Department of Control for Transportation and Vehicle Systems
Faculty of Transportation Engineering and Vehicle Engineering
Budapest University of Technology and Economics

Development of PLC-based traffic light control software for flexible testing of autonomous vehicles

Students' Scientific Conference Paper by:
Tamás Wágner
PGJ49N

Supervisor:
Dr. Balázs Varga
Department of Control for Transportation and Vehicle Systems
Budapest University of Technology and Economics

November 7, 2023

Abstract: The research work aims to overcome the limitations of current traffic light control systems by introducing an innovative and flexible PLC-based traffic light control platform adapted to automotive proving ground environment. The ZalaZONE Proving Ground is the focus of the presented system, which will operate centrally managed traffic lights in the near future while allowing full control at different intersections by a local PLC (Programmable Logic Controller). Existing industrial traffic light control systems have been developed for traditional, human-driven vehicles, therefore they are insufficient to meet the requirements of testing automated driving systems. The project objective is to address these limitations by developing a flexible platform supporting the testing and development processes and the safe integration of automated or autonomous vehicles into the existing road infrastructure. The main focus is on guaranteeing the safe operation of the system while ensuring compliance with international standards, and optimizing the real-time operation of the system. Users of the system will be mainly automotive professionals who will carry out measurements on test tracks and thus need an easy-to-use, flexible, and reliable system. The proposed system allows the testing of different traffic scenarios that conventional systems cannot fully support.

Absztrakt: A dolgozat célja a jelenlegi közúti forgalomirányító (jelzőlámpás) berendezések képességeinek kiterjesztése egy rugalmas PLC-alapú irányító platform fejlesztésével, amely az automatizált járművek tesztelési igényeihez is igazodik. A ZalaZONE Járműipari Tesztpálya áll a bemutatott rendszer fókuszában, ahol a közeljövőben központilag felügyelt, PLC (programozható logikai vezérlő) alapú közlekedési lámpák fognak üzemelni a Smart City zónában tesztelési feladatok kiszolgálása céljából. A iparban elterjedt jelzőlámpa-irányító rendszereket hagyományos járművek számára fejlesztették ki, így nem alkalmasak az automatizált vezetéstámogató rendszerek vagy akár teljesen autonóm járművek által támasztott követelmények kielégítésére. A projekt célja, hogy ezeket a korlátokat egy olyan nyílt és adaptálható platform kifejlesztésével orvosolja, amely elősegíti az automatizált járművek további fejlődését és biztonságos integrációját a meglévő közúti infrastruktúrába. A fő hangsúly a rendszer biztonságos működésének garantálásán, a nemzetközi szabványoknak való megfelelés biztosításán és a rendszer valós idejű működésének optimalizálásán van. A rendszer felhasználói elsősorban olyan szakemberek lesznek, akik tesztpályákon végeznek méréseket és fejlesztéseket, és akiknek ilyenformán rugalmasan használható és megbízható rendszerre van szükségük. Az új rendszer olyan forgalmi szcenáriók tesztelését teszi lehetővé, amelyeket a hagyományos rendszerek nem képesek teljesen kiszolgálni.

Keywords: PLC, Automotive Proving Ground, Traffic Light Controller, HiL Test, V2X, Automated Vehicles

Acknowledgements

Supported by the ÚNKP-23-2-1-BME-250 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

ABS	Anti-lock Braking System
ADC	Ampere/Current (Direct Current)
AES	Advanced Electronic Signature
API	Application Programming Interface
COM	Component Object Model
DCOM	Distributed Component Object Model
CPU	Central Processing Unit
FB	Function Block
GLOSA	Green Light Optimal Speed Advisory
HiL	Hardware-in-the-Loop
HMI	Human Machine Interface
I2C	Inter-Integrated Circuit
I/O	Input/Output
LED	Light-Emitting Diode
MiL	Module-in-the-Loop
NTP	Network Time Protocol
OAEP	Optimal Asymmetric Encryption Padding
OPC UA	Open Platform Communications Unified Architecture
PiL	Processor-in-the-Loop
PLC	Programmable Logic Controller
POU	Program Organization Unit
RPi4	Raspberry Pi 4
RSA	Rivest–Shamir–Adleman
SHA	Secure Hash Algorithms
SiL	Software-in-the-Loop
TTL	Transistor-Transistor Logic
VDC	Voltage (Direct Current)
VFD	Variable Frequency Drive
V2X	Vehicle-to-Everything

Chapter 1

Introduction

1.1 Scope

The main aim of this project is to design a flexible traffic light control system for use across various proving grounds. The primary objective of this control system is to address the needs of proving grounds where self-driving vehicles and their associated functions are tested, such as the ZalaZONE proving ground.

As part of this, project a system that involves connecting and controlling multiple groups of traffic lights in varying manners will be developed. This traffic management system will be made up of two major components: a central control system and programmable logic controller (PLC) units placed at intersections to manage traffic lights. The development of the PLC subsystem is the main focus of our current work, discussed in this thesis.

The main advantages of our system include its ability to implement phase plans that would be impossible with traditional traffic control devices, as well as its ability to handle conflicting green signals. Another important benefit of this system is that it requires no special training to use.

Our goal is to develop this kind of system by gradually implementing and testing all components. As a first step of this project a local control system has been developed and tested, which is discussed in detail in this work. It is essential to comply with international safety and security standards, while at the same time ensuring proper communication with other systems.

The system implementation places a strong emphasis on utilizing commercially available hardware components. However, the software design offers functionality that is not present in conventional traffic management systems.

Professionals conducting measurements on proving grounds will be the primary users, requiring an easy-to-use, flexible and reliable system. Thanks to its design, the system is operational on almost all automotive proving grounds equipped with

PLC-controlled signal heads and other basic hardware such as reliable network devices.

1.2 Motivation

The project aims to demonstrate the necessity of an advanced system for testing autonomous vehicles in today's world. It primarily focuses on the dynamic requirements of the modern world, which traditional traffic management systems are failing to satisfy. Although current systems have been verified as safe and reliable, they cannot cope with the complexity and dynamics of modern traffic systems, especially with autonomous vehicles.

In the current technological era, 'smart' test tracks are gaining recognition for their ability to validate the advanced functionalities of autonomous vehicles [1]. The latest driver assistance systems like parking assistance, lane keeping and Anti-lock Braking System (ABS) have considerably improved vehicular safety and driving comfort. These systems have fulfilled their functions with no communication with other vehicles or infrastructure [2]. The automotive test tracks currently in widespread use are designed primarily to test these functions with the physical properties of vehicles.

Recent advancements in information and communication technologies have resulted in the development of various innovative features, including accident warning systems, Green Light Optimal Speed Advisory (GLOSA), and slow vehicle warning systems [3]. These technical marvels have paved the way for truly autonomous vehicles to emerge [4].

Despite recent advancements, the progress of traffic light control systems has conspicuously lagged. Our reliance on systems developed several decades ago has limited our ability to adapt. We can classify traditional traffic lights into two categories: fixed-time and adaptive. Adaptive traffic light controls are more versatile because they collect information and make decisions based on the data collected. However, most traffic lights can only estimate the number of vehicles at an intersection and their throughput demand locally [5, 6, 7]. A common alternative is to set up traffic management centers where traffic lights can be directly controlled, or phase plans can be configured [8].

The present traffic light control systems lack the necessary equipment to handle the crucial edge cases that are vital in the tests of self-driving cars. For instance, when traffic lights fail entirely, a situation that seldom arises or, in any event, arises only infrequently. Another example is when the system enables two opposing green signals at the intersection concurrently [3, 9]. To effectively conduct malfunctions for testing purposes, it is necessary to have complete control over the systems.

Additionally, the business-focused approach taken by manufacturers of traffic

light control systems often involves the creation of their communication protocols. This further complicates matters, necessitating the acquisition of a license for the traffic light controllers to interact with equipment developed by various manufacturers. These closed-system devices do not offer the flexibility and open-source applications needed to test highly customized scenarios effectively.

Considering these challenges, our main objective is to create and execute a modern, flexible traffic light control system using PLCs. This will provide the critical opportunity to facilitate comprehensive testing scenarios for self-driving vehicles that are essential for the rapidly changing technological landscape. Moreover, establishing an open and flexible system would promote progress in the autonomous vehicle industry and improve the safe integration of autonomous vehicles into the existing transport infrastructure.

The implementation of PLCs to manage traffic lights is emerging as a potential solution to these challenges. The proposal entails the deployment of a dedicated PLC at each intersection to manage the traffic lights, while enabling different modes of operation and enhanced functionalities. Although the proposed control system is designed to integrate several modes of operation, such as fail-safe stop, local fixed program and remote control, only the local fixed program mode is currently at a stage of development where it can be productively presented.

