Traffic intersection simulator

Embedded Systems
IS1300
KTH Royal Institute of Technology

Aron Tiselius

December 20, 2023

Summary

This document describes the development of a traffic light simulator using a Nucleo-L476RG and a custom Traffic Light Shield. The project aims to allow the control the traffic lights in a junction with 4 cars and 2 pedestrian crossings. The paper outlines what has been done, the methods used, the structure of the program, how the program was tested and briefly discusses the results of the project.

Contents

1 Introduction **3**
1.1 Problem Description 3
1.2 Requirements 3

2 Method **4**
2.1 Equipment and Tools 4
2.2 Hardware architecture 4
2.3 Software architecture 5
2.3.1 Modules 6
2.3.2 Tasks 6
2.4 Testing and planning 8

3 Results **9**
3.1 Restrictions 9
3.2 Improvements 10

4 References **10**

5 Appendixes **11**
5.1 Appendix A 11

1 Introduction

1.1 Problem Description

The project calls for the development of a traffic light control simulation using the Nucleo-L476RG development board and a custom Traffic Light Shield. The goal of the project is to implement various tasks that were given by the course examiner. The project also requires that a written report is submitted. The timeline of the project spans ~ 5 weeks.

There were three tasks that could be implemented. For full points all three had to be implemented. These requirements are en excerpt from the given project document. [1]

1.2 Requirements

1. Implementation Task 1
   At initialization, the pedestrian crossing shall be red and the car signal green.
   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.
   All car signals of each active lane that cross the crosswalk shall be red `pedestrianDelay` ms after the pedestrian pushed the button.
   The pedestrian signal stays green for `walkingDelay` ms.
   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.
   A car signal transitions from red to orange to green, or from green to orange to red, remaining orange for `orangeDelay` ms.

2. Implementation Task 2
   Cars from each direction can either go forward, or turn right. Left turns are not allowed.
   Traffic lights must be set in a way that no two cars have overlapping possible paths.
   A car signal transitions from red to orange to green, or from green to orange to red, remaining orange for `orangeDelay` ms.
   If there is no active car in any direction the allowed (i.e. green) direction changes every `greenDelay` ms.
   A traffic light remains green if there are active cars in either allowed direction and no active cars are waiting on red traffic lights.
   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.

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

3. Implementation Task 3
   Requirements of Task 1 for each crossing.
   Requirements of Task 2 for car crossing.
   Only one pedestrian crossing is green at a time.
   Shift register shall be controlled with the SPI interface of the microcontroller.

2 Method

2.1 Equipment and Tools

Hardware used:

- Nucleo-L476RG. [2] This microcontroller was used for processing and interfacing between peripherals.
- Custom Traffic Light Shield [3]. The shield was mounted on the microcontroller and acted as the primary interface for the project. The board houses all the buttons, switches and LEDs (as well as the shift registers that control the LEDs) that are used in the project. The board also contains many other components not used in this project.

The software was developed using STM32CubeIDE - Integrated Development Environment for STM32. [4]
The IDE was used for all code writing and debugging, and served as a convenient way to interact with the microcontroller.
The project utilizes the Real Time Operating System FreeRTOS. [5]
The OS was used to control the program by scheduling the tasks.

2.2 Hardware architecture

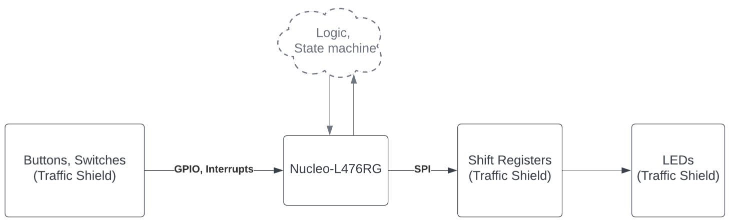


Figure 1: Block diagram of the hardware architecture

The general architecture of the hardware is quite simple. The user creates an input by pushing a button or flicking a switch, this sends a signal to one of the GPIO pins on the Nucleo, which triggers an interrupt. After some logic in the software, information is sent back to the Traffic Shield using SPI. This information is fed into the three, daisy chained shift registers on the Traffic Shield, which in turn lights up the correct LEDs.

