PLC Controlling Program of an Elevator

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor's Thesis

Valkeakoski

Electrical and Automation Engineering

Spring 2020

Thai Nguyen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Electrical and Automation Engineering
Valkeakoski

Author Thai Nguyen **Year** 2020

Subject PLC Controlling Program of an elevator

Supervisor(s) Mika Oinonen

ABSTRACT

This thesis was commissioned by HAMK University of Applied Sciences. The goal of the thesis was to create a PLC controlling program for a five-floor elevator system for future educational purposes at HAMK University of Applied Sciences.

The theory section discussed related background information such as the definition and classification of elevators and PLCs. In addition, a description of a PLC-based controlling system was added to explain the PLC's role in a realistic application along with other components.

The documentation contains a description of the program and guides for building the project, which can be viewed as a teaching material in ladder logic and function block diagram programming. The program controls elevator operations such as car calls, floor calls, open, close buttons as well as safety sensors. An HMI screen was also added in the project for visualization and a better monitoring of the program.

The program was simulated and visualized successfully, therefore it is reliable to be used as a teaching material in future courses. In addition, in the author's opinion, the program can potentially be adapted for testing operations as wells commercial applications in small apartment buildings.

Keywords Elevator, Function Block Diagram, Ladder Logic, PLC.

Pages 63 pages

CONTENTS

1 INTRODUCTION

Programmable logic controllers (PLC) have long been the core element in industrial automation. They provide machines with the possibility to analyse data and cooperate in automated production and service lines. Elevators are prime examples of such systems, since most of their operations are automatically controlled by microprocessors, including PLCs. This thesis will give an insight to the reader on how a PLC can control a single elevator to perform safely and efficiently. It can be also viewed as a tutorial of PLC programming for automation students, with thorough step-by-step instructions and explanations of the program.

2 THEORECTICAL BACKGROUND

2.1 **Elevator**

An elevator or a lift is a means of transportation of people and goods in-between floors of buildings. Common elevator systems are powered by electric motors and counterweight systems cables for drive transaction (Elprocus, n.d).

There are three main types of elevators: traction with a machine room, machine-room-less traction and hydraulic (Archtoolbox, n.d).

2.1.1 Hydraulic elevators

Figure 1 describes the structure of a hydraulic elevator system. The car is attached to a piston at the bottom that pushes it up when the electric motor pumps some hydraulic fluid into the piston. It is moves down by releasing fluid using a valve. Elevators of this type are used in buildings with up to eight floors.

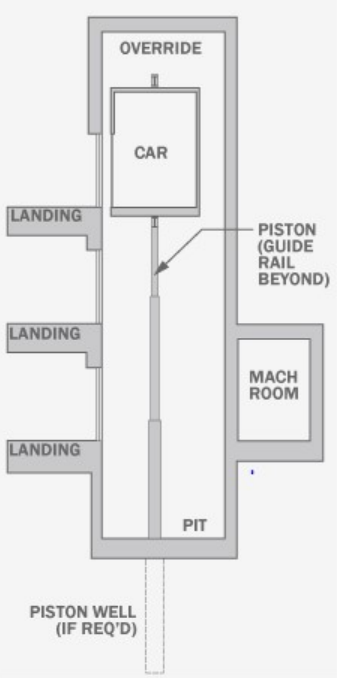

Figure 1. Hydraulic Elevator (Archtoolbox, n.d)

2.1.2 Traction elevators with Machine Room

As shown in Figure 2, traction elevators are moved using electrically driven wheels, ropes and counterweights. They produce higher travel speeds and hydraulic elevators and are used for mid and high-rise buildings.

A variation of this type, called a geared traction elevator is added with gearboxes attached to the wheel's motor. They can reach a maximum travel speed of 153m/s and a maximum distance of about 75 meters.

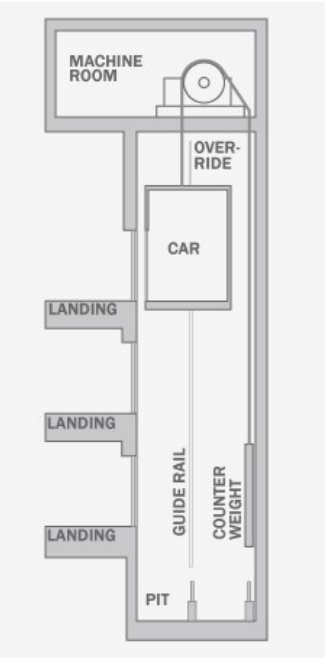

Figure 2. Traction Elevators (Archtoolbox, n.d)

2.1.3 Machine-Room-Less (MRL) elevators

MRL Elevators, as shown in Figure 3, are traction elevators with no machine room above the shaft. Instead, there are control boxes placed in a control room above the highest landing floor. Although they produce the same travel speed as geared traction elevators, MRL elevators are the most reliable for mid-rise building thanks to their space and energy efficiency.

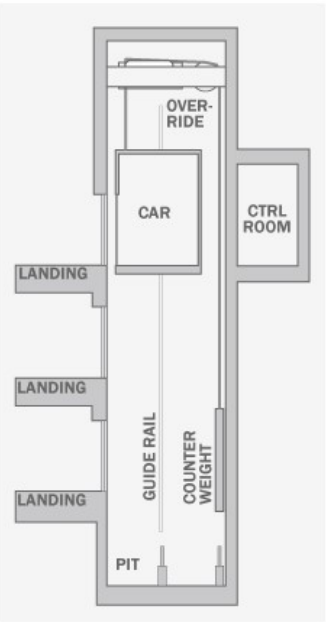

Figure 3. MRL Elevator (Archtoolbox, n.d)

2.2 **Programmable Logic Controller (PLC)**

A PLC is an industrial computer designed for data reading and controlling field devices in an automatic process. PLCs varies in sizes by the amount of inputs and outputs (I/O). A small device can have several I/O cards, while large and networked PLCs can have extended racks that contain hundreds of I/O cards. Since often put in harsh industrial environment, PLCs have robust designs that help them withstand extreme conditions such as cold, heat, dust and moisture.

2.2.1 Hardware components

Figure 4 describes the typical hardware configuration of a PLC system that includes a rack, a power supply unit and I/O modules and a CPU.

Figure 4. PLC installation (Muthukrishnan, n.d)

Rack or Chassis
A rack or chassis is a base used for mounting the power supply, the CPU and I/O modules. It has a backplane at the rear that enables I/O to communicate with the CPU.

Power Supply module
This module distributes power to all PLC components. It converts AC from the power source to DC that powers the CPU and I/O modules. PLCs usually require 24V DC power supplies. Other variations are supplied with isolated power sources.

CPU
The most important part of the PLC is the Central Processing Unit (CPU). A PLC's CPU is an either a 16 or 32 bit microprocessor that consists of memory chip and integrated circuits for monitoring, logic controlling and communicating (Gonzalez, 2015). The CPU can be understood as the

master device and the actuator is the slave. It receives control algorithms downloaded from a programming software, which then command the PLC to carry out instructions like actuating devices, analyzing field data. The CPU also makes routine memory checkups of the PLC to report back any errors and ensure the memory is undamaged (Gonzalez, 2015). Like any other common CPUs, PLC CPUs also have ROM (read-only memory) and RAM (random access memory). ROM stores data permanently into the operating system whereas RAM stores status data from field devices and other values like timers, counters, etc.

Input and output modules (I/Os)
There are two types of I/Os, digital and analog.

- Digital inputs (DIs) monitor a voltage over a specific threshold (LabJack, n.d). If it exceeds the threshold, the DI is set by the CPU to the value TRUE, 1 or HIGH. The value is set to 0, FALSE or LOW on the other hand. A digital output (DO) allows the CPU to control a voltage. If an output is set to TRUE, 1 or HIGH, it will produce voltage of about 5 or 3.3 V. If the value is FALSE, 0 or LOW, the output is connected to ground and therefore produce no voltage.
- Analog input (AI) devices are sensors that measure environment conditions like temperature, pressure and convert them into voltages. These values are then converted to digital values which are processable by the CPU.
- Difference between digital and analog I/O: Digital IOs works on 2 states. This can be ON/OFF or TRUE/FALSE for input devices or START/STOP mode for output devices. On the contrary, AIs have multiple values. They can be used to control DOs, for example opening or closing a valve based on water level (DOF, 2015).
- An input device provides field data to the PLC in the monitoring process. These devices can be pushbuttons, sensors (temperature, motion, light). Their signal types vary from 4-20 mA, 24VDC, to 110 VDC (LabJack, n.d). A PLC can be enabled to accept all these signal types by installing input cards for each type of signal. Output devices are controlled by commands coming from the PLC. Controllable outputs for the PLC can be light fixtures, motors, buttons, conveyers, etc.

2.2.2 Types of PLC

- Integrated or Compact PLC: Figure 5 shows a sample design of a compact PLC. These devices consist of several predetermined modules within a single case. Therefore, their I/O capabilities are limited.

Figure 5. Integrated or Compact PLC (Elprocus, n.d)

- Modular PLC: Figure 6 shows a typical design of a modular PLC with a power supply module connected to the left side of the CPU and I/O modules connected to the righthand side. PLCs of this type allows for multiple expansions and thus are more common in industrial uses.

Figure 6. Modular PLC (RS Automation & Controls, n.d)

2.2.3 Operation sequence

PLCs repeatedly carry out three basic steps in their functions. Prior to that, the logic is checked by the programming software to detect any errors. Figure 7 describes the order of the sequence

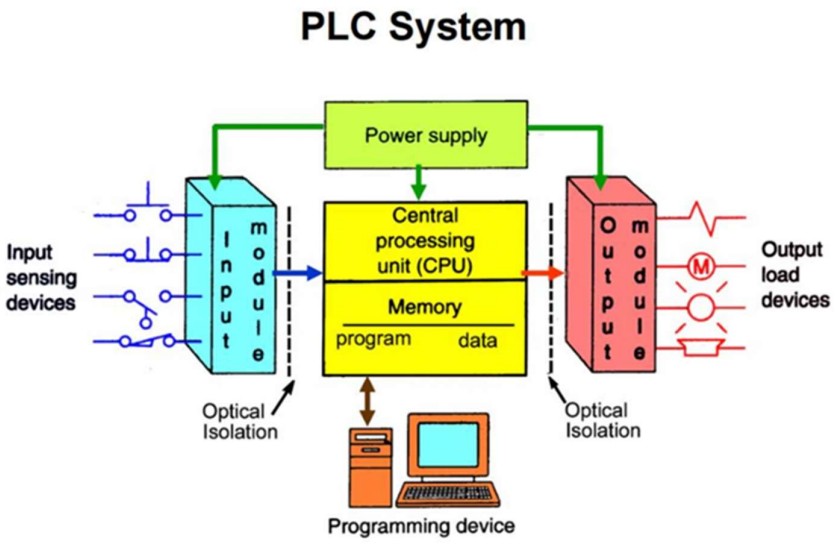

Figure 7. Operation Sequence (Gonzalez, p.2)

- Input scan: Read input values in I/O cards and copy them to the memory.
- Logic scan: Reads the input data and correspondingly execute the program. In this step, the CPU regularly check on input status and updates the output values.
- Output scan: Copy output values from the memory to the I/O cards that control output devices.

2.2.4 Communications

Common communication protocols for PLCs include 9-pin RS-232, EIA-485 or Ethernet (Gonzalez, 2015). Others are Modbus, BACnet, DF1 and fieldbuses such as DeviceNet and Profibus. Modern PLCs communicate over network connections with another system, such as SCADA (Supervisory Control and Data Acquisition) system or web browser. In systems where the I/O number is large, PLCs can cooperate through a P2P (peer-to-peer) communication link. This enables individual controllers in a complex system to exchange data between each other.

2.2.5 Programming Languages

Logic programs are designed or written in a computer software and downloaded to PLCs through a network or a connection cable. There are five PLC programming languages defined in the IEC (International Electrotechnical Commission) 6113-3 standard: functional block diagram (FBD), ladder diagram (LD), structured text (ST), instruction (IL) and sequential functional chart (SFC) (Gonzalez, 2015). Their names imply the logical organization of operation.

2.3 **PLC-based elevator controlling system**

A PLC controlled elevator requires a control room for placing a cabinet which is connected to the motor. Therefore, these can be classified as MRL elevators.

Figure 8 demonstrates the components inside the cabinet that controls the elevator's operation.

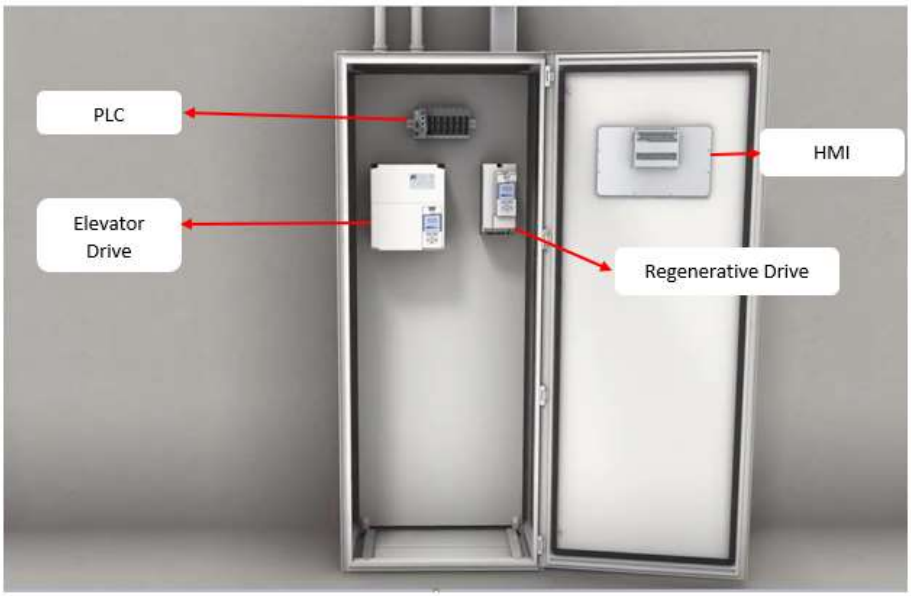

Figure 8. Controlling cabinet (KEB Automation, n.d)

The PLC is in charge of reading inputs such as push buttons, sensor signals and make logic commands, which are sent to the elevator drive. Logic programs can be downloaded into the PLC through a remote PLC gateway or by plugging a USB device that contains the program.

The elevator drive directly controls the elevator motor that lifts or lower the car.

The regenerative drive is a replacement for traditional braking resistors that dissipates excess kinetic energy. Particularly, when lifting a fully loaded car, the elevator drives transfer electrical power to the motor. When the car descends, energy stored in the mechanical system is converted back into electrical energy (KEB Automation, n.d). As a result, the braking the resistor dissipates this energy as wasted heat. With the addition of regenerative drive, it allows current to flow back in the system's DC circuit or onto the building power supply line.

The HMI (Human-Machine Interface) visualizes the activities of the elevators and enable the operator to interact with the system.

2.4 **Programming languages used in the program**

2.4.1 Ladder Logic Diagram (LAD)

Ladder logic performs logic operations through symbolic notation in ladder diagrams. It can be used to simulate automation-related tasks like counting, timing, sequencing. As of today, ladder logic is still among the most used PLC programming languages. The fundamentals of ladder logic is shown in Figure 3 below.

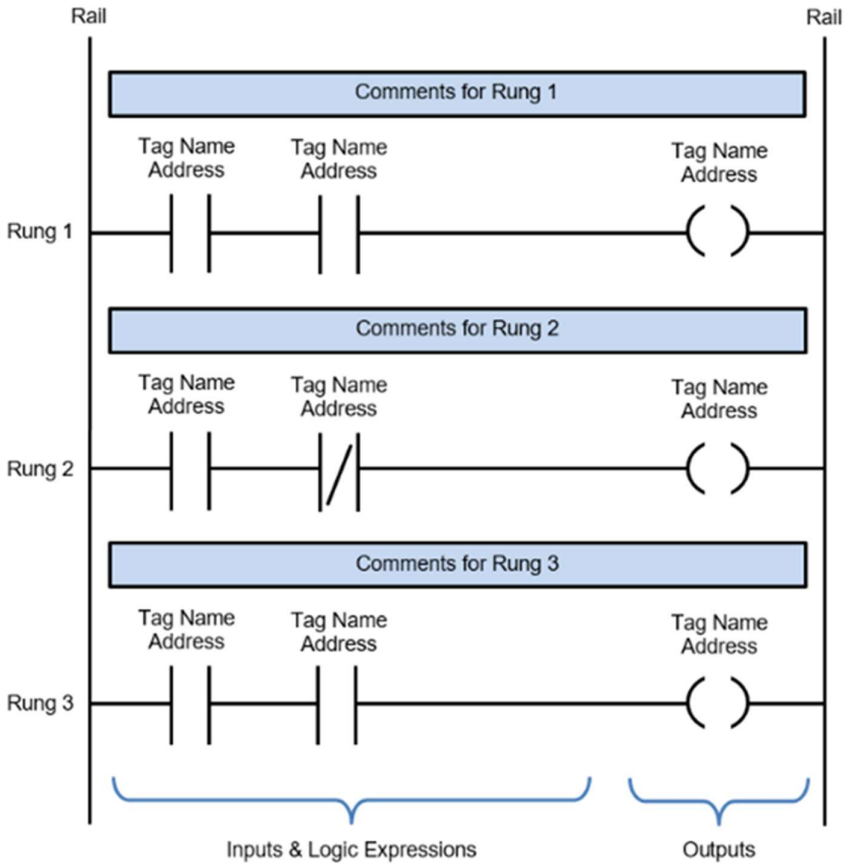

Figure 9. Fundamentals of ladder logic (Ladder Logic World, n.d)

- Ladder diagram: A type of schematic diagram that illustrates logic operations. A ladder diagram consists of two vertical power rails and horizontal logic rungs to form a ladder.
- Rails: Vertical lines in a ladder diagram that runs down the far most ends of the page.
- Rungs: Horizontal lines attached to rails that contains logic expressions.
- Inputs: Signals that represent the states of input devices such as sensors, buttons.

- Outputs: Signals that represent the states of output devices such as motors, conveyors. Outputs are triggered by input signals.
- Logic expressions: The logic operations that the PLC performs based on input and output data.
- Address Notation: Memory address of inputs and outputs in the PLC.
- Tag Name: A text line that describe the functions of inputs and outputs.
- Comments: Texts shown at the top of each rung that explain their functions.

2.4.2 Function Block Diagram (FBD)

FBD in PLC programming is a graphical language that express logic operations through blocks. In FBD, function blocks are used to execute functions and yield output values. Figure 9 shows how a function is expressed through a function block.

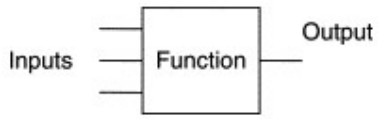

Figure 10. Function block in FBD

2.4.3 Logic expressions

- AND Operation

An AND Operation is an operation in which an output is only energized when there are two normally open switches. It can be understood in control systems that two input states should be both TRUE, 1 or HIGH for the output to be activated. Figure 11 shows the expression of AND Operation in LAD and FBD. The output is only activated when both input A and B in LAD, or 1 and 2 in FBD is true. Table 1 describes the relation between the output values and each value of the inputs.

Table 1. Truth table – AND Operation

Inputs		Outputs
A	B	
0	0	0
0	1	0
1	0	0
1	1	1

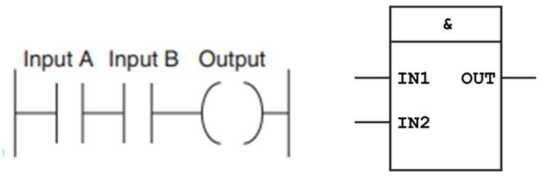

Figure 11. AND Operation in LAD and FBD

– OR Operation

On the contrary, an OR Operation output is energized when one of the two inputs is on. Figure 12 shows the expression of this operation. Table 2 describes the relation between the output values and each value of the inputs.