To ensure the successful completion of this project, great emphasis must be placed on accurate testing of the control system to ensure its safe operation. Optimization of the system is also essential to ensure its ability to operate in real-time, and thus provide a robust response to the complex and dynamic requirements of new transport systems.

Chapter 2

Methodology and Tools Used

2.1 PLC Programming: Basics of PLCs

PLCs are primarily embedded computers that are specifically designed to control and automate processes in manufacturing, production and other industrial environments. They have become extremely popular devices due to their robustness, reliability, and programming flexibility. Some of the more important features of PLCs are presented below, which will make it easy to see why this particular device family was chosen to control traffic lights [10, 11].

2.1.1 Design

The design of the PLCs is fundamentally modular. PLCs are composed of, among other things: backplane, power supply, central process unit (CPU), Input/Output (I/O) modules, communication modules.

The primary function of the backplane is to provide power and to transfer signals between modules [12]. It is primarily used in older systems because of the directly connected modules. Each manufacturer has its own proprietary solutions, which are often incompatible with each other.

The power supply is responsible for providing the appropriate voltage levels required for the operation of the PLC. Most PLCs can run on different voltage levels, including 3.3VDC, 5VDC, 12VDC, and 24VDC. However, the input voltage level of the power supply varies depending on the region and the site's type of electrical network. For instance, a factory might have a 24VDC busbar, but it may need to be supplied from a 3-phase network. Common input voltages are 120/240VAC and 24VDC [10, 11].

The central element in a PLC is the CPU, which executes the programmed code. The PLC CPU differs from the processors found in ordinary PCs in that it has integrated operating memory and storage. Recently, it has become prevalent to

use 32-bit processors in modern PLCs, which were once solely reserved for mobile devices, while older models often were limited to 16-bit operations.

PLCs with pre-installed I/O modules are available, which enable communication between the PLCs and a range of sensors and actuators. These modules may be either analog or digital; however, often there are no such modules on the PLCs, in which case additional I/O cards must be used. It is important to note that PLCs are typically designed to operate within a specified range of voltage levels and signal speeds for their inputs and outputs. If the voltage levels are incompatible or external devices are unreliable, it may be necessary to use galvanic isolation [13]. However, most modern PLCs are designed to protect the CPU.

Another key feature of PLCs is their communication modules. PLCs used to be primarily programmed through a serial port (RS232/485) in the past, but newer models predominantly use Ethernet ports. Nevertheless, serial ports are still present in some PLCs and are manufacturer-specific, serving mainly for diagnostic purposes. Additionally, some PLCs feature optical connectors that can significantly reduce communication latency [11].

2.1.2 PLC Types

PLCs are available in different designs to meet all kinds of requirements [10]:

- Nano PLC: Another name for these devices is Programmable Relay Controller. They have up to 15 I/O ports. They cannot be expanded with modules. It is possible to program some types without any extra devices.
- Micro PLC: These PLCs also have I/O ports (usually 12-32), and also communication interfaces. It is possible to further extend with modules.
- Standard PLC: These PLCs have a purely modular design and the number of I/O ports can be up to 512.
- Brick PLC: The largest size PLCs on the market. The maximum number of I/O ports is 4096. These systems are no longer used today.
- Special PLC: This includes PLCs of the *Ex* type, which are designed to withstand extreme conditions such as potentially explosive environments. The other type of special PLCs are the *Fail-Safe* PLCs, which are designed with redundant hardware elements to prevent failure.
- Soft PLC: SoftPLCs, or software programmable logic controllers, are PLC applications that run on general operating systems such as Linux or Windows. They offer the flexibility of running alongside other complex applications, enabling parallel execution of tasks like image processing or graphical user

interfaces. SoftPLCs can be deployed on a range of general-purpose hardware, including industrial PCs based on x86 architecture or even smaller platforms like Raspberry Pi.

2.1.3 Safety

In environments where the safety of people and equipment is a priority, the criticality of PLC to operate safely is particularly important. Consequently, a range of protective features have been incorporated into PLCs. Among these, redundancy takes a prominent place. This characteristic enables the installation of backup components in PLCs so that in case of a component failure, the backup takes over and prevents systemic failure. PLCs are also engineered to enter into a secure mode in the event of system failure, thereby reducing the potential risk of injuries to personnel [10, 11].

Additionally, with PLCs having monitoring and diagnostic features integrated, maintenance staff can receive timely notifications of any faults and undertake appropriate repair actions. The PLCs we used also have such functions, and the manufacturer of the PLC has even added a special monitoring option to the I/O modules, which can detect possible damage to the connected cables.

The role of cybersecurity is becoming increasingly important in our days given the frequency of hacking attacks, with attackers often targeting entire factories. Many traditional PLCs are easily hackable; therefore, it is important to operate them in an isolated network. However, the latest generation of PLCs now supports the use of asynchronous keys during login, which significantly reduces security risk.

2.1.4 Program Structure

PLC programs are typically written in one of the IEC 61131-3 [14] standard programming languages:

- Ladder Logic: This graphical language represents control circuits using symbols that resemble relay logic diagrams.
- Structured Text: This is a high-level, text-based language and allows for complex programming structures, such as loops and conditional statements.
- Function Block Diagram: This graphical language uses blocks to represent functions, such as timers, counters, and mathematical operations. The blocks are connected by lines, which represent data flow between the functions.
- Instruction List: This low-level, text-based language is similar to assembly language and consists of a series of instructions executed sequentially.

- Sequential Function Chart: This graphical language represents control programs as a series of interconnected steps and transitions, allowing for the clear representation of sequential and parallel processes.

These languages allow for a structured approach to programming, making it easier to develop, maintain, and troubleshoot the control program.

2.1.5 Special Features of PLCs

PLCs have special features that distinguish them from other computers and microcontrollers. These features make them ideal for industrial automation and control systems [10, 11, 15].

One of these key features is the real-time operation. PLCs are designed to process inputs and respond to them quickly and consistently at the outputs. This is critical in industrial applications, where accurate and timely control is essential to maintain efficiency and safety.

Another important feature is deterministic behavior, which means that PLCs perform tasks in a predictable and repeatable manner. This behavior ensures the consistent and reliable operation of the control system, even in complex and dynamic environments.

PLCs can also operate in harsh industrial environments with extreme temperatures, dusty and humid environments, and vibrations. This makes them ideal for a wide range of applications from the manufacturing industry to the oil and gas industry.

However, PLCs can be easily integrated with other industrial devices, such as human-machine interfaces (HMIs), variable frequency drives (VFDs), and remote I/O modules. This allows seamless communication and control between the different elements of an industrial automation system.

Modular and scalable designs are also an important feature of PLCs. This allows easy expansion and customization of the control system, as required, without the need for major reprogramming or hardware changes.

Finally, PLCs have robust communication capabilities, supporting a wide range of communication protocols, such as Ethernet, Modbus, and Profibus. This allows communication with other devices and systems in the industrial environment, enabling the centralized control and monitoring of multiple processes and equipment.

In summary, PLCs have unique features such as real-time operation, deterministic behavior, operation in harsh industrial environments, easy integration with other industrial devices, modular and scalable design, and robust communication capabilities, which make them an ideal choice for industrial automation and control systems. These features ensure that PLCs provide reliable, efficient, and safe control solutions for various applications.

2.1.6 I/O Cards

I/O cards or input/output modules are the basic components of PLC systems. They provide an interface between the PLC and the sensors and actuators involved in the process, enabling communication and control between devices [10, 11].

There are several types of I/O cards, such as:

- Digital input cards: These cards are used to read the signal of digital devices such as switches, push buttons, and proximity sensors. They typically detect ON/OFF states, which are represented as binary values (0 or 1).
- Digital output cards: These cards control digital devices such as solenoid valves, relays, and indicator lights. They send binary signals (0 or 1) to turn the devices ON or OFF, respectively.
- Analog input cards: These cards read continuous signals from analog devices such as temperature sensors, pressure transducers, and flow meters. The analog signals are converted into digital values that can be processed by the PLC.
- Analog output cards: These cards control analog devices, such as variable-speed drives, control valves, and analog meters. They generate continuous output signals based on the digital values provided by the PLC.

I/O cards can be built into the PLC or added as separate modules, allowing for system customization and expandability. This modular design allows users to configure the PLC system according to their needs and easily add or remove I/O cards as required.

In conclusion, I/O cards play an important role in PLC systems by providing an interface between the sensors and actuators of the PLC. Different types of I/O cards, including digital and analog input/output modules, allow the PLC to communicate with and control a wide variety of devices, thus ensuring the efficient and reliable operation of the control system.