2.3 Software architecture

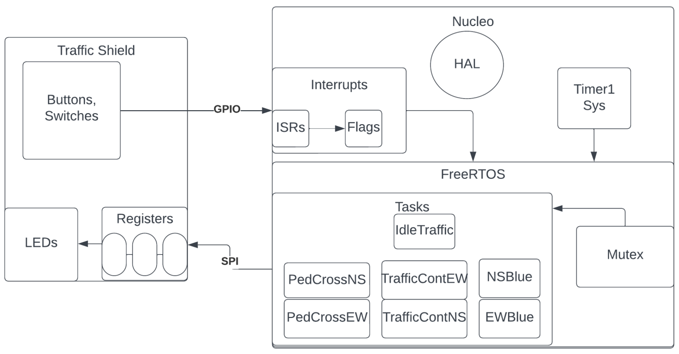


Figure 2: Software architecture

When an input from the Traffic Shield is recieved, an interrupt is triggered. There are several Interrupt Service Routines that more or less serve the same purpose. There is one ISR each for EW pedestrian button and the West car switch. The rest of the inputs share one ISR, and so positions have to be checked each time the ISR is entered. When exiting the ISR, a flag has been set or reset. For the switches, the flag is set to one when the switch is on, and 0 when it is not. For the buttons, the ISR only sets the flag to 1 when pressed. The flag is then reset manually in the FreeRTOS tasks.

There are also other 'flags' that are used to communicate between tasks. These are simple integers that eg. are set to 1 when the NS-traffic lights are turned on. These variables are crucial for the correct logic as they together with the ISR-flags communicate what state the program is in.

One mutex is used for the entire program. The mutex is used every time that a state is to be changed, with the exception of the tasks NSBlue and EWBlue. These tasks control the blue lights that blink when someone wants to use the crosswalk. They are allowed to do this without protection because they are the only ones with control over those lights. Further descriptions of the tasks are found in the Tasks section.

FreeRTOS's System Core Clock is powered by Timer1 from the Nucleo, and the Hardware Abstraction Layer runs under the surface to ease with the low-level stuff.

2.3.1 Modules

STM32CubeIDE generates quite a lot of code in different files for the project. These are files such as 'gpio.c', which handles the setup of the GPIO pins. Other than these, there were a few files where most of the work was done. This section gives an overview of how the project is structured in terms of different files.

- **functions.c**. This file contains all functions that were created to be used in the program. This includes functions for setting and resetting the LEDs, turning on/off parts of the intersection, etc. These functions are called from 'freertos.c':
- **freertos.c**. Contains some initialization for FreeRTOS and all the tasks created.
- **stm32l4xx_it.c**. This file is used to handle all interrupts by the system.
- **variables.c**. Contains all global variables used in the project. Included here are variables for the different LEDs, flags, delays and a variable which holds information about the current lights that are turned on.
- **test.c** Contains the tests that test the functions in 'functions.c'. More about the tests in the section Testing and planning.

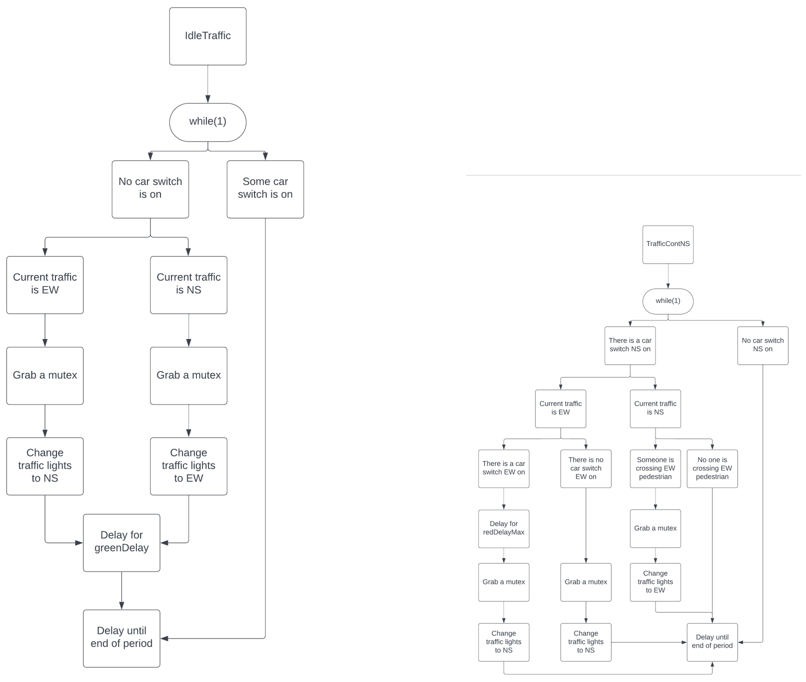
2.3.2 Tasks

There are in total 7 tasks developed. This section gives a quick overview of all tasks, their purpose and a basic description of how they were implemented.

