IS1300 Embedded Systems PRO1 Report

Name: Morgan Bjälvenäs
Email: morganbj@kth.se
Course: IS1300 Embedded Systems HT23

Summary

Documenting the development of an embedded systems project for the Embedded Systems course at the Royal Institute of Technology, this report emphasizes the primary objective of meeting practical criteria through the interpretation and implementation of project requirements in embedded software development. Focused on creating a simulated traffic system, the project utilized the STM32CubeIDE software, Nucleo-L476RG board, and Traffic Light Shield, incorporating three implementation levels of increasing complexity.

The report covers the project's architecture, detailing the hardware components, software layers, and finite state machines governing traffic and pedestrian lights. The implementation adhered to Test Driven Development principles, systematically testing modules and ensuring code quality. FreeRTOS, integrated through STM32CubeIDE, facilitated multitasking with four distinct tasks managing traffic and pedestrian lights.

The final software achieved a synchronized and simulated traffic and pedestrian light system, meeting the all specified complexity levels. The report identifies potential improvements, such as alternative pedestrian signaling and additional features on the Traffic Light Shield for a more realistic and accurate traffic simulation. This project serves as a testament to a comprehensive understanding of principles governing embedded systems development.

Index

Introduction and Background

This document serves as the primary project report for the IS1300 Embedded Systems course at the Royal Institute of Technology. The overarching objective of this project is to satisfy the practical criteria inherent to the course, encompassing the interpretation of provided project requirements and realizing them through the development of embedded software. Additionally, the project addresses criterias, such as the formulation of a comprehensive project plan, design considerations, the execution of a testing strategy, and the utilization of a real-time operating system to implement a program for a time-critical embedded system. The project concludes with the documentation of the system in the form of this written report.

The objective of this project is the development of a traffic system utilizing the "STM32CubeIDE" software in conjunction with the provided Nucleo-L476RG board and the Traffic Light Shield. The project encompasses three discrete implementation levels, each representing different aspects of the traffic light simulation. These tasks range from managing pedestrian crossings to controlling traffic flow, and the complexity increases progressively for each implementation.

In addition to the software development, the project mandates a detailed project plan and testing strategy. The project plan is expected to include comprehensive architecture diagrams for both hardware and software components, along with supplementary diagrams representing additional architectural components. During the testing phase, adherence to the Test Driven Development approach is required.

Project Equipment

The equipment used in the project is the following;

- Nucleo-L476RG
- Traffic Light Shield
- USB-cable
- PC
- STM32CubeIDE

Project Architecture

This section describes the architecture of various components in the project both from a hardware and software standpoint.

Hardware

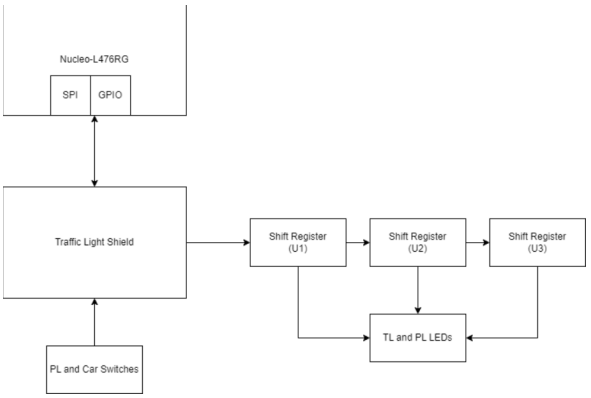

Figure 1: Hardware architecture block diagram implemented in project.

The hardware architecture of the project consists of the Nucleo-L476RG board and the Traffic Light Shield (see Fig 1.). Communication with these components is facilitated through a USB cable, with all interactions coordinated by a connected PC. The MCU establishes connectivity with the Traffic Light Shield through the GPIO and “SPI” ports on the Nucleo board. The GPIO ports on the Nucleo board are configurable to different IO modes through the utilization of the STM32CubeIDE software [1].

The relevant hardware components on the Traffic Light Shield are namely the “car” and “pedestrian light” switches. Additionally, three cascaded 8-bit shift registers are present on the Traffic Light Shield. The shift registers outputs are linked to the "traffic light" and "pedestrian light" LEDs.

Communication from the MCU to the LEDs is realized through the configuration of GPIO ports and the SPI interface, which is programmable using the STM32CubeIDE software. The SPI interface is instrumental in enhancing communication efficiency, as it enables the serial transmission of 24 bits to the interconnected shift registers in rapid succession.