Table 2. Truth table – OR Operation

Inputs		Outputs
A	B	
0	0	0
0	1	1
1	0	1
1	1	1

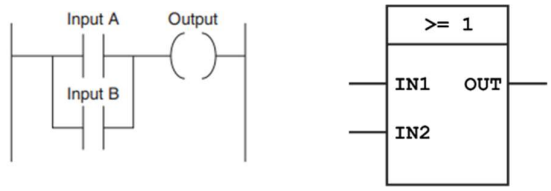

Figure 12. OR Operation in LAD and FBD

– NOT Operation

In a NOT Operation (figure 13), the output value is always the opposite to the input value (Table 3). This operation is also known as an inverter.

Input	Output
0	1
1	0

Table 3. Truth table – NOT Operation

Figure 13. NOT Operation in LAD and FBD

– NAND (NOT AND) Operation

The NAND Operation (figure 14) is the combination of the AND Operation and the NOT Operation. Unlike the AND Operation, the output of NAND is the opposite to AND's output (Table 4). Therefore, it can be activated in three cases instead of 1.

Inputs		Outputs
A	B	
0	0	1
0	1	1
1	0	1
1	1	0

Table 4. Truth table – NAND Operation

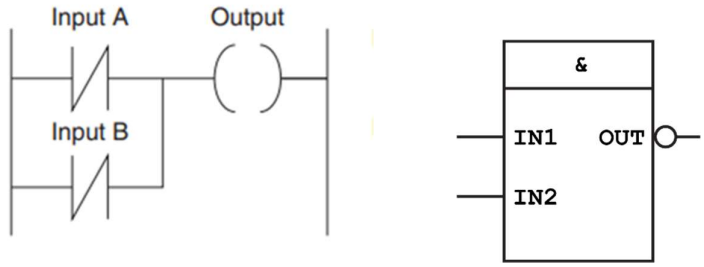

Figure 14. NAND Operation in LAD and FBD

– NOR (NOT OR) Operation

This is another combination of the OR and the NOT Operation. The output value in the OR operation is reversed in this case (Table 5). A NOR Operation (figure 15) can also be formed when putting a NOT gate on each input and then an AND gate for the output.

Inputs		Outputs
A	B	
0	0	1
0	1	0
1	0	0
1	1	0

Table 5. Truth table – NOR Operation

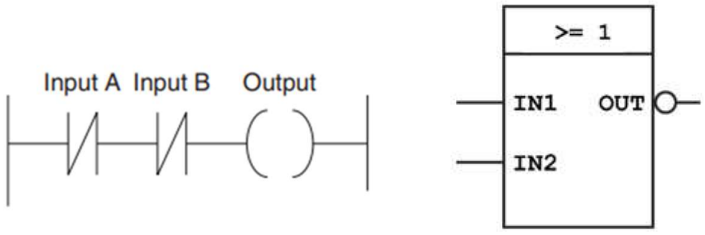

Figure 15. NOR Operation in LAD and FBD

– XOR (Exclusive OR) Operation

This is almost the same as the OR Operation, except that the output value is 0 when both inputs are activated (Table 6). As shown in figure 16 below, the output is True when only input A is True or only B is True. When both inputs are true then no conditions are met.

Inputs		Outputs
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Table 6. Truth table – XOR Operation

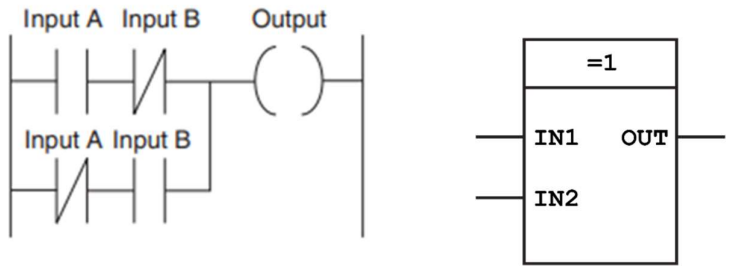

Figure 16. XOR Operation in LAD and FBD

– Assignment

''Assignment'' (figure 17) is an instruction used when one several or inputs can exclusively activate an output. This means that if the combined input values or result of logic operation (RLO) has the value of ''1'' then the output value is ''1, otherwise it will return to ''0''.

Figure 17. ''Assignment'' symbol in LAD and FBD

– Set output

The ‘’Set output’’ instruction (figure 18) has less impact over an output compared to Assignment. Particularly, the output is still activated when the RLO has the value of ‘’1’’, but it will remain unchanged even if the input condition is no longer met.

Figure 18. ‘’Set output’’ symbol in LAD and FBD

– Reset output

The reset output instruction (figure 19) has the opposite role to set output. When the RLO is ‘’1’’, the output will be deactivated, otherwise its state is unchanged.

Figure 19. ‘’Reset output’’ symbol in LAD and FBD

– TON (Generate on-delay)

A ‘’Generate on-delay’’ instruction (figure 20) is used to delay the setting of output “Q” by the programmed time (PT). It can be understood as a ‘’Assignment’’ with a timer. The PT is started when the RLO changes from 0 to 1 (positive signal edge). When the time is up, the output “Q” is set to ‘’1’’ until the RLO turned to ‘’0’’. The ET port represents the current time value. It is also reset when RLO changes to ‘’0’’.

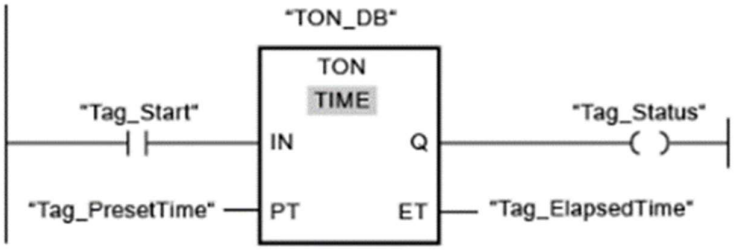

Figure 20. Example of ‘’TON’’ in LAD and FBD

3 IMPLEMENTATION OF THE PROGRAM

This chapter describes the details of the controlling program, step-by-step instructions of making the program project and explanation of each functions.

3.1 **Description of the controlling program**

In this section, features and logic commands made for the program are mentioned and explained.

3.1.1 Controlled and simulated elevator components

The listed components below are classified as either inputs or outputs for declaration in the program.

Floor door and cabin door (outputs): Since the car door and floor door operates simultaneously in real time, there is one output signal that simulates their operation.

Car motor (outputs): There are two output signals for moving the elevator car upwards and downwards.

Sensors (inputs)

- Obstacle sensor: Detects if there is an object or person between the door sides.
- Overload sensor: Detects if the elevator is overloaded.
- Floor level sensor: Indicate which floor the elevator is at.

Buttons (inputs)

- Open and Close buttons in elevator car.
- Car call buttons: Call buttons inside the elevator car.
- Floor call buttons: Call buttons on each floor.

3.1.2 Programmed operations

The elevator is designed to move between four floors. Initially, the car is at the first floor, meaning that the it is at the very start of the operation. If the elevator is at another floor from the beginning, it makes no problem for the program. Signals from call buttons will activate the movement of the car.

It will move in a power saving order, which means that if the car is moving in one direction and there is another call to an opposite direction, it will respond to all the calls in the initial direction before responding to the latter. For example, if the elevator just went up from floor 1 to floor 3 and there are two more calls to floor 5 and floor 1, it will continue moving up to floor 5.

The elevator finishes a call when the call signal matches the floor level sensor. For example, a call to the second floor is finished when the signal from that floor's level sensor is high and a floor call was made. Then, all the call signal to the second floor is reset to 0. The door is then opened for

a fixed period then closed before the car moves to another floor. The car can only resume moving when the door is fully closed. When the Open button is pressed, the door is reopened if it is closing, if it is still opened then the open time is reset. There is also a fixed period after the door is fully closed again to wait for the Open button signal before the car moves again.

If there is an obstacle between the doors or the elevator is overloaded, then the door is remained opened or reopened if is closing.

An HMI screen will be shown in the simulation step to for visualization of all operations.

3.2 **Device configuration**

The section demonstrates how to select the devices that is going to be simulated in the program.

After creating a new project, select ''Configure a device'' in the opening window ''First steps''.

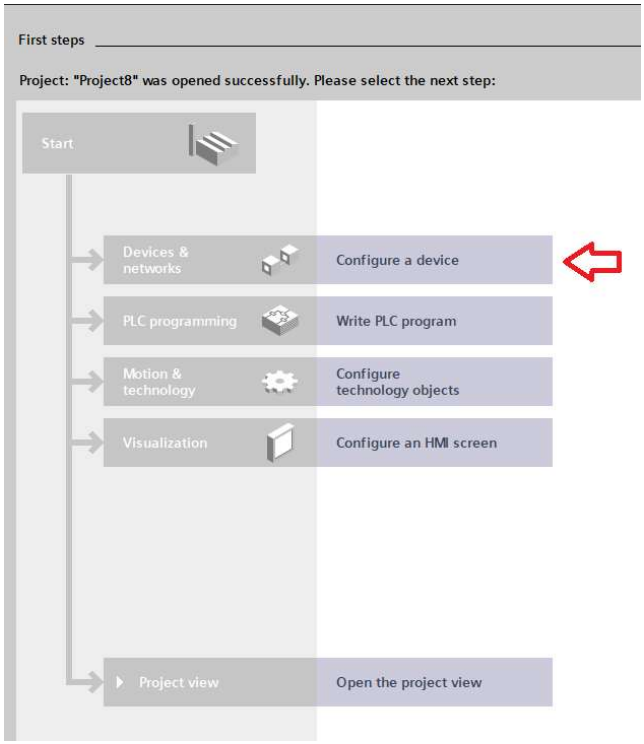

Figure 21. ''Devices & networks'' option

Next up, select ''Add new device'' and choose a desired controller. If project involves testing with real devices, then users should choose the same model number as in the software. In this project the selected device is a 1215 DC/DC/Rly CPU.

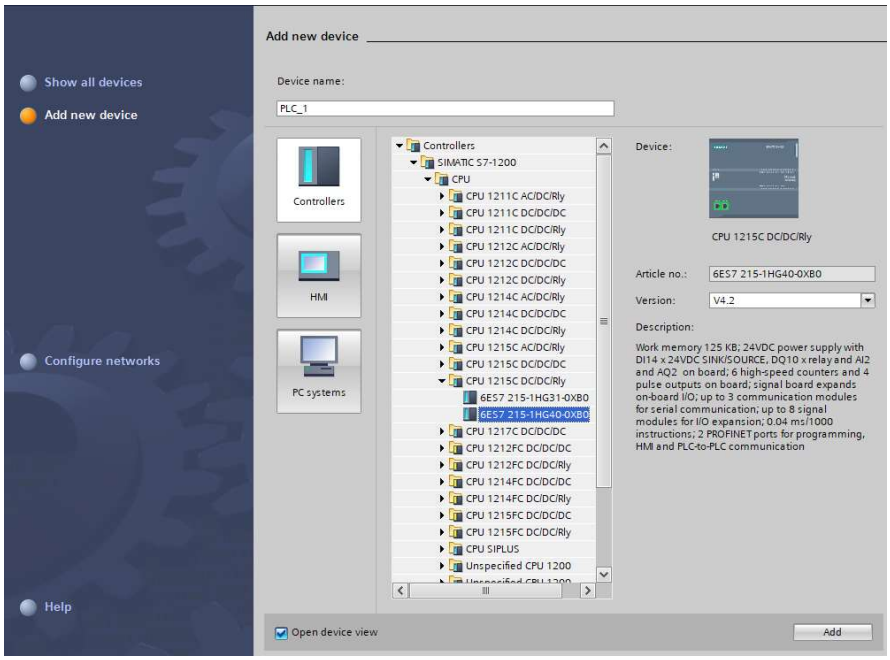

Figure 22. Controller selection

After selecting a controller, an HMI Screen is also needed for visualization. The selected device is a KTP700 Basic Panel.

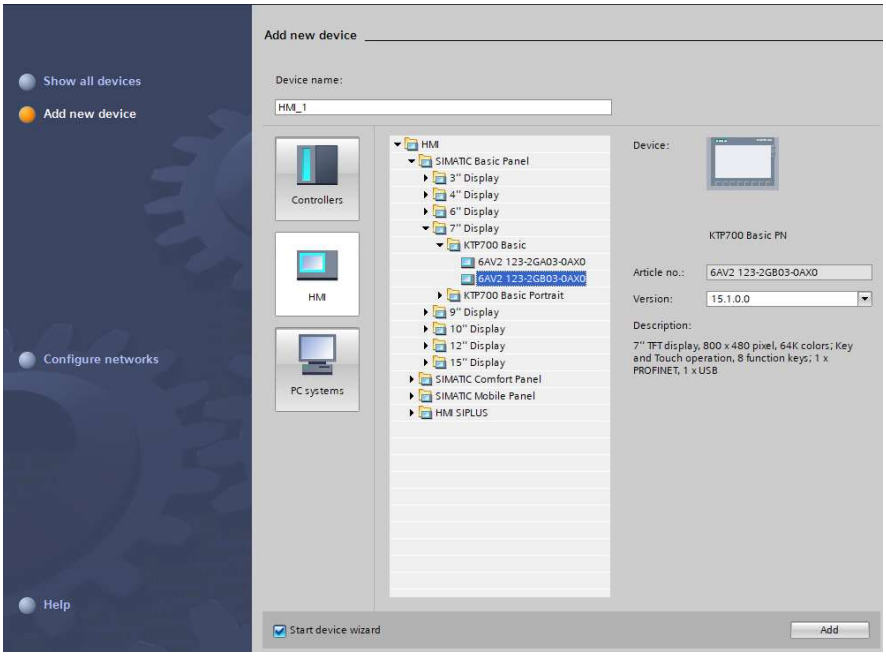

Figure 23. HMI Screen

When an HMI is added, a new window will open showing the connection between the devices. The PLC is connected to the HMI screen through a PROFINET subnet.

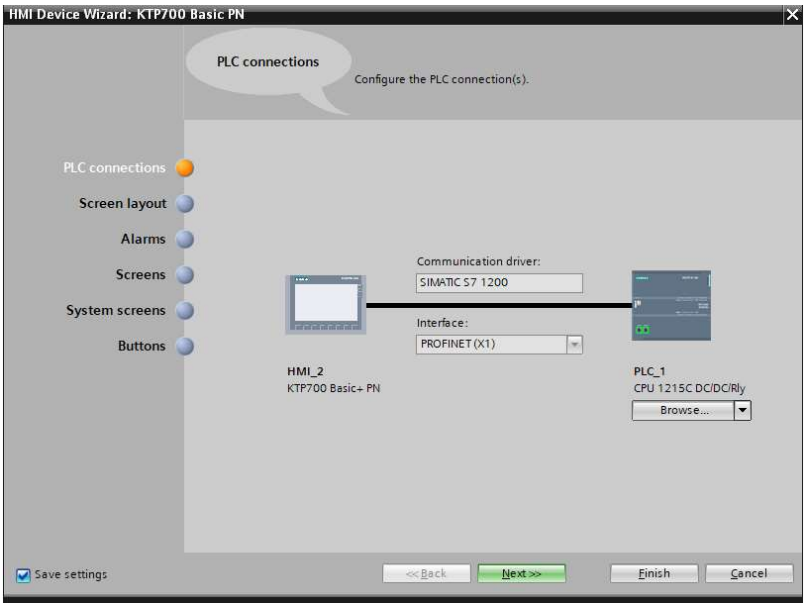

Figure 24. HMI Device Wizard

The hardware configuration of the system can be viewed by accessing the ''Device & networks'' tab.

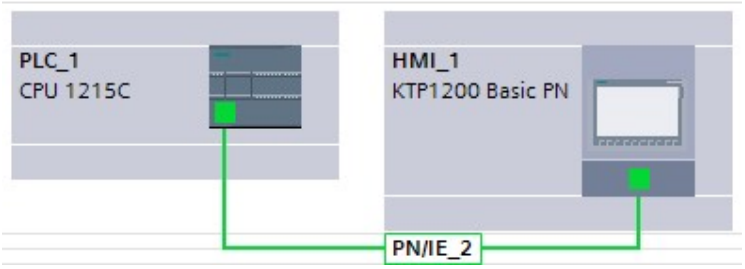

Figure 25. Hardware configuration

To view the PLC's information, double click ''PLC_1'' in figure 25. A rack installation of the CPU will pop up like in figure 26 below, then double click ''PLC_1'' again. The general tab below the rack indicates the PLC's data such as its Ethernet address or the number of I/O channels available.

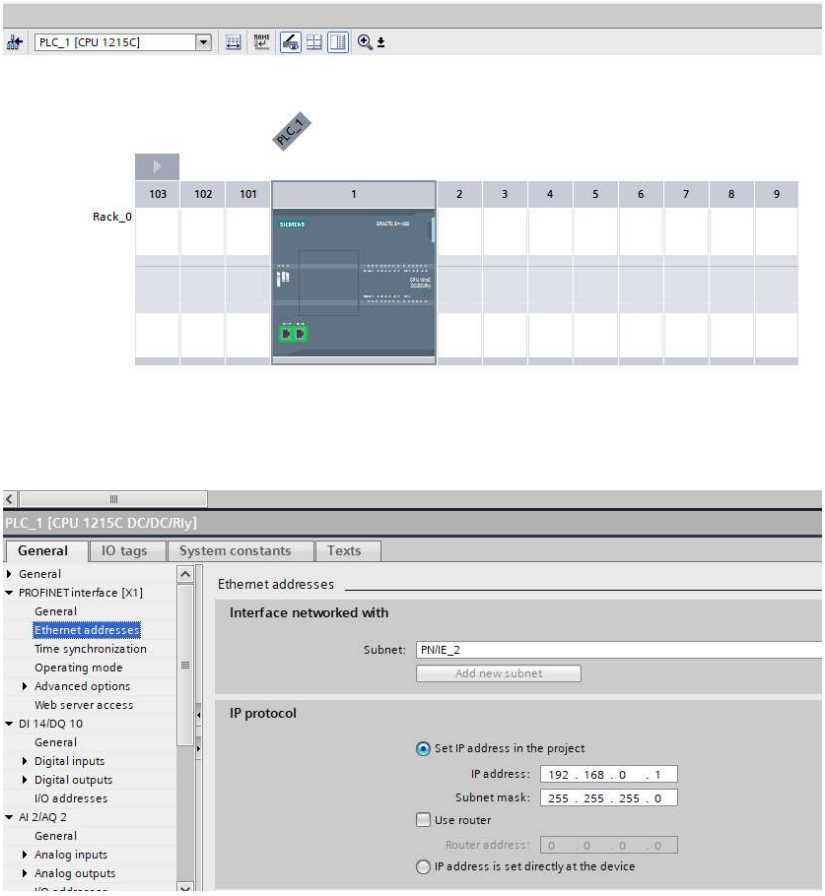

Figure 26. Information of the PLC

3.3 **Beginning programming**

This chapter demonstrates the preliminary steps prior to programming tasks.

A program block can be opened by double clicking ''Main'' or ''Add new device'' under ''Program blocks''/''PLC_1'' in the Project tree. It is also recommended to declare all I/O tags before programming. Tags can be made in either the ''Default tag table'' or in a new one under ''PLC tags''.

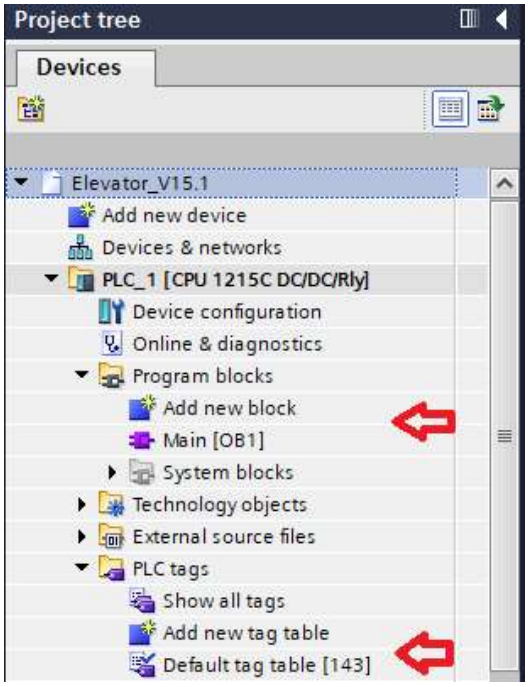

Figure 27. Open a new program block and a tag table

The program can be viewed both in LAD and FBD. Switching between these languages is possible by right clicking the program block and select ''Switch programming language''.