- **IdleTraffic**. This task controls the traffic and pedestrian lights when there is no car that is waiting. The task first checks if there are any active switches, and if not toggles the traffic and then waits for `greenDelay` ms until it repeats.
- **TrafficContNS**. This task's purpose is to determine if the traffic lights should change when there is a car on the NS road. The task checks if there are cars at the NS road, EW road and if there is an active 'crossing' at the EW pedestrian crosswalk to determine what should happen.
- **TrafficContEW**. This task does the same thing as TrafficContNS but for the EW road.

- **PedCrossNS**. This task checks if the NS crosswalk button has been pressed, and if so, makes sure that the crosswalk eventually turns green for `walkingDelay` ms. This task is also responsible for resetting the button's flag. The task is not responsible for making sure the traffic lights are correctly turned on/off. This is handled by the previous tasks. The communication from the PedCrossNS task is however crucial to function properly.
- **PedCrossEW**. This task does the same thing as PedCrossNS but for the EW crosswalk.
- **NSBlue**. This task is responsible for making sure the blue NS crosswalk light starts blinking when the respective button is pressed, and stops blinking when the crosswalk light is green. This is simply done by checking if the button is pressed, and if the crosswalk has been turned on. The blue light blinks at a frequency of `toggleFreq` Hz. This is done by setting the task's period to $\frac{1000}{\texttt{toggleFreq}}$ ms.
- **EWBlue**. This task does the same thing as NSBlue but for the EW crosswalk.

Below are flowcharts for two of the tasks. There are flowcharts for the rest of the tasks in Appendix A.


IdleTraffic
while(1)
No car switch is on
Some car switch is on
Current traffic is EW
Current traffic is NS
Grab a mutex
Grab a mutex
Change traffic lights to NS
Change traffic lights to EW
Delay for greenDelay
Delay until end of period
TrafficContNS
while(1)
There is a car switch NS on
No car switch NS on
Current traffic is EW
Current traffic is NS
There is a car switch EW on
There is no car switch EW on
Someone is crossing EW pedestrian
No one is crossing EW pedestrian
Delay for redDelayMax
Grab a mutex
Grab a mutex
Grab a mutex
Change traffic lights to EW
Change traffic lights to NS
Change traffic lights to NS
Delay until end of period

(a) IdleTraffic flowchart (b) TrafficContNS flowchart

Figure 3: Flowcharts of two of the tasks developed

2.4 Testing and planning

The project was developed by using *Test Driven Development*, TDD. All functions were tested in 'test.c'. These tests were written first, and then the function(s) used in that test were developed. The functions were declared working when the tests were completed as expected. To call the test functions, functions relating to FreeRTOS were commented out from the main file to avoid FreeRTOS from starting up. Then, in the main while-loop, the function `tester()` was called, which in turn calls the tests themselves.

Some tests were done using the debugger in combination with the function 'Live Expressions' to make sure flags were changed correctly.

The planning was done by first drawing up the general idea of the flow of the program as well as the software architecture. However, as the development progressed, it became clear that some parts of the planning was not properly thought out. For example, the initial idea for protecting/synchronizing tasks was to use three mutexes. This turned out to be more convoluted than necessary, so the structure was redrawn with just one mutex. In general, when it was time to write the actual task, planning had to be redone in more detail. The reason for this was insufficient experience/knowledge in creating this type of program. Had the project been redone, the planning would've been able to be done more elaborately, speeding up and simplifying the development.

3 Results

The three tasks were successfully implemented, with the exception of one subtask, highlighted in the subsection below. The program effectively simulates the traffic intersection with car traffic from north, south, west and east, as well as two pedestrian crosswalks. No possible paths ever intersect, avoiding collisions. Traffic lights transition with proper timing and always gives cars time to react to a change in lights.

3.1 Restrictions

The only subrequirement that was not completed properly is the requirement which says car signals that intersect the crosswalk should be red `pedestrianDelay` ms after the button has been pressed. This is because it does not seem as though that requirement is compatible with other subrequirements. It seems that the combination of this requirement, "The pedestrian signal stays green for `walkingDelay` ms", "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." and "Only one pedestrian crossing is green at a time." can not all be true. If both crosswalks are pressed roughly at the same time, for example, the first crosswalk would be green after `pedestrianDelay` ms since the intersecting traffic would be red. Shortly after, the other crosswalk would turn green. Now, either both crosswalks are green, which they are not allowed to be, or the first crosswalk has to been made red, which violates the `walkingDelay` requirement.
It seems there is no way to successfully implement all of these requirements. The most reasonable way of handling this was determined to be allowing longer wait times before the crosswalk turns green.