2.2 Open Platform Communications Unified Architecture (OPC UA)

This protocol is based on the former OPC standard, which uses Microsoft's own COM/DCOM technology. However, the OPC Foundation has recognized that the increasing complexity and diversity of industrial automation systems require a more flexible, secure, and platform-independent communication standard. As a

result, the OPC UA was developed to address the limitations of OPC Classic and better meet the requirements of modern industrial control systems [16].

The main reasons for developing OPC UA were to

- Platform Independence: The OPC UA was designed to work on any platform, including Windows, Linux, and embedded systems, unlike OPC Classic, which relies on Microsoft COM/DCOM technology. This enabled the OPC UA to support a wide range of industrial applications.
- Enhanced security: The OPC UA has built-in security features, such as encryption, authentication, and access control, that guarantee data protection and the integrity of communication between devices and systems.
- Scalability: The OPC UA is highly scalable, capable of supporting systems of all sizes and complexities, from simple single-device applications to distributed control systems involving thousands of devices.
- Information modeling. This enables more efficient and valuable communication between the devices and systems.
- Interoperability: The OPC UA is designed to be compatible with a wide variety of devices and systems, regardless of the vendor or underlying technology. This allows seamless integration and communication between different devices and systems in an industrial environment.

2.2.1 How to Collect Data From OPC UA Nodes?

The OPC UA offers three basic operations in communication between the server and client: standard data reading, subscription function, and item monitoring.[16, 17]

During standard data reading, clients periodically poll the server to update the value of each item. This method requires the client to send regular requests to the server regardless of whether the values of the items have changed.

The subscription feature allows clients to receive notifications of changes in the values of monitored items. Here, the client subscribes to certain items and the server sends a notification only when their values change. This reduces network traffic and improves communication efficiency.

Item monitoring involves monitoring the values of individual items on the OPC UA server. Clients can specify the items that they want to monitor and the conditions under which they want to be notified. This could be a change in value, status, or a specific event.

Comparison of subscription, item monitoring, and standard value reading

- Efficiency: Subscription and item tracking are more efficient than standard value reading because they reduce the need for continuous queries. Standard scanning requires a client to send regular requests to the server, which can generate unnecessary network traffic. Conversely, with subscription and item monitoring, the server sends updates only when the values of the monitored items change, thereby reducing network traffic and improving communication efficiency.
- Timeliness: Subscription and item monitoring provide more timely updates than standard value reading. The server immediately sends notifications when the values of the monitored items change, allowing customers to react faster to changes.
- Flexibility: Clients can define the items they want to monitor and the conditions under which they receive notifications, so they can focus on the data that matters most to them.

In short, the OPC UA subscription and item monitoring features allow more efficient, timely, and flexible data communication between clients and servers compared to the standard value reading. This reduces network traffic and allows clients to focus on the data that matters most, thereby improving the performance of industrial automation and control systems.

2.2.2 Encryption and Authentication

OPC UA's security model relies on the "Security Policy" to establish encryption and integrity mechanisms that devices utilize for securing data [18]. Several predefined security policies are available in OPC UA, offering various levels of protection. Examples of these policies are as follows:

- None: No security
- Basic128Rsa15: RSA 128-based encryption and RSA-15 signature
- Basic256: RSA/SHA-256-based encryption
- Basic256Sha256: RSA/SHA-256-based encryption and SHA-256 hash
- Aes128-Sha256-RsaOaep: AES-128 encryption, SHA-256 hash and RSA-OAEP

The OPC UA authentication scheme verifies the legitimacy of clients and servers. This mechanism is a fundamental aspect of the security architecture, as it maintains the confidentiality and integrity of the client-server communication.

OPC UA offers several ways for clients to identify themselves to the server:

- Anonymous authentication. This method is rarely used and is recommended only in specific cases as it is less secure.
- Username/password based authentication: OPC UA also supports username and password based authentication, which is commonly used to control user level access. However, this method is less secure than certificate-based authentication, so it is important to use it appropriately and to create strong passwords.
- Certificate-based authentication: OPC UA makes extensive use of X.509 certificates for mutual authentication between client and server. Each party has its certificate issued by a trusted certificate authority (or it can be self-signed). When a client tries to connect to a server, the server requests the client's certificate and checks its validity. The same is true in reverse. If either certificate is invalid or untrusted, the connection is refused.

2.3 In-the-Loop Tests

In-the-loop tests are techniques that help to test and improve a system (for example, a car's electronics) without actually having to carry out risky or expensive experiments. This is achieved by simulating different parts of the system (such as software or hardware) in a computer to test their operation and interactions. In-the-loop testing methods are important in the process of testing and developing systems. The use of these methods ensures that the system functions correctly in actual use and minimizes the risk of potential errors [19, 20].

Model-in-the-loop (MiL) testing is the initial phase in which a mathematical model of the system's operation is created. This phase occurs before the actual hardware or software development and allows for preliminary testing and optimization of the system's operation.

Software-in-the-loop (SiL) and processor-in-the-loop (PiL) testing are used later in the development process. In these phases, the system software is running on a simulated hardware (SiL) or an actual hardware processor (PiL), but the external inputs and hardware environment are simulated. This allows testing of the software operation and interaction with the hardware without the need to use physical hardware, thus reducing costs and risks.

Finally, hardware-in-the-loop (HiL) testing is the most complex, as it combines the actual hardware with a computer model of the system. HiL testing allows for a complete examination of the system, including interactions between the hardware and software.

The HiL test is the best method for the final testing because it can accurately simulate the operation of a real system. Because HiL testing uses real hardware,

it is the most expensive and time-consuming method; however, it is the most validated because it can simulate real system behavior, including hardware/software interactions, as accurately as possible. Therefore, HiL testing is the phase that is most capable of demonstrating the reliability and performance of a system under real-world conditions.

2.4 Intergreen Matrix

In traffic engineering, the so-called intergreen time or intergreen period is fundamentally important. It refers to the time span between the end of the green phase for a certain traffic flow and the beginning of the green phase for the following potentially conflicting traffic flow or permitted turning traffic flow. Intergreen time often includes the yellow and all-red phases of a traffic signal control system. Its primary objective is to clear the intersection of vehicles in the first phase before the vehicles in the next phase are set in motion. This period is critical for avoiding potential conflicts and collisions at intersections [21, 22].

The length of intergreen time is typically determined based on several factors, including the size and layout of the intersection, vehicle speeds, and traffic conditions. By optimizing intergreen time, traffic engineers strive to achieve a balance between safety needs and traffic flow efficiency.

The intergreen matrix is a concept used in traffic signal control to represent the different intergreen times between different phases of traffic signal control. Each cell in the matrix represents the intergreen time between two specific phases, with the rows and columns of the matrix representing the origin and destination phases, respectively. This matrix is a useful tool for traffic engineers to plan and optimize the sequencing of traffic signals to ensure efficient traffic flow while maximizing road user safety. The exact design of the intergreen matrix must be adapted to the specific conditions of each intersection and can therefore vary greatly from intersection to intersection.

Chapter 3

Realization

This section describes the structure of the light control software, its built-in safety features and the implementation of the hardware-in-the-loop test.

3.1 System Architecture

As shown in Figure 3.1, the entire traffic signal control system consists of 2 primary subsystems. The first subsystem, elaborated further in this paper, is the local traffic light control program that runs autonomously on the PLCs and controls signal lights through its outputs without any dependency on the other elements of the system elements. The PLCs communicate with the second subsystem, the central system, via the OPC UA protocol. Through the central system, information about the traffic lights is accessible by multiple external devices. This feature presents an opportunity for developers to integrate different V2X systems using the system's API [3].

