Traffic Light Project

Royal Institute of Technology

Arvin Kunalic

IS1300

Abstract

This report covers the design and implementation of a traffic light simulation system for a four-way intersection, completed as part of the IS1300 course at the Royal Institute of Technology. Using the STM32L476RG microcontroller and a custom Traffic Light Shield, the system manages pedestrian and vehicular traffic through modular software components, including shift register control, OLED display management, and timer-based synchronization. The project successfully fulfills requirements for three tasks, demonstrating effective use of hardware timers and interrupts for scheduling. Testing resulted in reliable performance, and challenges with ambiguous requirements are discussed to inform future improvements.

December 28, 2024

Contents

List of Figures

List of Tables

1 Introduction

Course IS1300 at KTH covers the design and implementation of embedded systems, focusing on both hardware and software aspects. The main topics include microcontroller (MCU) architecture, I/O port utilization, and programming within real-time operating systems using FreeRTOS. The course mainly focuses on practical experience through laboratory work and a final project, encouraging students to implement the theoretical information acquired during the course in real life.

This report presents the course project, discussing the implementations and the results. The project consists of three tasks that progressively build on traffic light logic for both pedestrians and vehicles. The project uses the `Nucleo-L476RG` development board and a custom-made Traffic Light Shield, both of which are provided by the university.

The goal of the project is to simulate realistic traffic lights of a four-way intersection, managing both vehicles and pedestrians.

2 Project Tasks Overview

The project consists of three tasks in total, and each task has multiple requirements. Since the goal is to simulate traffic lights, task scheduling and management play a crucial role. The project is graded based on how many tasks are implemented. I chose to implement all three tasks in my project.

Brief Task Overview:

Task 1: Focuses on implementing pedestrian crossing logic on the Traffic Light Shield.

Task 2: Focuses on the road crossings and emulating traffic light logic with cars on the road, ignoring pedestrians.

Task 3: Combines Task 1 and Task 2 with additional requirements, such as only allowing one pedestrian crossing to be green at a time and utilizing SPI communication for the shift registers on the Traffic Light Shield.

More detailed descriptions of the individual tasks and their requirements, can be viewed at Appendix A at the end of this report.

3 Methodology

This section outlines the hardware and software design and implementation processes for the Traffic Light Project. It details the hardware components used, the software modules developed, and the overall architecture of the system.

3.1 Hardware Description

The main hardware used during the project was the STM32L476RG microcontroller, a custom-made Traffic Light Shield, and various peripheral components to implement a traffic and pedestrian light control system. The hardware is configured to fulfill Tasks: 1, 2 and 3.

For further details about the hardware components, datasheets can be viewed at Appendix B.

3.1.1 STM32L476RG

The MCU used during the course and project is the `STM32L476RG`, a 32-bit ARM Cortex-M4 core operating at up to 80 MHz. It features with three SPI and I^2C busses, 114 I/O pins and 17 timers. Detailed technical specifications and documentation for the MCU can be found on STMicroelectronics's website [1].

The primary MCU-features used in this project are: 13 of the I/O pins, four 16-bit timers and two of the SPI busses.

3.1.2 Traffic Light Shield

The Traffic Light Shield is a custom-made printed circuit board (PCB) and is equipped with a variety of components. The main component consists of three daisy-chained, 8-bit shift registers that control the 24 traffic light LEDs. The LEDs make up the four traffic lights in each direction (three LEDs per direction) and the two pedestrian lights (six LEDs per crosswalk).

Switches on the shield represents cars, if the switch signal is pulled LOW (towards the intersection) the car is interpreted as ”active” otherwise, inactive. The buttons on the shield are configured in the same manner in hardware, a button press pulls the signal LOW, indicating a pedestrian wanting to cross a lane.