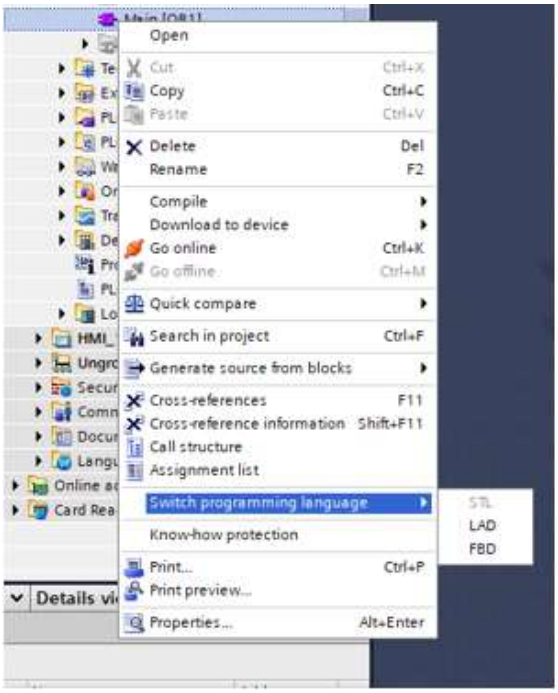

Figure 28. Switching between LAD and FBD

Figure 29 shows the view of program block. Default instructions which are normally open and closed contacts, assignment, empty box, open and close branch are placed on top of the network tabs. All other instructions are included in the right-hand side window.

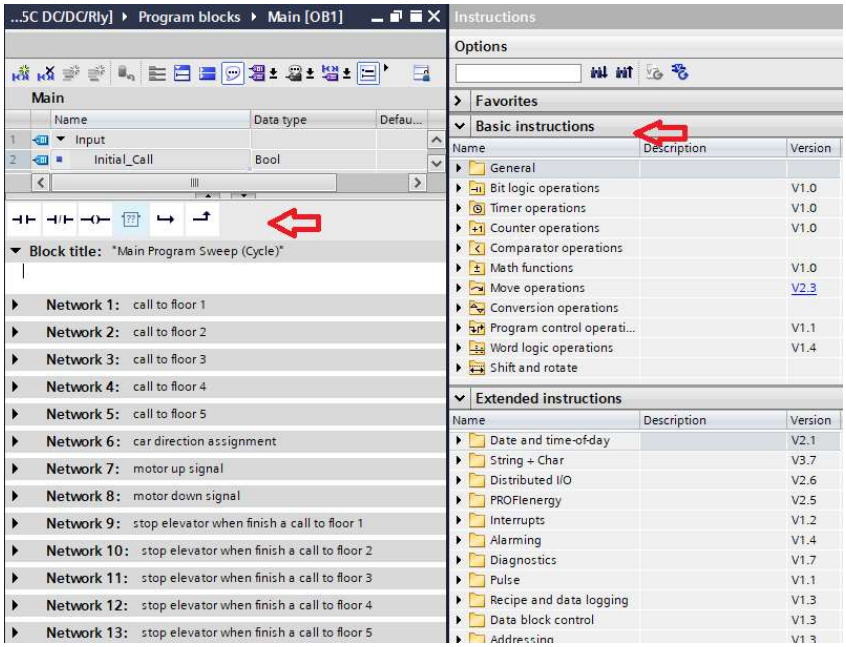

Figure 29. Instructions

3.4 **Description of logic operations**

3.4.1 Elevator calls

The first and most important logic operation is the elevator calls. These are the signals that trigger the movement of the elevator. Elevator calls consist of two types, car calls and floor calls. Car calls are requests made from buttons inside the elevator car to instruct the car to move towards a desired floor. Since there a total of five floors, there will be five car call buttons, each representing the number of a floor. Floor calls are made by pressing the call buttons on each floor. Therefore, there are also five floor call buttons. Figures 30 and 31 describe the logic operation of an elevator call.

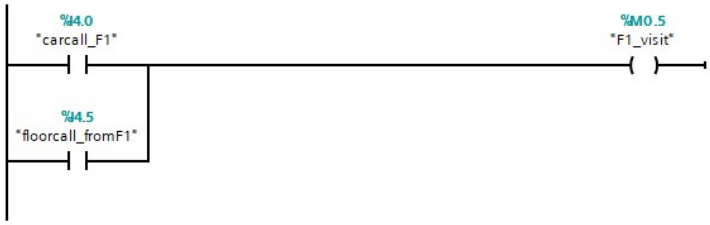

Figure 30. Elevator calls in LAD

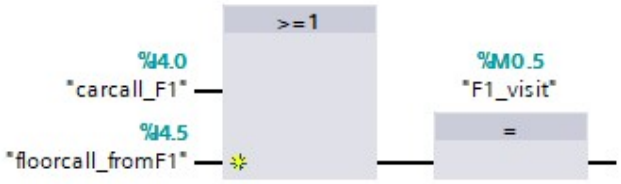

Figure 31. Elevator calls in FBD

Since only buttons can activate a call, the ''assignment'' instruction is used to activate the output. Tag name ''carcall_F1'' represents the car call to the first floor. Tag name ''floorcall_F1'' represents the floor call to the first floor. This syntax is the same for the other four calls.

3.4.2 Elevator location

In real life applications, floor level sensors are used to indicate elevator's location. This data can also be used to stop or move the elevator car. To simulate the signals of the sensors, a memory input with a changeable integer value is used ("step_count"). Its beginning value is 0. As the elevator moves up or down, the value will be added or subtracted, which gives out different values that represents a floor level signal. The logic operation that controls this input is shown in figures below.

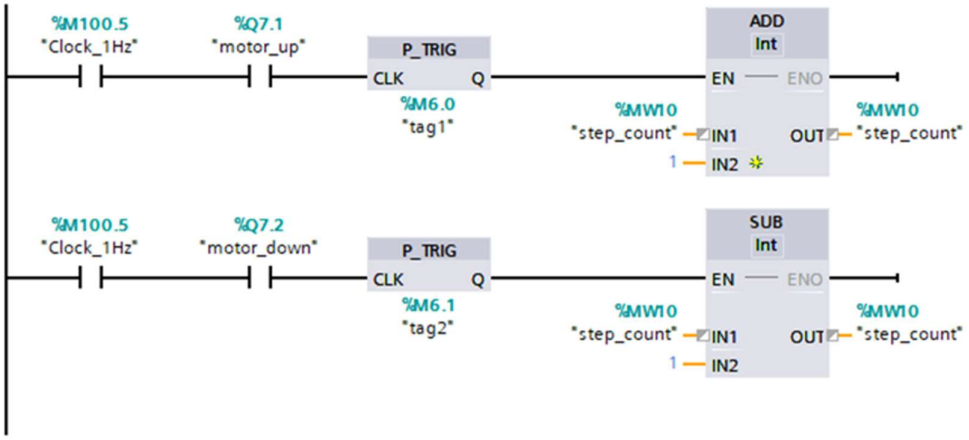

Figure 32. Control ''step_count" in LAD

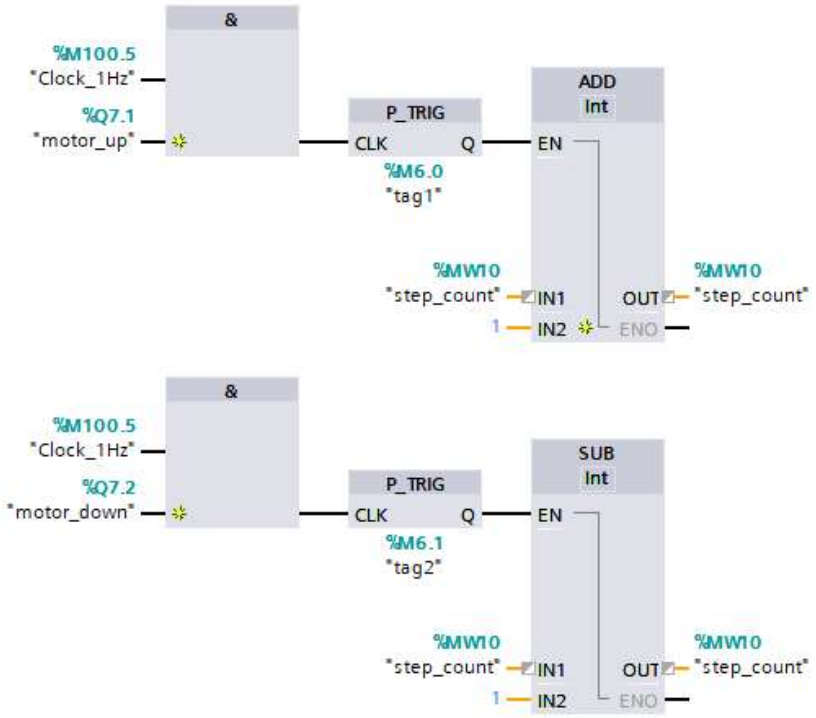

Figure 33. Control ''step_count'' in FBD

In each output end, the ''ADD''' and "SUB" (subtract) functions are responsible for increasing and decreasing ''step_count'' with the deviation of 1. If only inputs ''motor_up'' and ''motor_down" (elevator is moving up and down) are used then the outputs will be activated just once, which means that ''step_count'' can only reach to ''1'' at maximum. This is where a clock memory bit comes in to continuously feed the output. Clock memory bits are commonly used to activate flashing lights or to trigger periodic activities. Each bit is assigned to a frequency. Bit number 7 is used in the program since it best suited to the simulation speed for easy monitoring.

Table 7 below shows the clock memory bits and their corresponding periods and frequencies.

Table 7. Clock memory bits and frequency

Bit	7	6	5	4	3	2	1	0
Period (s)	2.0	1.6	1.0	0.8	0.5	0.4	0.2	0.1
Frequency (Hz)	0.5	0.625	1	1.25	2	2.5	5	10

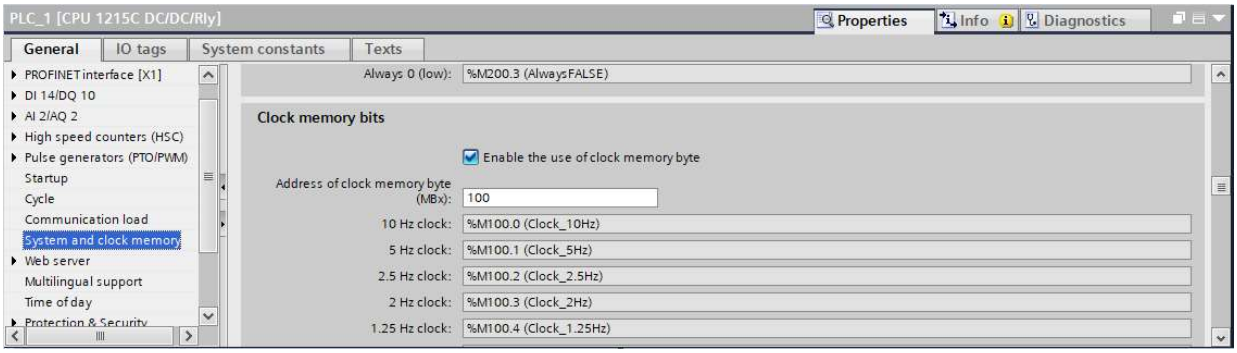

Figure 34. How to enable clock memory byte in TIA Portal

Elevator's corresponding location signal to the value of ''step_count'' is shown in the figures below. ''F1_sensor" means that the car is at the first floor, the syntax is the same for other outputs. Since these outputs can only be set when "step_count'' reach a certain value and are reset when the input changes, the ''assignment'' instruction is chosen for this operation.

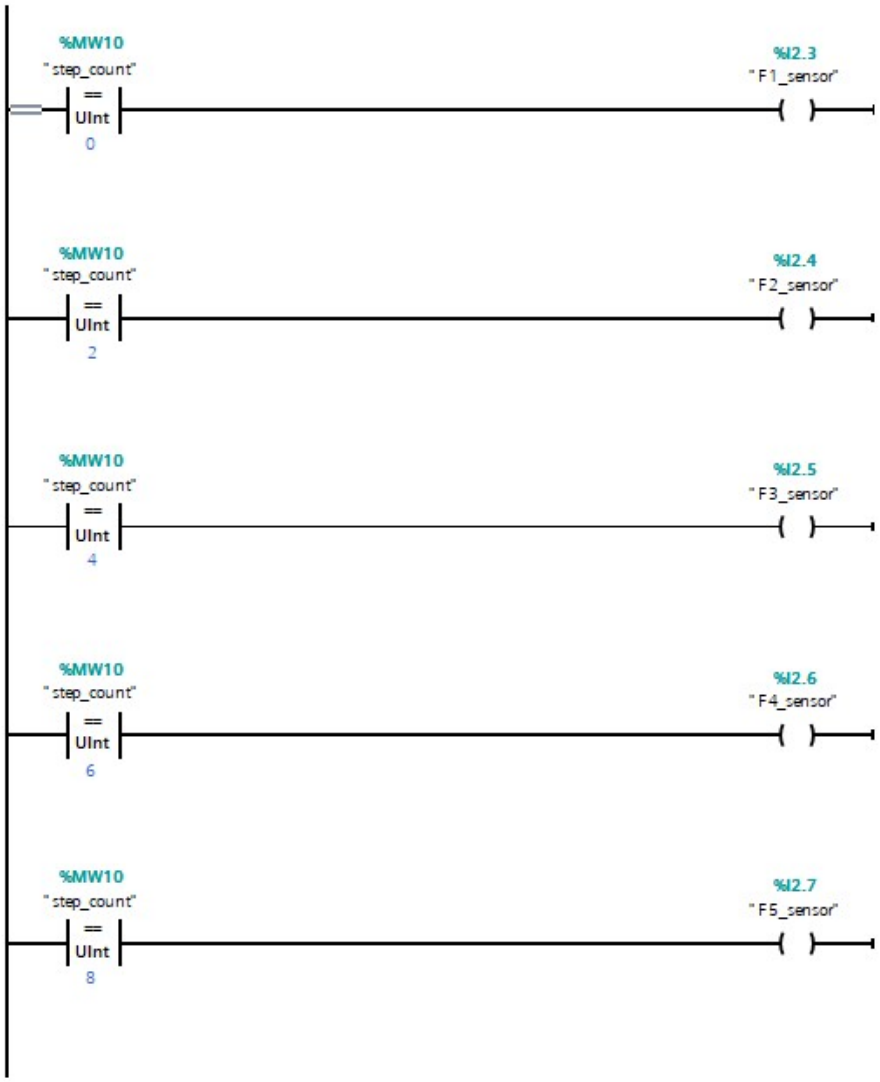

Figure 35. Elevator location signals in LAD

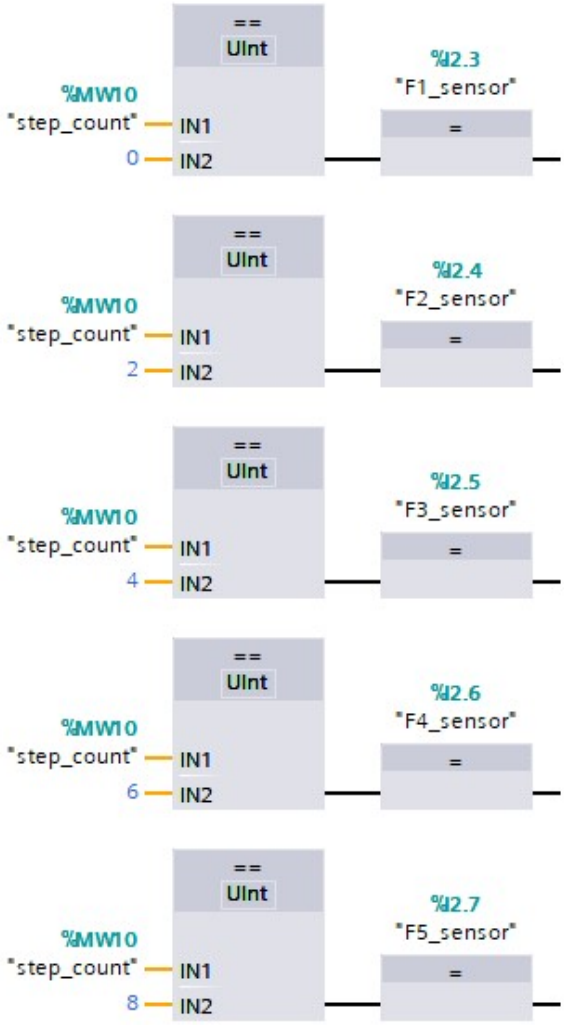

Figure 36. Elevator location signals in FBD

3.4.3 Elevator direction assignment

As previously mentioned, the elevator's movement is designed to save motor power efficiently. This is done by instructing it to finish all calls in one direction (upward or downward) before responding to the remaining calls. To do this, two input memories are made (''up_memory'' and ''down_memory'') to represent the current direction of the elevator car. The setting and resetting of these inputs can be seen in the figures below.