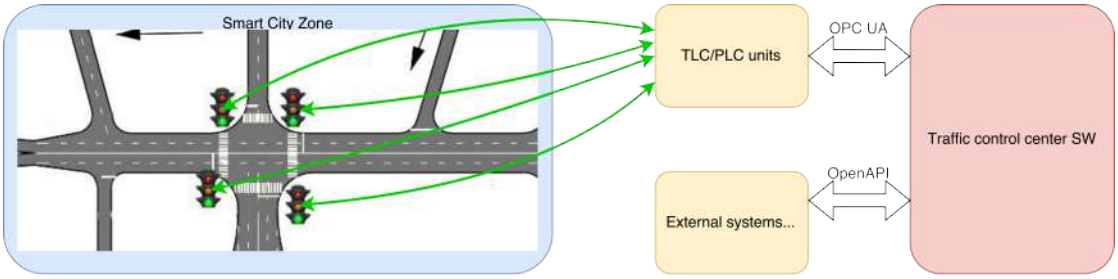

Figure 3.1: System elements

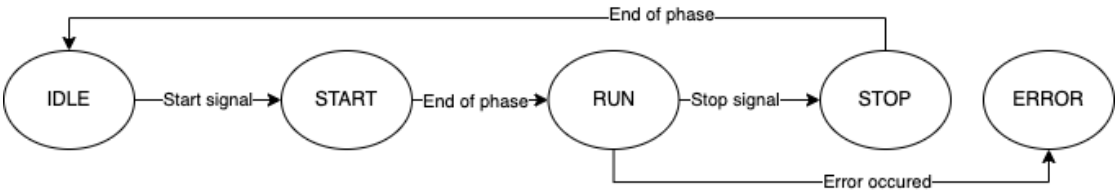

Figure 3.2: State Machine of the Main Program

3.2 PLC Program

The implemented PLC program that controls the traffic lights is overwhelmingly complex. In order to make the mechanism easy to understand for the reader, I will explain the structure with the help of the Figure 3.3.

The **main program** of the PLC is a Program Organization Unit (POU) that is written in Structured Text. The PLC task manager cyclically calls this program every 10 milliseconds. The main purpose of the primary program is to manage the outputs via a state machine, as shown in Figure 3.2.

In the IDLE state of the system, the lights are in a condition known as yellow flashing. The START state, which is only accessible from the IDLE state, initiates the starting phase plan. Upon completion of this phase plan, the system automatically transitions into the RUN state. The RUN state is the most intricate state of the system. During this state, several POUs are called, including the Light Control program, Intergreen Matrix Checker program, and the Connected Cable Checking program.

A notable constraint of the main program is that it can only transition from the RUN state to the ERROR state. This was a deliberate development decision because there are failures that can be resolved by rebooting the system, and the system is capable of rebooting automatically.

The system can only enter the STOP state from the RUN state, and this can only occur when an external stop signal is received, either from the Ethernet interface or from the I/O modules.

The ERROR state is the last state to mention. This state is responsible for alerting the system when an operational error occurs, for instance, a cable break or an intergreen violation. From the ERROR state, the system will automatically attempt to exit and reboot itself a predefined number of times. If rebooting does not resolve the issue, manual intervention is necessary.

The **light control program** POU is also written in structured text and is called directly in the RUN state of the main program. Its task is to control all the lights present at the intersection by calling the traffic light function block (FB).

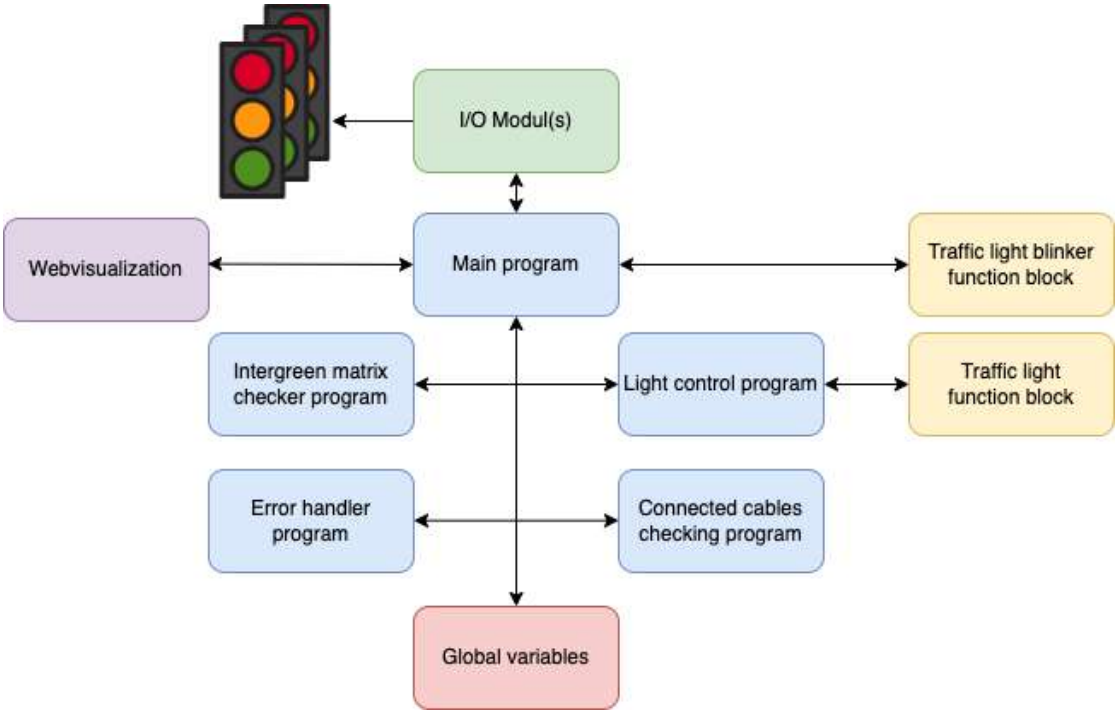

Figure 3.3: PLC Program Structure

Each traffic light head was assigned a separate traffic light FB entity. The **traffic light** FB is a simple structure in the structured text language, which contains a total of four timer elements and iterates through the different phases using the given time values. Similarly, **traffic light blinker** FB is a simple ladder diagram with two timer elements and controls only the amber light output.

The **intergreen matrix checker** and **connected cable checking** programs are responsible for signaling the system when a failure occurs. Both programs were written in structured text, and the outputs of the main program were monitored. The intergreen matrix checker is a redundancy element that improves security. This element works with predefined time constants that cannot be modified by the user, only by the programmer. It measures whether sufficient time has elapsed between green phases. The connected cable checking program detects if anything has happened to the connected cables through the internal feedback of the I/O modules. Both programs indicate global variables in the PLC if a fault has occurred, so that all POUs can detect the fault immediately.

The **error-handling program** is also called periodically, independent of the main program, and checks the global variables to determine if an error has occurred. If an error is detected, it checks the type of error (cable or intergreen matrix) and starts a reboot accordingly or signals the error to the central program (this program is still under development). If the system exceeds the allowed reboot attempt, it switches permanently to the error mode and signals the type and severity of the error to the administration and central programs. This POU was also written in structured text.

3.2.1 Webvisualization

In addition, it is important to note that the PLC has a web visualization interface where the PLC outputs, inputs and error variables can be monitored via a human machine interface (HMI). This interface is easily accessible from any web browser. The implemented HMI is shown in Figure 3.4.

3.2.2 PLC Output Logger

The PLC program contains another important POU, called **Datalogger** POU. This runs completely independently of all the PLC programs. One of its tasks is to synchronize the internal clock of the PLC with the world clock using the Network Time Protocol (NTP). This function is of particular importance because in the event that the PLC's internal clock is shifted when the device is powered off, the clock must be set to the correct time. Another function of the POU is to continuously log changes in PLC outputs to a CSV file to facilitate testing and debugging.

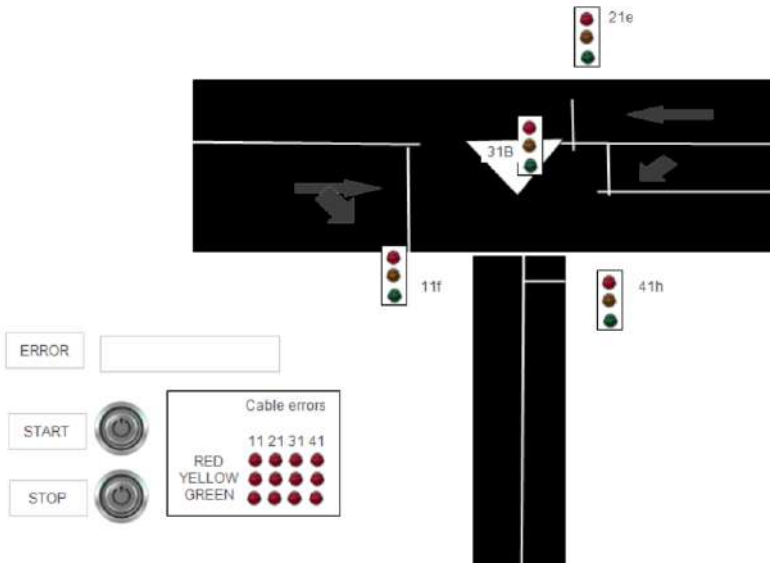

Figure 3.4: Human Machine Interface of The PLC Program

3.3 Hardware-in-the-Loop Test

The most challenging aspect of HiL testing is implementing the test environment. Connecting traffic lights to each PLC output is the simplest solution. However, this option is impractical due to its space requirements and would make the PLC highly immobile. Additionally, simulating a cable break between the PLC and traffic light heads would be difficult with this solution. Therefore, a test was needed that would allow the PLC to remain portable, simulate the cable break, provide a way to test the PLC outputs with instrumentation and keep the cost down.