Another restriction is the fact that cars cannot turn left. This is a feature of the intersection that might be desired. A third restriction is that unnecessary state changes occur sometimes. For example, after the NS crosswalk has been green for `walkingDelay` ms, and before the crosswalk was activated there was just about to be a state change from EW traffic to NS traffic, the NS traffic lights will turn red, and then almost instantly turn green again. It would probably be better if the traffic light stayed green for his short period.

One final restriction is that there is no warning for pedestrians when the crosswalk light is about to turn red. This is obviously not very safe since pedestrians could be in the middle of the crosswalk when the light turns red, and the overlapping traffic light turns green.

3.2 Improvements

To improve the program, a couple things could be done.

- Allow for the option of a left turn. Even if a specific intersection does not want to implement it, some other intersection might. This would allow for the program to be used on more intersections.
- Implement a warning for pedestrians. For example, the green light could flash for a few seconds before it is about to change.

4 References

References

[1] Becker, Matthias. "Project Task IS1300 HT23" Canvas, KTH Royal Institute of Technology, November 13. 2023, https://canvas.kth.se/courses/43011/files/7220482?wrap=1

[2] STM32 Nucleo-64 development board with STM32L476RG MCU, supports Arduino and ST morpho connectivity, STMicroelectronics, n.d., https://www.st.com/en/evaluation-tools/nucleo-l476rg.html

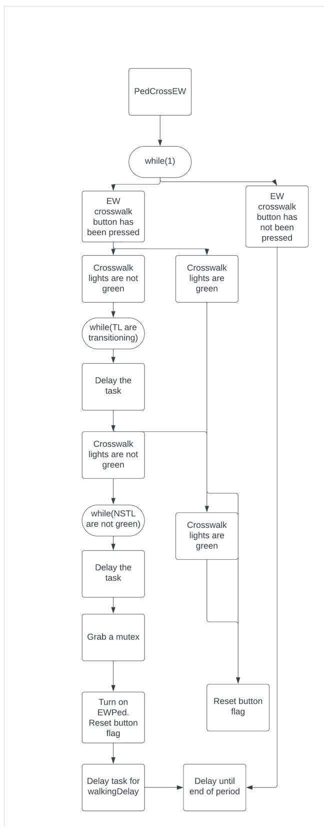
[3] Becker, Matthias. "Traffic Light" Canvas, KTH Royal Institute of Technology, December 15. 2021, https://canvas.kth.se/courses/43011/files/7020061?wrap=1

[4] Integrated Development Environment for STM32, STMicroelectronics, n.d., https://www.st.com/en/development-tools/stm32cubeide.html

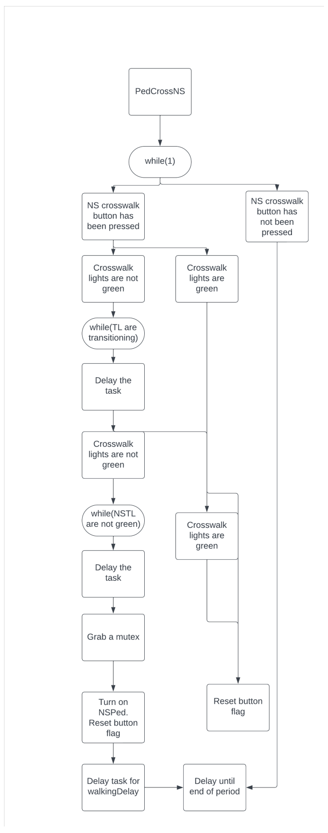
[5] FreeRTOS, FreeRTOS, n.d., https://www.freertos.org/

5 Appendixes

5.1 Appendix A

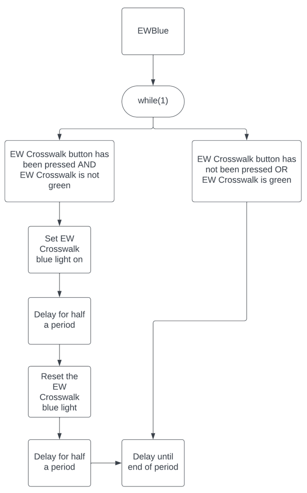

PedCrossEW
while(1)
EW crosswalk button has been pressed
EW crosswalk button has not been pressed
Crosswalk lights are not green
Crosswalk lights are green
while(TL are transitioning)
Delay the task
Crosswalk lights are not green
while(NSTL are not green)
Crosswalk lights are green
Delay the task
Grab a mutex
Reset button flag
Turn on EWPed. Reset button flag
Delay task for walkingDelay
Delay until end of period