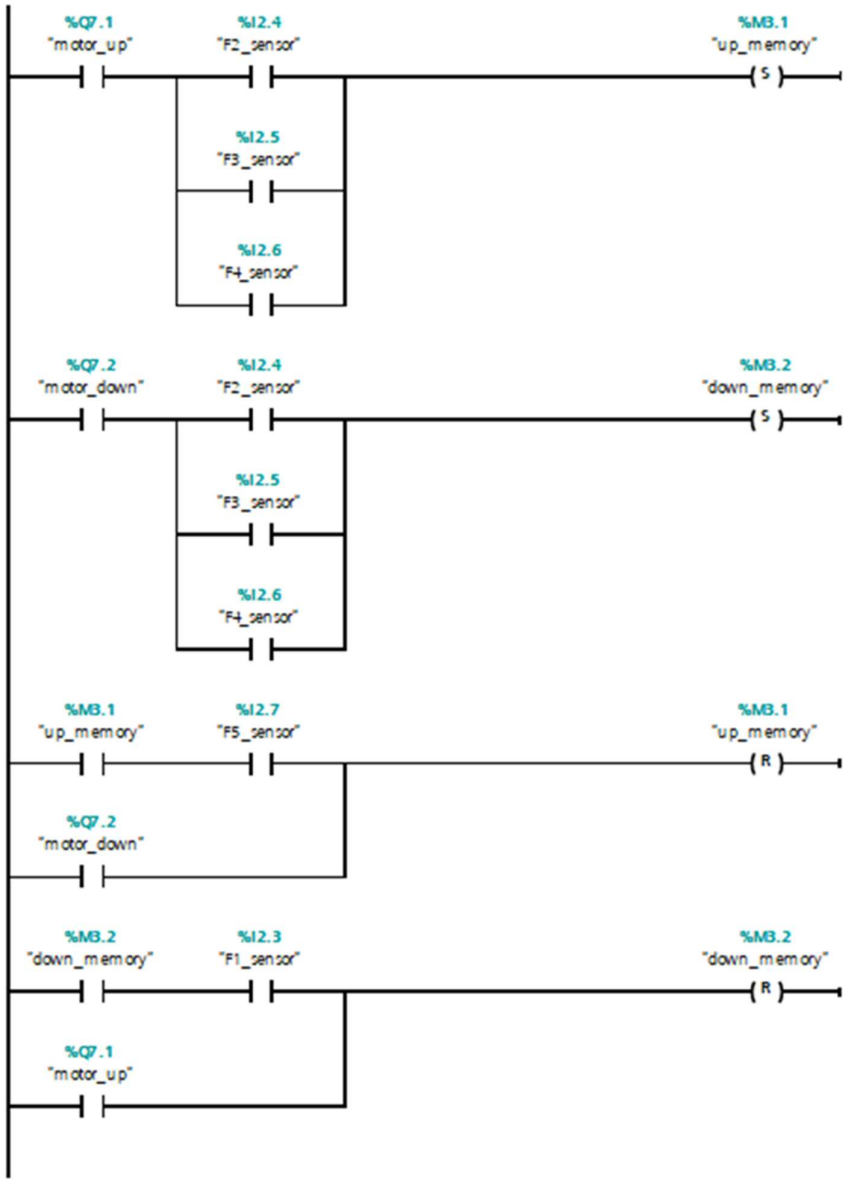

Figure 37. Direction assignment in LAD

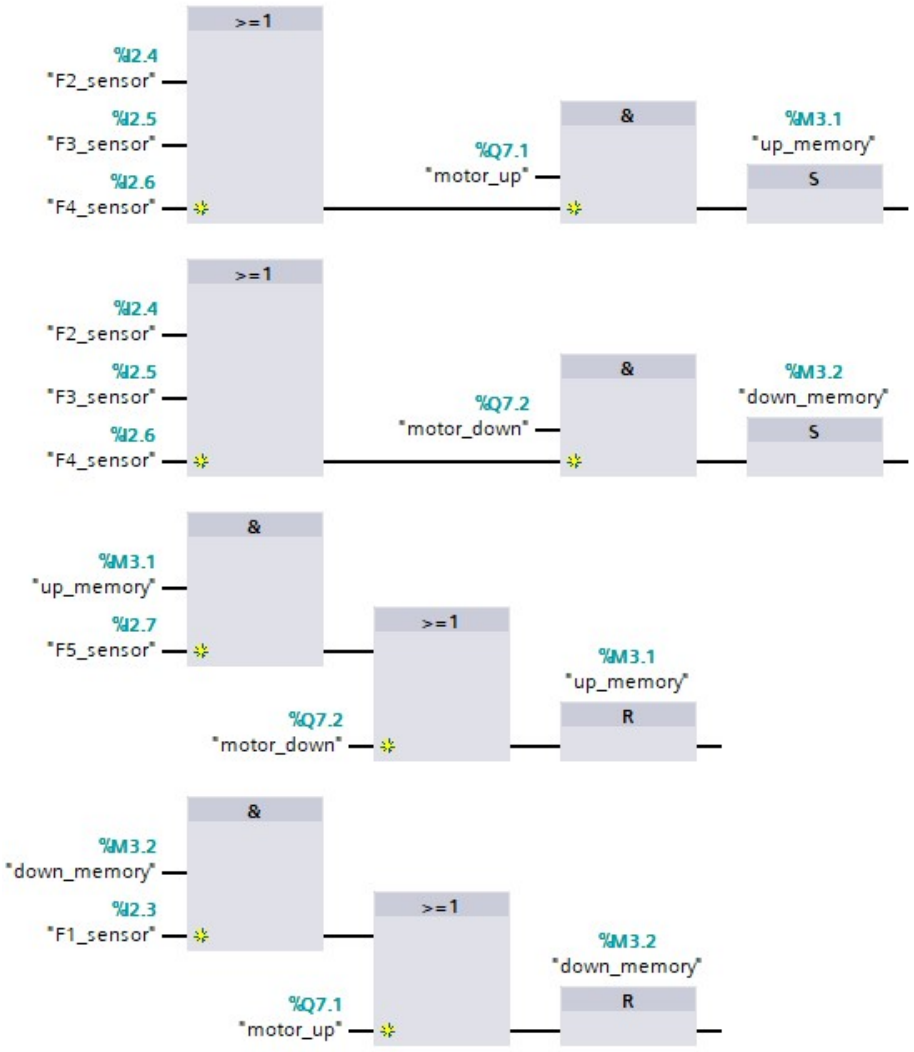

Figure 38. Direction assignment in FBD

In the first rung in LAD and in the first set of blocks in FBD, the ''up_memory" memory output is set by an AND operation between "motor_up'' and the OR operation of "F2_sensor'', ''F3_sensor'' and ''F4_sensor''. This means that if the system responds to the upward signal and the elevator goes to floor 2, 3 or 4 then the car is assigned with the upward direction. The output is reset (in the third rung) if the car reaches the fifth floor, which means that it can only go downward from this point.

One important notice is that direction assignment is only used when there are calls from both directions, in order to prioritize one over another. For example, if the upward direction is assigned to the car while it only has calls from lower floors left, it will still go downwards.

Likewise, the car is assigned to the downward direction if there is a downward signal at floors 2, 3, 4. The ‘‘down_memory’’ output is reset when the car comes down to floor 1.

Outputs used for direction assignment can also be reset using motor signals. The reason for choosing these inputs will be further explained in section 5.3.

3.4.4 Elevator movement

The following networks represents the logic operation that triggers the motor to move the elevator up or down.

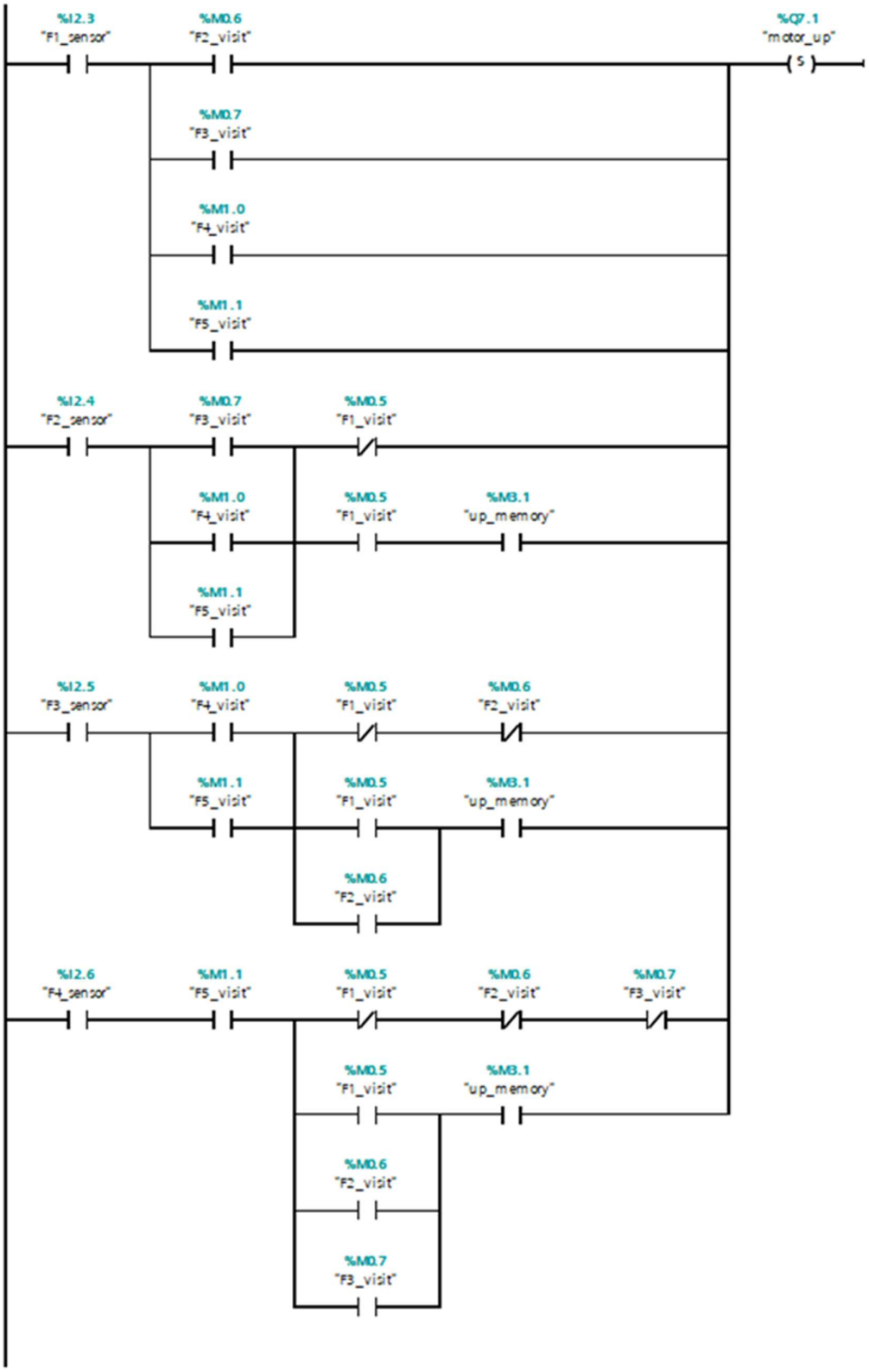

Figure 39. Elevator upward movement in LAD

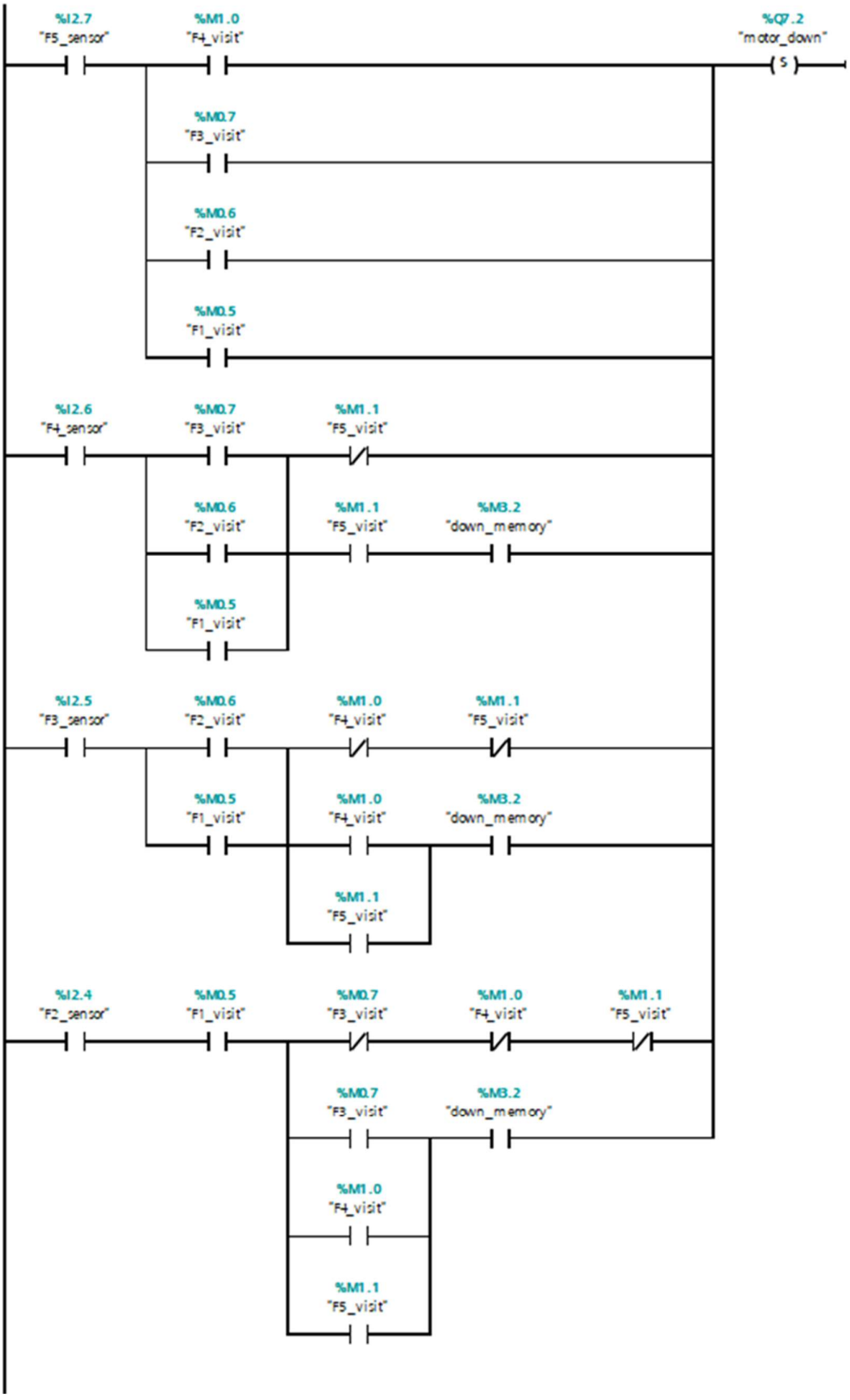

Figure 40. Elevator downward movement in LAD

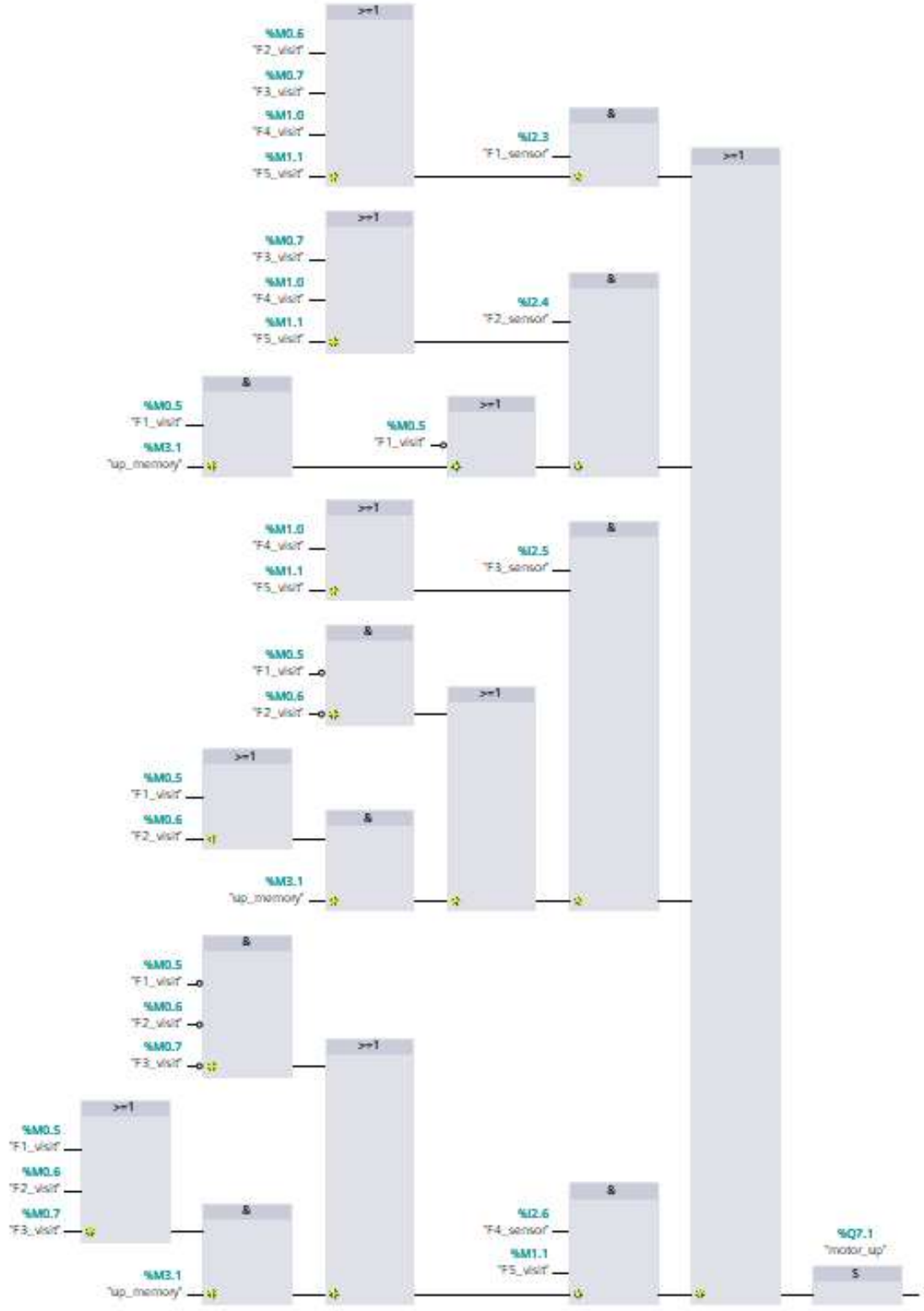

Figure 41. Elevator upward movement in FBD

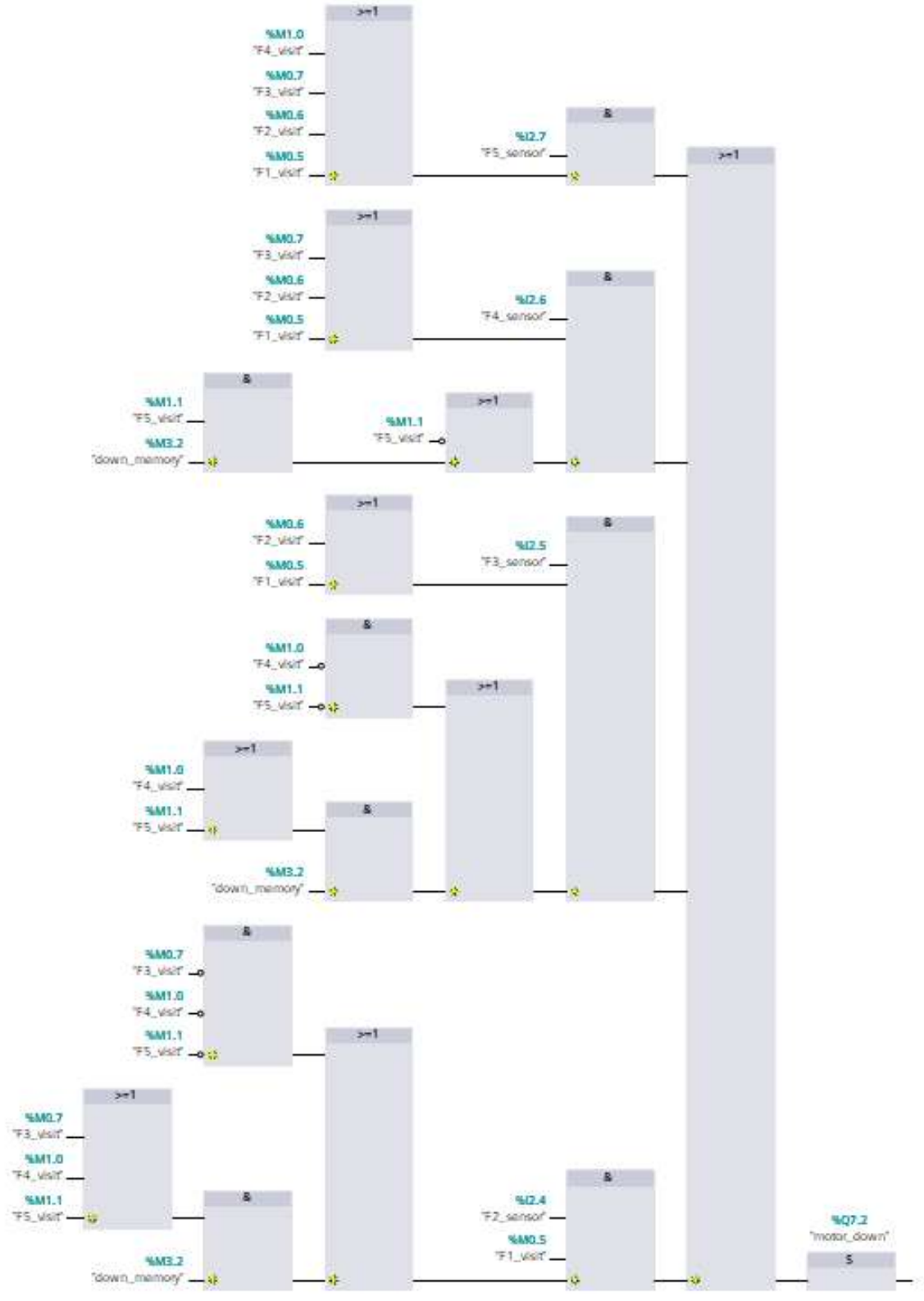

Figure 42. Elevator downward movement FBD

There are four cases where the elevator is instructed to move up:

- The first one is when the car is at the first floor and there are calls from the upper floors.
- In the second case where the elevator is at the second floor, there is another OR operation added to the initial condition which is similar to the first case. The first additional input (''F1_visit'') can be understood as: if there is no request to the first floor and there are calls from upper floors then the motor will go upwards. The second and third inputs (''F1_visit'' and ''up_memory'') make use of the direction assignment mentioned above. It means that if there are simultaneous calls to the upper floors (upward calls) and to the first floor (downward call) and ''up_memory'' is True (the car has just moved up), then the car will continue to move upwards.
- Similar to the second case, when the car is at the third and the fourth floor, it will move up on two conditions: there are only upward calls or there are calls from both directions, but it has just moved up.
- The logic operation for motor moving down signal is the same as moving up, with ''down_memory'' being the input to prioritize the downward movement.