The components on the shield:

1 x OLED Display [SSD1306]

3 x 8-bit Shift Registers [74HC595]

4 x Buttons

4 x Switches

Unused in this project:

1 x Accelerometer [LIS2DW12]

1 x Temperature & Humidity Sensor [SHT20]

1 x Joystick

1 x Potentiometer

3.1.3 Architecture

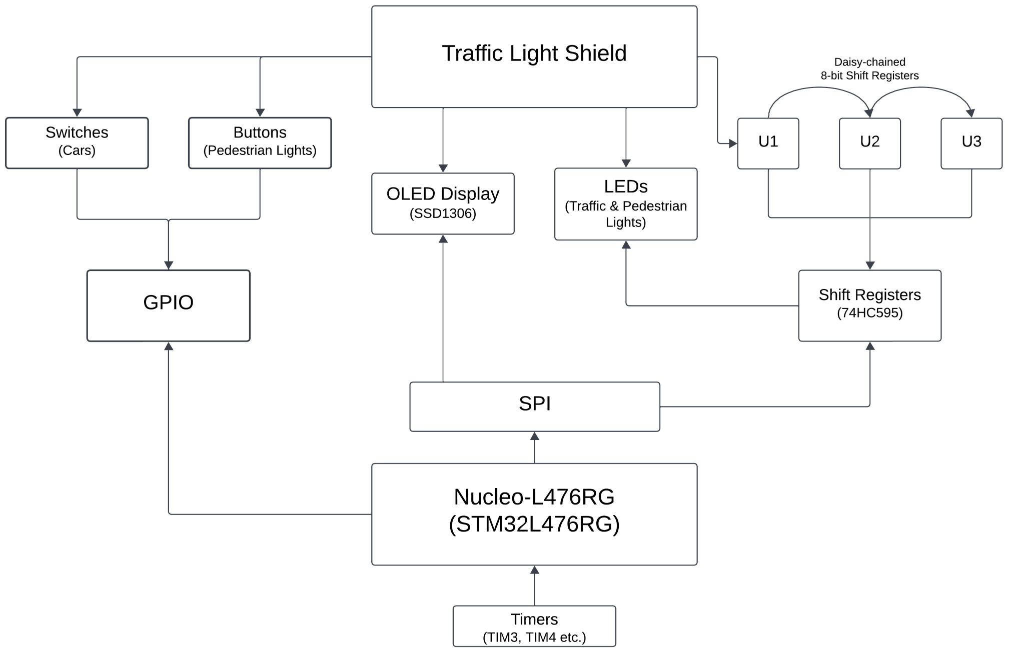

Figure 1: Project - Hardware Architecture

Figure 1 illustrates the hardware setup of the Traffic Light Project. The `STM32L476RG` MCU serves as the central unit, interfacing with the Traffic Light Shield that includes LEDs for traffic and pedestrian lights. Communication between the microcontroller and shield components is facilitated through GPIO and SPI interfaces. The daisy-chained shift registers handle the control of multiple LEDs, while timers are responsible for synchronized light transitions.

3.2 Software Description

The software for this project integrates the MCU with various hardware components to control traffic and pedestrian lights. Software is developed with modular C files which will be presented in more detail in subsection 3.3.

The STM32 Hardware Abstraction Layer (HAL) is used throughout the project development because of its peripheral configuration and interaction simplicity. The HAL allowed for configuration of each hardware peripheral without needing to manage and manipulate multiple hardware registers.

3.2.1 Development IDEs

Microsoft Visual Studio Code:

Microsoft Visual Studio Code was used for writing and documenting code due to its ease of use, flexibility and support for various extensions.

STM32CubeIDE

STM32CubeIDE was used during the project to configure GPIO, SPI, timers, and to debug and flash the MCU. The UI, HAL and code generation provided a seamless setup of all the pins and necessary peripherals.

The entire project, including the source code and STM32CubeIDE configuration files can be found at my GitHub repository [2].

