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

ESP32 Toy Car Assignment

Embedded Systems 2

Lars Grit

January 15, 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.

I had to use several components given to me.

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

I started by setting up the project through PlatformIO. This is a plugin for Visual Studio Code that allows you to work with the ESP32.
I created a new project and kept the following file structure:

```
Project
 src
    CMakeLists.txt
    main.c
 include
    README.md
    pin_config.h
    utilities.h
 README.md
 platformio.ini
 .gitignore
```

3.2 Component List

The assignment was to simulate a car. To do this, we needed several components:

- 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 and is divided into 3 files:

- **main.c**, *which contains the main loop*
- **pin_config.h**, *which contains the pin configuration*
- **utilities.h**, *which contains the utility functions and libraries*

To get started on the car, I had to figure out a way to make the different states of the car work. To do this I created some sort of state machine. I tried working with structs but I was not able to get it to work and it felt a bit too over engineered at some point.

3.3.1 Core Loop

1. The car is in the OFF state, which is the default state where the car checks for the presence of the key.
2. If the key is present, the car starts up, and the DIP switch is examined to determine the gearbox simulation. This switch determines the state of the car.
3. There are four states the car can be in: OFF, PARK, DRIVE, and BACK.
   (a) If the car is in the PARK state, the LEDs will flash, and it will wait until the car transitions to another state.
   (b) In the DRIVE state, the engine is activated, and the speed is controlled by the potentiometer. The engine rotates clockwise in this state.
   (c) In the BACK state, the engine is activated, and it rotates counterclockwise.
   (d) If the car is in the OFF state, the engine is turned off, and the key must be reinserted.
   (e) If the gearbox is not in any of these states, it is in the NEUTRAL state, and nothing happens.
4. The car can activate its turn signals by using the appropriate DIP switch. When the left DIP switch is on, the left LEDs will flash, and when the right DIP switch is on, the right LEDs will flash. If both are on, both LEDs will flash.
5. The car can honk the horn by pressing the horn button. This is an interrupt that triggers the horn.

6. The car can brake by pressing the brake button. This is an interrupt that shuts off the engine.
7. The car can display information about its state, speed, horn usage, and turn signals on the OLED screen.

4 Experiments

4.1 State machines

I wanted to create something where you could enter a state, invoke the start function once, and then repeatedly invoke the update function. I tried to do this with structs, but I couldn't get it to work. I ended up using a switch statement to check the state and then execute the corresponding code. I am not sure if this is the best way to tackle this, but it sure is a way that I think works pretty okay and is easily understandable.

4.2 Interrupts

I had never worked with interrupts before in embedded systems, so I had to do some research on how to use them. I found a good tutorial that explained how to use them. I used the same method for both the horn and the brake. Eventually, I also got word that I had to use interrupts for the Hall sensor, but sadly I didn't get this to work. I hope to eventually look into the Hall sensor again since it is a useful component.

4.3 Hall sensor

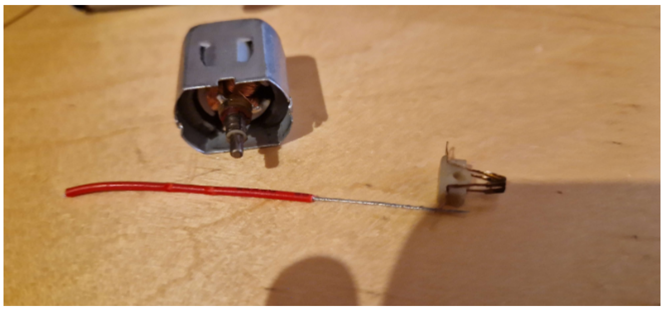
I have tried to get the Hall sensor working, but I was not able to get it working. I have tried multiple different ways to get it working, for example:

- Interrupts
- While loops
- GPIO readings

Sometimes I ended up with a value between 0-10 which spiked to more than 1000, or I ended up with backtrace errors. Eventually, I just wanted to be done with it so I decided to leave it out for now.

4.4 Motor

I was struggling a lot with my motor. All the documentation I could find was for Arduino. Not everywhere did it mention the need for an external power source, and when I thought I had it right, not much happened. I went through three DC motors because they stopped working. I tried to repair one, but unfortunately, it didn't yield any results.



Finally, I managed to get the motor running once and adjust the speed with the potentiometer. However, it only worked in one direction because as soon as I connected `MOTOR_IN2_PIN` correctly, it stopped, seemingly due to the current coming out of the driver. I tried to use a multimeter to figure out where it started or what caused it, but I couldn't get any clearer information. So I just hard coded it to the ground as a quick fix.

em2-motor-test1.mp4

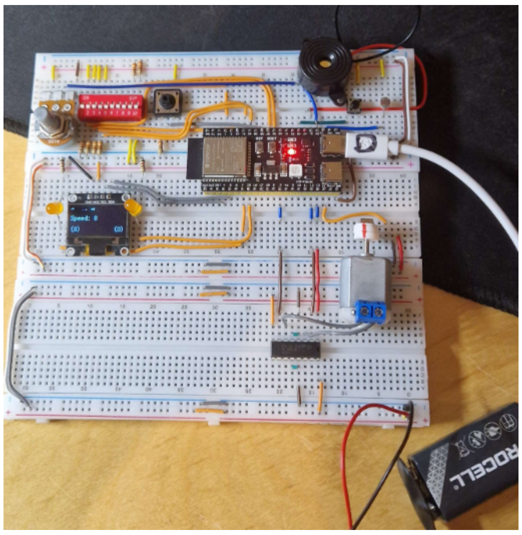
I was still super happy that at least it worked in one direction, but when I tried to start it again the next day, it didn't work anymore. And just like how it suddenly didn't work again, it suddenly started working again. I have no idea what caused it to stop working or what caused it to start working again. I'm just happy it works now.

em2-motor-both-sides.mp4

5 Result

Figuring out how to use the motor was the most difficult part of this project. I had never worked with a dc motor before.
It also took me a while before I finally settled on a way to implement some kind of core loop without blocking code.
I'm happy with the result, it looks clean, works pretty well and I learned a lot from it.

5.1 Photo of the result



5.2 Video

em2-final.mp4