3.4.5 Stopping elevator and opening door

Figures 43 and 44 describe the operation of the elevator when it finishes a call.

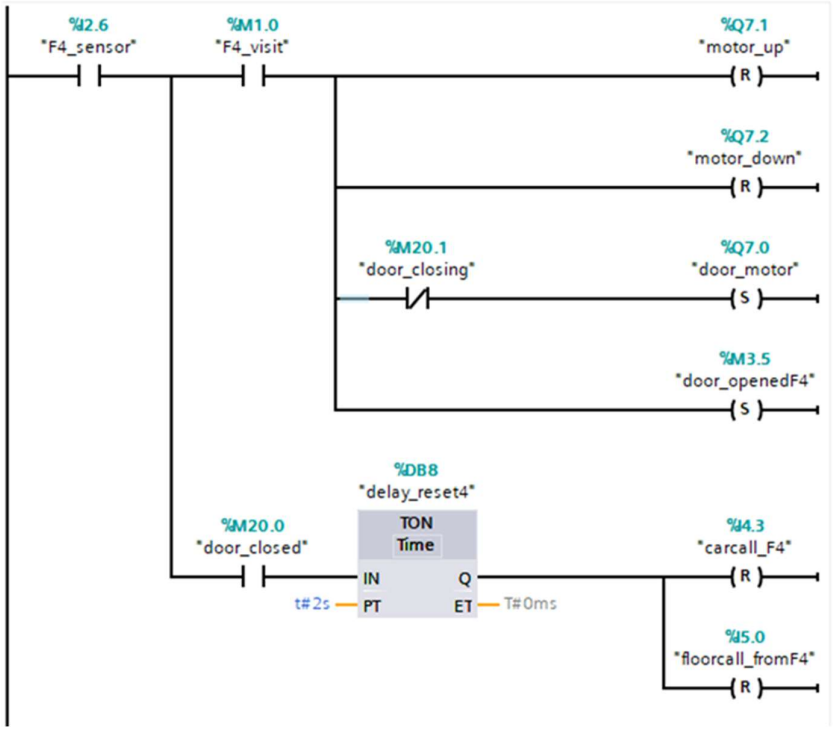

Figure 43. Stopping elevator and opening door in LAD

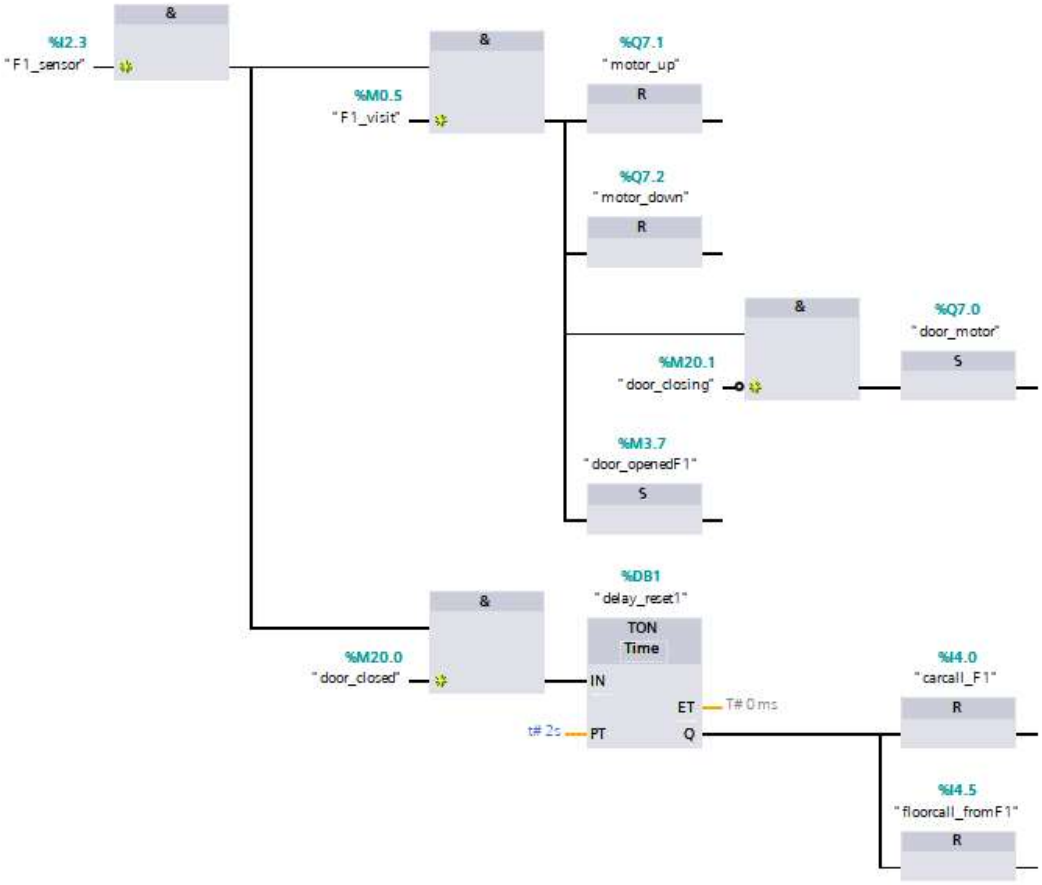

Figure 44. Stopping elevator and opening door in FBD

The elevator is stopped by resetting ''motor_up'' and ''motor_down''. The condition for stopping the motor is met when the elevator's floor number matches call requests to the same floor. For example, if ''F4_sensor'' and ''F4_visit'' is both True then the motor is stopped.

The elevator and floor doors are open by setting ''door_motor''. The input memory ''door_closing'' will be explained in the following network.

''door_openedF(n)'' represents the open-end signals, which indicates that the door is fully opened. It will be used for door closing operations. In real life applications this input is set by an actual open-end sensor, for example a relay sensor.

The input memory ''door_closed'' represents the door's close-end signal, which means that the door is fully closed. The condition for setting it will also be explained in the following networks. Before call buttons are reset in the last rung, which will unforce stopping the motor, an on-delay timer was added to wait if the open button was pressed to open the door again.

While the motor is stopped, ‘’motor_up’’ and ‘’motor_down’’ remains False until call buttons are reset when the door is fully closed. This is to disable the elevator from moving when the door is opened and to ensure passenger’s safety in real life. For example, in the picture above, when both calls are reset then ‘’F4_visit’’ is also reset, meaning that “motor_up’’ and ‘’motor_down’’ are no longer reset. Then the elevator can start moving again if there is any remaining call.

3.4.6 Closing door

The following networks illustrate the door closing operation when there is no interference from the open button, the obstacle sensor and the overload sensor.

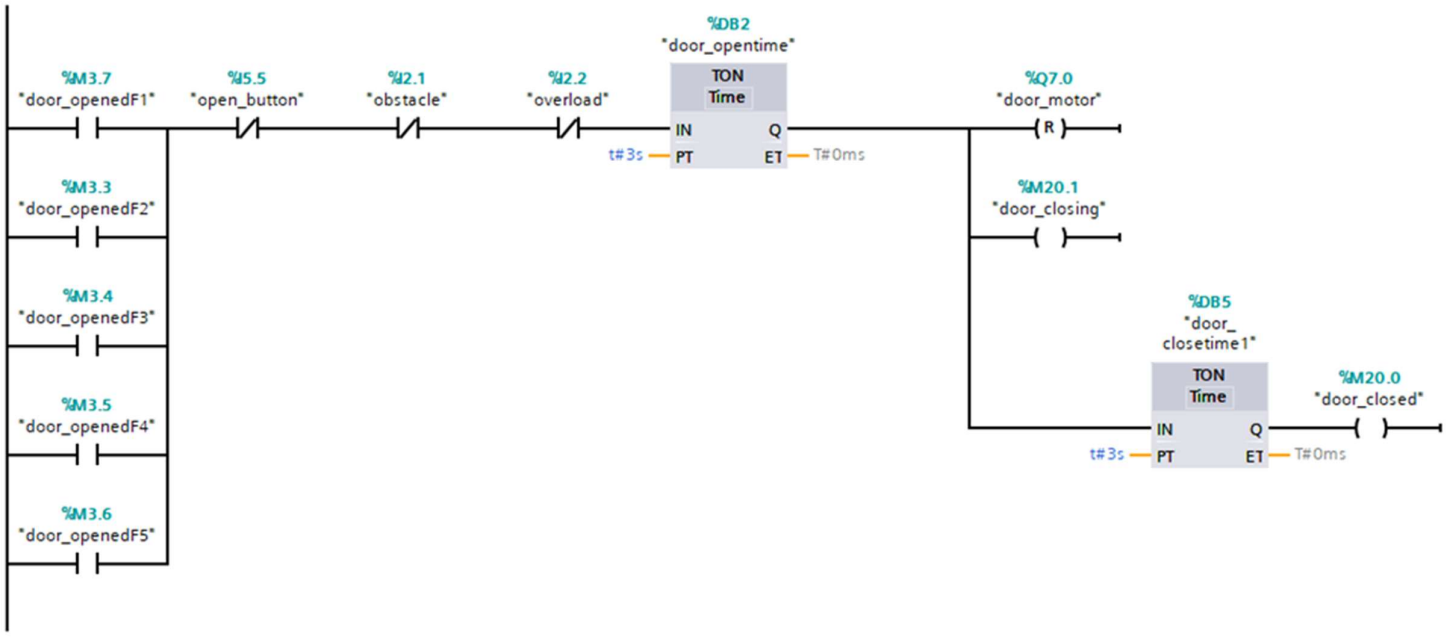

Figure 45. Closing door in LAD

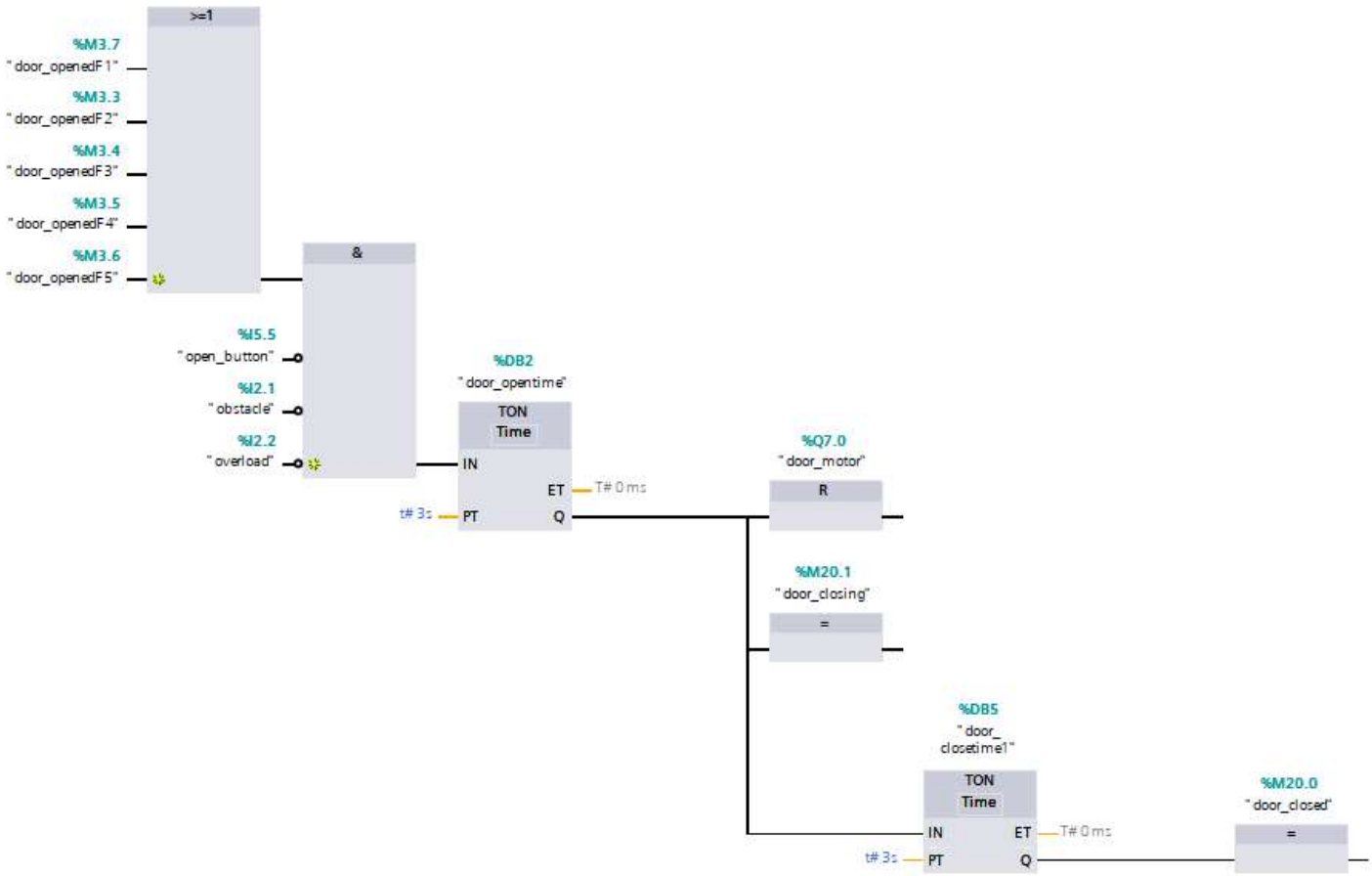

Figure 46. Closing door in FBD

Whenever an open-end signal is set (''door_openedF(n)''), an on-delay timer of is set to represents the time the door remains opened. After that, output Q of the timer makes the following instructions:

- ''door_motor'' is reset to close the door.
- Input memory ''door_closing'' is assigned to block ''door_motor'' from being set in the previous network.
- Another timer is set to represents the time it takes the door to fully close. After that ''door_closed'' is assigned.

3.4.7 Open button, overload sensor and obstacle sensor operation

This operation waits for signals from the open button, the door obstacle sensor or the overload sensor to cancel the operation of the previous network.

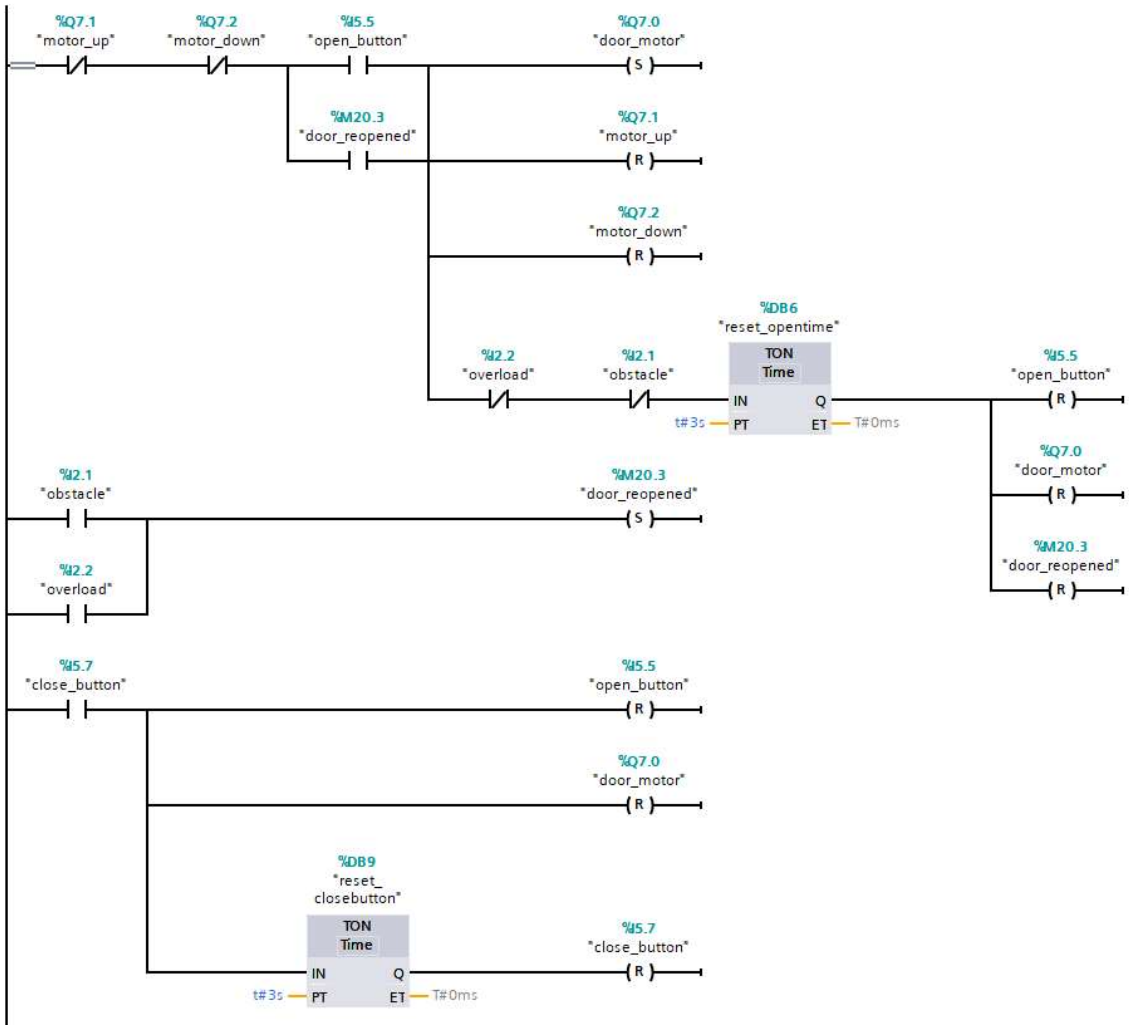

Figure 47. Open button, obstacle and overload operation in LAD

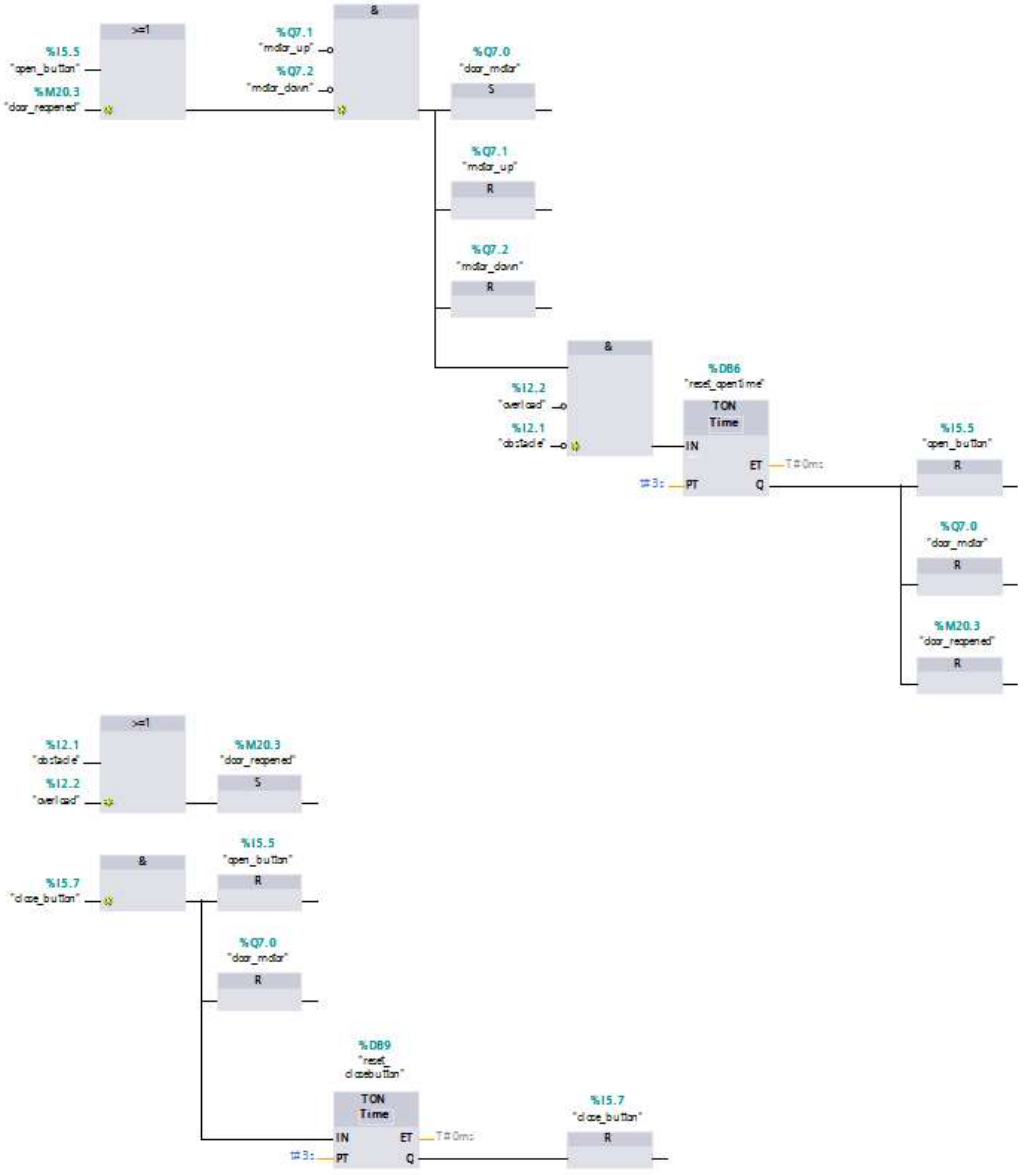

Figure 48. Open button, obstacle and overload operation in FBD

The first scenario where the Open button is used is when the car finishes a call and the door starts opening. At this time if the Open button is pressed then the open time in the network shown in figure 45 or 46 is halted. When the Open button is reset after three second (figure 47 or 48), the timer in previous network is initialized again.