Given these constraints, I designed a dedicated testing circuit utilizing simple electronic components such as transistors, LEDs, and a microcontroller to accurately measure the output signal level. However, there were two obstacles: simulating a cable break and reducing the voltage of the PLC outputs ($24VDC$) to a level that most microcontrollers could handle ($3.3VDC$).

3.3.1 Measurement Circuit

The main objective of this design task was to simulate a cable break. The simplest solution would be to use a bipolar transistor common emitter configuration, see figure 3.5. The purpose of this connection would be to use the output of the Arduino to saturate the transistor. However, our experiments have shown that

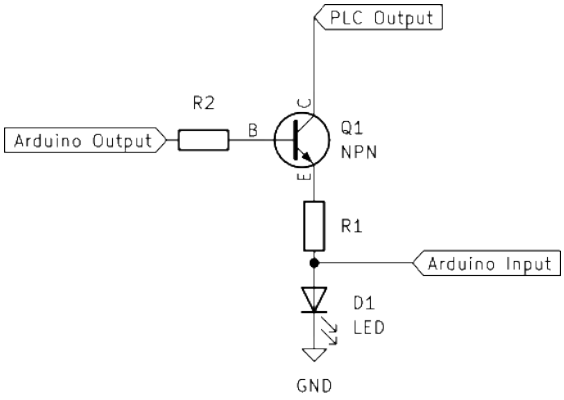

Figure 3.5: Common Emitter Transistor Configuration

this connection is not suitable in this case. The output card of the PLC constantly checks with a negligible current whether a short circuit or cable break has formed on the output. In this configuration, the transistors do not close fully and sufficient current is passed to the load, in this case an LED, which causes the PLC to not detect a cable break when the Arduino's output is deactivated.

For these reasons, a modified Sziklai pair circuit was chosen due to its high current gain and exceptional stability. Sziklai pair is also widely used in different audio equipments thanks to its excellent stability. Its stability is mainly due to the fact that its saturation voltage is about $0.6VDC$, which makes it less sensitive to load impedance. Another advantage of its low base saturation voltage is that it can be effortlessly operated by any microcontroller.

The second problem, related to the input voltage limitation of the microcontroller, can be easily solved. To address this, a basic solution would involve a resistance-based voltage divider and a zener diode.

During the design process, the following aspects should be taken into account and the circuit should be developed accordingly:

- Only use off-the-shelf parts - This is to reduce costs and to make it easy to repair the test panel in case of failure.
- All components must be able to withstand $24VDC$.
- Power dissipation should be kept to a minimum - In order to avoid overheating during testing, which could lead to dangerous situations.
- Use as few components as possible and keep the size of the final circuit as small as possible.
- Provide visual feedback to the tester that allows him to see the current status of the simulated traffic lights, e.g., LEDs.

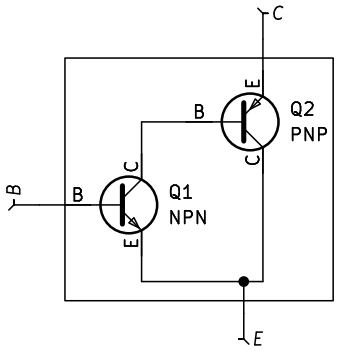

Figure 3.6: Classical Sziklai Pair

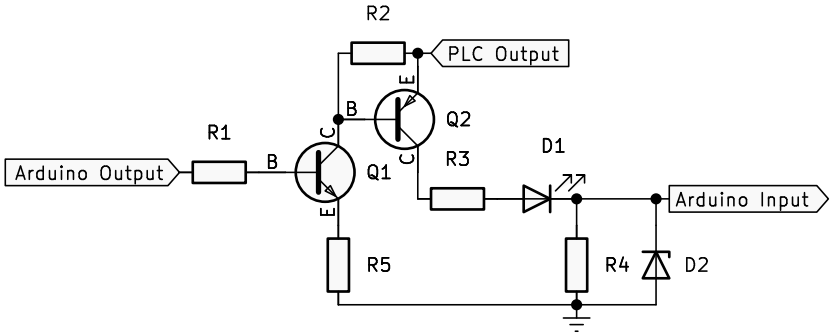

Figure 3.7: Individual Sziklai Pair Based Sub-Circuit

3.3.1.1 Designed Circuit

The designed circuit, shown in Figure 3.8, consists of several subcircuits, shown in Figure 3.7. Each output has a separate subcircuit, but these subcircuits are identical in all respects. Unlike the traditional Sziklai pair circuit, displayed in Figure 3.6, the circuit I have designed differs in several ways. In the classical Sziklai pair circuit, the collector of $\boldsymbol{Q_2}$ is connected to the emitter of the transistor $\boldsymbol{Q_1}$. In this scenario, the collector voltage of transistor $\boldsymbol{Q_2}$ is equivalent to the base voltage of transistor $\boldsymbol{Q_1}$ minus the opening voltage of transistor $\boldsymbol{Q_1}$. If this circuit had been implemented, the collector voltage of transistor $\boldsymbol{Q_2}$ would have been below $5VDC$. It is crucial to note that we intend to attach an LED to the output. An average LED requires approximately $20mADC$ of current to light up, so there is a limit to how much resistance we can put into the circuit as a maximum load.

$$R_{max} = \frac{U_{Q_2,C}}{I_{LED}} = \frac{5V}{20mA} = 250\Omega$$

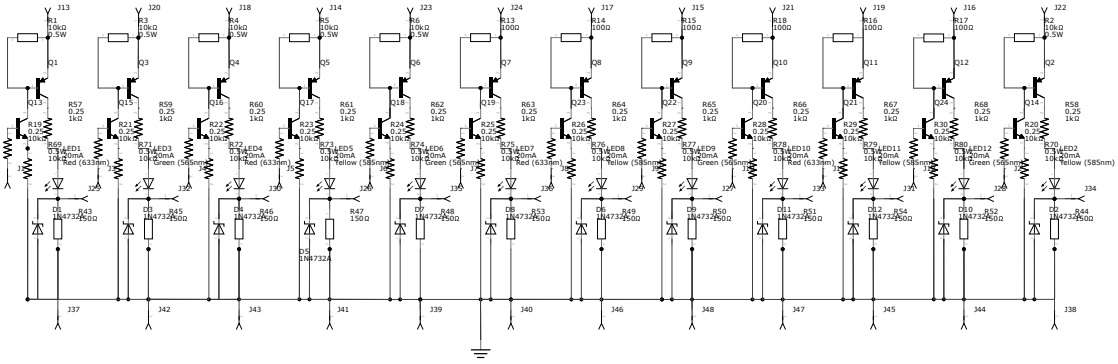

Figure 3.8: Designed Measurement Circuit

The resistance value is significantly small considering the low-cost circuit elements employed in the implementation of the circuit. Hence, the nominal resistance value of the resistors can deviate easily from their actual value. Consequently, a 10Ω alteration in resistance will lead to an approximately $0.8m$ ADC variation. It's possible to experience a 4% resistance value variation in reality since the resistors utilized possess a tolerance of 5%. It is also important to note that we had a larger inventory of higher value resistors available.

The initial stage of our design process was to determine the appropriate transistor types, as their parameters impact all other circuit elements, which must be fine-tuned accordingly. We chose the P2N2222A NPN transistor model, which offers an extensive voltage range [23]. Please refer to Table 3.1 for further details. The only issue regarding this transistor is the emitter-base voltage, as the other voltage limits are higher than the maximum voltage present in the circuit. Upon further examination, it becomes clear that this transistor will be replacing the $\boldsymbol{Q_1}$ transistor, meaning that only a maximum of $5VDC$ will be applied to its base, which does not exceed the maximum emitter-base voltage levels.

For the PNP transistor $\boldsymbol{Q_2}$, the 2N3906 model was selected [24]. The properties of this model are displayed in Table 3.2. It can be seen that both the collector-emitter and the collector-base voltages of this transistor exceed the maximum limit voltage level of $24VDC$, but the emitter-base voltage is lower than this. Since the base voltage is set internally in the circuit and not by an external voltage generator, it will not pose a problem.

Since the emitter of transistor $\boldsymbol{Q_1}$ is connected to the ground, a resistor such as $\boldsymbol{R_5}$ in Figure 3.7 is useful for limiting the emitter current. To minimize unnecessary power consumption, selecting a high-value resistor is recommended to minimize the emitter current of $\boldsymbol{Q_1}$.

