Low-cost Industrial Controller based on the Raspberry Pi platform

Gustavo Mendonça de Morais Rabelo Vieira

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Industrial Engineering.

Work oriented by:

Professor PhD Paulo Leitão

Professor PhD José Barbosa

Professor MsC Ulisses da Graça

Bragança

2018-2019

Low-cost Industrial Controller based on the Raspberry Pi platform

Gustavo Mendonça de Morais Rabelo Vieira

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Industrial Engineering.

Work oriented by:

Professor PhD Paulo Leitão

Professor PhD José Barbosa

Professor MsC Ulisses da Graça

Bragança

2018-2019

Dedication

Este trabalho é dedicado a mulher mais trabalhadora que conheço, a quem admiro muito e me inspira imensamente desde criança, minha querida e amada avó Terezinha Mendonça de Morais, singularmente importante em minha vida.

À minha mãe Janaína, à minha tia Gildete, melhores amigas que sempre me ofereceram apoio a perseverar em minhas conquistas.

Aos meus queridos irmãos mais novos, Lucas e Vitor que me orgulham pelas pessoas que estão se tornando e por toda a companhia que tive o prazer de desfrutar durante o nosso crescimento.

Acknowledgements

Agradeço a todo o apoio e suporte que tive da minha família por me proporcionar a oportunidade de chegar até aqui. Aos meus professores orientadores José Barbosa e Paulo Leitão pela rica e experiente orientação. A todos os amigos, colegas e professores que foram sumariamente importantes para o sucesso da realização deste trabalho.

Abstract

The low-cost automation field exhibits the need of innovation both in terms of hardware and software. There is a lack of devices that allow the development of control logic that is free from restrictions of domain-specific communication platforms and at the same time able to provide the capabilities aligned to the Industry 4.0 requirements.

The objective of this work is to develop an inexpensive, small Industrial Controller that supports the execution of programs in different industrial programming languages. So, it is intended to develop, manufacture and control a low-cost but powerful Industrial Controller based on the use of the single-board computer Raspberry Pi.

The study described in this document was carried out on the creation of a hardware platform that is capable of integrating with software frameworks compatible with standards updated and widely used in the industrial automation field. IEC 61131-3 is employed displaying the ease of use and implementation alongside multiple well established programming languages for automation through the OpenPLC platform while IEC 61449 is employed through the 4DIAC framework that has a clear and objective environment capable of providing the appropriate tools for implementation of a distributed control.

It is employed the single-board computer Raspberry Pi, a robust device with adequate processing power and communication capability for the elaboration of a platform in the low-cost automation scope. The elaboration of the Industrial Shields, responsible for providing the controller I/O interface took into consideration maintenance concerns of controller integrity through the application of galvanic isolation in the automaton input and output sections.

The proposed platform was successfully tested in an automation system prototype

comprising Fischertechnik's Punching Machine being possible to develop the control logic using IEC 61131-3 and IEC 61499.

Keywords: cyber-physical systems; I40; low-cost automation; micro PLC; IIoT; event-driven approach.

Resumo

O campo de automação de baixo custo demonstra a necessidade de inovação em termos de hardware e software. Há uma falta de dispositivos que permitem o desenvolvimento de lógica de controle livre de restrições de plataformas de comunicação específicas de domínio e, ao mesmo tempo, capazes de fornecer os recursos alinhados aos requisitos da Indústria 4.0.

O objetivo deste trabalho é desenvolver um pequeno Controlador Industrial de baixo custo que suporte a execução de programas em diferentes linguagens de programação industrial. Assim, pretende-se desenvolver, fabricar e controlar um Controlador Industrial de baixo custo, mas poderoso, baseado no uso do computador de placa única Raspberry Pi.

O estudo descrito neste documento foi realizado na criação de uma plataforma de hardware que é capaz de se integrar com frameworks de software compatíveis com padrões atualizados e amplamente utilizados na área de automação industrial. A IEC 61131-3 é empregada exibindo a facilidade de uso e implementação juntamente com várias linguagens de programação bem estabelecidas para automação através da plataforma OpenPLC enquanto a IEC 61449 é empregada através da estrutura 4DIAC que possui um ambiente claro e objetivo capaz de fornecer as ferramentas apropriadas para implementação de um controle distribuído.

É empregado o computador de placa única Raspberry Pi, um dispositivo robusto com capacidade de processamento e capacidade de comunicação adequados para a elaboração de uma plataforma no escopo de automação de baixo custo. A elaboração dos industrial shields, responsáveis por fornecer a interface de I/O do controlador levou em consideração

as preocupações de manutenção da integridade do controlador através da aplicação de isolação galvânica nas seções de entrada e saída do autômato.

A plataforma proposta foi testada com sucesso em um protótipo de sistema de automação compreendendo a Punching Machine da Fischertechnik sendo possível desenvolver a lógica de controle usando IEC 61131-3 e IEC 61499.

Palavras-chave: sistemas ciber-físicos; I40; automação de baixo custo; micro PLC; IIoT; abordagem orientada a eventos.

Contents

List of Tables

List of Figures

Acronyms

4DIAC "For" Distributed Industrial Automation and Control. 30

ARP Address Resolution Protocol. 8

BCM Broadcom SOC channel. 40

BFBs Basic Function Blocks. 36

BIOS Basic Input Output System. 16

BMS Building Management Systems. 9

BOOL Boolean. 34

CFBs Composite Function Blocks. 36

CNC Computer Numerical Control. 26

CODESYS Controller Development System. 11

CPU Central Processing Unit. 7

CV Counter Value. 35

DC Direct Current. 17

DRC Design Rules Checker. 26

ECC Execution Control Chart. 37

EDA Electronic Design Automation. 23

EMC Electromagnetic Compatibility. 26

EMF Electromotive Force. 20

ET Elapsed Time. 35

FB Function Block. 30

FBDK Function Block Development Kit. 30

FBRT Function Block Runtime. 30

GNU General Public License. 8

GPIO General Purpose Input Output. 16

HMI Human-Machine Interface. 38

I/O Input / Output. 8

I40 Industry 4.0. 10

ICT Information and Communications Technology. 1

IDE Integrated development environment. 30

IEC International Electrotechnical Commission. 5

IIoT Industrial Internet of Things. 3

INT Integer. 35

IPv4 Internet Protocol version 4. 8

JADE Java Agent Development Framework. 57

LED Light Emitting Diode. 20

N-MOSFET Negative - Metal Oxide Semiconductor Field Effect Transistor. 20

NPN Negative-Positive-Negative. 20

OS Operational System. 16

PCBs Printed Circuit Boards. 21

PCs Programmable Controllers. 1

PLCs Programmable Logic Controllers. 1

POUs Program Organization Units. 31

PV Preset Value. 35

R Reset. 35

RAM Random Access Memory. 14

RTE Runtime Environment. 30

SCADA Supervisory Control and Data Acquisition. 57

SD Secure Digital. 16

SIFBs Service Interface Function Blocks. 36

SMDs Surface-Mount Devices. 21

SoC System on a Chip. 14

SR Set-Reset. 40

SSH Secure Shell. 16

SSID Service Set Identifier. 16

ST Structured Text. 31

TCP/IP Transmission Control Protocol / Internet Protocol. 8

THT Through-Hole Technology. 22

USB Universal Serial Bus. 14

Wi-Fi Wireless Fidelity. 15

XML eXtensible Markup Language. 31

Chapter 1

Introduction

The Third Industrial Revolution occurred in the late 1960s and was characterized by the introduction of robotics and Information and Communications Technology (ICT) in the industry. Programmable Logic Controllers (PLCs), known as only Programmable Controllers (PCs) at that time was the most distinctive element of that revolution. PLCs are dedicated computers responsible for controlling industrial machines. That automaton was widely used in the factories of the last century as well as still remains strongly present. Since the beginning, it solved major connectivity problems between control devices drastically reducing system inflexibility. The control structures of industrial systems before the Third Industrial Revolution were strictly tied to hardwired electromechanical devices.

Those previous control systems were entirely composed of relays that required an immense amount of wiring and unavoidable expenses associated with its inflexibility. Those circuits could not be rearranged in order to fit another application in the same field. Even the smallest change in the number or nature of inputs and outputs demanded a complete new circuitry that could then properly communicate. That electrical concept was propitious to malfunction over time and it hindered proper maintenance since there was neither any standardized troubleshooting error system nor a pattern proceeding to solve such problems. Well trained technicians were the best solution to put at work those relay controlled systems. These characteristics are related to high costs.

With the emergence of the PLC, there was then immense versatility in the way the

control circuits could then be configured. Capable of storing instructions, such as sequencing, timing, counting, arithmetic, data manipulation, and communication, to control industrial machines and processes, these logic controllers revolutionized the way the mass production chains would be up and running [1]. So many important features now commonly found in electronic factory devices, prove just how reliable and robust these systems have become. The replacement of the old electromechanical systems for PLCs has greatly reduced maintenance costs. These industrial machines are designed to be connected to many inputs and outputs by collecting data from process states at the same time they answer accordingly to the logic deployed.

Pioneers in the concept of modularity. The physical division between the different types of modules allows the PLC to expand according to the needs of the user. This feature is very interesting from the economic point of view. As the need for additional inputs and outputs increases, investment can be made gradually in new modules according to the requirement of scalability. Communication between modules and different devices connected to the PLC is an important feature directly linked to the capabilities of the automaton. Control systems were developed using proprietary communication protocols. Therefore, the communication is closed and restricted to the devices belonging to the same manufacturer. In the same manufacturing environment, when you have controllers from different manufacturers, intercommunication becomes a problem. The difficulty in connecting the devices interferes directly in the performance of the activity to be controlled, since without proper communication, the control system loses the ability to identify the tasks to be performed along the production chain. The development of a platform that offers open communication is fundamentally necessary for the advancement according to the new technological reality. Allow to any device integrated with the network to communicate seamlessly unobstructed by proprietary protocols is essential to achieve a highly integrated control environment that can be implemented unimpeded from the communicative perspective of the system.

The Fourth Industrial Revolution is surpassing problems that arose in the previous revolution. Industry 4.0 is addressing solutions to merge physical and digital spheres

more broadly. The inclusion and combination of different electronic assets are possible in a higher level that wasn't possible before the outbreak of the newest technologies in the communication field. Namely, the wireless networks enabling Industrial Internet of Things (IIoT) development, cheaper and more powerful processors allowing fast data management through accessible and smart machines leading to the creation of intelligent industrial cyber-physical systems.

To continue the development and improvements in Programmable Logic Controllers domain is needed an special attention to the low-cost PLC niche. Many industrial and commercial activities are incompatible with high-end PLCs budget charges. Such operations are perfectly consonant with the amount of investment demanded by these cheap PLCs and the degree of reliability it can provide. Alongside, the advent of open source platforms in the software as well as in the hardware scopes make the low-cost PLC a goal even more achievable.

The low-cost automation field is a strategic approach of engineered solutions deployed in order to reduce costs through existing resources on the market. The main goal is to increase productivity and flexibility while decreasing waste with simple and cost-effective automation. Low-cost automation can be regarded as cheap hardware but not necessarily with low performance. The system automation is cheap when compared to the flagship devices on the market however they are disposed to accomplish tasks in environments and activities where their capacity of processing is well designed to handle. It's an engineering field focused on solving the needs of the industry at the same time it takes care of the costs involved. Exploiting deeply the capabilities that the hardware offers, demands are reached by this economical-intelligent approach. This methodology is also closely linked to decentralization of control. Less powerful devices spread throughout the workshop although able to communicate with each other empowers the system controls' trustworthiness since a malfunction in the assembly line can be bypassed and faster identified and repaired. This feature improves the system flexibility since the process relies not only in one important-centered device. Furthermore, the modular architecture allows direct replacement of the inoperative equipment [2].

Reliability is an essential condition for automation systems to operate. Due to the mandatory high standards and concerns about expenses related to maintenance, the stability of continuous and precise operation of the system is a very important characteristic highly required in the shop floor. As the devices are becoming smarter due to the growing number of embedded sensors, the data collected about the operation of the device provide the necessary information to enable a precise and higher quality predictive maintenance. Such procedure reduces machine downtime allows proper intervention about the use of the equipment to be taken without disrupting the production process. Prognosis allows real-time analysis to monitor equipment performance without adding unnecessary over-processing to the machine. Such kind of intelligent just-in-time maintenance reduces the costs what reduces the payback time of the product at the same time it offers better performance and compliance to the behavior required in the industrial field.

