

Lars Bonvik

# Designing and comparing system solutions for a Programmable Logic Controller alternative for educational use

Master's thesis in Produktutvikling og Produksjon

Supervisor: Amund Skavhaug

December 2023





Lars Bonvik

# **Designing and comparing system solutions for a Programmable Logic Controller alternative for educational use**

Master's thesis in Produktutvikling og Produksjon  
Supervisor: Amund Skavhaug  
December 2023

Norwegian University of Science and Technology  
Faculty of Engineering  
Department of Mechanical and Industrial Engineering





---

## Preface

This is the final product of my master's degree in Engineering for Norwegian University of Science and Technology in Trondheim. This is written for robotics and automation group in the course *TPK4960 - Robotics and Automation, Master's thesis*

I want to first thank Eskild Godli, which was a valuable partner for the preceding project assignment written in the autumn semester in 2022. This Master's thesis was delayed until autumn 2023 due to some missed classes during the Covid-19 pandemic. I would also give a special thanks to Amund Skavhaug for his guidance and support for this Master's thesis. I would also thank all of my friends and family which have supported me through all of these years.

- Lars Bonvik

---

## Summary

This is a Master's thesis written by Lars Bonvik in the autumn semester of 2023, and is a part of the final semester of the master program for Mechanical Engineering at the Norwegian University of Science and Technology, NTNU. This thesis studies how to use micro controllers to emulate the functionalities of a Programmable Logic Controller for the Indexed Line factory, manufactured by *Fischertechnik*. The thesis contains both software and hardware solutions, making this possible. The work is based on *Andreas Knudsen Sunds* discoveries in his master's thesis, for the need to improve the assignments in the course TPK4128 Industrial Mechatronics. As well as a project assignment written a year before, by *Lars Bonvik* and *Eskild Godli*.

The preceding project started with the research of how a Programmable Logic Controller work as well as the workings of Fischertechnik's mini-factory provided for the project.

It was researched and found open-source software that could make sure that with an added hardware solution, could emulate a Programmable Logic Controller. The hardware solution was developed and researched through the thesis, and rapid prototypes on breadboards created to test the researched solutions. With a satisfactory solution developed, a prototype Printed Circuit Board (PCB) was designed and produced. The PCB acts as the communication layer between the Raspberry Pi and the "Fischertechnik Indexed Line with two Machining Stations 24V" provided by the supervisor for use in exercises in Industrial Mechatronics.

The main focus of the Master's thesis was to further develop solutions that were found in the project thesis and directly compare these for use in a potential assignment. This led to development of a software for the Arduino Portenta Machine Control with OpenPLC, and Arduino MEGA in addition to the working solution made for the Raspberry Pi. Two different software solutions were tested for the project, OpenPLC and Arduino PLC IDE, where OpenPLC seems like the best choice for a potential assignment. The best micro controller depends on what qualities that are desired, but the Arduino Portenta Machine Control seems to be the best choice for use in an assignment in Industrial Mechatronics.

---

## Sammendrag

Dette er en masteroppgave som er skrevet av Lars Bonvik i høst-semesteret i 2023, og er den del av det siste semesteret på studieprogrammet Produktutvikling og Produksjon på Norges Teknisk-naturvitenskaplige Universitet, NTNU. Denne oppgaven handler om hvordan man kan bruke mikrokontrollere og annen maskinvare til å emulere funksjonalitet av en Programmerbar Logisk Styringsenhet (PLS) for Indexed Line fabrikk som er laget av *Fischertechnik*. Oppgaven handler både om programvare og maskinvare-løsninger som gjør dette mulig. Arbeidet er basert på *Andreas Knudsen Sunds* oppdagelser i sin egen masteroppgave, om at det er nødvendig å oppdatere øving-sopplegget i *TPK4128 Industriell Mekatronikk*. Denne oppgaven fortsetter arbeidet som ble gjort under prosjektoppgaven til *Lars Bonvik* og *Eskild Godli*.

Det forgående prosjektet startet med å undersøke hvordan en PLS fungerer, i tillegg til gjøre seg kjent med Fischertechniks mini-fabrikk som skulle bli brukt til prosjektet.

Det ble funnet open-source programvare som ville med litt skreddersydd maskinvare kunne fungere til å emulere en PLS. Maskinvareløsningen ble utviklet of testet i løpet av oppgaven, og prototyper ble lagd på breadboards for å lage og teste løsninger. Med en tilfredstillende løsning ble også dette videreutviklet og satt sammen til et fungerende kretskort. Kretskortet fungerer som et mellomledd for å omforme strømmen fra mikrokontrolleren og ”Fischertechnik Indexed Line with two Machining Stations 24V” som ble utdelt sammen med oppgaven fra veileder.

Hovedfokuset for denne oppgaven var å videreutvikle løsningene fra prosjektoppgaven og sammenligne disse for bruk i en potensiell øving i faget. Dette førte til programvare utvikling for OpenPLC til bruk med Arduino Portenta Machine Control og Arduino MEGA, i tillegg til den allerede fungerende løsningen for Raspberry Pi. To forskjellige programvarer ble også prøvd i prosjektet. Disse to programvarene var OpenPLC og Arduino PLC IDE, hvor OpenPLC ser ut til å være det beste valget for et potensielt øvingsopplegg. Den beste mikrokontrolleren varierer basert på hvilke kvaliteter man vektlegger, men Arduino Portenta Machine Control H7 ser ut til å være det beste valget for en øving i Industriell Mekatronikk.

---

## Acronyms

**I/O** - Input and Output

**LD** - Ladder Diagram

**OPC UA** - Open Platform Communications Unified Architecture

**OpenCV** - Open Source Computer Vision Library

**OS** - Operating System

**PLC** - Programmable Logic Controller

**ROS** - Robot Operating System

**RPi** - Raspberry Pi

**RT** - Real Time

**SFC** - Sequential Function Chart

**IDE** - Integrated development environment

**UI** - User Interface

---

# Table of Contents

<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and Project Description . . . . .	1
1.2 Previous Work . . . . .	2
1.3 Objective . . . . .	3
1.4 Structure of thesis . . . . .	3
<b>2 Background theory</b>	<b>5</b>
2.1 Programmable Logic Controller (PLC) . . . . .	5
2.1.1 Control systems and controllers . . . . .	5
2.1.2 The functions of PLCs . . . . .	6
2.1.3 The Hardware and Architecture of PLCs . . . . .	7
2.1.4 PLC Inputs and Outputs . . . . .	8
2.1.5 PLC Manufacturers and Models . . . . .	9
2.2 Scheduling algorithm for Ubuntu Linux . . . . .	9
2.2.1 Real-time kernel for Linux . . . . .	10
2.3 Raspberry Pi . . . . .	10
2.4 Arduino UNO . . . . .	11
2.5 Arduino MEGA . . . . .	12
2.6 Arduino Pro . . . . .	13
2.7 OpenPLC . . . . .	13
2.8 Arduino PLC IDE . . . . .	14
<b>3 First time setup of the devices</b>	<b>15</b>
3.1 First time setup of the Arduino PMC . . . . .	15
3.1.1 License Activation with Product key . . . . .	17
3.2 Setup of real-time kernel on Raspberry Pi . . . . .	17
3.3 Installing OpenPLC runtime on the Raspberry Pi . . . . .	22

---

<b>4</b>	<b>Technical specifications</b>	<b>24</b>
4.1	The interface circuitry . . . . .	24
4.2	Specifications of the mini-factory . . . . .	27
4.3	PLC standards . . . . .	28
4.4	Running the factory with Arduino PMC . . . . .	29
<b>5</b>	<b>Connecting the factory and the micro controllers</b>	<b>30</b>
5.1	The softwares . . . . .	30
5.2	Arduino Portenta Machine Control H7 . . . . .	30
5.2.1	PMC Hardware . . . . .	30
5.3	Arduinio PMC with OpenPLC software . . . . .	31
5.3.1	Arduino PMC analog pins . . . . .	32
5.3.2	Analog input . . . . .	32
5.3.3	Analog output . . . . .	33
5.3.4	Software driver for the programmable I/O . . . . .	33
5.4	Arduino PMC with Arduino PLC IDE . . . . .	36
5.5	Arduino MEGA with OpenPLC . . . . .	37
5.5.1	Testing the Arduinio MEGA . . . . .	37
5.5.2	Testing and fixing inputs . . . . .	38
5.6	Updating the circuit board . . . . .	38
5.6.1	Adding a pull-down resistor . . . . .	40
5.6.2	Changing the resistances . . . . .	41
5.7	Connecting micro-controller to the factory . . . . .	43
5.7.1	Connecting wires directly . . . . .	44
5.7.2	Hardware shields . . . . .	44
5.7.3	Hardware shield on Arduino PMC . . . . .	44
5.7.4	Hardware shield Arduino MEGA . . . . .	45
5.7.5	Hardware shield Arduino UNO . . . . .	46
<b>6</b>	<b>Software and Simulation</b>	<b>47</b>
6.1	Connecting to Raspberry Pi . . . . .	47
6.2	Programming and language . . . . .	48

---



---

6.3	Programming in Arduino PLC IDE . . . . .	48
6.4	Programming in OpenPLC . . . . .	49
6.5	Raspberry PI specific changes; PCB . . . . .	50
6.6	Connecting Arduino PMC to OpenPLC . . . . .	50
6.7	Compiling and running code in Arduino PLC IDE . . . . .	53
<b>7</b>	<b>PLC code for the factory</b>	<b>54</b>
7.1	Ladder Diagram code . . . . .	54
7.1.1	LD Code part 1 . . . . .	54
7.1.2	LD Code part 2 . . . . .	56
7.1.3	LD Code part 3 . . . . .	58
7.1.4	LD Code part 4 . . . . .	60
7.1.5	Discussing the Ladder Diagram Code . . . . .	61
7.2	SFC - Sequential Function Chart . . . . .	62
7.3	Arduino PLC IDE: SFC . . . . .	62
7.3.1	SFC section 1 . . . . .	63
7.3.2	SFC section 2 . . . . .	66
7.3.3	SFC section 3 . . . . .	68
7.3.4	Discussing the SFC code . . . . .	70
<b>8</b>	<b>Discussion</b>	<b>72</b>
8.1	Use of mini-factory in Industrial Mechatronics . . . . .	72
8.2	Raspberry PI . . . . .	72
8.2.1	RPi Advantages . . . . .	72
8.2.2	RPi Disadvantages . . . . .	73
8.3	Arduino PMC . . . . .	74
8.3.1	PMC Advantages . . . . .	74
8.3.2	PMC Disadvantages . . . . .	75
8.4	Arduino UNO . . . . .	76
8.4.1	UNO Advantages . . . . .	76
8.4.2	UNO Disadvantages . . . . .	77
8.5	Arduino MEGA . . . . .	77

---

---

8.5.1	MEGA Advantages . . . . .	78
8.5.2	MEGA Disadvantages . . . . .	78
8.6	Using other devices . . . . .	79
8.7	Hardware choice . . . . .	80
8.8	PCB . . . . .	81
8.9	Software choice: OpenPLC vs Arduino PLC IDE . . . . .	82
8.9.1	User Interface and programming . . . . .	82
8.9.2	Reliability . . . . .	83
8.9.3	Final verdict . . . . .	83
<b>9</b>	<b>Conclusion</b>	<b>84</b>
9.1	Further work and possible expansions . . . . .	84
9.1.1	PCB design . . . . .	84
9.1.2	Machine learning . . . . .	85
9.1.3	Robot arm and ROS2 . . . . .	85
9.1.4	Developing the UNO system . . . . .	85
	<b>Bibliography</b>	<b>86</b>
	<b>Appendix</b>	<b>88</b>
<b>A</b>	<b>PLC architecture from IEC 61131-2</b>	<b>89</b>
<b>B</b>	<b>Eduroam Guide</b>	<b>90</b>
<b>C</b>	<b>Milling, Drilling or not code Example</b>	<b>92</b>
<b>D</b>	<b>Attachments</b>	<b>94</b>
D.1	Attached files: . . . . .	94
D.2	Hardware: . . . . .	94
<b>E</b>	<b>Extended Raspberry Pi OpenPLC table</b>	<b>95</b>
 <b>List of Figures</b>		
1	The Fischertechnik factory [2] . . . . .	2

---

---

2	Example of relay control system . . . . .	5
3	Simple illustration of a PLC . . . . .	7
4	The general architecture of PLCs . . . . .	8
5	Architecture of PLC from IEC 61131 . . . . .	8
6	Siemens Simatic S7-1500 PLC . . . . .	9
7	Raspberry Pi . . . . .	11
8	Arduino UNO . . . . .	12
9	Arduino MEGA . . . . .	12
10	Arduino PMC . . . . .	13
11	IDE download . . . . .	15
12	IDE download . . . . .	16
13	Error message . . . . .	16
14	Modbus connection . . . . .	17
15	Fully preemptible . . . . .	21
16	The PLC architecture map . . . . .	25
17	Rapid interface circuit . . . . .	25
18	Old circuits . . . . .	26
19	Digital outputs for direct current table . . . . .	28
20	Arduino PMC . . . . .	31
21	The ladder logic lines for an analog input of Sensor5 . . . . .	32
22	The ladder logic lines for the analog outputs of the milling and drilling station . . . . .	33
23	Init digital programmables . . . . .	34
24	IO setup OpenPLC . . . . .	35
25	Input updates . . . . .	36
26	Output updates . . . . .	37
27	The breadboard circuit used for testing the input circuit . . . . .	39
28	Pulldown resistor . . . . .	40
29	PCBv5 Schematic . . . . .	43
30	HW shield PMC . . . . .	45
31	MEGA hardware shield . . . . .	45

---

32	Factory from top . . . . .	48
33	Resource tree . . . . .	49
34	PLC code variables . . . . .	50
35	Complete LD code . . . . .	51
36	IO config Arduino PMC . . . . .	52
37	Simple test of Ladder Diagram. . . . .	54
38	Part 1 birdview . . . . .	55
39	Ladder logic part 1 . . . . .	55
40	LD2 section1 . . . . .	56
41	LD2 section2 . . . . .	56
42	Part 2 in birdview . . . . .	57
43	Ladder diagram part 2 . . . . .	57
44	Ladder diagram alternate layout . . . . .	57
45	LD2 section3 . . . . .	58
46	Part 3 of factory in birdview . . . . .	59
47	Ladder diagram part 3 . . . . .	59
48	LD2 section4 . . . . .	60
49	LD2 section5 . . . . .	60
50	The Factory in bird view, showing part 4. . . . .	61
51	The fourth part of the Ladder Diagram. . . . .	61
52	LD2 section6 . . . . .	61
53	SFC code part 1 . . . . .	63
54	SFC code part 2 . . . . .	64
55	Arduino IDE SFC section 1 . . . . .	65
56	Arduino IDE SFC step 0 . . . . .	65
57	SFC code part 3 . . . . .	66
58	SFC code part 4 . . . . .	66
59	Arduino IDE SFC section 2 . . . . .	67
60	Arduino IDE SFC step 1 . . . . .	68
61	Arduino IDE SFC step 2 . . . . .	68

---

62	SFC code part 5 . . . . .	69
63	SFC code part 6 . . . . .	69
64	Arduino IDE SFC section 3 . . . . .	70
65	Arduino IDE SFC step 3 . . . . .	70
66	Arduino IDE SFC step 4 . . . . .	71
67	RPi with PCB . . . . .	73
68	Arduino PMC connected . . . . .	75
69	I/O expander . . . . .	76
70	MEGA with PCB . . . . .	78
71	Unoriginal Arduino . . . . .	79
72	Updated PCB . . . . .	82
73	PCB CAD . . . . .	82
74	Typical interface/port diagram of a PLC-system (from IEC 61131-2)[13] . . . . .	89

## List of Tables

1	Datasheet mini-factory . . . . .	27
2	Comparison table . . . . .	80

---

# 1 Introduction

## 1.1 Motivation and Project Description

The goal of this thesis is to make a system which can improve some of the learning material in TPK4128 Industrial Mechatronics. Which is a course that teaches the students more about Industry 4.0, and more specifically mechatronics for industrial production systems.

The description of the course follows:

”The course is on mechatronics for industrial production systems. This includes the implementation, use, and programming of single-board computers, PLC-based and other industrial computer systems. Embedded- and real-time systems, industrial bus systems, interfacing, operating systems and communication protocols for these. Use of C, Linux and TCP/IP on e.g. Raspberry Pi, Python, ROS, virtual machines, computer vision, OPC-UA and selected Industry 4.0 topics will be taught and practiced. Furthermore, sensors, actuators, power supplies, motor drives, pneumatic and hydraulic actuators, aspects of dependability for industrial computer systems, and development methodologies.” [1]

The focus for this thesis is going to be making a system which can be used as a base for PLC assignments. This would be helpful for students to understand and test out some of the PLC functionalities in the curriculum. The work done for this thesis will mainly focus on making systems which are able to run the Fischertechnik Indexed Line factory. A model of this factory can be seen in Figure 1. In addition, there will be developed hardware which can be able to run this factory using different micro controllers and other hardware.

In the preceding project thesis the task was to make the necessary equipment to be able to run the factory using the Raspberry Pi and make it emulate a Programmable Logic Controller (**PLC**) device using different hardware and software solutions. In this master thesis, the focus is going to be developing different systems that will be able to successfully do the same tasks, and then compare the solutions to the one developed in the project thesis. This will later provide the system solutions necessary to develop a fundamental base for future assignments in the course. The systems will also be compared to help decide what system solution would be most beneficial for the course.

Due to the thesis being directly working on things that were developed in the preceding project thesis, some of the parts are also taken from the project due to them still being highly relevant for this thesis. However, these will be clearly marked before each chapter.

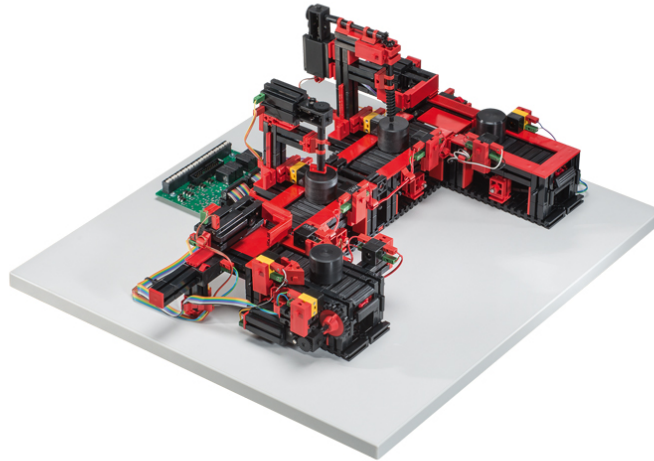


Figure 1: The Fischertechnik factory with two machining stations [2]

## 1.2 Previous Work

Use of the mini-factory from *Fischertechnik* in assignments has already been explored in the Master's thesis by *Andreas Knudssen Sund* [3]. In this project, it was investigated if there was a need for new assignments in *Industrial Mechatronics*. It was also investigated if tasks regarding the Fischertechnik mini-factory could be appropriate to replace the already existing assignments given in the course.

In *Sunds* research [3], it was found that the assignments in the course were ready for an upgrade. A thorough study has been done to see what the impact of introducing the mini-factory as an assignment in *Industrial Mechatronics* would be. The conclusion of the study was that an assignment relating to the mini-factory is most likely going to be beneficial for the learning experience for the students. This would also benefit the students experience and perception of the selected topics, hopefully making the curriculum more interesting to work with.

*Sund* was mainly investigating if this factory was appropriate to use in assignments with a PLC. After getting promising results, it was decided that this project assignment will develop the concepts further with a Raspberry Pi and *OpenPLC*, which makes the institute not dependant on 3rd party manufacturers to complete the assignments made for the course. This would save a lot of headaches regarding licensing and the process to buy PLCs. Some of the PLC vendors also have expensive software systems, so finding a solution which is free or low-cost is beneficial.

The next project regarding this topic was written by the author of this thesis and *Eskild Godli*. This assignment focused on developing parts to be able to use a Raspberry Pi to run the factory, while also developing the bare-bones for use in an actual assignment. The practical parts of the assignment was developed, while also developing custom parts to be able to use the factory and Raspberry Pi reliably in an educational environment.

Another research paper that has been used regarding this project is the master's thesis of *Arnholm and Henriksen* from 2021 [4]. This paper was mainly focusing on the use of Raspberry Pi with a

---

5G hat. Using a Raspberry Pi for a mechatronic system with *Linux* requires a pre-emptive kernel. Therefore, a guide from this paper was used to build such a kernel for the Raspberry Pi.

### 1.3 Objective

The main objective of this Master's degree, is to research and compare different solutions for running an embedded system with micro controllers. Ultimately using this information to update and improve the assignments already used in *TPK4128 Industrial Mechatronics*, mainly by focusing on PLC applications and programming. This will be achieved by testing different types of micro controllers and hardware devices, to further compare them to the Raspberry PI. In addition, hardware developed for the project assignment will be updated and used for this thesis. This preceding project assignment was written by *Lars Bonvik* and *Eskild Godli* in 2022. All of the work done regarding the Raspberry Pi comes from the preceding project thesis. This thesis has mainly focused on using the different types of micro controllers from the manufacturer *Arduino*.

The following points will be the primary objectives for this master's thesis:

- Running the Fischertechnik Indexed Line factory using different hardware solutions
- Make complete working prototypes with different kinds of single board computers
- Simplify the systems as much as possible
- Compare the different devices, to decide what is the best for a PLC assignment in Industrial Mechatronics

The secondary objectives:

- Look at possible expansions for incorporating more parts of the curriculum.
- Develop some solutions to further develop the assignments in *TPK4128 Industrial Mechatronics*

### 1.4 Structure of thesis

**Section 2** introduces the theory and key software used in this thesis. It begins with the presentation of what a Programmable Logic Controller is, and continues with how it works. In addition this chapter will present the different micro controllers and IDEs used in this thesis.

**Section 3** presents how the Raspberry Pi used in this thesis is set up to work in the desired way. In addition, it features how to set up the Arduino PMC with Arduino PLC IDE.

**Section 4** details the functionality that is needed and desired to achieve the objectives set for the thesis. Further it continues with the tests of the mini-factory, and the discoveries of how it functions.

**Section 5** goes through the development of the circuitry needed to run the mini-factory with the Raspberry Pi.



---

**Section 6** contains the softwares used to develop the prototype and goes through the development of the circuitry and related software. It also contains the start of the simulation development for the mini-factory.

**Section 7** contains PLC code developed in the preceding project assignment and the current Master's thesis.

**Section 8** is discussing the overall results and difficulties. Further works is also mentioned.

**Section 9** contains the conclusion of the Master's thesis.

**Appendix A** a figure showing a more advanced architecture representation of a PLC from the PLC standard IEC 61131-2.

**Appendix B** a guide to connect to the Eduroam network with an Raspberry Pi.

**Appendix C** presents the test program written to control the machining stations of the mini-factory.

**Appendix D** a listing of the attached files.

**Appendix E** is an overview of the different pins on the Raspberry Pi.

---

## 2 Background theory

This chapter includes the background theory needed to understand the work done in the rest of this project thesis. Most of this is taken from the project assignment, because the information is still highly relevant for this thesis. The specific sections that has been taken directly from the preceding project assignment is:

- Section 2.1
- Section 2.2.1
- Section 2.3
- Section 2.7

The rest of the subsections in Section 2 has been written specifically for this thesis.

### 2.1 Programmable Logic Controller (PLC)

”A programmable logic controller (**PLC**) is a type of device extensively used for different automation applications within industrial processes and manufacturing” [5]. As it’s name implies, it is a form of controller. This section will give an overview of controllers in general, as well as presenting the function, hardware and architecture of PLCs.

#### 2.1.1 Control systems and controllers

A controller or control system ”might be required to control a sequence of events, maintain some variable constant, or follow some prescribed change” [5]. They are used to automate and streamline tasks that were done manually by people, to drive cost down, and make a safer workplace by automating hazardous tasks.

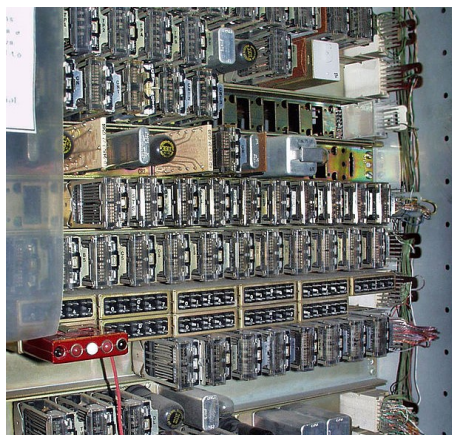


Figure 2: Example of a relay control system used in the Number Five Crossbar Switching System. This unit is in the museum of communication in Seattle. [6]

A popular way to automate tasks before the time of the PLCs, were the use of relay systems. An example of such a system can be seen in Figure 2.

---

Relays are magnetic switches that is switched on and off depending on the voltage of an input signal. This was an ideal system to automate tasks requiring high precision and tight time constraints. They excel at simple tasks, but are unable to do more complex tasks. However there are some significant problems with relay switches. One of them is that tasks are hard to modify. A small change in function might need a complete rewiring of the whole system. Due to all of these physical connections, relay systems also requires a lot of space [5], [7], [8].

Microprocessor control systems are a much more modern alternative to the PLC system. Instead of needing to hardwire the control system for each situation, it's possible to simply reprogram the microprocessor for the specific constraints and functions of a task. This type of control makes the system a lot more flexible than relay switches. Microprocessors are also cheap and space efficient compared to alternative solutions, which makes these systems a preferable way to automate industrial tasks. This type of control also allows for feedback from the system it is connected to, for example for maintenance purposes.

PLCs are a specific form of microprocessor controllers, made to standardise microcontroller systems with simple and robust programming languages. Mainly for use in the industry. One of the standardised languages used for PLCs are called Ladder Diagram, this language was developed to be used by people who had originally wired relay systems. This way it was simple for the programmers to adapt to PLC systems instead, since it is made to be similar to relay system schematics.

In 1969 the first PLC was developed, and it has been the primary solution for industrial automation since the 1970s. They have evolved from self-contained units with few digital I/O, to modular units with the possibilities to expand the I/O. They are able to use analog I/O as well.

Since it is developed new programming languages specific for the PLC's, the need for international standardisation were large, because of the importance of the industrial applications they are used for.

This have led to the creation of many standards, such as the most influential ones from the 90s and early 2000s, IEC 1131 and IEC 61499 respectively. The IEC 1131 standard was later renamed to IEC 61131, and got new extensions. This standard has now ten parts, but started with three parts released in 1992 and 1993. The rest of the parts has been released sporadically since then. Multiple parts were released in the year of 2000, and the newest part was released in 2019 [9], [5], [10], [11], [12], [7].

### **2.1.2 The functions of PLCs**

As said before, a PLC is a complete system with a microprocessor and I/O designed for use in the industry. It uses simple languages, which makes them easy to program for engineers or operators with little to no experience with programming. In figure Figure 3 it is added a simple illustration of how a PLC functions. The PLC get some inputs from sensors, the program interprets them, and some output signals are set to run some motors that for instance runs a conveyor belt.

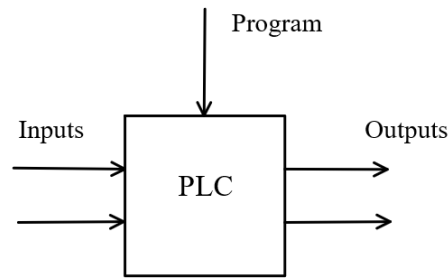


Figure 3: Simple illustration of a PLC

The PLCs are developed for use in the industry. Some of the tasks they are developed for includes, automating tasks for manufacturing, industrial processes, machining automated assembly and packaging. These kinds of tasks are important to not be interrupted or stopped unexpectedly. This scenario has the potential to create hazardous or dangerous situations, or the manufacturer can lose significant monetary value. These are some of the reasons PLCs are built to complete task in harsh environments, and are designed to run as long as possible without failure.

For some automation tasks the added robustness might not be necessary. For example a washing machine for home use might only need a microprocessor controller without the added robustness a PLC provides. Autopilots for airplanes needs more computational power than a PLC provides, to solve complex mathematics and high speed operations. Therefore, a normal computer is more beneficial to run autopilots. In the next subsection the hardware and architecture of the PLC will be looked into [5], [11] [8].

### 2.1.3 The Hardware and Architecture of PLCs

The functional parts of a typical PLC is shown inside the box in Figure 4.

- Where the **Processor** part is the microprocessor(s) that does the arithmetic's and the execution of the application program functions.
- The **Program and data memory** is storing the application program and the states/variables the application program is using.
- **Communications interface** is providing a function to communicate with third-party devices such as PLC's from other manufacturers and computers.
- The **Power supply** is supplying the necessary power to the different parts of the PLC.
- The I/O interfaces, **Input interface** and **Output interface**, is interfacing with the input and output devices respectively.

The I/O interfaces will be explored a bit further. Figure 5 (a) shows a similar architecture diagram as Figure 4 from the IEC 61131-1 standard, and (b) shows one more advanced version from IEC 61131-2 standard. See chapter 6 **Functional requirements** in IEC 61131-2 standard for in-depth specifications of the functional requirements of a PLC's hardware and architecture [11], [13], [5], [7], [8].

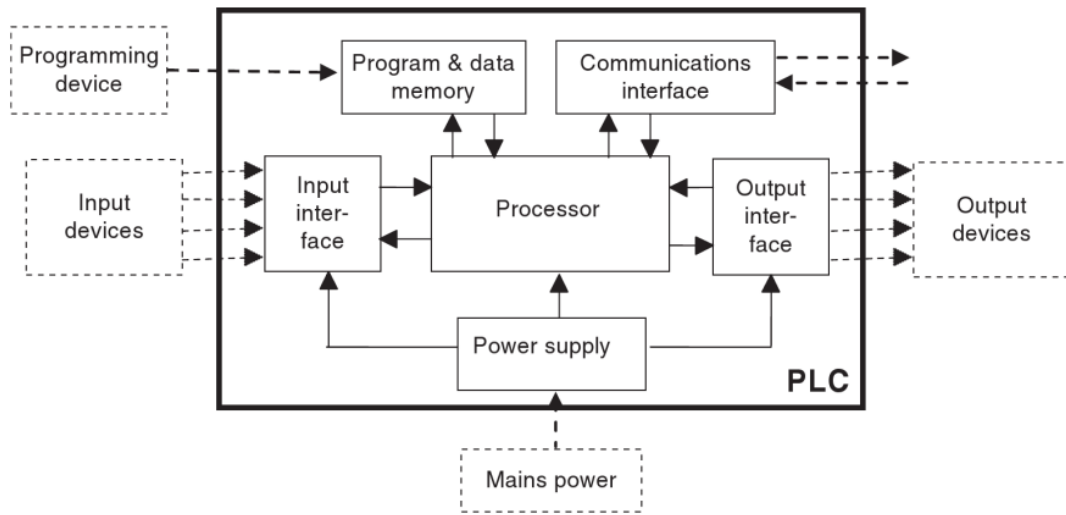


Figure 4: The general architecture of PLCs, [5]

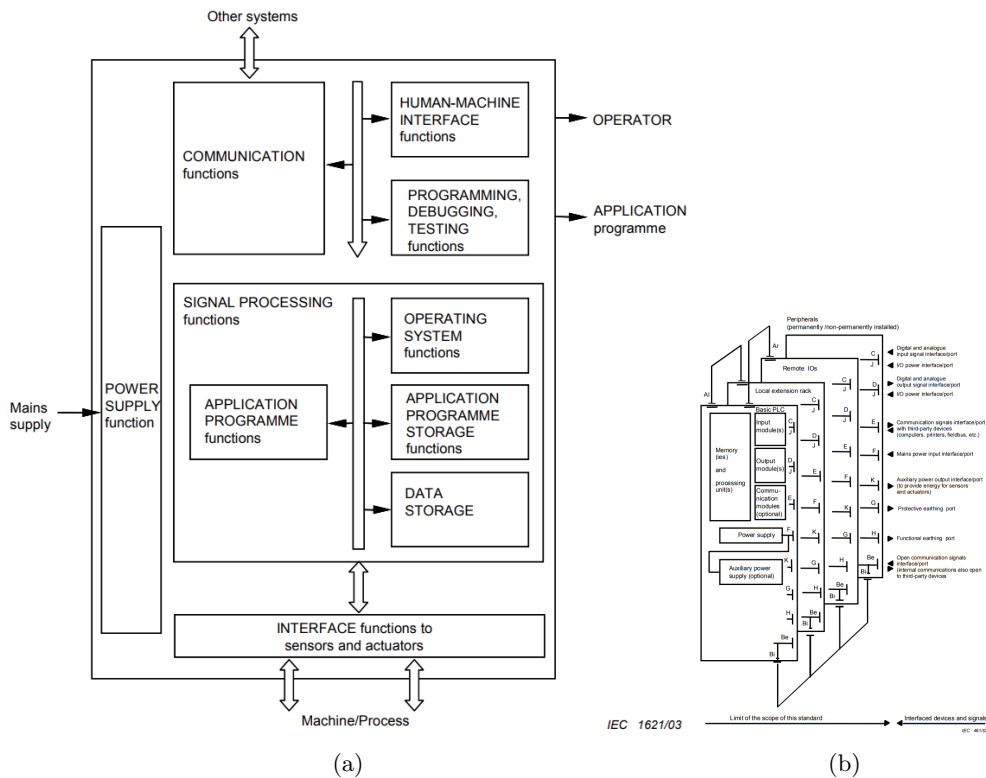


Figure 5: The architecture of PLC from IEC 61131, (a) shows a simple version from IEC 61131-1 [11]. And (b) shows an extended version from IEC 61131-2 [13]. In addition there is a larger version of it in Appendix A.

### 2.1.4 PLC Inputs and Outputs

The input and output interfaces is as said, the part of the PLC that interfaces with input and output devices. This is the part of PLCs that's used for communication with external devices. From this interface, the controlled devices receives and output signal from the controller, and the controllers receives input signals from the external devices. There are different kinds of inputs and

---

outputs associated with a PLC.

A PLC can have both discrete/digital I/O and analog I/O. Discrete inputs can for example be provided by push buttons or phototransistors. Analog inputs can be provided by for instance a temperature sensor. The discrete outputs can then be LED's or other lights with one intensity, or conveyor belts that run in only one speed. Whilst a motor that need speed control, might use an analog output from the PLC. One example is CNC machines with speed control for specific tools.

The processor part of the PLC is using low voltages for its operations. For such components, 3.3 volt to 5 volt is common as the source voltage. While the PLC can use many different voltages depending on the I/O module, the most common for discrete signals is 24V. The processor cannot give such voltages directly, or read them since that would break it.

That is why opto-isolators are commonly used for the inputs they read. There are three common ways to make the low voltage signals from the processor to the correct output voltage. These are relay-types, transistor-types and triacs. The relays and transistors work as switches that lets through the correct electrical signal, while the triacs only work for AC current [8], [5].

In the IEC 61131-2 chapter 6 standard, there are specifications of how the I/O and other functions are required to be compliant. For a specific output type there cannot be more than a given amount leaking current in its off state as an example. These requirements are important for the PLCs to behave predictably, and to be safe, robust, and have desired longevity in the environments they are used.

### 2.1.5 PLC Manufacturers and Models

There are many different manufacturers of PLCs to choose from if such a unit is desired. They all have their pros and cons, and usually requires proprietary environments for coding, uploading the code, and communication between units. There are also open standards for communications if there is a need for mixed environments. Modicon was the first manufacturer of PLCs, although Siemens is one of the most popular manufacturers of PLCs today. See Figure 6 for an example of a PLC from Siemens [5], [7], [8].

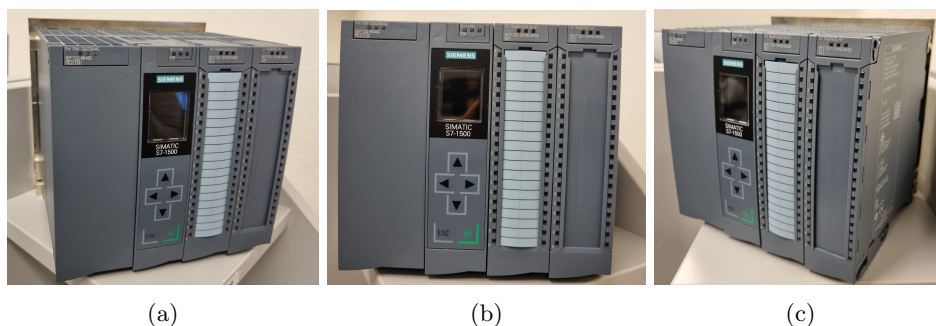


Figure 6: Three pictures were taken of a Siemens Simatic S7-1500 PLC

## 2.2 Scheduling algorithm for Ubuntu Linux

The normal scheduling algorithm used in the Ubuntu distribution of Linux is called Completely-Fair-Scheduler (CFS). Most scheduling algorithms are giving scheduled processes a fixed time-slice.

---

*CFS* is however designed to evenly divide the CPU between all of the competing processes.

The scheduler does this by choosing the process with the lowest virtual runtime (**vruntime**). All processes accumulates runtime while running and the process with the lowest *vruntime* is chosen to run when a scheduling decision occurs. The time slots used for running each process is allocated dynamically and is using a variable called `sched_latency` to decide the allocated time frame for the running process. This is usually in 48 milliseconds and is divided on  $n$  number currently scheduled processes. Ensuring that all processes get to run fairly on the *CPU* after a certain amount of time [14].

### 2.2.1 Real-time kernel for Linux

An operating system with a fully pre-emptible kernel is usually required to run certain mechatronic systems. This type of kernel is used when there are specific constraints and to secure a certain "worst time" possible when there are strict demands of the maximum allowed delay. Ubuntu Linux which in this thesis doesn't run on a fully pre-emptible kernel from the box, which means the OS has to be patched to allow real-time capabilities. A detailed explanation on how to do this is given in Section 3.2. According to Redhat.com [15], an RT kernel has the following advantages to an OS with a regular kernel.

- Checks task-priority under load
- High priority tasks are given preference for CPU execution
- Maintains a low latency execution time
- Possible to check, measure and configure response time

A kernel preemption makes the kernel able to change between processes using the scheduling algorithm, even if it already has another process running. This change makes the kernel able to process higher-priority tasks by interrupting already running tasks, and finish them later.

With a fully pre-emptible kernel, there can be set a maximum delay for certain high-priority tasks. This is due to the delay being independent on the complexity of the processes or tasks already running. This is a desired quality when running systems with strict demands of the maximum delay. Some examples are car production lines and pace makers.

## 2.3 Raspberry Pi

Raspberry Pi is a single-board computer originally made to encourage learning and data science in schools and developing countries. It is built and developed by the british Raspberry Pi foundation in cooperation with Broadcom. At the time of writing, the latest development, and the version used in this project is the Raspberry Pi 4 Model B. This is currently available for everyone to buy, and it exists with choices of 2GB, 4GB and 8GB RAM to name a few. The Raspberry Pi is using a CPU with ARM architecture, and uses an open-source version of Debian Linux as an operating system. For more information about the specifications of the Raspberry Pi see the official RPi page [16]. A picture of a Raspberry Pi is added below in Figure 7.

---

There are a lot of reasons why Raspberry Pi was chosen over other single board computers. Mainly it was due to accessibility and modularity. A Raspberry Pi is made as an educational tool with a lot of ports and signaling capabilities. In addition, there is a lot of community support online, which makes solving problems easier compared to computers with no online support. All of these reasons makes it ideal for prototyping. Raspberry Pis are normally not implemented nor used in industrial applications. One alternative to the Pi for this project, could be an industrial counterpart.



Figure 7: A picture of a Raspberry Pi 4 Model B

## 2.4 Arduino UNO

Arduino UNO is a micro controller that is made by the italian manufacturer Arduino for private use. It is a single chip micro controller that features a proprietary IDE called Arduino IDE. The UNO controller have been updated several times and been launched in several versions. The version used in this project is the UNO Rev3. This generation features the *ATmega 328p* processor and 14 programmable ports, which can be used for private projects. In addition, it has 6 PWM ports and 6 analog ports. The memory of this controller is 32 kB.

These controllers comes in many different configurations and versions. In addition, there is a lot of additional hardware that has been developed for the controller, both from Arduino or other 3rd party manufacturers such as *Sparkfun*. This includes for example wifi shields and I/O expanders, just to name a few. The cheap price of the controller and the open-source layout makes it ideal for prototyping. A picture of the Arduino Rev3 can be seen below in Figure 8.



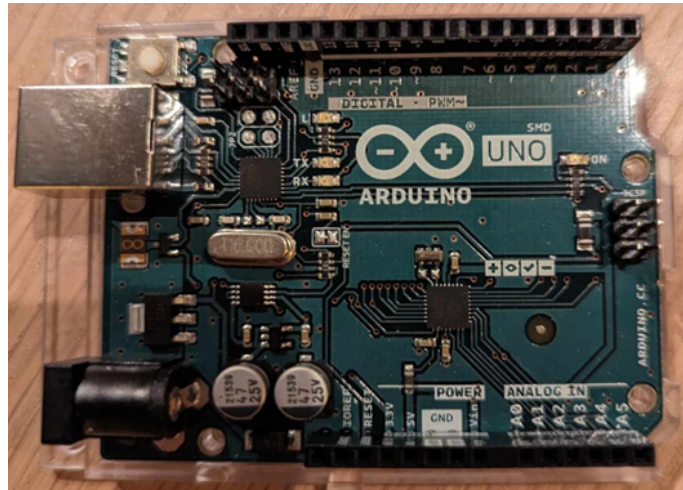


Figure 8: A picture of the Arduino UNO used for the project

## 2.5 Arduino MEGA

Arduino also have other micro controllers available for the private market. One of which is called Arduino MEGA. The one used in this project is called *Arduino MEGA 2560*. This micro controller is quite similar to the Arduino UNO. Although this controller have a lot more configurable pins, as well as higher memory speed. This device has a 256 kB flash memory, which means it has 8 times more memory than the standard UNO. This is mainly due to the more powerful ATmega 2560 CPU used by the Arduino MEGA. In addition, the Arduino MEGA features 54 programmable pins, which is a lot more than the UNO can provide. This makes it ideal as an alternative to the UNO, if the UNO doesn't provide enough programmable pins or lacks computing power. A picture of the Arduino MEGA is provided below in Figure 10.

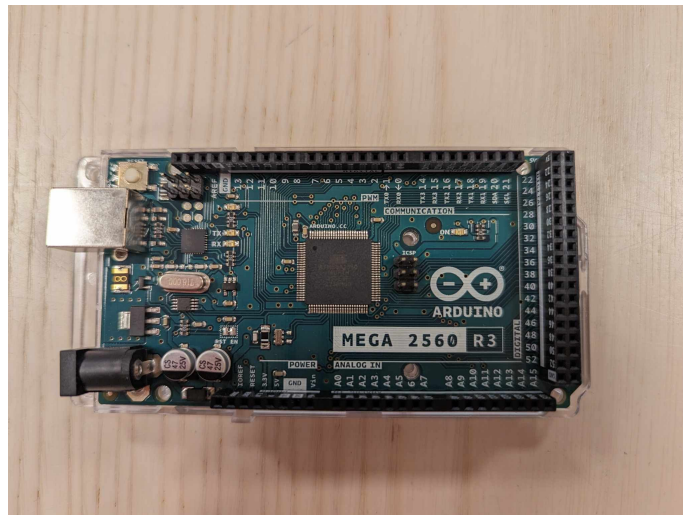


Figure 9: A picture of the Arduino MEGA used in the project

---

## 2.6 Arduino Pro

Arduino Pro is a series of Arduino devices featuring IoT-capabilities and is mainly designed for professional applications. Even though the target customers are companies, the devices can also be bought by private customers, due to its low cost compared to other PLCs available on the market. The Pro units are specifically designed for industrial control, AI edge processing and robotic applications.

This series of products have mainly two different models with smaller variations in each family. The two model lines are called Arduino Portenta and Arduino OPTA respectively. Specifically the Portenta Machine Control H7 is used in this Master's thesis. This model features an interface with wire plugs for fast connections, similar to PLCs from other manufacturers. The Portenta Machine Control also features an Embedded real time clock (RTC) which is required for real time systems. A picture of the Arduino Portenta can be seen in Figure 10. From this chapter and the rest of the assignment, the Arduino Portenta Machine Control H7 is going to be referred to as the *Arduino PMC*.

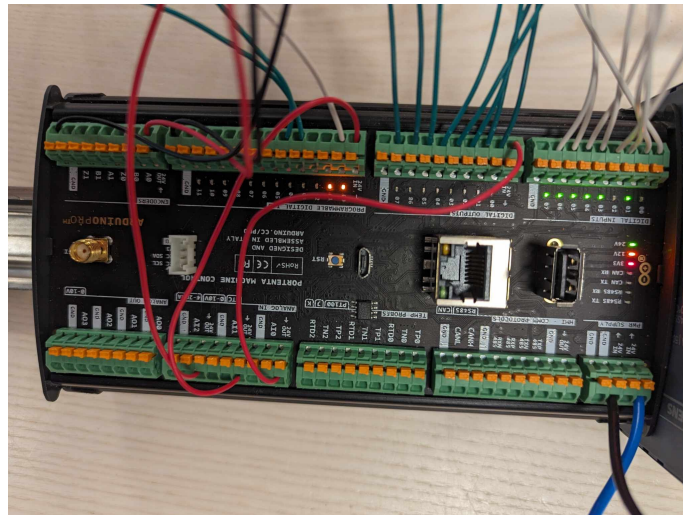


Figure 10: A picture of the Arduino PMC that's used in the project

## 2.7 OpenPLC

For this project, *OpenPLC* was chosen as an operating software for the mini-factory. According to the description on *GitHub*, it is an open-source Programmable Logic Controller (PLC) based on easy to use software. This is provided as a low-cost industrial solution for automation and research [17].

A benefit of OpenPLC is that the entire source code is provided and open-source. This also makes the software ideal for industrial cyber security research. However, this isn't relevant for this project. One of the most important aspects of this project is to be able to provide a framework to learn different types of PLC languages. OpenPLC follows the international IEC 61131-3 standard, which is the official standard for PLC programming languages [12]. In addition, OpenPLC is both easy to use and completely free. This makes the software highly relevant to use for this project.

---

## 2.8 Arduino PLC IDE

Arduino PLC IDE is an IDE developed for the Arduino PLC units. This includes the Arduino Portenta Machine Control and the Arduino OPTA, which are two PLCs Arduino provides. The software provides an IDE for programming in all of the 5 approved PLC languages defined by the IEC 61131-3 [12] standard, similarly to 2.7. The program is free to download, but needs a license activation to unlock all the capabilities in the program. This key is provided by *Arduino* and costs 16 € at the time of writing. The program also needs connection to a compatible device, this is done by connecting a device to your PC, and download the necessary drivers installed with the software. A guide on how to set this up is provided in Section 3.1.

According to their website a user is able to "mix PLC programming with Arduino sketches within the integrated sketch editor, and seamlessly share variables between the two environments" [18]. The program also provide different solutions for industrial communication, with the possibility to manage CANOpen, Modbus RTU and Modbus TCP communication. The program also provides integrated no-code fieldbus configurators.

---

## 3 First time setup of the devices

### 3.1 First time setup of the Arduino PMC

For the first startup of the Arduino PMC, the board needs to be connected to a computer using the Micro-USB port located on the board. On newer models, this is a USB-C port. After connecting it to your PC, the software drivers needs to be installed. There are several ways to do this, but the method chosen in this project is using the Arduino PLC IDE. This guide is inspired by the guide found on the Arduino website [19].

Step 1 is download the Arduino PLC IDE Tools and the Arduino PLC IDE from the official website. These programs require an operative system of Windows 10 or newer, based on the x64 architecture. On the time of writing, this is found at <https://www.arduino.cc/en/software#arduino-plc-ide> and is pictured in Figure 11. The version of the program used in this project is 1.0.4.

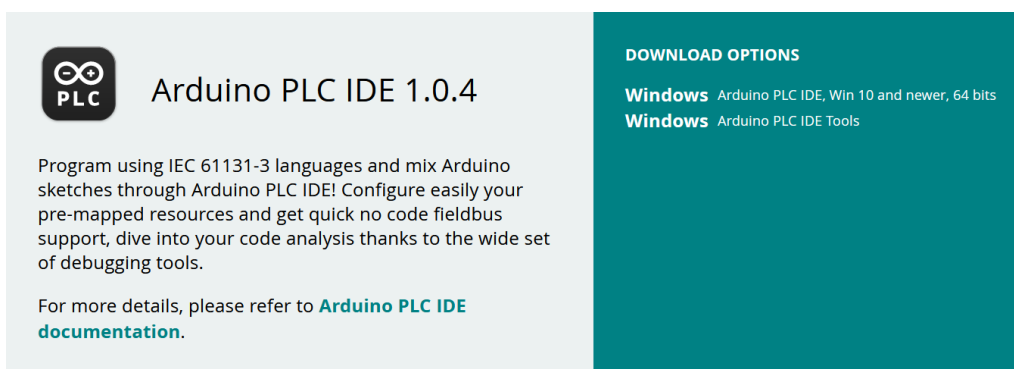


Figure 11: IDE download

The Arduino PLC IDE Tools package features the required drivers, libraries and cores that is needed for the program. The second executable installs the IDE itself.

The next step is to start the program, where a welcome screen will appear as seen in Figure 12. To be able to code and install the drivers, a new project needs to be created. When pressing the "New Project" button, a pop-up will appear. These text-boxes featuring a project name and a path to the directory the project will be saved. A target device also needs to be selected, which is the Arduino Portenta MC 1.0. After this, hit "OK".

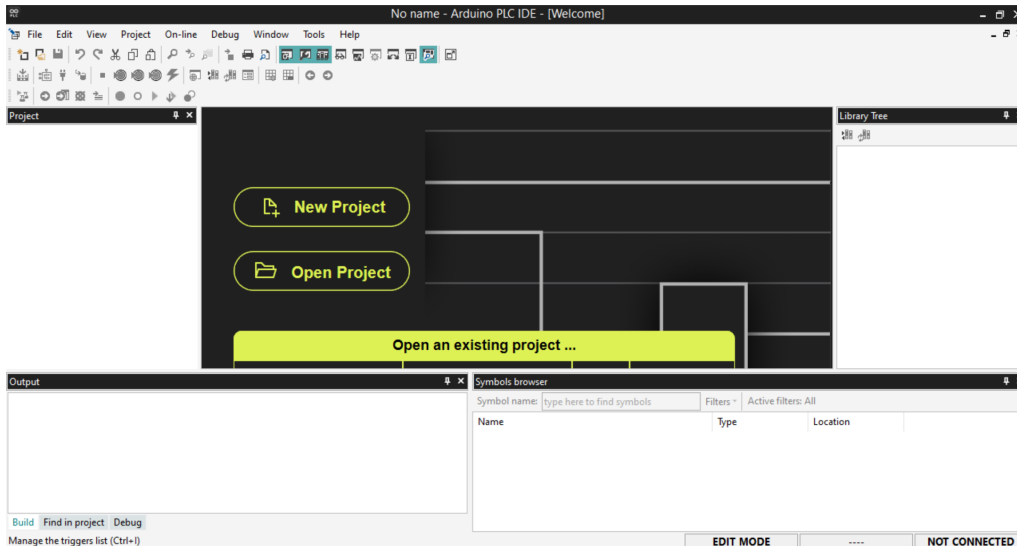


Figure 12: IDE download

After this step, a screen containing Arduino PMC configuration will appear. The drivers required for the Arduino PMC is found under "Other". To install the drivers on the PLC, find the correct COM-port, and press download.

- The device might show two serial ports. The usual one in most instances is the one with the lowest number. The one with the highest serial number is usually the port for enabling Modbus communication for the device. Take a note of this COM-port, as it will be needed in a later stage

If the error box seen in Figure 13 appears, double tap the reset button on the Arduino PMC. The LED will start flashing indicating that the device is ready to be flashed with the new firmware. After this, just press the download button again.

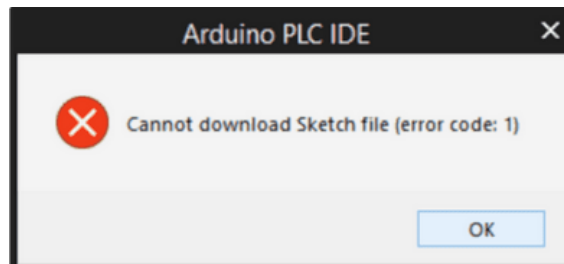


Figure 13: Error message pop-up

After the runtime is up and running, set up the communication by going to *On-line > Set up communication*. On the new appearing pop-up window, open the properties for the Modbus protocol as seen in Figure 14.

Make sure the Modbus protocol is using the secondary port number that was written down earlier and press OK. Press OK again to save the settings and connect the device using *On-line > Connect*.

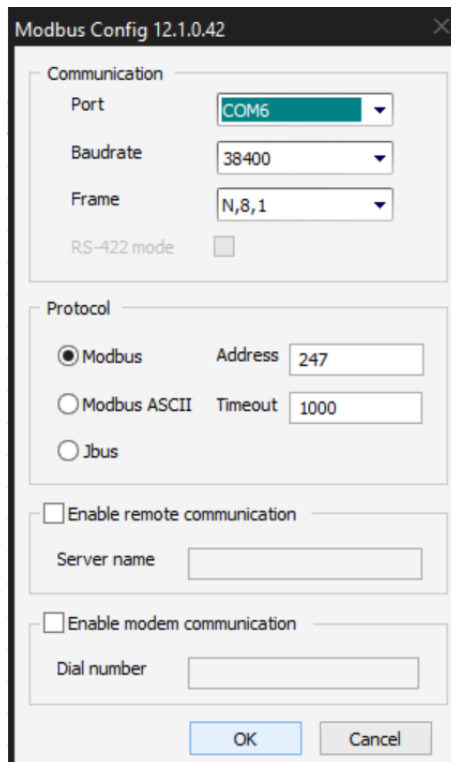


Figure 14: The modbus connection ports

### 3.1.1 License Activation with Product key

The Arduino PMC is possible to buy with a pre-activated license. If a device is bought without the activation key included, the following steps will tell how to get the key and activate it.

If the communication is successful, a status indicator will show up in the license section.

To buy the license key, go into the official Arduino website and buy the key. At the time of writing, one license key costs 16€ to buy. To activate the license key, paste in the key into the blank space on the Arduino PMC configuration page in the Arduino PLC IDE, and press the *Activate* button. After the license is activated, the key is bound to the hardware ID on the device. A popup telling the user to reboot the target device appears. When this message pops up, press the reset button on the Arduino PMC twice. Once the license is activated and the drivers are installed, the device can be connected. After successful communication, code can be uploaded from different computers. Meaning the license activation is following the device, and not the first computer that the Arduino PMC is connected to.

## 3.2 Setup of real-time kernel on Raspberry Pi

This section has been taken directly from the preceding project thesis due to still being highly relevant for this thesis. Therefore, this guide on patching the kernel for Ubuntu Linux is added here as well with none or minor changes.

A lot of mechatronic systems have strict requirements of the amount of delay allowed by the system. To be able to work with such a system, a real time kernel is needed. Therefore, it was desired to

---

run the mini-factory with a real time operating system as well. This real time kernel have been set up to work on Debian Linux for Raspberry Pis for this project. To do this, a guide inspired from [4] were followed.