(a) PedCrossEW flowchart

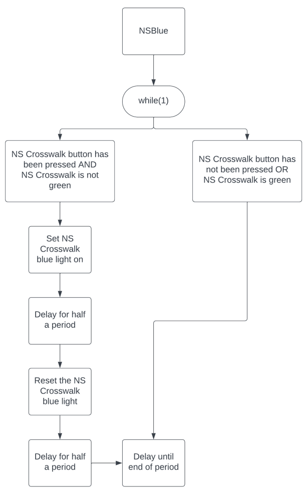

PedCrossNS
while(1)
NS crosswalk button has been pressed
NS crosswalk button has not been pressed
Crosswalk lights are not green
Crosswalk lights are green
while(TL are transitioning)
Delay the task
Crosswalk lights are not green
while(NSTL are not green)
Crosswalk lights are green
Delay the task
Grab a mutex
Reset button flag
Turn on NSPed. Reset button flag
Delay task for walkingDelay
Delay until end of period

(b) PedCrossNS flowchart

Figure 4: Flowcharts of PedCrossXX


EWBlue
while(1)
EW Crosswalk button has been pressed AND EW Crosswalk is not green
EW Crosswalk button has not been pressed OR EW Crosswalk is green
Set EW Crosswalk blue light on
Delay for half a period
Reset the EW Crosswalk blue light
Delay for half a period
Delay until end of period

(a) EWBlue flowchart


NSBlue
while(1)
NS Crosswalk button has been pressed AND NS Crosswalk is not green
NS Crosswalk button has not been pressed OR NS Crosswalk is green
Set NS Crosswalk blue light on
Delay for half a period
Reset the NS Crosswalk blue light
Delay for half a period
Delay until end of period

(b) NSBlue flowchart

Figure 5: Flowcharts of XXBlue

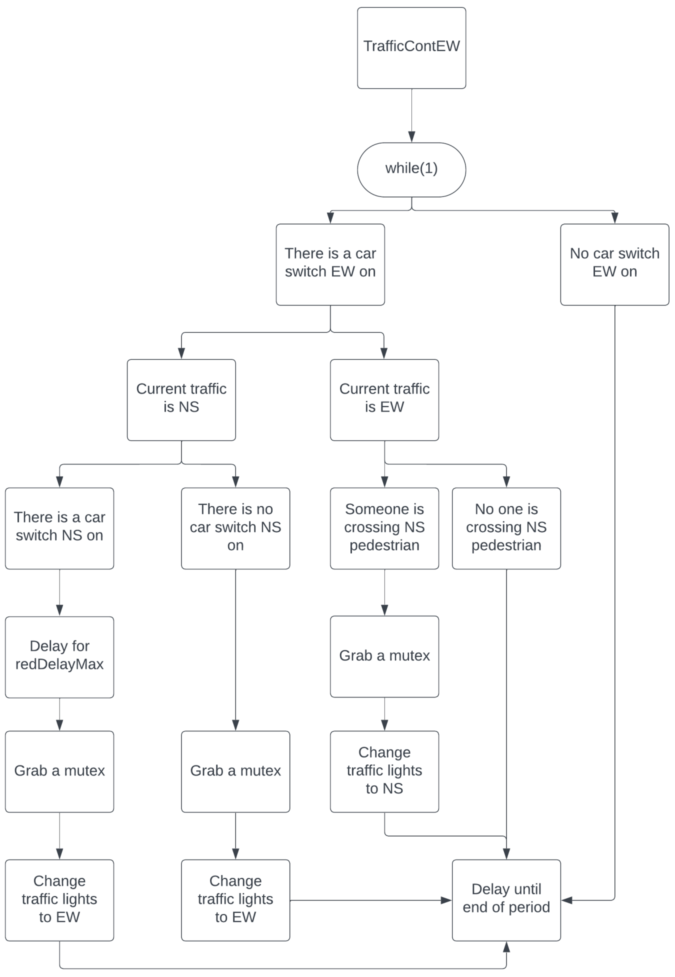

TrafficContEW
while(1)
There is a car switch EW on
No car switch EW on
Current traffic is NS
Current traffic is EW
There is a car switch NS on
There is no car switch NS on
Someone is crossing NS pedestrian
No one is crossing NS pedestrian
Delay for redDelayMax
Grab a mutex
Grab a mutex
Grab a mutex
Change traffic lights to NS
Change traffic lights to EW
Change traffic lights to EW
Delay until end of period

Figure 6: TrafficContEW flowchart