The high density of connection between the electronic assets enables a highly promising concept that increasingly proves its usefulness. Known as the decentralization of control, independent components able to communicate with each other in order to make the best decisions based on what they know on their own and spread the ability to process is an approach of insightful and innovative control. Systems with these characteristics are more responsive due to the lower processing concentration in the same component. The difficulty lies in implementing an efficient algorithm while obeying common rules over the maintenance of high interoperability between devices. From the advent of electronic resources with high processing capacity added to the low cost, it becomes possible to develop physical applications of control both centralized and most desired, distributed. In this sense, some approaches to programming environments embedded in the developed physical device are tested.

It is aimed in this CPS environment to have distributed systems that need to use low-cost industrial controllers, powerful in computation, just like the Raspberry Pi platform.

1.1 Objective

The objetive of this work is to develop an Industrial Controller, that seeks to achieve an engineering solution with all the benefits that the PLC obtained in the Third Industrial Revolution, now in the era of high communication capabilities of the fourth generation of the industrial revolutions. By uniting all the pros of this controller technology and managing the already well-known cons effectively, a viable and highly upgraded solution is planned within the scope of cyber-physical systems. Focusing on the development of an industrial controller system based on the single-board computer Raspberry Pi, capable of providing at the same time a low cost and powerful platform for controlling automation systems, running applications developed in different languages by means of International Electrotechnical Commission (IEC) standards. They are IEC 61131-3 [3] and IEC 61499 [4]. The proposed approach was used to control a manufacturing plant with sufficient reliability for its applications.

1.2 Structure

The rest of the work is organized as follows. Chapter 2, entitled "Related Work", states the low-cost automation field and some products already on the market not so mature. Chapter 3 displays the PLC Architecture and the Industrial Shield developed on this work pointing out centralized and distributed approaches. Chapter 4 sets the programming environments utilized to develop and deploy all logic code used to control through the automaton. Chapter 5, entitled "Experimental Implementation", establishes the case study as a testbed employed in order to analyze the controller system behavior. Chapter 6 rounds up the application of this controller under rather complementary than opposite approaches and points out some future work. Finally, in appendix A is presented some important images for reference and in appendix B is presented a Python script that tests and confirms the usability of Raspberry's GPIOs as inputs and outputs.

Chapter 2

Related Work

In this chapter, hardware platforms in the context of low-cost PLCs that use the Raspberry Pi as main board or have its own custom open-source board with modernized communication capabilities are addressed. Since controllers with these characteristics are new implementations, the maturity of these systems are relatively little. These platforms were chosen due to their close proximity to the purpose of this work.

2.1 OpenPLC

The easy implementation of the automation system, little need for programming knowledge and application of open source platforms are important features in the low-cost perspective. While getting rid of proprietary communication, the open source advent allows quick development and deploying of applications able to run in inexpensive hardware.

The OpenPLC hardware consists of a modular system that has a bus board in which other boards are added in order to provide control functionalities. Even the Central Processing Unit (CPU) is added through a board, as well as inputs and outputs. The bus board works as a rack in which up to 5 boards are connected through DB-25 connectors by RS-485 serial communication. The CPU card holds an AVR ATmega2560 Microship microcontroller used on the Arduino family. Well recognized and established in the market, there are a lot of libraries developed widely distributed that work with this chip. The

board also includes Ethernet communication supporting Transmission Control Protocol / Internet Protocol (TCP/IP) hardwired protocols as Internet Protocol version 4 (IPv4) Address Resolution Protocol (ARP) and Ethernet 100BaseTX. An USB port for communication with a Personal Computer is provided as well. The input card is composed by a microcontroller with the same core of the CPU card, an AVR ATmega328P, and 8 inputs isolated by optocouplers. The output card features 8 relays that additionally have an optocoupler each promoting double galvanic isolation. Alongside, the boards have some security measures such as reverse polarity protection, adequate ground isolation and noise filters for providing robustness and reliability to the system. The constituent software of this PLC is fully compatible with IEC 61131-3 and it is publicly available under General Public License (GNU) whereby it can be modified as desired, what makes possible arrangements to fit particular needs [5].

The automaton designed in that work presents very interesting initiatives in accordance with open-source guidelines that make it attractive. These characteristics are in line with the measures that Industry 4.0 has introduced to industrial reality currently. The concern to develop a system with attributes of protection added to the free access of the constructive characteristics of hardware and software of that PLC is a very strong purpose of that project.

2.2 UniPi Technology

O UniPi 1.1 in Figure 2.2 and the UniPi Neuron in Figure 2.3 are the main products of the homonym company that uses the Raspberry Pi 3 B+ as control board. Directly connected to it, the auxiliary boards have the objective of performing the Input / Output (I/O) interface under different structure approaches.

As an expansion board attached to the Raspberry Pi by a flat ribbon cable, UniPi 1.1 addresses the construction of a PLC in a straightforward manner like a compact PLC but not considering much about the mechanical structure of the automaton. The expansion board lies under the Pi containing all the inputs and outputs of the controller. Protected

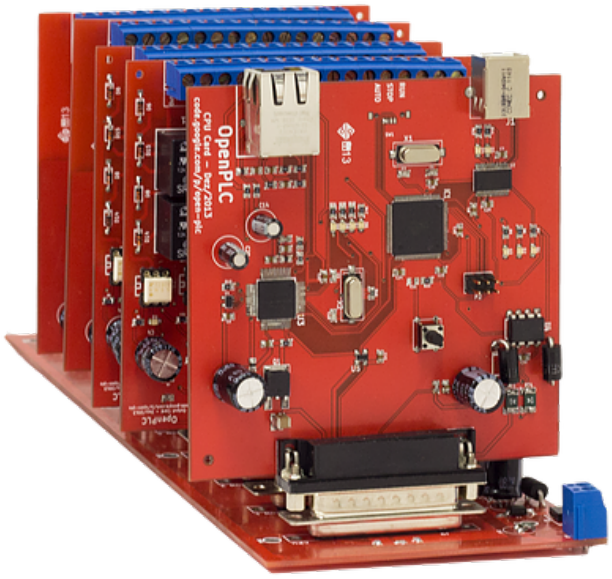

Figure 2.1: OpenPLC hardware composed by a bus, CPU, Input and Output boards [5].

only by an acrylic case, the referred PLC counts on digital and analog inputs and outputs galvanic isolated, featuring output relays over I^2C communication. They are capable of handling up to 5 A under 250 V (AC) or 30 V (DC). This controller is recommended and mostly suitable for home automation and Building Management Systems (BMS).

Meanwhile, UniPi Neuron comes in a DIN rail package what shows consideration with compliance with well established industrial standards. Developed as a modular construction, it includes the types of I/Os of the UniPi 1.1 but now they can be assembled and disassembled according to the user scalability needs. Each module has a different type, quantity and kind of inputs and outputs in order to allow system customization with the purpose of providing versatility control capabilities to the system. This set is recommended for industrial automation systems as the light industry alongside home automation and BMS due to its improved configuration capabilities and compliance to the industrial paradigm [6].

Both controllers are compatible with multiple software frameworks, some of them are paid including the CODESYS compatible with IEC 61131-3. Node-RED is another

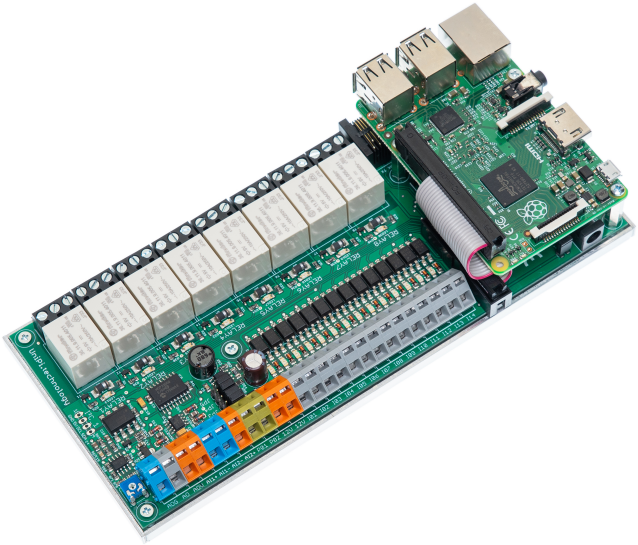

Figure 2.2: UniPi 1.1 controller [6].

compatible software, an important and very interesting platform strictly aligned with Industry 4.0 (I40) premises.

2.3 Sfera Labs

The Sfera Labs company developed the Iono Pi controller that is based on the Raspberry Pi 3 B+. It comes in a compact format protected by a DIN rail case. Iono Pi comprises

Figure 2.3: UniPi Neuron controller [6].

opto-isolated digital inputs and relay powered outputs along analog inputs and outputs. For I/O connection it provides a screw terminal block maintaining the screws accessible what does not require package disassemble for connection. It provides, along 5V Raspberry Pi power supply, a 9 V to 28 V power supply with surge and reverse polarity protection and a 1.1 A fuse. That power supply must be used for the PLC operation while the 5 V Pi's supply is only used for Raspberry setting up. It also features a battery-backed real time clock [7].

Figure 2.4: Iono Pi PLC [7].

The Iono Pi can be controlled by Raspberry native languages Python and Java as well as Controller Development System (CODESYS) that provides standardized IEC 61131-3 languages. Recommended application starts on home and building automation going forward until data acquisition and control in addition to environmental monitoring. The engineering solution presented by Sfera Labs display a simple but direct and low-cost platform which guarantees reliability in appropriate employment of the controller.

Chapter 3

Industrial Controller Architecture

The architecture of the proposed Industrial Controller, represented in Figure 3.1, comprises the concept of an accessible controller through an inexpensive CPU and expansion board that plays the role of the I/O interface. The controller consists of stackable industrial shields that provide the interface of inputs and outputs that are embedded in the Raspberry Pi control board. It is on this same board that the Runtime Environment frameworks run. It can be OpenPLC or FORTE. They are responsible for executing the control logic developed in the Integrated Development Environments that compile the developed code and perform the deploy in the Runtime Environment corresponding to the framework in which it was developed. It can be OpenPLC Editor represented by *PLCOpen logic* image on the figure, and 4DIAC as the IEC 61499 IDE.

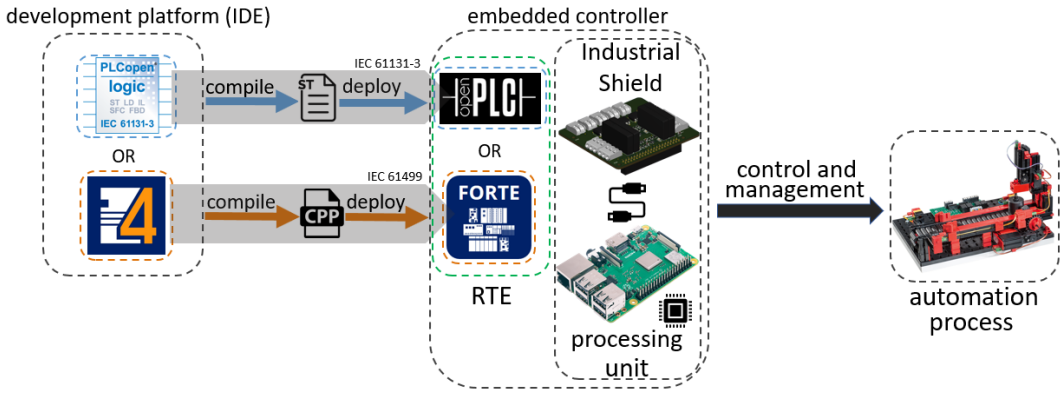

Figure 3.1: Industrial Controller Architecture.

In order to optimize the space used by the hardware, the circuitry was developed with the purpose of saving physical space at the max considering the manufacturing method of these boards. The PLC was thought to use cheap but time tested and well documented components. Starting by the main element, the Raspberry Pi 3 B+ and 3 A+ are efficient as well as a robust devices. The industrial shield, top-connected to the Pi through the 40-pin GPIO, it is in charge of interfacing inputs and outputs to the PLC's brain. It lodges all electronic components which receive the input signals and transmit the shifts to the output depending on the logic embedded in the core.

3.1 Raspberry Pi

Raspberry Pi is an interesting and recent low-cost hardware platform capable of providing a good number of I/Os for various automation applications and controlling devices through their digital ports. Designated as a single board computer, it fits the requirements for the development of a cheap and versatile PLC. Optimized over the years, Raspberry Pi has established itself as a reliable, economical electronic board capable of handling fast and continuous communication.