Characteristic	Symbol	Value
Collector-Emitter Voltage	V_{CE}	$40VDC$
Collector-Base Voltage	V_{CB}	$75VDC$
Emitter-Base Voltage	V_{EB}	$6VDC$
Collector Current (Continuous)	I_C	$600mADC$

Table 3.1: P2N2222A Transistor Maximum Ratings

Characteristic	Symbol	Value
Collector-Emitter Voltage	V_{CE}	$40VDC$
Collector-Base Voltage	V_{CB}	$40VDC$
Emitter-Base Voltage	V_{EB}	$5VDC$
Collector Current (Continuous)	I_C	$200mADC$

Table 3.2: 2N3906 Transistor Maximum Ratings

$$I_{Q_1,E} = \frac{U_{Q_1,E}}{R_5} \simeq \frac{U_{ArduinoOutput} - U_{Q_1,BE}}{R_5} \simeq \frac{5V - 0.7V}{10k\Omega} = 430\mu A$$

The current gain (β) of the PNP transistor we use is approximately 150[24]. Using this information, the current of the collector and base currents can be calculated.

$$I_{Q_1,B} = \frac{I_{Q_1,E}}{\beta_{Q_1} + 1} \simeq \frac{430\mu A}{151} = 2.85\mu A$$

$$I_{Q_1,C} = \frac{\beta_{Q_1}}{\beta_{Q_1} + 1} \times I_{Q_1,E} = \frac{150}{151} \times 430\mu A \simeq 427.15\mu A$$

In this scenario, it is evident that the base current's value is negligible, so the value of the resistor $\boldsymbol{R_1}$ is almost insignificant. A $10k\Omega$ resistor was used in our case.

The primary function of the $\boldsymbol{R_2}$ resistor is to stabilize the circuit. If transistor $\boldsymbol{Q_1}$ is in the cutoff state, then $\boldsymbol{R_2}$ functions as a pull-up resistance and lifts the base of transistor $\boldsymbol{Q_2}$ to the output voltage of the PLC. Typically, pull-up resistors are expected to have a high resistance. Therefore, a $10k\Omega$ resistor is used.

Base current $\boldsymbol{I_{Q_2,B}}$ can be calculated by using current $\boldsymbol{I_{Q_1,C}}$ and resistance of $\boldsymbol{R_2}$. From the data sheet of 2N3906 PNP transistor, it is inferred that the saturation voltage $\boldsymbol{U_{Q_2,EB}}$ is roughly $0.8VDC$ [24]. We can formulate the following equation using Kirchoff's current law:

$$I_{Q_2,B} = I_{Q_1,C} - I_{R_2} = I_{Q_1,C} - \frac{U_{Q_2,EB}}{R_2} = 427.15\mu A - \frac{0.8V}{10k\Omega} = 347.15\mu A$$

Finally, it is necessary to calculate only the current $\boldsymbol{I_{Q_2,E}}$ in order to determine how much current the system requires from the PLC. To determine $\boldsymbol{I_{Q_{Q_2,E}}}$, the value of $\boldsymbol{I_{Q_{Q_2,C}}}$ current is necessary. To determine $\boldsymbol{I_{Q_2,C}}$, we can take into account the collector-emitter saturation voltage of the PNP transistor, which is about $200mVDC$[24].

$$I_{Q_2,C} = \frac{U_{PLCOutput} - U_{Q_2,CE} - U_{LED}}{R_3 + R_4}$$

It was mentioned earlier that for LEDs it is advisable to limit the current to $20mADC$. Another consideration to keep in mind when designing the circuit is that the Arduino input should be set to a maximum voltage of $3.3VDC$. With this information, the maximum value of $\boldsymbol{R_4}$ can be calculated:

$$R_{4,Max} = \frac{U_{ArcuinoInputMax}}{I_{Q_2,C,Max}} = \frac{3.3V}{20mA} = 165\Omega$$

Taking into account the calculations and available circuit elements, a resistor of 150Ω was chosen to replace the $\boldsymbol{R_4}$ resistor. The next step was to calculate $\boldsymbol{R_3}$:

$$R_3 = \frac{U_{PLCOutput} - U_{Q_2,CE} - U_{LED}}{I_{Q_2,CMax}} - R_4 = \frac{24V - 0.2V - 2V}{20mA} - 150\Omega = 940\Omega$$

The value of the $\boldsymbol{R_3}$ resistor has been set to $1k\Omega$ due to the limited resistance options. Lastly, the value of $\boldsymbol{I_{Q_2,C}}$, $\boldsymbol{I_{Q_2,E}}$ and the PLC output current can be calculated:

$$I_{Q_2,C} = \frac{U_{PLCOutput} - U_{Q_2,CE} - U_{LED}}{R_3 + R_4} = \frac{24V - 0.2V - 2V}{150\Omega + 1000\Omega} = 18.96mA$$

$$I_{Q_2,E} = I_{Q_2,B} + I_{Q_2,C} = 0.35mA + 18.96mA = 19.31mA$$

$$I_{PLCOutput} = I_{Q_2,E} + I_{R_2} = I_{Q_2,E} + \frac{U_{EB}}{R_2} = 19.31mA + \frac{0.8V}{10k\Omega} = 19.39mA$$

Based on the performed calculations, the total current consumption of the circuit does not exceed $20mADC$, which is a very low current consumption and guarantees the safe operation of the $2W$ resistors we use. Moreover, an additional Zener diode is inserted in the circuit to ensure that the measurement microcontroller will not be damaged.

3.3.1.2 Simulating the Circuit Design

After successfully designing the circuit, it was crucial to validate the design before conducting real-life tests. Consequently, the circuit design underwent a simulation, with the results presented in Figure 3.9. It is evident from the results that

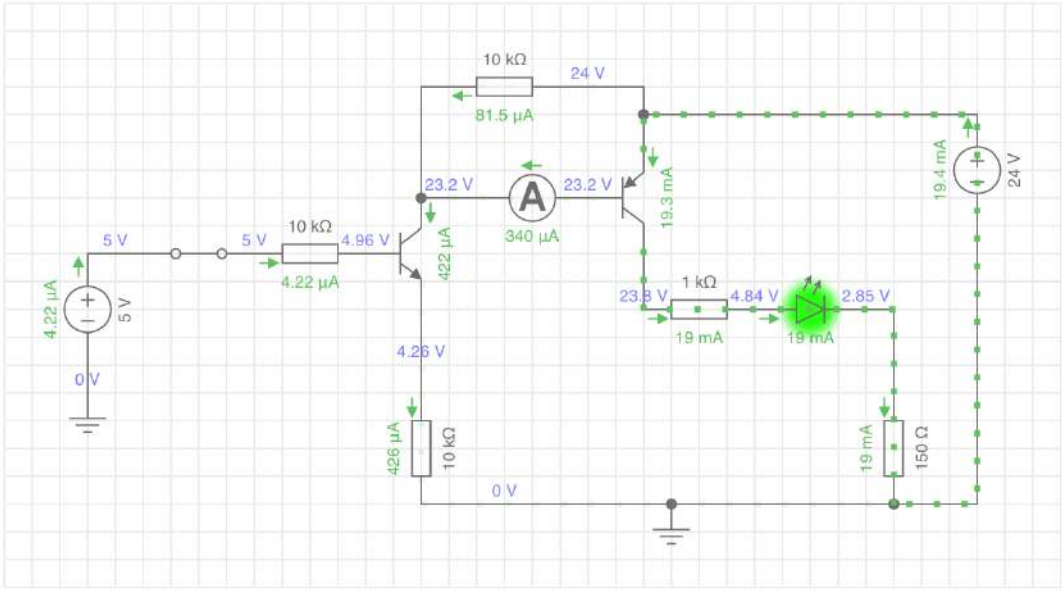

Figure 3.9: Circuit Simulation: Transistors in Saturated State

the voltage and current values match the earlier computations in Section 3.3.1.1. Therefore, the designed circuit is ready for assembly. It is noteworthy that the voltage level at the Arduino input satisfies the transistor-transistor logic (TTL) level condition. This condition indicates that a transistor-based digital input is considered true when the voltage value reaches a minimum of $2VDC$. According to the simulation, it reaches $2.85VDC$ in our case.