The first step to patch the Raspberry Pi kernel, is done by cross-compiling with a host computer. To do this, the appropriate tools needs to be installed on the computer:

```
$ sudo apt-get install build-essential libgmp-dev libmpfr-dev
↳ libmpc-dev libisl-dev libncurses5-dev bc git-core bison flex
$ sudo apt install libelf-dev
```

```
$ sudo apt-get install libncurses-dev libssl-dev
```

After installing the necessary tools, the next step is to compile the native build for cross-compiling. By typing the following line, *Binutils* is installed:

```
$ cd Downloads
$ wget https://ftp.gnu.org/gnu/binutils/binutils-2.35.tar.bz2
$ tar xf binutils-2.35.tar.bz2
$ cd binutils-2.35/
$ ./configure --prefix=/opt/aarch64 --target=aarch64-linux-gnu
↳ --disable-nls
```

After configuration, use the following commands to compile the program:

```
$ make -j4
$ sudo make install
```

The path needs to be exported after compilation. To do this, type the following commands:

```
$ export PATH=$PATH:/opt/aarch64/bin/
```

After exporting the path, build and install GCC with the following commands:

```
$ cd ..
$ wget https://ftp.gnu.org/gnu/gcc/gcc-8.4.0/gcc-8.4.0.tar.xz
$ tar xf gcc-8.4.0.tar.xz
$ cd gcc-8.4.0/
$ ./contrib/download_prerequisites
$ ./configure --prefix=/opt/aarch64 --target=aarch64-linux-gnu
↳ --with-newlib --without-headers --disable-nls --disable-shared
↳ --disable-threads --disable-libssp --disable-decimal-float
↳ --disable-libquadmath --disable-libvtv --disable-libgomp
↳ --disable-libatomic --enable-languages=c --disable-multilib
```

---

The next step is to compile *GCC*.

```
$ make -j4
$ sudo make install gcc
```

Due to the patching and installation of kernel is going to be done by cross-compilation, there is a need to be sure that the compiler is installed on the host-computer. To download this, type the following commands into the terminal:

```
$ sudo apt-get update
$ sudo apt-get install gcc-aarch64-linux-gnu
```

The tools required to build and install the patch onto the kernel, should now be downloaded and ready. The kernel version used in this project is *v5.15* with the corresponding patch *RT49*.

Continuing the process on the host-computer, make a new directory. The kernel and the corresponding real-time patch needs to be downloaded next. It is possible to patch a non-corresponding version of the kernel, but it's no guarantee that it will work properly. To do this, type the following lines in the terminal:

```
$ mkdir ~/rpi-kernel
$ cd ~/rpi-kernel
$ git clone https://github.com/raspberrypi/linux.git -b rpi-5.15.y
```

The patch used in this project is downloaded by typing the following lines in the terminal:

```
$ wget https://mirrors.edge.kernel.org/pub/linux/kernel
→ /projects/rt/5.15/older/patch-5.15.65-rt49.patch.gz
```

If it's desired to download another patch, the newest patches are found from <https://mirrors.edge.kernel.org/pub/linux/kernel/projects/rt/5.15/>. To download this, just change the fields by replacing "XX" and "YY" with the newest version numbers `patch-5.15.XX-rtYY.patch.gz`.

```
$ wget https://mirrors.edge.kernel.org/pub/linux/kernel
→ /projects/rt/5.15/patch-5.15.XX-rtYY.patch.gz
```

Applying the patch to the kernel is done by typing:

```
$ mkdir kernel-out
$ cd linux
$ gzip -cd ../patch-5.15.65-rt49.patch.gz | patch -p1 --verbose
```



---

Before building the patch, the configuration has to be set up to allow real-time capabilities for the kernel of the Raspberry Pi. To apply the default settings, type:

```
$ make O=../kernel-out/ ARCH=arm64
↪ CROSS_COMPILE=/opt/aarch64/bin/aarch64-linux-gnu-
↪ bcm2711_defconfig
```

In addition to these settings there is a need to change the setting of CONFIG\_KVM to unlock real-time capabilities. An explanation of why this is the case is discussed in the preceding project assignment. This is done by the following commands:

```
$ cd ..
$ cd kernel-out
$ echo -e "CONFIG_EXPERT=y\nCONFIG_KVM=n" >> .config
$ cd ..
$ cd linux
```

After these lines are written in the terminal, it should now be possible to choose the real-time kernel in the `menuconfig`. To open the `menuconfig`, type the following command:

```
$ make O=../kernel-out/ ARCH=arm64
↪ CROSS_COMPILE=/opt/aarch64/bin/aarch64-linux-gnu- menuconfig
```

Enable "FULLY-PREEMPTIBLE KERNEL (REAL-TIME)" in this menu. This is done by doing the following steps:

1. General setup
2. Preemption model
3. Fully preemptible (Real time)

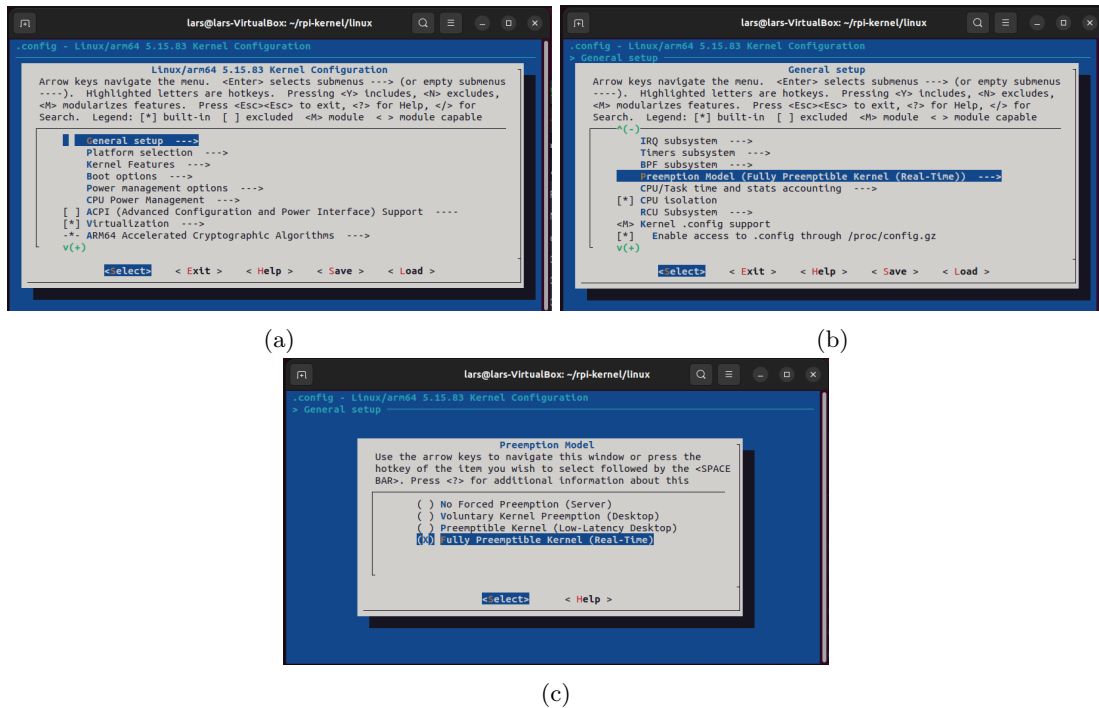


Figure 15: Images of the menuconfig (a) shows General setup, and (b) shows the Preemption-model and (c) fully pre-emptible kernel

Figure 15 shows what the menuconfig looks like on a PC.

After this step, the new rt-kernel needs to be built and compiled, this is done with the following line:

```
$ make -j4 O=../kernel-out/ ARCH=arm64
→ CROSS_COMPILE=aarch64-linux-gnu-
```

When the compilation is finished, the next step is to zip the kernel. This is done by the following lines in the terminal:

```
$ export INSTALL_MOD_PATH=~ /rpi-kernel/rt-kernel
$ export INSTALL_DTBS_PATH=~ /rpi-kernel/rt-kernel
$ make O=../kernel-out/ ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
→ modules_install dtbs_install
$ mkdir ../rt-kernel/boot
$ cp ../kernel-out/arch/arm64/boot/Image ../rt-kernel/boot/kernel8.img
$ cd $INSTALL_MOD_PATH
$ tar czf ../rt-kernel.tgz *
$ cd ..
```

The kernel should now be zipped inside "rt-kernel.tgz", which now needs to be sent to the Raspberry Pi. This can be done either by using a USB-Stick or through SCP. Write the following line in the terminal to send the file via SCP:

---

```
$ scp rt-kernel.tgz pi@<ipaddress>:/tmp
```

The last steps is done on the Raspberry Pi itself. To install the newly built kernel, the following commands needs to be executed:

```
$ cd /tmp
$ tar xzf rt-kernel.tgz
$ cd boot
$ sudo cp -rd * /boot/
$ cd ../lib
$ sudo cp -dr * /lib/
$ cd ../overlays
$ sudo cp -dr * /boot/overlays
$ cd ../broadcom
$ sudo cp -dr bcm* /boot/
```

After the commands above are executed, the file `"/boot/config.txt"` needs to be edited by appending the line `"kernel=kernel8.img"` at the end.

In order to apply the changes, and check if it installs successfully, the Raspberry Pi needs to be rebooted. After rebooting is complete, the following command can be executed:

```
$ uname -a
```

If the installation is successful, the output should be along the lines of `"Linux raspberrypi 5.15.65-rt49-v8+ 1 SMP PREEMPT RT Fri Dec 1 01:17:07 CET 2022 aarch 64GNU/Linux"`.

### 3.3 Installing OpenPLC runtime on the Raspberry Pi

The first step to install the OpenPLC runtime on the Raspberry Pi, is to connect it to internet. On a normal private network this can be done in the normal way, but on Eduroam it is a bit more inconvenient process. It is possible to circumvent this by sharing WiFi with a cellphone, but it is recommended to connect directly to Eduroam. A guide on how to connect to Eduroam for the Raspberry Pi is provided in [20], and can be seen in Appendix B.

The Runtime for OpenPLC can be installed on different kinds of devices, and on OpenPLC webpage there are guides on how to install each of them. Following the Linux version of the guide, the Raspberry Pi specific options is the recommended way of installing it. This guide can be found from [21]. In this thesis, it was installed exactly like this.

The easiest way to install the OpenPLC Runtime on a Raspberry Pi is to use *git*. This is usually done by getting it directly from the official site. To ensure git is installed, write this in the terminal:

---

```
$ sudo apt-get install git
```

To install the runtime, write the following lines in the terminal after git is installed:

```
$ git clone https://github.com/thiagoralves/OpenPLC_v3.git
$ cd OpenPLC_v3
$ ./install.sh rpi
```

When it is done, the RPi needs to be rebooted, which can be done by typing this in the terminal:

```
$ reboot
```

The runtime is now installed, but it will not work quite yet. The RPi version is depending on the WiringPi library. It can be downloaded from Github, where the latest version can be found on:

```
https://github.com/WiringPi/WiringPi/releases/
```

The `-armhf.deb` file should be used on 32-bit OS (Raspberry Pi 3 and under) and the `-arm64.deb` is meant for 64-bit OS (Raspberry Pi 4 and up). Download the appropriate file for your architecture on your Raspberry Pi. In this thesis, a Raspberry Pi 4B is used with a 64-bit OS, and therefore the `-arm64.deb` is the one downloaded. Then install it with one of the two `dpkg` command:

```
$ dpkg -i wiringpi-[version]-armhf.deb

or

$ dpkg -i wiringpi-[version]-arm64.deb
```

The newest version during the installation, and what used in this thesis are 2.61-1, and the file ended up in the Downloads folder. It can also be necessary to use the `sudo` command. The exact commands used in the thesis are:

```
$ cd Downloads
$ sudo dpkg -i wiringpi-2.61-1-arm64.deb
```

Test that the WiringPi installation finished successfully with the command:

```
$ gpio -v
```

---

## 4 Technical specifications

This chapter includes all of the technical specifications of the factory, and the means that were taken to test these. This were mostly done in the preceding project thesis, but is still relevant for the thesis since this information is used when updating the PCB. The following subsections mentioned in the list below, has been either taken directly, or added with minor adjustments from the project thesis. This includes sections:

- Section 4.1
- Section 4.2
- Section 4.3

### 4.1 The interface circuitry

The main purpose of this thesis is to look at different types of micro controllers to emulate a Programmable Logic Controller (**PLC**) and run the Indexed Line factory. PLCs are usually used in the industry to run production lines or other industrial sequential processes. PLCs are commonly used due to high reliability, ease of programming and process fault diagnostics. In educational purposes, it might be seen as unnecessary to get a PLC, mostly due to cost. Microcontrollers are usually easier to attain, and doesn't require manufacturer specific software. A microcontroller is additionally more flexible than a PLC, and have the ability to be used for much more diverse tasks.

As described in Section 2.3, a Raspberry Pi was chosen for this project. To be able to emulate a PLC, a fitting software needs to be used. The software used for this project was OpenPLC, which is an open-source PLC software. This software is made as an affordable and accessible method to learn PLC programming using the global IEC-61131-3 standard. The program is written in the programming language "C", which makes it highly portable over multiple devices. The examples of compatible operating systems that are given on their official site are Linux, Arduino, Windows and Raspbian [17].

In Figure 16, the general architecture map for a PLC is shown. The micro controller is going to be responsible for the modules shown with a green square. The modules marked by a red square is going to be emulated by an interfacing circuit. This circuit is not necessary for the Arduino PMC, since this device is capable of giving 24 volt signals from default. This interfacing circuit was made in the preceding project thesis, and has been further improved in this thesis. The devices connected to the PLC are marked with a blue square. The power for the system will be partially provided from the micro controller. In addition, an external power supply will be connected directly to the PCB.

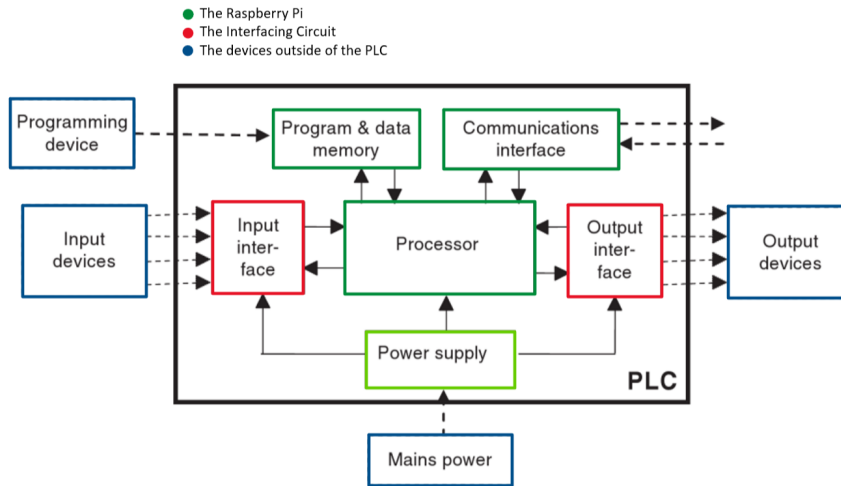


Figure 16: The planned way to emulate the general architecture of PLCs. Based on Figure 4 from [5].

The main purpose of the interface circuitry and the PCB, is to ensure that both the microcontroller and mini-factory gets the required voltage to run the required modules. The voltage provided from the IO ports on microcontrollers made for private use, are usually specified to be between 2.7 to 5 volts. The actuators, sensors and motors on the mini-factory requires 24 volts to run. Therefore it is desired to make a circuit that will safely translate low voltage signals to higher voltage signals. Meanwhile, it is important to translate back from 24 volts to 3.3 volts, to safely read the input signals given from the sensors of the factory. If the input voltage to the Raspberry Pi is too high, the power might ruin IO pins, or in the worst case, ruin the whole controller. A brief sketch of how this might be done, is given in figure Figure 17.

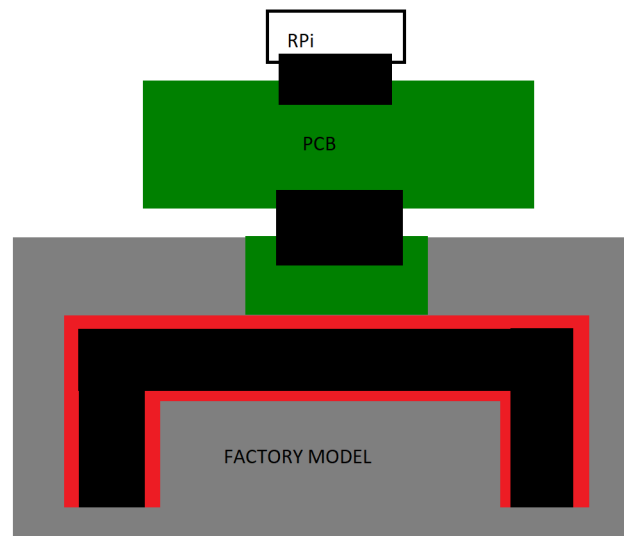


Figure 17: Rapid sketch of how to turn on a 24V signal from 5 volts and lower voltages

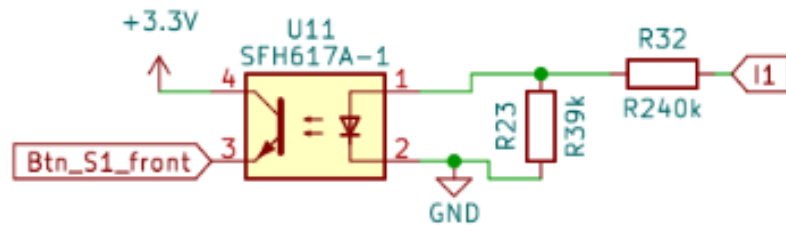
The circuits that were developed during the preceding project assignment are shown in Figure 18. There is one of the circuit seen in Figure 18a per input on the factory model. And one output

circuit seen in Figure 18b for each output on the factory.

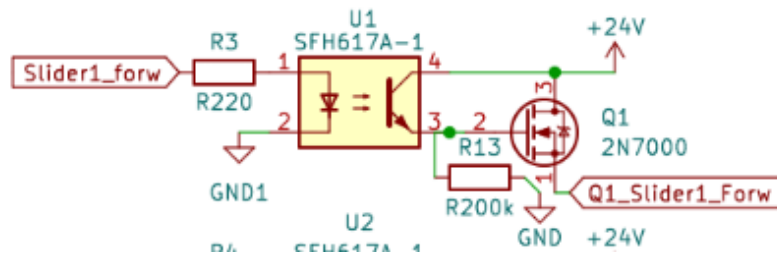
The input circuit is made by 3 components, two resistors and an optocoupler. When the signal from the sensors on the factory activates, power is sent through the optocoupler on the right side. This will activate the gate on the left side of the optocoupler, and 3.3 volt signal will pass through to the micro controller.

The output circuit consists of two resistors, an optocoupler and a 2N7000 MOSFET transistor. The signals from the micro controller is sent on the left side of the opto-coupler and opens the gate on the right side. This allows a power with 24 volts go through the optocoupler and transistor, sending a signal to the factory.

These circuits were developed to shield the micro controller from the 24 volt signals used by the factory, making the micro controller read signal in it's own voltage. If the reader is curious of how these were developed, the full description is added in the project assignment.



(a)



(b)

Figure 18: The output and input circuit developed for the project assignment

---

## 4.2 Specifications of the mini-factory

In the datasheet, Table 1 was provided as an overview of the pins and functions of the factory.

Table 1: Overview of the pins given by the data sheet of the factory

Terminal/Pin	Function	Input/Output
1	Power supply actuators (+)	24V DC
2	Power supply sensors (+)	24V DC
3	Power supply (-)	0V (GND)
4	Power supply (-)	0V (GND)
5	Push-button slider 1 front	I1
6	Push-button slider 1 rear	I2
7	Push-button slider 2 front	I3
8	Push-button slider 1 rear	I4
9	Phototransistor slider1	I5
10	Phototransistor milling machine	I6
11	Phototransistor loading station	I7
12	Phototransistor drilling machine	I8
13	Phototransistor conveyor belt swap	I9
14		
15	Motor slider 1 backward	Q1
16	Motor slider 1 forward	Q2
17	Motor slider 2 backward	Q3
18	Motor slider 2 forward	Q4
19	Motor conveyor belt feed	Q5
20	Motor conveyor belt milling machine	Q6
21	Motor milling machine	Q7
22	Motor conveyor belt drilling machine	Q8
23	Motor drilling machine	Q9
24	Motor conveyor belt swap	Q10

To check that everything on the mini-factory worked correctly, a powersupply with adjustable voltage and electric current was connected into the power slots on a breadboard. The voltage was adjusted to 24V and the current was set to 0.1 Amps. Two wires were then connected to pin 1 and 3. These were added to power up the actuators of the mini factory. When the actuators had power, the motors could be tested. A new wire connected the powered slots on the breadboard, to the input terminal 15. After observing that the motor was running, terminal 16-24 were tested accordingly.

A similar method were used to test the remaining pins. The wires providing power were connected to pin 2 and 4, to power up the sensors of the mini-factory. When the factory had power, a multimeter were connected to pin 5 and ground. The button connected to pin 5 was pressed, and a change in voltage could be observed. This indicated that the button was functional. The same method were used for the remaining buttons, found on terminals 5-8. The phototransistors were tested with the same setup as the buttons, although these sensors were activated by blocking the light signal between the LED light source and phototransistor. A similar change in voltage was observed, but opposite. Unlike the buttons, the phototransistors are normally open, and closes when an object is blocking the signal between them.

When the signal circuit were tested, it was discovered that the motors where pulling a current from their respective signal pins. The current they pulled was low, under 0.1 Ampere. This is still a significant current when the assumption was that they only required pure voltage signals. An



investigation of how much current they should theoretically pull at maximum power started. On Fischertechnik's website of the mini-factory [2], it says that all the motors are XS DC motors. This doesn't give a lot of information that could be used, so further investigation was conducted.

In the last question in their technical FAQ [22], it was found that the motors pull a maximum of 0.265A at 9V. This gives a maximum current draw at 24V of 0.0994A.

		d.c. output style																							
		Type 0,1	Type 0,25	Type 0,5	Type 1	Type 2																			
<b>Rated current for state 1</b>	$I_e$ (A)	0,1	0,25	0,5	1	2	<b>Normative items</b>																		
Current range for state 1 at maximum voltage (continuous)		Max. (A)	0,12	0,3	0,6	1,2	2,4																		
Voltage drop, $U_d$	Non-protected output	Max. (V)	3	3	3	3	3																		
	Protected and short-circuit-proof	Max. (V)	3	3	3	3	3	a																	
Leakage current for state 0		Max. (mA)	0,1	0,5	0,5	1	1	b, c																	
Temporary overload		Max. (A)	See Figure 15 or as specified by manufacturer																						
<p><sup>a</sup> For 1 A and 2 A rated currents, if reverse polarity protection is provided, a 5 V drop is allowed. This makes the output incompatible with a type 1 input of the same voltage rating.</p> <p><sup>b</sup> The resulting compatibility between d.c. outputs and d.c. inputs, without additional external load, is as follows:</p> <table border="1" style="margin-left: 40px;"> <thead> <tr> <th></th> <th>Output Type 0,1</th> <th>Output Type 0,25</th> <th>Output Type 0,5</th> <th>Output Type 1</th> <th>Output Type 2</th> </tr> </thead> <tbody> <tr> <td>Input Type 1:</td> <td>yes</td> <td>yes</td> <td>yes</td> <td>no</td> <td>no</td> </tr> <tr> <td>Input Type 3:</td> <td>yes</td> <td>yes</td> <td>yes</td> <td>yes</td> <td>yes</td> </tr> </tbody> </table> <p><sup>c</sup> With adequate external load, all d.c. outputs may become compatible with all Type 1 and Type 3 d.c. inputs.</p>									Output Type 0,1	Output Type 0,25	Output Type 0,5	Output Type 1	Output Type 2	Input Type 1:	yes	yes	yes	no	no	Input Type 3:	yes	yes	yes	yes	yes
	Output Type 0,1	Output Type 0,25	Output Type 0,5	Output Type 1	Output Type 2																				
Input Type 1:	yes	yes	yes	no	no																				
Input Type 3:	yes	yes	yes	yes	yes																				

Figure 19: The Digital outputs for direct current table from section 6.4.6.1 in [13].

### 4.3 PLC standards

One of the surprises found when testing, was that the mini-factory drew more current from the signal pins than first assumed. This assumption was mainly based on the existence of pin 1 in the spec sheet provided in Table 1, which was thought to provide enough current to run the whole factory.

To figure out if this could be a problem or not, an investigation of the output signal current of PLC's were started. On Siemens's site [23] there was found that the output modules for SIMATIC S7-1500 are capable of a minimum of 0.3A as their maximum current. There were other modules that could give more, but it was the module that gave the lowest maximum that was interesting. Since the modules can provide enough current to drive the motors, it seems to not be a problem. Therefore, it was reasonable to adjust the resistors of the PCB circuit to provide a similar amount of current through the output signals as well.

Since one of the goals of this project was to make a Raspberry Pi emulate a PLC, it was seen as relevant to investigate the international standards of the workings of a PLC. This was done to see exactly what functions needed to be replicated. In IEC-61131 part 2 [13], the hardware standards is set. In this standard, the minimum required current for an output signal from a PLC was found.

---

The PLC standards for the power-signals are found in Figure 19. As can be seen from this table, the smallest output type is 0.1A. Meaning that a PLC's discrete output should always be able to give that amount of current. In that case the mini-factory follows the PLC standard of the power consumption, which explains why the motors needed additional current from the signal pins to run.

#### 4.4 Running the factory with Arduino PMC

Since the Arduino Portenta follows the international PLC standards, it is fit for running the factory out of the box. One of the main problems with running the factory on micro controllers is the challenge of getting the correct voltage to the factory, which is already mentioned in Section 4.1. One of the huge benefits of using the Arduino PMC, is that the connections are able to provide the necessary power at 24V to the motors, without the need of an extra hardware layer. This layer is necessary for the rest of the different micro-controllers tested in this project.

When testing the factory first time with the Portenta Machine Control unit, the cables were put directly from the connection slots on the PLC, to the slots on the factory. When the factory was provided with power from the Arduino PMC, the LED sensors on the factory lighted up, just as expected. The Arduino Portenta has LED's on each of the connection slots to tell if they are active or not. When the connection between the factory and PLC was established, the LED's for the input slots lighted up. This is due to the sensors being **True** by default and sending an on-signal to the device.

The outputs didn't establish connection the same way, and it was harder to tell if the pins actually were connected to the factory. This was due to the LEDs not lighting up when the connection was established. Therefore, the motors had to be tested using OpenPLC to set the engine output to **True**. This would send a 24 volt signal to the motor, and the motor would start running. Under activation, the LEDs would also light up, indicating that the signal is being transmitted correctly.

---

## 5 Connecting the factory and the micro controllers

In this master and preceding project assignment, several variations of micro controllers have been used to power the factory-model. This lead to some adaptations based on how the software and hardware interacts. This is going to be explained in further detail in this section.

### 5.1 The softwares

The Arduino PLC IDE is the proprietary software which is specifically made for the Arduino PMC units. At the time of writing, this includes the Arduino Portenta H7 and the Arduino OPTA. Using the comparable software to the Arduinos has some benefits compared to open-source programs such as OpenPLC. One of them is that the software works most of the time without changes in the source code, which has been done to make OpenPLC work properly for this project.

One drawback however, is that Arduino PLC IDE is only available for Microsoft Windows and compatible with only two devices. In comparison to Arduino PLC IDE, OpenPLC is compatible with multiple Operating Systems. This includes MacOS and Linux.

OpenPLC is also compatible with several different types of micro-controllers, in addition to being completely open-source. This means that if a certain micro-controller is not compatible yet, it is possible to write software drivers to add the functionality to the program.

### 5.2 Arduino Portenta Machine Control H7

In this section, the Arduino Portenta Machine Control (**PMC**) H7 is going to be referred to as the Arduino PMC.

#### 5.2.1 PMC Hardware

The Arduino PMC features several usable I/O pins to connect to mechatronic systems.

- 8 digital input pins
- 8 digital output pins
- 11 programmable digital pins
- 3 analog input pins
- 3 analog output pins

As well as some power and ground ports to power up the necessary outputs. The Arduino PMC is capable of providing 24v directly on the pins, therefore there is no need for an interface layer to transform voltages for this PLC as already mentioned in Section 4.4. A schematic over all of the connections on the Arduino PMC is provided below in Figure 20.

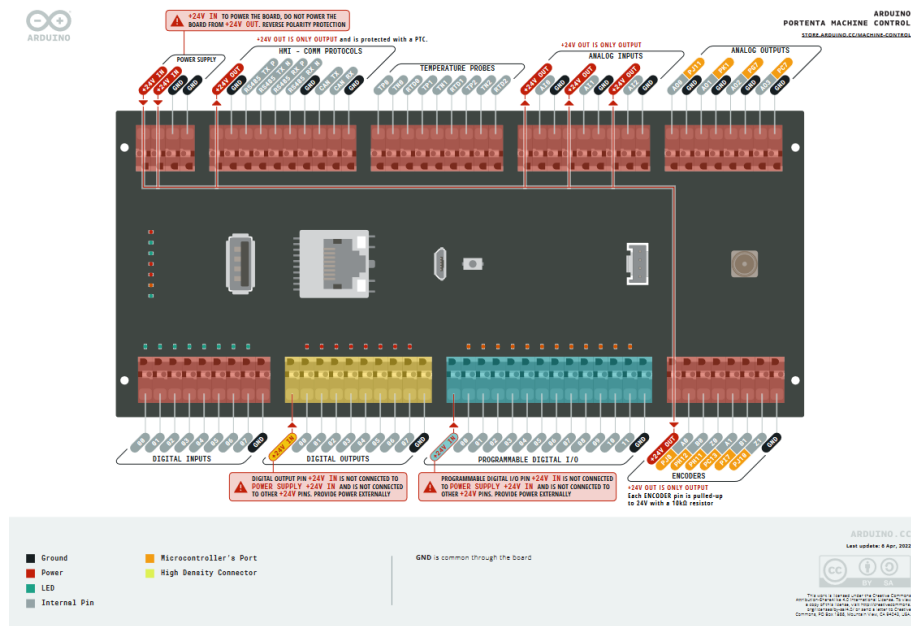


Figure 20: A schematic showing the functionality of each pin on the PMC. Picture is taken from the official datasheet

Even though the Arduino PMC features a lot of pins, it wasn't enough to directly run it from start. As seen in Figure 74, the factory features 5 sensors and 4 buttons. This is not including the input variables for the indication of milling and drilling. Anyway, that totals to 9 digital inputs which of whom should be connected to the 8 digital pins the Arduino board. The solution to this problem is better explained in Section 6, due to different methods being used for the two programs used for the project.

The same problem came up on the digital outputs, the factory features as mentioned 4 conveyors, 4 pins for sliders and 2 pins for milling and drilling. This makes for a total of 10 outputs that needs power from 8 output pins.

### 5.3 Arduino PMC with OpenPLC software

One of the main benefits for connecting the Arduino PMC to the factory is the simplicity of it. The factory is rated for a 24V voltage, which the Arduino Portenta is capable of delivering on each pin. As stated in the previous section, one problem was that the Arduino PMC did not have the amount of pins necessary to work with the OpenPLC software. This problem needed to be solved to properly use the Arduino PMC to run the factory. Therefore, different solutions is listed in the table under this section, as well as some advantages and disadvantages with each solution.

- Using the analog pins
- Writing a software driver

---

### 5.3.1 Arduino PMC analog pins

The first and seemingly simplest solution were to use the available analog pins on the PMC. This method allows the user to use the analog pins to emulate digital outputs, and program these to act as a switch for the **inputs** and **outputs**. There are 3 available analog inputs and 4 digital outputs on the Portenta H7 as previously seen in the schematic at Figure 20. This allows the user to connect the remaining motors and sensor to the factory without any larger changes.

The software solution is different for the analog inputs and outputs, but basically the main goal is to emulate digital pins using the analog pins. This can be done for both the input and output directly into the ladder logic code.

The solutions vary based on if it's an analog input or output, but the basic idea is to pass an integer to the pin instead of a boolean value, which indicates 0 and 1 respectively (**True** and **False**). The value that can be passed to the pin has a size of 16 bits. This means the passing integer must have a value between 0 and 65535 and therefore,  $2^{16}$  different combinations. The idea is to pass a value down to the pins, and make the pin switch state for a certain value. For example, the state can pass a low value when the integer is less than 30 000, and pass a high value if the value exceeds this number.

### 5.3.2 Analog input

The analog input, was the easiest to implement in this case. The last sensor (*Sensor5*) was connected to the analog inputs. The code for this particular sensor is added in 21.

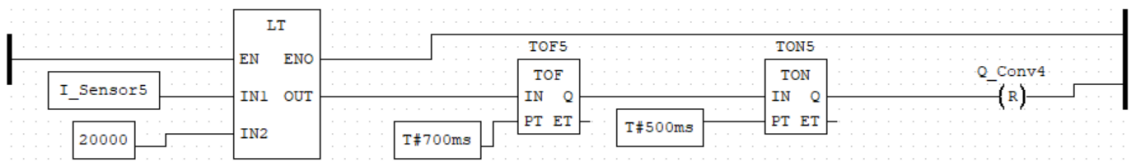


Figure 21: The ladder logic lines for an analog input of Sensor5

In the other parts of the code, contacts are used to read the digital input values from the factory. This can however, not be used for an analog input, since these are passing an integer value instead of a boolean value to the machine. Therefore, an "LT-block" is added to the code. The LT block is short for "Less Than", and it works by activating a signal if the input value is lower than a set boundary value. In this case, this boundary is set to 20000. When the input value **I\_Sensor5** is lower than this boundary, the signal from the power rails is allowed through the gate. Further leading to the rest of the code activating, and the program running normally. The addresses for the analog inputs are given as **IWy**, where "y" is the pin number.

Due to the sensor constantly sending a beam of light, the sensor will normally pass a high value to the input. This integer value will be lower when the diode on the sensor is blocked. Therefore, the "Less Than" block is ideal for this scenario.

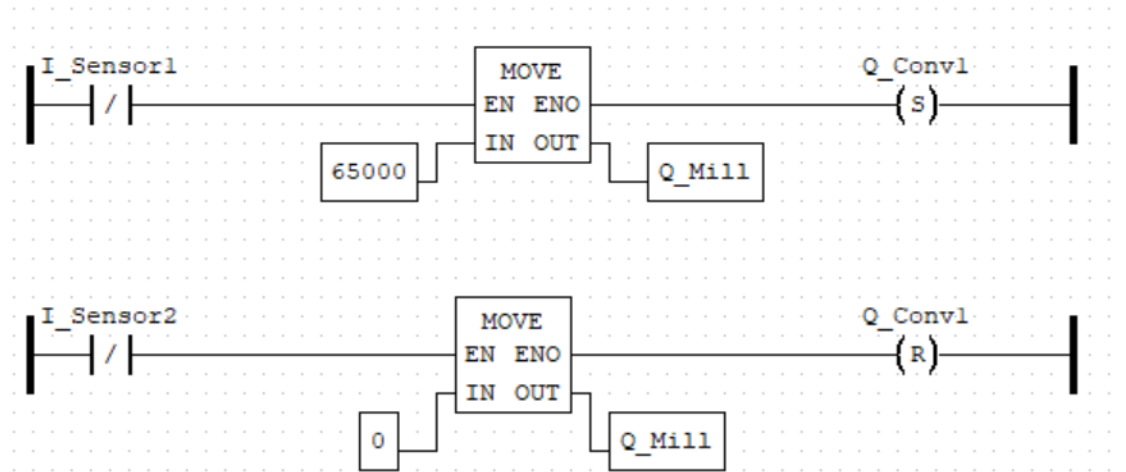


Figure 22: The ladder logic lines for the analog outputs of the milling and drilling station

### 5.3.3 Analog output

The analog output is a bit trickier. For the digital outputs in other parts of the code, **Set** and **Reset** has been used. These functions are only available for boolean values, which means they can't be used for the analog outputs directly. Solutions to this problem however, can be implemented using the programming environment in OpenPLC, similar to what was done with the analog inputs.

There are two motors that needs to be connected outside the programmable outputs of the Arduino PLC. For structural reasons, the motors connecting the analog outputs were chosen to be the milling and drilling station.

One problem of using the analog outputs to run the motors, is that the analog outputs on the device are rated for a current between 0-10V, found in the datasheet [24]. This can not be solved using software solutions, since this is implemented in the hardware of the device. This is another reason the milling and drilling station were chosen for the analog outputs. Even if the maximum current is rated for 10V, it will still run the motors on the factory. However, they will run slower than they otherwise would have, using the full 24 volts rated for the system. Passing the full 24V to the motors via the analog outputs, would require an interfacing layer and a custom circuit similar to the PCB made for the Raspberry Pi.

The solution in this case was to move the integer value through MOVE blocks as seen in Figure 22. When activated, the Move block pass the value from the input box into the output variable.

The example above was written as a simple test. Although this hasn't been implemented in the main factory code, it can easily be implemented into the ladder diagram.

### 5.3.4 Software driver for the programmable I/O

Another solution to lack of I/O, is to use the remaining digital pins on the Arduino PMC. Although these are programmable, per beginning of this Master's thesis these software drivers are not implemented into OpenPLC. Therefore there was a need to write drivers to be able to use the programmable I/O. Only the code that has been highlighted in the figures are written for this thesis. The code file is handed in with the thesis, but most of the code is written by other people

---

for OpenPLC.

The first thing that was done, was to look at the original driver for the Machine control. The driver file for the Arduino Portenta H7 is found using the following path:

```
C:\Users\user\OpenPLC_Editor\editor\arduino\src\hal\machine_control.cpp
```

The digital and analog I/O pins are initialized first. Which means that the programmable I/O has to be initialized before starting to use these. Luckily, all of the Arduino libraries are open-source. Which means codes should be accessible online to see potential ways of initializing the programmable pins. After doing some research, the official Arduino library for the Portenta Machine Control were found on **GitHub**. This contained all of the source code for the hardware drivers for the PLC itself. After looking through the files, the setup files were found on the following url:

[https://github.com/arduino-libraries/Arduino\\_MachineControl/blob/master/src/Arduino\\_MachineControl.h](https://github.com/arduino-libraries/Arduino_MachineControl/blob/master/src/Arduino_MachineControl.h)

This file is a header files that listed the functions that were implemented for the different types of pins. After looking at the OpenPLC code again, the realisation came that this library is directly imported into OpenPLC. This makes the programming easier, since it is possible to just call the functions. After looking at the official Arduino code. It was found that the the programmable pins were callable with the following variable:

```
digital_programmables
```

After finding this, it was found that the pins could be initialized using the functions from the library directly, and call them from the header file. This means that the initializing of the programmable pins can be done with:

```
1. digital_programmables.init()
2. digital_programmables.setLatch()
```

The code written in the cpp file for this is added in Figure 23. This is also how the programmable pins are initialized in the official Arduino libraries. If the pins fail to initialize, the print will write the string, letting the user know they failed initializing.

```
23 //Initializing the programmable I/O
24 if (!digital_programmables.init())
25 {
26     Serial.println("GPIO initialization failed");
27 }
```

Figure 23: Initializing the digital programmables

This is exactly the same process as how the other pins on the board are initialized in OpenPLC. The first line is initializing the pins, while the second line sets the **setLatch** function. The setLatch

---

function sets the pins into retry mode during overcurrent, which shields the pins from getting damaged by a possible spike in current. This function is used in Figure 24.

The next step, was to change the pins into a state which made them accessible for usage. This is done with the following line.

```
digital_programmables.set(pinNumber, State)
```

This is exactly the same way that the digital pins are configured, except the programmable pins can be selected to have two states: **Input** and **Output**. Another difference is the lack of the `setAll()` function. On the programmable pins, this function is not written in the library. Therefore there is a need to configure each pin individually. A picture of how these functions were written is seen in Figure 24. These lines of code will initialize the digital programmables if the macros `NUM_DISCRETE_OUTPUT` and `NUM_DISCRETE_INPUT` exceeds 8 bits.

```
39 //Setup programmable digital outputs
40 if (NUM_DISCRETE_OUTPUT > 8)
41 {
42     digital_programmables.setLatch();
43     for (int i = 0; i < NUM_DISCRETE_OUTPUT - 8; i++)
44     {
45         digital_programmables.set(i, OUTPUT);
46     }
47 }
48
49 //Setup programmable digital inputs
50 if (NUM_DISCRETE_INPUT > 8)
51 {
52     digital_programmables.setLatch();
53     for (int j = 0; j < NUM_DISCRETE_INPUT - 8; j++)
54     {
55         digital_programmables.set(j, INPUT);
56     }
57 }
```

Figure 24: Code initializing the programmable pins in OpenPLC

Another challenge was to find out what the address for the pins are. Instead of the pin number, the pin needs to be called directly on the address. This address will also change depending if outputs or inputs are going to be used. After doing some research and looking at the source code of Arduino PLC IDE, it was found that the programmable pins were callable by accessing the following addresses:

```
IO_READ_CH_PIN_XX
IO_WRITE_CH_PIN_XX
```

The addresses can be separated into three parts that changes based on the pins that are accessed. The **IO** part is addressing the programmable I/O rails. The next part indicating **READ** and **WRITE** is telling the pin if the pins are taking inputs or giving outputs, where **READ** indicates



---

an input and **WRITE** indicates an output. The last part, marked as "XX" indicates the pin number. Pin numbers are ranging from 0 - 11 on the programmable I/O rail. To use the pins, there is a need to write the pin address into the **I/O config** and address it directly as seen in Figure 36. It needs to be written into the form which is stated above.

To be able to address the pins directly like this, the device drivers for OpenPLC had to be changed into the driver code. Trying to tell the Portenta H7 that it had more than 8 output pins crashed the device, and the PMC needed to be reset. Making this functionality possible needed to be done in the hardware driver in *OpenPLC*.

Therefore there was a need to program a handler for this exact scenario. How this is programmed in the original script, the information is listed into an array, and sent to each individual pin using a for-loop. The code that's been added, sends the outputs to the programmable pins if the number of outputs exceeds the length of the array. The same function is added for the input code.

```
89 void updateInputBuffers()
90 {
91     for (int i = 0; i < NUM_DISCRETE_INPUT; i++)
92     {
93         if (bool_input[i/8][i%8] != NULL)
94             *bool_input[i/8][i%8] = digital_inputs.read(pinMask_DIN[i]);
95     }
96
97     //Adding a similar one for the initialization of the digital input class:
98     //Pins are addressed by IO_READ_CH_PIN_XX
99     for (int i = 0; i < NUM_DISCRETE_INPUT - 8; i++)
100    {
101        if (bool_input[3][i%8] != NULL)
102            *bool_input[3][i%8] = digital_programmables.read(pinMask_DIN[8 + i]);
103    }
104
105
106    for (int i = 0; i < NUM_ANALOG_INPUT; i++)
107    {
108        if (int_input[i] != NULL)
109            *int_input[i] = analog_in.read(pinMask_AIN[i]);
110    }
111 }
```

Figure 25: This code updates the inputs for OpenPLC. Line 97 to 104 is newly written for this thesis. The rest is un-edited driver code from OpenPLC.

These functions can be seen in Figure 25 for the input variables and Figure 26 for the output variables. As can be read from the code, the device gets a bool input from the factory and read them using the **digital\_programmables** set up for this task. From there it finds the pins that is addressed as IX3.y and send the information further, where it will find the address of the pin written in IO config. The same happens in output, where the pins will be addressed as QX3.y.

The huge benefit of doing it this way, is that with proper implementation and an update to the OpenPLC program, it is now possible to use the programmable I/O pins on the Portenta Machine Control. Which does not at the time, have official OpenPLC support.

## 5.4 Arduino PMC with Arduino PLC IDE

A benefit of using a proprietary program like the PLC IDE, is that all of the program is structured and tailored around one specific unit. This means all of the settings and structures in the program is already set for the device. On programs like *OpenPLC*, the settings usually have to be changed

---

```

113 void updateOutputBuffers()
114 {
115     for (int i = 0; i < NUM_DISCRETE_OUTPUT; i++)
116     {
117         if (bool_output[i/8][i%8] != NULL)
118             digital_outputs.set(pinMask_DOUT[i], *bool_output[i/8][i%8]);
119     }
120
121     //Put in another one for the programmable output class
122     //(setting the first programmable pin to output)
123     //Pins are addressed by IO_WRITE_CH_PIN_XX
124     for (int i = 0; i < NUM_DISCRETE_OUTPUT - 8; i++)
125     {
126         if (bool_output[3][i%8] != NULL)
127             digital_programmables.set(pinMask_DOUT[8 + i], *bool_output[3][i%8]);
128     }
129
130     for (int i = 0; i < NUM_ANALOG_OUTPUT; i++)
131     {
132         if (int_output[i] != NULL)
133             analogWrite(pinMask_AOUT[i], ((float)*int_output[i] / 1000.0));
134     }
135 }

```

Figure 26: This code updates the outputs for OpenPLC. Line 121 to 128 is newly written for this thesis. The rest is un-edited driver code from OpenPLC.

to work for each specific device. This changes how the code is uploaded to the unit, in addition to the structure of the code.

Another thing that was different in the Arduino PLC IDE is how the Portenta H7 is programmed. In Arduino PLC IDE, each line is set from the start, forcing the user to follow the standards of ladder logic. Where one of them is that the coils needs to be at the end of the code. OpenPLC does not force anything and it is entirely up to the user how the code is written.

## 5.5 Arduino MEGA with OpenPLC

Another controller that was tested for this thesis was the Arduino MEGA. The reason this was tested is that it has very similar structure, and uses the same software as the Arduino UNO. Meaning that if the code is working on the Arduino MEGA, the code will also work most likely for the UNO. The main benefit of using the Arduino MEGA over an Arduino UNO, is the amount of I/O available. The Arduino UNO doesn't have enough I/O to run the entire factory, therefore the MEGA works better for prototyping purposes.

### 5.5.1 Testing the Arduino MEGA

To run the factory with the Arduino MEGA, the circuit board that were developed in the project assignment needed to be utilized. This hardware was developed to work with several micro controllers outside of the Raspberry PI. First, the outputs were tested. The Arduino MEGA was capable to run the motors via outputs. This was expected due to the Arduino MEGA giving out similar amounts of voltage on outputs as the Raspberry PI.

The inputs however, was a different case entirely. The wires on the circuit that were connected to the input pins on the Arduino MEGA were tested. However, the optical sensors on the factory were not sending the correct signals to the Arduino MEGA. At first, it was thought that the problem

---

was electrical noise. All electrical wires generate an electric field around the wire when powered up. This is called *impedance*.

The reason this was thought to be a problem, was that the Arduino reacted even when the wire going into the input wasn't connected to anything. According to *Prof. Amund Skavhaug*, the Arduino MEGA and UNO should be very robust against influence by impedance.

After this was found, a multimeter was used to measure the voltage of the circuit going into the input of the Arduino. When the optical sensor was unblocked, the voltage of the signal was 1.2 volts. When the optical sensor were blocked, this value dropped down to 0.8 volts, making a difference of 0.4 volts. Considering the digital pins for the Arduino was used, it seemed reasonable to check the voltage needed to switch the states of the input pins. These values were found to be less than 1.5v for the **LOW**-state and over 3.3 volts for the **HIGH**-state from the Arduino UNO datasheet [25].

This means that the voltage that the Arduino takes from the circuit board makes the digital pins go into a "floating state", meaning the I/O pin is hovering between an off and on state. This leads to uncontrollable, random behaviour. To fix this, the circuit board needed to be updated.

### 5.5.2 Testing and fixing inputs

After doing a detailed search on the circuit that was made for this project, the problem was thought to be that the resistors was too large. With the 240k resistance that are placed on the 24v side of the board, there is just 0.01 mA going over the optocouplers. This might be the reason the inputs behaves uncontrollable. This is later addressed and explained in Section 5.6.

## 5.6 Updating the circuit board

The reason the Arduino didn't work properly, was due to the voltages being inconsistent. As stated in Section 5.5, the LOW voltage was at 0.8v, while the HIGH voltage was at 1.2v. The Raspberry PI worked fine with these voltages earlier, but the input pins on the Arduino MEGA were more sensitive for the voltage.

The first thing that was done, was to make new input circuits using breadboards. This way, it is easier to measure the voltages in the circuit to become more familiar with the circuit, as well as making it easier to implement solutions without destroying or tampering with the existing circuit board. A picture of this test-setup can be seen in Figure 27.

At first, the voltages on the high-current side of the circuit was measured. The circuit was coupled up to the factory, sending the necessary power to be able to activate the sensors on the factory. After this, the input channel on the circuit was connected to pin 9. Pin 9 is the terminal for optical sensor number 2. This is the sensor right before the first corner of the factory.

At first glance, the measured voltages seemed fine. When the input channel opened, the voltage sent in to the optocoupler was 24V, while the voltage was 1.1V when the the light sensor changed state. This were values that were realistic for the application. Seemingly, there was nothing wrong with the input circuit, other than the voltage on the low-voltage side being to low. This indicated that the optocouplers didn't fully open with the power provided.

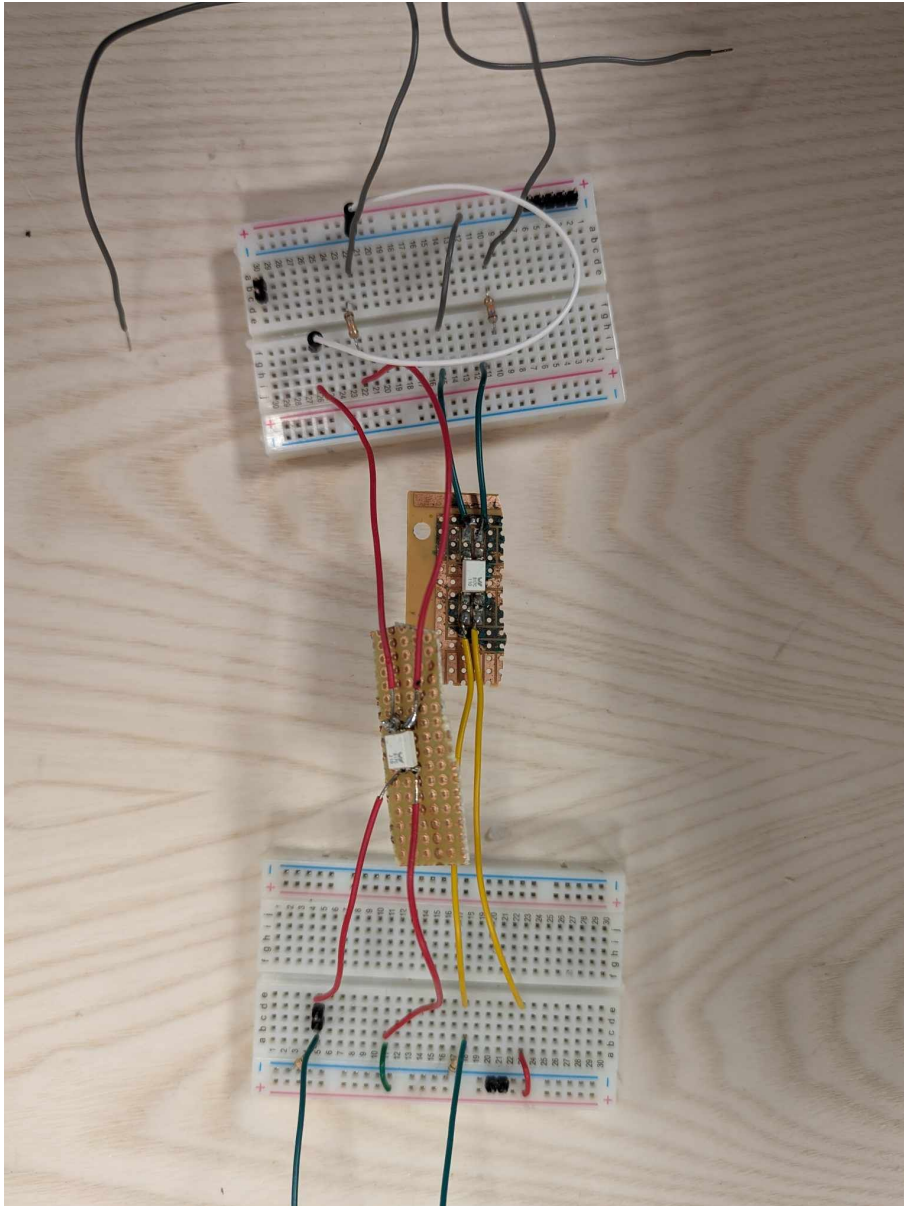


Figure 27: The breadboard circuit used for testing the input circuit

---

This lead to two major changes in the circuit:

- Change the resistance on the high-voltage side
- Add a pull-down resistor to the low-voltage side

### 5.6.1 Adding a pull-down resistor

Optimally, when the optocouplers are not letting any power through the gates, the voltage should read at 0V. This tells that the gate on the optocoupler is fully closed, and no current should be able to go through the diode on the gate. At this point in time, the current reads 0.8 volts when shut. This is not sufficient to make a clear border for the Arduino to switch the digital state of the input pins. Ideally, the voltage should be 0 volts when the sensor is not activated. This effect is accomplished with adding a pull-down resistor to the circuit.

According to *EEPower*, a pull down resistor is a resistor that pulls the pin down to logical low value [26]. The resistor is a normal resistor, which is connected between the pin on the microcontroller, and the ground. Similarly, a pull-up resistor is pulling the voltage up to a logical high value. This however, means adding a resistor before the wire going into the micro-controller. A picture of a pull down resistor is seen below in Figure 28.

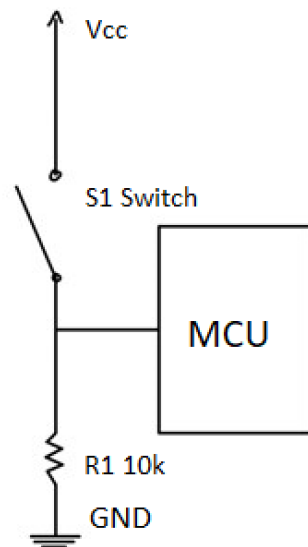


Figure 28: A drawing of a pulldown resistor

Both are used to eliminate the floating state of the pins on an Arduino. Pull-up resistors are implemented on the Arduino itself from the factory [25]. However, for this application, there is a need to add pull-down resistors instead. These are not found natively on the Arduino, meaning this has to be implemented in the circuit itself.

The next step is to choose the value of the resistor added to the circuit. According to *EEPower*, this is limited by primarily by two factors [26].

- Power dissipation

- 
- Pin voltage when the switch is open.

If the resistance value is too low, the current going through the pull-down resistor will be large, heating up the micro-controller, and using unnecessary amounts of power when the optocoupler is closed. This is due to the electric power being greater with a lower resistance.

The second dependency for the resistance value is decided by the pin voltage when the switch is open. In this case, this is the 3.3 volts that is provided by the microcontroller. If the pull-down resistance is too high, there is a possibility that the voltage is dragged down when the pin is open. This means that there is a risk of getting into the infamous floating state. If the resistance is high enough, it won't even change state at all.

Typically, pull-up and pull-down resistors will have a value between 5-10k depending on different factors. A 5k resistor was decided to be a good starting point. This resistance value gave 3.3 volt on the high end, and 1.2 volt on the low end. This won't be sufficient for the application due to the pin going into the "floating" state mentioned earlier. A higher value resistor will fix this problem. After changing to a 10k resistor, the voltages switch between around 0 volt and 3.2 volt. Exactly as intended.

### 5.6.2 Changing the resistances

The total resistance of the resistors on the high-voltage side of the circuit were measured, and it was found that the total resistance equalled  $118k\Omega$ . This was the resistance found after calculating the total resistance of the two resistances connected in series with  $100k\Omega + 18k\Omega$ .