Software

Software Layer Architecture

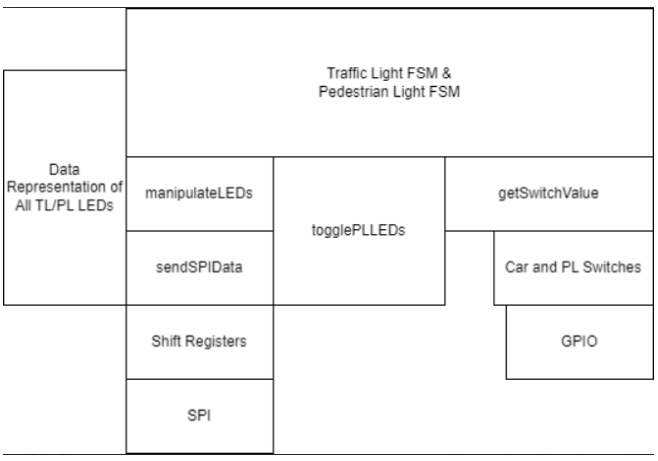

Figure 2: Software layer block diagram implemented in project

The software architecture for the software program used in the project can be sectioned into three layers (see Fig 2.). Positioned at the bottom is the foundational layer containing all the direct communication interfaces interfacing with the MCU. This layer encapsulates the Serial Peripheral Interface (SPI) and General Purpose Input/Output (GPIO) drivers, along with the data structures representing the state of the shift registers and the LEDs associated with "pedestrian and traffic lights".

The intermediate layer comprises all the functions and data structures indirectly manipulating the hardware components on the Traffic Light Shield. Functions within this layer would be functions manipulating the LEDs or functions returning the value of the current state of the switches on the Traffic Light Shield.

At the highest abstraction level resides finite state machines for the traffic lights and pedestrian lights, whose purpose is to orchestrate and govern the program's flow of execution. This hierarchical structure ensures a modular and organized approach to the software design of the project.

Finite State Diagrams

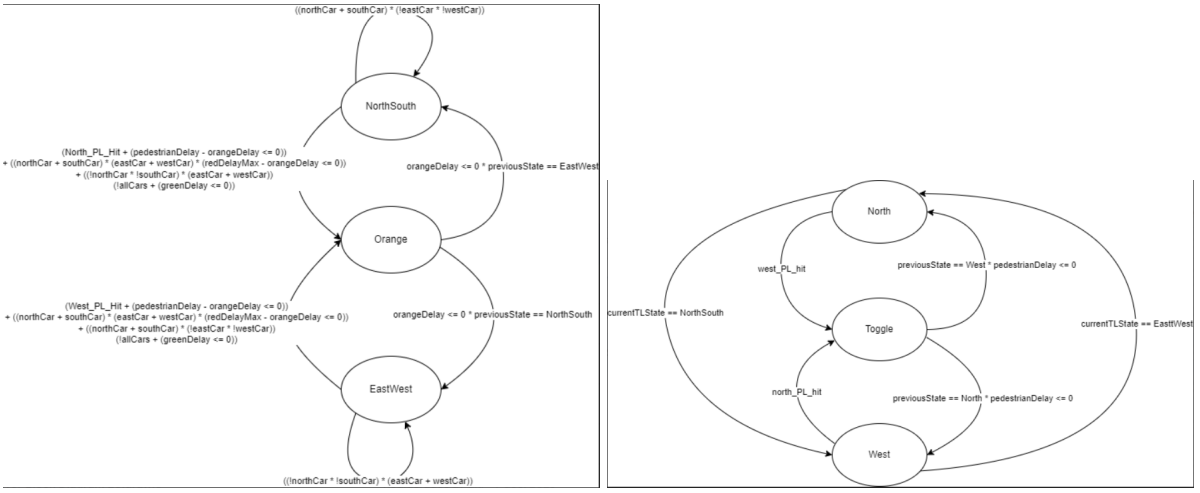

Figure 3 & 4: Traffic light and pedestrian light FSMs implemented in project

The traffic light FSM consists of three states; NorthSouth, Orange, and EastWest (see Fig 3.). During the NorthSouth state, the vertical traffic lights display a green signal, while the horizontal traffic lights display a red signal. Conversely, in the EastWest state, this configuration is inverted. During the Orange state, all traffic lights display an orange signal, in addition to all pedestrian lights emitting a red signal. The different state transitions are determined on the cars "active" and the elapsed time in each state.