The models chosen for this project were Raspberry Pi 3 A+ and 3 B+. Such choice is justified by the processing power offered by the platform added to the low cost associated with its acquisition and ease of operation. They embed the System on a Chip (SoC) ARM BCM2837B0 Chip architecture 64-bit capable of running up to 1.4 GHz and powered by 5 V / 2.5 A via MicroUSB cable. The only significant differences in terms of hardware are the smaller amount of RAM and number of Universal Serial Bus (USB) and Ethernet ports available. While the 3 B+ model has 1 GB of Random Access Memory (RAM) and 4 USB A 2.0 ports, the 3 A+ model has half RAM and only one USB port. The less amount of RAM does not affect the operation of the controller since it does not require more memory than the one available. The reduced number of USB ports is shown as an advantage since the application developed does not require this type of connectivity. This factor assists in weight reduction, controller size and ease of connection of I/Os of the shields as it frees

up more space, previously blocked by the column of USB ports. A more negative factor than positive is the absence of the RJ-45 connector. Although it is beneficial to reduce weight and clear the path between external connections with the lower shield, the loss of communication capability via Ethernet architecture is considerable. Despite the controller being focused on the wireless communication offered through two Wireless Fidelity (Wi-Fi) channels (2.4 GHz and 5 GHz), the Ethernet port offers network communication via physical medium which can provide better interference shielding.

Figure 3.2: Raspberry Pi 3 B+ on the left and Raspberry Pi 3 A+ on the right.

Features	**Raspberry Pi 3 A+**	**Raspberry Pi 3 B+**
No. of Cores	Quad-core	Quad-core
CPU Speed	1.4 GHz	1.4 GHz
Memory	512 MB	1 GB
USB 2.0 Ports	1	4
Ethernet Port	No	Yes
WLAN	Yes	Yes
Dimensions (mm)	66x56x14	85x56x17
Weight (g)	23	45

Table 3.1: Raspberry Pi comparison table.

In order to make the Raspberry Pi running the frameworks applied in this work is necessary to define some settings. It is important to make clear that the entire setup process was done equally for every Raspberry Pi used in this work.

To start off with the setting up, the Raspbian Operational System (OS) is installed through writing the boot image to the Secure Digital (SD) flash card inserted in the physical device. After the installation, it is possible to reach and manage the device through a handful Secure Shell (SSH) client. The SSH is a cryptographic network protocol that provides safe communication between the personal computer, used to manage the control system, and the single board computer used to control the automation processes while dismiss the use of a monitor.

For the purpose of connecting the Raspberry to a certain network that needs to be the same as the personal computer, a file with the Service Set Identifier (SSID) and password of the network must be transferred to the SD card right after the OS installation via a physical medium.

Since some General Purpose Input Output (GPIO) pins are set to a high logic value, it is important to set all GPIOs to the low level in order to not activate one of the outputs while the controller is idle. Such circumstances are undesired and must be avoided for safe and adequate utilization of the controller. In the boot folder, *config.txt* is a configuration text file that works as a Basic Input Output System (BIOS) in a conventional Personal Computer. Generally accessible in the boot file, it has several functions including to set GPIO pins digital values during boot time. For the purpose of this work, every digital pin accessible from Pi‘s header was set to low logic level, in which a segment of the configuration text is following.

```
# Set GPIO to be an output set to 0
gpio=14=op,dl
gpio=15=op,dl
gpio=18=op,dl
...
```

3.2 Industrial Shields

Modular, hierarchical and device independent control application are fundamental attributes for modern industrial automation systems. It is introduced in order to increase the operation flexibility mainly for ensuring its ability of easy reconfiguration in face of changing conditions [8].

The industrial shield was developed in order to perform the function of a PLC module. Each board will allow the use of more and more GPIOs from Raspberry Pi to take advantage of the connections offered by the single board computer's digital ports. This connection occurs through the stack of one shield over another until the available ports are exhausted while the current limitations of the device are not neglected. Three shields were developed taking advantage of eight digital ports each. Four are used as inputs and four as outputs according to the need of the application of the case study of this work. Those shields target a direct, seamless and efficient connection between the loads to be controlled and the controller itself. Divided into two sections, the four digital inputs and four digital outputs are the entrance and exit of information flowing back and forth.

3.2.1 First Prototype

Initially, the first prototype displayed in Figure 3.4 was developed in order to verify exactly how many inputs and outputs would fit appropriately in a board that would be directly embedded over the Raspberry. Once input and output section structures were achieved, each section was replicated by four on behalf of circuit cohesion and unity. While component density was confirmed as ideal, the digital communication was tested through a real control of activating and deactivating Direct Current (DC) motors of Fischertechnik's conveyor belts. However, it was not possible to make the motor run in the opposite direction due to the output connection. The relay was only able to energize and de-energize the device coupled with the output connector as it is possible to better visualize by Figure 3.3. That issue was addressed in the succeeding prototype.

Unwanted noise was detected which led the output level to oscillate between high

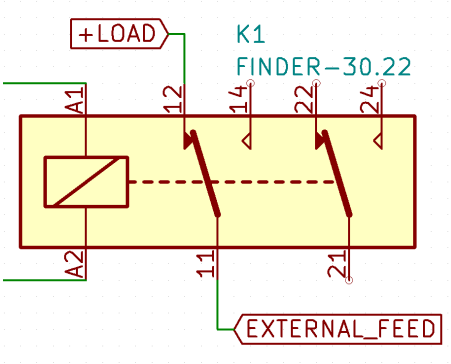

Figure 3.3: Load connection to the output relay on the first prototype version.

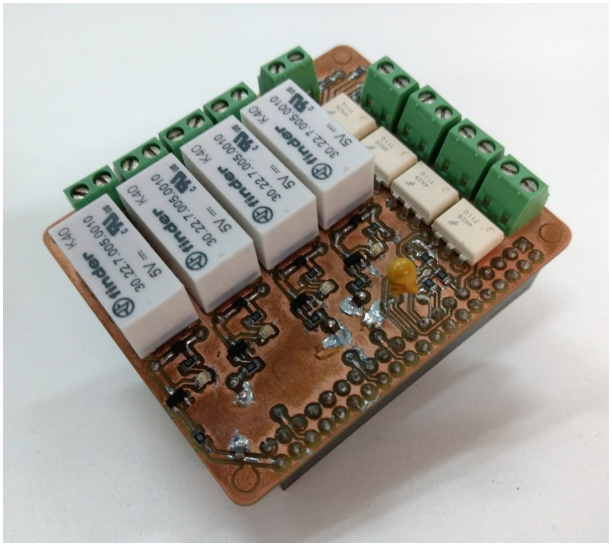

Figure 3.4: First prototype manufactured.

and low during unexpected moments. It was concluded the problem originated from the manufacturing method that is likely to cause the board to suffer from external noise. The answer for that matter was to include 22 nF ceramic capacitors [9] between each Pi's output track and its ground connection to filter the undesired disturbance. A 100 nF capacitor [9] of the same type was added between the 3V3 pin and Pi's ground also to support eliminating noise.

3.2.2 Final Prototype

Every Raspberry connection to the stackable industrial shields was conducted through three 40-pin socket represented in Figure 3.5. The choice for using each digital pin in each board was made mainly in function of the board making method which is not very favorable to the proximity between tracks. Then, it was thought to use input and output pins spaced between themselves throughout the socket in order to avoid short-circuit within them. By this method, it was possible to feed properly each board and get the Pi connected to every digital input and output efficiently.

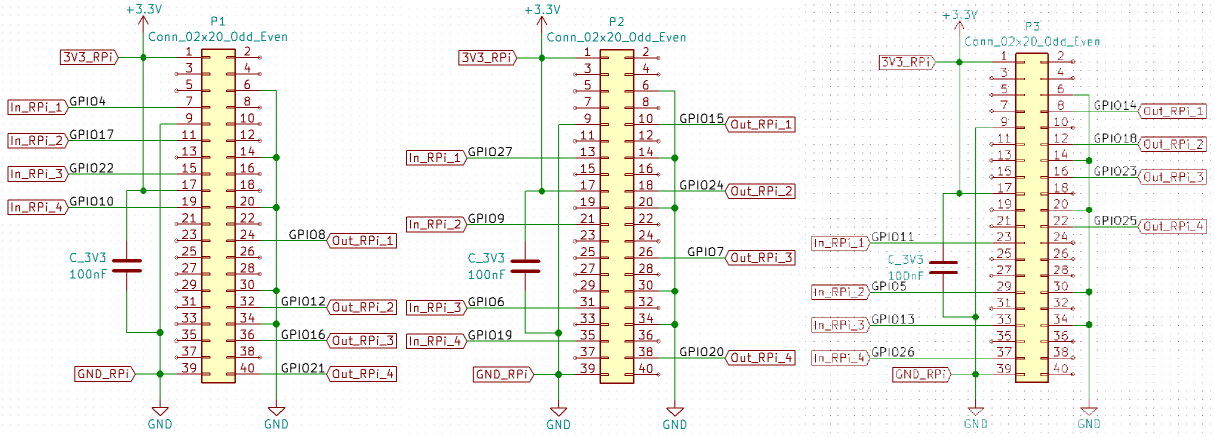

Figure 3.5: Raspberry connections to the industrial shields.

Beginning to describe the input, it has a Wago 2060-452/998-404 connector [10] capable of easy and fast wiring connection and disconnection. That front connection design along a smooth push clamp for decoupling the input device to the shield, deliver the appropriate connector component fitting satisfactorily on the board yet considering the stackable aspect. Followed by 4N26 optocoupler [11], this part provides the galvanic isolation required for voltage levels separation between input devices and the Raspberry. The resistor of 1200 Ω [12] was picked up for the purpose of limiting the current derived from the 24 V input directed to the optocoupler entrance. It has a higher dissipation power of 0.5 W since it handles almost entirely with the input voltage drop. On the other side of 4N26, there is a pull-up resistor connected between Pi's 3.3 V pin and its ground. That 1 kΩ resistor maintains Raspberry's input pin at high digital level while

opto's phototransistor is as an open switch. Every time the infrared LED inside the opto is activated, it sends a light signal which is received by the Negative-Positive-Negative (NPN) phototransistor electrically segregated. Whenever the phototransistor is excited, it shifts Pi's input connected to the collector terminal from low level to the 3.3 V digital high level through the pull-up resistor.

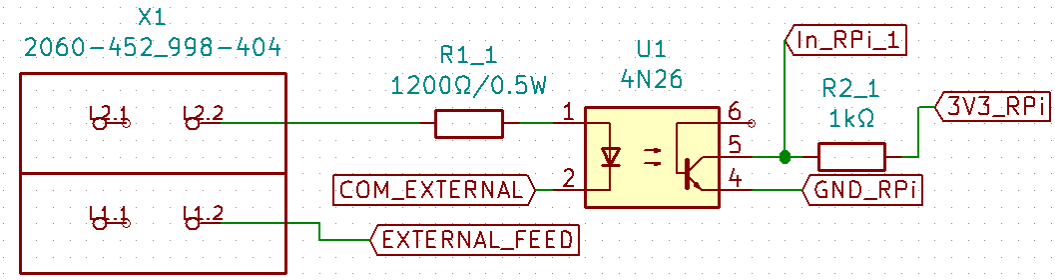

Figure 3.6: Schematic for a general input section of the the Industrial Shield.

The output section of the board comprises a 34.51 Finder relay performing load activation and the circuit that triggers such relay. A 2N7002 Negative - Metal Oxide Semiconductor Field Effect Transistor (N-MOSFET) enhancement type [13] is the component responsible for triggering the relay whenever Pi's output pin shifts to high digital value since the MOSFET is configured as switch. Once the Gate is raised to 3.3 V, it allows the flow of current between Drain and Source, which causes the relay to be activated. Two measures were taken in order not to let the MOSFET be damaged. One resistor of 1 kΩ [14] is situated between Pi's output pin and the Gate terminal in order to avoid heating due to the Gate capacitance in series with track's inductance, phenomenon called ringing. As a matter of protection to the MOSFET as well, one 1N4148WS [15] working as flyback diode is placed in parallel with the relay coil with the purpose of preventing any self generated back Electromotive Force (EMF) spike reach the MOSFET. One SMD Light Emitting Diode (LED) [16] with a 330 Ω resistor [12] in series provides a visual signal in the form of light whenever the relay is activated. It is important to note that despite the nominal activation voltage of the relay being 5 V, 3.3 V handles properly. One ceramic capacitor of 22 nF was added for the purpose of to divert high frequency signals to the ground and consequently eliminate such unwanted noises which may compromise

the proper operation of the application. For appropriate functioning, one of the load's terminal is connected to the movable contact of the relay while the normally closed contact is linked to the external ground and the normally open contact is linked to the +24 V external feed that supplies the amount of energy required by the load. Those connections were properly reached through the small Wago 2060-453/998-404 three-pin connector [10] that fits correctly considering the area and distance between boards.

