of embedded systems development.

4.1.3 OLED Display Challenges

The OLED display was another area where I encountered difficulties. I attempted to control the display without relying on an external library, which I thought would give me more flexibility. However, I struggled to get the display to function properly, as handling the low-level communication with the I2C interface and sending commands manually proved more complicated than expected. I wish I could've gotten this to work.



4.1.4 Multithreading with FreeRTOS Tasks

Multithreading using tasks in FreeRTOS was another unfamiliar area for me. Managing multiple tasks concurrently, such as reading sensor values, controlling the motor, and handling button presses. Learning this concept was really helpful.

4.1.5 Component Configuration

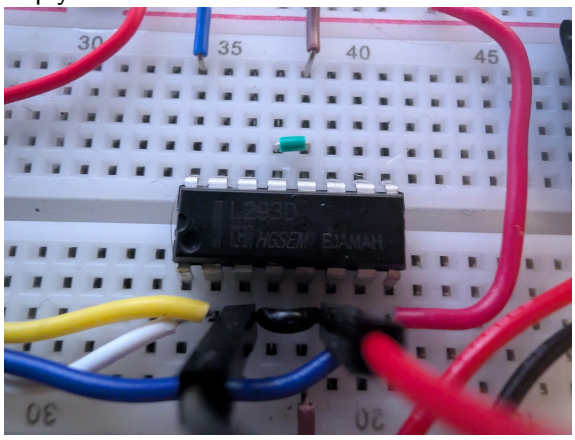
Another significant part of the project involved configuring all the components from scratch. I was able to get deeply involved in the technical details of each hardware component, from setting up the GPIOs for switches and LEDs to configuring the motor driver and sensor interfaces. I learned the importance of proper component initialization and configuration for reliable operation.

```c
/**
 * @brief Configure a button with an interrupt handler.
 *
 * This function configures a button with an interrupt handler that triggers on a falling edge.
 *
 * @param pin The GPIO pin number to configure with an interrupt.
 */
void configure_button_with_interrupt(gpio_num_t pin) {
    gpio_config_t io_conf = {
        .pin_bit_mask = (1ULL << pin),      // Configure the specific pin
        .mode = GPIO_MODE_INPUT,            // Set as input
        .pull_up_en = GPIO_PULLUP_ENABLE,   // Enable pull-up resistor
        .pull_down_en = GPIO_PULLDOWN_DISABLE,
        .intr_type = GPIO_INTR_NEGEDGE      // Trigger on falling edge
    };
    gpio_config(&io_conf);

    // Install the ISR service and attach the handler
    gpio_install_isr_service(0);
    gpio_isr_handler_add(pin, brake_button_isr_handler, NULL);
}
```

4.1.6 Using a DC Motor for the First Time

Using a DC motor with the L293D motor driver for the first time was also an exciting part of the project. I had never worked with a motor controller before, and getting the motor to work with a controller like that was really cool. It was really fun to do quite a bit of research on how to simply connect stuff.



4.1.7 Overall Project Experience

Overall, this project was a significant step in my embedded systems journey. It allowed me to dive into areas I had not explored before, and despite the challenges, it was really cool to see most things come together. I just wish I could've made the OLED screen work how I wanted it to.

5 Result

5.1 Photos of the result