Ohms law is the standard formula for calculating the relations between amperes, volt, and resistance. This formula was used to calculate the ampere of the power that opens the optocoupler when the signal is activated. This equals to

$$U = R \times I.$$

$$I = \frac{U[V]}{R[\Omega]} = \frac{24V}{118000\Omega} = 0.0002A = 0.2mA$$

An electric current of 0.2 mA is not sufficient to open the optocouplers completely as seen in the datasheet [27]. Therefore, it is likely that this is the reason the voltages on the low-voltage side of the circuit isn't distinct enough for the Arduino to read properly and the pins enter the floating state that were seen in the project.

After this, some testing began. The old resistors were removed and replaced with one resistor. The circuit would work the same without having two separate resistors, therefore one of them was removed. At first, the double resistor setup were replaced by a single resistor with  $47k\Omega$ . Using Ohm's law, this equals to:

$$I = \frac{U[V]}{R[\Omega]} = \frac{24V}{47000\Omega} = 0.0005A = 0.5mA$$

Even after this change, the voltage still indicated that the optocouplers didn't fully open. Most likely, the current is still not sufficient enough to open them properly, meaning there is a need to try a new resistor with an even lower value. The next resistor that was implemented in the circuit had a value of  $18k\Omega$ . This gives a current of  $1.3mA$ , using the same method as described above. After activating the sensor, the voltage was again measured on the low-voltage side of the circuit.

---

This equalled now to roughly 0 volts as a default, and 3.2 volts under activation of the sensor.

The output circuit developed in the project thesis was also looked at. The main reason for concern was again, the low-voltage side of the circuit. The resistances in place is  $220\Omega$ . These paired with the 5 volts provided by the Arduino will equal to roughly  $23mA$ . Looking at the datasheet [27], the optocoupler can handle up to  $60mA$  on that pin. Therefore, it was decided that the existing resistors would be sufficient.

These are the results that originally were desired for the circuit, and now the circuit works as intended, also for the Arduino UNO and MEGA. This will now work for all micro-controllers similar to Arduinos in terms of voltage and pins. Most likely, it would work on other micro controllers as well. On the next page in Figure 29, a schematic of the final iteration of the circuit board is added.

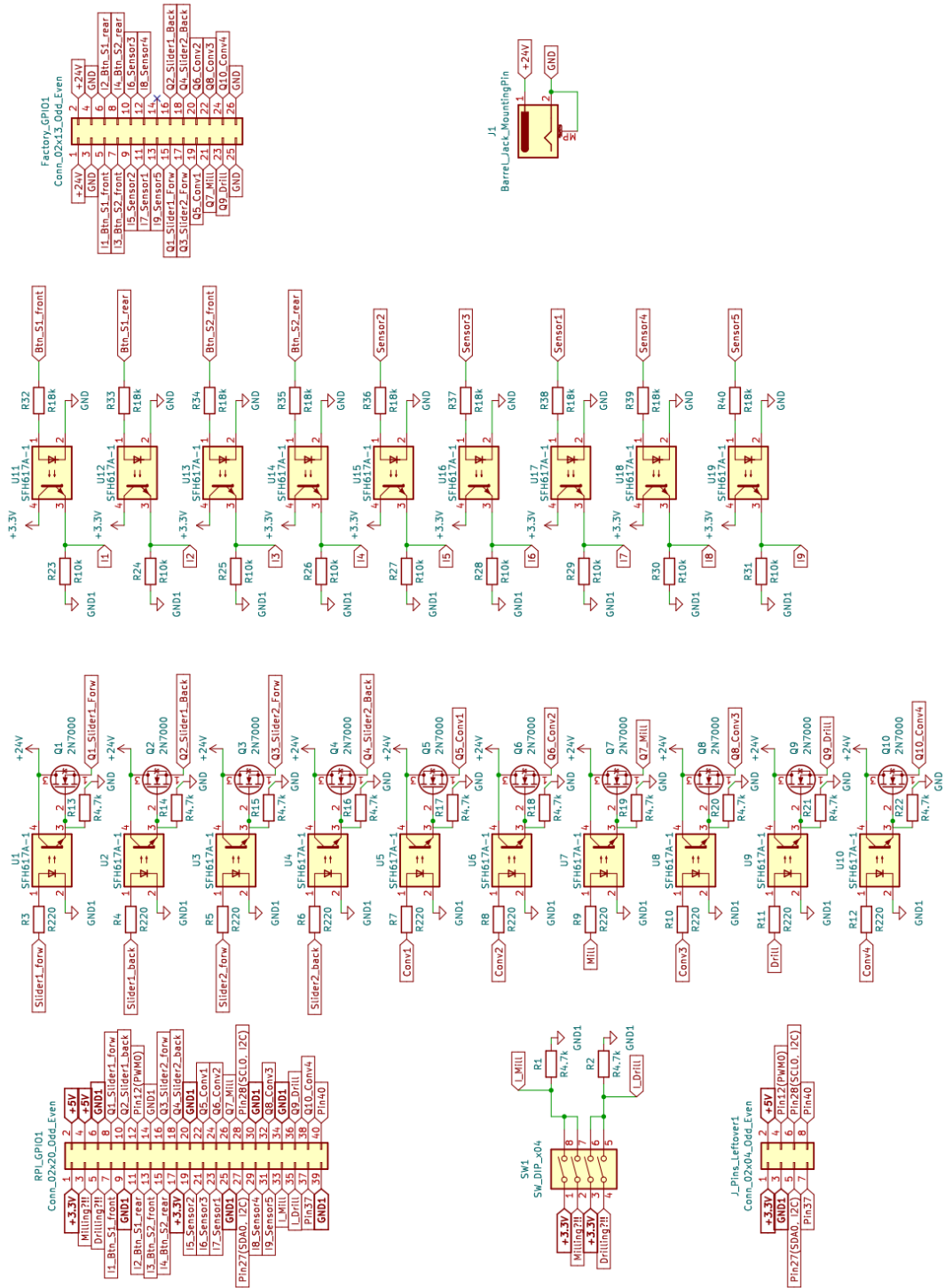


Figure 29: The schematic of the final PCB design

## 5.7 Connecting micro-controller to the factory

This system, as described earlier, is designed to become an assignment for Industrial Mechatronics. One of the key factors that needs to then be implemented, is to make the system easy to use. The learning outcome will not be great if the system is not designed with ease of use in mind. Therefore, there's added ribbon cables from the factory onto the micro controller. On the Raspberry Pi used



---

in the project thesis, the ribbon cables were directly connected to the I/O pins from the circuit board onto the Raspberry Pi. There was also ribbon cables connecting the PCB to the factory.

### 5.7.1 Connecting wires directly

Using different micro controllers requires different ways of connecting them to the factory. The easiest way for the developer, is just make users plugging the wires themselves between the factory and device. This is however sub-optimal for the users of the devices. Plugging wires directly can be troublesome for reliability. Wires can for example break internally, making them unusable. Another problem is ensuring the users plug the wires correctly. If the main objective of assignments is to learn students how to use PLC code, the system needs to be designed robustly to avoid potential problems of the system.

### 5.7.2 Hardware shields

To make the system as user friendly as possible, it's desired to make a hardware shield to use over the different micro controllers. This way, the user can easily put on the shield as a hat, and all the wires and connections are up and running using only a single cable. Due to the circuit board already having a connection that correlate directly to the Raspberry Pi, there is no need for another connection than a ribbon cable. But custom shields have to be made for the other controllers.

In addition to the Raspberry Pi, there's been used three other micro controllers for the project. The Arduino UNO, Arduino Pro Portenta Machine Control and Arduino MEGA. These Arduino devices would need custom shields to be easily connected to the factory. Preferably using a ribbon cable like the Raspberry Pi.

### 5.7.3 Hardware shield on Arduino PMC

Since the Arduino PMC doesn't need the circuit board to run the factory, connecting the PMC to the factory should be easy to enough. The unit features proper connections for plugging the wires into the device. Each I/O pin features a lock, which means that the user can be sure the wires are properly connected to the device.

Therefore, making a hardware shield with male connections like a normal Arduino Shield is impossible. Either way, each cable has to be connected individually into the device. However, the other ends can be connected to a common point. All of the lose ends of the factory wires are connected to this common point using soldering. This should eliminate possible sources of error, when this connection can be mounted using a ribbon cable. This shield should be robust enough for the application, although it is sub-optimal compared to a normal hardware shield made for other devices. Still, a hardware shield for the PMC was developed to increase the reliability of the system. This hardware shield can be seen below in Figure 30 connected to the device.

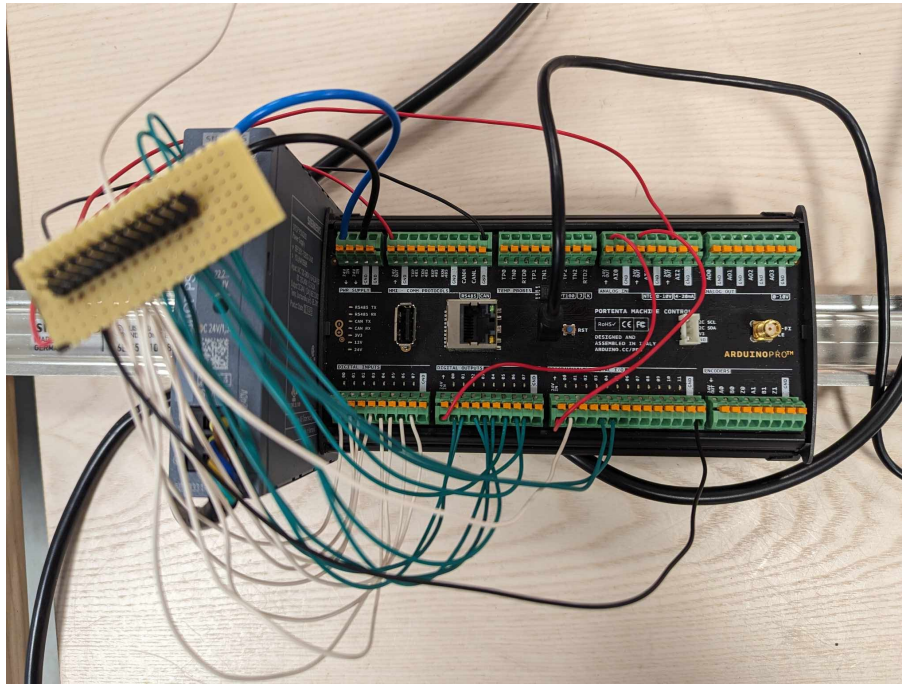


Figure 30: The hardware shield made for the Arduino PMC

#### 5.7.4 Hardware shield Arduino MEGA

The hardware shield on the Arduino MEGA is probably the easiest to make. Compared to the Arduino UNO, the MEGA have all of the required I/O pins from factory. This means that the shield will only be in one single part. Making it easy to make and implement onto the micro controller. This could even become a two layer circuit board if desired. A fast prototype was made at the end of the project to make testing easier for the device. This shield has female headers to connect easily to the PCB, and male headers to fit onto the Arduino MEGA. Between these there are wires soldered on to make the correct connections to each of the pins on the Arduino MEGA and the PCB. This hardware shield can be seen in Figure 31 on both sides. The white wires in Figure 31a indicates the inputs, the black ones is the outputs. The three wires that stand alone is one red wire for 3.3 volts, one for 5 volts and the black one for ground.

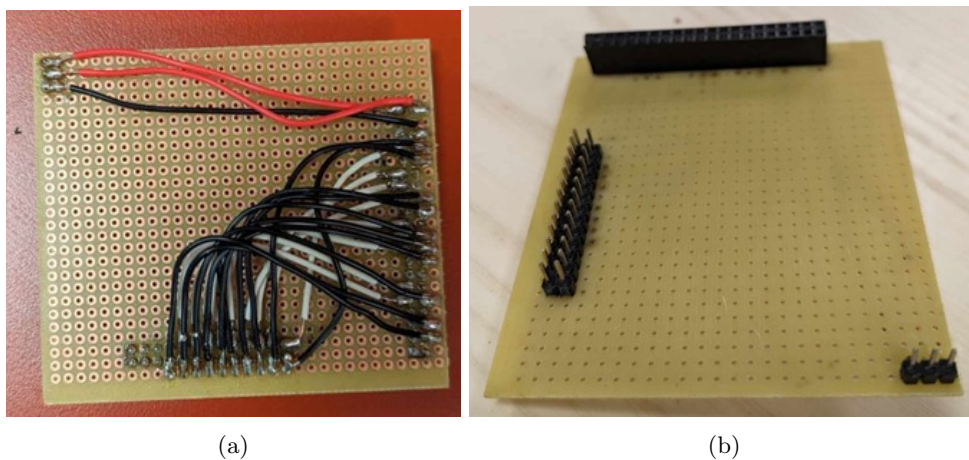


Figure 31: Both sides of the hardware shield made for the Arduino MEGA

---

### 5.7.5 Hardware shield Arduino UNO

The hardware shield for the Arduino UNO would be a bit different. Due to the UNO not having enough I/O from start, it means that there's a need for an I/O expander for this to work properly. Although this can ideally be a part of the shield, it makes for more overlaps on the shields itself. A shield like this would be similar to the one seen in Figure 31, although added hardware for an I/O expander, that needs to be connected both to the Arduino UNO as well as the PCB.

---

## 6 Software and Simulation

The main purpose of this project, is to make a solid platform for students to learn about Programming Logic Controllers. This includes the programming language used for these controllers. To do this, there are two potential IDEs that can be used for code development. The two programs are called *OpenPLC* and *Arduino PLC IDE*. Background theory for these softwares are written in Section 2. Both of these IDEs need to be downloaded to PCs. In addition, there is some extra software that needs to be downloaded on the Raspberry Pi. Both of the code developing softwares are easy to download and specific steps to download them is mentioned in Section 3.1. Most of the code that is developed on OpenPLC has been developed during the preceding project thesis. While the code developed on Arduino PLC IDE is new for this specific thesis, the following subsections in this chapter are taken directly from the preceding project thesis:

- Section 6.1
- Section 6.2

The rest of the sections in this chapter are newly written for this thesis.

### 6.1 Connecting to Raspberry Pi

To connect and control a Raspberry Pi, a webserver provided by OpenPLC was used. The guide from the official OpenPLC site [28] was used to connect the server and the Raspberry Pi. The first thing that needed to be done, was to connect the Raspberry Pi to *Eduroam*. A guide on how to do this is provided in [20]. The guide is also provided in Appendix B. After successfully connecting the Raspberry Pi to the *Eduroam* network, the IP-address of the Raspberry Pi needs to be found. This can be done by typing the following line in the terminal:

```
$ ifconfig
```

The IP-address is found under *Wlan0*.

A downside with using this webserver based system, is that the IP address of the Raspberry Pi is not static on the *Eduroam* network. There were multiple occasions where the IP-address changed while running the program. One fix for this, is to get a static address for the Raspberry Pi. This would be a stable fix on a private network. Unfortunately this is certainly not an option for a public network like *Eduroam*, where admin rights for the network is needed. The easiest solution by far is to simply check the IP address regularly.

To connect to the OpenPLC webserver, type in the IP-address on port 8080 in a browser. When successfully connected, the code is sent to the microcontroller via the webserver. From there, the different code files are sorted in a list and saved. OpenPLC saves the programs permanently, which makes it easy to run different code files without the need of uploading the code every time.

---

## 6.2 Programming and language

The code for this project has been mainly written in two languages. As described in earlier chapters, *OpenPLC* is using the IEC 61131-3 standard for PLC programming. Whichever of the 5 officially approved languages there is a desire to use, the code is compiled down to a ST-file. This is an abbreviation for Structured Text, and is a high-level programming language derived from *Python*.

It was decided to program the mini-factory in two different languages, Ladder Logic (LD) and Sequential Function Chart (SFC). Each of the languages have different structures and capabilities, but commonly, they are both widely used in the manufacturing industry. By programming in these different languages, it's also easier to decide what language to use in a later stage. Another option for the project could be to make scripts in both *Python* or *C*. However, this was later deemed irrelevant for the project due to the relevance of the industrial standard languages.

Since the programs in *OpenPLC* are compiled into an ST-file, it doesn't matter which of the 6 standard approved languages is used. The ST-file is uploaded to the webserver based software on the Raspberry Pi, as described in Section 6.1. From there, the program is compiled into the programming language *C*, which is the programming language mainly used on Raspberry Pis and other microcontrollers. To make the programming easier, the factory was divided into parts, as can be seen in Figure 32.

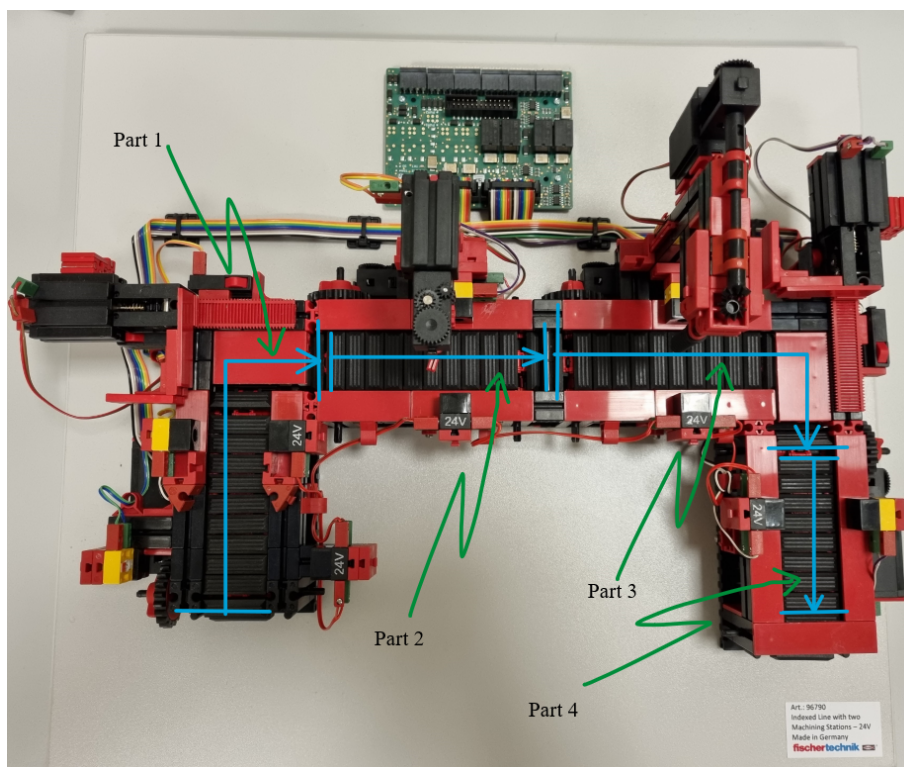


Figure 32: The Factory in bird view, divided into four parts

## 6.3 Programming in Arduino PLC IDE

Despite supporting 5 PLC programming languages like OpenPLC, there's some differences in both *User Interface (UI)* and differences in ways to program. The differences in both of the programming

---

languages will be elaborated further down with an addition of the code that's written in the Arduino PLC IDE.

In Arduino PLC IDE the first thing that needs to be done is making a new project folder. In this folder, it's possible to add variables and have easy access for each file. These can also be written in different languages. When creating a project, a "main" file is generated. This file can be used for writing *Structured Text (ST)* code. This is one of the standard languages for PLCs.

The project will be seen as a tree on the left side of the screen, with an overview of the different contents as seen in Figure 33. Here is where the programs will appear. When making a program in Arduino PLC IDE, a text box will appear, asking for name of the file and the language. When the program is created, the *UI* will change depending on the program language selected. For example, the Ladder Diagram *UI* appears with one line of the network, which includes a relay and a coil. The SFC *UI* on the other hand, starts with an empty grid.

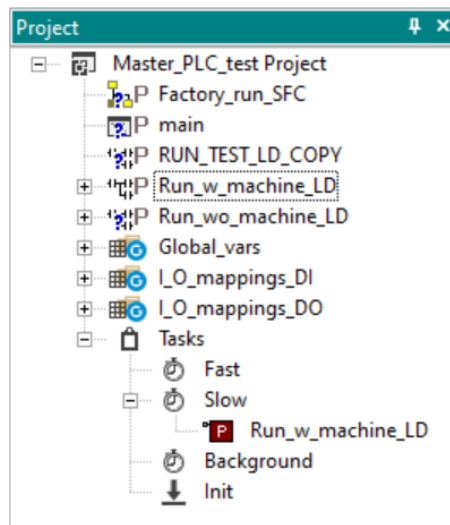


Figure 33: The resource tree from Arduino PLC IDE

## 6.4 Programming in OpenPLC

Running the factory with the code in OpenPLC was really straight forward. Minimal changes had to be done for the Ladder Diagram code that were developed for the Raspberry Pi in the project assignment. At first, the last line of the code were changed to account for an analog input, due to the Arduino Portenta Machine Control H7 not having enough digital I/O to run the factory.

The variables set up to work with the factory on a Raspberry Pi can be seen in Figure 34. These variables are unique for the PCB and can be used for any type of PLC code written. However, the autogenerated delay block variables will differ depending on the specific code. The table shown in Figure 34 is from the code that can be seen in Figure 35.

The variables needs to use these addresses with the Raspberry Pi, because of how the PCB was designed. If it's desired to rebuild the PCB with different locations, the documentation of the pin addresses for the Raspberry Pi or other microcontrollers are added on OpenPLCs official site [29]. A table of the different pin addresses and functions of the Raspberry Pi is also added in Appendix E.

#	Navn	Class	Type	Location	Initial Value	Option	Documentation
1	I_Btn_S1_F	Local	BOOL	%IX0.2			Button to detect Forward position of slider 1
2	I_Btn_S1_B	Local	BOOL	%IX0.3			Button to detect Back position of slider 1
3	I_Btn_S2_F	Local	BOOL	%IX0.4			Button to detect Forward position of slider 2
4	I_Btn_S2_B	Local	BOOL	%IX0.5			Button to detect Back position of slider 2
5	I_Sensor1	Local	BOOL	%IX1.0			First sensor to start The factory
6	I_Sensor2	Local	BOOL	%IX0.6			Second sensor to detect end of conveyer1
7	I_Sensor3	Local	BOOL	%IX0.7			Third sensor to detect front of milling
8	I_Sensor4	Local	BOOL	%IX01.1			Fourth sensor to detect front of drilling
9	I_Sensor5	Local	BOOL	%IX1.2			Fifth sensor to detect the end
10	I_Mill	Local	BOOL	%IX1.3	0		Input to check if the part is supposed to be milled
11	I_Drill	Local	BOOL	%IX1.4	0		Input to check if the part is supposed to be drilled
12	Q_Slid1_F	Local	BOOL	%QX0.0			Slider 1 forward drive
13	Q_Slid1_B	Local	BOOL	%QX0.1			Slider 1 backward drive
14	Q_Slid2_F	Local	BOOL	%QX0.2			Slider 2 forward drive
15	Q_Slid2_B	Local	BOOL	%QX0.3			Slider 2 backward drive
16	Q_Conv1	Local	BOOL	%QX0.4			Conveyer 1 at the start
17	Q_Conv2	Local	BOOL	%QX0.5			Conveyer 2 at the milling station
18	Q_Conv3	Local	BOOL	%QX0.7			Conveyer 3 at the drilling station
19	Q_Conv4	Local	BOOL	%QX1.1			Conveyer 4 at the end
20	Q_Mill	Local	BOOL	%QX0.6			The milling station
21	Q_Drill	Local	BOOL	%QX1.0			The drilling station
22	Deley	Local	TIME				
23	TON0	Local	TON				
24	TOF0	Local	TOF				
25	TON1	Local	TON				
26	TOF1	Local	TOF				
27	TOF2	Local	TOF				
28	TON2	Local	TON				
29	TON3	Local	TON				
30	TOF3	Local	TOF				
31	TON4	Local	TON				
32	TOF4	Local	TOF				
33	TON5	Local	TON				
34	TOF5	Local	TOF				

Figure 34: The PLC code variables

## 6.5 Raspberry PI specific changes; PCB

To make the factory work with the specific Raspberry Pi used in the project assignment, the location variable for **Q\_Slid2\_F** on row 14, needed changing from `%QX0.2` to `%QX1.2`. The corresponding pin of the Raspberry Pi used in the project was defective, therefore a jumper cable was added between pin 16, and jumped to the only leftover pin.

Pin 16 was broken, and the jumper cable were connected to pin 40 on the old PCB. The factory will not work with a Raspberry Pi if more than one of the output pins are broken. There's a total of 11 pins on the Raspberry PI, meaning 10 of these has to work to be able to use all the functionality of the factory.

## 6.6 Connecting Arduino PMC to OpenPLC

In an earlier chapter Section 5.3.4, it was described how the programmable pins on the Arduino PMC was unlocked, and able to be used together with OpenPLC. However, being able to use this with the factory is a bit complicated. Therefore, this chapter is going to show a user how to properly be able to unlock the functionality, by following the steps given in this subsection. This guide is showing how to do this on a Windows computer, but an experienced user should be able



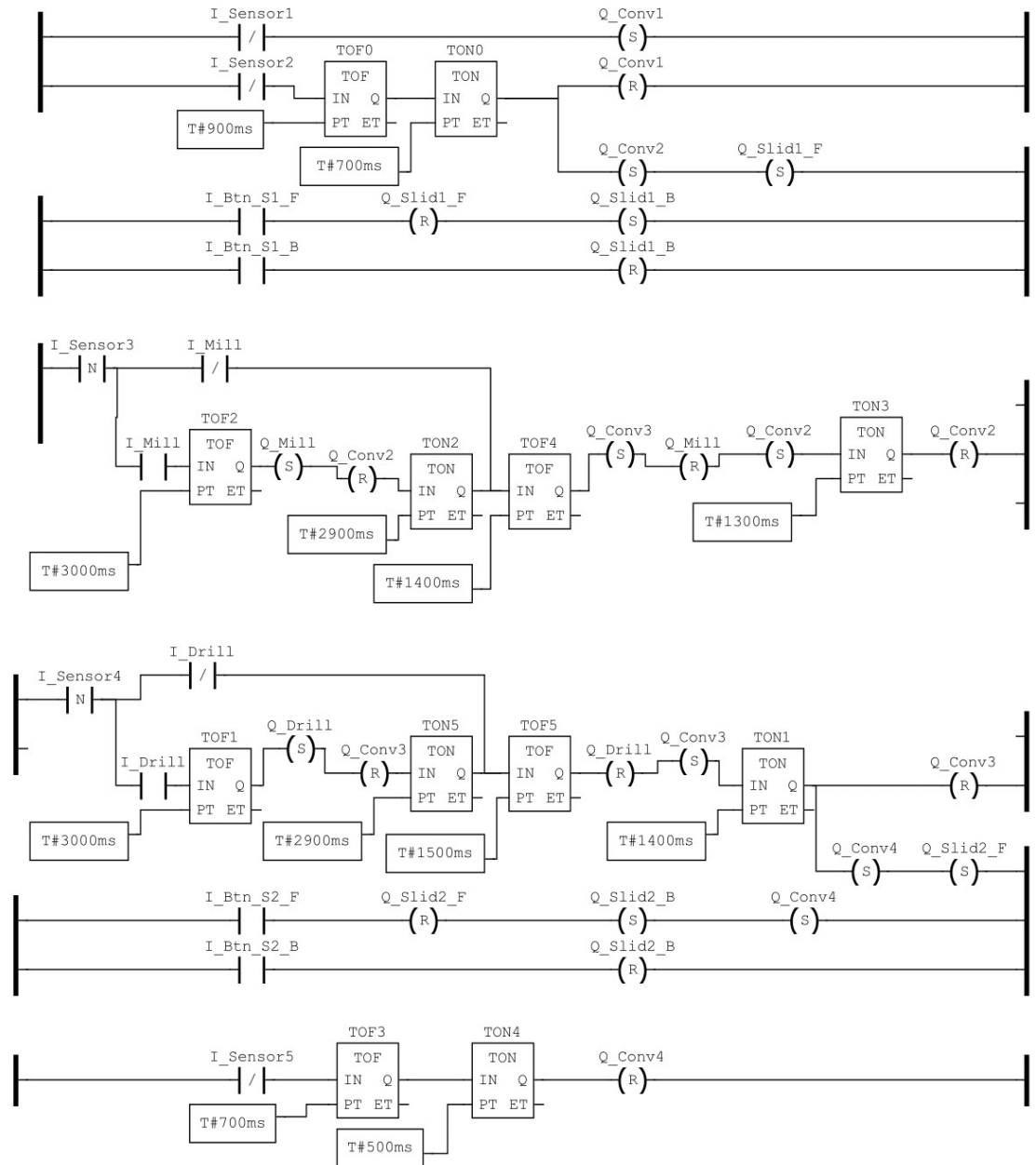


Figure 35: The complete Ladder Diagram

to the same steps in a different Operating Systems. This consists of mainly 3 steps.

- Update OpenPLC driver
- Addressing the programmable pins
- Configure the pins before compiling

Step 1 is to update the driver code into OpenPLC. This step is going to update the driver file in OpenPLC for the Arduino PMC. This is done to ensure that the program understands the addressing of the pins when configuring the pins in the program. If OpenPLC is installed regularly, this file is going to be found at this path:



---

```
C:\Users\user\OpenPLC_Editor\editor\arduino\src\hal\machine_control.cpp
```

To ensure that the new file working as it should, the new file must have EXACTLY the same name as the one it replaces. Therefore, if asked if the old file is going to be overwritten, choose yes.

Another disclaimer is if the program is updated, the file have to be overwritten again. When updating OpenPLC, the newest version of the program is gotten directly from *GitHub*. The current version (version 3) does not support the programmable pins on the device. Therefore, the functionality will be lost again, when updating the program.

Step 2 is done in OpenPLC itself and concerns the pin addresses. This step is a bit more intricate than step 1. The first thing that needs to be done is to decide what addresses that is going to be used for the different functions on the factory. Due to the input sensors being addressed first on the board, this sensor was chosen to be the first pin of the programmable pins.

These pins are addressable by either "IX3.y" or "QX3.y". Since the programmable pins have more capabilities than the regular ones, it is important to tell the device what capabilities that is desired from each pin. "I" in this case represents inputs, and outputs are represented by "Q". Where "y" is indicated to the pin number on the addressable pins, going from 0-7. In the case of this project, "IX3.0" was assigned to **L\_Sensor5**. The "QX3.3" and "QX3.4" addresses were connected to the milling and drilling machine on the factory.

Step 3 is addressing the pins before compilation. The normal pin configuration of the device is added into the program. Therefore, it's important to tell the program what pins to configure. This is done in the I/O config on OpenPLC. If done incorrectly, the program might crash or corrupt the software on the Arduino PMC. Correct implementation should look like something seen in Figure 36.

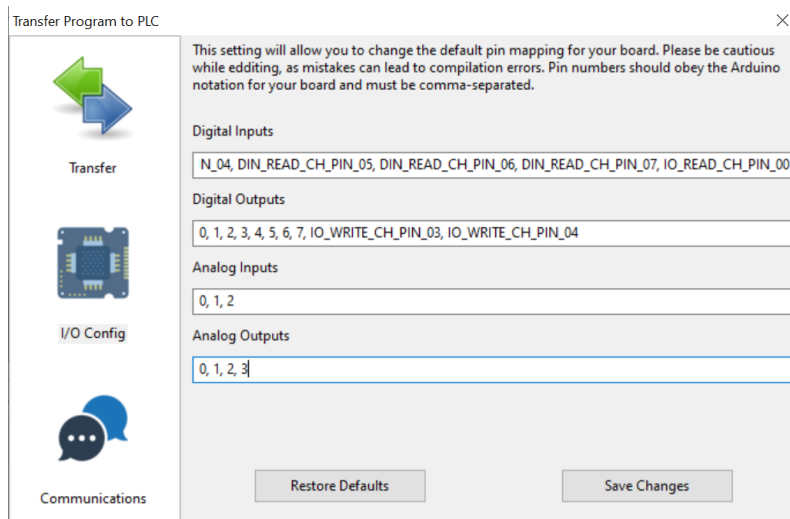


Figure 36: IO config for OpenPLC

---

## 6.7 Compiling and running code in Arduino PLC IDE

When the user is finished developing a program, the program needs to be compiled. This is done by pressing a button, like most other programming IDE's. However, running the program on the PLC IDE is a bit different compared to OpenPLC. In OpenPLC, the code is compiled and sent to a device via WiFi or a physical cable. After the code is sent, it's downloaded to the micro-controller and running the program as intended.

In Arduino PLC IDE it is possible to run programs at different speeds. When a program is chosen to run on the PLC IDE, the program needs to be dragged to one of the stages under the **Tasks** section. These are found in the resource tree on the left side of the screen as previously seen in Figure 33. The task section features four different stages, with different features. The explanation of these were found under task attachment in the official Arduino documentations [19].

- *Init*: Executes the code one time on initialization
- *Fast*: Runs the code in cycles of 10 ms default
- *Slow*: Runs the code in cycles of 100 ms
- *Background*: Loop that execute the program every 500 ms.

This organization of running programs makes it easier to switch the programs in the software. It grants the ability to develop and run multiple programs simultaneously, without the need to run other instances of the IDE. For example, the "*main*" file can be used to test the individual components without going in and out of the program to change the file. This is useful if a system consists of several larger parts with their own separate sub-programs. An example of where this can be useful is in a car washing machine or a full sized factory with robots.

---

## 7 PLC code for the factory

This chapter is going to discuss code that is developed for two of the PLC standard languages: Ladder Diagram and SFC. The code has been developed using OpenPLC and Arduino PLC IDE. In this chapter, the code developed for each program is compared. This is to show the difference of implementation in the two programs as well as the experience of programming in the different environments. The code developed in OpenPLC has not changed since the preceding project assignment, while the code developed in Arduino PLC IDE is newly written for this thesis. Therefore, some of the subsections regarding the languages and code developed in OpenPLC is either taken directly or had minor adjustments from the project assignment.

### 7.1 Ladder Diagram code

As mentioned in the theory part in Section 2.1.1, the Ladder Diagram language is made to be similar to relay system schematics. That means the name of the different functions is inspired from these systems. The bold lines at each end of the horizontal lines is called power-rails, and the power is going down these vertical lines. The input signals such as **I\_Sensor1** and **I\_Btn\_S1.F** are called contacts. The output signals such as **Q\_Conv1** are called coils. The program works by running power from the left rail to the right one. The contacts will stop or let the power through depending on its input signal. If the power runs through a coil, the output that the coil controls will also be powered up. In addition, there are modifiers that will change how the contacts and coils work, those will be described when encountered. An example of the LD code is given in Figure 37.

#### 7.1.1 LD Code part 1

First, some simple conceptual tests were done to see if the OpenPLC program would work as intended. This were done in the preceding project assignment for this thesis. An example of this can be seen in Figure 37. When the puck blocks the light at sensor 1, it sets the first conveyor belt, indicated by **S** in the coil. This conveyor will be activated until the conveyor is reset, like a latch catching a signal. It is reset again when seen by the second sensor, indicated by the **R** in the coil. Sensors in a LD code have a slash symbol in them, which symbolizes that they have a normally high value. This symbol indicates that the signal is negated. Buttons do not have this symbol, because they normally have a low value.

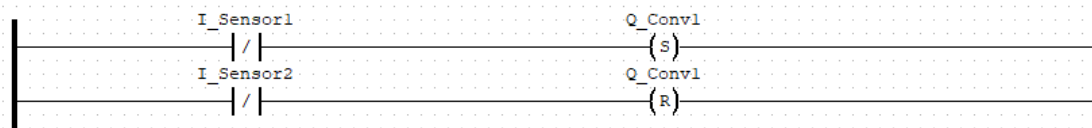


Figure 37: Simple test of Ladder Diagram.

To evaluate the programming for the whole mini-factory, the first corner was programmed first. Since the factory attributes are quite similar the whole way through, the rest of the program should be simple to write when solving the first corner. The first part including the first corner is called Part 1 in Figure 32, and can be seen in Figure 38. In Figure 39 the code for the first part can be seen, and it starts similarly to the simple test seen in Figure 37. There is however, added two

delay blocks that activates when the signal at sensor 2 is blocked, and times the deactivation of the conveyor belt. This is to ensure that the puck is in front of the slider before further actions are taken.

The rail is also forked off, to start the slider and next conveyor belt at the same time. The delays are in a pair of **TOF** (The off-delay timer) and **TON** (The on-delay timer). The **TON** is there to hinder the power signal resetting to early. Using only the **TON** would not work, because if the signal becomes low before the timer is over, the next parts would never be set properly. This happens when the puck is moved away from the sensor before the delay is over. A **TOF** is used to catch the signal for the **TON**.

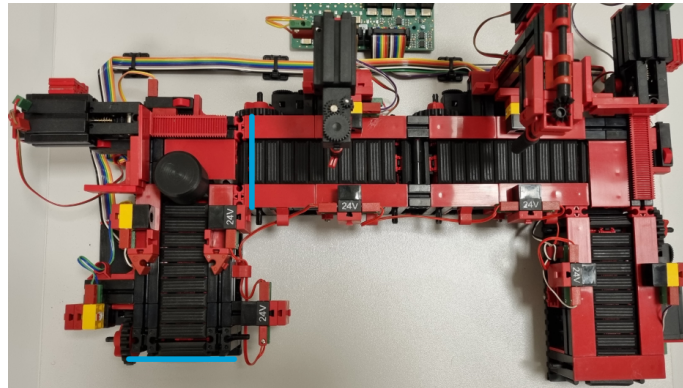


Figure 38: The Factory in bird view, showing part 1.

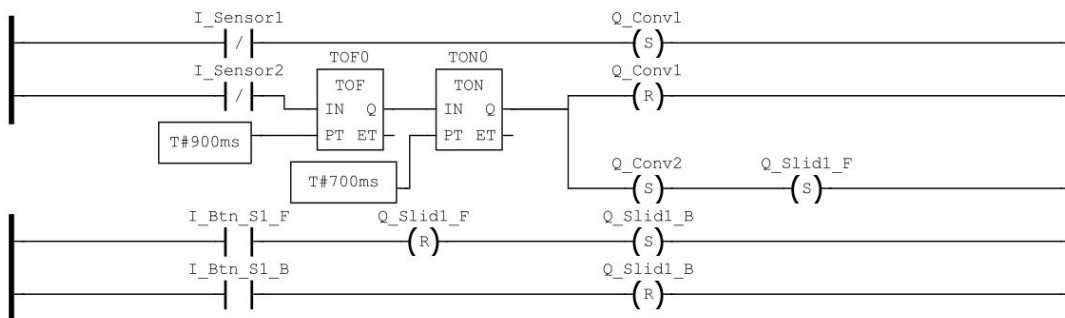


Figure 39: The first part of the Ladder Diagram.

The slider starts to move forwards, and it continues to do so until the first button is pushed, which is indicated by **I\_Btn\_S1\_F** in the code. The forward movement of the slider is subsequently reset, and sets **Q\_Slid1\_B**, which makes the slider move backwards. This is done until the slider pushes the second button, indicated by **I\_Btn\_S1\_B**. The backwards movement is reset, and the slider stops. The next conveyor belt is started while the slider is moving forwards. This was done to prevent the puck crashing into the belt. Which happened sometimes if the conveyor belt wasn't already running.

Part 1 of the code written in Ladder Logic is quite similar in OpenPLC and Arduino PLC IDE. In the first and second part of the code, there isn't much difference between the code in Figure 39 and Figure 40. Both the coils and the switches looks the same, and got the same functionality. Not surprising considering both programs follows the official PLC standards as mentioned in Section 2.7 and Section 5.1. In section 1 and 2, the differences are mainly **UI** and structure of the code. Code line numbers are featured in Arduino PLC IDE, and there are no possibility of crossing lines in

this program. The program also forces the user to place the contact at the front, and coil at the end, as seen in the lines of Figure 40 and Figure 41.

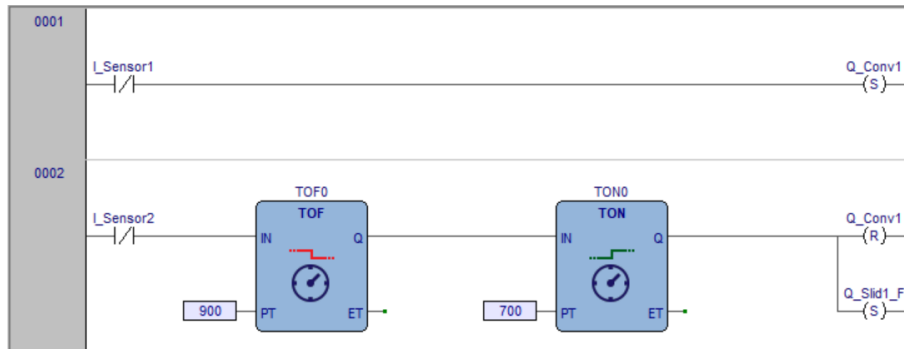


Figure 40: The first section of the Ladder Diagram code in Arduino PLC IDE

In Figure 40 on line 1, there is a negated contact that activates on the light sensor number 1. This again, activates the first conveyor belt of the factory. The second line features one contact, one **TON**-timer and one **TOF**-timer plus 2 contacts, which subsequently resets conveyor belt 1 and activates the first slider on the factory.

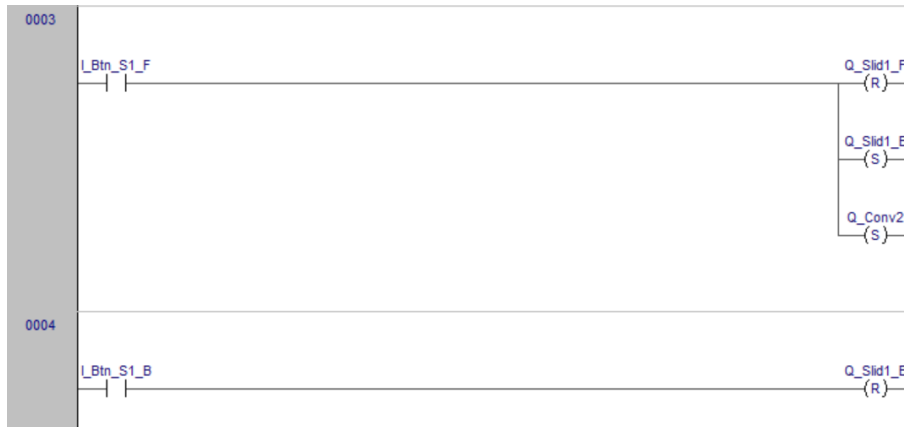


Figure 41: The second section of the Ladder Diagram code in Arduino PLC IDE

The second section in the LD code, is seen in Figure 41. This snippet of code is quite easy to understand as well. Both line 3 and 4 features one contact and several coils. When the front button of the first slider activates, slider 1 stops going forward. In addition, the slider starts going backwards and the second conveyor belt on the indexed line starts operating.

### 7.1.2 LD Code part 2

The code for the second part can be seen in Figure 43. Additionally, the bird view of the factory can be seen in Figure 42. This part includes the second conveyor belt and milling station. The first interaction starts when sensor 3 detects the puck. In the contact, there is a modifier **N**, that only detects a falling edge signal pulse. A falling edge is detected by a high value signal falling down to a low value. A falling edge on the input was necessary due to the puck will stay in the same location for a significant time and continuously reset the belt if the puck is supposed to be milled. After the third sensor detects the puck, the code splits into two paths. The path chosen depends on the sensor reading telling the code if it should be milled or not. It will always choose

only one of the paths, since one has normal **I\_Mill** and the other path negated **I\_Mill**. This line will work in the same way as an or-gate. The paths converges at a later point, the negated path just skips the milling part. In Figure 44 an alternate layout of the code can be seen. Although the code does exactly the same, it illustrates better what is activated at what time. That is because the coils is set and reset after the **TOF** delays. Since coils only need a pulse of a signal in that case, the coils can be put in front of the delay as well.

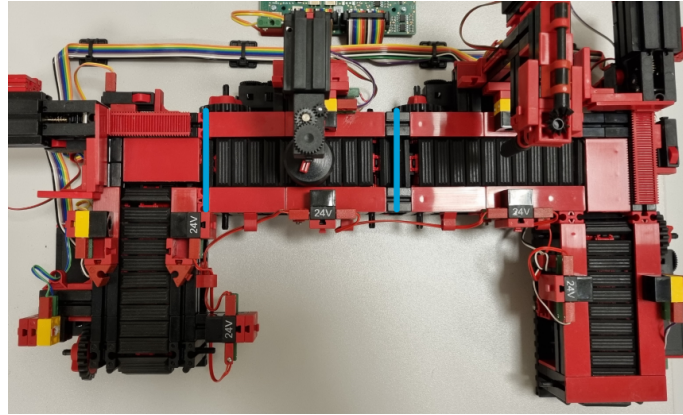


Figure 42: The Factory in bird view, showing part 2

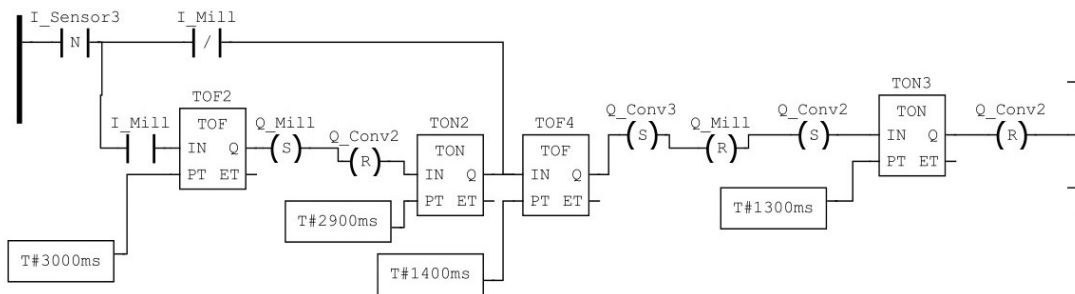


Figure 43: The second part of the Ladder Diagram.

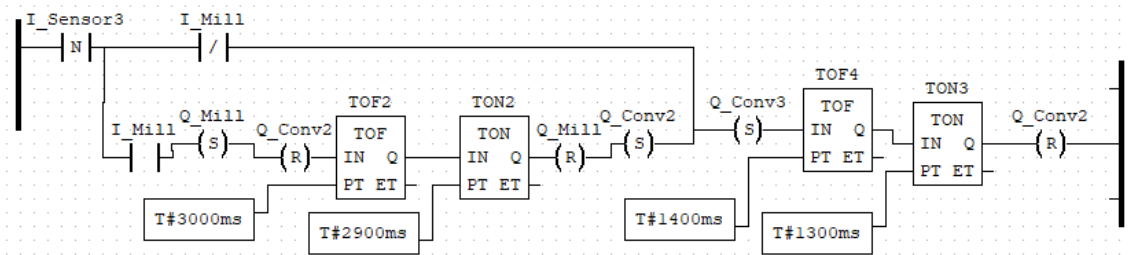


Figure 44: The second part of the Ladder Diagram in an alternate layout.

On the path with **I\_Mill**, **Q\_Conv2** is first reset and **Q\_Mill** is set. Next, there is a delay pair that ensures a certain time for milling. The code will thereafter set **Q\_Conv2** and reset **Q\_Mill**. It is in this location the paths converges, and the next conveyor belt **Q\_Conv3** is set.

In the alternate layout, the negated path only sets **Q\_Conv3**. Since **Q\_Conv2** is never reset and **Q\_Mill** is never set, the path can converge in front of the **Q\_Conv2** set and **Q\_Mill** reset. The variables will already be in these states. After **Q\_Conv3** is set, there is a new delay pair that waits long enough for the puck to settle on the third belt and subsequently resets **Q\_Conv2**.

There was not any intersection available for the Ladder Diagram code in Arduino PLC IDE. Therefore, two programs were written. One of these programs had machining included, and one didn't have machining involved. Other methods were tried to get it to work, but this would required an OR-gate, which were not found in the program. Adding more lines with the **I\_Mill** contact were also tried, but the program did not work properly. In Figure 45 the third section of the code can be seen. As can be seen on line 5, when **I\_Sensor3** changes state, **Q\_Conv2** gets reset and **Q\_Mill** starts the milling procedure.

On line 6, the timer gets activated on the same sensor. Which means the timer starts as soon as the sensor is activated. This times the execution of the milling machine, and sets **Q\_Conv2** and **Q\_Conv3** when the milling machine is reset. The timers are set arbitrarily and the value of 3 seconds is chosen because the factory seemed to run smoothly with these values.

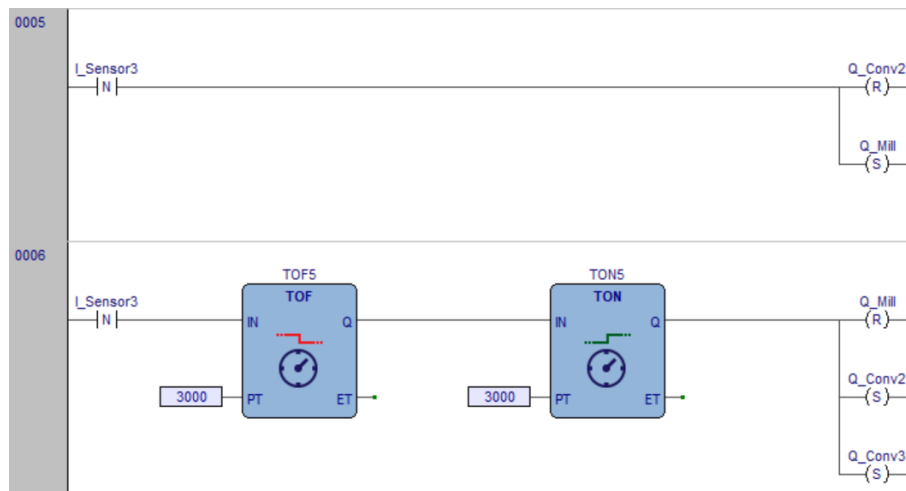


Figure 45: The third section of the Ladder Diagram code in Arduino PLC IDE

### 7.1.3 LD Code part 3

Part three is seen in Figure 46, with the corresponding code in Figure 47. This code is almost identical to the code in Section 7.1.2. The section includes some extra modules, and the milling station is replaced by a drilling station. The only difference with the not highlighted part in the code is not setting the next conveyor belt, when it has either finished, or bypassed the drilling. This is because the next actuator is a slider and not a conveyor belt. Meaning that the highlighted part is similar to the slider part of Figure 39.

This section of the code starts by looking for a falling edge on sensor 4. When this is detected, the code continues in one of the two paths. If the indicator **I\_Drill** has a high value, the puck will be drilled in the drilling station. The **TOF** delay will be activated while the factory drills, before the conveyor belt **Q\_Conv3** is set and the drill is reset. This is the point where the paths converges. The path with the deactivated drilling station does nothing until this point. Then, the delay is long enough for the puck to be in front of the second slider, before it resets **Q\_Conv3**. **Q\_Conv4** and **Q\_Slid2\_F** is simultaneously set. The slider will then move forward until the front button indicated by **I\_Bt\_S2\_F** is pushed.

At this point **Q\_Slid2\_F** will reset and **Q\_Slid2\_B** set, **Q\_Conv4** will also start running again. **Q\_Conv4** in the highlighted section could be removed, since it's already set earlier in the code.



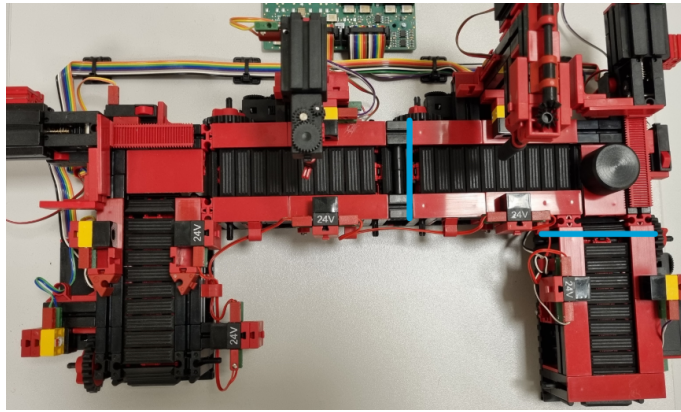


Figure 46: The Factory in bird view, showing part 3

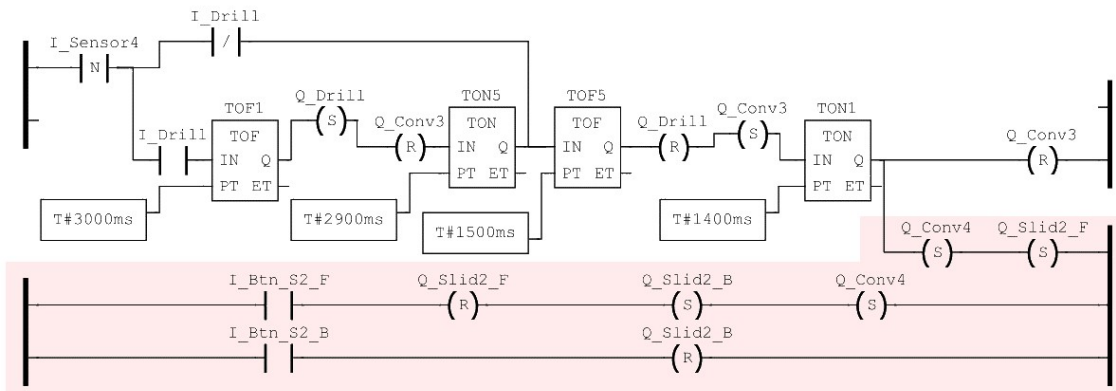


Figure 47: The third part of the Ladder Diagram

The pushbutton **I\_Btn\_S2\_B** is then pushed by the slider, and stops.

The same part of the code in Arduino PLC IDE is seen in Figure 48 and Figure 49. The fourth section is quite similar to Figure 45 which were talked about in the preceding subsection. The only two differences between these two sections are which sensor is changing state. In addition, this part is activating the drilling machine, instead of the milling. Both of the timers are set to 4 seconds to ensure that the puck has settled in front of the second slider before activation.

Code line 9 and 10, can be seen in Figure 49. These lines are responsible for the section going from the drilling machine and down to the fourth conveyor belt on the factory. This timer will as well start counting when **I\_Sensor4** is activated, and when the TON and TOF timer has finished, it will activate **Q\_Slid2\_F** which is the forward motion of the second slider. This will be activated until the slider hits the button at the front of the slider, indicated at **I\_Btn\_S2\_F**. When this button is pushed down, the slider will reset the forward motion, and the backwards motion will be started.