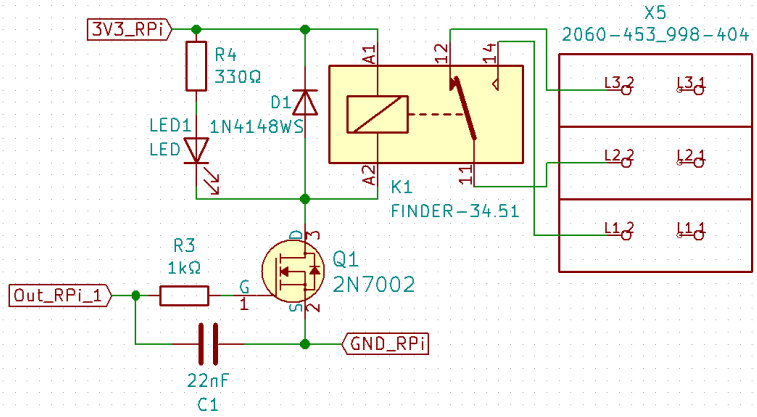

Figure 3.7: Schematic for a general output section of the the industrial shield.

The ground plane is a technique widely used in the manufacturing of Printed Circuit Boards (PCBs), besides being a good practice that mitigates the loss of energy since it provides an extensive area serving the ground, it reduces the resistance of the return path which, thereafter, helps to decrease noise coming from return current variations. Also, it establishes a more uniform ground voltage since a broader path to the current means less opposition to the flow of electrons. Hence, less voltage is dropped than if ground was made on tracks. On the top layer, it is located the Raspberry Pi's ground plane, while on the bottom layer a ground plane for the external supply is disposed.

The use of Surface-Mount Devices (SMDs) were essential for accommodation of the required components in a small space. Such kind of electronic parts saves plenty of PCB space. Higher component density is then achieved, what contributes as well for product ruggedness. The employment of surface mounted devices improves board's mechanical performance as under vibration condition. Nonetheless, the ability for heat dissipation is

a significant matter to be taken into account. As higher voltages implies higher currents, greater power resistors are necessary to controller's proper operation for long periods of time, complying at least to the minimum levels of reliability. Generally, SMDs cost much less than their Through-Hole Technology (THT) components counterparts.

The analysis of the currents circulating in the opto-isolated input and the relay powered output activated by MOSFET configured as a switch is described below.

On the input section as shown in Figure 3.6, the first resistor limits the current flowing through the infrared LED located inside the optocoupler. This LED drops 1.3 V of the power supply and an operating current that varies from approximately 10 mA to the 60 mA. It has been found that when the input section is used with the conveyor belts phototransistor, the operating current is 18 mA due to a conduction resistance when the photo was operating as a closed switch that varies from phototransitor to phototransitor. It adds an undesirable resistance in series. Considering this limiting factor of 18 mA for optocoupler operation when connected to the input phototransistor added to the worst case scenario of current passing through the input resistor that occurs when the input section is connected to the mechanical switches, the resistance was chosen to maintain the minimum current of 18 mA and still be able to dissipate the heat generated by that current. The calculations were rounded to the third decimal place.

$$R_{LED} = \frac{24V - 1.3V}{0.018A} = 1261.111\Omega$$

Then, the commercial resistor of value immediately below SMD 0805 of 1200 Ω with 0.5 W of power dissipation was chosen.

$$I_{IN} = \frac{22.7V}{1200\Omega} = 0.019A$$

$$P_{diss} = 1200\Omega \times (0.019A)^2 = 0.433W$$

As the resistor with such parameters was not available, the SMD of 330 Ω / 0.5 W

soldered on the board was used in series with a resistance of 820 Ω / 0.5 W connected via breadboard. The equation below proves the dissipation power is still under 0.5 W since the current keeps the same at the milliamps level.

$$I_{IN} = \frac{22.3V}{330\Omega + 820\Omega} = 0.019A$$

At the output side of the galvanic isolator, the 1kΩ pull-up resistor maintains the Raspberry Pi input at a 3.3 V high level. Whenever the optocoupler's phototransistor is activated by the infrared LED, it closes grounding the current from the 3.3 V pin.

On the output section, the resistor R1 with value of 1kΩ is employed in order to protect the MOSFET Gate from signal rippling due to ringing which possibly affect its heat dissipation capability. The 22 nF capacitor bypass any high frequency noise to ground. Since the N-MOSFET Enhancement type is biased on the switch mode, when the Raspberry Pi output pin is high, its 3.3 V exceeds the Gate Threshold Voltage (V_{GS}) of 2.1 V which consequently creates the channel between Drain and Source for the current flow from the 3.3 V supply pin. Such current excites the relay coil in order to switch the key on its output channel. The LED has its current limited to a maximum of 3.3 mA by the resistor of 330 Ω. It is lit indicating the relay drive. When the relay is deactivated, the SMD flyback diode 1N4148WS of 100 V Peak Reverse Voltage protects the MOSFET and consequently the Raspberry output pin from the back EMF spike when the relay is deactivated. The output side of the relay connects the load between its 24 V external feed and external ground.

3.3 Board Design and Manufacturing

Designed in KiCAD, that open source suite for printed circuit board development is distributed for free and has several advantages. The use of an Electronic Design Automation (EDA) is essential for structuring a high quality electronic circuit design. KiCAD offers a range of tools that allows not only the definition of standards for circuit parameters e.g.

track widths, clearance between tracks, but also, through tools such as the background grid and the Measure tool that provide precise visualization of the distance between components, allowing optimization of components positioning, tracks and vias.

The platform is divided between two main stages. The first one, Eeschema deals with the elaboration of the schematic. Crucial part of PCB development, the schematic provides the immediate and clear perception of the connections between the components of the circuit. Through the Eeschema window it is also possible to access symbol and footprint libraries. If there is a need to create symbols or footprints, they can be created through this stage as well. The symbols represent clearly the terminals, inputs, and outputs of the components while footprints are critical to know the size and shape of the component, how they occupy the board surface and type of connection, as well the shape of the pads and edges of the component body.

Next, comes the development of the layout of the board itself. Through Pcbnew, the placement and routing of components is accomplished. At this stage, 90% of the time spent is devoted to optimized placement of components. The better positioned, the shorter are the tracks that connects them and denser they fit. Then, routing is done manually, albeit with the aid of very useful tools as the Interactive Router which allows routing intelligently and interactively with the surrounding components making the process design more effective. Three routing modes are available through this tool. Highlight Collisions leaves the cursor free while highlighting any track that is on the way, leaving the designer to decide which path to follow. Shove makes the cursor routing the current track literally shove sideways the tracks that are in the way, very useful tool for dynamic allocation of tracks on the go. Walk Around completes the connection automatically between two pads by the smallest possible path only by dragging the cursor between the two connection points. Pcbnew also provides 3D visualization, an interesting feature that allows a three-dimensional preview of the board as displayed in Figure 3.9.

A brief step-by-step process of the shield development is described in the following lines. Initially, when you start KiCAD, you create a new project. For this work, it is made use of the feature Creating a new Project from Template. The Raspberry Pi 40-pin

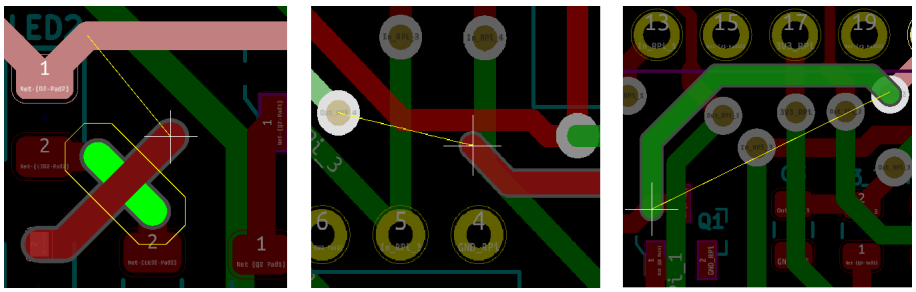

Figure 3.8: Highlight Collisions, Shove and Walk Around from left to right.

Expansion Board template was selected. This template saves a lot of design time of the board since it automatically imports the elementary symbols and footprints of the shield, namely the 40-pin socket, mounting holes and the edge cut lines at the size of a standard Raspberry Expansion Board. Afterwards, when necessary, those lines can be expanded in the interest of giving more room for circuit expansion. After created, Eeschema provides the appropriate environment for the initial symbol selection development if necessary and connection. On the right sidebar there are the Place Symbol Icon used for accessing the symbols' library and to pick the components' symbols needed; the Place Wire icon used for connecting parts by their terminals; and the third most important, the Place Global Label icon utilized for reducing the number of wiring on the schematic view since it provides connection through labels named identically. Once set, those symbols need to be assigned to their footprints counterparts. It can be done by the icon named Assign PCB footprints to schematic symbols. Located in the top bar as well as the previous icon, the Generate Netlist Icon is the the last step of this segment. The Netlist is responsible for addressing each symbol deployed on Eeschema to the footprints in the Pcbnew perspective. Then, it is possible to access the Pcbnew clicking on the Run Pcbnew icon represented by the figure of a tiny PCB portion. Remembering that is a good practice saving the project after each significant step.

Therefore, when Pcbnew opens, the user will come across a black screen and a set of new Tools bar. In order to get access to the footprints it is necessary to import them. It is done through the Read Netlist icon in the top bar. When read, the footprints and ratsnest (lines representing the connections between components) are imported to the Pcbnew

screen with components disposed all scrambling. Among the main features is the grid resolution which can be rearranged in accordance with the size of the smallest component. That feature is very handful when it comes to component alignment. Right after disposing and placing rightly every component considering the most optimized arrangement, it is time for routing. With the intention of developing a board accordingly to safe constraints, the Design Rules feature is located in the Setup menu. It provides a manager with the function of setting up some essential characteristics of PCB design by a EDA software. The majors among them are the Clearance which limits the minimum distance between tracks, self-explanatory Track Width, Via and micro-Via diameter and drill sizes. Careful work is required during the routing with regard to connect the terminals by the shortest path. Vias are placed for linkage of component terminals connected in different layers of the board.

After a cautious and prudent routing, it comes the appropriate moment to apply the ground planes. The employing of ground planes avoids Electromagnetic Compatibility (EMC) issues due to long return current paths as well as ground loops made by track connections. It is executed by drawing a polygon over the desired area utilizing the Add Filled Zone button on the right sidebar. Every shield printed in this work used a double sided PCB comprising a top ground plane for Pi's ground and a bottom ground plane for the external power supply of loads and inputs.

When the board is finished, it is recommended to run the Design Rules Checker (DRC). Pictured by a ladybug icon on the top bar, the Design Rules Checker is a handful tool that verifies potential connection irregularities before printing the board. When completed and assured, the board can be plotted using the Plot icon on the top bar. Top layer (F.Cu), bottom layer (B.Cu) and the edge lines of the board (Edge.Cuts) are selected and then plotted in Gerber format. Generate Drill Files is located on the bottom right side of the window. That button creates the files related to the drill sizes and their locations on the board. Finally, the board can be printed, the parts placed and soldered to the board.

Every PCB was manufactured in a CirQoid printed circuit board prototyping machine. It consists on a Computer Numerical Control (CNC) machine in charge of perform the

board milling process. It consists in different types and sizes of drills responsible for milling as it is possible to see in Figure A.1, drilling and cutting the phenolic board in order to give it the size, shape and electric connections of the developed board. This process automates and gives a lot of precision and uniformity when compared to manual milling production, it has some limitations though. The rough grinding, milling portion of the production process, can be seen by the Figure 3.9. Since the machine needs to be calibrated every time a new board needs to be produced due to the wear of the drills that makes the depth of the milling varies. Thus, the uniformity between plates does not remain constant.