In the second scenario, when the door is closing (''door_motor'' is reset), if the open button is pressed then the door is reopened. The timer in the previous network is also reinitialized after three seconds.

The third scenario happens when the door is fully closed and then the Open button is pressed, which can be understood as when passengers cannot come out of the car in the first attempt.

In all scenarios the three second timer cannot be activated if there is a signal from either the obstacle or the overload sensor. The door will then be remained opened until both signals are false.

When the door is closing and one of these signals is activated then memory input ‘’door_reopened’’ is set, which then set ‘’door_motor’’ and opens the door. It is also remained open until both signals are false.

If all inputs are true (open button, obstacle and overload sensors), the operation keeps the door opened until both obstacle and overload sensor signals are False, then the open button is reset after three seconds.

3.4.8 Reset door open memory

Since ‘’door_openedF(n)’’ inputs are used to close the door (section 3.4.6), there must be a reset operation for them, otherwise they would be kept closed even when the car finishes a call.

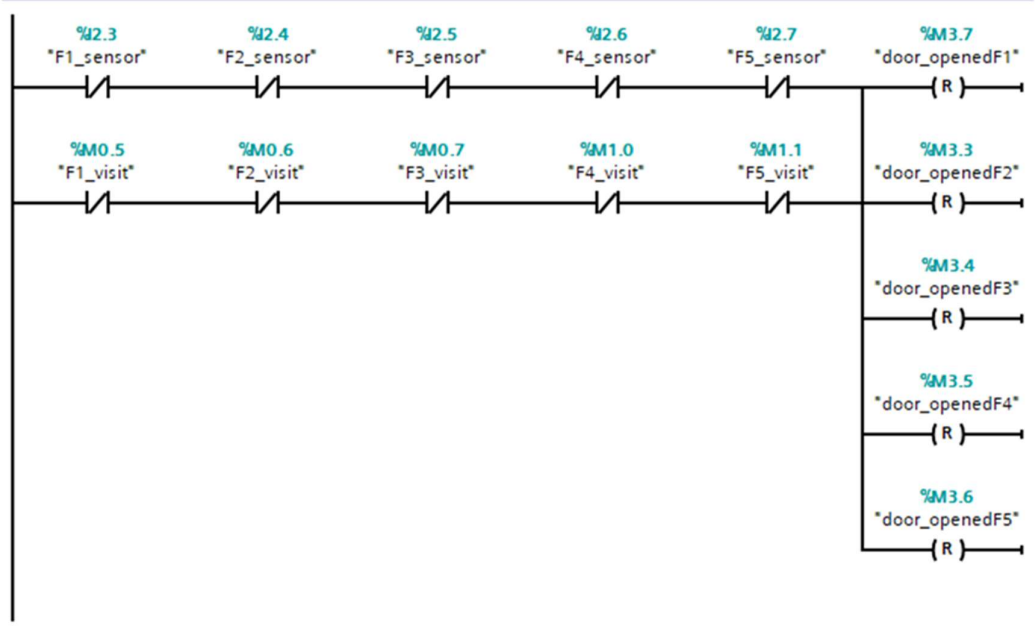

Figure 49. Reset door open memory in LAD

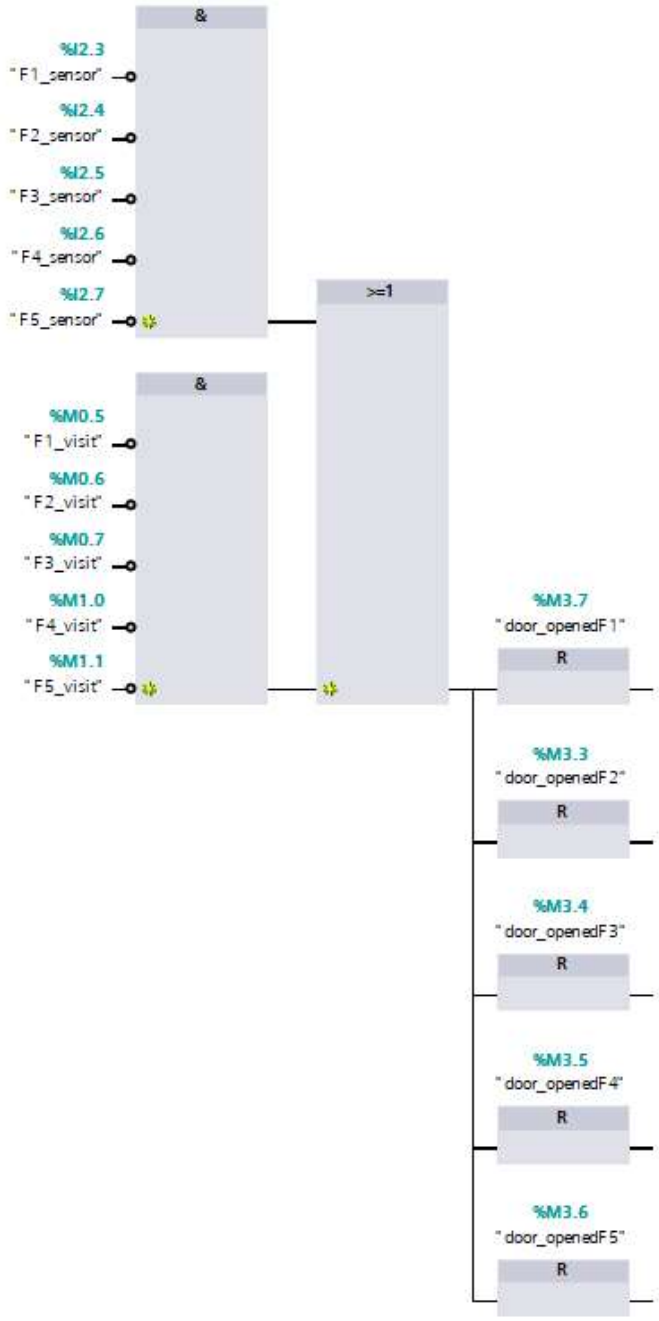

Figure 50. Reset door open memory in FBD

The first condition to reset these inputs is when the elevator leaves a floor and no floor sensor is active. Therefore, a normally closed contacts for all floor sensor inputs is used in the first rung. However, when there is no other call then the car will just stay put, so at least one of the sensors will be active. This require for another condition which is shown in the second rung: there are also five normally closed contacts, but the inputs are elevator calls to all five floors, meaning that if there is no call left then the door open memory inputs will also be reset.

3.5 Possibility of program scaling in taller buildings applications

The controlling program is suitable to be adjusted for managing single elevator systems for buildings with several more floors. The fundamental logic would remain the same and require only the addition of pushbuttons, floor level sensors and some changes to existing networks:

- More instructions for elevator calls will be added, in correspond to additional call buttons.
- The elevator direction assignment network will be extended since their logic is based on the elevator calls.
- Similarly, networks that instruct the motor to move the car up or down will be added with inputs that indicates the elevator's floor location and call requests.

4 VISULIZATION WITH WINCC HMI SCREEN

The simulation screen shown in figure 51 consists of six types of signals, which are Car Location, Floor Buttons, Car Buttons, Car Movement (Direction), Door Status and Signal Triggers. These signals are represented by the Button element in the HMI Toolbox (figure 52).

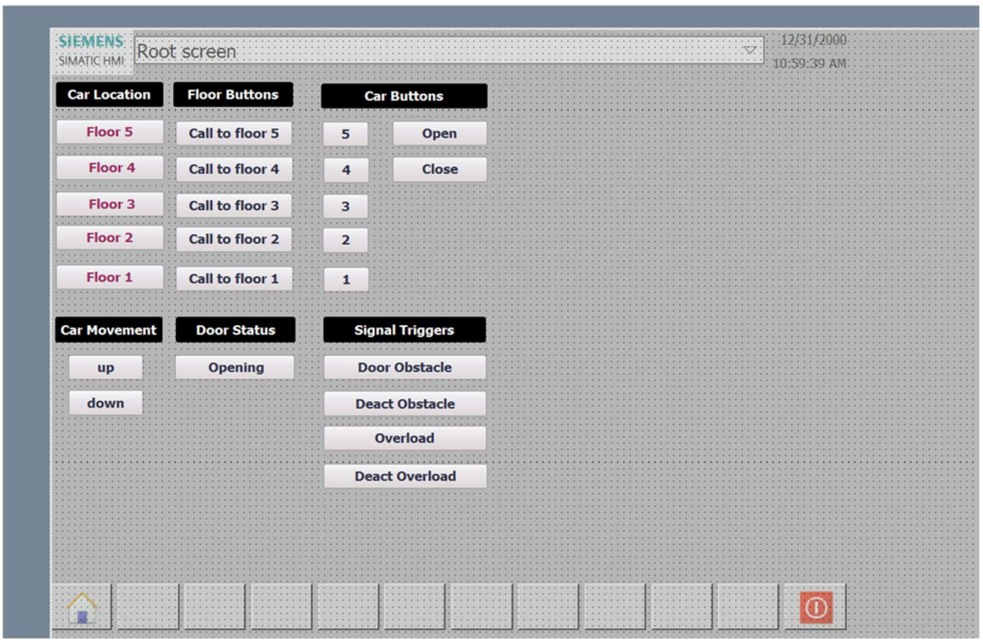

Figure 51. HMI screen in WinCC

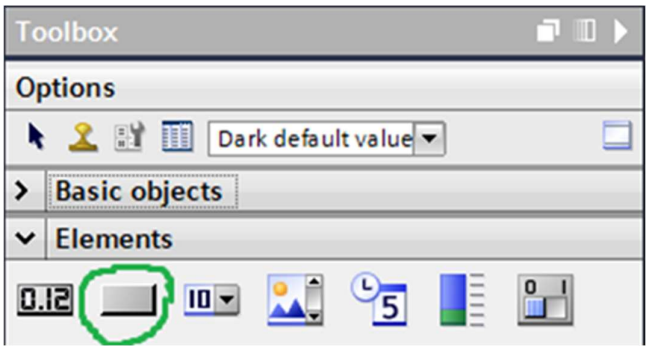

Figure 52. Button element in the HMI toolbox

Signal operations in simulation

- ''Car Location'' signals which are ''Floor 5'', ''Floor 4', etc will turn green if the elevator is at floor 5, 4 and likewise.
- "Floor Buttons'' signals simulate call buttons from a floor, ''Car Buttons'' simulate call buttons inside the elevator and open, close buttons. These will turn green when pressed in the simulation.
- ''Car Movement'' signals indicate the direction of the elevator. Therefore, ''Up'' and ''Down'' will turn green in the simulation corresponding to the movement.
- ''Door Status'' signal will turn green in the simulation if the door is opening or not.
- "Signal Triggers'' are pressable buttons to simulate signals from the obstacle and the overload sensors.

Setting functions of elements

- Activate call buttons when pressed: When pressed, ''Floor Buttons'' and ''Car Buttons'' are set to activate inputs in the logic program. This is done using the ''SetBit'' function in ''Press'', ''Events'' tab in the element's Properties settings. This also applies to ''Signal Triggers''. For example, in figure 36, button ''call to floor 2'' is set to activate ''floorcall_fromF2'' in the logic program.

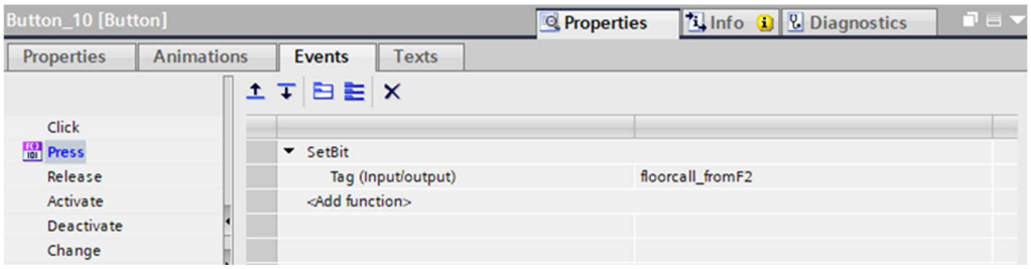

Figure 53. Setting a tag with a ''press'' action in ''Events'' tab

- Activate and deactivate ''Signal Triggers'': Button settings are slightly different for Signal Triggers buttons. Since there are no events to activate and deactivate the overload and obstacle sensors like in real

life situations, buttons are created to do this. In this case, the SetTag function is used to activate and deactivate the signals. Tagged values are ''obstacle'' and ''overload''. The value is set to 1 to activate and to 0 to deactivate. The reason for not using SetBit function is that once a button pressed, the value cannot be reset unless there is a reset logic operation.

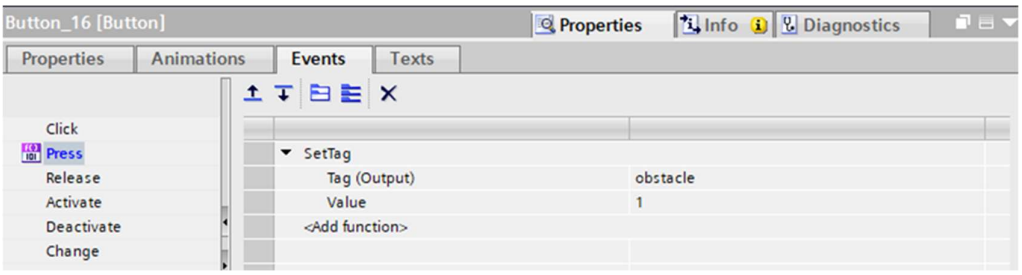

Figure 54. Activate an input using ''SetTag'' to set its value to 1

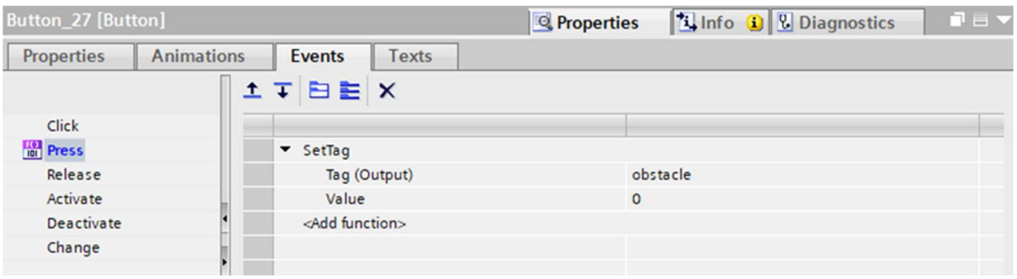

Figure 55. Deactivate an input using ''SetTag'' to set its value to 0

– Change colour when input/output is true: Most buttons/elements are set to turn green when its assigned PLC tag (inputs and outputs in the logic program) is true. This is done in ''Appearance''/''Display'' in the ''Animations'' tab. For ''Deact Obstacle'' and ''Deact Overload'', they are set to be green when signals from the obstacle and overload sensors are false to show their current state. Therefore, the initial color of these signals at the start of the simulation is green.

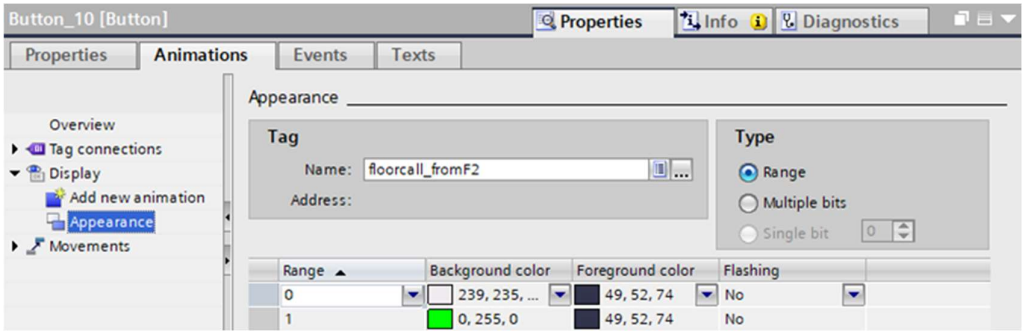

Figure 56. Setting green color in simulation in ''Animation'' tab

5 SIMULATION

This chapter describes the simulation of the previously made logic operations. Since the operations in FBD in some networks are quite large, which made inputs' names hardly visible, only the LAD networks are monitored and added to the report. It is noticeable that for sophisticated operations, LAD networks provide a much more compact view, thus makes it easier to understand the simulation.

5.1 Normal movement

In the simulation screen (figure 57), the elevator is moving up from floor 2 when there is a car call to the third floor.

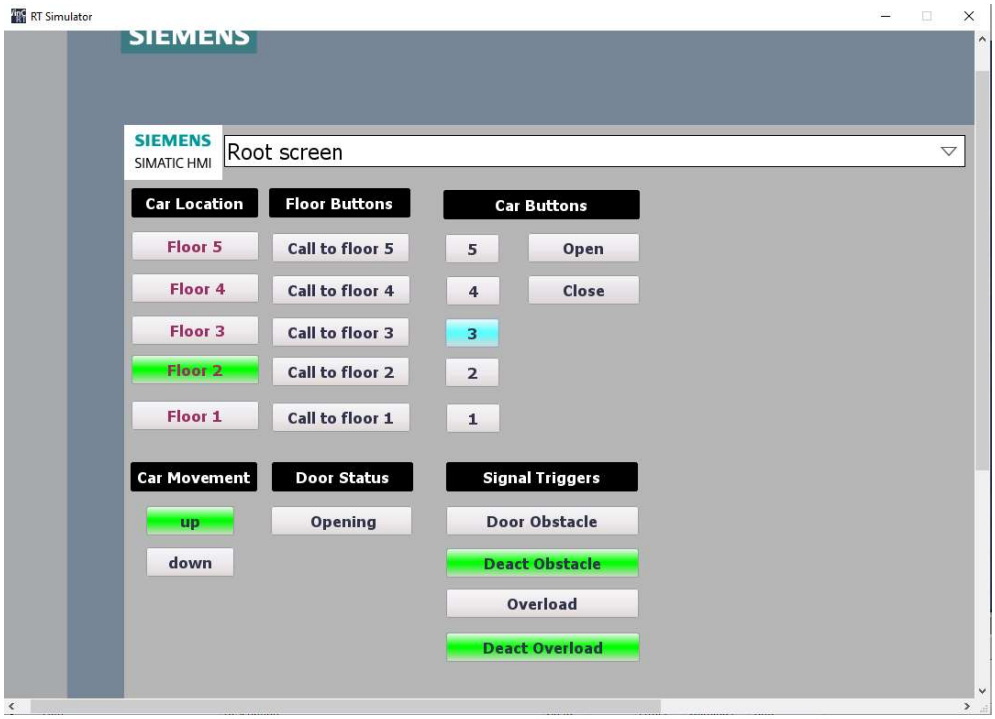

Figure 57. Up movement simulation in HMI screen

In figures 58 and 59, the logic operation for moving the elevator up is indicated in the second rung in LAD and in the second lower block in FBD. Active inputs are ‘’F2_sensor’’, ‘’F3_visit’’ and a normally closed contact of ‘’F1_visit’’, which activate the ‘’motor_up’’ output. The call can also be made by pressing ‘’Call to floor 3’’ button.