3.2.2 Architecture

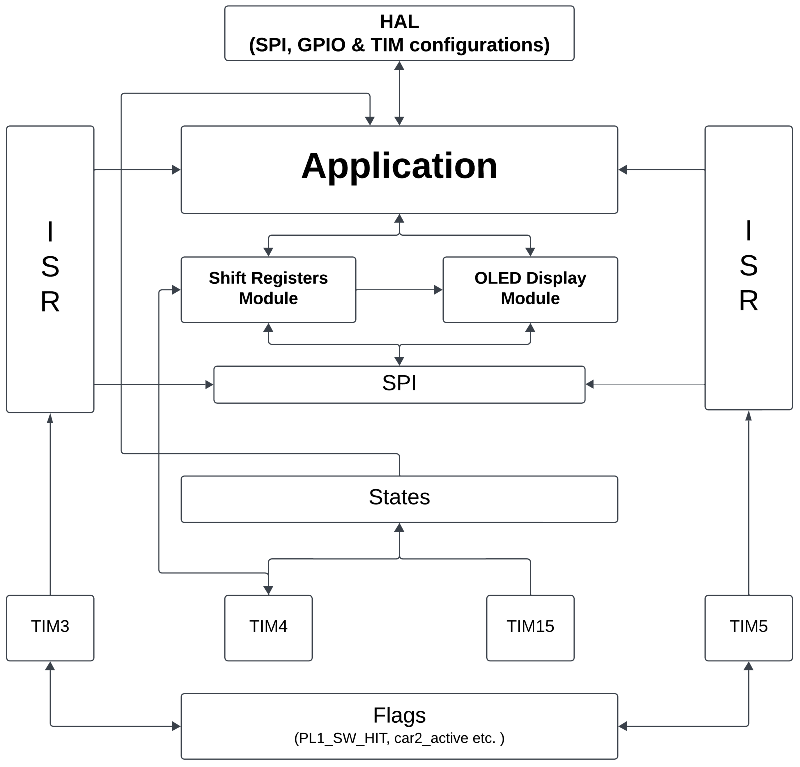

Figure 2: Project - Software Architecture

Figure 2 shows the software design, highlighting the modular approach adopted for traffic light control. Shift Register and OLED Display modules encapsulate specific functionalities, such as LED control and visual feedback. The state-machine logic manages transitions between different traffic states (scenarios), while interrupt service routines (ISRs) handle user inputs and timer events. This architecture promotes maintainability and efficient debugging.

3.3 Developed Modules

During the project, three software modules were developed for the Traffic Light Shield. Each module serves a specific purpose and integrates with the hardware components of the system, this made the development process easier in terms of debugging, scaling and testing.

3.3.1 Shift Register Module

This module is a software implementation designed to manage traffic and pedestrian lights using the three shift registers. The module provides functions which help mimic realistic behaviors of traffic lights. The functions are defined in `595_shiftreg.c`.

Features:

1. **Shift Register Management:**

   The module initializes a software buffer to store data before writing to the registers. Data is prepared in the buffer and transmitted via SPI to update the state of the traffic lights.

   Functions perform bitwise operations and bitmasking in order to specifically set one (or multiple) of the shift registers pins HIGH or LOW. By combining the current buffer state with a bitmask representing the desired pins to turn on, no unwanted pins (LEDs) will be affected. Which, during development simplified the registers updates, since it's not possible to alter unspecified LEDs.

2. **Traffic Light Control:**

   Functions manage the transitions of traffic lights for a given intersection Red $\rightarrow$ Yellow $\rightarrow$ Green and vice-versa. Transitions are staged and time-controlled by a hardware timer. Staging the transitioning, is intuitive and results in realistic traffic light behavior.

3. **Pedestrian Light Control:**

   The module handles management of pedestrian traffic signals, including toggling the blue waiting indicators and activating or deactivating pedestrian lights.