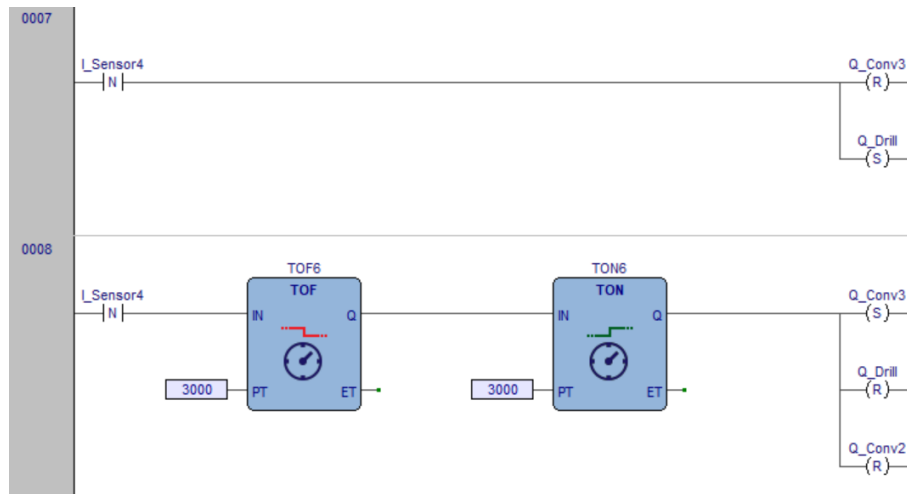


Figure 48: The fourth section of the Ladder Diagram code in Arduino PLC IDE

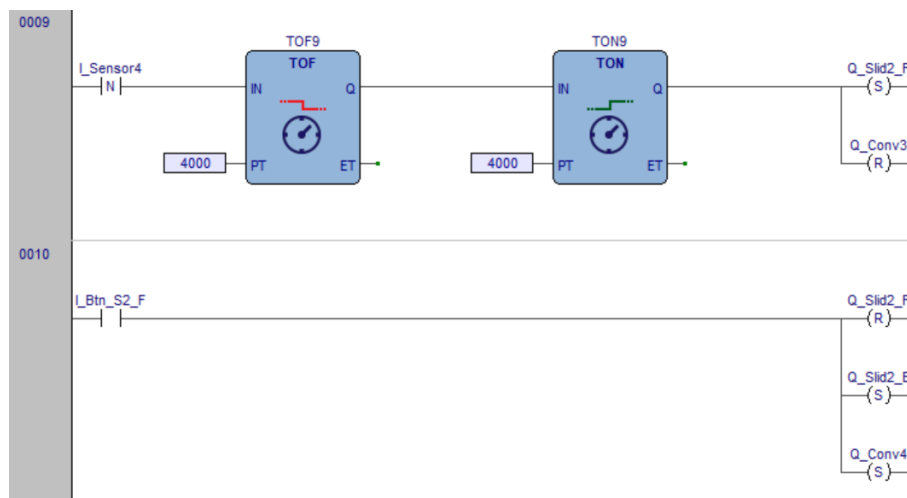


Figure 49: The fifth section of the Ladder Diagram code in Arduino PLC IDE

#### 7.1.4 LD Code part 4

The last part of the code, shown in Figure 51 is controlling the fourth part of the factory. This code controls the last section, which can be seen in Figure 50. The code is short, and its only functionality is to stop the last conveyor belt **Q\_Conv4** when sensor 4 detects the puck. A delay was also added, since it was preferred that the puck travels a bit further than the last sensor.

The last section of the Arduino PLC IDE can be seen in Figure 52. Line 11 is responsible for resetting the backwards motion of this slider. When **I\_Btn\_S2\_B** changes state, the slider is reset. When the puck activates **I\_Sensor5**, **Q\_Conv4** is reset and the process is finished.

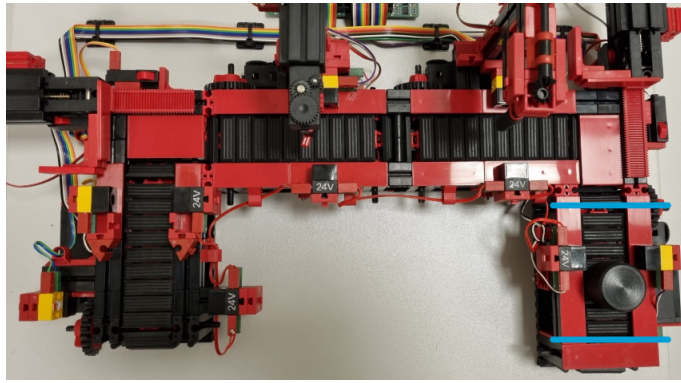


Figure 50: The Factory in bird view, showing part 4.

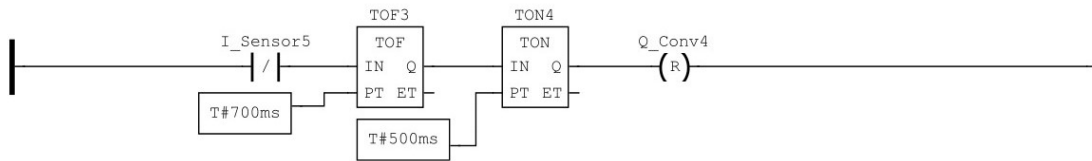


Figure 51: The fourth part of the Ladder Diagram.



Figure 52: The sixth section of the Ladder Diagram code in Arduino PLC IDE

### 7.1.5 Discussing the Ladder Diagram Code

The ladder diagram code in OpenPLC was developed in the preceding project assignment to this thesis and is not optimal. It's going to need major changes if it's going to be proposed as a solution of an assignment. The code developed in OpenPLC doesn't follow standards of the Ladder Diagram language either. This is due to the contacts not being at the end of the code lines. It was created in parts, and functionalities were added along the development. The alternate version of the second part seen in Figure 44, were created to easier describe its functionality. If it's decided that the work done in this thesis is going to be used for an assignment, developing new code in either program is recommended.

There were some differences between making the ladder diagrams in Arduino PLC IDE than OpenPLC. Other than the fact that the device needs to connect with ModBus on Arduino PLC IDE, the program also forces the user to follow the standards of the language. On OpenPLC, the program starts with a completely blank sheet.

This means that the user is able to drag items freely into the sheet when programming. In Arduino PLC IDE however, the program features pre-programmed lines which needs to be used. In addition, the program blocks the user from doing "illegal" operations. This also leads to an inability to add

---

multiple coils to the same line. The program makes an intersection instead, and cover the coils on multiple lines. For a complete rookie, it might be easier to program ladder logic in the Arduino PLC IDE due to forcing the standards.

## 7.2 SFC - Sequential Function Chart

Another standard language used for PLC, is called SFC. In addition to the LD code, a code in SFC has been made for this thesis. Like the Ladder logic code discussed in Section 7.1, the SFC code from OpenPLC was also made in the preceding project assignment.

The SFC code starts with an initial statement. Action blocks are then built into the steps. These blocks then activate some outputs, which can further be connected to motors etc. The action blocks features different qualifiers that represents commands. These are in the form of different letters, where each letter represent one command. These are bound to features like timing features, set and reset.

Each step is divided by transition steps. These transitions are often connected to a sensor or other types of input signals. This makes it easy to regulate how long a step is active. One of the main differences between Ladder Logic and SFC, is the ability to jump in a sequence randomly. In Ladder Logic, activating a sensor in the middle of the sequence will start a new sequence starting where the activated sensor is located. SFC has to finish the whole sequence before the program can be started again.

This type of language is commonly used in production lines, as well as other autonomous installations like car washes. Full action blocks or programs can be connected to a **Step**. For example, on a car wash program, "wash" and "dry" can be independent programs connected together with steps and transitions. These smaller programs are usually called actions [30].

The variables set for this code, has the same addressing as the variables in the Ladder Logic code, seen in Figure 34. Disclaimer, this is only a prototype code which could be streamlined if it was going to be used as a solution in an exercise. However, it's working correctly, and shows the basic structure of a SFC code.

## 7.3 Arduino PLC IDE: SFC

When comparing Ladder Diagram for the two different IDEs used in this project, there are a lot of similarities between the programs, making the gap between the different programs smaller. Although for the SFC code, OpenPLC and Arduino PLC IDE are fundamentally different in how the code is implemented.

When a SFC program is made in Arduino PLC IDE, the program will open a blank sheet. From there, steps are going to be implemented into the program. However, Arduino PLC IDE has a fundamentally different method of implementing the "actions". In OpenPLC, the SFC program features a lot more steps than Arduino PLC, this is mainly because each action isn't bound to one step.

The main reason the SFC code will look different in the Arduino PLC IDE compared to OpenPLC, is the lack of qualifiers available for the user. In the standard for the language, as well as in

OpenPLC, the user have many more possibilities regarding programming the blocks. The action steps in Arduino PLC IDE are limited by only two qualifiers, "P" and "N". The "P" stands for Pulse, and when using this qualifier the action will be activated twice. Once on activation, and once on deactivation of the step. This is following the IEC standard according to *CODESYS* [31]. The "N" qualifier however, stands for "Non-stored". Using this qualifier will make the code active as long as the step is active.

Not being able to use the **Set** and **Reset** qualifiers in Arduino PLC IDE, means that the code is looking fundamentally different. The actions in Arduino PLC IDE exists as **ST** code files. Each action is bound to a little snippet of code in a separate Structured Text (**ST**) file. In OpenPLC, the actions are represented as blocks which can be added directly onto the step. In the **ST** files written for each steps, the variables are configured as booleans, mimicking the functionality of the "Set" and "Reset" commands used in the OpenPLC program.

### 7.3.1 SFC section 1

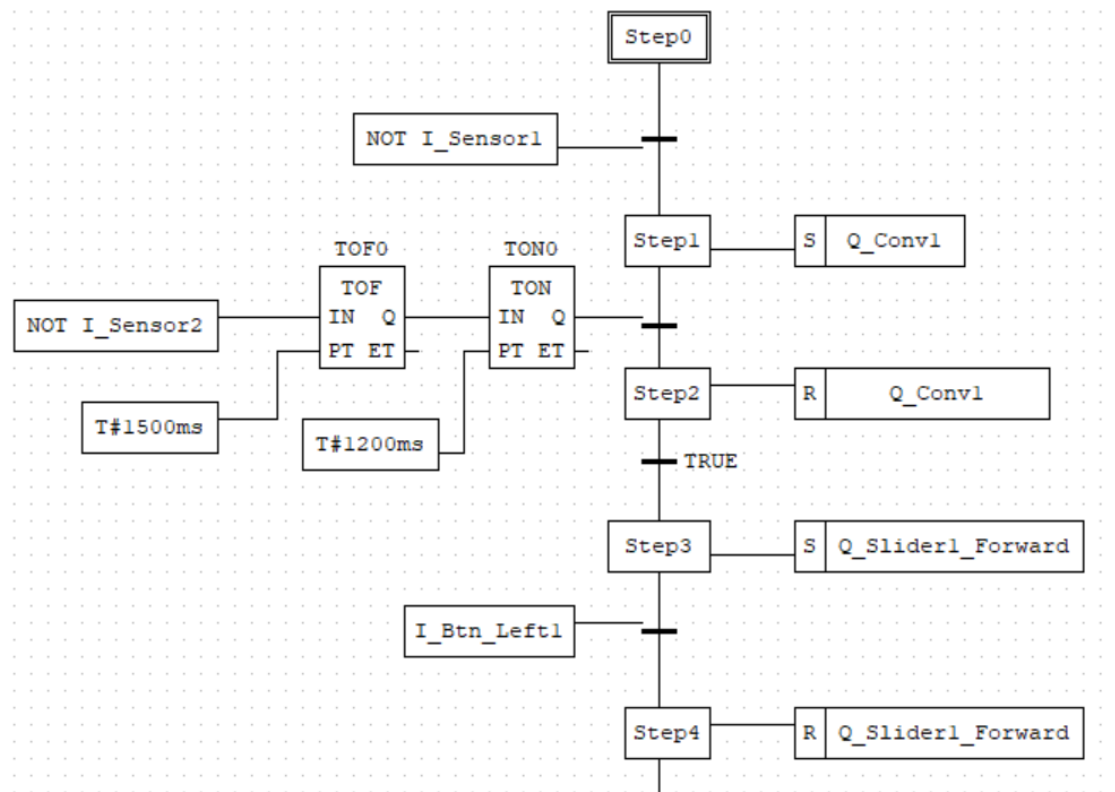


Figure 53: The first part of the SFC code

This section highlights the first part of the code, seen in Figure 53. As can be seen, the code starts to run from the initial step. The initial step is called **Step0** in this instance. On the first transition between **Step0** and **Step1**, an input variable has been set. These input boxes works with normal boolean values. Since **I\_Sensor1** is normally set to true, the program will continue when this signal is set to a low value. This happens when the object on the conveyor breaks the light signal, and the input signal on the transition is set to **NOT False**. To keep the code tidy and easier to read, all of the input variables have been set as inputs to the transition steps on the left side of the chart. This also sets checkpoints in the code, ensuring that the code won't leap

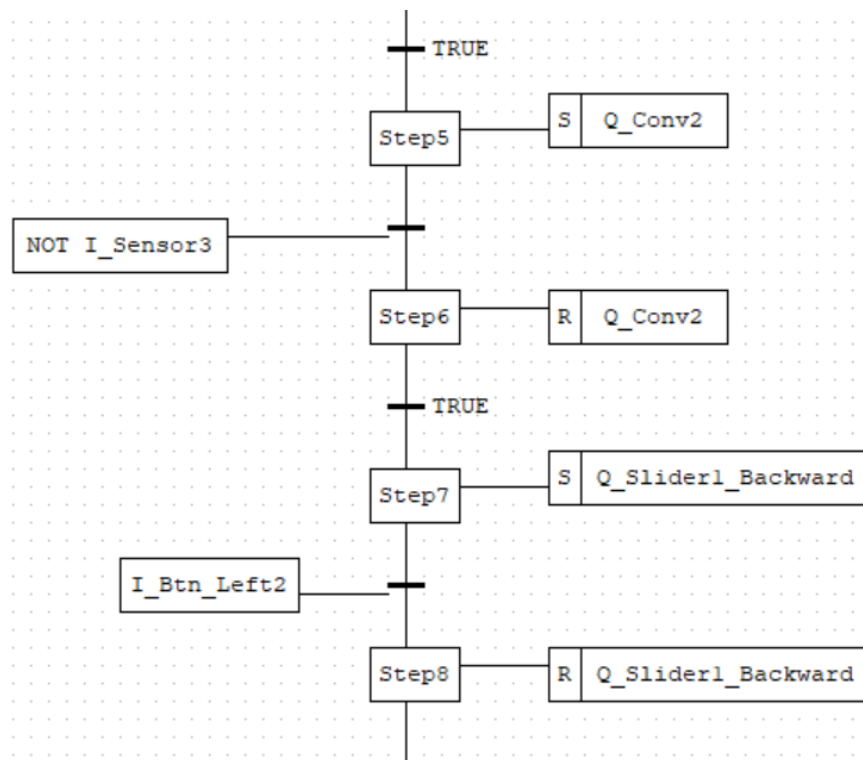


Figure 54: The second part of the SFC code

ahead of the puck on the Indexed Line.

An action is connected to **Step1**. This action starts the conveyor as soon as the step is activated. It has the qualifier **S**, which starts the belt when activated, and runs the belt until a specific reset command is given. This command is given by the qualifier **R**, which can be seen connected to **Step2**.

Due to the layout of the mini-factory, some timers needs to be added to ensure that the puck is in the correct position before each action is proceeded. For this, a **TON** and **TOF** timer has been used. Together, they start the timer when the light signal on **I.Sensor2** is blocked, and activate again after 2.7 seconds. This seemed like an appropriate delay for the program. When the timer runs out, **Step2** activates, and the second conveyor is reset. How the **TOF** and **TON** timer works together is described in Section 7.1.1.

**Step3** to **Step8** describes the slider mechanism. At **Step3**, the conveyor is set, and is reset again when the push-button is pressed down by the slider. This will set the input signal to **True**, and the program will continue further to **Step4**. As can be seen from Figure 54, **Step5** starts the conveyor. The conveyor is then reset when the light signal at **I.Sensor3** is blocked. This sensor is located at the milling station. When the sensor at the milling station is activated, the belt is reset, and the slider goes back to it's original position in **Step7** and **Step8**.

The next code that is going to be looked at is the SFC code that has been developed using Arduino PLC IDE and compare it to the code written in OpenPLC. The first part of the code, consists of **Init** and **Step0**, very similar to the code that is implemented in OpenPLC.

However, in the next figure, Figure 55 the code contained in the action is showed. Instead of binding one action to each step like in OpenPLC, a complete code file written in **ST** is implemented into

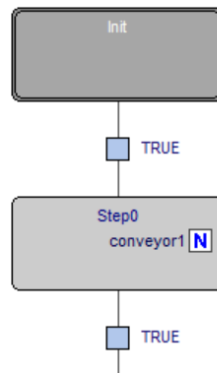


Figure 55: The first section of the SFC code in Arduino PLC IDE

the step. The reason that this method was chosen, is that adding code to the transition did not work. This problem required a different method to program compared to OpenPLC. The Arduino PLC device would not run when conditions were added to the transitions.

```

0001 IF NOT I_Sensor1 THEN
0002     Q_Conv1 := TRUE;
0003
0004 END_IF;
0005
0006
0007 IF NOT I_Sensor2 THEN
0008     TOF0(IN:= NOT I_Sensor2, PT:= 900);
0009     TON0(IN:= TOF0.Q, PT:= 700);
0010
0011     IF TON0.Q THEN
0012         Q_Conv1 := FALSE;
0013         Q_Conv2 := TRUE;
0014         Q_Slid1_F := TRUE;
0015
0016     END_IF;
0017 END_IF;
0018

```

Figure 56: The step in an ST file in Arduino PLC IDE

The code shown in Figure 56 is pretty straight forward. If **I\_Sensor1** is activated, the belt starts running. The belt will continue running until **I\_Sensor2** is activated. This code is seen at line 1 through 4.

On line 7 to 17, the code is a bit more complex. Due to the mini-factory not having sensors in the intersections, the de-activation of the conveyor belt needs to be delayed by a small margin. This ensures that the puck is properly positioned on the conveyor belts before slider activation. This is done by implementing the TOF and TON timer the same way as the project thesis for both the LD and SFC code.

```

TOF0(IN:= NOT I_Sensor2, PT:= 900);
TON0(IN:= TOF0.Q, PT:= 700);

```

Another large difference between **SFC** and **ST** is the syntax. Instead of having a **TON** and **TOF** timer like blocks, they are added as lines of code instead. They still work fundamentally the same. The **IN:** channel in the function symbolises the activation point, and the **PT:** is the length of the timer activation. **TOF0.Q** activates when **TOF0** is finished running. Other than that, line 11 to

17 activates when the timer is out and stops conveyor 1, in addition to starting Conveyor 2. Slider 1 will also start going forward.

### 7.3.2 SFC section 2

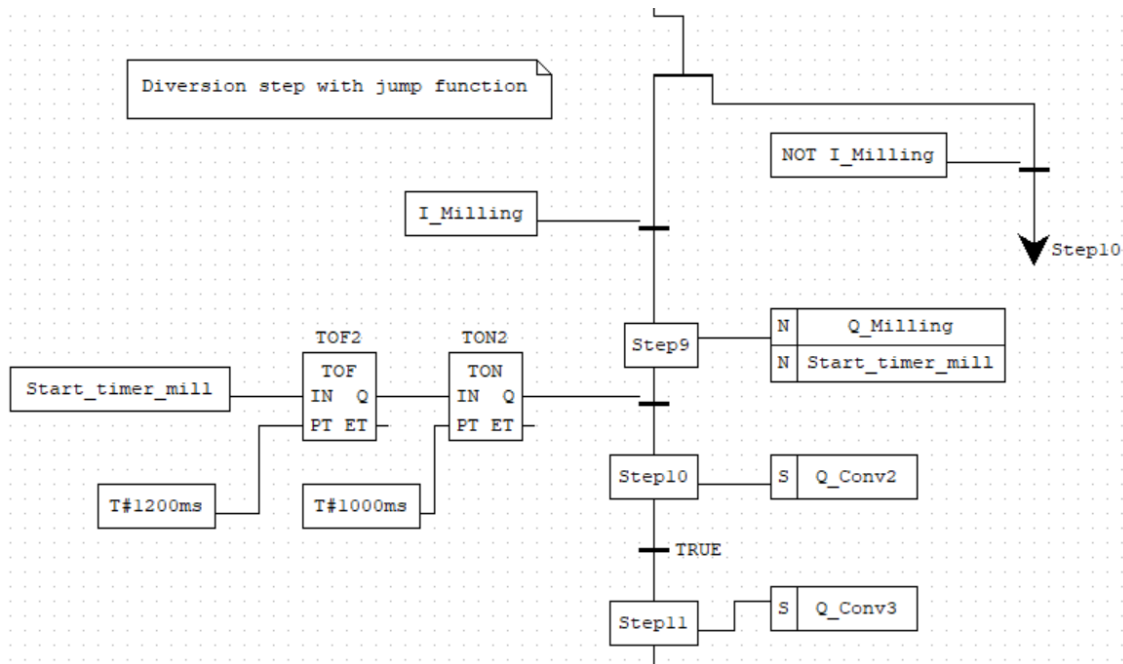


Figure 57: The third part of the SFC code

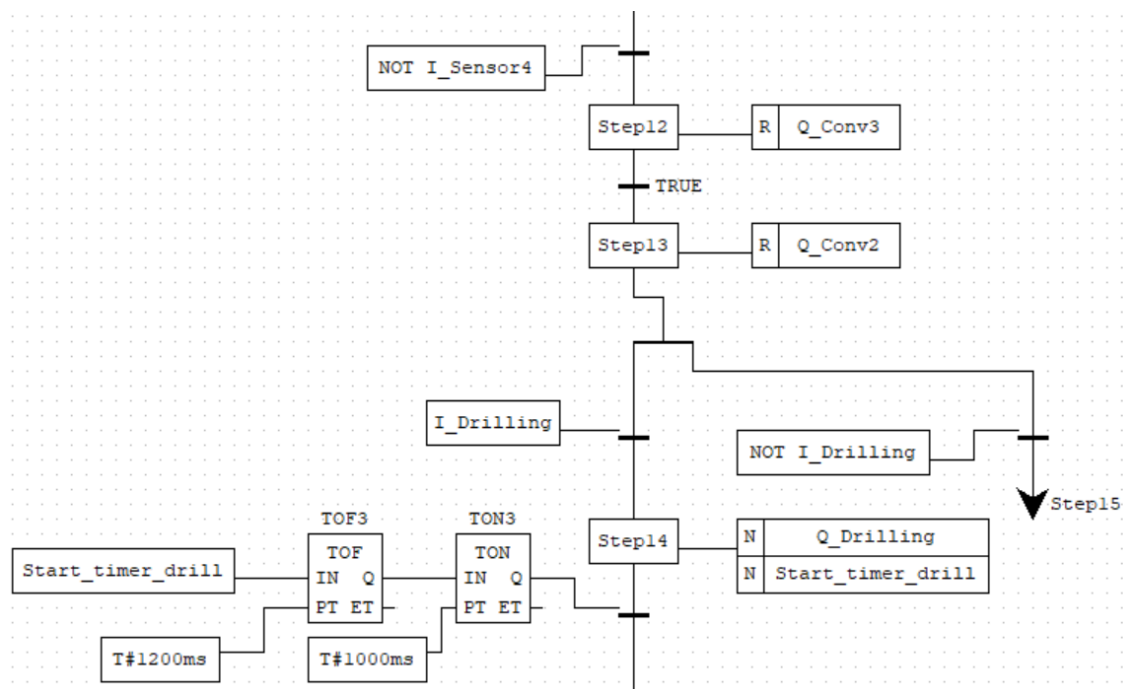


Figure 58: The fourth part of the SFC code

It was desired to use an other code to decide if the machining stations should run. To implement this in SFC, a division step was made. This can be seen in Figure 57. This division step is corresponding to the **OR** command in boolean terms.

---

This section of the code, decides what path that should be taken based on a **True** or **False** statement. This statement tells if the milling is going to be activated or not. If this statement is **False**, **NOT I\_milling** is **True**, and the code jumps down to **Step10** without executing the path via **Step9**. If the object is going to be milled, **I\_Milling** is **True** and the milling starts. This step has the **N** qualifier, which means that the conveyor will run as long as the step is active. The time delay is currently set to 2.2 seconds.

In **Step10** and **Step11**, conveyor 2 and 3 are set almost simultaneously. As seen from figure 58, these will run until **I\_Sensor4** is activated. This was done to ensure that conveyor 3 was running when the puck came onto it. **Step10 - 13** could have been shortened down to two steps, although it was decided to use only one action per step, to minimize potential errors. The division step for the drilling station is exactly the same as for the milling station with the same delay.

The second part of the code developed in Arduino PLC IDE consists of step 1 and 2, which is seen in Figure 59. This part includes the code that controls the interaction of the milling machine and drilling machine.

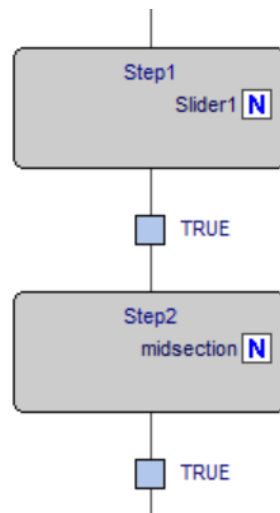


Figure 59: The second section of the SFC code in Arduino PLC IDE

The code in **Step1** is seen in Figure 60. These lines are pretty straight forward and easy to understand. The first **IF** statement checks if the front button of the slider is activated. On activation, the forward motion of the slider is stopped, and the slider is going backwards until it hits the button on the back of the track of the slider.

In the code for **Step2** seen in Figure 61, the timing for going through the middle section of the mini-factory is added. At line 1, the code starts checking if the light sensor 3 is activated. **I\_Sensor3** is the sensor which is set in front of the milling machine. On line 3, there is a boolean called **I\_Mill** which indicate if the milling machine is going to be active or not. If **I\_Mill** is set to **TRUE**, the motors will start the following sequence; Conveyor 2 stops, the milling machine starts spinning and the timers are set for 3 seconds each. When the timer is finished, the milling machine stops running and conveyor 2 starts again. After this code sequence is finished, conveyor belt number 3 starts running.



---

```

0001 IF I_Btn_S1_F THEN
0002     Q_Slid1_F := FALSE;
0003     Q_Slid1_B := TRUE;
0004
0005 END_IF;
0006
0007 IF I_Btn_S1_B THEN
0008     Q_Slid1_B := FALSE;
0009
0010 END_IF;
0011

```

Figure 60: The step in an ST for Arduino PLC IDE

---

```

0001 IF NOT I_Sensor3 THEN
0002     (*Adding a timer for the mill*)
0003     IF I_Mill THEN
0004         Q_Conv2 := FALSE;
0005         Q_Mill := TRUE;
0006         TOF1(IN:= I_Sensor3, PT:= 3000);
0007         TON1(IN:= TOF1.Q, PT:= 3000);
0008
0009         IF TON1.Q THEN
0010             Q_Mill := FALSE;
0011             Q_Conv2 := TRUE;
0012         END_IF;
0013     END_IF;
0014
0015     Q_Conv3 := TRUE;
0016 END_IF;
0017

```

Figure 61: The second step in an ST for Arduino PLC IDE

### 7.3.3 SFC section 3

At **Step15** in Figure 62, it was set that conveyor 3 should run for 1.5 seconds. This was done with the **L** qualifier, which runs the action for a specified amount of time. Due to it being no sensors at the end of conveyor 3, this was the easiest way to ensure that the puck has come onto the slider-platform. At the same step, it was set that the slider was going to run after a delay of 2.2 seconds.

**Step16** and **Step17** moves the slider the same way as described in Section 7.3.1. **Step18** sets conveyor 4, and resets when the light signal in **LSensor5** is blocked. As can be seen in Figure 63, the last two steps resets the slider the same way as described in Section 7.3.2. The final jump skips to the initial step, the sequence is finished, and the factory is ready to start over again.

The third section of the Arduino PLC IDE code consists of **Step3** and **Step4**, seen in Figure 64. This part includes the drilling machine, the last slider as well as the ending mechanism for the factory.

**Step3** is comparable to the ST code block found in **Step2** of the code. The action block code is seen in Figure 65. However, this code takes care of the drilling machine on the factory instead of the milling. Line 1 to line 12 is exactly the same as **Step2**, except the mill variables are replaced by drill variables.

Line 12 to 23 represents conveyor 3 and 4, as well as the second slider on the factory. When the

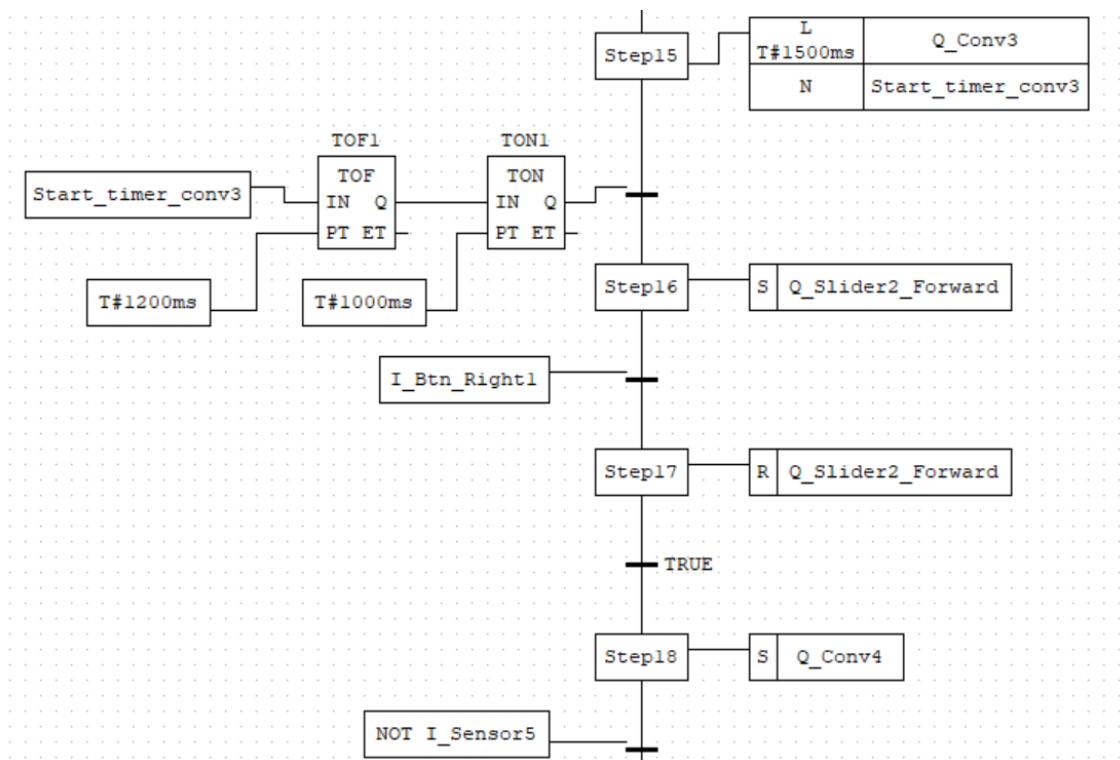


Figure 62: The fifth part of the SFC code

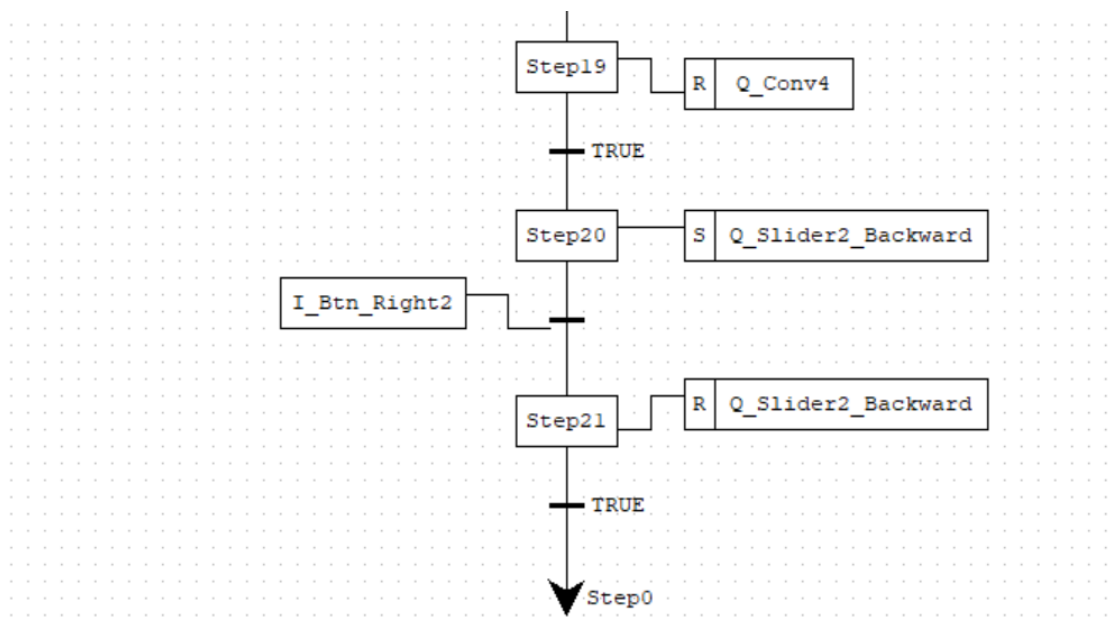


Figure 63: The sixth part of the SFC code

drilling machine is finished running, conveyor 3 starts running again for 4 seconds. This is enough time to let the item going around the factory go completely on the slider before execution. When the time is over, slider 2 starts going forwards, and the last conveyor on the factory starts running.

**Step4** in the consists of stopping conveyor 4 from running, as well as setting the rest of the conveyors as false in case some of the other conveyors are still running, why this is the case is going to be discussed in a later chapter.

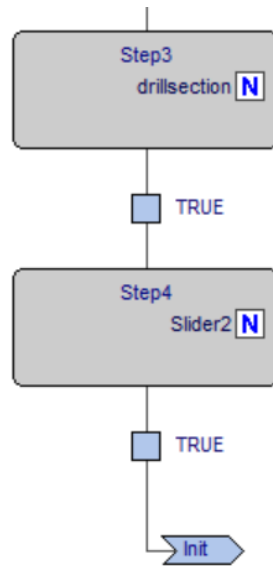


Figure 64: The third section of the SFC code in Arduino PLC IDE

```

0001 IF NOT I_Sensor4 THEN
0002   Q_Conv2 := FALSE;
0003   (*Again, adding a timer, this time for drill*)
0004   IF I_Drill THEN
0005     Q_Drill := TRUE;
0006     TOF2(IN:= NOT I_Sensor4, PT:= 3000);
0007     TON2(IN:= TOF2.Q, PT:= 3000);
0008
0009     IF TON2.Q THEN
0010       Q_Drill := FALSE;
0011     END_IF;
0012   END_IF;
0013
0014   Q_Conv3 := TRUE;
0015   TOF3(IN:= NOT I_Sensor4, PT:= 4000);
0016   TON3(IN:= TOF3.Q, PT:= 4000);
0017
0018   IF TON3.Q THEN
0019     Q_Slid2_F := TRUE;
0020     Q_Conv4 := TRUE;
0021
0022   END_IF;
0023 END_IF;
0024

```

Figure 65: The step 3 in ST for Arduino PLC IDE

The if statement going between line 1 to 5 stops the slider going forward when the button in front of the slider track on the factory is pressed. In addition, it will start the backward motion of the slider, until it hits the button on the back of the track. Line 12 to 17 just deactivates all of the conveyors when the last sensor on the factory is activated. If desired, a delay can be added to make the item on the belt stop on the end of the belt. This would be added similarly as the timers for the milling and the drilling machine.

### 7.3.4 Discussing the SFC code

One of the major surprises encountered when programming in SFC is the inability to use conditionals on transitions. When running the code on the Arduino PMC, it also started to behave a bit weird. It seemed like the code in the **ST** code files was not able to catch the signal from the

---

```
0001 | IF I_Btn_S2_F THEN
0002 |     Q_Slid2_F := FALSE;
0003 |     Q_Slid2_B := TRUE;
0004 |
0005 | END_IF;
0006 |
0007 | IF I_Btn_S2_B THEN
0008 |     Q_Slid2_B := FALSE;
0009 |
0010 | END_IF;
0011 |
0012 | IF NOT I_Sensor5 THEN
0013 |     Q_Conv2 := FALSE;
0014 |     Q_Conv3 := FALSE;
0015 |     Q_Conv4 := FALSE;
0016 |
0017 | END_IF;
0018 |
```

Figure 66: The step 4 in ST for Arduino PLC IDE

puck passing a sensor. These were later blocked with a finger, which seemed to work. This is most likely a software problem. Either caused by the code written in SFC or the program itself. This was not a problem that were seen in OpenPLC.

---

## 8 Discussion

### 8.1 Use of mini-factory in Industrial Mechatronics

*Sund* has in his master's thesis [3] found that there is a desire to use the mini-factory to replace existing assignments in Industrial Mechatronics. The thesis was discussing the options of using actual PLCs to run the factory. Due to licensing issues and the issue of actually acquiring PLCs, it was deemed an impracticable solution. However, his research stated that it was valuable experience for the students taking the course.

The project assignment written by *E. Godli* and *L. Bonvik* in the preceding project thesis has found that a Raspberry Pi have a lot of potential regarding educational content, and is a fine replacement for a proper PLC unit. Especially with useful tools like proprietary hardware such as the developed PCB as well as the open-source software of OpenPLC. The advantages of using such a micro-controller is adaptability, and a micro controller can easily be repurposed for use in different assignments.

This is also true for Arduino UNO and MEGA. These are really popular micro controllers and is used for a multiple of open source projects all over the world. These controllers are also used in assignments in multiple courses at *NTNU*, for example *TPK4125 - Mechatronics* and other similar courses. These controllers would also work great for a potential assignment writing and running code for the mini-factory.

### 8.2 Raspberry PI

The Raspberry PI has several advantages that makes the device ideal for use in an assignment in Industrial Mechatronics. Although it also have some drawbacks not found in the comparable micro controllers. These advantages and flaws are bound to the potential use of the device in assignments in the course specifically. A list consisting of these points is seen in the next subchapter. A picture of the Raspberry Pi with the PCB version from the project assignment connected to the factory can be seen in Figure 67.

#### 8.2.1 RPi Advantages

- The PCB is developed to work directly with the Raspberry PI
- Huge potential for using it in other assignments

One of the main advantages of using a Raspberry PI, is that the PCB from the project assignment is developed to work with the Raspberry PI. When the project assignment started, the main goal was to see if the RPi could be used to emulate a proper PLC. Therefore most of the work done in the project assignment was done with this idea in mind. This is already described in Section 1.2. When it was found that this was possible, the PCB was developed with this in mind. Although with the later changes done to the PCB, there should not be anything that keeps it from working just as good with different devices.

Another even greater advantage of using a Raspberry PI for a potential assignment is that it is easy to convert for use in other assignments as well. As described in Section 2.2.1, there was a

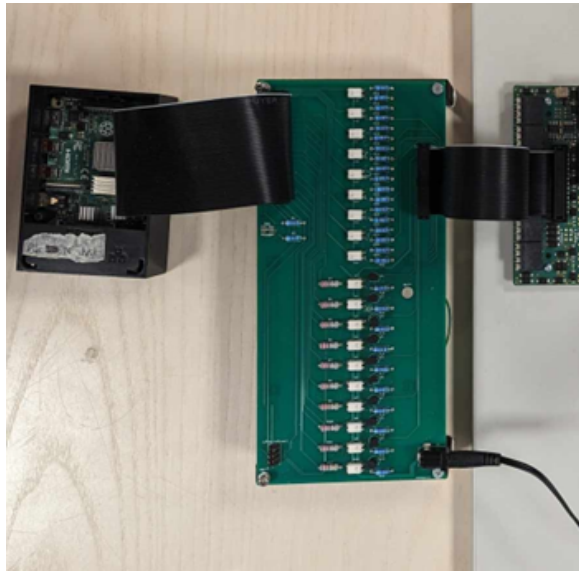


Figure 67: The Raspberry Pi with the old PCB connected

need to install a custom Operating System on the device. This experience could also be valuable as a potential assignment. Doing all of these steps is quite time consuming, however it can be valuable for making students learning more about Operating Systems on micro controllers.

However, the Raspberry Pi isn't just limited to this. Other potential assignments can be to write a server and client system and make the micro controller interact with a PC over WiFi using IP addressing. There are also some leftover pins on the PCB, these can be used to add some external device in addition to the factory, for example a camera. The Raspberry Pi also has a lot of RAM and computing power. These capabilities can be used to handle multiple devices.

### 8.2.2 RPi Disadvantages

- Expensive
- Need for external devices
- Can be cumbersome to use
- Only one software choice

However, even though the Raspberry PI comes with a lot of advantages. There are also some drawbacks that needs to be discussed. One of them, is the price. As per 09.12 - 2023, one Raspberry PI 4 model B with 4 GB of RAM costs 779 NOK at *Komplett.no*. This price does however not include a power adapter. Therefore at the time of writing, one Raspberry PI with a power adapter costs roughly 1000 NOK. This isn't too bad, although according to *Prof. Amund Skavhaug*, there is a desire to buy roughly 20 units. This totals to an initial cost of 20000 NOK. In addition, one would also need to use money for PCBs. These were bought from *PCBWay*, and cost 2500 NOK for manufacturing, shipping and additional taxes. This price is the total amount for 25 units, making each PCB cost around 100 NOK a piece. However, this price was not including the various parts that needs to be soldered on the PCB itself. This includes the resistors, opto-couplers and the headers. This will add an additional cost to all of the devices utilizing the PCB boards.

---

For an experienced solderer, the PCB would take around an hour or two to complete. However, it is possible to avoid this cost by giving the boards to students to solder as an assignment. To summarize, the initial cost of using a Raspberry Pi for this assignment is estimated to be around 25000 NOK.

Another disadvantage of using the Raspberry Pi, is that the initial work required to start preparing it for an assignment is much higher than the other mentioned devices. At first, a customized Operating System is required, which in the case of the project assignment Section 1.2 is done with the steps found in Section 2.2.1. If the goal is to only use the Raspberry PI to teach students how to use the PLC standard languages, there is a lot of work required to make the RPIs "PLC-ready".

In addition, to make the Raspberry Pi communicate via OpenPLC, there is a need to install software on the Raspberry Pi itself. This can easily be done by downloading it from a web browser or from the terminal. However, to be able to do this, the user needs to be able to connect to a screen, as well as connecting it to a mouse and keyboard. This isn't a problem for one person, but when talking about around 20 students, this might be a cumbersome process and a lot of devices need to be sorted. It is unreasonable to think every student has access to this.

Another drawback which was also discussed in the project thesis is that the Raspberry Pi needs to be connected to a wireless network when the code is transferred. This is the only one of the selected devices that runs PLC code via a *Runtime*. This means that the Raspberry Pi needs to be connected to *Eduroam*, or a similar wireless network. A comprehensive guide on how to do this is found in Appendix B. The problem of connecting the Raspberry Pi to the Eduroam network is that the device is changing the IP address repeatedly. Therefore, the Raspberry Pi should be connected to a screen even while developing and testing the PLC programs for the factory. This was discussed in further detail in the preceding project assignment.

### 8.3 Arduino PMC

The next device to be considered for a possible assignment is the Arduino H7 Portenta Machine Control. This is by far the least versatile of the bunch, but this device is a real PLC. This is the only device mentioned in this thesis that follows the PLC standards. Therefore, the Arduino PMC has some unique capabilities that are going to be discussed below. In addition, a picture of the Arduino PMC connected to the ribbon cable to the factory is added in Figure 68.

#### 8.3.1 PMC Advantages

- Robust
- Easy to use
- Several software choices
- No need for the PCB layer

One of the greatest advantages with this device, is that it is easy to use. This unit with a proper customized shield can be connected directly to the factory without any need of an additional layer. This is possible by it delivering 24 volts straight from the outputs as discussed earlier in the thesis

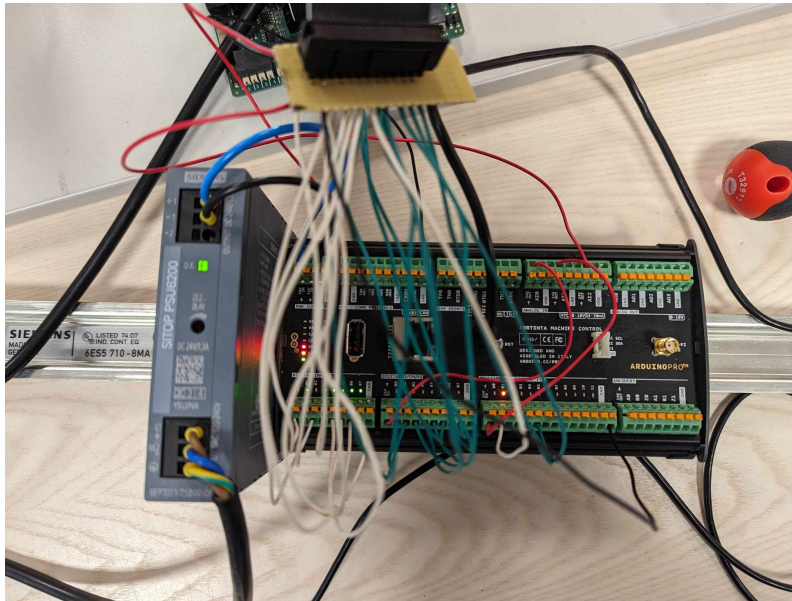


Figure 68: The Arduino PMC connected to the factory

at Section 3.1. To connect to the factory, all that is needed is connecting the 24 wires directly to the pins on the factory. Alternatively, make a shield that can do the connection with a simple ribbon cable which is done in this thesis as seen in Figure 30. All these qualities makes the system more reliable than the other devices. With less parts involved, there's less potential to develop problems in the future.

The Arduino PMC is also compatible with two different programming softwares, both OpenPLC and Arduino PLC IDE. This ability is unique compared to the other devices. To run code on the device, one simply needs to compile it on OpenPLC and transfer it directly to the device. The PMC unit has a micro usb port which can be used, by simply connecting it to a PC.

### 8.3.2 PMC Disadvantages

- Expensive
- Currently some incompatibility with OpenPLC
- Small degree of versatility

If it's decided to use OpenPLC for a potential coding project, there are some qualities that are missing that makes the user unable to use all of the features this device provides. The most notable one is the ability to use the programmable ports. In regard to the work done in this thesis, driver code for using the programmable pins was developed. Further details of this was discussed earlier and is found in Section 5.3.4. These ports needs to be available when using the Arduino PMC together with the Indexed Line, to have enough I/O available. This functionality needs to be unlocked using an updated driver, which is custom made. This process might be a bit cumbersome, but it isn't something students taking the course *Industrial Mechatronics* shouldn't be capable of doing. There is also provided a guide on how to use the Arduino PMC with the updated driver. This is given in Section 6.6.



---

One of the major drawbacks of using Arduino PMC, is that they are expensive to purchase. One unit costs a total of 3000 NOK if it's desired to buy it directly from *Arduino*. In addition, the unit that were used for experimentation in this thesis, had a power supply from *Siemens* attached to it. The particular power supply costs around 1000 NOK according to the supervisor. This means for each complete device, the estimated cost would be 80000 NOK for 20 units. This is over three times the amount that would be needed to buy Raspberry Pis, which is the second most expensive item on the list.

The last drawback is the functionality. This device is a highly specialized full function PLC unit. Compared to the Raspberry Pi, this means there is no value in using the unit for non-PLC purposes. A lot of the curriculum in *Industrial Mechatronics* is regarding wireless communication between devices and Operating Systems. In addition, there are some coding exercises in the C programming language. These things cannot be done on the Arduino PMC, due to it being a highly specialized PLC device. It can neither be used to run other devices which are not running a 24 volt system. This would require an interfacing layer similar to the one developed in the project thesis.

## 8.4 Arduino UNO

The next item to be discussed, is the Arduino UNO. Although the Arduino doesn't have enough hardware itself to run the factory, it's possible to buy an I/O expander, which increases the number of I/O pins on the Arduino UNO. This section will then describe the advantages and disadvantages of using an Arduino UNO with such an expander. One expander that can be used is the *Sparkfun SX1509* which were bought for the project, but not used. This I/O expander is seen in Figure 69.



Figure 69: The I/O expander that was purchased for thesis

### 8.4.1 UNO Advantages

- Cheap
- Can be used for more assignments

- 
- A lot of internet support

The main advantage of using an Arduino UNO is that the device is cheap. One Arduino UNO from the official Arduino store costs 24 euros per unit. This is much cheaper than the other controllers mentioned in this thesis. This means that 20 units of Arduino UNOs would cost 480 euros. In the local currency, this equals to roughly 5000 NOK with the exchange rates at 10.12-2023. This is under half the price of the Arduino MEGA. However, more purchases are needed to be made. As stated earlier, there is a need to use an I/O expander such as the SX150 seen in Figure 69. These can be bought for 3 dollars a piece. Meaning there is an additional cost of 660 NOK for the I/O expanders, again with the current exchange rates as per 10.12 - 2023. This does not include the cost of the PCBs which totals to roughly 2500 NOK. This was calculated in Section 8.2. Therefore, the total cost of using an Arduino UNO would be roughly 8000 NOK, which is still cheaper than the equivalents mentioned in this thesis.

Another benefit of using Arduino UNOs is that they can be used for assignments in other courses. Because of the controllers simplicity, the Arduino UNO is a popular choice for hobby electronics. It's also used in assignments in related courses of *TPK4128 Industrial Mechatronics*, for example *TPK4125 Mechatronics*, which is the preceding course. This means that the Arduino UNOs can be used over multiple courses, which makes it more viable to buy for an assignment in the course.

Another great benefit of using an Arduino UNO, is the huge amount of internet support which is available for the device. If someone has any problem with the micro controller, then "google" it. Someone has most likely encountered the problem before, and posted a solution. This reduce the pressure on student assistants and the professor when the regular students are doing assignments. Most of their problems is going to have solutions online.

#### 8.4.2 UNO Disadvantages

- Needs additional hardware
- Needs customized drivers

The major disadvantage of using an Arduino UNO for the assignment is the lack of I/O. This can be changed by adding additional hardware in the form of I/O expanders as suggested in this thesis. However this isn't a perfect solution, this adds external hardware to the device. It also requires a hardware shield that needs to be developed for the combination of the units. With additional hardware, additional drivers on OpenPLC is needed. Neither OpenPLC or Arduino PLC IDE currently supports Arduino UNO with I/O expanders, which means the drivers have to be written. Therefore, there is still some development that needs to be done for this combination to be a viable solution. With more parts, there is an increased amount of parts that can ultimately fail. This means the reliability also can be compromised compared to the other units.

### 8.5 Arduino MEGA

The last micro controller that were tested, was the Arduino MEGA. The Arduino MEGA is basically an Arduino UNO with an upgraded number of I/O ports. Other than this, it is as barebone as a normal Arduino UNO and has all of the same functionalities with more I/O ports.

---

The advantages and drawbacks of running such a device is listed and discussed below in a similar fashion as before. A picture of the developed system can be seen in Figure 70.

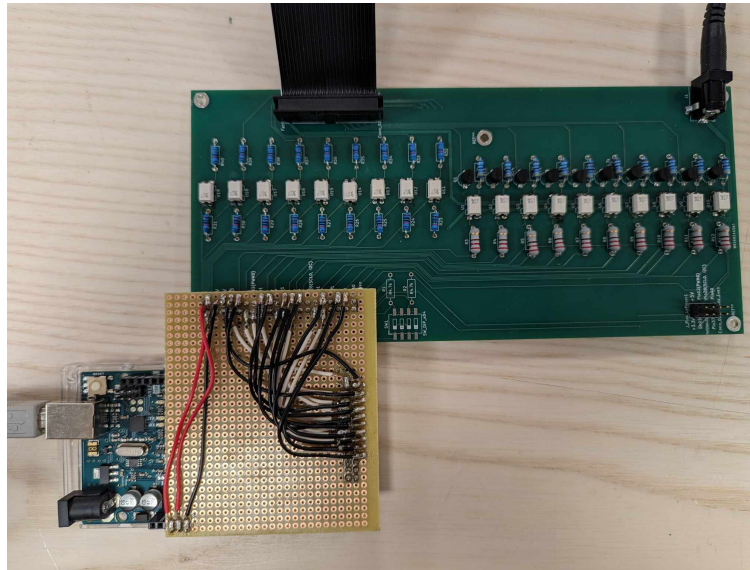


Figure 70: The Arduino MEGA with hardware shield connected to the PCB

### 8.5.1 MEGA Advantages

- Easy to use
- No need for external devices
- No custom drivers needed
- Can be used for more assignments

The first advantage of using an Arduino MEGA, is that it is simple to use and understand. There is a lot of support online, and it runs OpenPLC perfectly without the need to customize specialised drivers. Compared to the normal Arduino UNO, this has more than enough I/O ports to run the whole factory. This eliminates the need of using complicated solutions as customized drivers, or adding additional hardware to compensate for the lack of ports in any other way.

This also have the same capabilities and chipset as the original Arduino UNO. This means that this device is also capable of being used in other assignments, even in other courses. Similarly to what was described about the UNO in Section 8.4.1.

Similarly to the Arduino PMC, it is also easy to run code on the device. In OpenPLC, all of the I/O configurations are available from the start, and the code compiles and uploads directly to the device.

### 8.5.2 MEGA Disadvantages

- Expensive
- Can be cumbersome to connect without HW shield

---

Even though this seems like a better option compared to an Arduino UNO, the price is a major flaw. One unit of the Arduino MEGA costs 42 euros on the official Arduino Store. 20 of these units would cost roughly 9000 NOK with the exchange rate as per 10.12 - 2023. Making a for a total amount of roughly 11500 NOK with the PCBs counted for. The price of the PCBs were calculated in Section 8.4.1. The MEGA will then lay somewhere between the Raspberry PI and Arduino UNO in price.

With a proper hardware shield for the device like the prototype made in this project, should be able to connect directly to the PCB using a ribbon cable. Without such a shield, it can be a bit cumbersome to connect the MEGA to the factory. The MEGA has a lot of I/O ports, so the programming and deciding what pins to use can be tricky.

## 8.6 Using other devices

There is also an alternative to explore different devices for the mini-factory. As a test, a micro controller using the *ATMega 168* processor was tested to see if knock off Arduino products would be able to run the factory. An example of a knock off device is seen in Figure 71. This should in theory be able to run the factory, since it uses the same chipset as previous generation Arduino UNOs. Unfortunately, OpenPLC doesn't feature drivers for this chipset, since the drivers are updated to the newest version of the device at each update. This means that new drivers had to be written. It was decided that there was no desire to write drivers for outdated hardware.

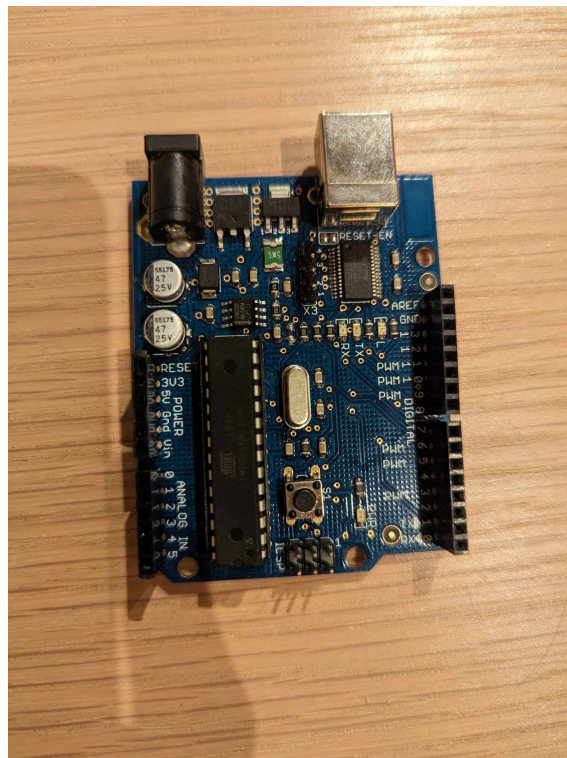


Figure 71: Picture of the unoriginal Arduino tested for the project

However, this doesn't mean it's impossible to use other micro controllers than the ones listed. There are several knock off Arduino UNOs that use the newer *ATMega 328* chipset. These will work like a normal Arduino UNO and should have no problem running code from OpenPLC. It

would also most likely work with the PCB that was made in this project. The input and output voltages required has a great voltage span, so the PCB would most likely be compatible with a large variety of different micro controllers.

## 8.7 Hardware choice

This brings us down to the first important question answered in this thesis:

Which micro-controller is the best choice for an assignment regarding PLC languages in Industrial Mechatronics?