The pedestrian light FSM comprises three states; North, Toggle, and West (see Fig 4.). The North state dictates the pattern such that the north traffic light is green while the west traffic light is red, and this configuration reverses during the West state. During the Toggle state, one of the pedestrian light LEDs toggle depending on the previous state for a set amount of time before transitioning toward the next state.

These state machines synchronize with each other through mainly the pedestrian light FSM transitioning in accordance with the traffic light FSM. Moreover, such state transitions may also occur depending on the state of the pedestrian light switches. Both FSMs integrate internal timers to ensure synchronous state transitions, thereby facilitating a coordinated modulation of the traffic and pedestrian light systems.

Hardware and Software Integration

The intercommunication between hardware and software components is established via a USB cable, wherein a connected PC serves as an intermediary, facilitating data exchange. The microcontroller governs communication through its ports, regulated by dedicated GPIO and Serial Peripheral Interface drivers, facilitated through the STM32CubeIDE software. The SPI interface is configured in simplex mode, specifically "transmit only," ensuring unidirectional transmission from the microcontroller to the shift registers [2]. Configurable through this software, the GPIO and SPI ports provide a versatile mechanism for the transmission and reception of data to and from the microcontroller. This configuration not only ensures the controlled operation of ports but also contributes to a straightforward and adaptable means of data interaction with the microcontroller.

Test Driven Development of Software

The development of the project's software adhered to the principles of the Test Driven Development (TDD) methodology. The TDD approach involves a systematic structuring of software, with individual testing of modules conducted in a predetermined sequence [3]. This method is characterized by a recurring "Test Driven Development cycle," where the process initiates with the formulation of tests preceding the implementation of corresponding modules. The iterative nature of this cycle ensures that each module satisfactorily passes its associated test, thereby ensuring the code meets the initial specifications of the module. This software development practice provides a robust and flexible approach in developing high quality and reliable software.

The preliminary phase of software development was the modularization of the project application into various different components within a software layer diagram (see Fig 2.). This diagram indicates five distinct modules to test. The majority of the tests can be found in the test.c source file in the project source code. Additionally, function definitions and data structures of the tests can be found in the test.h header file in the project source code. The testing's main target was the functions contained within the "traffic_light_functions.c" source file.

The initial module subjected to testing was the "sendSPIData" component within the software layer diagram. This test is encapsulated within the function "test_sending_SPI_data()," designed to verify the precise transmission of specified data from the microcontroller to the receiving shift registers and associated LEDs. The objective of this

test was to validate the expected outcome, wherein the activation of all four red traffic light LEDs was intended.

Subsequent to the validation of the "sendSPIData" module, the ensuing testing phase targeted the "manipulateLEDs" component, as depicted in the software layer diagram. This test is nested within the function “test_LEDs()”. The test is designed to activate and deactivate specific LEDs on the Traffic Light Shield, thereby the expected outcome being all traffic and pedestrian light LEDs lighting up in a predetermined sequence.

Following the approval of the preceding test of the “manipulateLEDs” module, the next testing phase is directed towards the verification of the "getSwitchValue" component. Encapsulated within the function named "test_switches()," this evaluative procedure aims to analyze the return values emanating from the switch functions. Consequently, the outcome of this test is expected to reflect the states of the traffic and pedestrian light switches by manifesting corresponding indications on the red traffic and pedestrian LEDs.

The final modular test that is found within the “test.c” source file is nested within the “test_toggle()” function. This test is the focus of the next testing phase within the Test Driven Development cycle and analyzes the implementation of the module “togglePLLeds”. This test integrates various previous tested modules to effect the toggling of a specific blue pedestrian LED at a predetermined frequency and for a specified duration. The anticipated outcome of this test involves toggling of a designated blue pedestrian LED following the activation of the corresponding pedestrian light switch.

The final module subjected to testing comprised the finite state machines governing both the traffic and pedestrian lights. However, unlike the preceding tests, this module lacked distinct and concrete modular tests due to the distributed freeRTOS implementation of the state machines across independent tasks. Consequently, the formulation and execution of modular tests became challenging within this context. Instead, the testing phase adopted an approach aligned with the specified complexity levels outlined in the project description. The modular testing procedure involved the systematic validation of all concrete requirements associated with each complexity level and then the module as a whole before progressing to the subsequent level, restarting the procedure. Testing the module as a whole before advancing to the next implementation level is essential in finding elusive bugs in the code by testing as many scenarios/corner cases as possible. This testing method was implemented from implementation level one through three delineated in the project description.