4. **Timing:**

   The module also handles some configured hardware timers. For example: timer 4 is used in the functions transitioning the traffic lights, which ensures that the yellow light stays lit for `'orange_Delay'` seconds. It takes five seconds for an intersection to transition from green to red and vice-versa.

3.3.2 OLED Display Module

The OLED display module can be found in `ssd1306_config.c`, it is responsible for setting up and controlling the 128 x 64 SSD1306 OLED display on the shield. The module contains basic display-controlling functions and communicates via SPI. The display serves as an interface to provide visual feedback about the state of cars and pedestrians.

Similar to the shift register module, the display module uses a 1024-long software framebuffer where each byte (element in the array) represents eight vertical pixels on the display. The module has two primary functions `draw_char()` and `draw_string()`, which are responisble for displaying characters on the display using a 5x7 bitmap font stored in `fonts.c`.

Features:

1. **SPI communication:**

   Through SPI communication, functions write data from the framebuffer to the command/data registers of the display. Pulling the display's Data/Command pin LOW writes to the command register, HIGH writes to the data register.

2. **Display Initialization:**

   The initialization function configures the display based on SSD1306 datasheet specifications. Settings include display offset, memory addressing mode, contrast control and enabling the charge pump [3].

3. **Displaying Characters & Strings:**

   `draw_char()` renders a character on the display.

   `draw_string()` renders strings by arranging characters horizontally with a 1-pixel spacing, wrapping to the next line if the end of the screen is reached.

3.3.3 Clock Module

The clock module provides functionality for the traffic light project, along with precise timing and synchronization for various functions. It combines system clock setup, timer configuration, and interrupt handling. Four 16-bit hardware timers are used in the project, two of which are interrupt-enabled and the other two are used for timekeeping, polling and delaying.

This module contains the ISRs of the configured timers, car switches and pedestrian buttons. Switches and buttons are configured as external interrupts, and they set global flags, start timers and display states on the display.

Both ISRs are defined in `clock.c`: `HAL_GPIO_EXTI_Callback()` for the buttons/switches, `HAL_TIM_PeriodElapsedCallback()` for the interrupt-enabled timers.

The two remaining timers are used for timekeeping such as: light transitioning duration, delaying 5-seconds and delaying 20 or 30-seconds. Constants in `timer_config.h` consists of tick values, representing the duration of a task and/or delay, they are necessary per task requirements found in Appendix A.

Features:

1. **System Clock Configuration:**

   The STM32's clock runs at 80MHz. All timers are configured to use the system clock with a prescaler of 40,000 which reduces the timers' frequency down to 2kHz. This results in a tick occurring every 0.5ms, in each active timer.

   Each timer's Auto-Reload-Register (ARR) is configured to represent the maximum duration (in ticks) they count up to. For the interrupt-

enabled timers, one full count triggers an interrupt. For the polling-timers, a state change, global flag and/or delay is set. The possible ARR values for the 16-bit timers has to be within the range:
$0 - 65535$ $(2^{16} - 1)$

See Table 1 and Table 2 below for the full configuration of timers and constants.

2. **Timer Configuration:**

   TIM3: Toggles blue pedestrian lights, signaling waiting pedestrians.

   TIM4: Used during transitioning, delaying etc.

   TIM5: Ensures pedestrian lights remain green for a safe crossing duration.

   TIM15: Accommodates longer delays and accounts for transition times.

Timer	**ARR**
TIM3	250
TIM4	10,000
TIM5	30,000
TIM15	60,000

Table 1: Clock Module - Timer ARR Values

Constants	**Value** [ticks]	**Value** [s]
`toggle_Freq`	249	0.125
`orange_Delay`	5,999	3
`pedestrian_Delay`	9,999	15
`walking_Delay`	29,999	15
`transition_Time`	29,999	15
`red_delay_Max`	40,000 - `transition_Time`	20
`green_Delay`	60,000 - `transition_Time`	30

Table 2: Clock Module - Timing Constants

3.4 Main Program

The project's main program resides in the `traffic.c` file. The state-machine, which the project is built upon represents the core logic for managing traffic and pedestrian lights at two intersections. It uses a structured approach with defined states and transitions.