This subchapter will discuss each separate micro controller regarding use and capabilities. It will also land a conclusion on which controller should be the best choice for use a potential assignment in *TPK4128 - Industrial Mechatronics*.

All of the advantages and disadvantages from the preceding chapters has been summed up in the following table seen in Table 2. This table includes some of the criterias, in addition to the arguments that were mentioned in the previous subchapters.

<b>Criteria</b>	<b>Arduino PMC H7</b>	<b>Raspberry PI</b>
Price	80000 NOK	25000 NOK
Work required for setup	Not much	OS patching and connecting to internet
Easy to use	Yes	No, requires uploading code to a runtime via internet
Extra features	Not many outside the task	A lot of extra features
Biggest flaw	Expensive	Complicated to use and set up
<b>Criteria</b>	<b>Arduino MEGA</b>	<b>Arduino UNO</b>
Price	11500 NOK	8000 NOK
Work required for setup	Need to connect a lot of pins	Make a hardware shield + I/O expander
Easy to use	Yes	Yes, with custom hardware
Extra features	Some extra features	Some extra features
Biggest flaw	Expensive considering features	Lacks I/O

Table 2: Comparison table

From this table it can be seen that all of the micro controllers have their strength and weaknesses. For example, the Arduino UNO is cheap to buy, Raspberry PI can be used for a multitude of different functions. Arduinio MEGA is decent at everything, and Arduino PMC is the best at doing the PLC emulation.

---

Choosing between these micro controllers can be tricky. Although the choice ultimately comes down to what qualities are desired for the device. The goal of this exercise is to see what micro controller could be the best at emulating a PLC for an assignment in the course *TPK4128 Industrial Mechatronics*. After all things are considered, this controller should be the Arduino H7 Portenta Machine Control (PMC). This unit already follows all of the standards set for modern PLCs. The code is open source, and is currently compatible with two different software programs. The main problem with this unit is pricing. However, compared to other PLC units from *Siemens* or other manufacturers, it is much cheaper to buy. There is also no need to buy a subscription that is needed to use the device. Which in total makes the unit much cheaper.

If there is a desire to make more assignments for the course, the Raspberry Pi can have great educational value. Students taking the course can learn to patch the Operating System, solder the PCB and learn how to connect to the Raspberry Pi via internet. All of these subjects are relevant to the course.

The other micro controllers have their strengths and weaknesses, but if the Indexed Line assignment is isolated from the rest of the assignments, the Arduino PMC is the best choice for the task. The RPi is too cumbersome to use for this task only, the Arduino MEGA is expensive considering the features that it currently offers, and the Arduino UNO needs more development to become a viable solution. There is also no need to make PCBs to interface the Arduino PMC with the factory.

## 8.8 PCB

The PCB that were first sent in for the project thesis was a prototype, and it was mentioned in the assignment that the PCB had room for improvement. With this Master thesis there was made a new PCB with some of the updated changes that were mentioned in the preceding project thesis. In addition, the input circuits were renewed to be able to work with the Arduino UNO and MEGA. There still is some features that can be added to the PCB for future works, but it's nothing major. The PCB that is delivered with this thesis should be fit for use in an assignment. Down below in Figure 72, this new updated PCB can be seen. In addition, a picture of the CAD model developed in *KiCad* is seen in Figure 73.

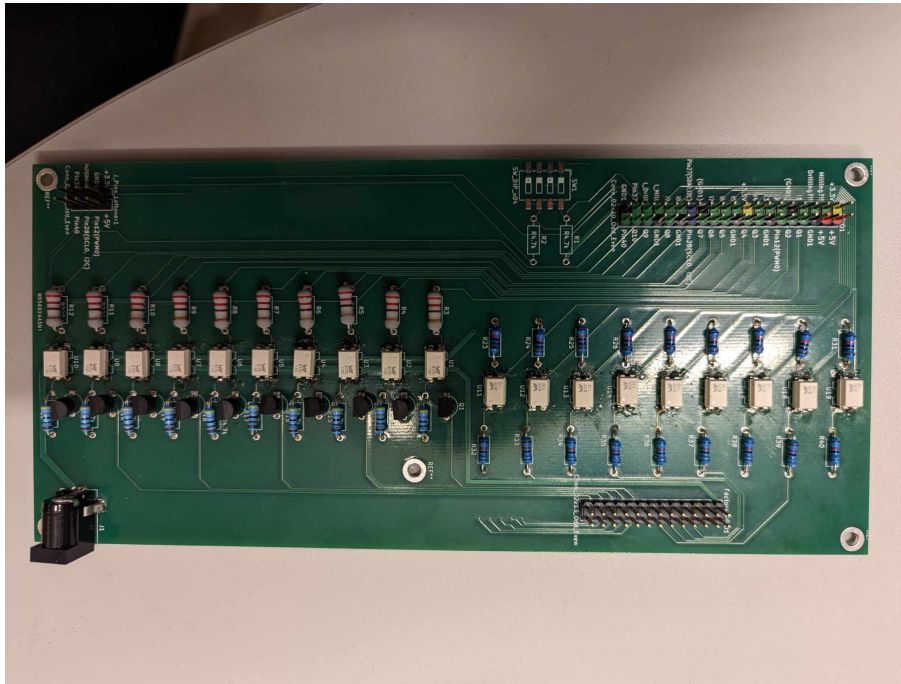


Figure 72: Picture of the updated PCB

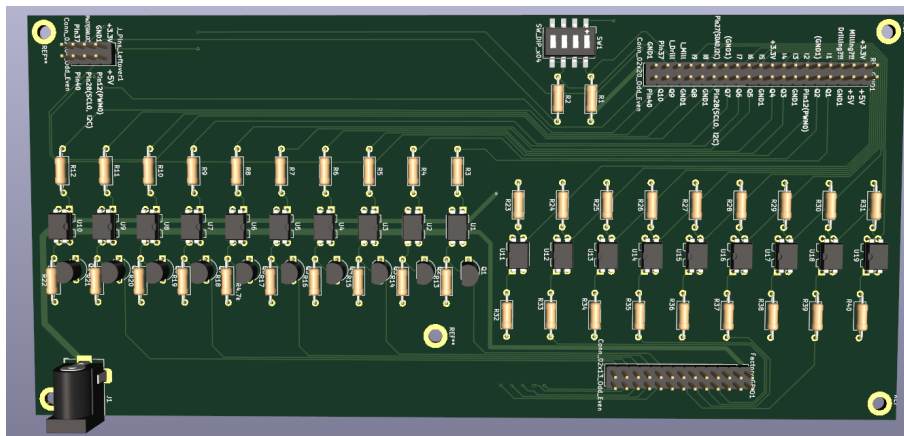


Figure 73: Picture of the updated PCB in KiCad

## 8.9 Software choice: OpenPLC vs Arduino PLC IDE

The second question that needs to be answered in this thesis, is what software to use for possible PLC assignments. Assuming the micro controller of choice is the Arduino PMC, there are two compatible softwares that can be used for an assignment. Therefore the difference of using the User Interface (UI) and reliability of the programs is gonna determine which program should be the preferred program for assignments.

### 8.9.1 User Interface and programming

As mentioned in Section 5.1, the programming in Arduino PLC IDE is a bit different than Open-PLC. In the Ladder Diagram code, the user is forced to use predefined lines by the program. This



---

might make programming easier for a newbie, but there were some problems implementing things that were easily implemented on OpenPLC. One example of this is code intersections. On Arduino PLC IDE it was written two different scripts depending on the milling and drilling variable. This would not be necessary on OpenPLC.

One thing that was better in Arduino PLC IDE however, is that it is possible to add several program files in the same project. On OpenPLC each program will have a different file. On Arduino PLC IDE all of the programs were set in the same folder, and it was easy to switch the files if the user wants to run different programs. Arduino PLC IDE also allows the possibility of running multiple files at a time. This can be handy if the mini-factory is connected to a larger system of conveyors and other mechatronic systems.

### 8.9.2 Reliability

Arduino PLC IDE has some bugs that impaired the user experience. For example, when writing the Ladder Diagram code, the timers didn't want to update without restarting the program. Another problem that were encountered on Arduino PLC IDE was that the program was crashing when trying to compile the code or connect to the device. As mentioned in Section 7.3.1, the conditionals didn't work for the SFC code, meaning there were no natural interrupts in the program. It seemed that the program were running faster than the Indexed Line when running, meaning the factory couldn't keep up. This compromised the sensor activation, and belts started suddenly running before they were told to. Therefore, it was harder to develop a code that would work correctly.

### 8.9.3 Final verdict

All things considered, OpenPLC is the program that should be preferred for doing PLC assignments. It's easier to understand and work with compared to Arduino PLC IDE, and is compatible with multiple controllers and Operating Systems. It also seems to be more reliable than the PLC IDE program.

If the institute is able to overcome the cost of buying the required units of this device, the micro controller chosen for such an assignment should be the Arduino PMC. It is easy to use and doesn't require specific hardware to interface the mini-factory. Otherwise there could be some value in using the Raspberry Pi, although this would acquire work over multiple assignments. Still, it can be a valuable learning experience for the students taking the course.

In the preceding project to this thesis, the system were shown to a group of students taking TPK4125 Mechatronics. Although this was the system using an older version of the PCB and a Raspberry Pi, the students showed interest in having something physical to play around with. This can further enhance the interest in the course and the curriculum. There is no reason to believe that the students would be less interested in using the system with a different hardware solution, therefore multiple platforms can be considered for a possible assignment for the course.

These results were aligned with *Sunds* findings in his own master's thesis, that students would prefer education with practical elements. In addition, having the ability to apply the theory learned in classes is valuable for the students for learning the curriculum.



---

## 9 Conclusion

All of the primary objectives from Section 1.3 has been met. Software drivers for the Arduino PMC has been developed and a working system is developed for both the Arduino PMC and Arduino MEGA. In addition, the PCB from the preceding project thesis has been updated to support Arduino MEGA and UNO.

In addition, some extensions and quality of life improvements were implemented successfully. These were not critical for the project, but will ease the process of turning this thesis into future exercises for Industrial Mechatronics.

Choosing a micro controller that should be used for future assignments, is entirely dependant on what qualities is wanted for the task. The Raspberry Pi fulfills all of the requirements envisioned for the assignment. If there's a need for a controller that can do it all, this is the best choice. The PCB is easy to connect both to the factory and the Raspberry Pi. This should be an adequate solution for use in an assignment.

However, if the mini-factory itself is isolated as an assignment, there are reasons to believe the best choice is using Arduino PMC unit with OpenPLC. The Arduino PMC is a highly specialised device, and is perfectly capable of running the mini-factory along with the PLC software. The code is easy to load into the device, and there is no need for a middle layer that adds complexity. Arduino PMC also follows all of the standards set for PLCs. There might be some value for the students to see a real PLC in the assignment, since this is widely used in the industry.

OpenPLC is the easier and more reliable software platform for developing PLC code. It also fully functions with the same micro controllers as Arduino PLC IDE. In addition OpenPLC provides support for a lot of micro controllers which aren't compatible with Arduino PLC IDE. More can also be added in the future, since this is an open-source software that upgrades the software regularly with several users of *GitHub* developing code for the program.

### 9.1 Further work and possible expansions

Some of these subchapters are still remaining from the preceding project assignment. But they are still relevant if there is a desire to continue adding functionalities to the system, and therefore it was decided relevant to add to this thesis. Section 9.1.4 is new for this thesis.

#### 9.1.1 PCB design

The PCB was updated in this project thesis, but there still is some additions to the PCB that can be considered. One of them is adding toggle switches to control the machining stations. This solution can make them easier to control, compared to using separate software on the RPi. Understanding the functionality is also easier with this solution. It can be implemented directly on the PCB, or on a daughterboard. The reason for using toggle switches instead of dip switches, was mentioned in the project assignment. The reason is that surface mount dip switch packets cannot withstand continuous use for a long time.

There is also a possibility to create another design that uses surface mount components instead. This can be a good solution if there is a need for many boards, since the PCBs can be designed to

---

be smaller. They can also be produced with the components already soldered on. This is a good solution in the example of; if a professor needs 40 of the boards to use in their subject.

In addition, another connection point could be added to the PCB to directly connect Arduino to the PCB, although this requires custom hardware shields.

### 9.1.2 Machine learning

Machine learning and camera vision are modules that can be added to the assembly as future expansions. There are existing camera modules that can be added onto a Raspberry Pi, on which OpenCV can be used to recognise different objects that travels on the conveyors. The machining modules are already run by a separate program written in *Python*. This program can be replaced with a program running image recognition to decide what operations should be done on the part. For this task, it is possible to send different objects on the conveyors, and the program decides if the part is going to be milled, drilled, both, or none, based on the shape, color or other distinct characteristics.

### 9.1.3 Robot arm and ROS2

Another big part of the Industrial Robotics course is to learn the basics of ROS and robot control. There are also current assignments in the course related to these topics. ROS2 can also be implemented in an assignment using the mini-factory. One way to do this is to implement a small robot arm to put objects on the conveyor belts. This is a great opportunity to possibly automate a currently manual operation.

### 9.1.4 Developing the UNO system

One option to further cut the cost of the assignments, is to add a proper support system for the Arduino UNO or a similar knock off variant using the same chip. Unfortunately, there weren't enough time in the semester to implement the hardware solution required to run the Arduino UNO properly for the task. If such a solution is desired, there's multiple steps that should be taken for making a properly functioning system. The first step would be to make a custom hardware shield for the Arduino UNO and adding an I/O expander to this shield. The *SparkFun SX150* should be a solid option for the task due to loads of information and support about the expander being available online. Other expanders should still be capable of doing the task. In addition, the software drivers for the system needs to be written. There is currently no support for the Arduino UNO with an I/O expander on OpenPLC.

---

## Bibliography

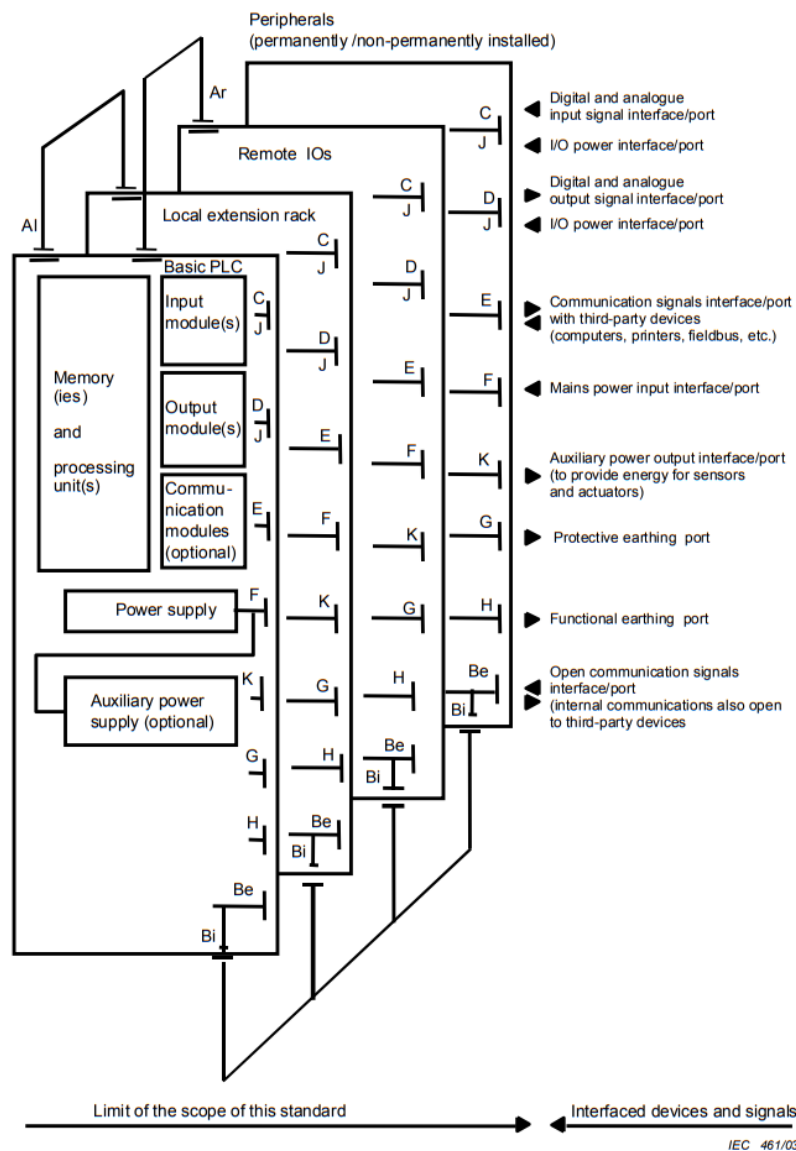
- [1] A. Skavhaug, *Tpk4128 - industrial mechatronics*, 2022. [Online]. Available: <https://www.ntnu.edu/studies/courses/TPK4128>.
- [2] F. GmbH, *Indexed line with two machining stations 24v*, 2022. [Online]. Available: <https://www.fischertechnik.de/en/products/learning/training-models/96790-edu-indexed-line-with-two-machining-stations-24v>.
- [3] A. K. Sund, ‘Improving mechatronics education with design thinking’, 2022.
- [4] A. C. Arnholm and M. N. Henriksen, ‘Combining industry 4.0 and 5g connectivity with robots in digital production factories’, 2021.
- [5] W. Bolton, *Programmable logic controllers*. Newnes, 2015.
- [6] Wikipedia, *File:originating register, number five crossbar switching system (museum of communications, seattle).jpg*, 2007. [Online]. Available: [https://en.wikipedia.org/wiki/File:Originating\\_Register,\\_Number\\_Five\\_Crossbar\\_Switching\\_System\\_\(Museum\\_of\\_Communications,\\_Seattle\).jpg](https://en.wikipedia.org/wiki/File:Originating_Register,_Number_Five_Crossbar_Switching_System_(Museum_of_Communications,_Seattle).jpg).
- [7] M. G. Hudedmani, R. Umayal, S. K. Kabberalli and R. Hittalamani, ‘Programmable logic controller (plc) in automation’, *Advanced Journal of Graduate Research*, vol. 2, no. 1, pp. 37–45, 2017.
- [8] E. R. Alphonsus and M. O. Abdullah, ‘A review on the applications of programmable logic controllers (plcs)’, *Renewable and Sustainable Energy Reviews*, vol. 60, pp. 1185–1205, 2016.
- [9] G. Frey and L. Litz, ‘Formal methods in plc programming’, in *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0, vol. 4, 2000, 2431–2436 vol.4. DOI: 10.1109/ICSMC.2000.884356*.
- [10] Standard-Norge, *Nek iec 61131-3:2013*, 2013. [Online]. Available: <https://www.standard.no/no/Nettbutikk/produktkatalogen/Produktpresentasjon/?ProductID=627454>.
- [11] IEC, ‘Programmable controllers – part 1: General information’, en, International Electrotechnical Commission, Geneva, CH, Standard IEC 61131-1:2003, 2003. [Online]. Available: <https://webstore.iec.ch/publication/4550>.
- [12] IEC, ‘Programmable controllers – part 3: Programming languages’, en, International Electrotechnical Commission, Geneva, CH, Standard IEC 61131-3:2013, 2013. [Online]. Available: <https://webstore.iec.ch/publication/31007>.
- [13] IEC, ‘Industrial-process measurement and control – programmable controllers – part 2: Equipment requirements and tests’, en, International Electrotechnical Commission, Geneva, CH, Standard IEC 61131-2:2017, 2017. [Online]. Available: <https://webstore.iec.ch/publication/31007>.
- [14] A.-D. R.H.Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, 2023.
- [15] Redhat.com, *Working with the real-time kernel for red hat enterprise linux*, 2022. [Online]. Available: <https://www.redhat.com/sysadmin/real-time-kernel>.
- [16] Raspberry-Pi-foundation, *Raspberrypi.com*, 2022. [Online]. Available: <https://www.raspberrypi.com/>.

- 
- [17] thiagorvalves, *Tpk4128 - industrial mechatronics*, 2022. [Online]. Available: [https://github.com/thiagorvalves/OpenPLC\\_v3](https://github.com/thiagorvalves/OpenPLC_v3).
- [18] Arduino, *Arduino plc ide. boost production and building automation with your own industry 4.0 control system*, 2023. [Online]. Available: <https://www.arduino.cc/pro/software-plc-ide>.
- [19] Arduino, *Programming introduction with arduino plc ide*, 2023. [Online]. Available: <https://docs.arduino.cc/software/plc-ide/tutorials/plc-programming-introduction>.
- [20] A. Wordpress, *Connecting raspberry pi to eduroam*, 2016. [Online]. Available: <https://autottblog.wordpress.com/raspberry-pi-arduino/connecting-raspberry-pi-to-eduroam/>.
- [21] OpenPLC, *1.4 installing openplc runtime on linux*, 2022. [Online]. Available: <https://openplcproject.com/docs/installing-openplc-runtime-on-linux-systems/>.
- [22] F. GmbH, *Technical faq's*, 2022. [Online]. Available: <https://www.fischertechnik.de/en/service/faq/technical-faqs>.
- [23] *Simatic s7-1500 signal modules*, 2022. [Online]. Available: <https://new.siemens.com/global/en/products/automation/systems/industrial/plc/simatic-s7-1500/signal-modules.html>.
- [24] Arduino, *Arduino® portenta machine control*, 2023. [Online]. Available: <https://docs.arduino.cc/resources/datasheets/AKX00032-datasheet.pdf>.
- [25] Arduino, *Arduino® uno r3*, 2023. [Online]. Available: <https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf>.
- [26] *Resistor applications: Pull-up and pull-down resistor*, 2023. [Online]. Available: <https://eepower.com/resistor-guide/resistor-applications/pull-up-resistor-pull-down-resistor/#>.
- [27] Vishay, *Optocoupler, phototransistor output, high reliability*, 2015. [Online]. Available: <https://www.vishay.com/docs/83740/sfh617a.pdf>.
- [28] OpenPLC, *Openplc runtime overview*, 2022. [Online]. Available: <https://openplcproject.com/docs/2-1-openplc-runtime-overview/>.
- [29] OpenPLC, *2.4 physical addressing*, 2022. [Online]. Available: <https://openplcproject.com/docs/2-4-physical-addressing/>.
- [30] S. B. Reddy, *Instrumentation tools*, 2022. [Online]. Available: <https://instrumentationtools.com/what-is-sequential-function-chart-sfc/>.
- [31] CODESYS, *Qualifiers for actions in sfc*, 2023. [Online]. Available: [https://help.codesys.com/api-content/2/codesys/3.5.14.0/en/\\_cds\\_sfc\\_action\\_qualifier/#id1](https://help.codesys.com/api-content/2/codesys/3.5.14.0/en/_cds_sfc_action_qualifier/#id1).

---

## Appendix

## A PLC architecture from IEC 61131-2



### Key

- AI Communication interface/port for local I/O
- Ar Communication interface/port for remote I/O station
- Be Open-communication interface/port also open to third-party devices (for example, personal computer used for programming instead of a PADT)
- Bi Internal communication interface/port for peripherals
- C Interface/port for digital and analogue input signals
- D Interface/port for digital and analogue output signals
- E Serial or parallel communication interfaces/ports for data communication with third-party devices
- F Mains power interface/port. Devices with F ports have requirements on keeping downstream devices intelligent during power-up, power-down and power interruptions.
- G Port for protective earthing
- H Port for functional earthing
- J I/O power interface/port used to power sensors and actuators
- K Auxiliary power output interface/port

Figure 74: Typical interface/port diagram of a PLC-system (from IEC 61131-2)[13]

---

## B Eduroam Guide

This Guide is exactly the same as the one from [20], but is added to make it possible to recreate our testing with only reading this document. It is tested to work on Eduroam at NTNU's campus Gløshugen as of December 2022.

First, we must add a few lines of text in the file `/etc/wpa_supplicant/wpa_supplicant.conf` (it must be done with root permission):

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Then append the following lines (yes, change username and password to something appropriate)

```
network={
  identity="username@ntnu.no"
  password="password"
  eap=PEAP
  phase1="peaplabel=0"
  phase2="auth=MSCHAPV2"
  priority=999
  disabled=0
  ssid="eduroam"
  scan_ssid=0
  mode=0
  auth_alg=OPEN
  proto=RSN
  pairwise=CCMP
  key_mgmt=WPA-EAP
  proactive_key_caching=1
}
```

Then hit `< control > +x`, then `y` and `< enter >` to save and exit.

Depending on your version of Pi and your Pi's operating system you might or might not have a connection now (check with `ifconfig`). If you do not, you should try to stop networking and start `wpa_supplicant`:

```
sudo service networking stop
```

```
sudo wpa_supplicant -i wlan0 -c
↳ /etc/wpa_supplicant/wpa_supplicant.conf -B
```

If you still don't have a connection you should try to reboot

---

```
sudo reboot
```

Still no connection? Check that all of the special characters in `/etc/wpa_supplicant/wpa_supplicant.conf` have been copied correctly, for example

```
' '
```

is not the same as

```
" "
```

Is Eduroam being stubborn? You can do as we did, search around a bit and try tweaking the settings in `/etc/wpa_supplicant/wpa_supplicant.conf` until you win.



---

## C Milling, Drilling or not code Example

This is the the code we used to control the mill and drill. It is written fast to make it possible to choose which states is desired without having to re run the program each time. This can be written more efficiently, but that was not the purpose of this code. It should just work, which it does.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)

mill = 3
drill = 5

GPIO.setup(mill, GPIO.OUT)
GPIO.setup(drill, GPIO.OUT)

m = ''
d = ''
cont = ''

while True:
    while True:
        m = input('Type y for milling and n for not milling: ')
        if m == 'y':
            break
        elif m == 'n':
            break

    while True:
        d = input('Type y for dilling and n for not dilling: ')
        if d == 'y':
            break
        elif d == 'n':
            break

    if m == 'y':
        GPIO.output(mill, GPIO.HIGH)
        print('m->high')
    else:
        GPIO.output(mill, GPIO.LOW)
        print('m->low')

    if d == 'y':
        GPIO.output(drill, GPIO.HIGH)
```

---

```
    print('d->high')
else:
    GPIO.output(drill, GPIO.LOW)
    print('d->low')

while True:
    cont = input('Type y for continue and n for quitting: ')
    if cont == 'y':
        break
    elif cont == 'n':
        break
if cont == 'n':
    break
```

---

## D Attachments

### D.1 Attached files:

- ArduinoPLCIDE\_PMC.zip - Arduino PLC IDE project files
- Factory\_run\_ArduinoMEGA.mp4 - Video of the factory running with the Arduino MEGA
- Factory\_run\_ArduinoPMC.mp4 - Video of the factory running with the Arduino PMC
- machine\_control.cpp - The custom hardware driver for the Arduino PMC in OpenPLC
- OpenPLC\_Arduino\_MEGA\_LD.zip - Ladder diagram in OpenPLC with the addresses for the MEGA
- OpenPLC\_Arduino\_PMC\_LD.zip - Ladder diagram in OpenPLC with the addresses for the Arduino PMC
- OpenPLC\_SFC\_code.zip - SFC code in OpenPLC, written with RPi addresses
- PCB\_with\_opto\_master.zip - The PCB files

### D.2 Hardware:

The mini-factory with the prototype.

---

## E Extended Raspberry Pi OpenPLC table

Pin	Name	OpenPLC address	PCB functionality		PCB functionality	OpenPLC address	Name	Pin
1	3.3V DC Power	-	3.3V	□	5V	-	5V DC Power	2
3	GPIO02 (SDA1, I2C)	%IX0.0 *	Milling?!!	●	5V	-	5V DC Power	4
5	GPIO03 (SDL1, I2C)	%IX0.1 *	Drilling?!!	●	GND1		Ground	6
7	GPIO04 (GPCLK0)	%IX0.2	Btn_S1_front	●	Slider1_forw	%QX0.0	GPIO14 (TXD0, UART)	8
9	Ground	-	GND1	●	Slider1_back	%QX0.1	GPIO15 (RXD0, UART)	10
11	GPIO17	%IX0.3	Btn_S1_rear	●	Pin12 pass through	%QW0	GPIO18 (PWM0)	12
13	GPIO27	%IX0.4	Btn_S2_front	●	GND1	-	Ground	14
15	GPIO22	%IX0.5	Btn_S2_rear	●	Slider2_forw	%QX0.2	GPIO23	16
17	3.3V DC Power	-	3.3V	●	Slider2_back	%QX0.3	GPIO24	18
19	GPIO10 (SP10_MOSI)	%IX0.6	Sensor2_conv1	●	GND1	-	Ground	20
21	GPIO09 (SP10_MISO)	%IX0.7	Sensor3_conv2	●	Conv1	%QX0.4	GPIO25	22
23	GPIO11 (SP10_CLK)	%IX1.0	Sensor1_Start	●	Conv2	%QX0.5	GPIO08 (SPI0_CE0_N)	24
25	Ground	-	GND1	●	Mill	%QX0.6	GPIO07 (SPI0_CE1_N)	26
27	GPIO00 (SDA0, I2C)	-	Pin27 pass through	●	Pin28 pass through	-	GPIO07 (SDL0, I2C)	28
29	GPIO05	%IX1.1	Sensor4_conv3	●	GND1	-	Ground	30
31	GPIO06	%IX1.2	Sensor5_conv4	●	Conv3	%QX0.7	GPIO12 (PWM0)	32
33	GPIO13 (PWM1)	%IX1.3	I_Mill	●	GND1	-	Ground	34
35	GPIO19	%IX1.4	I_Drill	●	Drill	%QX1.0	GPIO16	36
37	GPIO26	%IX1.5	Pin37 pass through	●	Conv4	%QX1.1	GPIO20	38
39	Ground	-5	GND1	●	Pin40 pass through	%QX1.2	GPIO21	40



Kunnskap for en bedre verden

DEPARTMENT OF MECHANICAL AND INDUSTRIAL  
ENGINEERING

TPK4560 - PROJECT ASSIGNMENT ROBOTICS AND AUTOMATION

---

# Combining Mechatronics and Opensource software to create a Programmable Logic Controller alternative for educational use

---

*Authors:*

Lars Bonvik

Eskild Godli

*Supervisor:*

Amund Skavhaug

December, 2022

---

## Preface

Doing this project thesis has been an interesting and fun experience, where we have been able to use a lot of the information learnt in different classes of the year. Due to COVID-19, Industrial Mechatronics was not as good as it should have been, and it's been little doubt that the current assignments in the class could be improved. Applying theory in a project environment is a rewarding experience, and the ability to contribute, making mechatronics more interesting for future students is meaningful for us.

We would like to extend a special thanks to the following people:

- Amund Skavhaug, our supervisor for support and guidance before and during the project. His inputs have been invaluable for the progress and finalization of the project.
- Håvard Vestad for his guidance on the risk assessment used to get access to the mechatronics lab.
- Lars Tingelstad for his guidance on the simulation topic.

We also would like to thank our classmates writing project thesis along side us. Planning social events and discussing relevant and irrelevant topics around the project assignment keeping the spirits high.

---

## Summary

This is a project thesis written by Lars Bonvik and Eskild Godli in the autumn semester of 2022, and is a part of the penultimate semester of the master program for Mechanical Engineering at the Norwegian University of Science and Technology, NTNU. This thesis studies how to use a Raspberry Pi microcontroller to emulate a Programmable Logic Controller. The thesis contains both software and hardware solutions, making this possible. The work is based on Andreas Knudsen Sunds discoveries in his master's thesis, for the need to improve the assignments in the course TPK4128 Industrial Mechatronics.

The project started with the research of how a Programmable Logic Controller work as well as the workings of Fischertechnik's mini-factory used for the project.

It was researched and found open-source software that could make sure that with an added hardware solution, could emulate a Programmable Logic Controller. The hardware solution was developed and researched through the thesis, and rapid prototypes on breadboards created to test the researched solutions. With a satisfactory solution developed, a prototype Printed Circuit Board (PCB) was designed and produced. The PCB acts as the communication layer between the Raspberry Pi and the "Fischertechnik Indexed Line with two Machining Stations 24V" provided by the supervisor for use in exercises in Industrial Mechatronics.



---

## Acronyms

**BJT** - Bipolar Junction Transistor

**I/O** - Input and Output

**LD** - Ladder Diagram

**MOSFET** - The metal-oxide-semiconductor field-effect transistor

**NPN** - Negative, Positive, Negative

**OPC UA** - Open Platform Communications Unified Architecture

**OpenCV** - Open Source Computer Vision Library

**OS** - Operating System

**PLC** - Programmable Logic Controller

**PNP** - Positive, Negative, Positive

**ROS** - Robot Operating System

**RPi** - Raspberry Pi

**RT** - Real Time

**SFC** - Sequential Function Chart

---

# Table of Contents

<b>List of Figures</b>	<b>vii</b>
<b>List of Tables</b>	<b>viii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and Project Description . . . . .	1
1.2 Previous Work . . . . .	1
1.3 Objective . . . . .	2
1.4 Structure of thesis . . . . .	2
<b>2 Background theory</b>	<b>4</b>
2.1 Programmable Logic Controller (PLC) . . . . .	4
2.1.1 Control systems and controllers . . . . .	4
2.1.2 The functions of PLCs . . . . .	5
2.1.3 The Hardware and Architecture of PLCs . . . . .	6
2.1.4 PLC Inputs and Outputs . . . . .	7
2.1.5 PLC Manufacturers and Models . . . . .	8
2.2 Real-time kernel for linux . . . . .	8
2.3 Raspberry Pi . . . . .	9
2.4 OpenPLC . . . . .	9
2.5 OPCUA . . . . .	9
2.6 MultiSim . . . . .	10
2.7 Visual Components . . . . .	10
<b>3 The setup of the Raspberry Pi</b>	<b>11</b>
3.1 Setup of real-time kernel on Raspberry Pi . . . . .	11
3.2 Installing OpenPLC runtime on the Raspberry Pi . . . . .	15
<b>4 Technical specifications</b>	<b>17</b>
4.1 The interface circuitry . . . . .	17
4.2 Specifications of the mini-factory . . . . .	18
4.3 PLC standards . . . . .	20

---

4.4	Testing with other microcontrollers . . . . .	21
<b>5</b>	<b>Circuitry</b>	<b>22</b>
5.1	Push-pull circuit version 1 . . . . .	22
5.2	Push-Pull version 1 evaluation . . . . .	23
5.3	IO signal circuit . . . . .	23
5.4	IO signal circuit evaluation . . . . .	24
5.5	Push-pull version 2 . . . . .	24
5.6	Push-Pull version 2 evaluation . . . . .	25
5.7	MOSFET circuit . . . . .	25
5.8	MOSFET circuit evaluation . . . . .	26
5.9	Sensor circuits . . . . .	26
5.10	Sensor circuit with optocoupler . . . . .	27
5.11	Sensor circuits evaluation . . . . .	27
5.12	Breadboard circuit . . . . .	27
5.13	PCB circuit . . . . .	28
5.14	PCB version 1 . . . . .	30
5.15	PCB version 1 soldering . . . . .	32
5.16	PCB version 2 . . . . .	33
5.17	PCB version 3 - Final version . . . . .	33
5.18	OpenPLC . . . . .	34
<b>6</b>	<b>Software and Simulation</b>	<b>36</b>
6.1	Connecting to Raspberry Pi . . . . .	36
6.2	Programming and language . . . . .	36
6.3	Ladder Diagram Code . . . . .	37
6.3.1	LD Code part 1 . . . . .	38
6.3.2	LD Code part 2 . . . . .	41
6.3.3	LD Code part 3 . . . . .	42
6.3.4	LD Code part 4 . . . . .	43
6.3.5	Discussing the Ladder Diagram Code . . . . .	43
6.4	SFC . . . . .	44

---

---

6.4.1	SFC code part 1 and 2 . . . . .	44
6.4.2	SFC code part 3 and 4 . . . . .	46
6.4.3	SFC code part 5 and 6 . . . . .	47
6.5	Simulation . . . . .	48
6.5.1	Modelling . . . . .	48
6.6	Connecting to VC . . . . .	49
6.6.1	Possible solutions . . . . .	49
6.6.2	Implement FreeOPC . . . . .	50
6.6.3	Implement <i>OPCUA</i> into <i>OpenPLC</i> . . . . .	50
6.6.4	TCP/IP addressing . . . . .	51
6.7	PCB design software . . . . .	51
<b>7</b>	<b>Discussion</b>	<b>52</b>
7.1	Use of mini-factory in Industrial Mechatronics . . . . .	52
7.2	The hardware . . . . .	52
7.3	PCB . . . . .	53
7.4	Multisim . . . . .	53
7.5	CONFIG_KVM . . . . .	53
7.6	The final results . . . . .	54
7.7	Further work and possible expansions . . . . .	54
7.7.1	Simulation . . . . .	54
7.7.2	PCB design . . . . .	54
7.7.3	Machine learning . . . . .	55
7.7.4	Robot arm and ROS2 . . . . .	55
<b>8</b>	<b>Conclusion</b>	<b>56</b>
	<b>Bibliography</b>	<b>57</b>
	<b>Appendix</b>	<b>59</b>
<b>A</b>	<b>PLC architecture from IEC 61131-2</b>	<b>60</b>
<b>B</b>	<b>Eduroam Guide</b>	<b>61</b>

---

---

<b>C Milling, Drilling or not code Example</b>	<b>63</b>
<b>D Attachments</b>	<b>65</b>
D.1 Attached files: . . . . .	65
D.2 Hardware: . . . . .	65
<b>E Extended Raspberry Pi OpenPLC table</b>	<b>66</b>

## List of Figures

1 The Fischertechnik factory [2] . . . . .	1
2 Example of relay control system . . . . .	4
3 Simple illustration of a PLC . . . . .	5
4 The general architecture of PLCs . . . . .	6
5 Architecture of PLC from IEC 61131 . . . . .	7
6 Siemens Simatic S7-1500 PLC . . . . .	8
7 Fully preemptible . . . . .	14
8 The PLC architecture map . . . . .	17
9 Rapid interface circuit . . . . .	18
10 Digital outputs for direct current table . . . . .	20
11 Circuit for 2.7-5.5V signals . . . . .	22
12 IO signal activation . . . . .	24
13 Push-pull circuit version 2 . . . . .	25
14 MOSFET circuit with octocoupler . . . . .	26
15 Neatly connected breadboard . . . . .	27
16 The final Breadboard . . . . .	28
17 PCBv1 Schematic . . . . .	31
18 PCBv1 images . . . . .	32
19 PCBv1 soldered images . . . . .	33
20 PCBv3 images . . . . .	34
21 RPi Pin1 Location . . . . .	35
22 Factory from top . . . . .	37

---

23	PLC code variables . . . . .	38
24	Complete LD code . . . . .	39
25	Simple test of Ladder Diagram. . . . .	39
26	Part 1 birdview . . . . .	40
27	Ladder logic part 1 . . . . .	40
28	Part 2 in birdview . . . . .	41
29	Ladder diagram part 2 . . . . .	41
30	Ladder diagram alternate layout . . . . .	42
31	Part 3 of factory in birdview . . . . .	42
32	Ladder diagram part 3 . . . . .	42
33	The Factory in bird view, showing part 4. . . . .	43
34	The fourth part of the Ladder Diagram. . . . .	43
35	SFC code part 1 . . . . .	45
36	SFC code part 2 . . . . .	45
37	SFC code part 3 . . . . .	46
38	SFC code part 4 . . . . .	46
39	SFC code part 5 . . . . .	47
40	SFC code part 6 . . . . .	48
41	Factory model in Visual Components . . . . .	49
42	Connection OpenPLC to Visual Components . . . . .	51
43	Typical interface/port diagram of a PLC-system (from IEC 61131-2)[13] . . . . .	60

## List of Tables

1	Datasheet mini-factory . . . . .	19
2	Required specifications . . . . .	22
3	Comparison MOSFET vs BJT . . . . .	26
4	Overview of the layout on the RBPi . . . . .	34

---

# 1 Introduction

## 1.1 Motivation and Project Description

The goal of this project is to make the learning material more concrete and realistic for students taking the class TPK4128 Industrial Mechatronics. A course that teaches the students more about Industry 4.0, and more specifically mechatronics for industrial production systems. The course description from NTNU's website:

"The course is on mechatronics for industrial production systems. This includes the implementation, use, and programming of single-board computers, PLC-based and other industrial computer systems. Embedded- and real-time systems, industrial bus systems, interfacing, operating systems and communication protocols for these. Use of C, Linux and TCP/IP on e.g. Raspberry Pi, Python, ROS, virtual machines, computer vision, OPC-UA and selected Industry 4.0 topics will be taught and practiced. Furthermore, sensors, actuators, power supplies, motor drives, pneumatic and hydraulic actuators, aspects of dependability for industrial computer systems, and development methodologies. The students will get practical skills through extensive, weekly laboratory exercises with a focus on practical programming." [1]

The goal for this project is to make the necessary equipment to improve assignments, making them more interesting and relevant. The part of the task given in this project is to run a "mini-factory" using a Raspberry Pi, which task will be to emulate a PLC (Programmable Logic Controller). This is due to supervisor having troubles acquiring PLCs. The project should also be able to be used as stand-alone, in addition to being expandable with future projects.

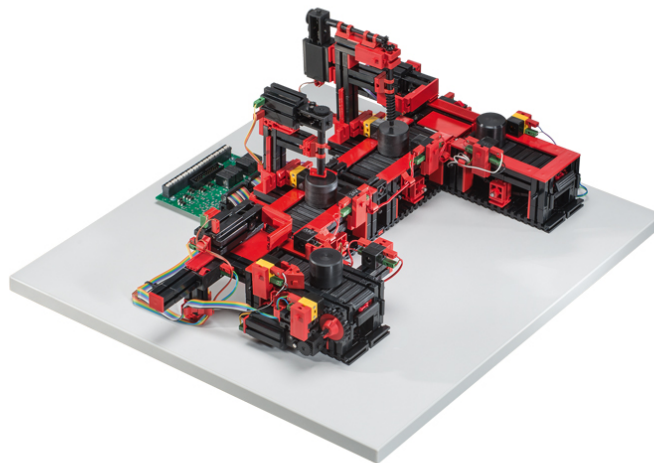


Figure 1: The Fischertechnik factory with two machining stations [2]

## 1.2 Previous Work

Use of the mini-factory from *Fischertechnik* in assignments has already been explored in the Master's thesis by *Andreas Knudssen Sund* [3]. In this project, it was investigated if there was a need

---

for new assignments in *Industrial Mechatronics*. It was also investigated if tasks regarding the Fischertechnik mini-factory could be appropriate to replace the already existing assignments given in the course.

In *Sunds* research [3], it was found that the assignments in the course were ready for an upgrade. A thorough study has been done to see what the impact of introducing the mini-factory as an assignment in Industrial Mechatronics would be. The conclusion of the study was that an assignment relating to the mini-factory is probably going to be beneficial for the students learning. This would also benefit the students experience and perception of the selected topics.

*Sund* was mainly investigating if this factory was appropriate to use in assignments with a PLC. After getting promising results, it was decided that this project assignment will develop the concepts further with a Raspberry Pi and *OpenPLC*, which makes the institute not dependant on 3rd party manufacturers to complete the assignments made for the course. This would save a lot of headaches regarding licensing and the process to buy PLCs.

Another research paper that has been used regarding this project is the master's thesis of *Arnholm and Henriksen* from 2021 [4]. This paper was mainly focusing on the use of Raspberry Pi with a 5G hat. Using a Raspberry Pi for a mechatronic system with *Linux* requires a pre-emptive kernel. Therefore, a guide from this paper was used to build such a kernel for the Raspberry Pi.

### 1.3 Objective

The main objective of this project assignment, is to improve the assignments already used in *TPK 4128 Industrial Mechatronics*. In this project thesis it is confined to improving the assignment about PLC programming. This will be achieved by using a Raspberry Pi as a PLC to control the mini-factory. The different objectives that were set for this project to achieve this was:

- Make a Raspberry Pi emulate PLC hardware
- Make a Raspberry Pi emulate PLC software
- Make a complete working prototype
- Simplify the system enough to be made into a relevant assignment

The secondary objectives:

- Make a simulation of the mini-factory
- Make the solution expandable
- Make it possible to use other microcontrollers

### 1.4 Structure of thesis

**Section 2** introduces the theory and key software used in this thesis. It begins with the presentation of what a Programmable Logic Controller is, and continues with how it works.



---

**Section 3** presents how the Raspberry Pi used in this thesis is set up to work in the desired way.

**Section 4** details the functionality that is needed and desired to achieve the objectives set for the thesis. Further it continues with the tests of the mini-factory, and the discoveries of how it functions.

**Section 5** goes through the development of the circuitry needed to run the mini-factory with the Raspberry Pi.

**Section 6** contains the softwares used to develop the prototype and goes through the development of the PLC code used in it. It also contains the start of the simulation development for the mini-factory.

**Section 7** is discussing the overall results and difficulties. Further works is also mentioned.

**Section 8** contains the conclusion of the project thesis.

**Appendix A** a figure showing a more advanced architecture representation of a PLC from the PLC standard IEC 61131-2.

**Appendix B** a guide to connect to the Eduroam network with an Raspberry Pi.

**Appendix C** presents the test program written to control the machining stations of the mini-factory.

**Appendix D** a listing of the attached files.

**Appendix E** is an extended version of Table 4.

---

## 2 Background theory

This chapter includes the background theory needed to understand the work done in the rest of this project thesis.

### 2.1 Programmable Logic Controller (PLC)

”A programmable logic controller (PLC) is a type of device extensively used for different automation applications within industrial processes and manufacturing” [5]. As it’s name implies, it is a form of controller. This section will give an overview of controllers in general, as well as presenting the function, hardware and architecture of PLCs.

#### 2.1.1 Control systems and controllers

A controller or control system ”might be required to control a sequence of events, maintain some variable constant, or follow some prescribed change” [5]. They are used to automate and streamline tasks that were done manually by people, to drive cost down, and make a safer workplace by automating hazardous tasks.

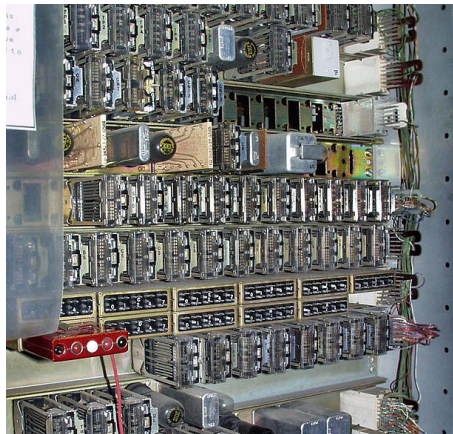


Figure 2: Example of a relay control system used in the Number Five Crossbar Switching System. This unit is in the museum of communication in Seattle. [6]

A popular way to automate tasks before the time of the PLCs, were the use of relay systems. An example of such a system can be seen in Figure 2.

Relays are magnetic switches that is switched on and off depending on the voltage of an input signal. This was an ideal system to automate tasks requiring high precision and tight time constraints. They excel at simple tasks, but are unable to do more complex tasks. However there are some significant problems with relay switches. One of them is that tasks are hard to modify. A small change in function might need a complete rewiring of the whole system. Due to all of these physical connections, relay systems also requires a lot of space. [5], [7], [8]

Microprocessor control systems are a much more modern alternative. Instead of needing to hard-wire the control system for each situation, it’s possible to simply reprogram the microprocessor for the specific constraints and functions of a task. This type of control makes the system a lot

---

more flexible than relay switches. Microprocessors are also cheap and space efficient compared to alternative solutions, which makes these systems a preferable way to automate industrial tasks.

PLCs are a specific form of microprocessor controllers, made to standardise microcontroller systems with simple and robust programming languages. Mainly for use in the industry. One of the standardised languages used for PLCs are called Ladder Diagram, this language was developed to be used by people who had originally wired relay systems. This way it was simple for the programmers to adapt to PLC systems instead, since it is made to be similar to relay system schematics.

In 1969 the first PLC was developed, and it has been the primary solution for industrial automation since the 1970s. They have evolved from self-contained units with few digital I/O, to modular units with the possibilities to expand the I/O. They are able to use analog I/O as well.

Since it is developed new programming languages specific for the PLC's, the need for international standardisation were large, because of the importance of the industrial applications they are used for.

This have led to the creation of many standards, such as the most influential ones from the 90s and early 2000s, IEC 1131 and IEC 61499 respectively. The IEC 1131 standard was later renamed to IEC 61131, and got new extensions. This standard has now ten parts, but started with three parts released in 1992 and 1993. The rest of the parts has been released sporadically since then. Multiple parts were released in the year of 2000, and the newest part was released in 2019. [9], [5], [10], [11], [12], [7].

### 2.1.2 The functions of PLCs

As said before, a PLC is a complete system with a microprocessor and I/O designed for use in the industry. It uses simple languages, which makes them easy to program for engineers with little to no experience with programming. In figure Figure 3 it is added a simple illustration of how a PLC functions. The PLC get some inputs from sensors, the program interprets them, and some output signals are set to run some motors that for instance runs a conveyor belt.

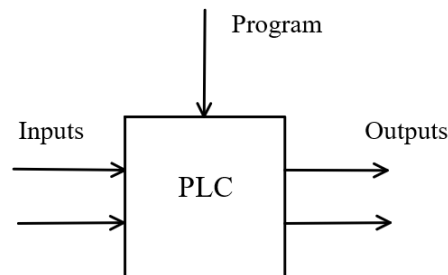


Figure 3: Simple illustration of a PLC

The PLCs are developed for use in the industry. Some of the tasks they are developed for includes, automating tasks for manufacturing, industrial processes, machining automated assembly and packaging. These kinds of tasks are important to not be interrupted or stopped unexpectedly. This scenario has the potential to create hazardous or dangerous situations, or the manufacturer

---

can lose significant monetary value. These are some of the reasons PLCs are built to complete task in harsh environments, and are designed to run as long as possible without failure.

For some automation tasks the added robustness might not be necessary. For example a washing machine for home use might only need a microprocessor controller without the added robustness a PLC provides. Autopilots for airplanes needs more computational power than a PLC provides, to solve complex mathematics and high speed operations. Therefore, a normal computer is more beneficial to run autopilots. In the next subsection the hardware and architecture of the PLC will be looked into. [5], [11] [8].

### 2.1.3 The Hardware and Architecture of PLCs

The functional parts of a typical PLC is shown inside the box in Figure 4.

- Where the **Processor** part is the microprocessor(s) that does the arithmetic's and the execution of the application program functions.
- The **Program and data memory** is storing the application program and the states/variables the application program is using.
- **Communications interface** is providing a function to communicate with third-party devices such as PLC's from other manufacturers and computers.
- The **Power supply** is supplying the necessary power to the different parts of the PLC.
- The I/O interfaces, **Input interface** and **Output interface**, is interfacing with the input and output devices respectively.

The I/O interfaces will be explored a bit further. Figure 5 (a) shows a similar architecture diagram as Figure 4 from the IEC 61131-1 standard, and (b) shows one more advanced version from IEC 61131-2 standard. See chapter 6 **Functional requirements** in IEC 61131-2 standard for in-depth specifications of the functional requirements of a PLC's hardware and architecture. [11], [13], [5], [7], [8]

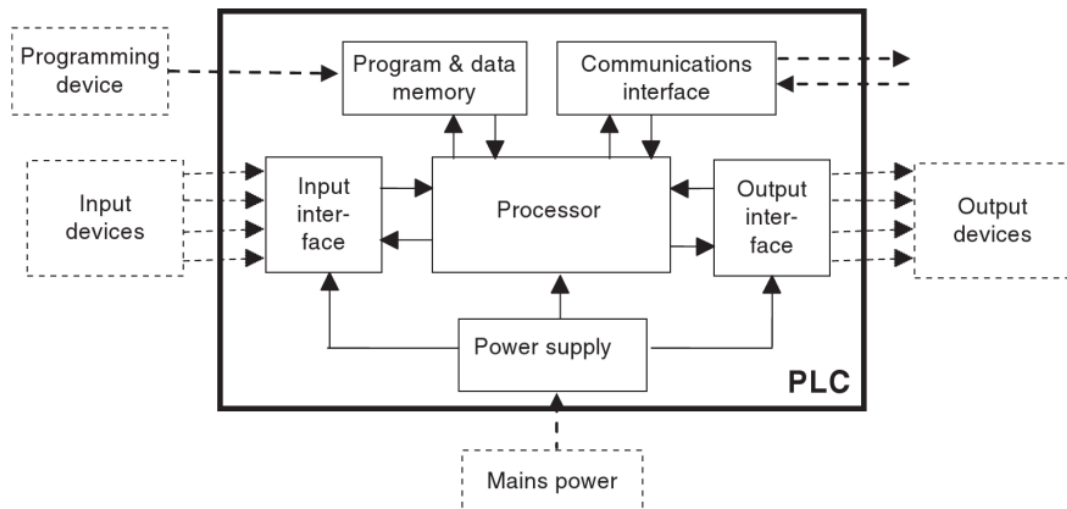


Figure 4: The general architecture of PLCs, [5]

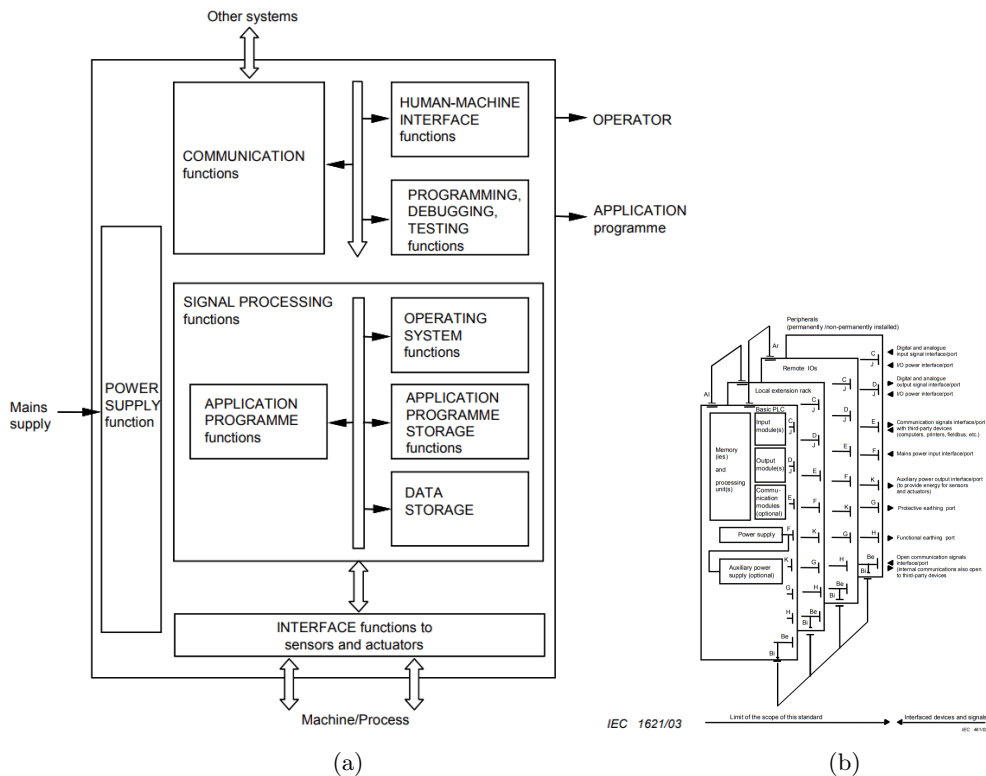


Figure 5: The architecture of PLC from IEC 61131, (a) shows a simple version from IEC 61131-1 [11]. And (b) shows an extended version from IEC 61131-2 [13]. In addition there is a larger version of it in Appendix A.

### 2.1.4 PLC Inputs and Outputs

The input and output interfaces is as said, the part of the PLC that interfaces with input and output devices. This is the part that PLCs use for communication with the external devices. From this interface, the controlled devices gets output signal, and the controlling devices gets input signals. There are different kinds of inputs and outputs associated with a PLC.

A PLC can have both discrete/digital I/O and analog I/O. Discrete inputs can either be provided by push buttons or phototransistors to name a few. Analog inputs can be provided by for instance a temperature sensor. The discrete outputs can then be LED's or other lights with one intensity, or motors that run in only one speed, such as a conveyor belt. Whilst a motor that need speed control, might use an analog output from the PLC. One example is CNC machines with speed control for specific tools.

The processor part of the PLC is using low voltages for it's operations. For such components, 3.3 volt to 5 volt is common as the source voltage. While the PLC can use many different voltages depending on the I/O module, the most common for discrete signals is 24V. The processor cannot give such voltages directly, or read them since that would break it.

That is why opto-isolators are commonly used for the inputs they read. There are three common ways to make the low voltage signals from the processor to the correct output voltage. These are relay-types, transistor-types and triacs. The relays and transistors work as switches that lets through the correct electrical signal, while the triacs only work for AC current. [8], [5].

---

In the IEC 61131-2 chapter 6 standard, there are specifications of how the I/O and other functions are required to be compliant. For a specific output type there cannot be more than a given amount leaking current in its off state as an example. These requirements are important for the PLCs to behave predictably, and to be safe, robust, and have desired longevity in the environments they are used.

### 2.1.5 PLC Manufacturers and Models

There are many different manufacturers of PLCs to choose from if such a unit is desired. They all have their pros and cons, and usually requires proprietary environments for coding, uploading the code, and communication between units. There are also open standards for communications if there is a need for mixed environments. Modicon was the first manufacturer of PLCs, although Siemens is one of the most popular manufacturers of PLCs today. See Figure 6 for an example of a PLC from Siemens. [5], [7], [8]

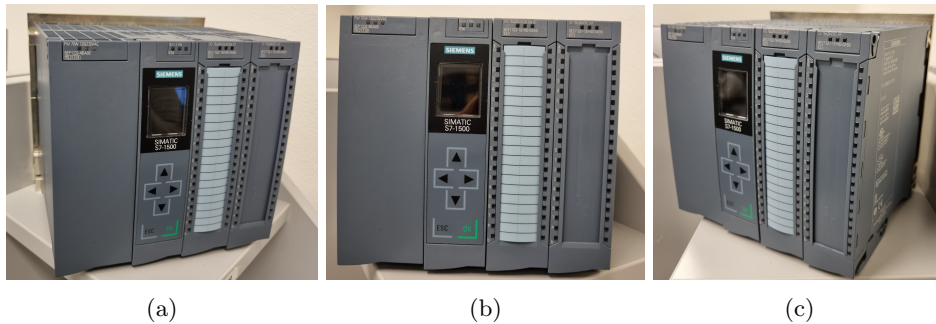


Figure 6: Three pictures we took of a Siemens Simatic S7-1500 PLC.

## 2.2 Real-time kernel for linux

An operating system with a fully preemptible kernel is usually required to run certain mechatronic systems. This type of kernel is used when there are specific constraints and to secure a certain "worst time" possible when there are strict demands of the maximum allowed delay. Linux doesn't run on a fully preemptible kernel from the box, which means the OS has to be patched to allow real-time capabilities. A detailed explanation on how to do this is given in Section 3.1. According to [14], an RT kernel has the following advantages to an OS with a regular kernel.

- Checks task-priority under load
- High priority tasks are given preference for CPU execution
- Maintains a low latency execution time
- Possible to check, measure and configure response time

A kernel preemption makes the kernel able to change between processes, even if it already has a process running. This makes the kernel able to process higher-priority tasks by interrupting already running task, and finish them later. With a fully preemptible kernel, there can be set a maximum delay for certain high-priority tasks. This is due to the delay being independent on

---

the complexity of the processes or tasks already running. This is a desired quality when running systems with strict demands of the maximum delay. Some examples are car production lines and pace makers.

## 2.3 Raspberry Pi

Raspberry Pi is a single-board computer originally made to encourage learning and data science in schools and developing countries. It is built and developed by the british Raspberry Pi foundation in cooperation with Broadcom. At the time of writing, the latest development, and the version used in this project is the Raspberry Pi 4 Model B. This is currently available for everyone to buy, and it exists with choices of 2GB, 4GB and 8GB RAM to name a few. The Raspberry Pi is using a CPU with ARM architecture, and uses an open-source version of Debian Linux as an operating system. For more information about the specifications of the Raspberry Pi see [15].

There are a lot of reasons why Raspberry Pi was chosen over other single board computers. Mainly it was due to accessibility and modularity. A Raspberry Pi is made as an educational tool with a lot of ports and signaling capabilities. In addition, there is a lot of community support online, which makes solving problems easier compared to computers with no online support. All of these reasons makes it ideal for prototyping. Raspberry Pis are normally not implemented nor used in industrial applications. One alternative to the PI for this project, could be an industrial counterpart.

## 2.4 OpenPLC

For this project, *OpenPLC* was chosen as an operating software for the mini-factory. According to the description on *GitHub*, it is an open-source Programmable Logic Controller (PLC) based on easy to use software. This is provided as a low-cost industrial solution for automation and research [16].

Another benefit of OpenPLC is that the entire source code is provided. This also makes it ideal for industrial cyber security research. However, this isn't relevant for this project. One of the most important aspects of this project is to be able to provide a framework to learn different types of PLC languages. OpenPLC follows the international IEC 61131-3 standard, which is the official standard for PLC programming languages [12]. In addition, OpenPLC is both easy to use and completely free. This makes the software optimal to use for this project.

## 2.5 OPCUA

OPC United Architecture (UA) was launched in 2008 and is an independent platform that integrates all of the features from OPC Classic into one extensive framework. This model is mainly used as a common language for communication between interfaces provided by different manufacturers. This includes robots, operating systems or different hardware platforms. The extension works as a server-client feature and enables communication between multiple Field-busses.

---

## 2.6 MultiSim

MultiSim is an online and free simulation program provided by National Instruments. With this program it's possible to build and simulate circuits.

## 2.7 Visual Components

*Visual Components* is a developer and provider of manufacturing simulation software and solutions [17]. It was started in 1999 to provide a solution to make simulation technology and manufacturing design more accessible for companies of all sizes. Some of the functions that *Visual Components* provides are machine builders and system integrators.



---

## 3 The setup of the Raspberry Pi

### 3.1 Setup of real-time kernel on Raspberry Pi

A lot of mechatronic systems have strict requirements of the amount of delay allowed by the system. To be able to work with such a system, a real time kernel is needed. Therefore, it was desired to run the mini-factory with a real time operating system as well. This real time kernel have been set up to work on Debian Linux for Raspberry Pis for this project. To do this, a guide inspired from [4] were followed.

The first step to patch the Raspberry Pi kernel, is done by cross-compiling with a host computer. To do this, the appropriate tools needs to be installed on the computer:

```
$ sudo apt-get install build-essential libgmp-dev libmpfr-dev  
↪ libmpc-dev libisl-dev libncurses5-dev bc git-core bison flex  
$ sudo apt install libelf-dev
```

```
$ sudo apt-get install libncurses-dev libssl-dev
```

After installing the necessary tools, the next step is to compile the native build for cross-compiling. By typing the following line, *Binutils* is installed:

```
$ cd Downloads  
$ wget https://ftp.gnu.org/gnu/binutils/binutils-2.35.tar.bz2  
$ tar xf binutils-2.35.tar.bz2  
$ cd binutils-2.35/  
$ ./configure --prefix=/opt/aarch64 --target=aarch64-linux-gnu  
↪ --disable-nls
```

After configuration, use the following commands to compile the program:

```
$ make -j4  
$ sudo make install
```

The path needs to be exported after compilation. To do this, type the following commands:

```
$ export PATH=$PATH:/opt/aarch64/bin/
```

After exporting the path, build and install GCC with the following commands:

```
$ cd ..  
$ wget https://ftp.gnu.org/gnu/gcc/gcc-8.4.0/gcc-8.4.0.tar.xz
```

---

```
$ tar xf gcc-8.4.0.tar.xz
$ cd gcc-8.4.0/
$ ./contrib/download_prerequisites
$ ./configure --prefix=/opt/aarch64 --target=aarch64-linux-gnu
↪ --with-newlib --without-headers --disable-nls --disable-shared
↪ --disable-threads --disable-libssp --disable-decimal-float
↪ --disable-libquadmath --disable-libvtv --disable-libgomp
↪ --disable-libatomic --enable-languages=c --disable-multilib
```

The next step is to compile *GCC*.

```
$ make -j4
$ sudo make install gcc
```

Due to the patching and installation of kernel is going to be done by cross-compilation, there is a need to be sure that the compiler is installed on the host-computer. To download this, type the following commands into the terminal:

```
$ sudo apt-get update
$ sudo apt-get install gcc-aarch64-linux-gnu
```

The tools required to build and install the patch onto the kernel, should now be downloaded and ready. The kernel version used in this project is *v5.15* with the corresponding patch *RT49*.

Continuing the process on the host-computer, make a new directory. The kernel and the corresponding real-time patch needs to be downloaded next. It is possible to patch a non-corresponding version of the kernel, but it's no guarantee that it will work properly. To do this, type the following lines in the terminal:

```
$ mkdir ~/rpi-kernel
$ cd ~/rpi-kernel
$ git clone https://github.com/raspberrypi/linux.git -b rpi-5.15.y
```

The patch used in this project is downloaded by typing the following lines in the terminal:

```
$ wget https://mirrors.edge.kernel.org/pub/linux/kernel
↪ /projects/rt/5.15/older/patch-5.15.65-rt49.patch.gz
```

If it's desired to download another patch, the newest patches are found from <https://mirrors.edge.kernel.org/pub/linux/kernel/projects/rt/5.15/>. To download this, just change the fields by replacing "XX" and "YY" with the newest version numbers `patch-5.15.XX-rtYY.patch.gz`.

---

```
$ wget https://mirrors.edge.kernel.org/pub/linux/kernel
→ /projects/rt/5.15/patch-5.15.XX-rtYY.patch.gz
```

Applying the patch to the kernel is done by typing:

```
$ mkdir kernel-out
$ cd linux
$ gzip -cd ../patch-5.15.65-rt49.patch.gz | patch -p1 --verbose
```

Before building the patch, the configuration has to be set up to allow real-time capabilities for the kernel of the Raspberry Pi. To apply the default settings, type:

```
$ make O=../kernel-out/ ARCH=arm64
→ CROSS_COMPILE=/opt/aarch64/bin/aarch64-linux-gnu-
→ bcm2711_defconfig
```

In addition to these settings there is a need to change the setting of CONFIG\_KVM to unlock real-time capabilities. An explanation of why this is the case is discussed in Section 7.5. This is done by the following commands:

```
$ cd ..
$ cd kernel-out
$ echo -e "CONFIG_EXPERT=y\nCONFIG_KVM=n" >> .config
$ cd ..
$ cd linux
```

After these lines are written in the terminal, it should now be possible to choose the real-time kernel in the `menuconfig`. To open the `menuconfig`, type the following command:

```
$ make O=../kernel-out/ ARCH=arm64
→ CROSS_COMPILE=/opt/aarch64/bin/aarch64-linux-gnu- menuconfig
```

Enable "FULLY-PREEMPTIBLE KERNEL (REAL-TIME)" in this menu. This is done by doing the following steps:

1. General setup
2. Preemption model
3. Fully preemptible (Real time)

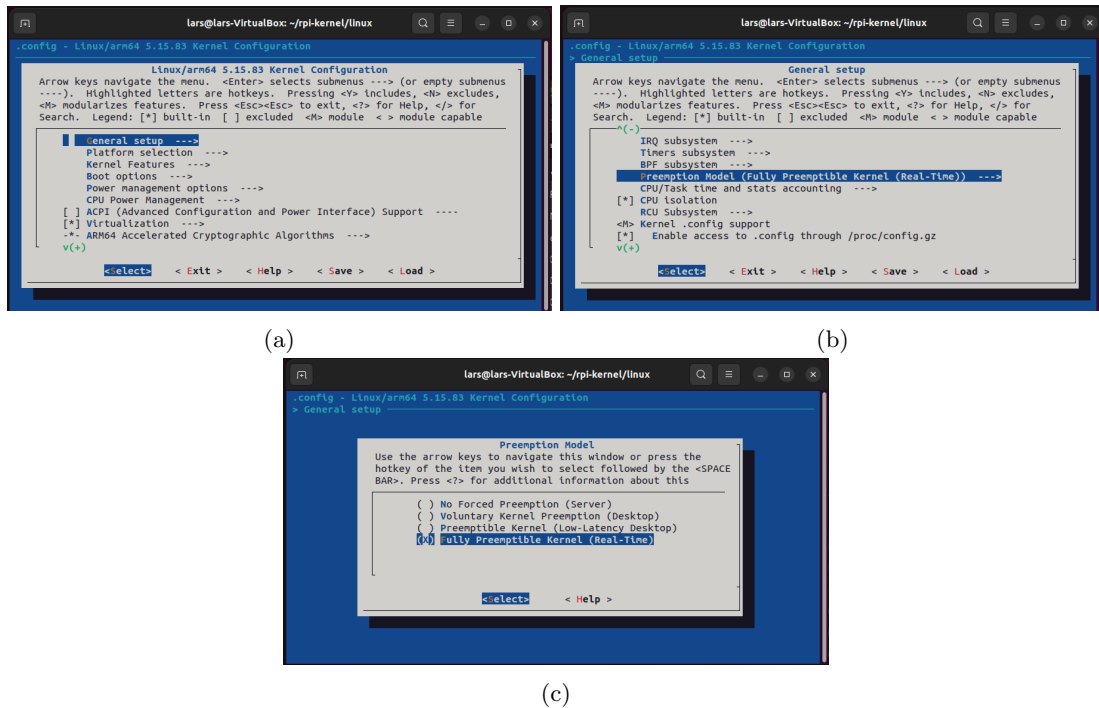


Figure 7: Images of the menuconfig (a) shows General setup, and (b) shows the Preemption-model and (c) fully pre-emptible kernel

Figure 7 shows what the menuconfig looks like on a PC.

After this step, the new rt-kernel needs to be built and compiled, this is done with the following line:

```
$ make -j4 O=../kernel-out/ ARCH=arm64
→ CROSS_COMPILE=aarch64-linux-gnu-
```

When the compilation is finished, the next step is to zip the kernel. This is done by the following lines in the terminal:

```
$ export INSTALL_MOD_PATH=~ /rpi-kernel/rt-kernel
$ export INSTALL_DTBS_PATH=~ /rpi-kernel/rt-kernel
$ make O=../kernel-out/ ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
→ modules_install dtbs_install
$ mkdir ../rt-kernel/boot
$ cp ../kernel-out/arch/arm64/boot/Image ../rt-kernel/boot/kernel8.img
$ cd $INSTALL_MOD_PATH
$ tar czf ../rt-kernel.tgz *
$ cd ..
```

The kernel should now be zipped inside "rt-kernel.tgz", which now needs to be sent to the Raspberry Pi. This can be done either by using a USB-Stick or through SCP. Write the following line in the terminal to send the file via SCP:

---

```
$ scp rt-kernel.tgz pi@<ipaddress>:/tmp
```

The last steps is done on the Raspberry Pi itself. To install the newly built kernel, the following commands needs to be executed:

```
$ cd /tmp
$ tar xzf rt-kernel.tgz
$ cd boot
$ sudo cp -rd * /boot/
$ cd ../lib
$ sudo cp -dr * /lib/
$ cd ../overlays
$ sudo cp -dr * /boot/overlays
$ cd ../broadcom
$ sudo cp -dr bcm* /boot/
```

After the commands above are executed, the file `"/boot/config.txt"` needs to be edited by appending the line `"kernel=kernel8.img"` at the end.

In order to apply the changes, and check if it installs successfully, the Raspberry Pi needs to be rebooted. After rebooting is complete, the following command can be executed:

```
$ uname -a
```

If the installation is successful, the output should be along the lines of `"Linux raspberrypi 5.15.65-rt49-v8+ 1 SMP PREEMPT RT Fri Dec 1 01:17:07 CET 2022 aarch 64GNU/Linux"`.

### 3.2 Installing OpenPLC runtime on the Raspberry Pi

The first step to install the OpenPLC runtime on the Raspberry Pi, is to connect it to internet. On a normal private network this can be done in the normal way, but on Eduroam it is a bit more inconvenient process. It is possible to circumvent this by sharing WiFi with a cellphone, but it is recommended to connect directly to Eduroam. A guide on how to connect to Eduroam for the Raspberry Pi is provided in [18], and can be seen in Appendix B.

The Runtime for OpenPLC can be installed on different kinds of devices, and on OpenPLC webpage there are guides on how to install each of them. Following the Linux version of the guide, the Raspberry Pi specific options is the recommended way of installing it. This guide can be found from [19]. In this thesis, it was installed exactly like this.

The easiest way to install the OpenPLC Runtime on a Raspberry Pi is to use *git*. This is usually done by getting it directly from the official site. To ensure git is installed, write this in the terminal:

---

```
$ sudo apt-get install git
```

To install the runtime, write the following lines in the terminal after git is installed:

```
$ git clone https://github.com/thiagoralves/OpenPLC_v3.git
$ cd OpenPLC_v3
$ ./install.sh rpi
```

When it is done, the RPi needs to be rebooted, which can be done by typing this in the terminal:

```
$ reboot
```

The runtime is now installed, but it will not work quite yet. The RPi version is depending on the WiringPi library. It can be downloaded from Github, where the latest version can be found on:

```
https://github.com/WiringPi/WiringPi/releases/
```

The `-armhf.deb` file should be used on 32-bit OS (Raspberry Pi 3 and under) and the `-arm64.deb` is meant for 64-bit OS (Raspberry Pi 4 and up). Download the appropriate file for your architecture on your Raspberry Pi. In this thesis, a Raspberry Pi 4B is used with a 64-bit OS, and therefore the `-arm64.deb` is the one downloaded. Then install it with one of the two `dpkg` command:

```
$ dpkg -i wiringpi-[version]-armhf.deb

or

$ dpkg -i wiringpi-[version]-arm64.deb
```

The newest version during the installation, and what used in this thesis are 2.61-1, and the file ended up in the Downloads folder. It can also be necessary to use the `sudo` command. The exact commands used in the thesis are:

```
$ cd Downloads
$ sudo dpkg -i wiringpi-2.61-1-arm64.deb
```

Test that the WiringPi installation finished successfully with the command:

```
$ gpio -v
```

---

## 4 Technical specifications

### 4.1 The interface circuitry

The main purpose of this project is to use a microcontroller to emulate a Programmable Logic Controller (PLC). PLCs are usually used in the industry to run production lines or other industrial sequential processes. PLCs are commonly used due to high reliability, ease of programming and process fault diagnostics. In educational purposes, it might be seen as unnecessary to get a PLC, mostly due to cost. Microcontrollers are also usually<sup>1</sup> easier to attain, and doesn't require manufacturer specific software. A microcontroller is a lot more flexible than a PLC, and have the ability to be used for much more diverse tasks.

As described in Section 2.3, a Raspberry Pi was chosen for this project. To be able to emulate a PLC, a fitting software needs to be used. The software used for this project was OpenPLC, which is an open-source PLC software. This software is made as an affordable and accessible method to learn PLC programming using the global IEC-61131-3 standard. The program is written in the programming language "C", which makes it highly portable over multiple devices. The examples of compatible operating systems that are given on their official site are Linux, Arduino, Windows and Raspbian [16].

In Figure 8, the general architecture map for a PLC is shown. The Raspberry Pi is going to be responsible for the modules shown with a green square. The modules marked by a red square is going to be emulated by an interfacing circuit. These will be developed as a part of the project. The devices connected to the PLC are marked with a blue square. The power for the system will be partially provided from the Raspberry Pi. In addition, an external power supply will be connected.

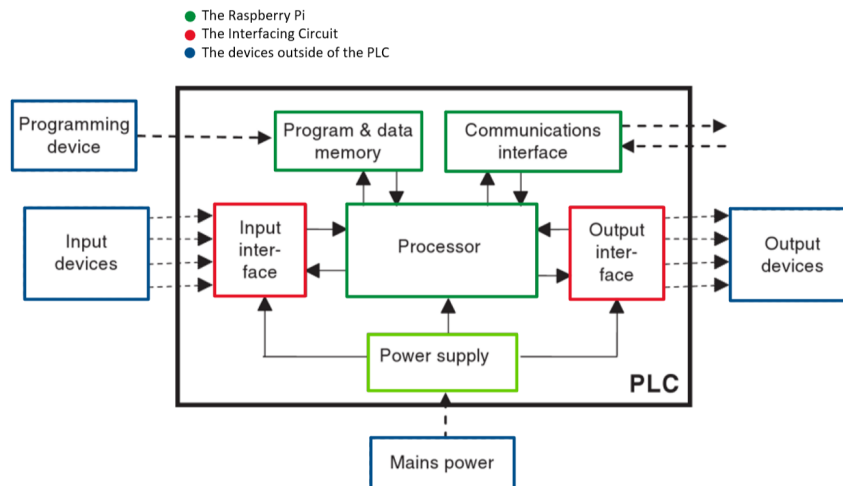


Figure 8: The planned way to emulate the general architecture of PLCs. Based on Figure 4 from [5].

The main purpose of the interface circuitry, is to ensure that both the microcontroller and mini-factory gets the required voltage to run the modules. The voltage provided from the IO ports

<sup>1</sup>This project thesis was written under a semi-conductor crisis, microcontrollers like Raspberry Pi are not easy to get at the time of writing

---

on microcontrollers made for private use, are usually specified to be between 2.7 to 5 volts. The actuators, sensors and motors on the mini-factory requires 24 volts to run. Therefore it is desired to make a circuit that will safely translate low voltage signals to higher voltage signals. Meanwhile, it is important to translate back from 24 volts to 3.3 volts, to safely read the input signals given from the sensors of the factory. If the input voltage to the Raspberry Pi is too high, the power might ruin IO pins, or in the worst case, ruin the whole controller. A brief sketch of how this might be done, is given in figure Figure 9.

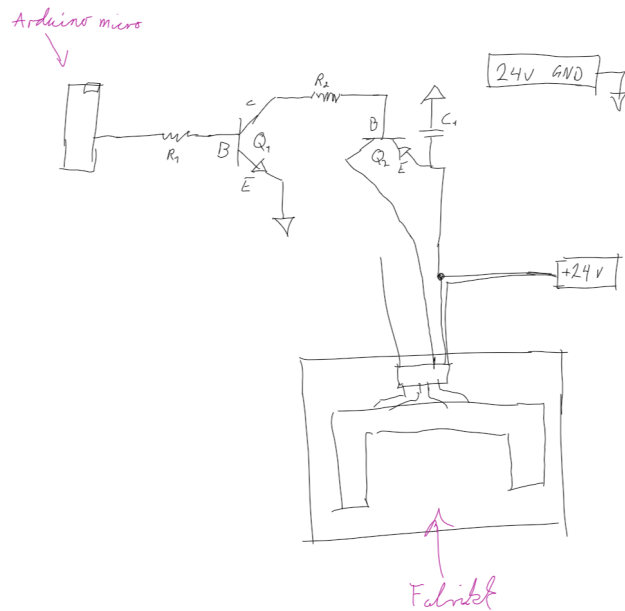


Figure 9: Rapid sketch of how to turn on a 24V signal from 5 volts and lower voltages

## 4.2 Specifications of the mini-factory

In the datasheet, Table 1 was provided as an overview of the pins and functions of the factory.



---

Table 1: Overview of the pins given by the data sheet of the factory

Terminal/Pin	Function	Input/Output
1	Power supply actuators (+)	24V DC
2	Power supply sensors (+)	24V DC
3	Power supply (-)	0V (GND)
4	Power supply (-)	0V (GND)
5	Push-button slider 1 front	I1
6	Push-button slider 1 rear	I2
7	Push-button slider 2 front	I3
8	Push-button slider 1 rear	I4
9	Phototransistor slider1	I5
10	Phototransistor milling machine	I6
11	Phototransistor loading station	I7
12	Phototransistor drilling machine	I8
13	Phototransistor conveyor belt swap	I9
14		
15	Motor slider 1 backward	Q1
16	Motor slider 1 forward	Q2
17	Motor slider 2 backward	Q3
18	Motor slider 2 forward	Q4
19	Motor conveyor belt feed	Q5
20	Motor conveyor belt milling machine	Q6
21	Motor milling machine	Q7
22	Motor conveyor belt drilling machine	Q8
23	Motor drilling machine	Q9
24	Motor conveyor belt swap	Q10

To check that everything on the mini-factory worked correctly, a powersupply with adjustable voltage and electric current was connected into the power slots on a breadboard. The voltage was adjusted to 24V and the current was set to 0.1 Amps. Two wires were then connected to pin 1 and 3. These were added to power up the actuators of the mini factory. When the actuators had power, the motors could be tested. A new wire connected the powered slots on the breadboard, to the input terminal 15. After observing that the motor was running, terminal 16-24 were tested accordingly.

A similar method were used to test the remaining pins. The wires providing power were connected to pin 2 and 4, to power up the sensors of the mini-factory. When the factory had power, a multimeter were connected to pin 5 and ground. The button connected to pin 5 was pressed, and a change in voltage could be observed. This indicated that the button was functional. The same method were used for the remaining buttons, found on terminals 5-8. The phototransistors were tested with the same setup as the buttons, although these sensors were activated by blocking the light signal between the LED light source and phototransistor. A similar change in voltage was observed, but opposite. Unlike the buttons, the phototransistors are normally open, and closes when an object is blocking the signal between them.

When the signal circuit in Section 5.3 were tested, it was discovered that the motors where pulling a current from their respective signal pins. The current they pulled was low, under 0.1 Ampere. This is still a significant current when the assumption was that they only required pure voltage signals. An investigation of how much current they should theoretically pull at maximum power started. On Fischertechnik's website of the mini-factory [2], it says that all the motors are XS DC motors. This doesn't give a lot of information that could be used, so further investigation were

conducted. In the last question in their technical FAQ [20], it was found that the motors pull a maximum of 0.265A at 9V. This give a maximum current draw at 24V of 0.0994A.

		d.c. output style																							
		Type 0,1	Type 0,25	Type 0,5	Type 1	Type 2																			
<b>Rated current for state 1</b>	$I_e$ (A)	0,1	0,25	0,5	1	2	<b>Normative items</b>																		
Current range for state 1 at maximum voltage (continuous)	Max. (A)	0,12	0,3	0,6	1,2	2,4																			
Voltage drop, $U_d$	Non-protected output	Max. (V)	3	3	3	3																			
	Protected and short-circuit-proof	Max. (V)	3	3	3	3	a																		
Leakage current for state 0	Max. (mA)	0,1	0,5	0,5	1	1	b, c																		
Temporary overload	Max. (A)	See Figure 15 or as specified by manufacturer																							
<p><sup>a</sup> For 1 A and 2 A rated currents, if reverse polarity protection is provided, a 5 V drop is allowed. This makes the output incompatible with a type 1 input of the same voltage rating.</p> <p><sup>b</sup> The resulting compatibility between d.c. outputs and d.c. inputs, without additional external load, is as follows:</p> <table border="1" style="margin-left: 40px;"> <thead> <tr> <th></th> <th>Output Type 0,1</th> <th>Output Type 0,25</th> <th>Output Type 0,5</th> <th>Output Type 1</th> <th>Output Type 2</th> </tr> </thead> <tbody> <tr> <td>Input Type 1:</td> <td>yes</td> <td>yes</td> <td>yes</td> <td>no</td> <td>no</td> </tr> <tr> <td>Input Type 3:</td> <td>yes</td> <td>yes</td> <td>yes</td> <td>yes</td> <td>yes</td> </tr> </tbody> </table> <p><sup>c</sup> With adequate external load, all d.c. outputs may become compatible with all Type 1 and Type 3 d.c. inputs.</p>									Output Type 0,1	Output Type 0,25	Output Type 0,5	Output Type 1	Output Type 2	Input Type 1:	yes	yes	yes	no	no	Input Type 3:	yes	yes	yes	yes	yes
	Output Type 0,1	Output Type 0,25	Output Type 0,5	Output Type 1	Output Type 2																				
Input Type 1:	yes	yes	yes	no	no																				
Input Type 3:	yes	yes	yes	yes	yes																				

Figure 10: The Digital outputs for direct current table from section 6.4.6.1 in [13].

### 4.3 PLC standards

One of the surprises found when testing, was that the mini-factory drew more current from the signal pins than first assumed. This assumption was mainly based on the existence of pin 1 in the spec sheet provided in Table 1, which was thought to provide enough current to run the whole factory.

To figure out if this could be a problem or not, an investigation of the output signal current of PLC's were started. On Siemens's site [21] there was found that the output modules for SIMATIC S7-1500 are capable of a minimum of 0.3A as their maximum current. There were other modules that could give more, but it was the module that gave the lowest maximum that was interesting. Since the modules can provide enough current to drive the motors, it seems to not be a problem. Therefore, it was reasonable to adjust the resistors of the PCB circuit to provide a similar amount of current through the output signals as well.

Since one of the goals of this project was to make a Raspberry Pi emulate a PLC, it were seen as relevant to investigate the international standards of the workings of a PLC. This was done to see exactly what functions needed to be replicated. In IEC-61131 part 2 [13], the hardware standards is set. In this standard, the minimum required current for an output signal from a PLC was found. The PLC standards for the power-signals are found in Figure 10. As can be seen from this table, the smallest output type is 0.1A. Meaning that a PLC's discrete output should always be able to give that amount of current. In that case the mini-factory follows the PLC standard of the power

---

consumption, which explains why the motors needed additional current from the signal pins to run.

#### **4.4 Testing with other microcontrollers**

The circuits created to run the mini-factory with the Raspberry Pi in Section 5, were also tested with Arduinos. During the development of the Ladder Diagram code of the first corner presented in Section 6.3.1, it was tested with an Arduino UNO in addition to the Raspberry Pi. This was done to test that it would work with other controllers. It worked fine, but the UNO does not have enough pins to run the whole factory. When the whole Ladder Diagram code was developed, it was tested with an Arduino Mega. Since the Mega has a lot more pins, it had enough to run the whole factory and is a good alternative to the Raspberry Pi.

---

## 5 Circuitry

To run the factory at all, the circuit must make the factory work at an input signal with a voltage of 3.3 volt. This is the voltage of the signal the Raspberry Pi chosen for this project can provide. However, it is desired to be able to run the factory on microcontrollers from different manufacturers. In addition, it would be beneficial to also have the ability to run the factory from a PC, which provides 10 volts. To be able to run on different types of controllers, the span of the output signal must be between 2.7V - 10V. To future proof the concept, it's also desired to make the signals run on even lower voltages. The reasons for this, is in case of a fall in voltage, or similar. Other microcontrollers in the future could also have a lower voltage on output signals than normal today. Allowing higher voltages could also be beneficial for the project. Therefore, it was decided that the interface circuit should be able to work for all voltages between 1V - 12V.

Table 2: Different aspects of the design graded based on importance. The numbers range from 5-1 where 5 is considered most important.

Description	Must have	Should have	Grade
Flexible voltage	3.3V	1.5V - 12V	5
User friendly	Marked pins <sup>2</sup>	Plug and play	4
Simulation opportunity	No	Yes	2
Control from micro controllers	Raspberry Pi	All	3

??Kanskje legge inn et avsnitt angående kommentar over

### 5.1 Push-pull circuit version 1

One obstacle in this project was to get the correct voltage on the power supplied for the electric motors. The motors on the mini-factory is rated for 24V. It is desirable to run the voltage to the motor as close to the specification as possible. Running a voltage to low can lead to burn out, and permanently damage the motor.

The Raspberry Pi used for this project is only capable of delivering a maximum of 5v direct current. This is not high enough for the desired application. Therefore there is a need to design a circuit that can run the 24V DC motor from an output signal provided by the Raspberry Pi. The circuit that was designed for the application is provided in Figure 11.

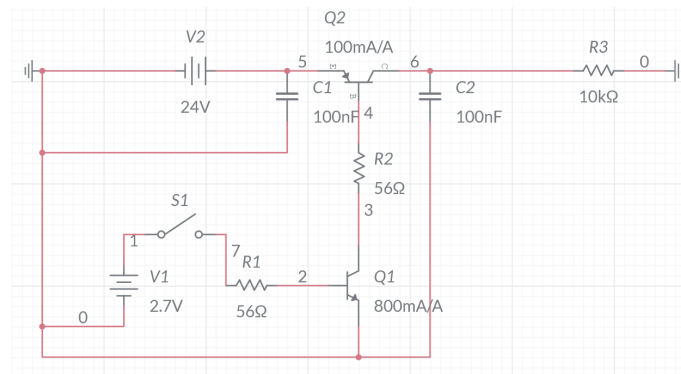


Figure 11: The circuit required to run the motors on 24v from a signal from 2.7 - 5.5V

---

<sup>2</sup>Meaning there will be documentation of what the pins does.

---

This circuit is using Bipolar Junction Transistors (BJT) as switches. The circuit works by sending a voltage into the BJT, making it activate and pass electrons. BJTs are however vulnerable to high currents, which means there's a need for resistors to limit the current. The circuit provided in Figure 11 have two transistors, one PNP and one NPN. More accurately it's the BC337 PNP and BC557 NPN transistor. The maximum current rated for the BC557 is 100mAmps. When running a signal on a voltage of 5.5V, the signal had a current of 96 mAmps. This means 5.5V is the maximum voltage which is still under the allowed current of the BJT.

One of the biggest hazards in a circuit like this, is the voltage of the input signal. If the voltage of the input signal is too low, the BJT will open partially, or not open at all. This will happen when the output voltage at the end of the circuit is lower than the required 24V. After testing the circuit, both on *MultiSim* and breadboards, it would seem like 2.5 volt is the lowest signal that gives the desired output voltage in this circuit. At least with the resistors and transistors that were chosen.

## 5.2 Push-Pull version 1 evaluation

This circuit ran the factory on a Raspberry Pi well, but there were some minor inconveniences that made the circuit sub-optimal for the application. At the time, it seemed like the mini-factory ran purely on signals. Based on the assumption that the factory could be run purely on signals from the Raspberry PI, the circuit ran a current that was outside the transistors specifications. The BJTs used in this circuit was rated for a maximum of 100mA. According to *MultiSim*, the current running through the circuitry was 90mA at 5.5V. To fix these problems, a different solution or a big update of the circuit was necessary.

## 5.3 IO signal circuit

The next step in the process, was making a circuit to run the mini-factory purely on signals. According to the spec sheet in Table 1, there is a separate power supply channel at pin 1 for the actuators. Therefore, it seemed like motor controllers was already a part of the circuitry of the factory. Based on this, it was made an assumption that there was no need to run current through the circuit to provide power to the electric motors.

A circuit was designed in *MultiSim* to take care of the signalling to the factory. This circuit used the 2N3904 transistor. This is a NPN silicon BJT that opens for voltages between 0V - 1V. A picture of this circuit is provided in Figure 12.

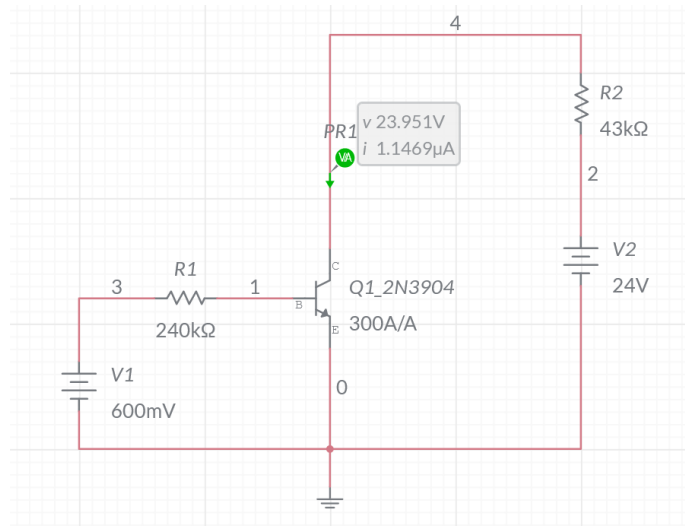


Figure 12: Circuit made to have IO activation for all voltages between 0V-1V

The resistors of the circuit were selected to decrease the current in the circuit as much as possible. After trying the circuit in *MultiSim*, it was found that a relation of 5:1 was reasonable for the desired output current. This relation decides the voltage needed to open or close the BJT. Resistances of 240k  $\Omega$  and 43k  $\Omega$  (Ohm) were chosen for convenience and accessibility.

#### 5.4 IO signal circuit evaluation

Theoretically, this circuit was a good option for the project, due to the assumption that the factory could be run purely from signals. The circuit ran a voltage of 24V steadily and with a running current of around 1 $\mu$ A with the transistors fully opened.

This proved however, to be insufficient current to run the motors on the slider properly. After testing the factory further, as explained in Section 4.2. It became evident that the factory can't run on output signals alone. The signal couldn't provide the current required to start the motors, even with smaller resistors. Therefore, another solution was required to run the factory.

#### 5.5 Push-pull version 2

The third iteration of the circuitry was a heavily upgraded version of the first push-pull circuit in Section 5.1. After finding that the motors on the factory can't be run with a signal alone, it became evident that there was a need to run current with the output signals. Since the first circuit ran the motors properly, it was natural to use this as a baseline for the new circuit.

The transistors chosen in Section 5.1 were adequate for the application, so these were unchanged. The resistors were however heavily modified. More effort were done to calculate some fitting resistors for the project. It were found in Section 4.2 that the motors were drawing some current from the circuit, but not much. Therefore a threshold of 5-10mA were deemed acceptable for the project. In addition, a 5k  $\Omega$  resistor were added in *MultiSim*, to imitate the internal resistance of the factory.

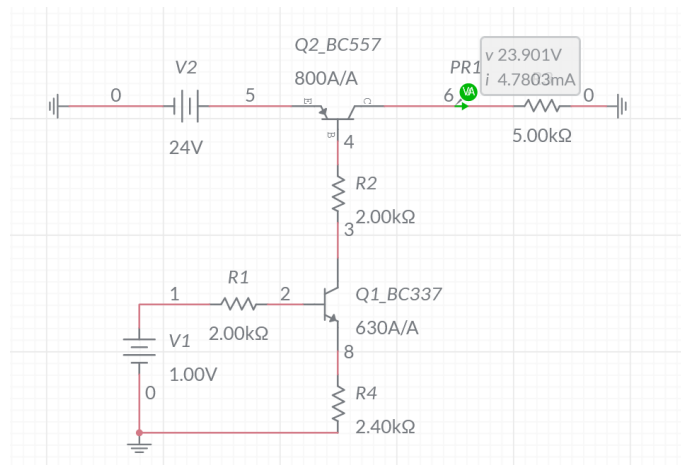


Figure 13: The second iteration of the push-pull circuit.

## 5.6 Push-Pull version 2 evaluation

The push-pull version 2 circuit worked properly for the application. The downside however, is that the circuit is hard to read from the circuit diagram, and is really complex for the application. This type of circuit is also typically used for amplifiers in audio, and not for transistors as switches, which is intended for the mini-factory. The regulation of current and activation voltage depends on the relationship between the resistors in the circuit. This makes current and activation voltage inconvenient to adjust.

## 5.7 MOSFET circuit

Due to safety and less complexity, it was decided that using an optocoupler with a MOSFET might be a better solution. MOSFET is preferred for this type of application, mostly due to being opened by voltage and not current. There is less current loss over a MOSFET than a BJT, which ultimately makes the MOSFET more suited for switches.

An optocoupler was applied to the circuit to isolate the Raspberry Pi from the 24V part of the circuit. This would act as a surge protector for the Raspberry Pi, in addition to protecting the controller from a short-circuit. To show how this was implemented, a circuit diagram is given in Figure 14. The specific MOSFET chosen for this circuit is the TN0604N3-G MOSFET transistor, and the optocoupler chosen is the 140817144300 Optocoupler Phototransistor.

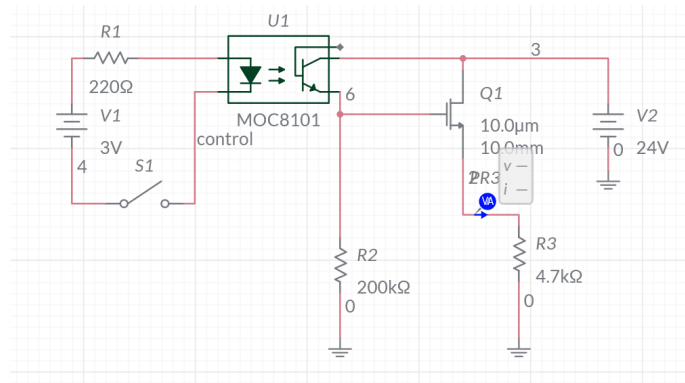


Figure 14: Circuit using a MOSFET with an octocoupler

## 5.8 MOSFET circuit evaluation

Table 3 is a table where the drawn current between the BJT- and MOSFET-circuits has been compared. This was tested using a variable powersupply, with a current and voltage display. The drawn current was observed when supplying voltage to the different actuators. In addition, another variable power supply was set up to provide current to the actuators of the factory. This was via pin 1 as stated in Table 1. As can be seen by Table 3, the loss in current is lower in the MOSFET than the BJT, as expected. Another advantage of the MOSFET circuit compared to the BJT, is less complexity. With the fewer resistors of the MOSFET circuit, it is also simpler to adjust the current than its BJT counterpart.

Table 3: Comparison between the MOSFET circuit and the Push-Pull configuration

	MOSFET-circuit	BJT-circuit
Conveyor	30-34 mA	37-40 mA
Mill	37-42 mA	49-55 mA
Drill	49-55 mA	57-60 mA
Slider	39-42 mA	47-50 mA

## 5.9 Sensor circuits

The sensors of the factory is powered the same way as the actuators. A power supply pin is added to supply 24V power to the sensors of the factory. In addition, there is a separate pin for each of the sensors. These can be used to read the input values from the sensors, and later interpreted by the Raspberry Pi. Buttons are also powered by this pin. In total, there are 4 buttons and 5 phototransistors.

Since the sensors are powered by 24V, the input signals from these will have the same voltage. A simple voltage divider was made for each sensor to ensure that the voltage going into the Raspberry Pi was no higher than 3.3 volts. The resistors used, have a resistance of 39k and 240k  $\Omega$  respectively. Figure 15 shows nine voltage dividers connected to a breadboard.



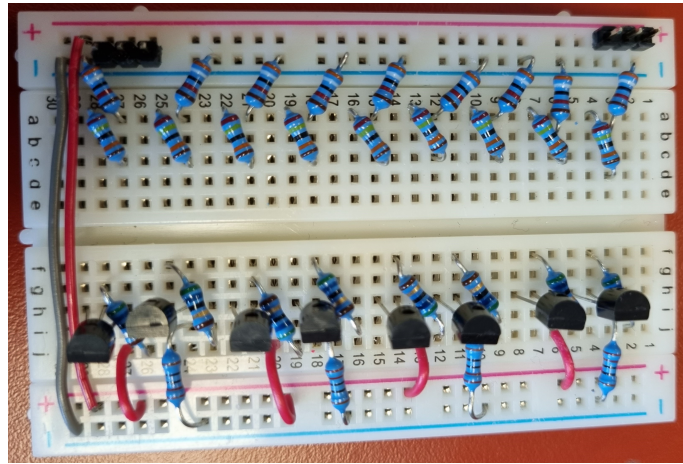


Figure 15: One of the breadboard parts neatly connected up before connected to the factory and the Raspberry Pi

### 5.10 Sensor circuit with optocoupler

For the same reasons as in Section 5.7, optocouplers were used in this circuit as well. The voltage divider is used the same way as in Section 5.9. It lowers the voltage to both the anode pin on the optocoupler, as well as the cathode connected to the 24V sides. These ground the LED within, and can be opened and closed the same way as the voltage divider. In the same fashion as the input pins on the Raspberry Pi.

The Raspberry Pi 3.3V source can further be connected to the collector of the optocoupler, and the sensor pin to the emitter. In this way the Raspberry Pi will only read its own safe signals controlled by the optocoupler.

### 5.11 Sensor circuits evaluation

The simple sensor circuit with a voltage divider is a simple, yet effective solution that worked well during the testing. However, there is a possibility that the Raspberry Pi could be connected incorrectly, and get power with a high voltage into the input pin. This will inevitably ruin the pin, therefore a protection for the pin is necessary. This is solved by adding an optocoupler, as described in Section 5.10.

An additional benefit of using an optocoupler is that the sensor circuit is more flexible, and can be connected to any microcontroller no matter the voltage being used. This is due to the inputs that reads the signals on the microcontroller is independent of the voltage used.

Buckconverters were also explored as an alternative to optocouplers, but the ones that were available were all adjustable, which were not desired for the project. They are also expensive and unnecessarily complex compared to optocouplers.

### 5.12 Breadboard circuit

Most of the prototyping was done using breadboards. These are easily accessible and easy to modify. They were also used for testing that the circuits made in MultiSim. When the breadboard

---

setup was able to run the factory properly, it was decided not to change the main layout seen in Figure 16. It was rather used to test out PLC code. However, later circuits were tested using a second breadboard.

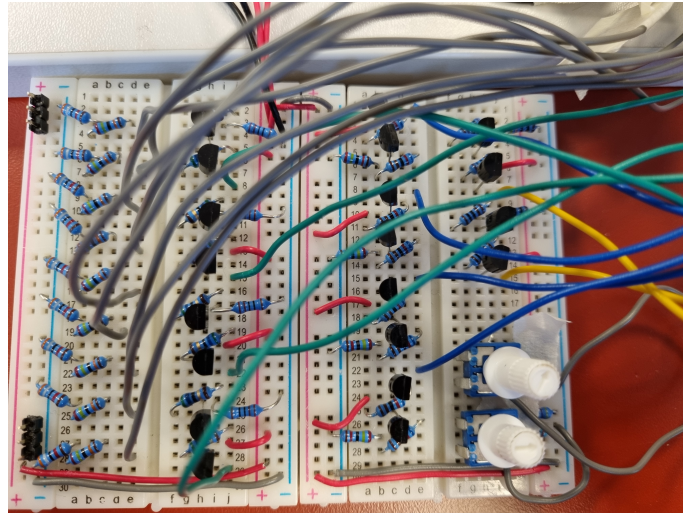


Figure 16: The final Breadboard layout connected to the factory, with the cables connecting it to the RPi disconnected.

One of the drawbacks of using breadboards however, is the reliability. Wires were falling out, and the connections were bad. Other problems were that the exposed parts of the resistors and transistors occasionally touched, and caused a short circuit. Due to having 24 pins on the factory, a lot of wires were needed to connect the corresponding pins of the microcontroller and mini- factory. As can be seen in Figure 16, all of the wires meant it was easy to lose track of what the wires were connected to. Even though the wires were labelled properly, connecting wires to the wrong pin happened frequently.

Especially if the mini-factory is going to be used in a setting as an exercise, it is extremely inconvenient to use breadboards as the main circuit. It's not easy to connect properly. In addition, unstable connection can lead to a lot of troubleshooting that students doing the exercise would waste a lot of time doing.

### 5.13 PCB circuit

One of the main goals of this project assignment was to make a PCB to interlink the Raspberry PI and the mini-factory. There are several benefits a PCB will provide over a conventional circuit or a breadboard. Some of them are:

- Reliability
- Easy to make more duplicates without complex or expensive equipment
- Easy to connect/disconnect to RPI and/or mini-factory

The objective of developing a PCB, is for the PCB to be used by students in the course Industrial Mechatronics over several years without problems. This desire makes it really important that the

---

PCB works as a reliable connection between the Raspberry Pi and the mini-factory. The goal of the task associated with the mini-factory, is for the students to learn PLC programming. Therefore, wasting the students time with an unreliable interlinking layer is not desired. With a breadboard, common problems are wires falling out of slots, bad connections and wire breaks. These are all problems that are non-existent with the use of a PCB.

Another desired feature of the PCB, is the ease of connecting the Raspberry Pi and mini-factory. When working with breadboards, a lot of problems occurred when assembling and disassembling the connecting layer. This was usually done to use the Raspberry Pi or mini-factory separately. Inconveniences of using breadboards has already been described in Section 5.12.

Therefore, a 40-pin- and a 26-pin-connector was added to the PCB. This way the Raspberry Pi and mini-factory could be connected using flat cables. These can only be slid on, and makes connecting the assembly a lot easier. Advantages of this setup, is making the mini-factory easier to use, as well as increasing the overall reliability of the assembly.

The mini-factory didn't require all of the available gpio pins on the circuit board. Therefore it was included a way to use the remaining, available pins from the Raspberry Pi. The last gpio pins were re-routed into a separate connection where they later could be used, if desired.

It was decided that the PCB should be quite large. There were multiple reasons for this, but the main reason was that the PCB should be able to be constructed with relatively simple tools in a simple lab, to ease the manufacturing of these in the future. The parts needed to make the PCB work was ordered separately, including transistors, resistors etc. Therefore it was seen as beneficial to make the PCB somewhat large, to make it easier to solder the parts onto it.

To future proof the PCB design, the decision to add two extra input variables controlled from separate code on the Raspberry Pi was made. This solution makes controlling the machining stations from external devices easier. This adds the possibility to make code which will choose if a part should be machined or not.

In the thesis, a simple Python program was written to control the mill and drill. This is possible by using the *RPi.GPIO* library. An example of how to implement such a code is given below:

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
pin = 3      #The pin number one wish to use
GPIO.setup(pin, GPIO.OUT)
```

The Pin value can then be set to "HIGH" with the code line:

```
GPIO.output(pin, GPIO.HIGH)
```

And back Low again with:

```
GPIO.output(pin, GPIO.LOW)
```

The python script that were used to control the machining stations is added in Appendix C.

---

## 5.14 PCB version 1

The design of the PCB is optimized for use as an interlinking layer between a Raspberry Pi 4, and the specific mini-factory used in this thesis. However, it is still possible to use other micro controllers as a master, and connect it to other PLC controlled systems using 24V. In that case, it will not be as easy as just plugging in the 40 pin and 26 pin connectors. The connectors on other controllers and other PLC controlled systems might not have the same layout or number of pins as the RPi and mini-factory. In that case adapter cables, or single wires would be needed to connect the correct pins.

Even if this version of the PCB is called version 1, there were another PCB in the design process before this one. It was designed with the same circuitry as the breadboards. PCB version 1 was developed through a design process. This version is the first complete design that was made. The most crucial change, was adding the footprint used for the pins connecting the PCB to other devices, such as the Raspberry Pi.

For the output signals on this PCB, the MOSFET circuit from Section 5.7 was chosen. The reasons for this choice are based on the discussions in Section 5.7 and Section 5.8. In the schematics given in Figure 17, there is one MOSFET circuit per output. This makes for 10 circuits in total. The MOSFET circuit can be seen in the middle left column of the schematic.

The input signals sensor circuit with optocouplers described in section Section 5.10, were chosen based on the reasons described in section Section 5.11. In the schematics given in Figure 17, there is one sensor circuit for per input, 9 in total. They can be seen in the middle right column of the schematic.

The connector to the left is the one that will be connected to the Raspberry Pi, and the one to the right in the picture will be connected to the factory. The extra control pins are connected to a dip switch package, in contrary to connecting them from the RPi directly back into **I\_Mill** and **I\_Drill**. The control pins are indicated by **Milling?!!** and **Drilling?!!** in the figure.

This reroute is made, to be able to control the machining stations independently with switches instead of a code written on the Raspberry Pi. This is meant for testing, and not for general use. This is because a surface mount switch packet is not a good fit for letting students switch it back and fourth, as the soldering could be weakened and the packet can lose contact or even fall off. In the schematic, the switch is in the left column, between the RPi connector and the connector for the leftover pins. Lastly, in the bottom right corner the barrel jack is located. This is the primary power source providing 24V power for both the PCB and the mini-factory.

Instead of drawing lines between the pins that are electrically connected on the schematic, labels are used. This is to make the schematic more readable, as over forty lines crossing each other is not easy to follow. Places that are electrically connected have the same labels on the schematic, and on the physical PCB there are traces connecting them.

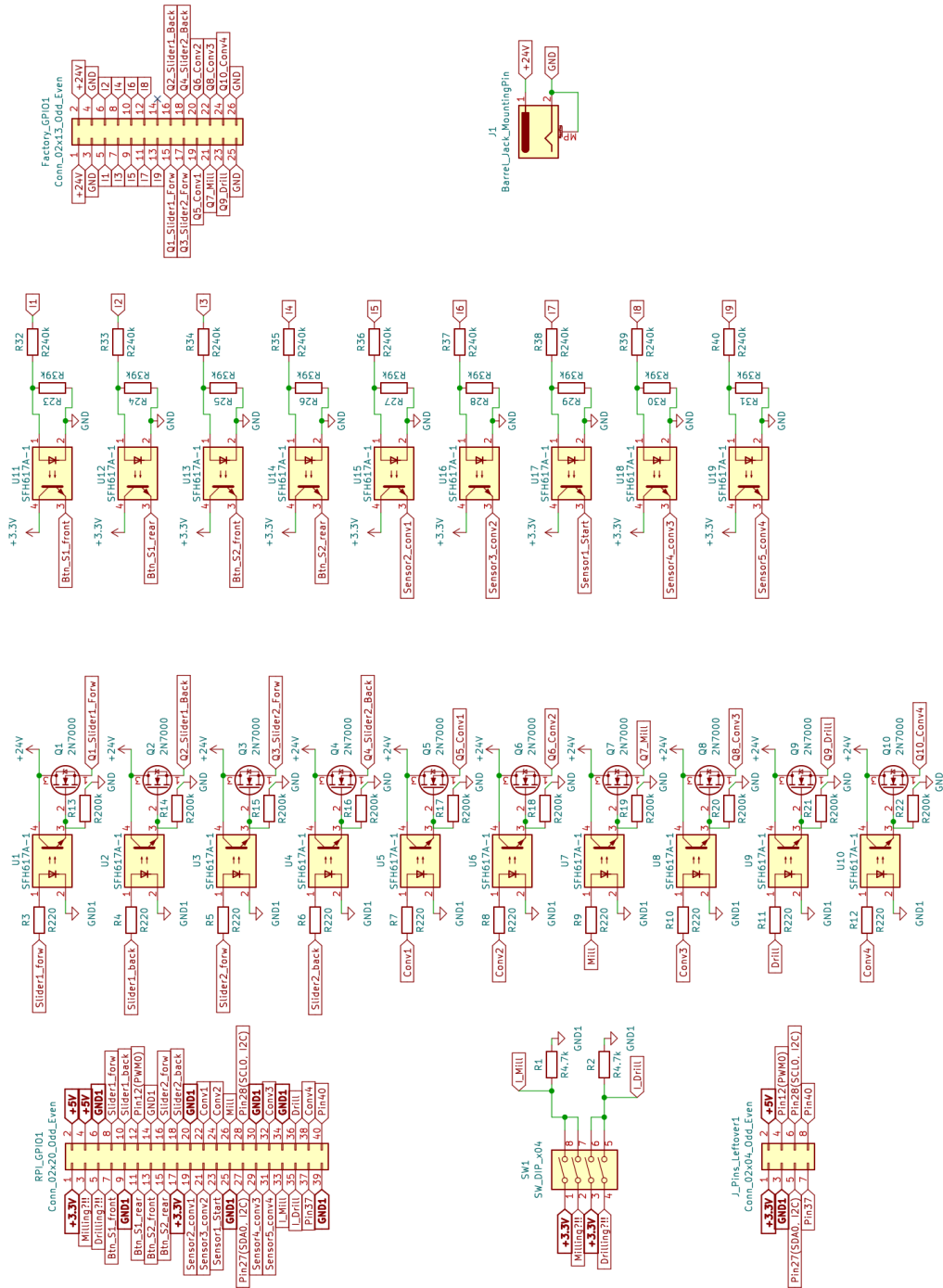


Figure 17: The schematic of the PCB version 1 design.