Another important aspect is the arrangement of components when considering this manufacturing method. The components can not be disposed only in accordance with the Design Rules set forth in the EDA software. It is necessary to provide a wider margin of clearance between tracks and components during the layout design. The GPIOs set for output and input were chosen in each board to be distant from each other with the purpose of avoiding short circuits what would be avoidable if it was possible to manufacture the PCB through a modern chemical method. The current layout of the shields was also heavily impacted by the impossibility of making plated through pads. A high number of tracks was arranged in order to meet the need for connection between the opposing sides of the board. A higher number than necessary since there is not the possibility of manufacturing through-plated pads.

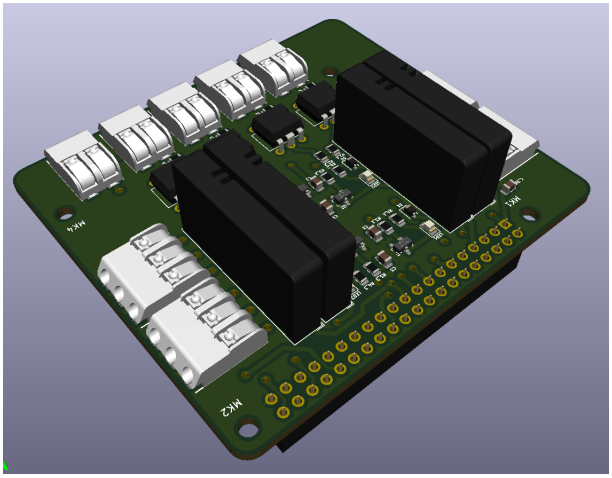

Figure 3.9: 3D visualization of the first stack.

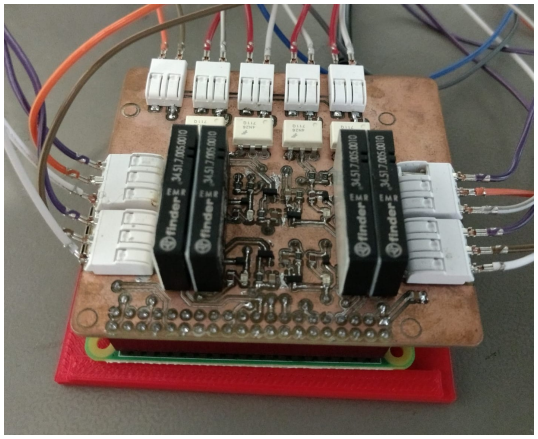

Figure 3.10: Final version of the first stack on the Raspberry 3 A+ connected to the Case Study automation process.

Chapter 4

Programming Environment: IEC Compliant Profiles and Graphic Logic Development

The programming environment consists on the frameworks that make possible the implementation of the control logic. These environments offer a graphical user interface to provide tools that enable the proper development of the logic circuit. They facilitate simple integration for the systems to be controlled since they deliver the tools needed for smooth communication between the parts of the whole system. Between those tools that can be mentioned the multiple standardized languages available alongside other options yet being incorporated to new technologies. The monitoring interfaces making possible debugging in real time. Fast and seamless deployment of control logic into multiple controller devices if desired.

The OpenPLC platform as a runtime environment is embedded in the Raspberry. Able to receive the control logic in Structured Text format from the IDE, it runs the script in the kernel. Through it, the control intentions applied by the operator are deployed on the core and implemented by the OpenPLC. Working accordingly to the IEC 61131-3 standard, the framework enables intercompatibility among different devices including

open source hardware. Since the OpenPLC adopts 61131-3 standard, the lockdown related to the proprietary communication is overcome.

The "For" Distributed Industrial Automation and Control (4DIAC) is a software provided by Eclipse that offers a solution for the distributed control domain. Consisting of the 4DIAC Integrated development environment (IDE) and FORTE as the Runtime Environment (RTE). That framework is codified accordingly to the IEC 61499. Such standard splits the information propagated through the logic between data and events. Designed as function blocks, the logic circuit has the events on the top of the Function Block (FB) structure triggering its functionality and the data on the bottom storing important values that route the way logic will follow. Both IDE and RTE communicate with each other wirelessly whenever connected to the same network.

This IDE is designated to compile every function block into C++ and integrating their code into the RTE. FORTE compiles and starts each FB instance in every single device creating data and event connections between these instances. 4DIAC has already the Raspberry Pi as a native device in the system. Easy to run, after following the tutorials the software shows itself as a good graphic interface and efficient deployer. It features a helpful monitoring interface that allows the user to verify the logic control running in real time as shown in Figure 4.11. Watches can be added to any input or output function block proving to be a supervisor structure while debugging tool.

Regarding function block development, there are some tools available for the development of customized function blocks according to the specific needs of the user's system. The predominant is Function Block Development Kit (FBDK). It is a development tool that provides a graphical interface using Java Developer Kit in order to make possible the prototyping and creation of function blocks compliant with IEC 61499. Concurrently, there is the Function Block Runtime (FBRT) providing a remotely running platform on a configurable device to execute the function blocks system IEC 61499 compliant. Then, the intercommunication between different devices that can be used in the field is achieved [8].

4.1 Programming using the OpenPLC Editor

The OpenPLC Editor is encoded in accordance with IEC 61131-3 by virtue of Program Organization Units (POUs). Functions, Function Blocks and Programs that can be programmed in any of five languages standardized by the IEC 61131-3 [17].

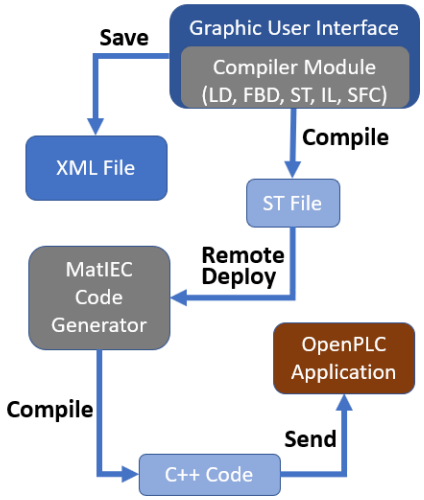

Figure 4.1: OpenPLC Editor building and deployment flow.

As shown in the flow chart illustrated in the Figure 4.1, the process from building to deploy the code follow some steps. Initially, the logic is developed in any of the five languages established by IEC 61131-3 in the graphical user interface. The program can then be saved in the eXtensible Markup Language (XML) format regulated by IEC 61131-10 [18] that allows the exchange of code developed between any device compatible with IEC 61131-3. Through a converter module, the code is converted to Structured Text (ST) and then sent to a back-end compiler (MatIEC Code Generator) belonging to the IDE. As soon as the code is compiled for C ++, it is remotely deployed to the embedded OpenPLC in Raspberry Pi. Through a web graphic interface, the OpenPLC will access remotely the terminal where the OpenPLC Editor is hosted, download the structured text archive, save it in his repository and compile the script. The interface allows the start and stop of the routine and also the management of previous scripts downloaded. The usage of OpenPLC is direct and simple.

Therefore, the PLC program is uploaded as a single file containing the code generated by the graphical editor. Once the file is received, it is sent to MatIEC Compiler where the program is compiled into a C++ program [19]. Additional functionality must be added to the system for network connectivity, internal image tables, hardware interface (to handle physical I/O) and a real-time library [20].

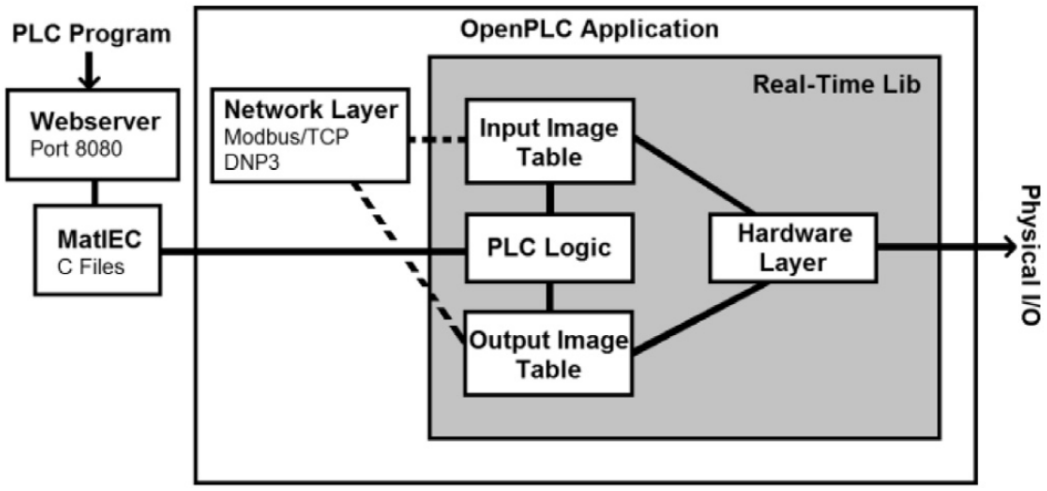

Figure 4.2: OpenPLC internal architecture diagram [20].

The step-by-step since the download of the IDE OpenPLC Editor v1.0, RTE OpenPLC, development of the control program in the IDE until the deployment in the RTE is explained next.

First, the website that hosts the Runtime Environment is accessed and the Raspberry Pi platform is chosen [21]. The software is downloaded as described on that web page.

Through the Raspberry Pi's terminal, the RTE is downloaded to the Raspberry and installed as explained in the previous section. Then, it is possible to access the OpenPLC built-in web server that allows device designation and program upload to run on the embedded device. This communication happens with the Personal Computer connected to the same Raspberry's network. The web server is accessed by opening a web browser tab and typing the Raspberry's IP address followed by 8080 port. The login OpenPLC's page will show up as in the Figure 4.3. The default username and password are both "openpcl". After logged in, it is important to change both credentials through the User

side bar for the purpose of giving security to the programs that will be stored in the server. The very next step is to define the hardware layer code through the Hardware side bar choosing the Raspberry Pi option. Once done, the OpenPLC will be able to recognize Pi's GPIO pins and map according to its input and output designation as can be seen attached at the end of this work.

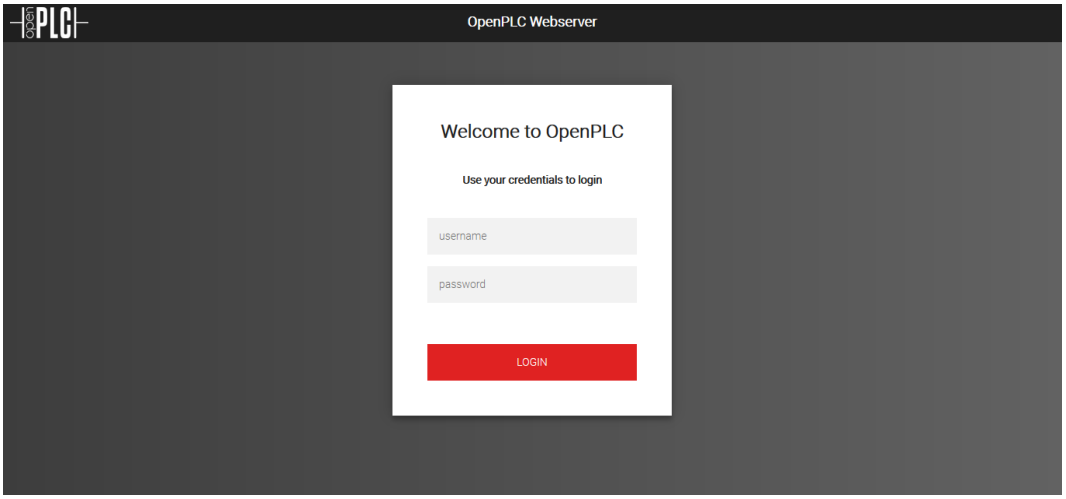

Figure 4.3: OpenPLC web server login page.

The Integrated Development Environment OpenPLC Editor can then be downloaded [22] choosing PC's OS (OpenPLC Editor v1.0 for Windows was the one applied in this work) and unzipped . After unzipped, the software is ready for use. The workspace defined as default by IDE is called *Projeto*. There, every project containing the XML file, ST file, Resource and Configuration accordingly to IEC 61131-3 will be saved after logic development.

Beginning, to start a new project is necessary to go to the upper toolbar, select *Arquivo* and click on *Novo*. A folder with the project's name need to be created inside workspace for then be selected as the local repository of developed control logic. As the project is created, a window will pop-up asking to name the Program that needs to be the same of the project in the case of this software, and define its language among the five possible. After that, Resource inside Configuration which includes Task and Instance are automatically built. It instantiates the Program assigning it to the created task, and the

task itself defining scan-time through cyclic triggering. So, it is possible to develop the control logic.