The system operates in four states:

1. `Intersection1` and `Intersection2`:

   Handles traffic and pedestrian management for their respective intersections. These states control transitions between active and inactive intersections.

2. `Wait20s`:

   Establishes a 20-second delay before transitioning from one active intersection to another, when cars are active at both intersections.

3. `Wait30s`:

   Implements a 30-second delay when no cars are active at any intersection, while accommodating for late arrivals.

The framework transitions between states based on conditions such as timer expirations, pedestrian button presses, and car activity at intersections.

3.5 Testing

To make sure each developed module and its functions worked properly, testing was done methodically throughout the project. A separate file `test.c`, was used to simulate various scenarios, confirming the traffic lights behavior under the same state-machine framework used in the final implementation. Testing revealed several edge cases, such as timing conflicts and incorrect initialization, which were resolved before the final implementation. The modular design allowed for easy identification and rectification of issues.

The main goals of testing were to validate:

- Individual functions expected output and logic implementation.
- The expected time-durations of all configured timers and declared constants.
- The project's adherence to the requirements of Tasks 1, 2 and 3.

When developing a module, each function prototype was tested individually in isolation to verify its logic and expected output. For example: `set_pin()` and `clear_pin()` had to be tested to make sure that the bitwise operations worked properly, and no other lights were altered. This also confirmed the shift register pin definitions correctly represented each traffic light LED.

The hardware timers were verified using LEDs and the OLED display to check that delays matched the configured and expected duration.

Scenarios with invalid inputs such as: pressing a pedestrian button while its crosswalk was green were tested to confirm no faulty input was allowed, or in some cases properly handled.

STM32CubeIDE's debugging tool accessed via ST-Link, was heavily used during testing. The tool helped monitor global variables using the built-in 'Live Expressions' viewer. It also verified correct interrupt-triggering and exits, the timing of different sequences and accuracy of constants such as `orange_Delay`. The debugging tool was essential during development, it drastically reduced the time and effort required to identify and resolve issues in function and logic implementation.

4 Results

The project successfully achieved its goal of simulating realistic traffic lights for vehicles and pedestrians. The final implementation met all specified requirements for Tasks 1, 2 and 3. The system effectively manages task scheduling and synchronization using hardware timers and interrupts instead of FreeRTOS. Timing constants and time-driven functionalities, such as delays and transitions, performed as expected with a precision of $\approx\ \pm 0.5s$.

The state-machine framework allowed seamless transitions between traffic and pedestrian signals, dynamically handling pedestrian requests while maintaining vehicle traffic flow. Scenarios such as a pedestrian pressing a button during an ongoing green light were handled.

The SSD1306 OLED display provided as expected, real-time feedback with updates on car activity and pedestrian crossing statuses. Blue waiting indicators and green pedestrian signals were synchronized with traffic light transitions.

5 Discussion

The modular implementation of the software including, shift register control, display management and clock/timer configuration, contributed to the project's success by simplifying debugging and testing. This modular design enabled scalability and easy troubleshooting.

This project presented a mix of enjoyable challenges and significant learning opportunities. The most difficult aspect was finding an effective solution for task scheduling without the use of FreeRTOS. Once I identified hardware timers and external interrupts as the primary tools for managing scheduling, the implementation process became more manageable.

In conclusion, the project showcased a reliable and robust traffic light management system, achieving all its objectives and operating smoothly under various simulated scenarios. The implementation not only fulfilled all specified requirements but also provided a scalable foundation for future enhancements.