The physical PCB design is mainly designed to be functional, and easy to understand by observation. The only major design choice made here, is separating the 24V and low voltage sides.

---

Optocouplers are used as the separating layer in this case. This is both to make the PCB design easier to understand, as well as making sure that the RPi or other microcontroller connected won't get 24 volts directly on the input pins.

The traces for sources and grounds that are routed to several locations are also thicker than the rest. This is done to ensure that the resistance is low enough to not heat up the traces and potentially break under high current. The produced PCBs of this design can be seen in Figure 18. The PCBs were ordered from PCBWay [22].

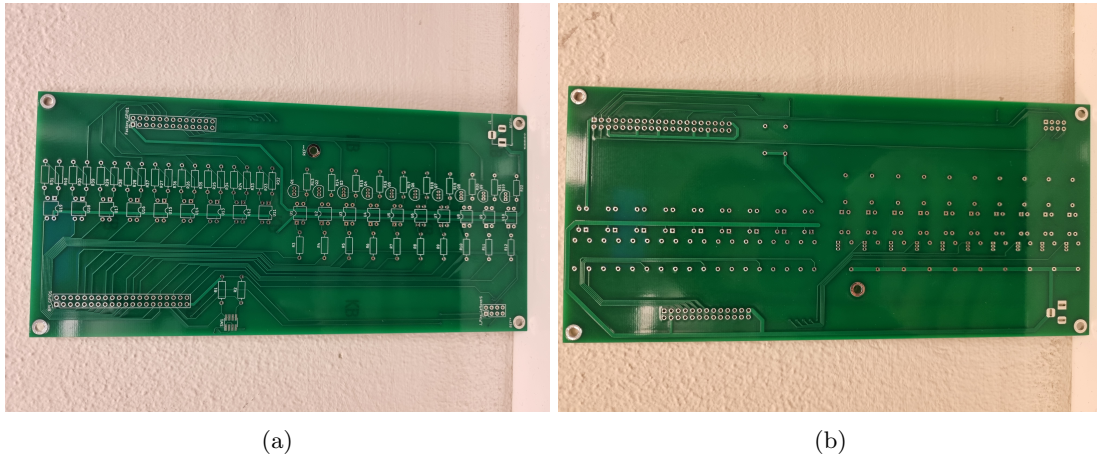


Figure 18: Images of the PCBv1 designed. (a) shows the front, and (b) shows the back of the PCB

### 5.15 PCB version 1 soldering

Soldering the components on to the PCB is a relatively straight forward process. The only hazard is ensuring that the correct resistors is soldered to the correct place, and optocouplers in the right orientation. There were some faults in the design, such as the footprint for the switches being too small. This can however be bypassed by soldering contact between the pads turning on and off the software control of the machining stations. Doing this makes the machining stations only controllable with software.

An additional wire has to be soldered on as a jumper cable between two pins on the 40 pin connector, however this is intentional, since this reroutes the defective pin on the RPi used for the project. This is also mentioned in Section 6.3. With a fully functional RPi, the jumper cable is not necessary to add. This can be seen in Figure 19.

The soldered PCB at this stage produces unsatisfactory results. This is because it isn't functional. The sensor readings is unstable and the outputs do nothing. An investigation of the PCB were conducted, and it was found that there were missing traces. The 24V input traces to the drain-pin of the transistors are missing. The signal trace for the gate of the second transistor is also missing. This was solved by soldering extra wires on the back of the PCB, and was done to make sure this version of the PCB would work for the project.

In this stage, they still did not work as intended. This came from the resistor values being too large. In the simulations, the signal on the gate is pulled down to zero through a 200k resistor when the optocoupler is closed. In real life the voltage was at 8-10V and the transistors were partially open. The problem got fixed by simply lowering the resistor value, and a 4.7k is used on the PCB in



---

Figure 19.

The same problem occurred for the sensor circuit. The values of the voltage divider gave the desired voltage for the RPi to read the signal. But it was too high, restricting the current, such that the phototransistor in the optocoupler didn't fully turn on. This makes the signals from the optocoupler unstable. This was fixed by lowering the values, but retaining the same ratio of the resistors. In Figure 19 the values of 100k and 18k resistors are used.

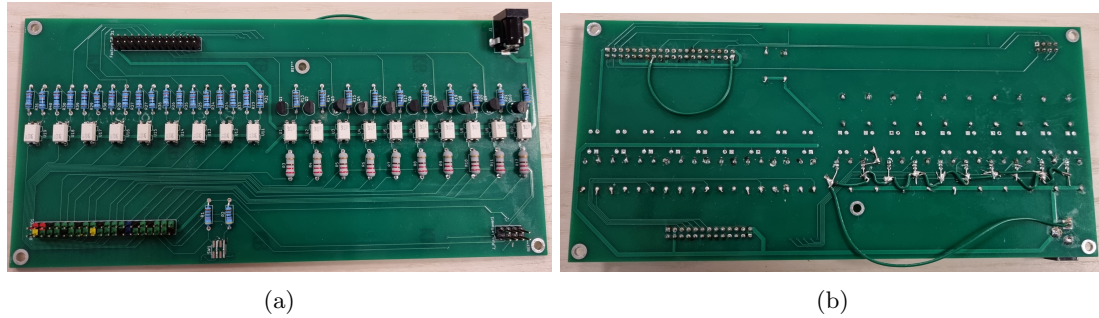


Figure 19: Images of the PCBv1 with the components soldered on. (a) shows the front, and (b) shows the back of the PCB.

### 5.16 PCB version 2

The second version of the PCB is updated with corrections of the direct faults that were made in the first iteration. These were mainly seen after soldering the components on and testing the PCB. In addition, a quality of life improvement were made as well. The footprint of the dip-switches were changed to be big enough to solder the switch package on directly. The most significant change done to the PCB adding traces missing from the first iteration. The schematic were unchanged from the first design.

### 5.17 PCB version 3 - Final version

Version 3 of the PCB is the last and final iteration of the PCB designed for this project. The resistor values were changed compared to iteration 2, and tested in advance to be sure they would work for the circuit. The footprint of the transistors were also changed to a wider form. This change was implemented to ease the process of soldering the transistors onto the PCB. The traces around the transistors also had to be modified when changing the footprint.

The silkscreen on the PCB was also modified, such as the functionality of the leftover pins, as well as the resistor values. The other components were unique, and therefore it was seen as unnecessary to add their type to the silkscreen. The ability to see what kinds of resistors goes where, is helpful during soldering. Some renders of the third PCB design is added in Figure 20b.

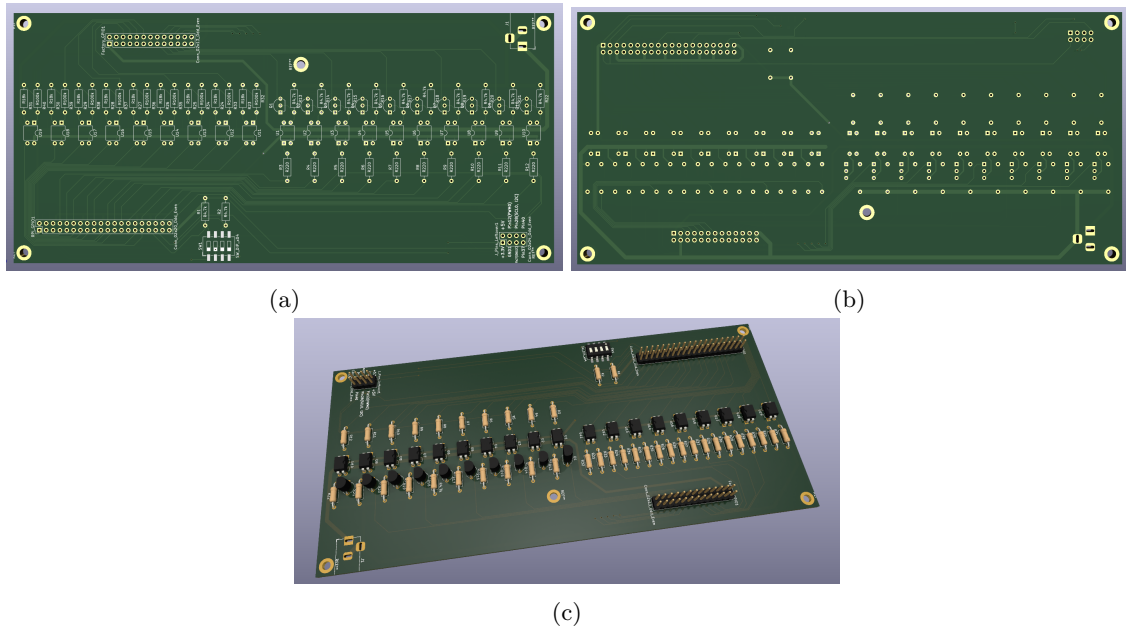


Figure 20: Images of the PCBv3 that was designed by the 3D viewer in KiCad. (a) Shows the front, (b) shows the back, and (c) shows an isometric view of the PCB with the soldered components

## 5.18 OpenPLC

The physical layout of the pins with their corresponding functionality is indicated by the coloured dots in the middle of Table 4. Figure 21 shows where pin 1 is located on the board itself. See the extended table in Appendix E for an overview of the official functionalities of the pins on the Raspberry Pi.

Table 4: An overview of the RPi pin layout with the OpenPLC address corresponding to that pin, as well as the default functionality on the PCB that were designed in this thesis. \* indicates that these pins does not function with OpenPLC across different RPis from our tests.

Pin	OpenPLC address	PCB functionality		PCB functionality	OpenPLC address	Pin
1	-	3.3V	□ ●	5V	-	2
3	%IX0.0 *	Milling?!!	● ●	5V	-	4
5	%IX0.1 *	Drilling?!!	● ●	GND1		6
7	%IX0.2	Btn_S1_front	● ●	Slider1_forw	%QX0.0	8
9	-	GND1	● ●	Slider1_back	%QX0.1	10
11	%IX0.3	Btn_S1_rear	● ●	Pin12 pass through	%QW0	12
13	%IX0.4	Btn_S2_front	● ●	GND1	-	14
15	%IX0.5	Btn_S2_rear	● ●	Slider2_forw	%QX0.2	16
17	-	3.3V	● ●	Slider2_back	%QX0.3	18
19	%IX0.6	Sensor2_conv1	● ●	GND1	-	20
21	%IX0.7	Sensor3_conv2	● ●	Conv1	%QX0.4	22
23	%IX1.0	Sensor1_Start	● ●	Conv2	%QX0.5	24
25	-	GND1	● ●	Mill	%QX0.6	26
27	-	Pin27 pass through	● ●	Pin28 pass through	-	28
29	%IX1.1	Sensor4_conv3	● ●	GND1	-	30
31	%IX1.2	Sensor5_conv4	● ●	Conv3	%QX0.7	32
33	%IX1.3	I_Mill	● ●	GND1	-	34
35	%IX1.4	I_Drill	● ●	Drill	%QX1.0	36
37	%IX1.5	Pin37 pass through	● ●	Conv4	%QX1.1	38
39	-	GND1	● ●	Pin40 pass through	%QX1.2	40



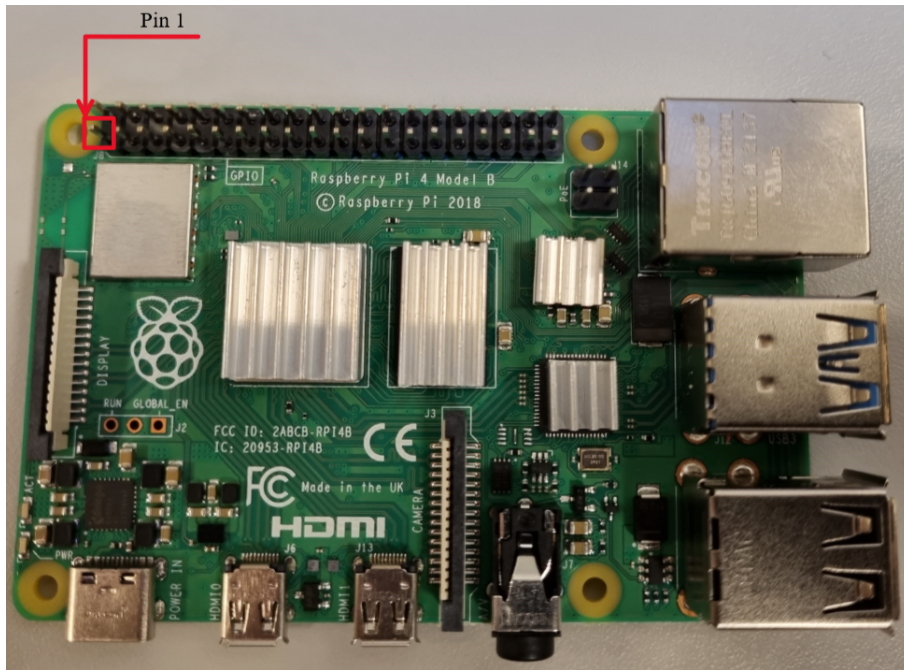


Figure 21: The physical location of Pin1 on the Raspberry Pi. When looking on the RPI in this orientation, Pin1 is located at the bottom left, as the red square indicates.

---

## 6 Software and Simulation

The main purpose of this project, is to make a solid platform for students to learn about Programming Logic Controllers, and the programming language used for these controllers. To do this, *OpenPLC* were chosen. Background theory for this software is written in Section 2.4. This software needs to be downloaded to the PC. There is also additional software that needs to be downloaded to the Raspberry Pi. Downloading the IDE program on PC is straight forward and is done by downloading it from the official site [23].

### 6.1 Connecting to Raspberry Pi

To connect and control an external micro controller, a webserver provided by OpenPLC was used. The guide from the official OpenPLC site [24] was used to connect the server and the Raspberry Pi. The first thing that needed to be done, was to connect the Raspberry Pi to *Eduroam*. A guide on how to do this is provided in [18]. The guide is also provided in Appendix B. After successfully connecting the Raspberry Pi to the *Eduroam* network, the IP-address of the Raspberry Pi needs to be found. This can be done by typing the following line in the terminal:

```
$ ifconfig
```

The IP-address is found under `Wlan0`.

A downside with using this webserver based system, is that the IP address of the Raspberry Pi is not static on the *Eduroam* network. There were multiple occasions where the IP-address changed while running the program. One fix for this, is to get a static address for the Raspberry Pi. This would be a stable fix on a private network. Unfortunately this is certainly not an option for a public network like *Eduroam*, where admin rights for the network is needed. The easiest solution by far is to simply check the IP address regularly.

To connect to the OpenPLC webserver, type in the IP-address on port 8080 in a browser. When successfully connected, the code is sent to the microcontroller via the webserver. From there, the different code files are sorted in a list and saved. OpenPLC saves the programs permanently, which makes it easy to run different code files without the need of uploading the code every time.

### 6.2 Programming and language

The code for this project has been mainly written in two languages. As described in earlier chapters, *OpenPLC* is using the IEC 61131-3 standard for PLC programming. Whichever of the 5 officially approved languages there is a desire to use, the code is compiled down to a ST-file. This is an abbreviation for Structured Text, and is a high-level programming language derived from *Python*.

It was decided to program the mini-factory in two different languages, Ladder Logic (LD) and Sequential Function Chart (SFC). Each of the languages have different structures and capabilities, but commonly, they are both widely used in the manufacturing industry. By programming in these different languages, it's also easier to decide what language to use in a later stage. Another option for the project could be to make scripts in both *Python* or *C*. However, this was later deemed

---

irrelevant for the project due to the relevance of the industrial standard languages.

Since the programs in *OpenPLC* are compiled into an ST-file, it doesn't matter which of the 6 standard approved languages is used. The ST-file is uploaded to the webserver based software on the Raspberry Pi, as described in Section 6.1. From there, the program is compiled into the programming language *C*, which is the programming language mainly used on Raspberry Pis and other microcontrollers. To make the programming easier, the factory was divided into parts, as can be seen in Figure 22.

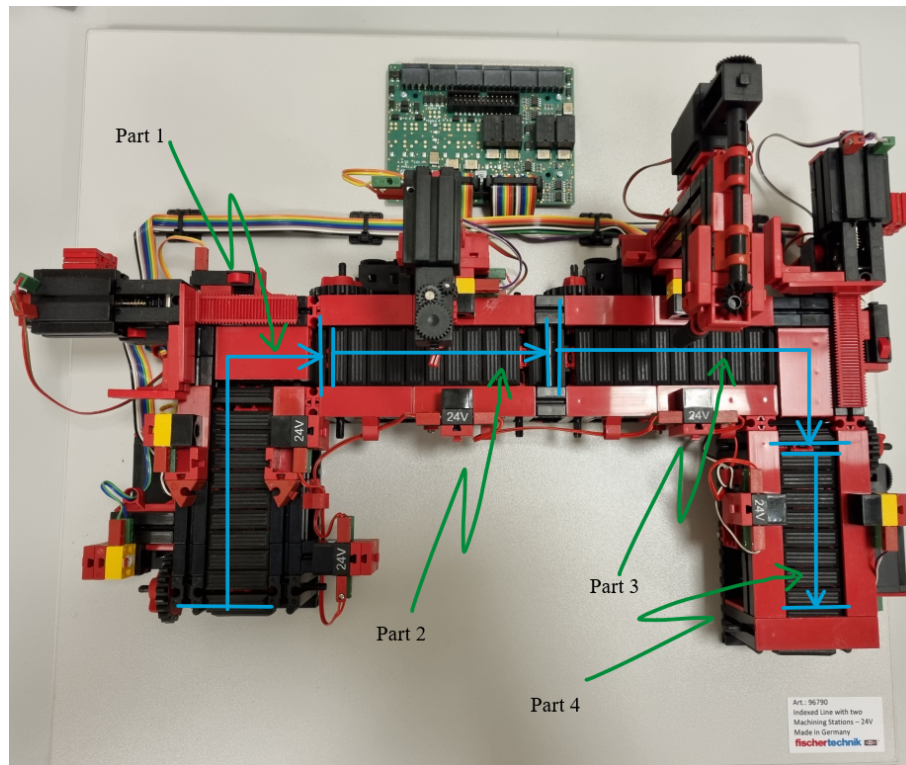


Figure 22: The Factory in bird view, divided into four parts

### 6.3 Ladder Diagram Code

The variables set up to work with the factory on a Raspberry Pi can be seen in Figure 23. These variables are unique for the PCB and can be used for any type of PLC code written. However, the autogenerated delay block variables will differ depending on the specific code, the table shown in Figure 23 is from the code that can be seen in Figure 24.

The variables needs to use these addresses with the Raspberry Pi, because of how the PCB was designed. The variables is the same as was shown in Table 4. If it's desired to rebuild the PCB with different locations, the documentation of the pin addresses for the Raspberry Pi or other microcontrollers are added on OpenPLCs official site [25]. A table of the different pin addresses and functions of the Raspberry Pi is also added in Table 4.

To make the factory work with the specific Raspberry Pi used in the project, the location variable for *Q\_Slid2\_F* on row 14, needed changing from  $\%QX0.2$  to  $\%QX1.2$ . This change was necessary, because the corresponding pin on the used Raspberry Pi was defective.

#	Navn	Class	Type	Location	Initial Value	Option	Documentation
1	I_Btn_S1_F	Local	BOOL	%IX0.2			Button to detect Forward position of slider 1
2	I_Btn_S1_B	Local	BOOL	%IX0.3			Button to detect Back position of slider 1
3	I_Btn_S2_F	Local	BOOL	%IX0.4			Button to detect Forward position of slider 2
4	I_Btn_S2_B	Local	BOOL	%IX0.5			Button to detect Back position of slider 2
5	I_Sensor1	Local	BOOL	%IX1.0			First sensor to start The factory
6	I_Sensor2	Local	BOOL	%IX0.6			Second sensor to detect end of conveyer1
7	I_Sensor3	Local	BOOL	%IX0.7			Third sensor to detect front of milling
8	I_Sensor4	Local	BOOL	%IX01.1			Fourth sensor to detect front of drilling
9	I_Sensor5	Local	BOOL	%IX1.2			Fifth sensor to detect the end
10	I_Mill	Local	BOOL	%IX1.3	0		Input to check if the part is supposed to be milled
11	I_Drill	Local	BOOL	%IX1.4	0		Input to check if the part is supposed to be drilled
12	Q_Slid1_F	Local	BOOL	%QX0.0			Slider 1 forward drive
13	Q_Slid1_B	Local	BOOL	%QX0.1			Slider 1 backward drive
14	Q_Slid2_F	Local	BOOL	%QX0.2			Slider 2 forward drive
15	Q_Slid2_B	Local	BOOL	%QX0.3			Slider 2 backward drive
16	Q_Conv1	Local	BOOL	%QX0.4			Conveyer 1 at the start
17	Q_Conv2	Local	BOOL	%QX0.5			Conveyer 2 at the milling station
18	Q_Conv3	Local	BOOL	%QX0.7			Conveyer 3 at the drilling station
19	Q_Conv4	Local	BOOL	%QX1.1			Conveyer 4 at the end
20	Q_Mill	Local	BOOL	%QX0.6			The milling station
21	Q_Drill	Local	BOOL	%QX1.0			The drilling station
22	Deley	Local	TIME				
23	TON0	Local	TON				
24	TOF0	Local	TOF				
25	TON1	Local	TON				
26	TOF1	Local	TOF				
27	TOF2	Local	TOF				
28	TON2	Local	TON				
29	TON3	Local	TON				
30	TOF3	Local	TOF				
31	TON4	Local	TON				
32	TOF4	Local	TOF				
33	TON5	Local	TON				
34	TOF5	Local	TOF				

Figure 23: The PLC code variables

To make this change possible, the PCB was modified by adding a jumper cable between the broken pin (in this case pin 16) and the only leftover output pin, pin 40. If a Raspberry Pi has more than one of the eleven output pins broken, it will not be able to run all the functions on the mini-factory.

As mentioned in the theory part in Section 2.1.1, the Ladder Diagram language is made to be similar to relay system schematics. That means the name of the different functions is inspired from these. The end sides on the LD code are called power-rails. The input signals such as **I\_Sensor1** and **I\_Btn\_S1\_F** are called contacts. The output signals such as **Q\_Conv1** are called coils. The program works by running power from the left rail to the right one. The contacts will stop or let the power through depending on its input signal. If power runs through a coil, the output that the coil controls will also be powered up. In addition, there are modifiers that will change how the contacts and coils work, those will be described when encountered.

### 6.3.1 LD Code part 1

First, some simple conceptual tests were done to see if the OpenPLC program would work as intended. An example of this can be seen in Figure 25. When the puck blocks the light at sensor 1, it sets the first conveyer belt, indicated by **S** in the coil. This conveyer will be activated until the conveyer is reset, like a latch catching a signal. It is reset again when seen by the second

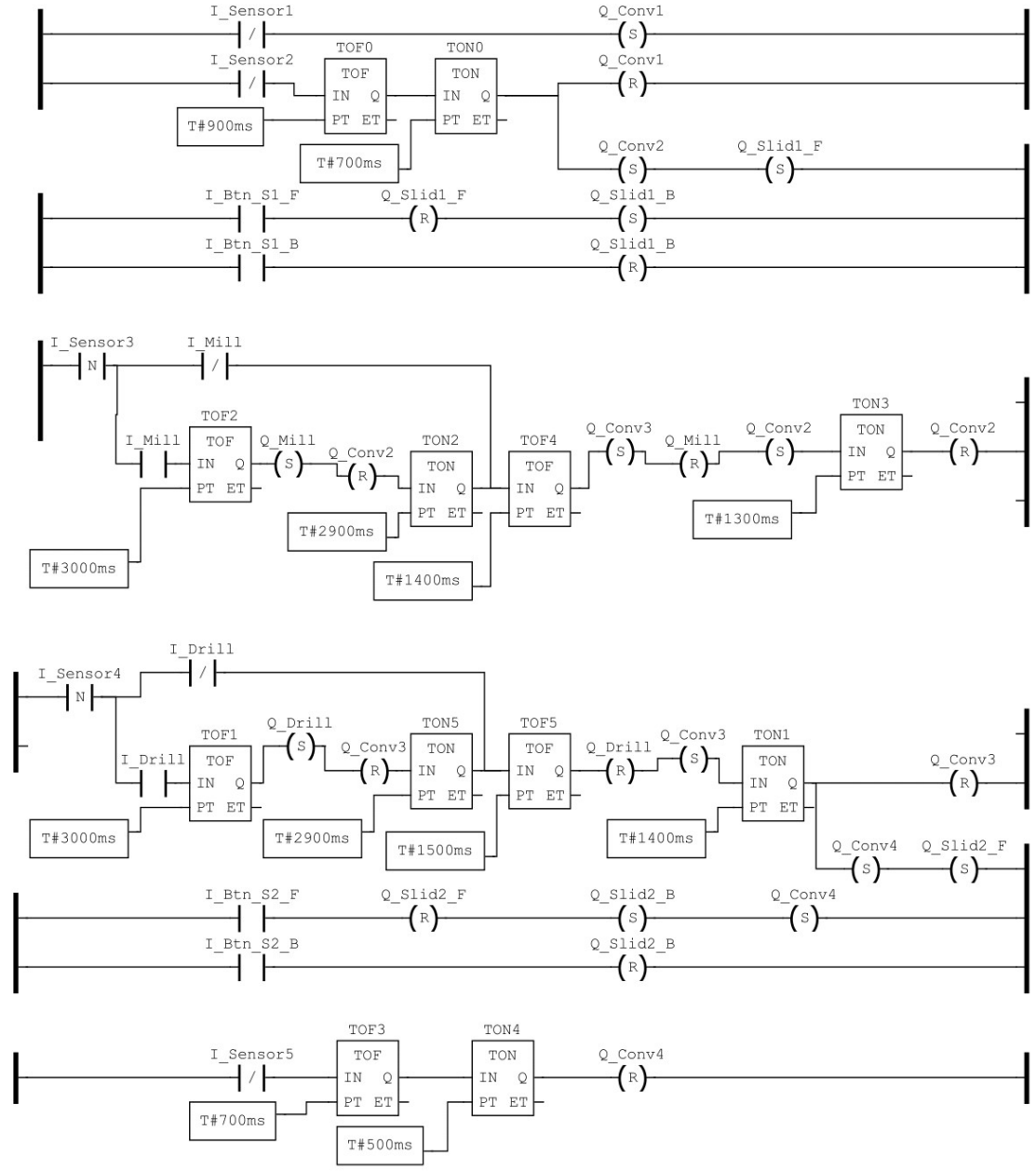


Figure 24: The complete Ladder Diagram

sensor, indicated by the **R** in the coil. Sensors in a LD code have a slash symbol in them, which symbolizes that they have a normally high value. This symbol indicates that the signal is negated. Buttons do not have this symbol, because they have normally low value.

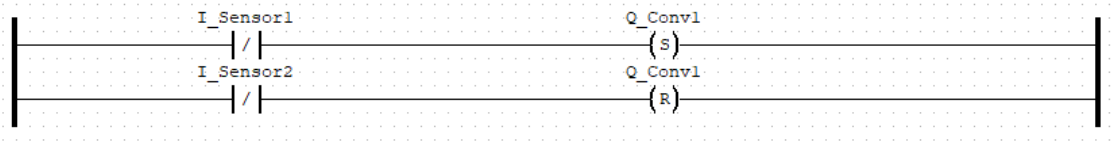


Figure 25: Simple test of Ladder Diagram.

To evaluate the programming for the whole mini-factory, the first corner was programmed first. Since the factory attributes are quite similar the whole way through, the rest of the program should

be simple to write when solving the first corner. The first part including the first corner is called Part 1 in Figure 22, and can be seen in Figure 26. In Figure 27 the code for the first part can be seen, and it starts similarly to the simple test Figure 25. There is however, added two delay blocks that activates when the signal at sensor 2 is blocked, and times the deactivation of the conveyor belt. This is to ensure that the puck is in front of the slider before further actions are taken.

The rail is also forked off, to start the slider and next conveyor belt at the same time. The delays are in a pair of **TOF** (The off-delay timer) and **TON** (The on-delay timer). The **TON** is there to hinder the power signal resetting to early. Using only the **TON** would not work, because if the signal becomes low before the timer is over, the next parts would never be set properly. This happens when the puck is moved away from the sensor before the delay is over. A **TOF** is used to catch the signal for the **TON**.

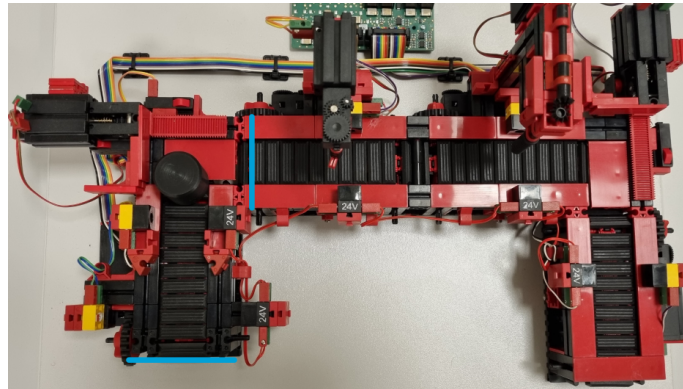


Figure 26: The Factory in bird view, showing part 1.

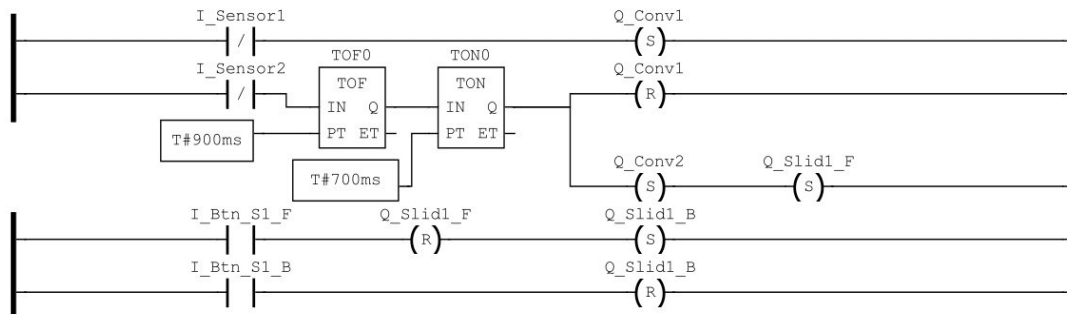


Figure 27: The first part of the Ladder Diagram.

The slider starts to move forwards, and it continues to do so until it the first button is pushed, which is indicated by **I.Btn.S1.F** in the code. The forward movement of the slider is subsequently reset, and sets **Q.Slid1.B** which makes the slider move backwards. This is done until the slider pushes the second button, indicated by **I.Btn.S1.B**. The backwards movement is reset, and the slider stops. The next conveyor belt is started while the slider is moving forwards. This was done to prevent the puck crashing into the belt. Which happened sometimes if the conveyor belt wasn't already running.



### 6.3.2 LD Code part 2

The code for the second part can be seen in Figure 29. Additionally, the bird view of the factory can be seen in Figure 28. This part includes the second conveyor belt and milling station. The first interaction starts when sensor 3 detects the puck. In the contact there is a modifier **N**, that only detects a falling edge signal pulse. A falling edge is detected by a high value signal falling down to a low value. This was necessary because the puck will stay in the same location for a significant time and continuously reset the belt if the puck is supposed to be milled. After the third sensor detects the puck, the code splits into two paths. The path chosen depends on the sensor reading telling the code if it should be milled or not. It will always choose only one of the paths, since one has normal **I\_Mill** and the other path negated **I\_Mill**. The paths converges at a later point, the negated path just skips the milling part. In Figure 30 an alternate layout of the code can be seen. Although the code does exactly the same, it illustrates better what happens when. That is because the coils is set and reset after the **TOF** delays. Since coils only need a pulse of a signal in that case, the coils can be put in front of the delay as well.

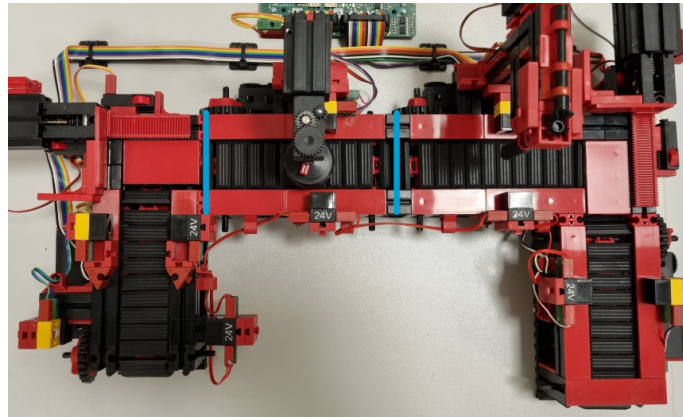


Figure 28: The Factory in bird view, showing part 2

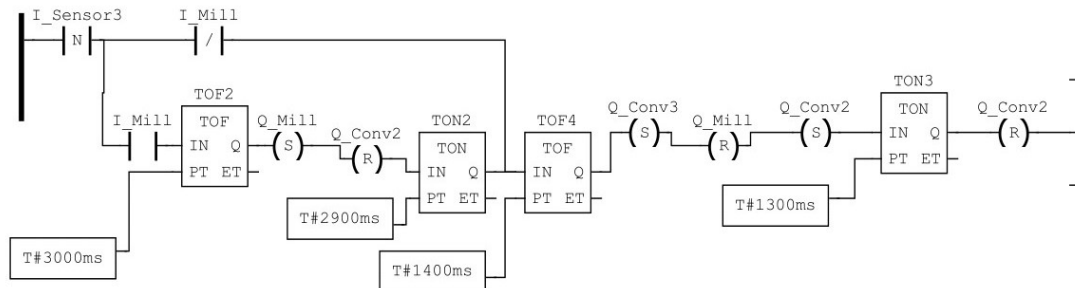


Figure 29: The second part of the Ladder Diagram.

On the path with **I\_Mill**, **Q\_Conv2** is first reset and **Q\_Mill** is set. Next, there is a delay pair that ensures a certain time for milling. The code will thereafter set **Q\_Conv2** and reset **Q\_Mill**. It is in this location the paths converges, and the next conveyor belt **Q\_Conv3** is set.

In the alternate layout, the negated path only sets **Q\_Conv3**. Since **Q\_Conv2** is never reset and **Q\_Mill** is never set, the path can converge in front of **Q\_Conv2** set and **Q\_Mill** reset. Because they will already be in these states. After **Q\_Conv3** is set, there is a new delay pair that waits long enough for the puck to settle on the third belt and subsequently resets **Q\_Conv2**.

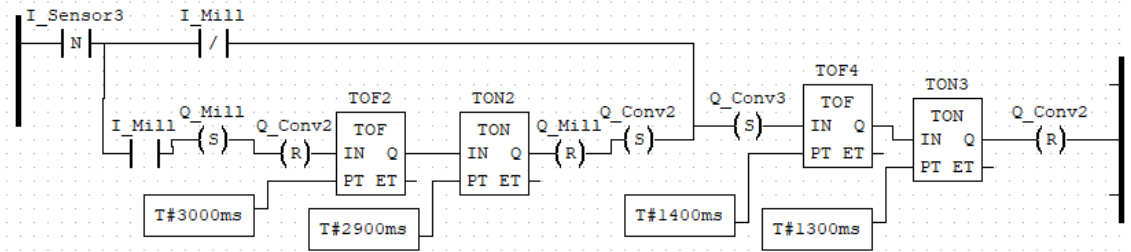


Figure 30: The second part of the Ladder Diagram in an alternate layout.

### 6.3.3 LD Code part 3

Part three is seen in Figure 31, with the corresponding code in Figure 32. This code is almost identical to the code Section 6.3.2. The section includes some extra modules, and the milling station is replaced by a drilling station. The only difference with the not highlighted part in the code is not setting the next conveyor belt, when it has either finished, or bypassed the drilling. This is because the next actuator is a slider and not a conveyor belt. Meaning that the highlighted part is similar to the slider part of Figure 27.

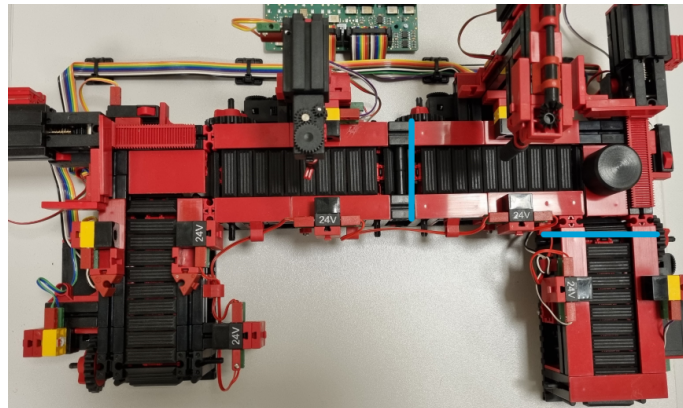


Figure 31: The Factory in bird view, showing part 3

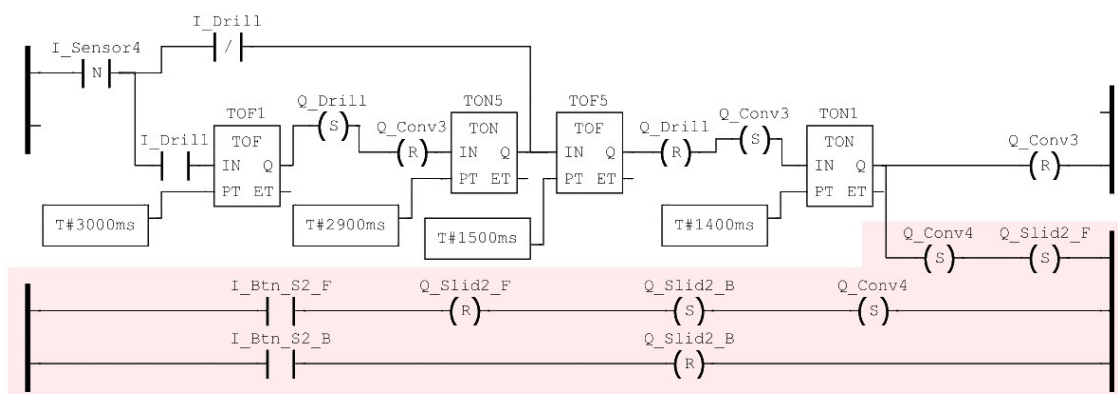


Figure 32: The third part of the Ladder Diagram

This section of the code starts by looking for a falling edge on sensor 4. When this is detected, the code continues in one of the two paths. If the indicator **LDrill** has a high value, the puck will be drilled in the drilling station. The **TOF** delay will be activated while the factory drills, before the



conveyor belt **Q\_Conv3** is set and the drill is reset. This is the point where the paths converges, and the path with the deactivated drilling station does nothing up until this point. Then, the delay is long enough for the puck to be in front of the second slider, before it resets **Q\_Conv3**. **Q\_Conv4** and **Q\_Slid2\_F** is simultaneously set. The slider will then move forward until the front button indicated by **I\_Bt\_S2\_F** is pushed.

At this point **Q\_Slid2\_F** will reset and **Q\_Slid2\_B** set, **Q\_Conv4** will also be set again. **Q\_Conv4** in the highlighted section could be removed, since it's already set earlier in the code. The push-button **I\_Btn\_S2\_B** is then pushed by the slider, and stops.

### 6.3.4 LD Code part 4

The last part of the code, shown in Figure 34 is controlling the fourth part of the factory. This code controls the last section, which can be seen in Figure 33. The code is short, and its only functionality is to stop the last conveyor belt **Q\_Conv4** when sensor 4 detects the puck. A delay was also added, since it was preferred that the puck travels a bit further than the last sensor.

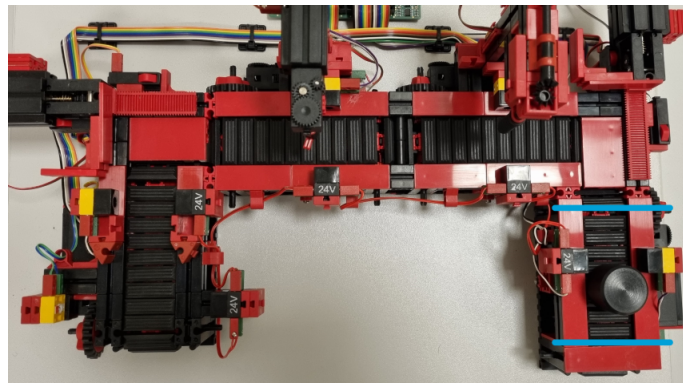


Figure 33: The Factory in bird view, showing part 4.

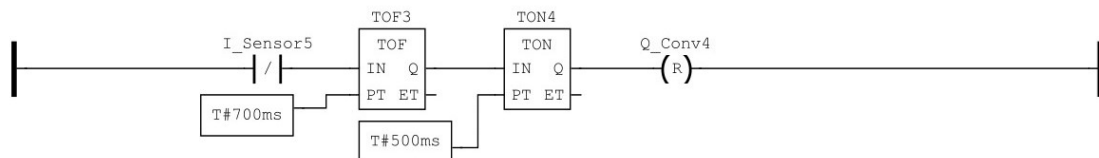


Figure 34: The fourth part of the Ladder Diagram.

### 6.3.5 Discussing the Ladder Diagram Code

The ladder diagram code is not the prettiest nor the most readable code written, neither is it the most optimal. However that was not the point of its development either, the point was to create a code that works, which it does. In this sense it is perfect. It was created in parts, and functionalities were added along the development. The alternate version of the second part seen in Figure 30, were created to easier describe its functionality. To use the code as a proposed solution, the whole code should be streamlined, but for prototyping it works as intended.

---

## 6.4 SFC

Another standard language used for PLC, is called SFC. In addition to the LD code, a code in SFC has been made for the project. Like the Ladder logic code discussed in Section 6.3, the SFC code was made in OpenPLC as well.

The code starts with an initial statement. Action blocks are then built in several steps. The action blocks features different qualifiers that represents commands. These are written as letters. Some examples are timing features, set and reset.

Each step is divided by transition steps. These transitions are often connected to a sensor or other types of input signals. This makes it easy to regulate how long a step is active. One of the main differences between Ladder Logic and SFC, is the ability to jump in a sequence randomly. In Ladder Logic, activating a sensor in the middle of the sequence will start a new sequence starting where the sensor is located. Compared to SFC, which has to finish the whole sequence before starting all over again.

This type of language is commonly used in production lines, as well as other autonomous installations like car washes. Full action blocks or programs can be connected to a **Step**. For example, on a car wash program, "wash" and "dry" can be independent programs connected together with steps and transitions. These smaller programs are usually called actions [26].

The variables set for this code, has the same addressing as the variables in the Ladder Logic code, seen in Figure 23. Disclaimer, this is only a prototype code which could be streamlined if it was going to be used as a solution in an exercise. However, it's working correctly, and shows the basic structure of a SFC code.

### 6.4.1 SFC code part 1 and 2

This section highlights the first part of the code, seen in Figure 35. As can be seen, the code starts to run from the initial step. The initial step is called **Step0** in this instance. On the first transition between **Step0** and **Step1**, an input variable has been set. These input boxes works with normal boolean values. Since **I.Sensor1** is normally set to true, the program will continue when this signal is set to a low value. This happens when the object on the conveyor breaks the light signal, and the input signal on the transition is set to **NOT False**. To keep the code tidy and easier to read, all of the input variables have been set as inputs to the transition steps on the left side of the chart.

On **Step1**, an action is connected to the step. This action starts the conveyor as soon as the step is activated. It has the qualifier **S**, which starts the belt when activated, and runs the belt until a specific reset command is given. This command is given by the qualifier **R**, which can be seen connected to **Step2**.

Due to the layout of the mini-factory, some timers needs to be added to ensure that the puck is in the correct position before each action is proceeded. For this, a **TON** and **TOF** timer has been used. Together, they start the timer when the light signal on **I.Sensor2** is blocked, and activate again after 2.7 seconds. This seemed like an appropriate delay for the program. When the timer runs out, **Step2** activates, and the second conveyor is reset. How the **TOF** and **TON** timer works together is described in Section 6.3.1.

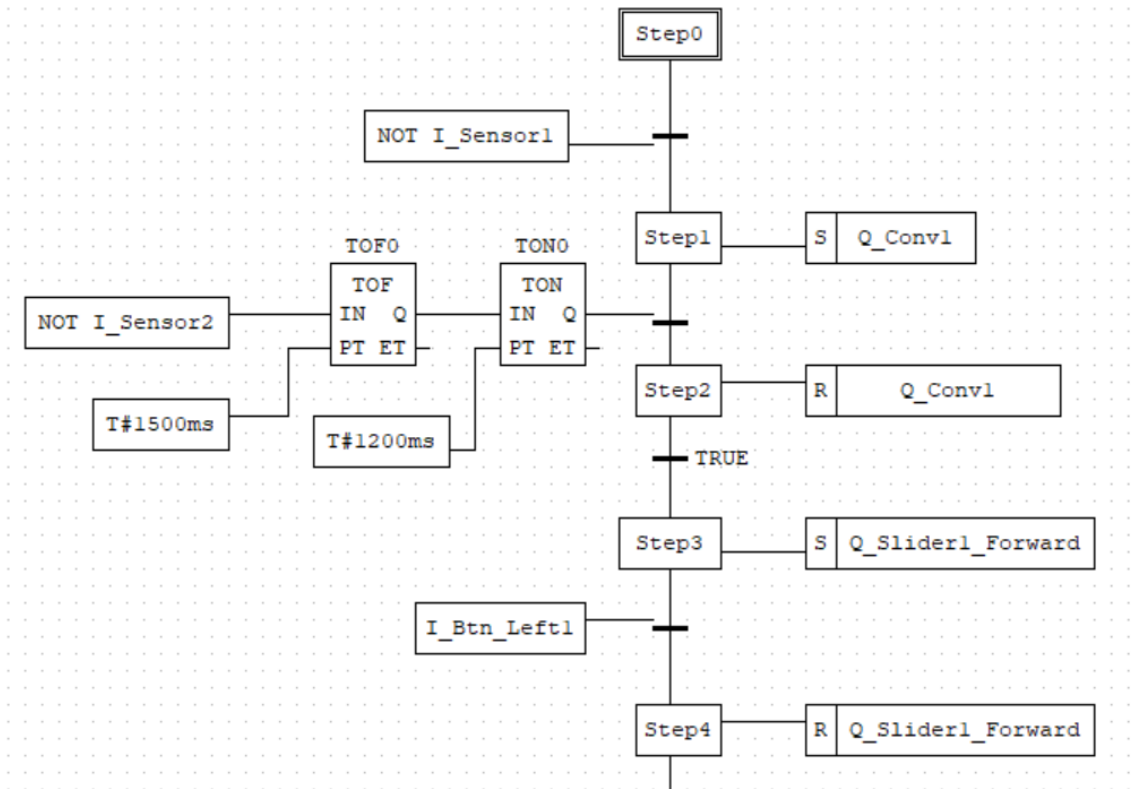


Figure 35: The first part of the SFC code

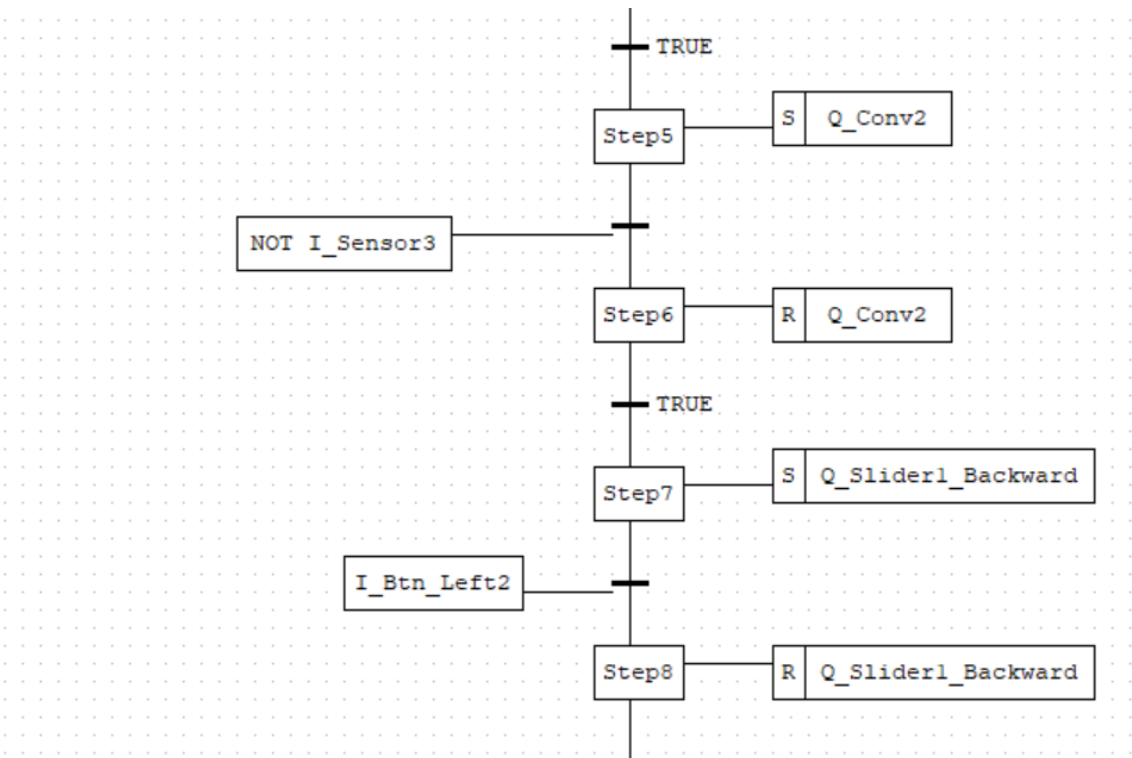


Figure 36: The second part of the SFC code

**Step3** to **Step8** describes the slider mechanism. At **Step3**, the conveyor is set, and is reset again when the pushbutton is pressed down by the slider. This will set the input signal to **True**, and the

program will continue further to **Step4**. As can be seen from figure 36, **Step5** starts the conveyor. The conveyor is then reset when the light signal at **I\_Sensor3** is blocked. This sensor is located at the milling station. When the sensor at the milling station is activated, the belt is reset, and the slider goes back to it's original position in **Step7** and **Step8**.

#### 6.4.2 SFC code part 3 and 4

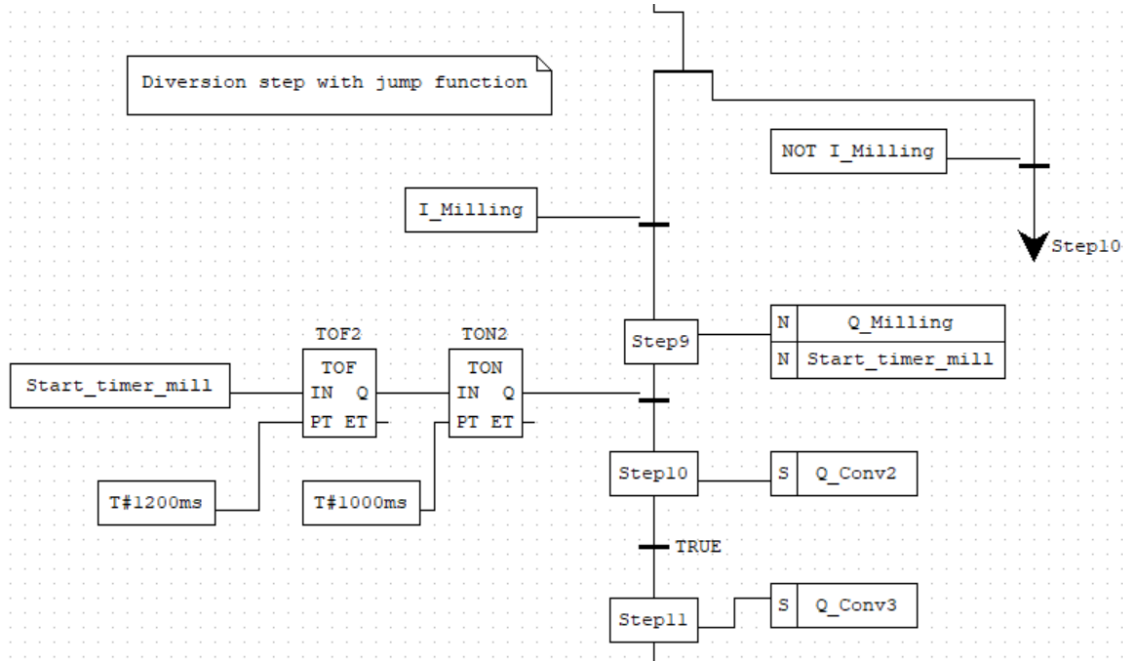


Figure 37: The third part of the SFC code

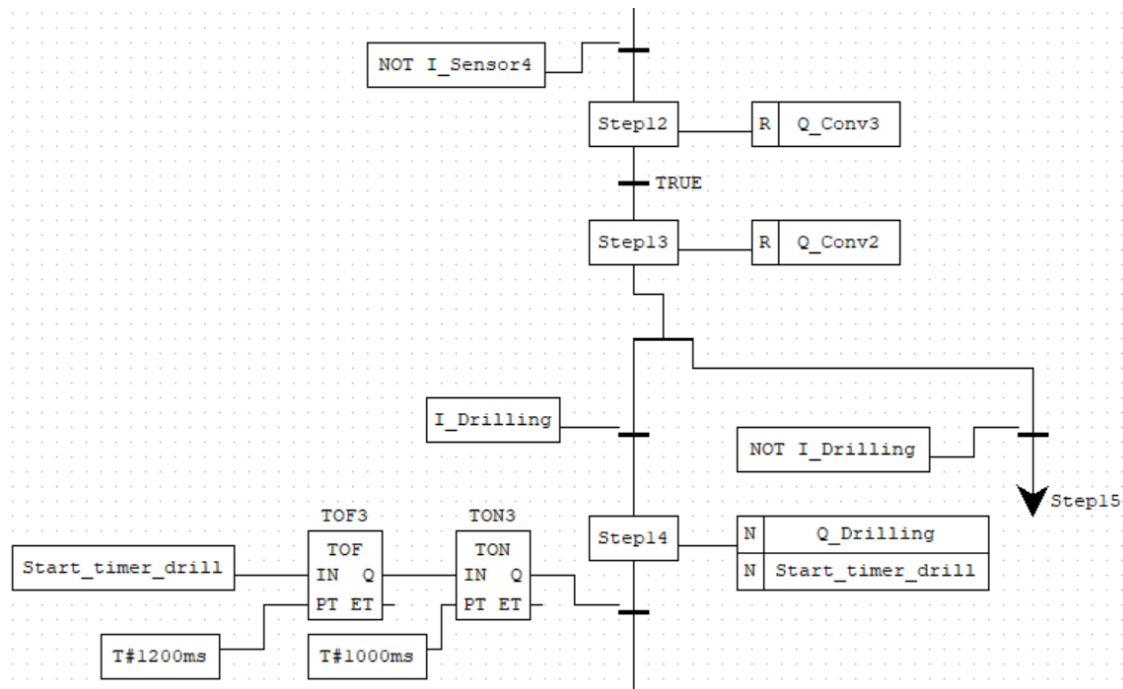


Figure 38: The fourth part of the SFC code

It was desired to use an other code to decide if the machining stations should run. To implement

this in SFC, a division step was made. This can be seen in Figure 37. This division step is corresponding to the **OR** command in boolean terms.