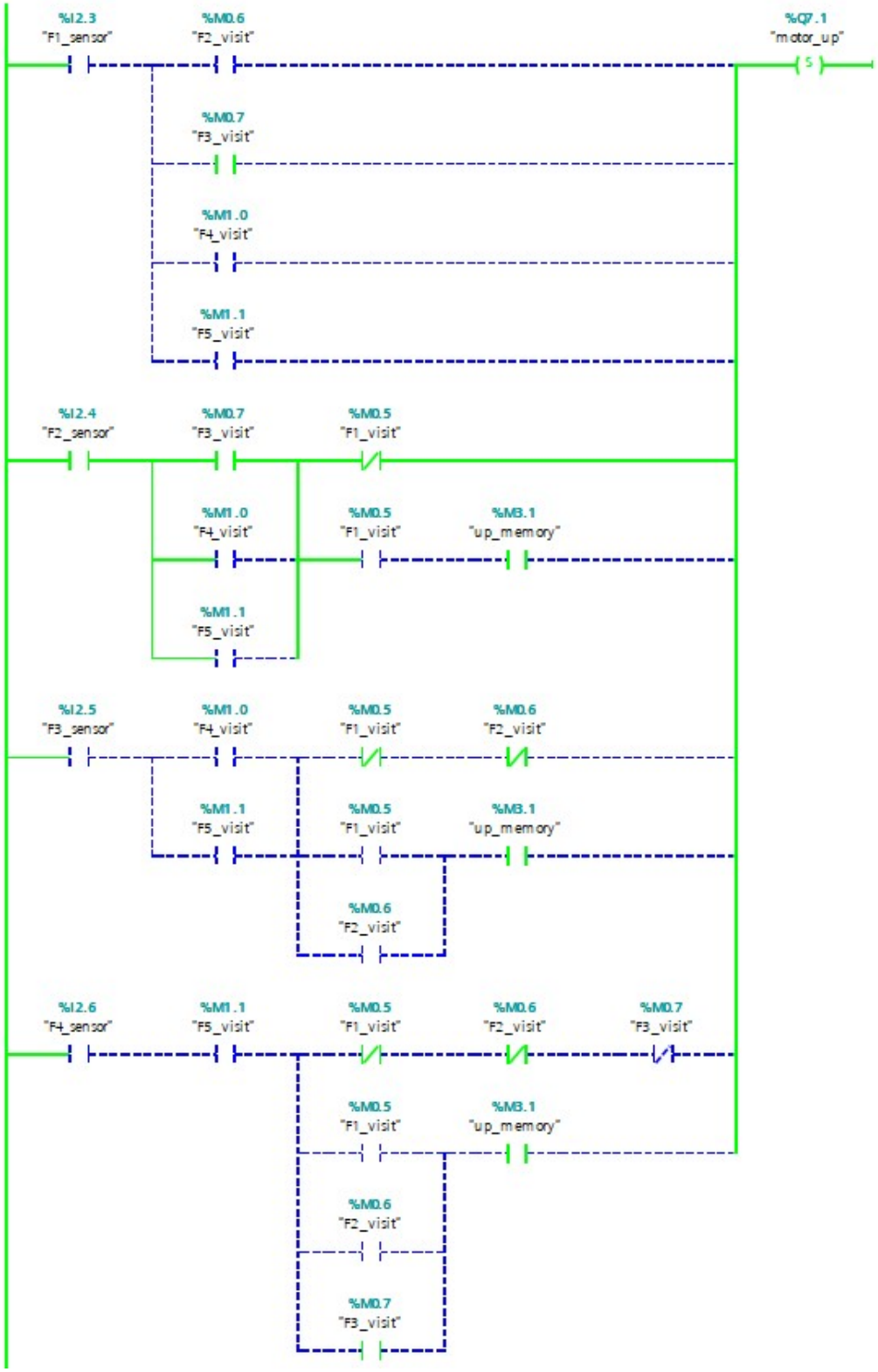

Figure 58. Up movement simulation in LAD

Similarly, a downward movement signal is made when the car is at the second floor and there is a car call to the first one, which is shown in figure 59. In both cases there are no other calls from the upper floors.

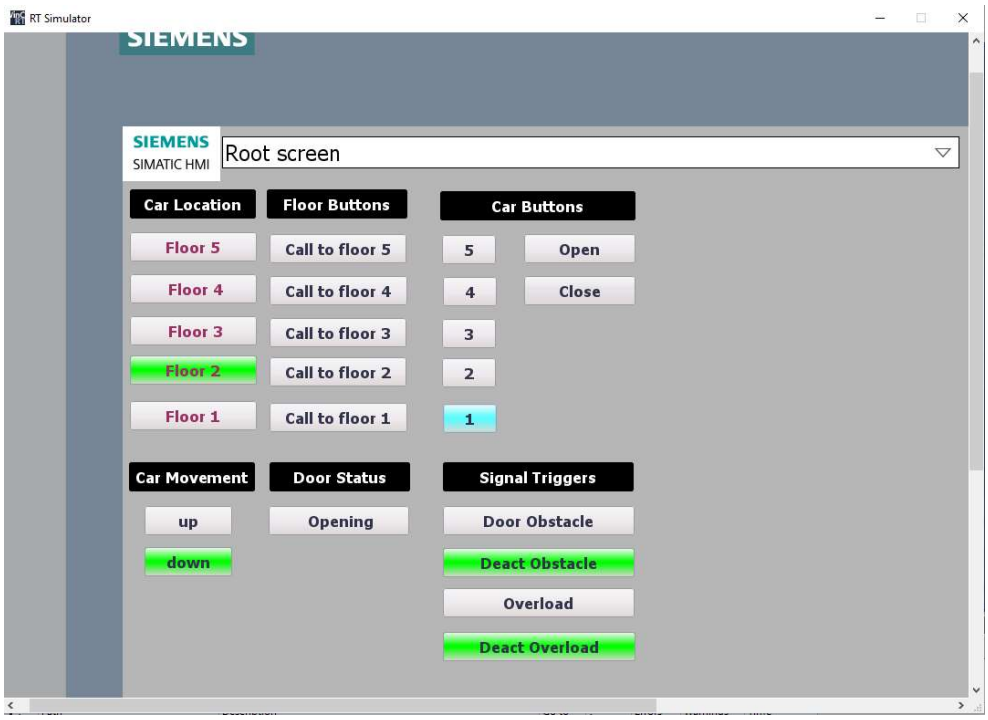

Figure 59. Down movement simulation in HMI screen

5.2 **Prioritized movement**

Figures 60 and 61 represents a scenario where the car is at floor 3 and there are calls from both directions. In this case, it has just moved up from the second floor, which set ''up_memory'' and reset ''down_memory''. Since these two memory inputs are used when the car needs to prioritize one direction over another, ''motor_up'' is set because ''up_memory'' is True and ''motor_down'' cannot be set because ''down_memory'' is False. As a result, the car will move up to the fourth floor before it goes to the second one.

Figure 60. Elevator reached floor 3 and needs to keep moving up

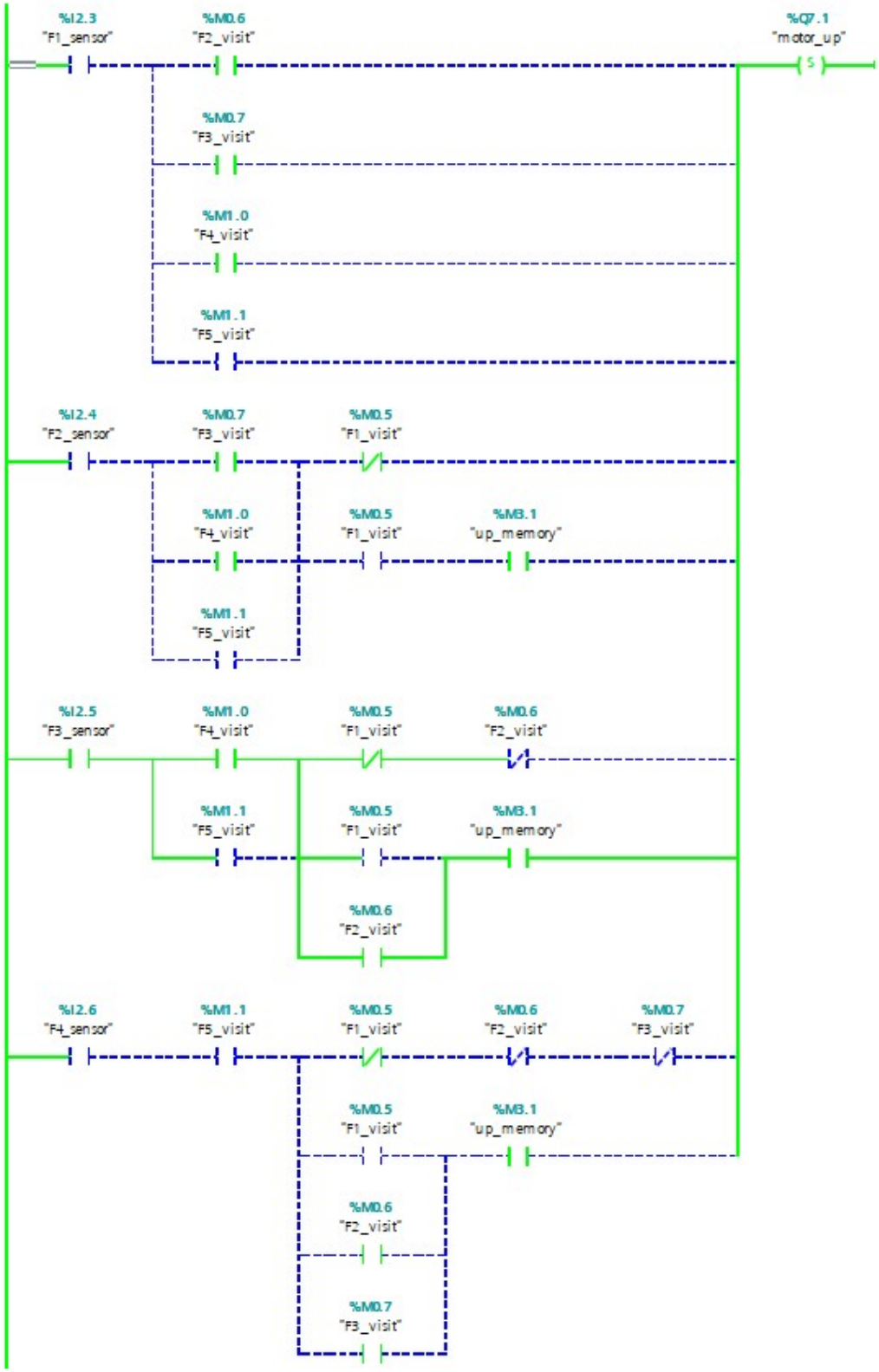

Figure 61. Prioritizing upward movement in LAD

Similarly, the logic operation used for prioritizing the downward movement is shown below.

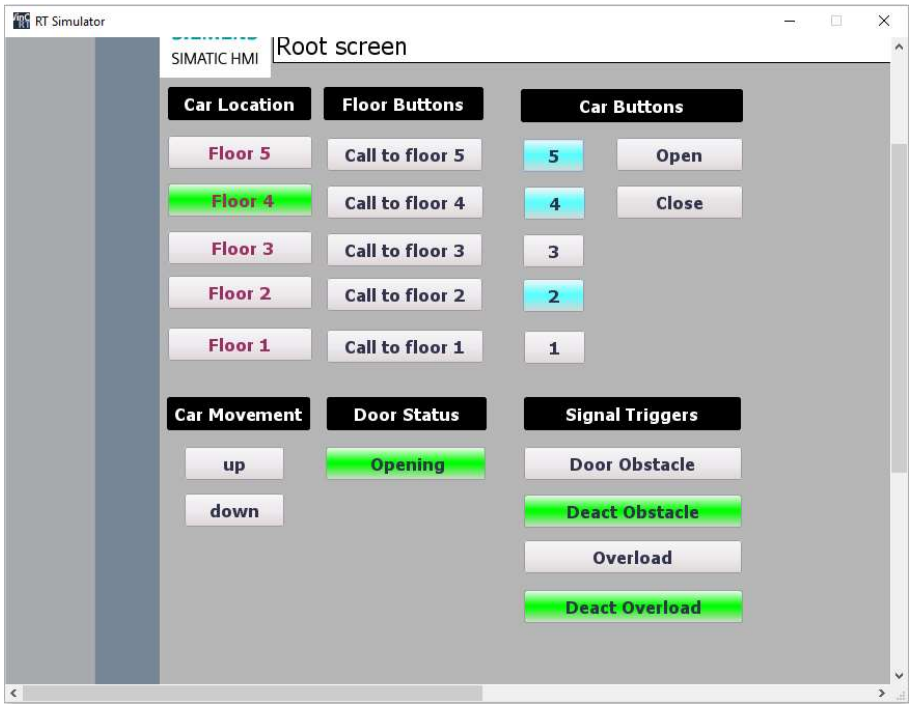

Figure 62. Elevator reached floor 4 and needs to keep moving down

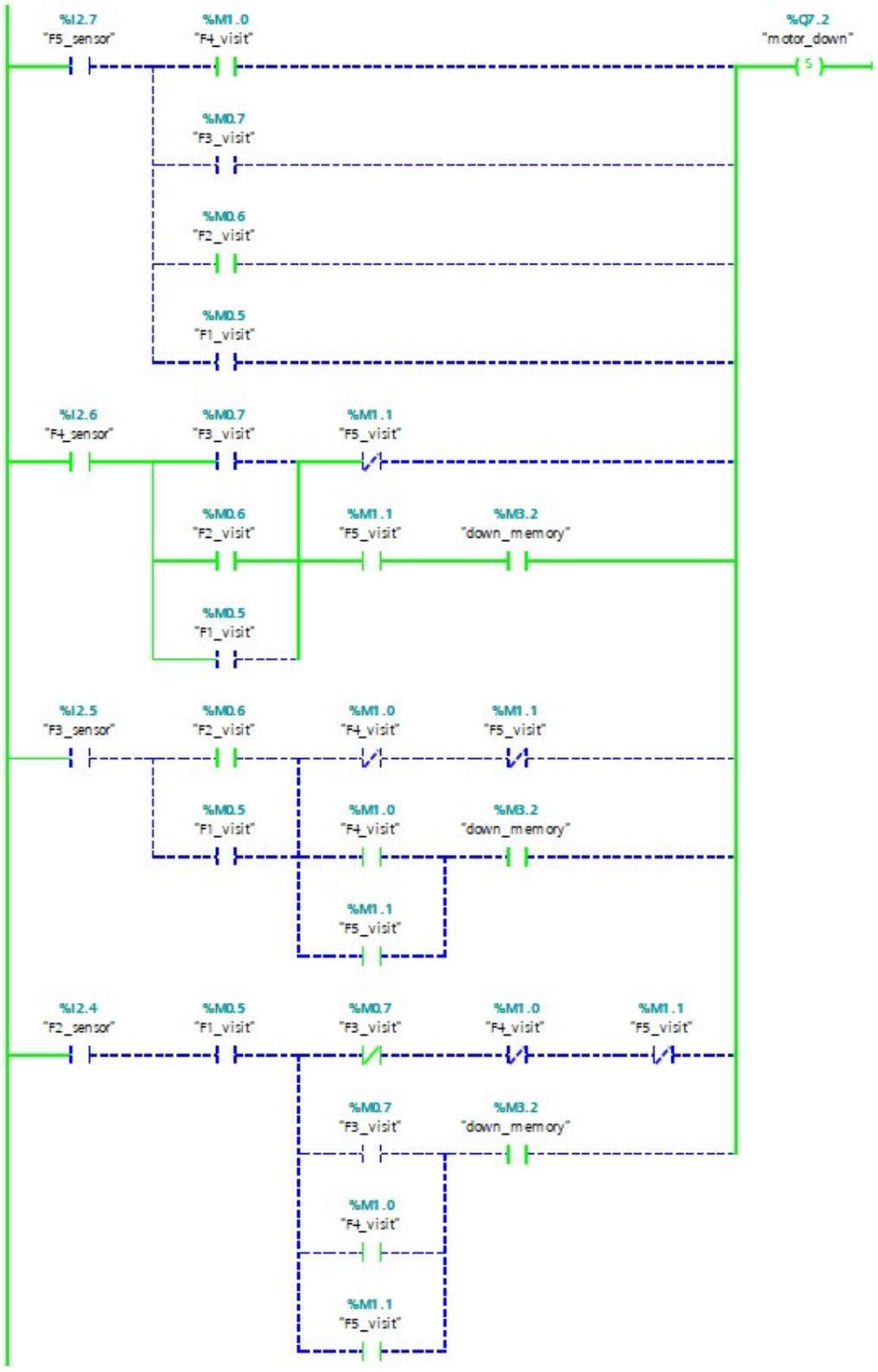

Figure 63. Motor prioritizing downward movement in LAD

5.3 Explaining the reset condition of ''up_memory'' and ''down_memory''

In the simulation process, an error came up because the initial condition for resetting ''up_memory'' and ''down_memory'' was inefficient. Particularly, when the car was set to moving up and down between the fourth floor and the second floor, both inputs are set. This makes the elevator unable to prioritize calls when there are calls from both directions. For example, when the car comes down to floor 2 from floor 3, ''down_memory'' is set. Then after that the car moves up to floor 4, ''up_memory'' is also set. If by that time, there are simultaneous calls to floor 5 and floor 2 then the elevator cannot go up because ''motor_down'' is also set. This is shown in the figures below.