3.3.2 Measurement

To conduct the measurement, we utilized multiple devices to simulate the real-life system. The measurement setup is shown in Figure 3.10. The following models were utilized during the testing:

- PLC: PFC200, 2nd gen, Ex. (450-8211/040-000)
- I/O cards: 2x 8-channel digital output card, 24VDC, 0.5A with diagnostics, Ex. (750-537/040-000)
- Microcontroler: 2x Arduino Nano
- Measurement control device: Raspberry Pi 4

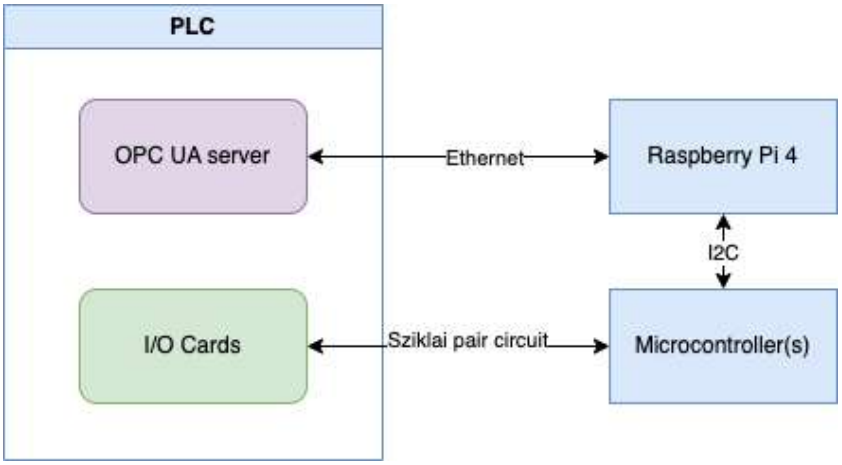

Figure 3.10: Devices and Modules in the HIL Test

The entire measurement was controlled using a Raspberry Pi 4 (RPi4) single-board computer. There are two versions of the measurement program run by RPi4: one written in Rust and one written in C. The reason is that the central system is written in Rust and this allows not only to test the correct operation of the PLC programs, but also to test the OPC UA implementation of the central system, which uses the C library open62541 [17]. For these reasons, the C version of the measuring program does not include all OPC UA functions, see Table 3.3.

	Read	Write	Subscription	Encryption	authentication
`C`	✓	✓	✓	✗	✗
`Rust`	✓	✓	✓	✓	✓

Table 3.3: Tested Functions in Open62541 Library

The measurements evaluated both the correct operation of the PLC programs and the capabilities of the OPC UA protocol. Additionally, we checked whether the PLC processor was able to execute its embedded programs and the OPC UA communication tasks in parallel.

3.3.2.1 Network Wide Testing

The purpose of this test was to measure the latency between two system endpoints - the PLC and the RPi4. Latency was defined as the time lapse between the signal change in the PLC and its detection by the RPi4 through the OPC UA protocol. All devices must retrieve the current time from the same NTP server to ensure the accurate evaluation of the measurement log files.

Furthermore, we analyzed the amount of data transmitted between network nodes as we aimed to determine the optimal network data traffic capacity. To quantify the data traffic, we utilized Wireshark, which is a software tool that captures and sorts network traffic for further examination [25].

3.3.2.2 CPU Load Testing

If the PLC's processor is overloaded, it can cause serious system failures. This occurs because the PLC task manager, which oversees system processes, lacks allocated resources. In an unoptimized program, the developer may overlook the added burden of communication protocols on the processor while writing the program, which can lead to potentially dangerous outcomes. Firstly, during high CPU loads, system timings may drift due to the task manager's inability to keep up with the pre-defined cycle times. A simple solution is to leave a computing buffer for the CPU with at least 20% extra capacity. Continuous monitoring of the CPU load by means of HiL tests during the execution of PLC programs and OPC UA communication is recommended. Optimal results were achieved if the CPU load remained below 80%.

3.3.2.3 Test Cases

It was previously noted in Section 3.3.2 and in Table 3.3 that multiple versions of the control software for the HiL test are operating on RPi4. This section aims to provide a more detailed explanation of these versions to highlight the differences between them.

The initial version of the HiL test control software utilized the C programming language. One reason for this decision was that the open612541 library, on which the software was based, was also written in C. Additionally, as the RPi4 interacts with the microcontrollers via I2C communication [26], creating the software in C minimized potential compatibility issues.

However, as development progressed, it was discovered that the I2C library was poorly optimized, which resulted in an unreliable connection. Thus, the test controller program was rewritten, and our own implementation of the open62541 C library was tested using Rust. The initial version of the Rust program did not include encryption and authentication functions, as the primary objective was to test the basic OPC UA functions described in Section 2.2.1.

Surprisingly, the Rust implementation of I2C was significantly better quality than the C implementation, so we decided to test all the OPC UA features we wanted to implement in the system in Rust and not in C.

Data collection was done consistently in all three test cases. The PLC recorded

	Average Network Latency	Max Network Latency	Average Network Load
C	$115.3311ms$	$171.00ms$	$857bytes/s$
Rust	$193.1405ms$	$301.00ms$	$768bytes/s$
Rust with Encryption and Authentication	$276.2109ms$	$362.00ms$	$954bytes/s$

Table 3.4: Network Test Results, 1-Hour Long Measurements

all output changes in real-time in an internal log file [1]. In addition, a Wireshark capture process has been started on the RPi4 to monitor the incoming and outgoing traffic. Finally, the I2C communication with the microcontrollers and the OPC UA communication with the PLC are started in parallel, whose results are also saved to log files for later analysis.

The I2C is used to continuously check the signals on the PLC outputs and to trigger a simulated cable break, but this function has not been used not yet. Each microcontroller runs a simple, custom-made program whose overall task is to manage the I2C communication via interrupts and execute the instructions contained in I2C messages. The detailed program codes for the microcontrollers can be found in Appendix A, B.

3.3.3 Results

All test cases were successfully executed, and the results are presented in Tables 3.4 and 3.5. The results show that the lowest network latency was obtained by using the control program written in C. This finding is not unexpected, as Rust's performance, although nearly equivalent to that of C, still lags behind [27]. It was expected that encryption would increase the computational demand since the program must run the encryption algorithms. The Rust program's performance is also impacted by its reliance on an internal C linker to use the open62541 library instead of utilizing pre-existing C libraries. To enhance the runtime of the Rust program, employing a specialized OPC UA library written in or optimized for Rust is recommended.

Unfortunately, these results indicate that this implementation requires further optimization. The latency can be minimized by utilizing appropriate networking hardware, including optical networking, industrial-grade networking equipment, and high-quality network chipsets on the measurement computers.

[1] Note: This timestamp may be different from the timestamp when the signal change actually appeared on the output. This is because the PLC changes the outputs after the entire program cycle has run.

	Max CPU Load	Average CPU Load
C	30 %	20.09 %
Rust	29 %	20.01 %
Rust with Encryption and Authentication	29 %	20.36 %

Table 3.5: CPU Test Results, 1-Hour Long Measurements

The average traffic load of the network shows that the OPC UA protocol does not transmit and receive a large number of network packets. This aligns with our expectations, as OPC UA is primarily intended for industrial environments, where communication occurs among hundreds of devices rather than just two, as in our particular scenario. These traffic results indicate that the network's primary design focus should prioritize response times over high bandwidth.

The measured results indicate that the PLC processor has significant computational capacity reserves. During all three tests, the processor experienced an average load of about 20%, and the maximum load never exceeded 30%. The measured results indicate that the PLC processor has significant computational capacity reserves. Therefore, the target computational capacity reserves have been achieved.

During the testing, we analyzed the performance of the PLC programs presented in Section 3.2 but mostly manually and not in an automated way. Initial results indicated that all programs performed satisfactorily, although further examination of the HiL test findings revealed some areas for improvement. Here, I observed that each minute, the traffic light phases were extended by an additional cycle time. At first, I was puzzled by this, but after running additional tests and disabling several of the PLC programs, I was able to determine that the problem was caused by the "light function block". Originally, this block was written in ladder diagram language, but this language was ill-equipped to handle state machines with time-bound state change conditions, as our results showed. As a result, I rewrote the block in a structured text programming language, and the error was eliminated. Although a cycle time error may appear insignificant, when you consider that it can cause a delay of up to 14 seconds in a day, the seriousness of the error becomes more apparent.

However, upon later examination of the test results, an additional error was noticed. In this case, the phase plans were slipping by one cycle time every ten minutes. The error's source was not the PLC programs but the operating system of the PLC. Today's PLCs use an embedded Linux operating system and emulate various runtime systems. This simplifies the development process by using one operating system for different types of PLCs. A bug has recently been detected,

with the NTP updater feature of the operating system identified as the root cause. Specifically, the NTP updater interferes with the execution of PLC programs during every ten-minute cycle. To solve the problem, I disabled NTP during PLC program execution and created a PLC script that synchronizes the internal clock with the NTP server every time the PLC is restarted after a shutdown.