This section of the code, decides what path that should be taken based on a **True** or **False** statement. This statement tells if the milling is going to be activated or not. If this statement is **False**, **NOT I\_milling** is **True**, and the code jumps down to **Step10** without executing the path via **Step9**. If the object is going to be milled, **I\_Milling** is **True** and the milling starts. This step has the **N** qualifier, which means that the conveyor will run as long as the step is active. The time delay is currently set to 2.2 seconds.

In **Step10** and **Step11**, conveyor 2 and 3 are set almost simultaneously. As seen from figure 38, these will run until **I\_Sensor4** is activated. This was done to ensure that conveyor 3 was running when the puck came onto it. **Step10 - 13** could have been shortened down to two steps, although it was decided to use only one action per step, to minimize potential errors. The division step for the drilling station is exactly the same as for the milling station with the same delay.

### 6.4.3 SFC code part 5 and 6

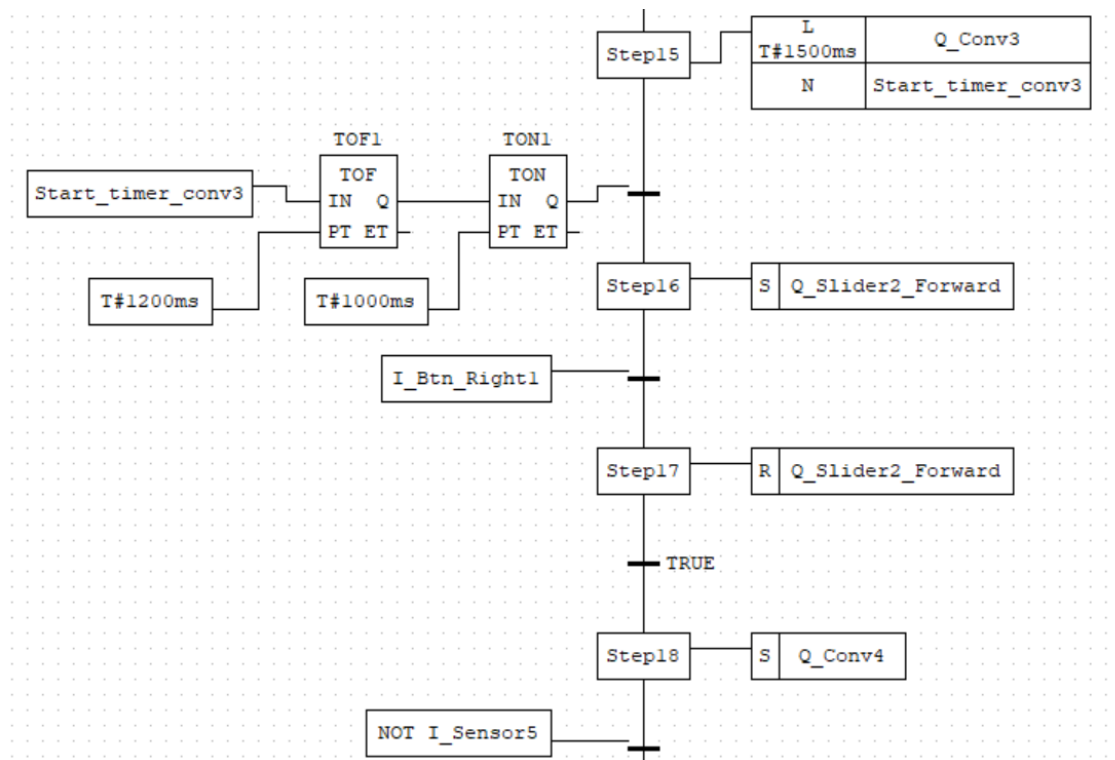


Figure 39: The fifth part of the SFC code

At **Step15** in Figure 39, it was set that conveyor 3 should run for 1.5 seconds. This was done with the **L** qualifier, which runs the action for a specified amount of time. Due to it being no sensors at the end of conveyor 3, this was the easiest way to ensure that the puck has come onto the slider-platform. At the same step, it was set that the slider was going to run after a delay of 2.2 seconds.

**Step16** and **Step17** moves the slider the same way as described in Section 6.4.1. **Step18** sets conveyor 4, and resets when the light signal in **I\_Sensor5** is blocked. As can be seen in Figure 40,

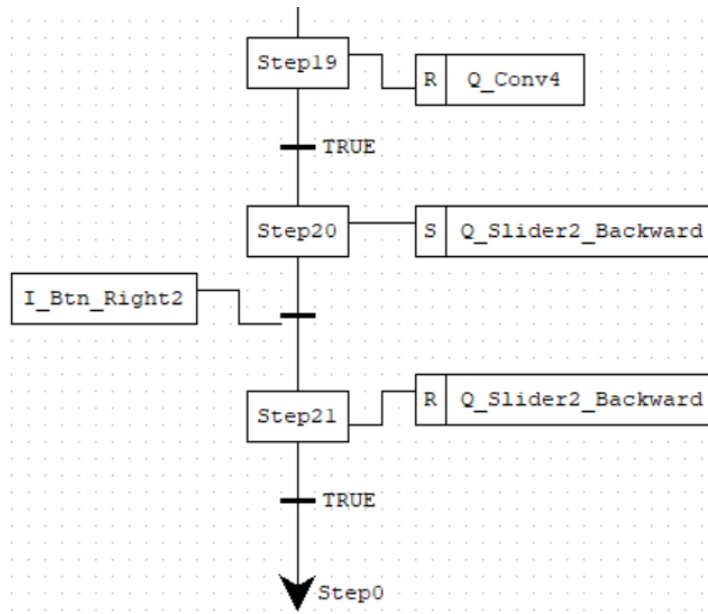


Figure 40: The sixth part of the SFC code

the last two steps resets the slider the same way as described in Section 6.4.2. The final jump skips to the initial step, the sequence is finished, and the factory is ready to start over again.

## 6.5 Simulation

In addition to running the project on the mini-factory, there was also a desire to simulate it. This way the students taking *TPK4128* should be able to run the code developed by the students on a simulation before running the actual factory. Due to limited amounts of factories and Raspberry Pis, this will streamline the process of coding and logistics. The software chosen for this task was *Visual Components*, mainly due to accessibility and capability. As per 2022, *MTP* has an agreement with *Visual Components*, which makes the software available for students taking certain classes. The version that were used for this project was *Visual Components Premium 4.5*.

### 6.5.1 Modelling

The modelling starts with setting conveyor belts into the working environment, in roughly the same shape as the factory. These are quite easy to connect due to the *PnP* tool, which makes the parts snap on easily to each other. To simulate the phototransistors, similarly functioning sensors are found in the menus and added onto the conveyors.

On the mini-factory, there are stations which machines the parts on the belt itself. These kind of stations cannot be found on *Visual Components*, so a different solution is needed. To overcome this, two *Doosan* robot arms are added to the factory. The robot arms are chosen due to their size. These robot arms performs a simple pick and place operation to move the boxes from the belt, to each separate machining station. These are programmed using the internal programming tools and functions. The same operation is used in reverse, to pick up the boxes from the machining stations back onto the belt. It is not necessary to choose 6-axis robotic arms, since 4 axes are

---

sufficient for the operation.

One of the main features of the mini-factory are the sliders located at the corners. The sliders go on tracks that is unable to stop them if they are going too far off. Therefore, there needs to be a way to stop the sliders before they go off the tracks. The way this has been solved by *Fischertechnik* is adding push-buttons at both ends of the tracks.

These sliders doesn't seem to have any counterpart on *Visual Components*. There are some sliders, but they stop by themselves without pressing push buttons. For the model to work like the real factory, a different solution for these have to be made. Therefore, a visual model has been made instead to show how the configuration of the mini-factory could look like in real life. Time constraints and connection issues described in the subsequent sections are also reasons for this simulation to not be explored further in this project thesis. An image of the model is provided in Figure 41.

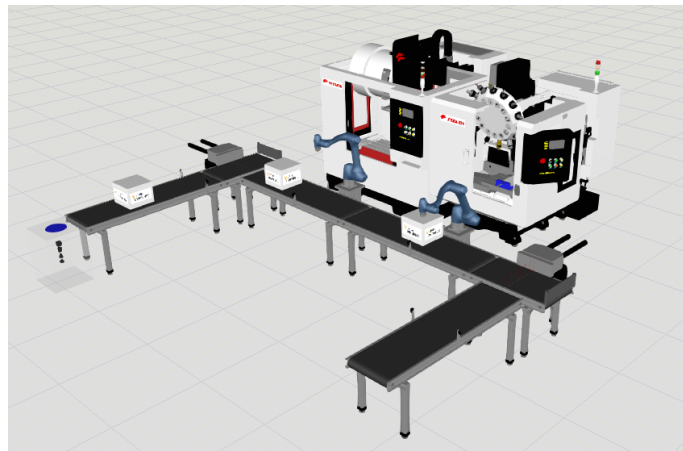


Figure 41: The mini-factory modelled in Visual Components

## 6.6 Connecting to VC

There are mainly two ways to run code in *Visual Components*, one is with a Python script inserted directly into the program. The second option is making a connection from an external program to run the simulation. Visual Components allows a connection in the form of OPCUA explained further in Section 2.5.

A problem that was encountered in this project, is that OpenPLC doesn't support OPCUA yet. This made some difficulties connecting to the virtual factory, and running PLC code on the simulation. To avoid this problem, more ways of connecting were investigated. One of the methods investigated, was the use of *ModBus*. Modbus is supported by OpenPLC and is often used to connect and import code to controllers without I/O. The problem using Modbus, is that it's not supported by Visual Components, therefore this solution can't be used directly to run the factory.

### 6.6.1 Possible solutions

There are multiple solutions that can be applied to run the factory with the desired code. First, there is a need to set some ground rules about what is wanted out of the simulation, and the

---

solution making this possible.

- Must be able to use code-files provided by *OpenPLC*
- Easy and user-friendly to connect and run the code
- Free and ideally Open-Source software
- As few layers as possible

This is the demands that the solution have to fulfill, in order to be considered a viable solution for the set requirements in the objectives set in the project.

### 6.6.2 Implement FreeOPC

One of the first solutions that comes to mind, is using *FreeOPC* to make a connection between OpenPLC and Visual Components. FreeOPC is an open-source *GitHub* library, mainly used for creating servers and clients in Python. Using this requires writing both a server and a client. The client will receive information from a *Modbus* from the OpenPLC Runtime, the server will further send the information to *Visual Components* using OPCUA.

The problems of this solution is complexity, in addition to adding an extra communication layer between the programs.

### 6.6.3 Implement OPCUA into OpenPLC

Another solution to this problem, is if *OPCUA* gets integrated into OpenPLC. OPCUA is explained in Section 2.5. According to "thiagoralvez" which is one of the main contributors to OpenPLC, an attempt has been made to implement this in 2022. This was however not implemented correctly, and therefore never added to the main OpenPLC repository on GitHub [27].

This doesn't mean there will never be an OPCUA expansion on OpenPLC, just that it's yet to be implemented into the program. Since OpenPLC is an Open-Source project, it's possible to contribute and make the expansion. This is very time-consuming and requires a lot of programming. The benefit of this solution is that it contributes positively to an interesting Open-Source project, in addition to being helpful for many people in the future. A variation of this would also be the easiest solution to use of the ones stated. Figure 42 is an illustration of how this connection would work. OpenPLC act as a master to send signals to Visual Components. Visual Components will further send replies back to the computer using OPCUA. This way of communication is not implemented in Visual Components, therefore there is a need to implement FreeOPC to give the replies back to the master.

OPCUA is already in use for other assignments in Industrial Mechatronics. Therefore it will prove beneficial for the course if this gets implemented in the near future. This will make other assignments regarding the mini-factory possible.



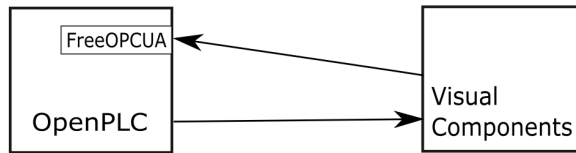


Figure 42: Connection OpenPLC to Visual Components using FreeOPC

#### 6.6.4 TCP/IP addressing

Another solution to the problem, is possibly the use of TCP/IP and direct addressing. Every device with an internet connection has their own IP-address on the connected network. It's possible to connect directly to these devices using the direct addressing.

### 6.7 PCB design software

For the printed circuit boards designed in this thesis, the Open Source program *KiCAD* is used. It is chosen due to being a free program, with all the functionality needed to design PCBs. It can be downloaded from their website [28]. KiCad uses a unique file format, so opening these files in the future will require downloading the program.

---

## 7 Discussion

### 7.1 Use of mini-factory in Industrial Mechatronics

*Sund* has in his master's thesis [3] found that there is a desire to use the mini-factory to replace existing assignments in Industrial Mechatronics. The thesis was discussing the options of using actual PLCs to run the factory. Due to licensing issues and the issue of actually acquiring PLCs it was an impracticable solution. However, his research stated that it was valuable experience for the students taking the course.

This project thesis focuses on further developing *Sunds* research and present an alternative option to replace PLCs for the assignments. The Raspberry Pi has a lot of benefits over a PLC regarding educational content. One major benefit is the adaptability and ability to be used for multiple assignments in the course. In addition, all of the software used in this project are both free and open-source, which eliminates cost and licensing errors.

### 7.2 The hardware

The solution to the main objective of this project was to use a Raspberry Pi to emulate a PLC.

The assembly of the Raspberry Pi and the PCB can be considered an actual PLC, which does not fulfill the formal standards for PLCs. Therefore, the PCB and Raspberry Pi together can be seen as an emulated PLC, since they together have all of the software features to be considered a PLC with OpenPLC installed. This is according to the PLC software standards [12]. The Raspberry Pi with OpenPLC will also interpret all of the 5 languages supported in the standard [12].

Several other solutions were also considered to work as a PLC emulator, one of which included using an Arduino as the microcontroller. OpenPLC provides a program compatible with multiple Arduinos, the software would then work the same way as if it were on a Raspberry Pi. One of the problems with using an Arduino, is the limited amount of GPIO pins. A normal Arduino Uno is short of the number of pins required to read all the inputs and outputs from the factory. There are possibilities to fix this, including using an I/O extender, or just use an Arduino MEGA. Arduino MEGAs and I/O extenders were not in immediate access, therefore it seemed better to use a Raspberry Pi. Another benefit of using a Raspberry Pi is that it's already a big part of the assignments in Industrial Mechatronics, which means that the institute does not have to buy new, or modify existing microcontrollers.

One of the main problems of this assignment was to separate the Raspberry Pi from the higher voltage required to run the mini-factory. A solution that was considered to solve the problem, was to make a master-slave configuration with the Raspberry Pi and Arduino, totally separating the Raspberry Pi from the system. However, due to the added complexity, and other problems stated above, it was decided to use a Raspberry Pi directly and solve the problem using hardware instead. How the specific solution was implemented is found in Section 5.7. One way to implement TCP/IP in an assignment regarding the mini-factory, could be to use an Arduino MKR WiFi<sup>3</sup> as a slave to control the first corner of the factory, and use a Raspberry Pi to control it with TCP/IP. With the current PCB design, it's possible to implement this solution, although a bit inconvenient.

---

<sup>3</sup>This is a specific Arduino from the maker series with WiFi capabilities.

---

### 7.3 PCB

The PCB produced and used in the project thesis is the version 1 design in Section 5.14. As was mentioned in both Section 5.14 and Section 5.15 there are some faults in the design. In these sections, it is also described how to fix or circumvent these faults. This is fine for the prototype however, this is not a good solution for a finished product. Since multiple will be produced for educational purposes, it will be beneficial to have a version that does not need quick fixes during manufacturing. It is recommended to produce a newer PCB design. Version 3 described in Section 5.17 is an adequate solution. This is an improvement of the version 1 design, with the faults fixed. The best solution however, is to further refine the design before new PCBs are produced. Some of the recommendations in further works about PCB in Section 7.7.2 can then be implemented.

### 7.4 Multisim

One of the key programs that were used for development of the circuitry, was *MultiSim* described in Section 2.6. Every circuit that were tested were made in MultiSIM first. MultiSIM provides a simple solution on prototyping circuits quickly. In addition, the circuits can be tested before implementation in real life. This eliminates some sources of error that can be encountered when connecting the circuits physically. Some examples are short-circuiting, breaks in the wires and faulty connections. All of which are a common occurrence when prototyping on a breadboard.

Another key benefit of using MultiSim, is the ability to change resistance continuously. It's possible to monitor both the voltage and current in real time anywhere in the circuit. Therefore, it's easy to adjust the resistance to reach the desired voltage and current of the circuit. This was done in all of the circuits stated in Section 5. One problem with using MultiSIM to calculate resistance though, is that the program simulates with ideal conditions. Physical parts are never perfect. There were instances that the resistance had to be lowered compared to MultiSIM due to this problem. Although small adjustments were done accordingly.

### 7.5 CONFIG\_KVM

When adjusting the menuconfig for the real time kernel for the Raspberry Pi, there was a problem adding Fully pre-emptible kernel for the ARM64 processor. In the `config` file for the patch, a change needed to be done to allow the user to choose a fully pre-emptible kernel. To fix this problem, the config `CONFIG_KVM=y` needed to be changed to `CONFIG_KVM=n`. The fix was implemented by inspiration from this blogpost [29].

`CONFIG_KVM` is the configuration of the Kernel Virtual Machine virtualization support. KVM is an open-source software tool which allows running a virtual machine on unmodified Linux and Windows versions. Atleast on the Linux versions that were tested (5.4 and 5.15.65) this had to be disabled for a fully preemptible kernel to function. This was done on two separate Raspberry Pis. The fully preemptible kernel were built on 3 different systems. One PC booted with Linux Ubuntu 22.04 Jammy and two Raspberry Pis. The Ubuntu PC did not require to disable `CONFIG_KVM` to work, but it was necessary on both Raspberry Pis.

A reasonable explanation why KVM needs to be disabled while running a real-time kernel might be

---

because of the limited processing power of the ARM64 processor and small amounts of storage usually found in Raspberry Pis. This choice limits the storage taken and processing power needed to run the OS on the Raspberry Pis. This might be the reason the developers of the RT patch chose to make the choice of building the kernel with both options enabled unavailable for the ARM64 architectures.

## 7.6 The final results

The prototype developed as a part of this thesis is fully functional and fulfills the main objective of the task. The final solution was also showed in the course TPK4125 Mechatronics, which is recommended previous knowledge for TPK4128 Industrial Mechatronics. The students in this class showed interest in the prototype, and was positive to incorporate this as an assignment in Industrial Mechatronics, which several of them are likely to take the next semester.

With these results it was found more confirmation of *Sunds* findings, that the students prefers education with more practical elements. As well as the ability to apply the theory learnt in classes in practical assignments.

## 7.7 Further work and possible expansions

### 7.7.1 Simulation

Simulation of the factory is a possibility for further work. The possibility for a simulation expansion has already been explored as a part of the existing project assignment. Although, this part of the project was not successful, due to time constraints and obstacles found on the way. Some ways to overcome these obstacles has been provided in section 6.6. This topic provides a lot of opportunities for future project assignments/master thesis' and topics to explore further. One of these expansions could also be implementing a digital twin of the factory.

### 7.7.2 PCB design

Another part of the project with room for improvement is the PCB design. One possible update is to add silkscreen to the 40pin connector. This will make it easier to connect other microcontrollers with only the PCB, and not having to read it in the documentation and count pins. It is a bit easier to wire the cables with the PCB, than it was directly on the 40pin connector of the RPi. Since the pins on the PCB connector are color coded.

Another addition to the PCB that can be considered, is toggle switches to control the machining stations. This solution can make them easier to control, compared to using separate software on the RPi. Understanding the functionality is also easier with this solution. It can be implemented directly on the PCB, or on a daughterboard. The reason for using toggle switches instead of dip switches, mentioned in Section 5.14, is that surface mount dip switch packets cannot withstand the continuous use.

There is also a possibility to create another design that uses surface mount components instead. This can be a good solution if there is a need for many boards, since the PCBs can be designed to

---

be smaller. They can also be produced with the components already soldered on. This is a good solution in the example of; if a professor needs 40 of the boards to use in his/hers subject.

### **7.7.3 Machine learning**

Machine learning and camera vision are modules that can be added to the assembly as future expansions. There are existing camera modules that can be added onto a Raspberry Pi, on which OpenCV can be used to recognise different objects that travels on the conveyors. The machining modules are already run by a separate program written in *Python*. This program can be replaced with a program running image recognition to decide what operations should be done on the part. For this task, it is possible to send different objects on the conveyors, and the program decides if the part is going to be milled, drilled, both, or none, based on the shape, color or other distinct characteristics.

### **7.7.4 Robot arm and ROS2**

Another big part of the Industrial Robotics course is to learn the basics of ROS and robot control. There are also current assignments in the course related to these topics. ROS2 can also be implemented in an assignment using the mini-factory. One way to do this is to implement a small robot arm to put objects on the conveyor belts. This is a great opportunity to possibly automate a currently manual operation.

---

## 8 Conclusion

All of the main objectives from Section 1.3 has been met. In addition, some extensions and quality-of-life improvements were implemented successfully. These were not critical for the project, but will ease the process of turning this thesis into future exercises for Industrial Mechatronics.

Some secondary objectives were also met. An expandable solution was implemented successfully with the additional vacant pins of the PCB easy to connect to, and with the ability to run separate code to control the machining stations.

With this setup, it is possible to run the mini-factory with several types of microcontrollers. An Arduino Uno was used to run the first corner, but another controller would have been needed to run the factory in its completeness.

The excitement from students during the demonstration shows the interest in using practical assignments in education. This confirms the findings in *Sunds* Master's thesis, which is the basis for why this project was started.

All in all, the project gave a working prototype, that can with small adjustments be used as an assignment in Industrial Mechatronics.

---

## Bibliography

- [1] A. Skavhaug, *Tpk4128 - industrial mechatronics*, 2022. [Online]. Available: <https://www.ntnu.edu/studies/courses/TPK4128>.
- [2] F. GmbH, *Indexed line with two machining stations 24v*, 2022. [Online]. Available: <https://www.fischertechnik.de/en/products/learning/training-models/96790-edu-indexed-line-with-two-machining-stations-24v>.
- [3] A. K. Sund, ‘Improving mechatronics education with design thinking’, 2022.
- [4] A. C. Arnholm and M. N. Henriksen, ‘Combining industry 4.0 and 5g connectivity with robots in digital production factories’, 2021.
- [5] W. Bolton, *Programmable logic controllers*. Newnes, 2015.
- [6] Wikipedia, *File:originating register, number five crossbar switching system (museum of communications, seattle).jpg*, 2007. [Online]. Available: [https://en.wikipedia.org/wiki/File:Originating\\_Register,\\_Number\\_Five\\_Crossbar\\_Switching\\_System\\_\(Museum\\_of\\_Communications,\\_Seattle\).jpg](https://en.wikipedia.org/wiki/File:Originating_Register,_Number_Five_Crossbar_Switching_System_(Museum_of_Communications,_Seattle).jpg).
- [7] M. G. Hudedmani, R. Umayal, S. K. Kabberalli and R. Hittalamani, ‘Programmable logic controller (plc) in automation’, *Advanced Journal of Graduate Research*, vol. 2, no. 1, pp. 37–45, 2017.
- [8] E. R. Alphonsus and M. O. Abdullah, ‘A review on the applications of programmable logic controllers (plcs)’, *Renewable and Sustainable Energy Reviews*, vol. 60, pp. 1185–1205, 2016.
- [9] G. Frey and L. Litz, ‘Formal methods in plc programming’, in *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0, vol. 4, 2000, 2431–2436 vol.4. DOI: 10.1109/ICSMC.2000.884356*.
- [10] Standard-Norge, *Nek iec 61131-3:2013*, 2013. [Online]. Available: <https://www.standard.no/no/Nettbutikk/produktkatalogen/Produktpresentasjon/?ProductID=627454>.
- [11] IEC, ‘Programmable controllers – part 1: General information’, en, International Electrotechnical Commission, Geneva, CH, Standard IEC 61131-1:2003, 2003. [Online]. Available: <https://webstore.iec.ch/publication/4550>.
- [12] IEC, ‘Programmable controllers – part 3: Programming languages’, en, International Electrotechnical Commission, Geneva, CH, Standard IEC 61131-3:2013, 2013. [Online]. Available: <https://webstore.iec.ch/publication/31007>.
- [13] IEC, ‘Industrial-process measurement and control – programmable controllers – part 2: Equipment requirements and tests’, en, International Electrotechnical Commission, Geneva, CH, Standard IEC 61131-2:2017, 2017. [Online]. Available: <https://webstore.iec.ch/publication/31007>.
- [14] Redhat.com, *Working with the real-time kernel for red hat enterprise linux*, 2022. [Online]. Available: <https://www.redhat.com/sysadmin/real-time-kernel>.
- [15] Raspberry-Pi-foundation, *Raspberrypi.com*, 2022. [Online]. Available: <https://www.raspberrypi.com/>.
- [16] thiagoralves, *Tpk4128 - industrial mechatronics*, 2022. [Online]. Available: [https://github.com/thiagoralves/OpenPLC\\_v3](https://github.com/thiagoralves/OpenPLC_v3).

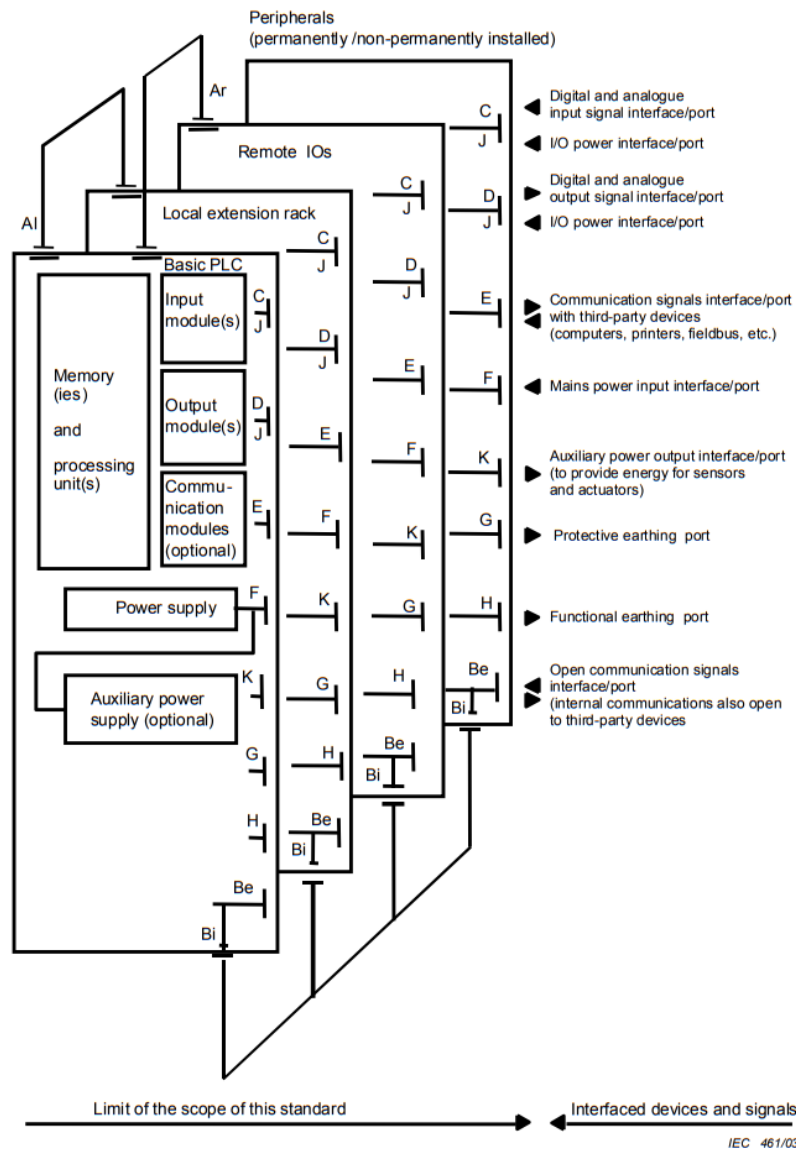
- 
- [17] V. Components, *Visual components*, 2022. [Online]. Available: <https://www.visualcomponents.com/about-us/>.
- [18] A. Wordpress, *Connecting raspberry pi to eduroam*, 2016. [Online]. Available: <https://autottblog.wordpress.com/raspberry-pi-arduino/connecting-raspberry-pi-to-eduroam/>.
- [19] OpenPLC, *1.4 installing openplc runtime on linux*, 2022. [Online]. Available: <https://openplcproject.com/docs/installing-openplc-runtime-on-linux-systems/>.
- [20] F. GmbH, *Technical faq's*, 2022. [Online]. Available: <https://www.fischertechnik.de/en/service/faq/technical-faqs>.
- [21] *Simatic s7-1500 signal modules*, 2022. [Online]. Available: <https://new.siemens.com/global/en/products/automation/systems/industrial/plc/simatic-s7-1500/signal-modules.html>.
- [22] PCBWay, *Pcbway, pcb prototype the easy way*, 2022. [Online]. Available: <https://www.pcbway.com/>.
- [23] OpenPLC, *Openplc editor download*, 2022. [Online]. Available: <https://openplcproject.com/download/>.
- [24] OpenPLC, *Openplc runtime overview*, 2022. [Online]. Available: <https://openplcproject.com/docs/2-1-openplc-runtime-overview/>.
- [25] OpenPLC, *2.4 physical addressing*, 2022. [Online]. Available: <https://openplcproject.com/docs/2-4-physical-addressing/>.
- [26] S. B. Reddy, *Instrumentation tools*, 2022. [Online]. Available: <https://instrumentationtools.com/what-is-sequential-function-chart-sfc/>.
- [27] thiagoralves, *Openplc forum*, 2022. [Online]. Available: <https://openplc.discussion.community/post/how-to-make-openplc-support-opc-ua-12342333>.
- [28] KiCAD, *Kicad eda. a cross platform and open source electronics design automation suite*, 2022. [Online]. Available: <https://www.kicad.org/>.
- [29] D. Zundel, *Testing preempt rt on the i.mx8mm soc in 15 minutes*, 2022. [Online]. Available: <https://blog.lazy-evaluation.net/posts/embedded/imx8mm-rt-preempt.html#>.



---

## Appendix

## A PLC architecture from IEC 61131-2



### Key

- AI Communication interface/port for local I/O
- Ar Communication interface/port for remote I/O station
- Be Open-communication interface/port also open to third-party devices (for example, personal computer used for programming instead of a PADT)
- Bi Internal communication interface/port for peripherals
- C Interface/port for digital and analogue input signals
- D Interface/port for digital and analogue output signals
- E Serial or parallel communication interfaces/ports for data communication with third-party devices
- F Mains power interface/port. Devices with F ports have requirements on keeping downstream devices intelligent during power-up, power-down and power interruptions.
- G Port for protective earthing
- H Port for functional earthing
- J I/O power interface/port used to power sensors and actuators
- K Auxiliary power output interface/port

Figure 43: Typical interface/port diagram of a PLC-system (from IEC 61131-2)[13]

---

## B Eduroam Guide

This Guide is exactly the same as the one from [18], but is added to make it possible to recreate our testing with only reading this document. It is tested to work on Eduroam at NTNU's campus Gløshugen as of December 2022.

First, we must add a few lines of text in the file `/etc/wpa_supplicant/wpa_supplicant.conf` (it must be done with root permission):

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Then append the following lines (yes, change username and password to something appropriate)

```
network={
identity="username@ntnu.no"
password="password"
eap=PEAP
phase1="peaplabel=0"
phase2="auth=MSCHAPV2"
priority=999
disabled=0
ssid="eduroam"
scan_ssid=0
mode=0
auth_alg=OPEN
proto=RSN
pairwise=CCMP
key_mgmt=WPA-EAP
proactive_key_caching=1
}
```

Then hit `< control > +x`, then `y` and `< enter >` to save and exit.

Depending on your version of Pi and your Pi's operating system you might or might not have a connection now (check with `ifconfig`). If you do not, you should try to stop networking and start `wpa_supplicant`:

```
sudo service networking stop
```

```
sudo wpa_supplicant -i wlan0 -c
↳ /etc/wpa_supplicant/wpa_supplicant.conf -B
```

If you still don't have a connection you should try to reboot

---

```
sudo reboot
```

Still no connection? Check that all of the special characters in `/etc/wpa_supplicant/wpa_supplicant.conf` have been copied correctly, for example

```
' '
```

is not the same as

```
" "
```

Is Eduroam being stubborn? You can do as we did, search around a bit and try tweaking the settings in `/etc/wpa_supplicant/wpa_supplicant.conf` until you win.

---

## C Milling, Drilling or not code Example

This is the the code we used to control the mill and drill. It is written fast to make it possible to choose which states is desired without having to re run the program each time. This can be written more efficiently, but that was not the purpose of this code. It should just work, which it does.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)

mill = 3
drill = 5

GPIO.setup(mill, GPIO.OUT)
GPIO.setup(drill, GPIO.OUT)

m = ''
d = ''
cont = ''

while True:
    while True:
        m = input('Type y for milling and n for not milling: ')
        if m == 'y':
            break
        elif m == 'n':
            break

    while True:
        d = input('Type y for dilling and n for not dilling: ')
        if d == 'y':
            break
        elif d == 'n':
            break

    if m == 'y':
        GPIO.output(mill, GPIO.HIGH)
        print('m->high')
    else:
        GPIO.output(mill, GPIO.LOW)
        print('m->low')

    if d == 'y':
        GPIO.output(drill, GPIO.HIGH)
```

---

```
    print('d->high')
else:
    GPIO.output(drill, GPIO.LOW)
    print('d->low')

while True:
    cont = input('Type y for continue and n for quitting: ')
    if cont == 'y':
        break
    elif cont == 'n':
        break
if cont == 'n':
    break
```

---

## **D Attachments**

### **D.1 Attached files:**

- The Ladder Diagram PLC code
- The SFC PLC code
- The PCB files for design version 3
- Visual Components files

### **D.2 Hardware:**

The mini-factory with the prototype.

---

## E Extended Raspberry Pi OpenPLC table



Pin	Name	OpenPLC address	PCB functionality	OpenPLC address	PCB functionality	OpenPLC address	Name	Pin
1	3.3V DC Power	-	3.3V	□	●	-	5V DC Power	2
3	GPIO02 (SDA1, I2C)	%IX0.0 *	Milling?!!	●	●	-	5V DC Power	4
5	GPIO03 (SDL1, I2C)	%IX0.1 *	Drilling?!!	●	●	-	Ground	6
7	GPIO04 (GPCLK0)	%IX0.2	Btn_S1_front	●	●	%QX0.0	GPIO14 (TXD0, UART)	8
9	Ground	-	GND1	●	●	%QX0.1	GPIO15 (RXD0, UART)	10
11	GPIO17	%IX0.3	Btn_S1_rear	●	●	%QW0	GPIO18 (PWM0)	12
13	GPIO27	%IX0.4	Btn_S2_front	●	●	-	Ground	14
15	GPIO22	%IX0.5	Btn_S2_rear	●	●	%QX0.2	GPIO23	16
17	3.3V DC Power	-	3.3V	●	●	%QX0.3	GPIO24	18
19	GPIO10 (SP10_MOSI)	%IX0.6	Sensor2_conv1	●	●	-	Ground	20
21	GPIO09 (SP10_MISO)	%IX0.7	Sensor3_conv2	●	●	%QX0.4	GPIO25	22
23	GPIO11 (SP10_CLK)	%IX1.0	Sensor1_Start	●	●	%QX0.5	GPIO08 (SPI0_CE0_N)	24
25	Ground	-	GND1	●	●	%QX0.6	GPIO07 (SPI0_CE1_N)	26
27	GPIO00 (SDA0, I2C)	-	Pin27 pass through	●	●	-	GPIO07 (SDL0, I2C)	28
29	GPIO05	%IX1.1	Sensor4_conv3	●	●	-	Ground	30
31	GPIO06	%IX1.2	Sensor5_conv4	●	●	%QX0.7	GPIO12 (PWM0)	32
33	GPIO13 (PWM1)	%IX1.3	I_Mill	●	●	-	Ground	34
35	GPIO19	%IX1.4	I_Drill	●	●	%QX1.0	GPIO16	36
37	GPIO26	%IX1.5	Pin37 pass through	●	●	%QX1.1	GPIO20	38
39	Ground	-5	GND1	●	●	%QX1.2	GPIO21	40

## RISK ASSESSMENT FOR: Mini-factory testing

*(Name of the project is entered in the sheet "Risk assessment")*

An approved risk assessment implies that you have received authorization to carry out the activities described in this document. If there are any changes to your activities, you must update your risk assessment and have the update approved before you can proceed with the new work and/or any new equipment.

An approved risk assessment grants you access to the laboratory, provided that you have also completed and documented the online HSE course and the HSE facility tour. This access is exclusive to you; you should not allow others to enter.

To access the equipment, specific equipment training must be completed. Your supervisor will provide you with information about the rules that apply to the laboratory you are gaining access to. Note that different laboratories have different rules.

The risk assessment is converted to PDF before signing (done by you). The template is prepared for printing in landscape A4 format (do not alter the formatting).

### Identification:

Your name:	Lars Bonvik
Your institute:	MTP
Your phone number:	91361010
Your e-mail:	larsbon@ntnu.no
Name of your supervisor:	Amund Skavhaug
Project number:	985512113
Laboratory(ies) you are requesting access to:	Ubåten + Flex
Period you want access, from-to:	28.08 - 23.12 2023

Completed HSE Online course and HSE Room tour:  Yes

Phd:   
 Bachelor:   
 Post-Doc:   
 Sintef:   
 Other:  Master

### Signatures, approved by:

	Date:	Name:	Signature:
Applicant:	06/09 - 2023	Lars Bonvik	<i>Lars Bonvik</i>
Supervisor:	06/09 - 2023	Amund Skavhaug	<i>Amund Skavhaug</i>
Equipment training:			
Room responsible:	07.09.23	Håvard Vestad	<i>Håvard Vestad</i>
Lab leader:	11.09.2023	Arve Skorstad	<i>Arve Skorstad</i>

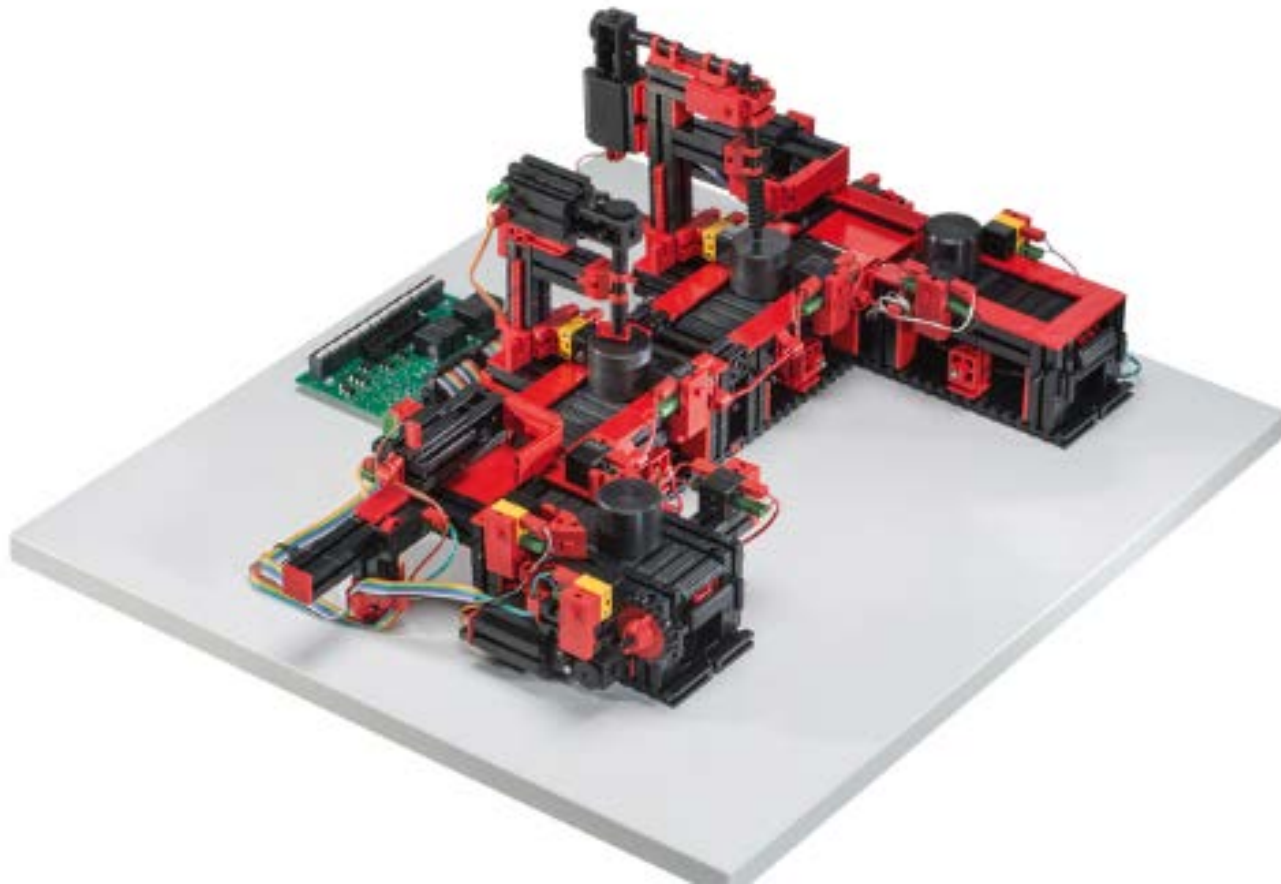
### **Description of project**

**Provide a comprehensive description of your project, making it detailed enough for the reader to understand what activities you intend to do, and (briefly) to what purpose. Include details about the instruments you plan to use, specifying the names of the instruments.**

Write here (line breaks = Alt+Enter):

I am running a model factory using an Arduino PLC and other micro controllers. I wanna do prototyping to make embedded solutions to be able to get the micro controllers to run the 24V system. Therefore I'm gonna work some with breadboards making circuits. Cutting and fitting wires, measure voltages and other stuff related to circuit prototyping. I might also use the soldering iron, but I'm not 100% I'll need it yet.

Visual depiction- (Insert a photo/illustration/drawing of your project):



## RISK ANALYSIS

<b>Unit / Department:</b> Department of Mechanical and Industrial Engineering (MTP)	<b>Areas of consequence:</b> Human health, Environment, Material values, and Reputation	<b>Project name:</b> <b>Mini-factory testing</b>
--	--	---

<b>Responsible supervisor:</b> Amund Skavhaug
<b>Candidate:</b> Lars Bonvik
<b>Date of evaluation</b> 29.08.2023

<b>Existing risk-reducing measures</b>
Online HSE course
Personal risk assessment
HSE room tour
Machine card
PPE - Personal Protective Equipment
Equipment - Emergency stop/guarding, etc.









Dangers to consider	Relevance: not relevant(nr) or yes
Loud noise:	nr
Hazardous dust:	nr
Pinch hazard:	nr
Falling:	nr
Cuts:	nr
Tripping:	nr
Parts with high velocity:	nr
Rotating parts:	nr
Vibration:	nr
Fire:	nr

Dangers to consider	Relevance: not relevant(nr) or yes
Burns:	Ja
Poisoning:	nr
Radiation:	nr
Work at heights:	nr
High / low pressure:	nr
High / low temperatures:	nr
Electric shock:	Ja
	nr
	nr
	nr

Nr.	Situation	Possible unwanted event	Prevention measure/ PPE	Propability (P)	Consequence (C) (0-5)				Risk value (P x C)			
				(1-5)	Human health	Environment	Material values	Reputation	Human health	Environment	Material values	Reputation
1	Bruk av loddebolt	Kan få brannskader om man tar på enden av loddebolten. Inhalering av avgasser.	Bruk av avtrekk og tidsur på loddeboltene. Vite hvor nærmeste vask er sånn at man kan raskt avkjøle en eventuell skade. Vite hvor brannslukkeren er. Pass på å ha avtrekket nærmere for å ikke inhalere avgassen	2	2	1	2	0	4	2	4	0
2	Bruk av elektriske håndverktøy	Kan få små kuttskader eller sprut av små biter som kan komme i øynene	Vernebriller	2	2	0	0	1	4	0	0	2
3	Jobbe med strøm (24v)	Kan gi elektrisk sjokk, eller generere mye varme om det skjer en kortslutning	Ikke lettantennelig arbeidsplass	1	2	1	1	1	2	1	1	1
4									NR	NR	NR	NR
5									NR	NR	NR	NR
6									NR	NR	NR	NR
7									NR	NR	NR	NR
8									NR	NR	NR	NR
9									NR	NR	NR	NR
10									NR	NR	NR	NR
11									NR	NR	NR	NR
12									NR	NR	NR	NR
13									NR	NR	NR	NR
14									NR	NR	NR	NR
15									NR	NR	NR	NR
16									NR	NR	NR	NR
17									NR	NR	NR	NR
18									NR	NR	NR	NR
19									NR	NR	NR	NR
20									NR	NR	NR	NR
21									NR	NR	NR	NR
22									NR	NR	NR	NR

## RISK ANALYSIS FOR CHEMICALS

Fill in 'yes' if these hazards apply to your work.

Dangers to consider	Relevance: not relevant(nr) or yes	Dangers to consider	Relevance: not relevant(nr) or yes
 Explosive	<i>nr</i>	 Oxidising	<i>nr</i>
 Hazardous to the environment	<i>nr</i>	 Acute toxicity	<i>nr</i>
 Serious health hazard	<i>nr</i>	 Gas under pressure	<i>nr</i>
 Corrosive	<i>nr</i>	Hazardous fumes	<i>nr</i>
 Health hazard / Hazardous to the ozone layer	<i>nr</i>	Disposal (Hazardous waste)	<i>nr</i>

Information about the chemicals you will use is available in the Safety Data Sheets (SDS) in EcoOnline.

List the chemicals you intend to use and make an estimate of the quantity you need. Ethanol and acetone for general cleaning of equipment does not need to be listed here.

Name of chemical	Estimated use (g/kg/mL/L/etc.)	Concentration (if relevant)
<i>Superlim</i>	10 mL	
<i>Flussmiddel</i>	50 mL	

If two or more chemicals are to be mixed, describe the procedure you will perform here (leave blank if not relevant):

(Line break: Alt+Enter)

Plan for chemical waste disposal:

Not relevant

**RISK ANALYSIS**

Nr.	Chemical name	H-number and H-sentence	Prevention measure/ PPE	Propability (P)	Consequence (C) (0-5)				Risk value (P x C)			
				(1-5)	Human health	Environment	Material values	Reputation	Human health	Environment	Material values	Reputation
1	Kjemikalie til lodding (flusmiddel)	H319: Gir alvorlig øye-irritasjon	Avtrekk med "skjold", og muligens vernebriller	1	2	1	0	0	2	1	0	0
2	Kjemikalie håndtering (Superlim)	H315: Irriterer huden	Ha tørkepapir tilgjengelig	3	1	1	0	0	3	3	0	0
3	Kjemikalie håndtering (Superlim)	H319: Gir alvorlig øye-irritasjon, osen kan være irriterende, og få det direkte på øynene kan være veldig irriterende	Åpent ventilert område, med tilgjengelige vernebriller og se hvor øyeskyllestasjonen er	1	3	0	0	0	3	0	0	0
4	Kjemikalie håndtering (Superlim)	H335: Kan forårsake irritasjon av luftveiene	Brukes i åpent ventilert område	1	2	0	0	0	2	0	0	0
5									NR	NR	NR	NR
6									NR	NR	NR	NR
7									NR	NR	NR	NR
8									NR	NR	NR	NR
9									NR	NR	NR	NR
10									NR	NR	NR	NR
11									NR	NR	NR	NR
12									NR	NR	NR	NR
13									NR	NR	NR	NR
14									NR	NR	NR	NR
15									NR	NR	NR	NR
16									NR	NR	NR	NR
17									NR	NR	NR	NR
18									NR	NR	NR	NR
19									NR	NR	NR	NR
20									NR	NR	NR	NR
21									NR	NR	NR	NR
22									NR	NR	NR	NR
23									NR	NR	NR	NR
24									NR	NR	NR	NR
25									NR	NR	NR	NR
26									NR	NR	NR	NR
27									NR	NR	NR	NR

## RISK ASSESSMENT FOR: Mini-factory testing

### WHAT IS A RISK ANALYSIS, AND WHAT IS MEANT BY RISK?

Risk analysis is a systematic approach to describing and/or calculating risk. It involves mapping out unwanted events, their causes, and consequences. Risk refers to the possibility of undesired events occurring and the potential consequences they can have on people, material assets, the environment, and the reputation of the department or university.

Risk is expressed by the probability of, and the consequence of, the undesired events.

$$RISK = PROBABILITY LEVEL \times CONSEQUENCE LEVEL$$

A risk analysis consists of asking four simple questions regarding an activity:

1. What can go wrong?
2. What is the probability of this going wrong?
3. What is the consequence if it does happen?
4. What can we do to prevent something from going wrong or to reduce the consequence if something still happens?

Two tabs in this document contain informative guidance on what criteria to consider – a form with probability and consequence, along with an overview showing various criteria available, including types of protective equipment, hazards to assess, and already established measures. The list is not exhaustive; you're allowed to think beyond what's listed.

### Tabs with a blue color should be filled out:

Front page:	Fill in the necessary information. If you don't know who the room manager is, leave the field blank. All fields under identification should be completed.
Description of project:	Please do it thoroughly and detailed; this will make it easier to assess your risk assessment, and it will speed up the access process. <i>Example of not approved: Grinding of materials.</i> <i>Example of approved: In this lab, I will use a manual grinding machine to polish around 20 steel discs. The goal is to prepare the samples for friction testing.</i>
Picture:	Select an image that can describe what you are going to do, or draw a sketch. If this is not relevant to your work, explain why.
Risk assessment:	The sheet is formatted so that you can fill in the necessary fields. Here, you should enter all the risks associated with the activities of your work, except for hazards from chemicals (which are assessed separately). In the "situation" columns, you describe the hazard to be assessed, and then evaluate the hazardous factors. You indicate the type of hazard by marking "yes" or "nr" (not relevant) in the table at the top right. In the risk assessment table, you specify the situation as "Pinch hazard" if you have marked this as a relevant hazard for your work, and so on. Write detailed enough that it is easily understood for the reader when/where/how the danger arises.



	<b>Health</b>	<b>Material values</b>	<b>Reputation</b>	<b>Environment</b>
<b>Grade</b> 1	Minor injury/strain that requires simple treatment. Reversible injury. Short recovery time.	Operational shutdown, or shutdown of activities <1 day.	Little effect on credibility and respect.	Negligible injury and short recovery time.
2	Injury/strain that requires medical treatment. Reversible injury/strain. Short recovery time.	Operational shutdown, or shutdown of activities <1 week.	Negative effect on credibility and respect.	Minor injury and short recovery time.
3	Serious injury/strain that requires medical treatment. Lengthy recovery time.	Operational shutdown, or shutdown of activities <1 month.	Reduced credibility and respect.	Minor injury and lengthy recovery time.
4	Serious injury/strain that requires medical treatment. Possible disability /permanent disability.	Operational shutdown > 1/2 year. Shutdown of activities up to 1 year.	Credibility and respect considerably reduced.	Long-lasting injury. Lengthy recovery time.
5	Death or disability / permanent disability.	Operational shutdown, or shutdown of activities >1 year.	Credibility and respect considerably and permanently reduced.	Very long-lasting and irreversible injury.

<b>Consequence (C)</b>	<b>Very serious</b>	5	10	15	20	25
	<b>Serious</b>	4	8	12	16	20
	<b>Moderate</b>	3	6	9	12	15
	<b>Little</b>	2	4	6	8	10
	<b>Very little</b>	1	2	3	4	5
		<b>Very little</b>	<b>Little</b>	<b>Medium</b>	<b>Big</b>	<b>Very big</b>
<b>Probability (P)</b>						

<b>Red</b>	Unacceptable risk. Measures need to be implemented.
<b>Yellow</b>	Medium risk. Measures need to be considered.
<b>Green</b>	Acceptable risk. Measures can be considered.

# Risikostyring

## Safety Measures Overview

### PPE - Personal Protective Equipment

- Protective glasses
- Lab coat
- Gloves
- Safety shoes
- Hearing protection

### Alternative measures

- Reduce scale of experiment
- Eliminate experiment
- Substitute for chemicals
- Adjust experiment for HSE

### Ventilation and encapsulation

- Fume arm
- Fumehood
- Oven/refrigerator
- Room

### Safety barriers and emergency equipment

- Emergency shower and eye wash
- First aid equipment
- Fire extinguishers and fire blankets
- Communication system
- PPE - Personal Protective Equipment (Eyes, head, hands, feet, and body)
- Safety data sheet
- Procedures

### Established measures

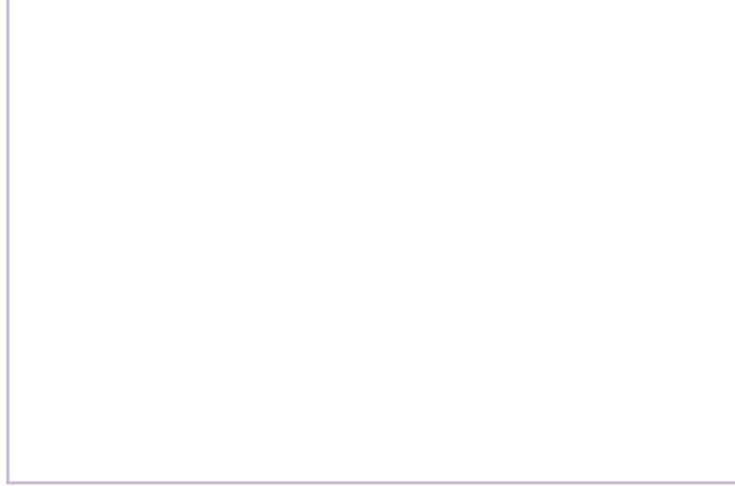
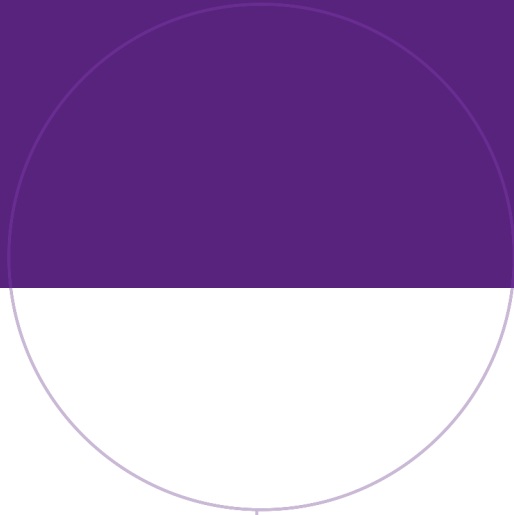
- "Experiment in progress" signs
- Experiment procedures and routines
- Labeling containers with contents
- Buddy check
- Equipment card
- Barrier equipment and screens
- Approved lifting equipment
- Safety data sheet
- Chemical database - Eco Online
- Working alone - alarm/prohibition
- First aid equipment
- Emergency shower
- Eye wash station
- Fire extinguishers
- Fire alarms connected to the fire department
- Room access card

### Dangers to consider

- Loud noise
- Hazardous dust
- Pinch hazard
- Falling
- Cuts
- Tripping
- Parts with high velocity
- Rotating parts
- Vibration
- Fire

### Dangers to consider

- Burns
- Poisoning
- Radiation
- Biological/infectious
- High / Low pressure
- High / Low temperature
- Electric shock
- Work at height



Norwegian University of  
Science and Technology