5.1 Side Note

However, some ambiguities in the task requirements impacted my ability to complete the project on time. Specifically, requirements R1.4 and R1.5 were in my **opinion** not clearly defined and seemed to contradict each other. R1.4 did not explicitly state whether the 'walking_delay' duration was strictly tied to a pedestrian button press. On the other hand, R1.5 stated that "The pedestrian signal must be red when the car signal of each active lane that crosses the crosswalk is either green or orange, and green otherwise", without any mention of a mandatory duration for the pedestrian signal to remain green [Figure 3]. If R1.4 is indeed meant to apply only after a pedestrian button press, it should have been explicitly clarified.

In my implementation, I chose to allow the crosswalk for inactive car-lanes to remain green by default, independent of a pedestrian button press. Additionally, my implementation handles transitions when a pedestrian requests to cross an active lane. For example, if lane 1 and 3 are green, lane 2 and 4 are red, and a pedestrian requests to cross lane 1 and 3, the traffic light system transitions as follows:

1. Lane 1 and 3 transition to red.
2. The crosswalk light for lane 1 and 3 turns green, while the crosswalk light for lane 2 and 4 turns red.
3. After 5 seconds lanes 2 and 4 turns green.

If walking_delay is required to be applied universally, regardless of a pedestrian button press, it would create an unrealistic scenario where a crosswalk might remain red indefinitely without justification. Such behavior would not align with real-world traffic systems, where pedestrian crossing should remain accessible when no conflicting traffic is present.

While I successfully implemented a solution that balances realism and functionality, clearer requirements would have reduced ambiguity and perhaps allowed me to submit the project in time.

Appendix A Project Requirements

Implementation Task ①

This implementation task focuses on the upper pedestrian crossing of the traffic light shield. Only the traffic lights shown in Fig. 4 are considered in this task. Cars are simulated by the respective switches on the road.

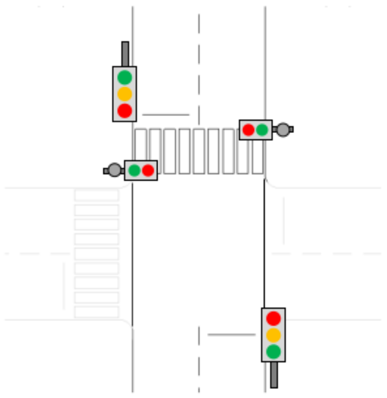

Figure 2: Sketch of the active components in implementation task 1.

The concrete requirements of this implementation task are as follows:

R1.1 At initialization, the pedestrian crossing shall be red and the car signal green.

R1.2 After a user pushed the button of the crosswalk, the indicator light shall toggle with a frequency of `toggleFreq` ms until the pedestrian crossing is green.

R1.3 All car signals of each active lane that cross the crosswalk shall be red `pedestrianDelay` ms after the pedestrian pushed the button.

R1.4 The pedestrian signal stays green for `walkingDelay` ms.

R1.5 The pedestrian signal must be red when the car signal of each active lane that crosses the crosswalk is either green or orange, and green otherwise.

R1.6 A car signal transitions from red to orange to green, or from green to orange to red, remaining orange for `orangeDelay` ms.

Figure 3: Project Requirements - Task 1

Implementation Task (2)

This implementation task focuses on the road crossing of the traffic light shield, ignoring the pedestrian crossings. Cars are simulated by the switches on the road. For simplicity, we assume a car is only allowed to drive forward to take a right turn.

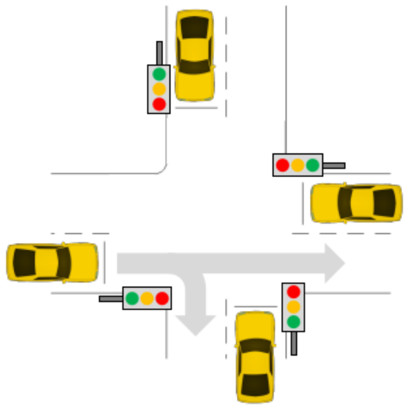

Figure 3: Sketch of the active components in implementation task 1.