It is explained how to create the logic in the scope of graphical languages that are used in this project (Ladder Diagram and Function Block Diagram) through a simple example using both in this section. The logic used in this project is presented in the Case Study chapter.

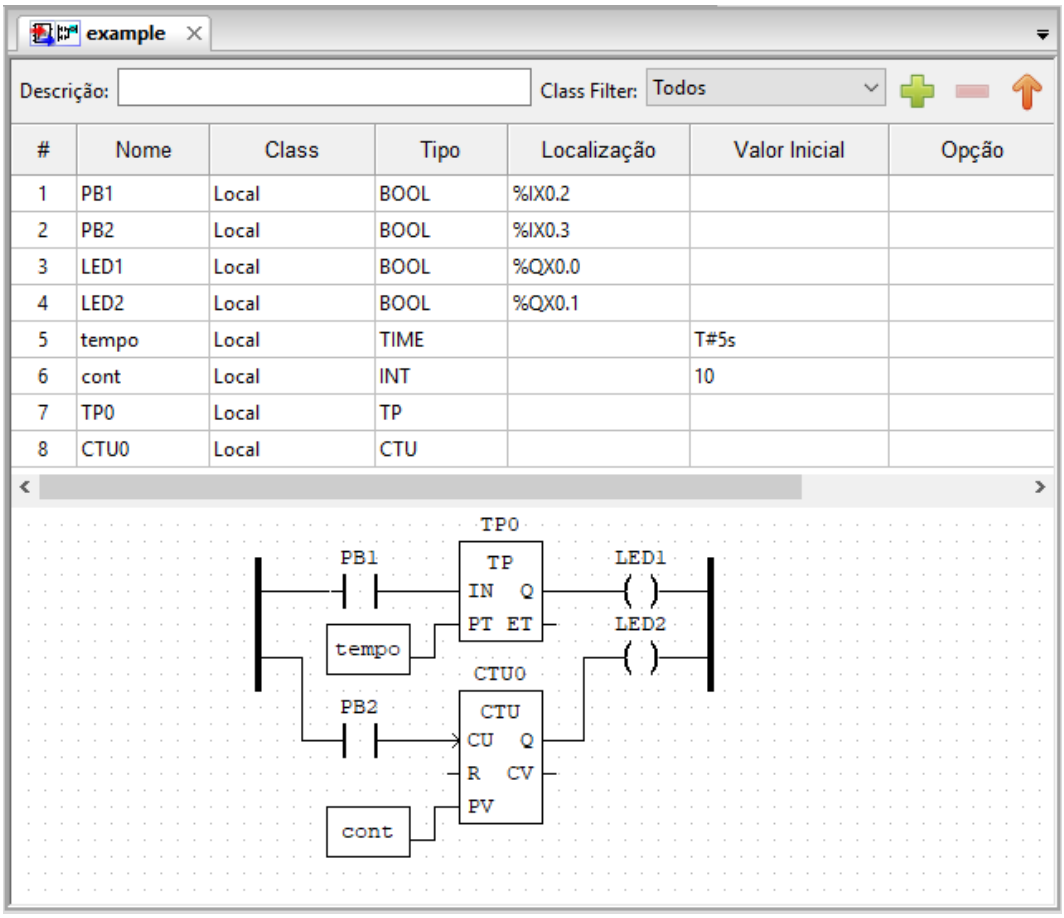

Figure 4.4: OpenPLC Editor logic code example.

In the example presented in Figure 4.4, there are two push buttons as inputs represented by PB1 and PB2, and two LEDs as outputs depicted as LED1 and LED2. Those variables are added by the green plus symbol on the right upper corner. They are named on the column *Nome*, maintained as a local class, their type are set to Boolean (BOOL) on column *Tipo* since buttons vary only as digital levels, and finally they are addressed on

the software as they are connected physically to the Raspberry GPIO and accordingly to OpenPLC's GPIO pin mapping. In this case, PB1, PB2, LED1 and LED2 are connected physically to the pins 7, 11, 8 and 10 represented by the address %IX0.2, %IX0.3, %QX0.0 and %QX0.1, respectively. Two rungs are added and connected to the input contacts and output coils. Every symbol can be chosen by right clicking on the grid and placed with the mouse.

After choosing the contacts and assigning them to the variables, it is time to add the function blocks that will enable the logic to perform more complex activities. On the first rung, TP0 performs the pulse timer function through its graphical interface defined as a IEC 61131-3 function block. While PB1 is being pressed, TP0 is activated by the IN input. The assigned input TIME variable *tempo* provides the value of 5 seconds for the function block. Then, connected to output Q, LED1 is lit every 5 seconds while PB1 is being pressed.

On the second rung, CTU0 block performs the up-counter function. Assigned to Preset Value (PV), the input Integer (INT) variable *cont* sets as 10 the number of times is needed to PB2 be pressed in order to lit the LED2 coil connected to the output Q. Elapsed Time (ET), Reset (R) and Counter Value (CV) are terminals providing extra functions that are not mandatory to be assigned.

Finally, the diagram can be saved in XML format (IEC 61131-10) through the upper toolbar selecting *Arquivo* then clicking on *Salvar* for the purpose of being accessed again by the same or another PLC complaint to IEC 61131-3. Right after, in order to be readable by the Runtime Environment located in the embedded device, it is necessary to compile the diagram to Structure Text format through the orange arrow pointing down on the toolbar right over the program developed. Thus, accessing the OpenPLC web server is possible to upload the ST file and run the control logic developed for controlling the automation process required.

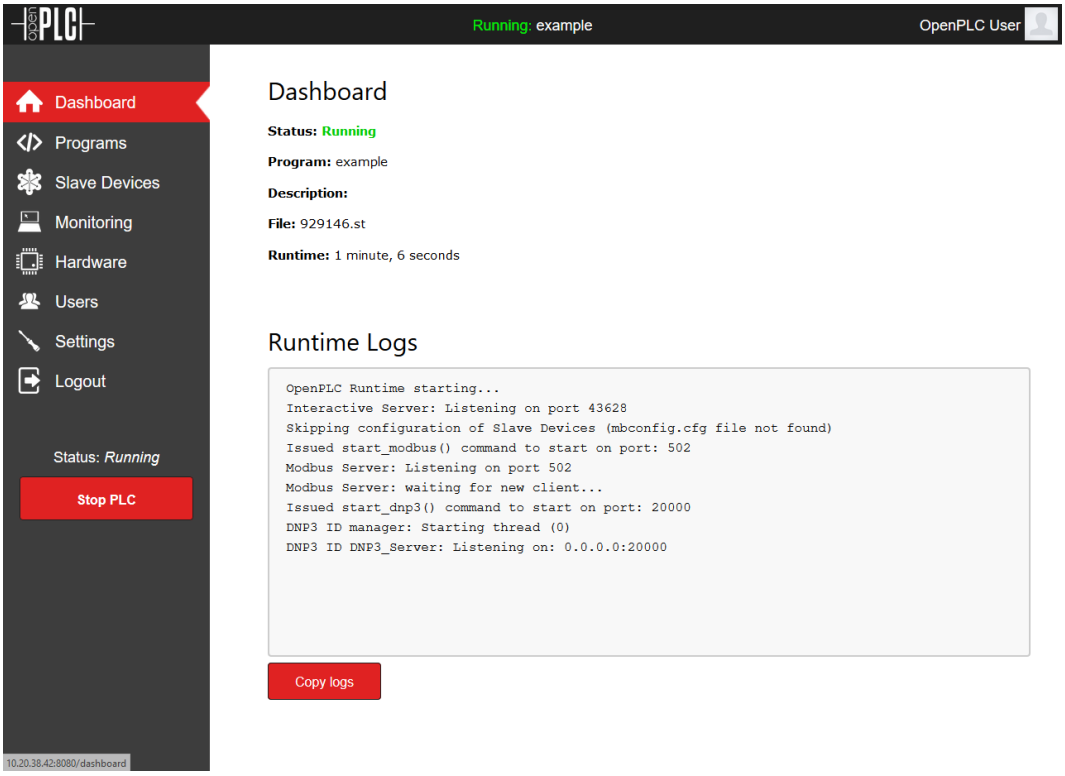

Figure 4.5: Web Server overview while running the example program.

4.2 Programming using 4DIAC

The Function Blocks oriented to distributed systems standardized by IEC 61499 are crucial control elements of this framework. Before explaining how to program with 4DIAC, it is essential to know those key elements structure and how they work.

The Function Blocks are divided among different types. Basic Function Blocks (BFBs) are applied to determine general behavior performing elementary control functions such reading inputs and writing outputs. Service Interface Function Blocks (SIFBs) are employed to interface function blocks situated in different devices spread throughout the shop floor. Composite Function Blocks (CFBs) consist of an aggregation of multiple function blocks in order to create an encapsulated control function performed comprising the specific connections among each block [8].

Divided between events at the top (FB head) and data on the bottom (FB body), these instances are displayed by tiny squares using red for events and blue for data. They

can be interconnected by arrows using the same color for each type of instance. Events and data can never be directly linked [23]. Inputs are always on the left side of the block while outputs are on the right. It is possible to fan out data connections although a fan in is forbidden, i.e. it is possible to connect a data output to many separate data inputs of a next stage however a data input cannot receive connection from more than a single data output. Introduced in the second edition of the norm, *With* constructs associate events and data to the input and on the output of the function blocks, separately, in order to solve data consistency issues [24].

The operation of the Function Blocks can be described through a sequence that depicts the interaction between events and data. First, an event reaches the input of the FB. It triggers data inputs associated to this event. The Execution Control Chart (ECC), a state machine is in charge of providing explicitly the logical sequence through the interactions between the states. It runs activating the FB functionality what gives new values for data outputs [24]. One ECC example can be seen in Figure 4.6. At the end, the output event is refreshed and it is sent to the next stage.

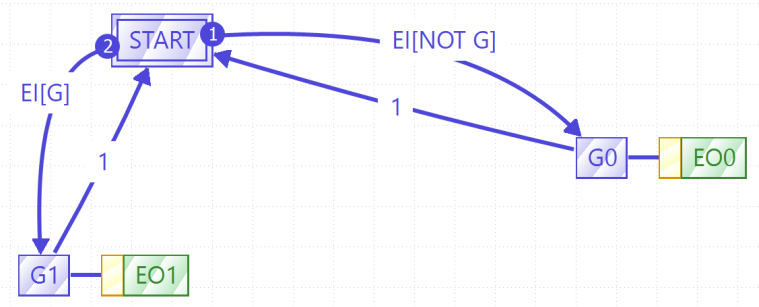

Figure 4.6: Event Execution Control of a Switch Function Block.

The distributed applications are built by mapping each set of function blocks to the related device where it will run or interconnecting FBs' networks through SIFBs with appropriate event and data connections. Such Function Blocks are notable for the purpose of IEC 61449. By means of Resources, messages that are sent and received between devices, synchronization of the function blocks operation throughout the control equipment on field and the sensing information about shifts in the physical world as well as proper

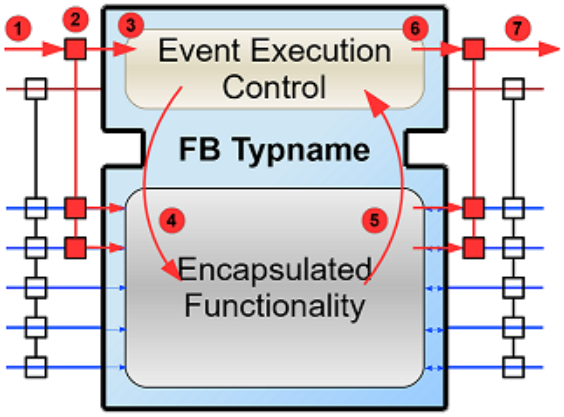

Figure 4.7: Function Block internal operation [25].

reactions in response to those inputs are performed [24].

In charge of integrating different sections of an application, they play an important role since the function inside each device is linked among the others achieving the distributive characteristic and aspect desired by IEC 61499 approach.

4DIAC is mainly divided into three sections. System Perspective, it provides the place for developing the logic itself. The access and management of every control logic developed as well as the definition of connections between devices occurs from this perspective. Deployment Perspective, responsible for selecting, performing and verifying the deployment of the desired systems to devices in the field. At last, Debugging Perspective is used for monitoring the logic functioning through an artifice named Watch capable of providing state visualization of every Function Blocks inputs and outputs and verify errors contained in logic. It resembles Human-Machine Interface (HMI) interface when monitoring Watches are enabled.