Software Implementation

RTOS Implementation

The “STM32CubeIDE” software provides an open source real time operating system kernel by the name of FreeRTOS, designed to facilitate the development of embedded systems. This provides a multitasking environment, allowing concurrent execution of multiple threads within the context of embedded applications [4].

The project software application is implemented through FreeRTOS, and sectioned into four distinct tasks. One task is dedicated to the implementation of the finite state machine

governing the traffic lights (see Fig 3.), configured with a periodicity of 100 milliseconds and assigned a normal priority. In parallel, another task is allocated for the finite state machine managing pedestrian lights, possessing the same priority and period. This concurrency is achieved through a round-robin scheduling strategy, thereby enabling the execution of these tasks in a seemingly simultaneous manner.

Conversely, the remaining two tasks are devised to execute a toggling algorithm for their respective blue pedestrian light LEDs. These tasks are characterized by a period corresponding to the toggle frequency in milliseconds and are assigned a lower priority. The deliberate assignment of a lower priority to the toggle tasks in comparison to the finite state machine tasks ensures that the latter will preempt the former in the event of unforeseen occurrences, such as a toggle task surpassing its given deadline.

Additional Implementation

Throughout the project software development, it became apparent that certain scenarios, particularly in corner cases, posed challenges for pedestrians to have adequate time to cross the crosswalks. To address this issue, a solution was implemented to guarantee that the green pedestrian light remains active for a minimum duration specified by the "walkingDelay" parameter before traffic light switch. This approach ensures that pedestrians consistently have a sufficient timeframe to complete the crossing, thereby enhancing the realism of the traffic system simulation.

Result

The final developed project software realizes a simulated and synchronized traffic and pedestrian light system on the designated Traffic Light Shield, implementing the entire complexity delineated in the project description. This achievement signifies the harmonized operation of traffic and pedestrian lights, incorporating timing conditions contingent upon the presence of active switches known as "cars" or "pedestrians”.

Potential limitations within the project outcome include instances where the pedestrian LEDs exhibit a constant green signal when the corresponding traffic LEDs display a red signal. A more realistic implementation, akin to conventional traffic light systems, would involve the activation of the pedestrian LED's green signal only upon the initiation of a pedestrian signaling to cross. This necessitates a higher level of complexity in the program implementation, for example entailing the incorporation of two distinct finite state machines dedicated to each pedestrian light LED. The proposed augmentation in complexity provides a more realistic simulation of real world traffic systems.

A potential enhancement for future implementations within this project involves augmenting the Traffic Light Shield with additional features, such as switches or LEDs that convey information regarding the amount of active cars on each lane. This modification should give a more precise assessment of the duration for each car switch to remain active, preventing the need for excessive manual toggling by the user. Consequently, this augmentation would facilitate a more realistic and automated simulation of a traffic system.

Another possible improvement in future work involves the integration of pedestrian crossings on all four sides, aiming to achieve a more faithful representation of real-world traffic systems. The current implementation imposes constraints on pedestrian movement, disallowing direct crossings from the north to south or east to west. The proposed improvement seeks to address this limitation, fostering a more realistic simulation wherein pedestrians are afforded the opportunity to traverse the traffic system in a manner consistent with actual pedestrian pathways in urban environments.

References

[1] STMicroelectronics, “RM0351, Reference manual, STM32L47xxx, STM32L48xxx, STM32L49xxx and STM32L4Axxx advanced Arm®-based 32-bit MCUs”, ST, version 9, p. 300-302, Available: https://www.st.com/resource/en/reference_manual/dm00083560-stm32l47xxx-stm32l48xxx-stm32l49xxx-and-stm32l4axxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[2] STMicroelectronics, “RM0351, Reference manual, STM32L47xxx, STM32L48xxx, STM32L49xxx and STM32L4Axxx advanced Arm®-based 32-bit MCUs”, ST, version 9, p. 1452-1454, Available: https://www.st.com/resource/en/reference_manual/dm00083560-stm32l47xxx-stm32l48xxx-stm32l49xxx-and-stm32l4axxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[3] James Grenning, “Applying Test Driven Development to Embedded Software”, IEEE Instrumentation & Measurement Magazine, 2007, Available: https://ieeexplore-ieee-org.focus.lib.kth.se/stamp/stamp.jsp?tp=&arnumber=4428578&tag=1

[4] STMicroelectronics, “UM1722, User manual, Developing applications on STM32Cube with RTOS”, ST, version 3 (2019), p. 1, Available: https://www.st.com/resource/en/user_manual/dm00105262-developing-applications-on-stm32cube-with-rtos-stmicroelectronics.pdf