The concrete requirements of this implementation task are as follows:

R2.1 Cars from each direction can either go forward, or turn right. Left turns are not allowed.

R2.2 Traffic lights must be set in a way that no two cars have overlapping possible paths.

R2.3 A car signal transitions from red to orange to green, or from green to orange to red, remaining orange for `orangeDelay` ms.

R2.4 If there is no active car in any direction the allowed (i.e. green) direction changes every `greenDelay` ms.

R2.5 A traffic light remains green if there are active cars in either allowed direction and no active cars are waiting on red traffic lights.

R2.6 If a car arrives at a red light and there are active cars in either allowed direction it waits `redDelayMax` ms until the signal has changed to green.

R2.7 If a car arrives at a red light and there are no active cars in either allowed direction the signal transitions immediately to green.

R2.8 At initialization the vertical lane shall be green and the horizontal lane shall be red.

Figure 4: Project Requirements - Task 2

Implementation Task (3)

This implementation task combines the results of implementation task (1) and implementation task (2) to manage the complete traffic crossing, including both pedestrian crossings. Cars are simulated by the switches on the road. A modular design of the previous implementation tasks will allow for an easy integration of the different parts. In addition to the control of the traffic lights, this task requires to control the shift registers using the SPI interface.

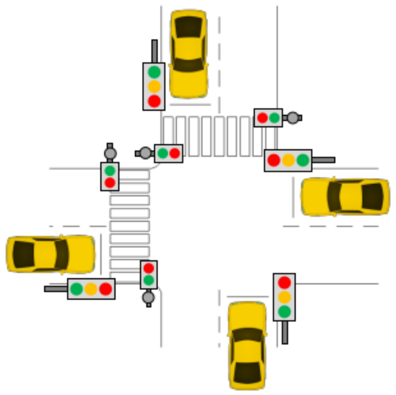

Figure 4: Sketch of the active components in implementation task 3.

The concrete requirements of this implementation task are as follows:

R3.1 Requirements of Task 1 for each crossing.

R3.2 Requirements of Task 2 for car crossing.

R3.3 Only one pedestrian crossing is green at a time.

R3.4 Shift register shall be controlled with the SPI interface of the microcontroller.

R3.5 A car is allowed to turn right when a crossing on the right lane is green.

Figure 5: Project Requirements - Task 3

Appendix B References

Websites

[1] STMicroelectronics, "STM32L476RG, Ultra-low-power with FPU Arm Cortex-M4 MCU 80 MHz with 1 Mbyte of Flash memory, LCD, USB OTG, DFSDM." [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32l476rg.html (visited on 12/01/2024).

[2] A. Kunalic, "IS1300-Embedded Systems," GitHub repository. [Online]. Available: https://github.com/Hazofinho/IS1300-Embedded_Systems (visited on 12/28/2024).

Datasheets

[3] Solomon Systech, "SSD1306, Advance Information 128 x 64 Dot Matrix OLED/PLED Segment/Common Driver with Controller," Rev. 1.1, Apr. 2008. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf (visited on 12/01/2024).

[4] Nexperia, "74HC595; 74HCT595," Rev. 12, Mar. 2024. [Online]. Available: https://assets.nexperia.com/documents/data-sheet/74HC_HCT595.pdf (visited on 12/01/2024).

[5] STMicroelectronics, "LIS2DW12, MEMS digital output motion sensor: high-performance ultralow-power 3-axis "femto" accelerometer," Rev. 9, Sep. 2024. [Online]. Available: https://www.mouser.se/datasheet/2/389/lis2dw12-1156431.pdf (visited on 12/06/2024).

[6] Sensirion AG, "Datasheet SHT20, Humidity and Temperature Sensor IC," Rev. 5, Oct. 2022. [Online]. Available: https://sensirion.com/media/documents/CCDE1377/635000A2/Sensirion_Datasheet_Humidity_Sensor_SHT20.pdf (visited on 12/06/2024).