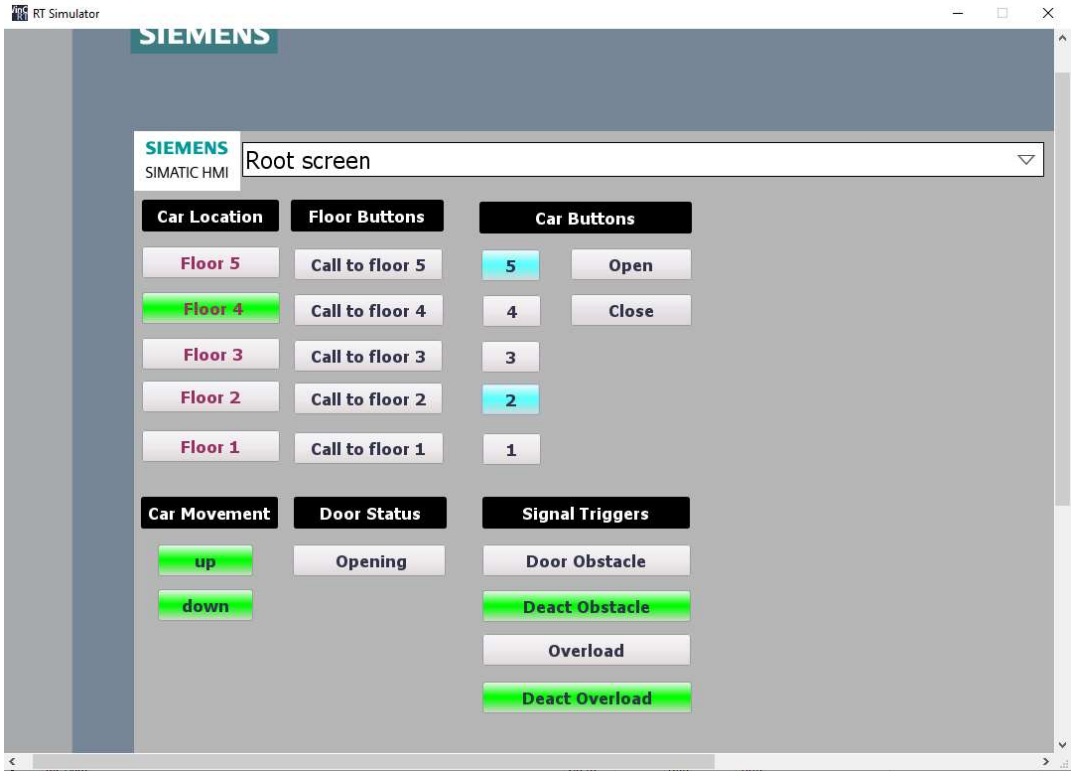

Figure 64. Error shown in HMI Screen where both ''up'' and ''down'' direction signals are active

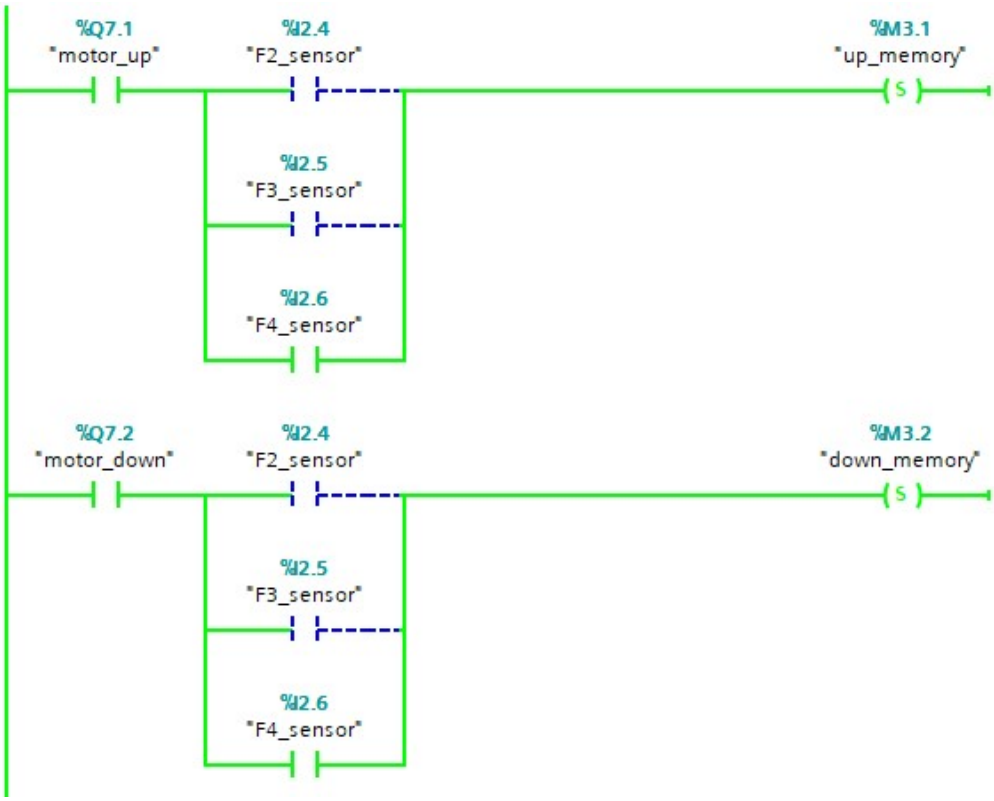

Figure 65. Error when both ‘’up_memory’’ and ‘’down_memory’’ are set in LAD

To fix the problem, another is input is added in each rung to reset ‘’up_memory’’ and ‘’down_memory’’, they are ‘’motor_up’’ and ‘’motor_down’’. This means that whenever that car goes up or down, it will dismiss the assignment of the opposite direction. Initially, “up_memory” and ‘’down_memory’’ were used to reset each other but it resulted in the same outcome. The reason is that if ‘’down_memory’’’is set in the first place then ‘’up_memory’’ can never be set so it cannot reset ‘’down_memory’’ either.

5.4 **Stop motor and open the door when finishing a call**

In the example simulation shown below, when the car reaches the fourth floor, it is stopped by resetting ‘’motor_up’’ and ‘’motor_down’’ and the door is also opened. The open-end signal, ‘’door_openedF4’’ indicates that the door is fully opened and initiates the door closing operation.

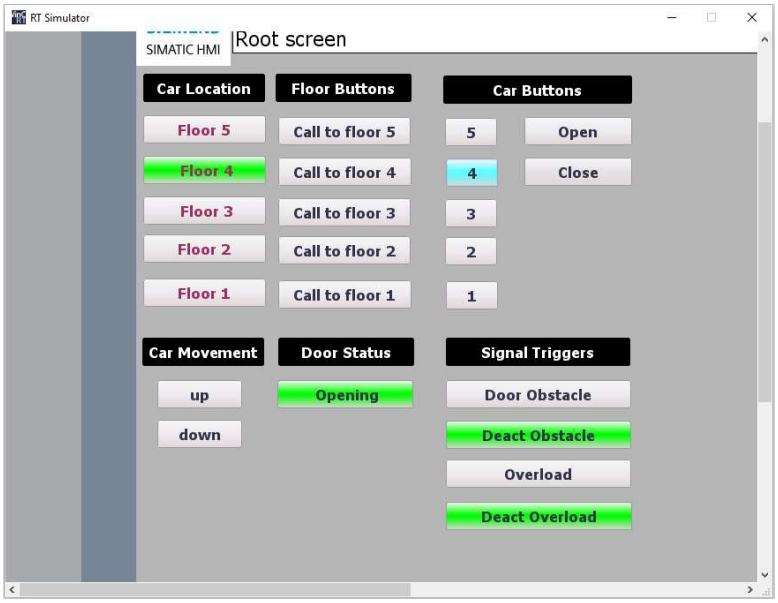

Figure 66. Elevator stops at floor 4 and opens the door in the HMI Screen

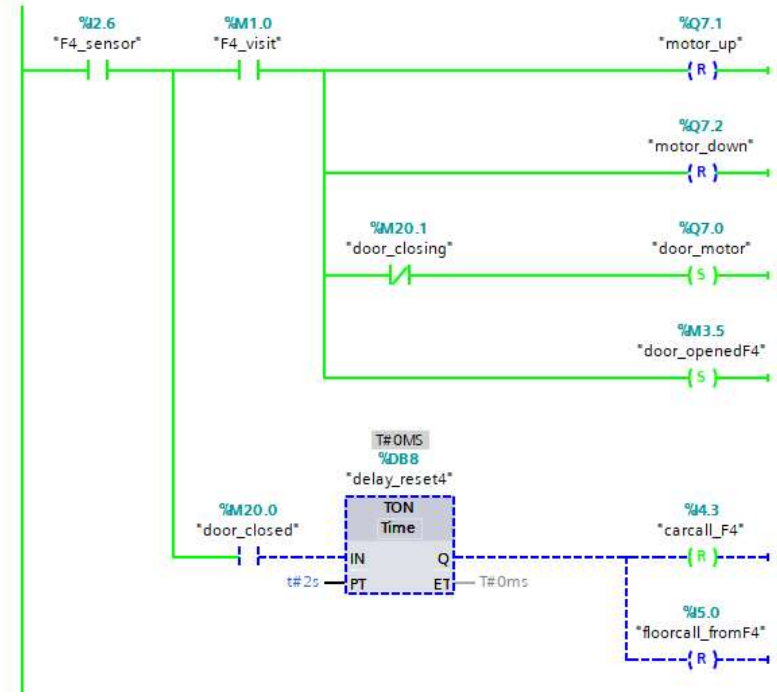

Figure 67. Elevator stops at floor 4 and opens the door in LAD

5.5 **Closing door**

In the below network, when the open-end signal is true, a three second timer is started, followed by the door closing signal (resetting ''door_motor''). Memory input ''door_closing'' is also assigned to keep

''door_motor'' from being set in the previous network so that the door is not opened again. If the open button is pressed and either the overload or the obstacle sensor is active, then this operation is halted.

After another three second timer, input memory ''door_closed'' is assigned, indicating that the door is fully closed. The program then waits for two more seconds, as shown in figure 62 to see if the open button is pressed before resetting all request to the fourth floor.

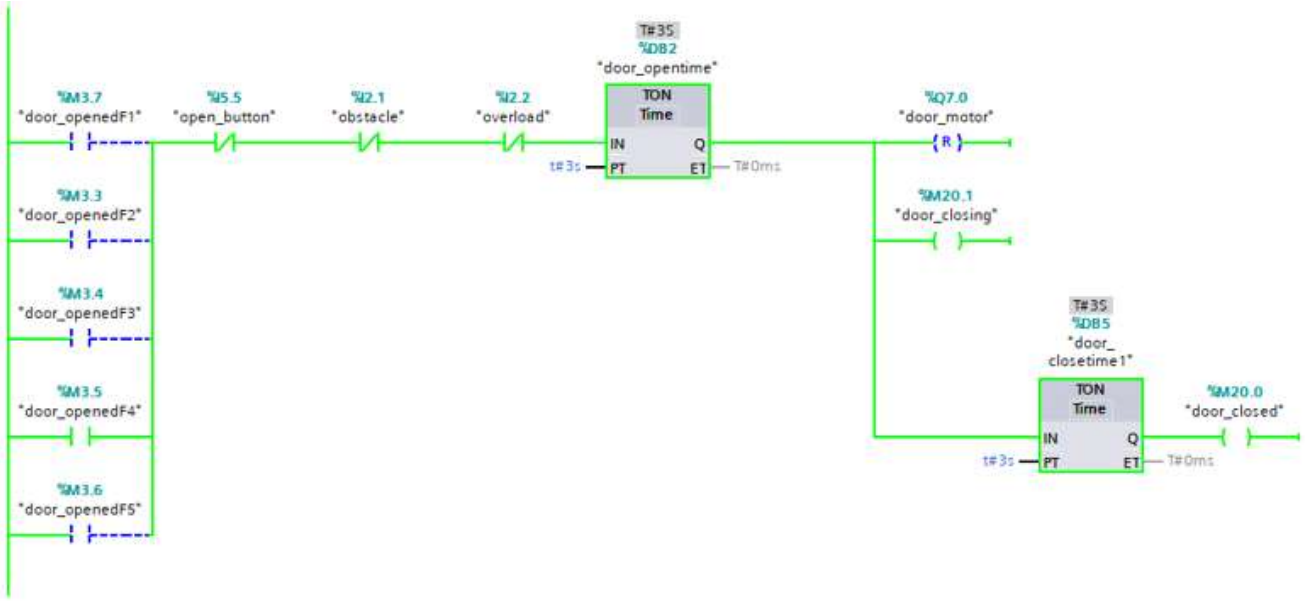

Figure 68. Door closing operation in LAD

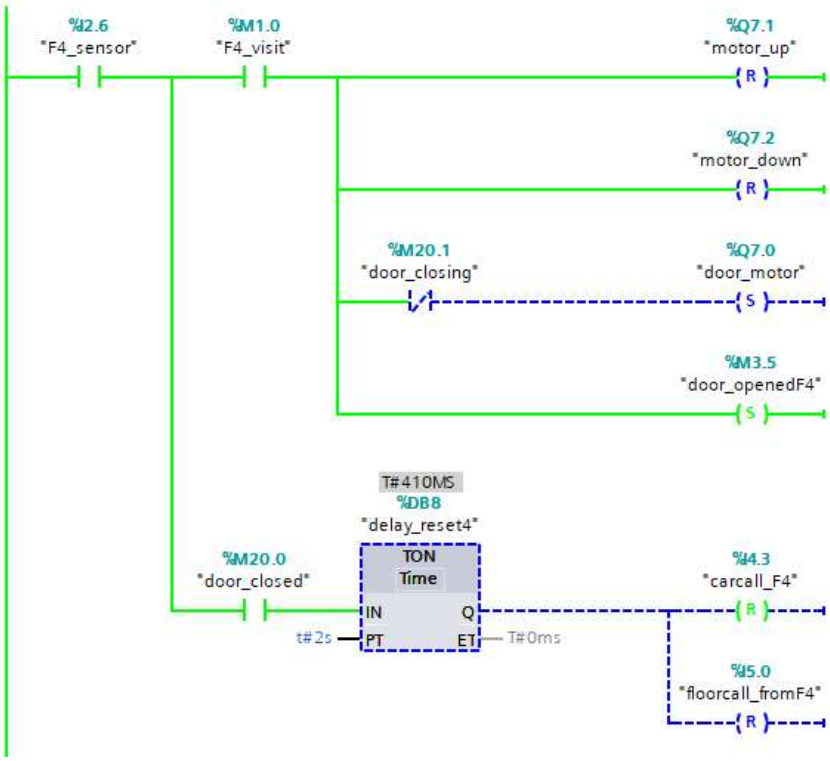

Figure 69. Resetting all calls when the door is fully closed

5.6 **Operation with signals from open button and overload, obstacle sensors**

Figures below shows the logic operation when the open button is pressed, and the obstacle sensor is triggered.

If the door has been opened when the elevator finished a call, then the door closing operation in figure 68 is halted until the sensor is deactivated and the open button is reset after a three second duration.

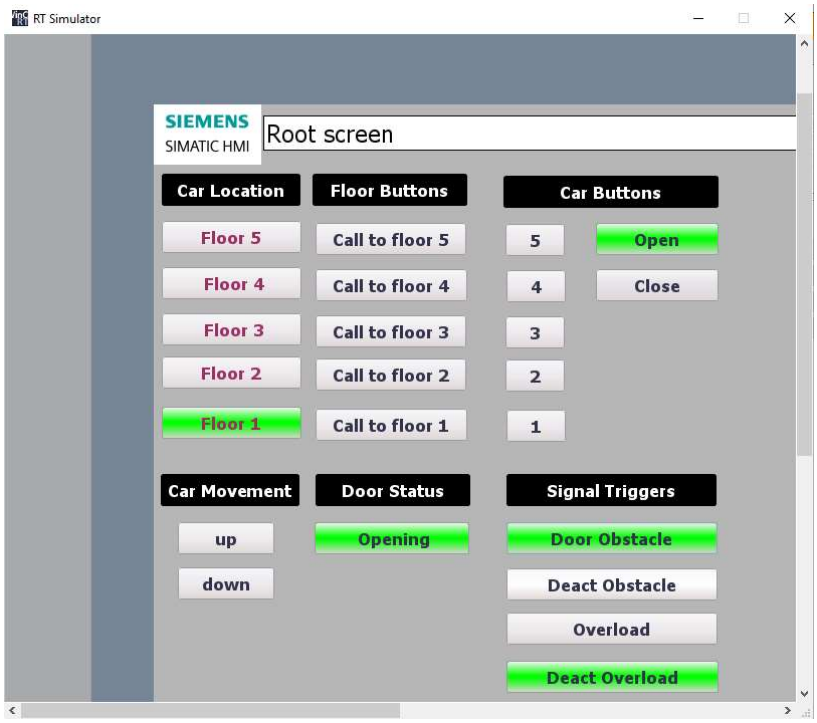

Figure 70. Open button is pressed and obstacle sensor is activated in the HMI screen

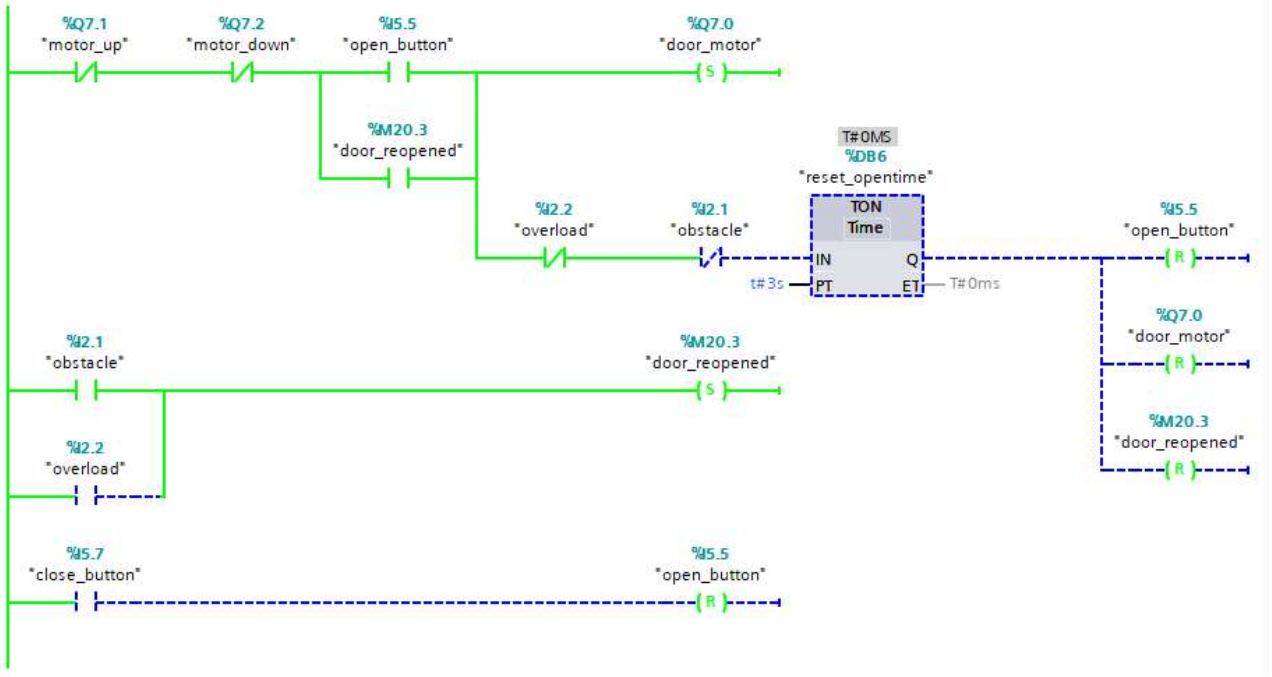

Figure 71. Open button and sensors' operation in LAD

6 CONCLUSION

In summary, the goals of the project were successfully achieved. All planned logic operations were thoroughly programmed and tested. This report contains detailed explanations and guides so that viewers can use it as a tutorial for programming in the ladder logic or the function block diagram.

Throughout the thesis process, the author has gained valuable experiences in PLC programming. The most evident one is that, to build large logic networks as time-efficient and error-free as possible, one needs to deeply understand the fundamental functions and beware of all the outcomes even before the testing step. Otherwise, it would be a time-consuming process even if the functions seem to be simple. In addition, the author finds that programming in the ladder logic is more efficient compared to the function block diagram, since large networks in ladder logic are more compact and easier to monitor.

Finally, considering the program's applicability, it contains all the necessary features for a single elevator system. Therefore, in the author's opinion, with adaptation to different variations of PLCs, the program can potentially be used for testing operations or it can be even adapted for controlling elevators in small apartment buildings.

REFERENCES

Archtoolbox. (n.d). Elevator Types. Retrieved 22 April 2020 from https://www.archtoolbox.com/materials-systems/vertical-circulation/elevatortypes.html

C.Gonzalez. (2015). Engineering Essentials: What Is a Programmable Logic Controller?. Retrieved 25 February 2020 from https://www.machinedesign.com/learning-resources/engineering-essentials/article/21834250/engineering-essentials-what-is-a-programmable-logic-controller

KEB Technology. (n.d). Elevator Regenerative Drives – How They Work. Retrieved 22 April 2020 from https://www.kebamerica.com/blog/elevator-regenerative-drives-how-they-work/

Ladder Logic World. (n.d). Parts of ladder logic. Retrieved 14 February 2020 from https://ladderlogicworld.com/

Mitsubishi Electric. (n.d). Elevators & Escalators. Retrieved 11 February 2020 from https://www.mitsubishielectric.com/elevator/overview/elevators/b_operations04.html

Newark. (n.d). Sensing in Elevator. Retrieved 11 February 2020 from https://www.newark.com/elevator-sensor-applications

Peter. (2017). Ladder Logic Tutorial for Beginners. Retrieved 23 February 2020 from https://www.plcacademy.com/ladder-logic-tutorial/

Using Clock Memory and Timers. Retrieved 20 February from http://www.plccenter.cn/Siemens_Step7/Verwenden_von_Taktmerkern_und_Zeiten.htm