FORTE is a compliant IEC 61499 runtime for small embedded devices implemented in C++ with portability to several operating systems such as Linux. While 4DIAC (IDE) is only executed in the development platform, FORTE (RTE) must be executed on all nodes of the distributed computing system at runtime [26].

The step-by-step process from the installation of 4DIAC (IDE) to the development of control logic to code deployment in FORTE (RTE) is described subsequently through a

simple example.

For downloading and installing FORTE on Raspberry is needed to download it from Eclipse webpage [27]. After built, Raspberry is ready to be one of the many possible embedded devices running FORTE controlling an automation process.

To download 4DIAC is, first, necessary to choose the Personal Computer's OS (Windows 64-bit in this work) where the IDE will run and download it. After installing and opening the software, to create a new controller system the user must select *File* on the upper toolbar, point to *New* and click on *New System*. After the project name is filled in, a folder is created in the workspace set inside the 4DIAC's folder within all data related to IEC 61499 architecture. When the project is created, one Application and the System Configuration are automatically generated in the System Perspective.

In System Configuration, the devices applied in the field need to be connected. Raspberry Pi is located on the right side Pallete. It is dragged to the main grid along with Ethernet segment representing connection to the Ethernet network. After connected, it is important to set device's IP adress. The port connection is free of choice, though the first device is commonly set as port 61499 resembling the number of the standard that governs this framework.

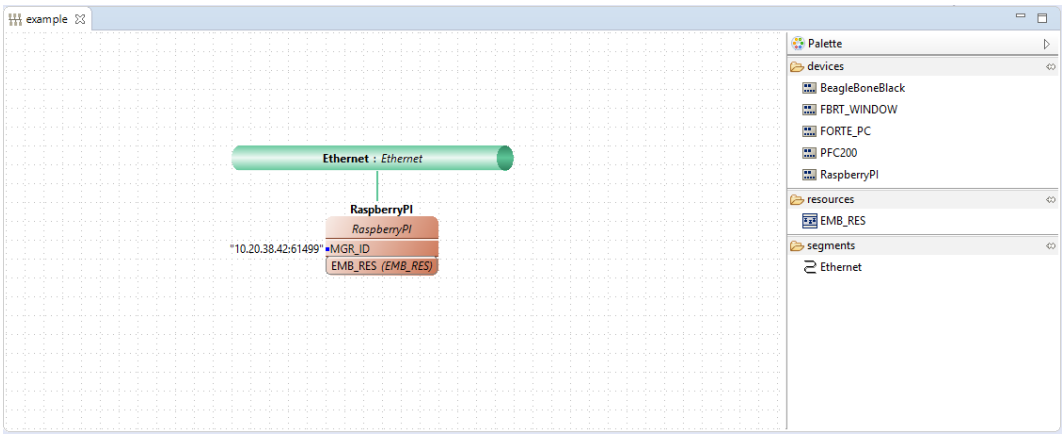

Figure 4.8: System Configuration tab.

Back in the Application tab, the event-driven function block diagram is designed to turn on the conveyor belt whenever a piece is put on its first phototransistor. When

the piece reaches the other edge and triggers the second phototransistor, the conveyor belt stops. The block *E_CYCLE* is in charge of activating input, reading from the phototransistors every 20 ms. Input blocks *IX SENSOR_1* and *SENSOR_2* represent the phototransistors physically connected to 11 and 13 board pins that must be indicated as 17 and 27 accordingly to Broadcom SOC channel (BCM) numeration. Since sensors' normal state is the low logic-level, when the part is placed it shifts sensors to high level. So, the rise edge detection FB is positioned before the Set-Reset (SR) flip flop that activates and deactivates the conveyor belt motor physically connected to 22 board pin (25 BCM) represented by the output *QX* named *MOTOR*. Each function block is dragged from the right side Palette to the main grid.

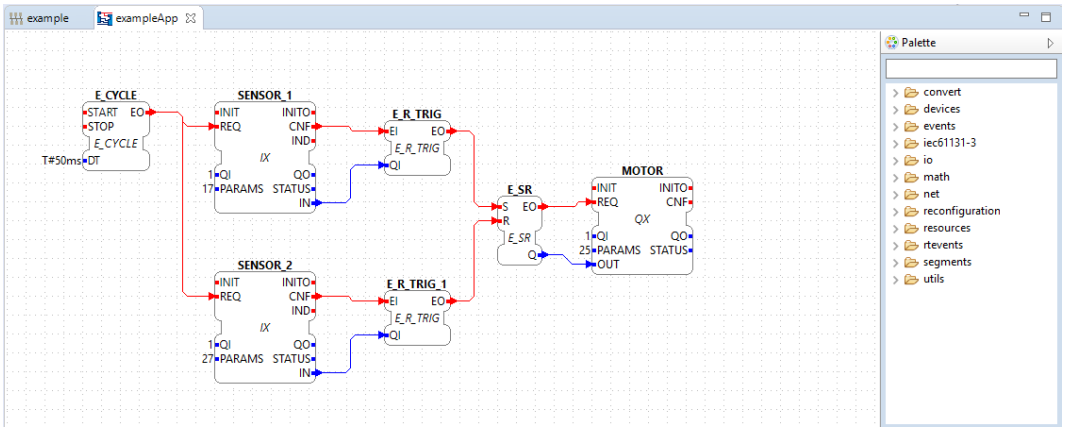

Figure 4.9: Application tab.

Then, these blocks must be mapped to the device Resource granting to Raspberry Pi charge of control over the process. A unique color is assigned to every block mapped to theirs corresponding Resource. In this case, there is only one color attributed since there is only one Resource. Accessing System Configuration and clicking on *EMB_RES*, the Raspberry's Resource is reached and afterwards every block mapped to the Resource is connected to the default Resource's starter function block.

Lastly, switching to Deployment Perspective, and selecting the system intended to be deployed to the controller within its System Configuration and Resource, it is possible to send the diagram to FORTE Runtime framework running on the field device and then

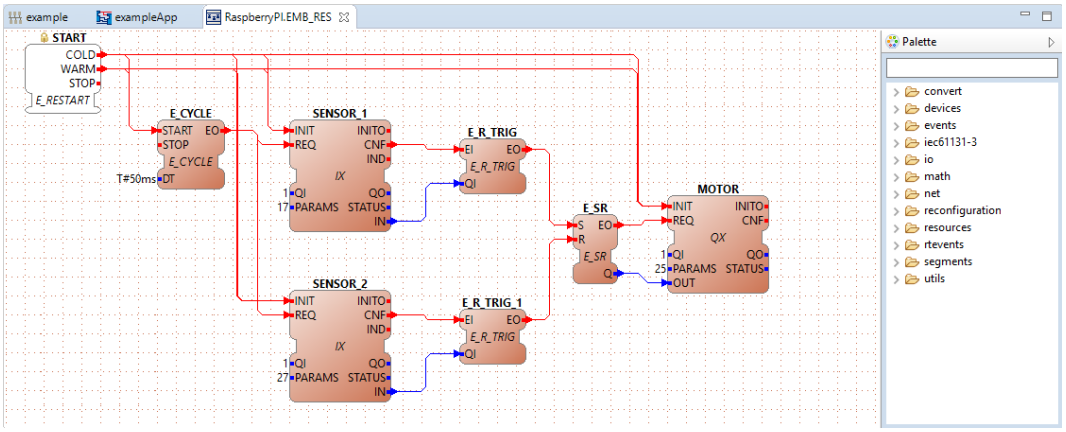

Figure 4.10: Resource within the device.

manage the automation process. If no red sign emerge from the Console, deployment was successful. For monitoring purposes, it is possible to add Watches to every FB's inputs and outputs at Debugging Perspective and enable them on the magnifying glass symbol on the toolbar above the System Explorer tab.

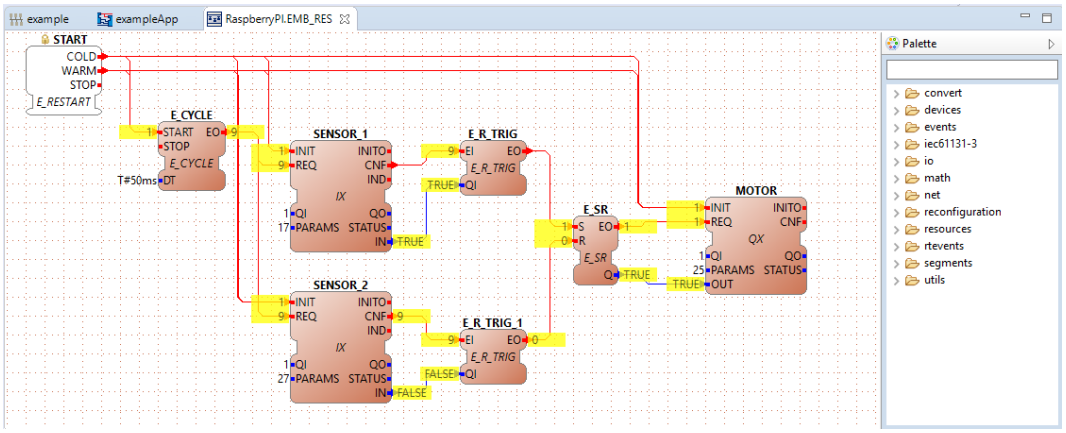

Figure 4.11: Debugging Perspective with Watches enabled.

Chapter 5

Experimental Implementation

In this chapter, the controller application methodology developed in this work is described and exemplified through the use of the Industrial Controller in a industrial simulation hardware representing the automation process to be managed by the control unit.

5.1 Description of the Case Study

The case of study deals with the control and management of the Fischertechnik Punching Machine. The physically simulated manufacturing plant has four sensors (each one working as one input) and two DC motors (each terminal working as one output, i.e. four outputs), fitting the industrial shield I/O resources. At the beginning of the operation, the facility carry a piece of plastic performing as the product to be manufactured over its conveyor belt. At both edges of the belt, there are two phototransistors responsible for communicating the PLC about where the product is along the production chain. Those sensors change the values of the input signals whenever the piece pass through. The first one detects the arrival of the product, it sends a signal to the PLC. Then, by the logic developed on the OpenPLC Editor and 4DIAC IDE subsequently deployed to the OpenPLC Runtime and FORTE respectively, the control function is performed. The output value is shifted by the CPU making the belt's motor throttle. When the part reaches the end of the path, the second phototransistor transmit this information to the controller that

stops the belt's motor and starts the second motor bringing the punching tool way down to the workpiece. It triggers the bottom limit sensor what stops the tool maintaining the piece under the tool pressure. After some seconds the piece is released, the puncher comes back to its original position reaching the top limit sensor, the belt's motor starts running backwards. So, the product is returned to its starting point.

5.2 Implementation with OpenPLC

Firstly, it was developed a program compliant with IEC 61131-3 by use of OpenPLC Editor. The language of choice was the Function Block Diagram and a scan time of 20 ms for comparing purposes between the data-driven approach offered by OpenPLC software and the event-driven provided by 4DIAC. For a more comprehensible explanation, the program was divided in 4 sections. After each section is explained, at the end the integration of each part will be elucidated.