Chapter 4

Conclusions and Future Work

This thesis presents a modern, flexible, and reliable traffic control system ideal for use in various automotive proving grounds, especially where self-driving features are being tested and traditional traffic light control systems cannot meet the requirements.

One of the system's primary goals was to have as many safety features as possible in the system, but these were implemented in such a way that there would be no restrictions on the introduction of new features. To evaluate the system, a hardware-in-the-loop test was performed using an innovative Sziklai pair based measurement circuit to maintain system portability and allow advanced fault simulation.

Unfortunately, the test results were mixed. The PLC program performed as expected, but the OPC UA data reading functions underperformed, resulting in higher than desired latency. Future efforts should aim to optimize this aspect by improving communication protocols. Nonetheless, thanks to the careful testing process, the system in its current form is already fully capable of controlling various traffic light intersections.

Furthermore, its implementation in larger systems is hassle-free due to the use of widely available and easy-to-understand technologies. The system's communication protocols are perfectly compatible with different V2X technologies, making it simple to implement various V2X communication protocols into the system.

When a central control system for the system is fully developed in the future, and the interactive mode is successfully implemented with sufficiently low latency, this system will be a highly functional tool for automotive test facilities.

Appendix A

Arduino Input Measurement Code

```
#include <Wire.h>

#define SLAVE_ADDRESS 0x40

// Digital pins to be read (including A0)
const byte digitalPins[] = { 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, A0 };
const byte numDigitalPins = sizeof(digitalPins) / sizeof(digitalPins[0]);

// Function to handle I2C requests
void receiveEvent(int howMany);

void setup() {
  // Configure digital pins as input
  for (byte i = 0; i < numDigitalPins; i++) {
    pinMode(digitalPins[i], INPUT);
  }

  // Initialize I2C as slave
  Wire.begin(SLAVE_ADDRESS);
  Wire.onRequest(receiveEvent);
  Serial.begin(9600);
}

void loop() {
  // Main loop is empty as the device only responds to I2C requests
  for (byte i = 0; i < numDigitalPins; i++) {
    Serial.print(digitalRead(digitalPins[i]));
```

```
  }
  Serial.println("!");
  delay(50);
}

void receiveEvent(int howMany) {
  byte pinStatus[numDigitalPins];

  // Read digital pin states
  for (byte i = 0; i < numDigitalPins; i++) {
    pinStatus[i] = digitalRead(digitalPins[i]) ;
  }

  // Send digital pin states via I2C
  Wire.write(pinStatus, numDigitalPins);
}
```

Appendix B

Arduino Cable Break Simulation Control Code

```
#include <Wire.h>

#define SLAVE_ADDRESS 0x20

// Digital pins to be controlled (including A0)
const byte digitalPins[] = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
    12, A0};
const byte numDigitalPins = sizeof(digitalPins) / sizeof(
    digitalPins[0]);

// Function to handle I2C requests
void receiveEvent(int howMany);

void setup() {
  // Configure digital pins as output
  for (byte i = 0; i < numDigitalPins; i++) {
    pinMode(digitalPins[i], OUTPUT);
  }

  // Initialize I2C as slave
  Wire.begin(SLAVE_ADDRESS);
  Wire.onReceive(receiveEvent);

  //Set up serial monitor
  Serial.begin(9600);

}

void loop() {
  // Main loop is empty as the device only responds to I2C
```

```
   requests
}

void receiveEvent(int howMany) {
  //print received data to serial monitor
  Serial.println("Received data:");
  Serial.println(howMany);


  if (howMany != numDigitalPins) {
    return; // Ensure that the received data matches the expected length
  }

  byte pinValues[numDigitalPins];

  for (byte i = 0; i < numDigitalPins; i++) {
    pinValues[i] = Wire.read();

  }

  // Set digital pin states based on received values
  for (byte i = 0; i < numDigitalPins; i++) {
    digitalWrite(digitalPins[i], pinValues[i] == 0 ? LOW : HIGH);
  }
}
```

Bibliography

[1] Z. Szalay, T. Tettamanti, D. Esztergár-Kiss, I. Varga, and C. Bartolini, "Development of a Test Track for Driverless Cars: Vehicle Design, Track Configuration, and Liability Considerations," *Periodica Polytechnica Transportation Engineering*, vol. 46, p. 29, Mar. 2017. 2

[2] H. Winner, S. Hakuli, F. Lotz, and C. Singer, eds., *Handbook of Driver Assistance Systems*. Cham: Springer International Publishing, 2016. 2

[3] T. Wágner, T. Ormándi, T. Tettamanti, and I. Varga, "SPaT/MAP V2X communication between traffic light and vehicles and a realization with digital twin," *Computers and Electrical Engineering*, vol. 106, p. 108560, Mar. 2023. 2, 13

[4] O.-R. A. D. O. Committee, *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. SAE, Apr. 2021. 2

[5] V. Martins, J. Rufino, L. Silva, J. Almeida, B. Miguel Fernandes Silva, J. Ferreira, and J. Fonseca, "Towards Personal Virtual Traffic Lights," *Information*, vol. 10, p. 32, Jan. 2019. 2

[6] S. S. R and L. Rajendran, "Real-time adaptive traffic control system for smart cities," in *2021 International Conference on Computer Communication and Informatics (ICCCI)*, IEEE, jan 2021. 2

[7] J. D. Trivedi, M. S. Devi, and D. H. Dave, "A vision-based real-time adaptive traffic light control system using vehicular density value and statistical block matching approach," *Transport and Telecommunication Journal*, vol. 22, no. 1, pp. 87–97, 2021. 2

[8] "Traffic Management Centers - Texas A&M University." https://mobility.tamu.edu/mip/strategies-pdfs/traffic-management/technical-summary/traffic-management-centers-4-pg.pdf. 2

[9] CEN, "Traffic signal controllers. Functional safety requirement. EN 12675:2018," 2018. 2

[10] G. Kovács, "Ipari irányítástechnika, PLC-k," 2020. BMEVIIIAC03. 4, 5, 6, 7, 8

[11] B. G. Lipták, ed., *Instrument engineers' handbook*. Boca Raton, FL: CRC Press, 4th ed ed., 2003. 4, 5, 6, 7, 8

[12] P. Kundmueller, "Backplane Power Protection in PLC Systems," Tech. Rep. SLVAEC4, Texas Instruments Incorporated, Texas, USA, July 2019. 4

[13] V. Yang, S. Mu, and D. Hartmann, "Plc dcs analog input module design breaks barriers in channel-to-channel isolation and high density," *Analog Dialogue 50-12*, Dec 2016. 5

[14] I. E. Commission, "Programmable controllers - part 3: Programming languages," Standard IEC 61131-3:2013, International Electrotechnical Commission, Geneva, CH, 2016. 6

[15] P. Chevtsov, S. Higgins, S. Schaffner, and D. Seidman, "PLC Support Software at Jefferson Lab," tech. rep., Thomas Jefferson National Accelerator Facility (TJNAF), Newport News, VA (United States), Oct. 2002. 7

[16] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC unified architecture*. Berlin: Springer, 2009. OCLC: ocn268784080. 9

[17] open62541 authors, "open62541: Open Source Implementation of OPC UA (IEC 62541)." `https://github.com/open62541/open62541`, 2023. 9, 24

[18] Paul Hunkar, "OPC 10000-2 UA Part 2: Security," Nov. 2022. 10

[19] Balázs Scherer, "Software Verification and Validation - BMEVIMIMA11 Design and integration of embedded systems," Oct. 2022. 11

[20] Z. Szalay, "Next Generation X-in-the-Loop Validation Methodology for Automated Vehicle Systems," *IEEE Access*, vol. 9, pp. 35616–35632, 2021. 11

[21] "ÚTÜGYI MÚSZAKI ELÖIRÁS, e-UT 04.01.21, Közúti forgalomirányító berendezések," July 2017. 12

[22] "Guidelines for Traffic Signals (RiLSA), Traffic Lights for Road Traffic," 2015. 12

[23] On Semiconductor, *P2N2222A NPN Silicon Amplifier Transistors*, 7 2013. Rev. 7. 20

[24] On Semiconductor, *2N3906 PNP Silicon Amplifier Transistors*, 2 2010. Rev. 4. 20, 21, 22

[25] J. Beale, A. Orebaugh, and G. Ramirez, *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006. 25

[26] J. Mankar, C. Darode, K. Trivedi, M. Kanoje, and P. Shahare, "Review of I2C protocol," *International Journal of Research in Advent Technology*, vol. 2, no. 1, 2014. 25

[27] W. Bugden and A. Alahmar, "Rust: The programming language for safety and performance," 2022. 26