The first section begins declaring the first sensor represented as a variable. PHOTO1 is then connected to R_TRIG0 which activate its own output Q whenever the block detects PHOTO1 shifting from low level to high level. The following four blocks consisting of one NOT, two AND as logic operators and one SR flip-flop comprise a flip-flop T structure. It toggles between 0 and 1 logic levels when activated. In section 2, R_TRIG1 detects when the output of SR0 toggles from 0 to 1 which then activate SR1 and hence the timer-on-delay block TON0 with the variable delay1 of 1 second on its PT input. That corresponds to the one second delay imposed prior to driving the conveyor belt motor through the flip-flop SR2 that is connected to the OUT1 variant. The variable OUT1 is one of the motors terminals that when activated while its other terminal is in low level, takes the part over the conveyor belt towards the other edge. In section 3, when the part reaches the second phototransistor portrayed as PHOTO2, it triggers the rising edge detector R_TRIG2 responsible for deactivating the conveyor's belt motor through SR2. At the same time, PHOTO2 turns the punching motor on right down to the piece by OUT3 variable through SR5 and the 5 seconds-set TON1 through SR3. In section 4, the

Description: Class Filter: Tudo

#	Nome	Class	Tipo	Location	Initial Value
1	PHOTO1	Local	BOOL	%IX0.2	
2	PHOTO2	Local	BOOL	%IX0.3	
3	UP_S	Local	BOOL	%IX0.5	
4	DOWN_S	Local	BOOL	%IX0.6	
5	OUT1	Local	BOOL	%QX0.5	
6	OUT2	Local	BOOL	%QX0.7	
7	OUT3	Local	BOOL	%QX1.0	
8	OUT4	Local	BOOL	%QX1.2	
9	delay1	Local	TIME		T#1s
10	delay2	Local	TIME		T#5s
11	delay3	Local	TIME		T#4s
12	R_TRIG0	Local	R_TRIG		
13	R_TRIG1	Local	R_TRIG		
14	TON0	Local	TON		
15	TON1	Local	TON		
16	SR0	Local	SR		
17	SR1	Local	SR		
18	SR2	Local	SR		
19	SR3	Local	SR		
20	SR4	Local	SR		
21	R_TRIG2	Local	R_TRIG		
22	SR5	Local	SR		
23	SR6	Local	SR		
24	TON2	Local	TON		
25	SR7	Local	SR		
26	R_TRIG3	Local	R_TRIG		
27	R_TRIG4	Local	R_TRIG		

Figure 5.1: Variables and function blocks of the OpenPLC program for punching machine control.

variable DOWN-S represents the limit sensor positioned at the punching arm's bottom end. It turns the punching motor off for 4 seconds maintaining the press against the piece. After that time, the other terminal of the punching motor is activated uplifting the press until it reaches the top limit sensor represented by UP_S variable which deactivates the punching motor. Meanwhile, after 5 seconds of PHOTO2 trigger, the second terminal of conveyor belt's motor turns high what runs the belt backwards, returning the piece to the initial position. When the part reaches PHOTO1, it resets OUT2 through SR4 finishing the process.

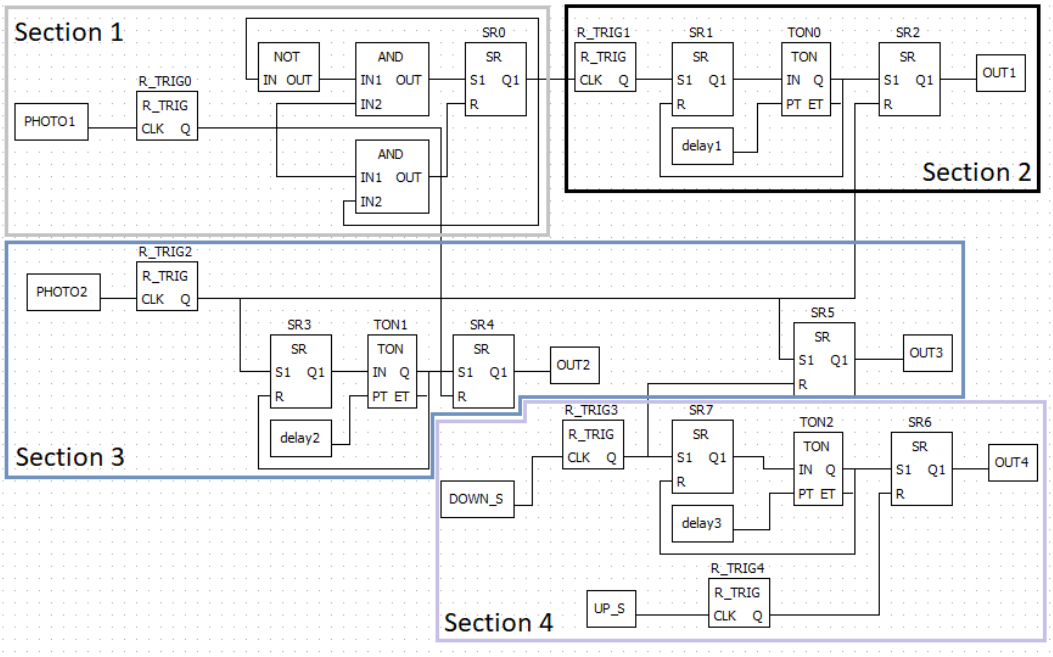

Figure 5.2: Punching machine control program by IEC 61131-3 sectioned.

5.3 Implementation with 4DIAC

Next, the control of the same industrial simulation plant performed by 4DIAC compliant with IEC 61499 is presented. The control follows the same algorithm, but now the function blocks differ from the data-driven to event-driven approach.

It begins with a startup delay of 2 seconds necessary to let the system be initialized and all inputs be available. The problem related to lag on inputs function blocks initialization made it necessary to implement the STARTUP_DELAY as can be seen in Figure 5.3. The next block determines how often the system activates its inputs and outputs for read and write operations which was set to 20 ms. Every 20 ms, a request is sent to verify each input state in order to not overload CPU processing. This value was chosen in order to provide unnecessary stress to the CPU while it is considered an adequate amount of time between the input readings.

As PHOTO_1 detects the piece, it triggers the rising edge function block next to it, UP_TRIGGER_1, which subsequently activates a switch function block. That switch is responsible for shifting the conveyor belt direction after the part being pressed. At

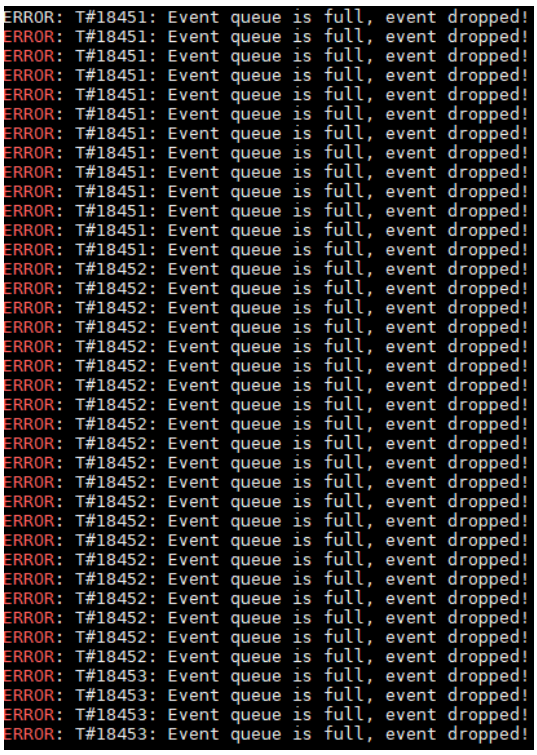

Figure 5.3: An event overflow observed when the system starts running without proper device input initialization.

the beginning, the E_SWITCH data input G is equal to 0. Then, when event input EI is triggered, it sends a signal through EO0 starting a delay function block. ENTRANCE_DELAY keep the conveyor belt inactive until 1 second after the part is detected at the first sensor PHOTO_1. After that time, the conveyor belt's motor terminal which is connected to Pi's output pin 24 (board GPIO numbering) is raised to 24 V making the belt rotate towards the punching structure. It is set through the SR flip flop function block named SET_RESET_1. When the piece reaches the second phototransistor, PHOTO_2 triggers UP_TRIGGER_2 which simultaneously deactivates conveyor's belt motor RPi_OUT_24, starts the punching motor down to the piece through RPi_OUT_32 and the 5 seconds delay that sets to high the second terminal of conveyor belt's motor RPi_OUT_32 after the piece being pressed.

When the press reaches DOWN_S sensor, it stops the punching motor maintaining it pressing the part concurrently triggering the processing delay of 4 seconds. After

that time, the second terminal of the punching motor is activated carrying the press up until it reaches the UP_S sensor where the motor stops. After 5 seconds the motor has arrived to PHOTO_2, PROCESSING_DELAY sets conveyor's motor to return the piece through SET_RESET_2. When it reaches PHOTO_1, the switch function block called FOWARD_BACKWARDS is triggered again by UP_TRIGGER_1. However, since the second belt's motor terminal RPi_OUT_32 was activated, it shifted G data input of the switch function block to level 1. Then, when FOWARD_BACKWARDS input event EI was triggered this second time, it enabled EO1, not EO0. Therefore, the belt's motor is deactivated and the piece finish its return to the starting point.

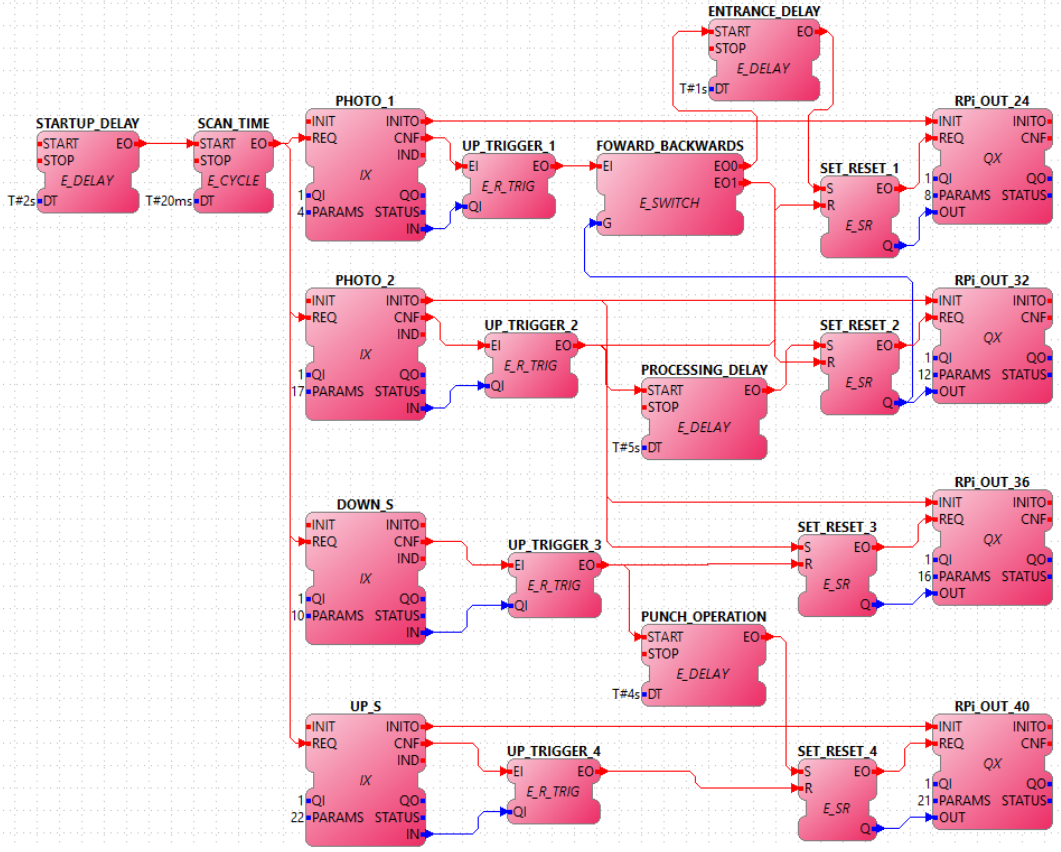

Figure 5.4: Punching machine control program by IEC 61499 approach.

After these proposed tests, the piece was put on the middle of the controlling process, i.e. on PHOTO_2. Bypassing the PHOTO_1, is then possible to observe the functioning of the controlling system when accessed otherwise than at the intended beginning of the

process. Under 4DIAC, the piece was pressed and released as expected, and then carried to the PHOTO_1 where it stopped and stayed. While under OpenPLC control, the piece after being pressed and released, it headed to PHOTO_1, and after 1 second it was carried to PHOTO_2 where the pressing, releasing, and transport back to PHOTO_1 was performed. After the second time PHOTO_1 was activated, then, the piece stayed on place.

Input Name	**Input Address**	**Output Name**	**Output Address**
PHOTO1 (PHOTO_1)	%IX0.2 (BCM 4)	OUT1 (RPi_OUT_24)	%QX0.5 (BCM 8)
PHOTO2 (PHOTO_2)	%IX0.3 (BCM 17)	OUT2 (RPi_OUT_32)	%QX0.7 (BCM 12)
UP_S (UP_S)	%IX0.5 (BCM 22)	OUT3 (RPi_OUT_36)	%QX1.0 (BCM 16)
DOWN_S (DOWN_S)	%IX0.6 (BCM 10)	OUT4 (RPi_OUT_40)	%QX1.2 (BCM 21)

Table 5.1: Input and outputs adresses on OpenPLC and 4DIAC, respectively.

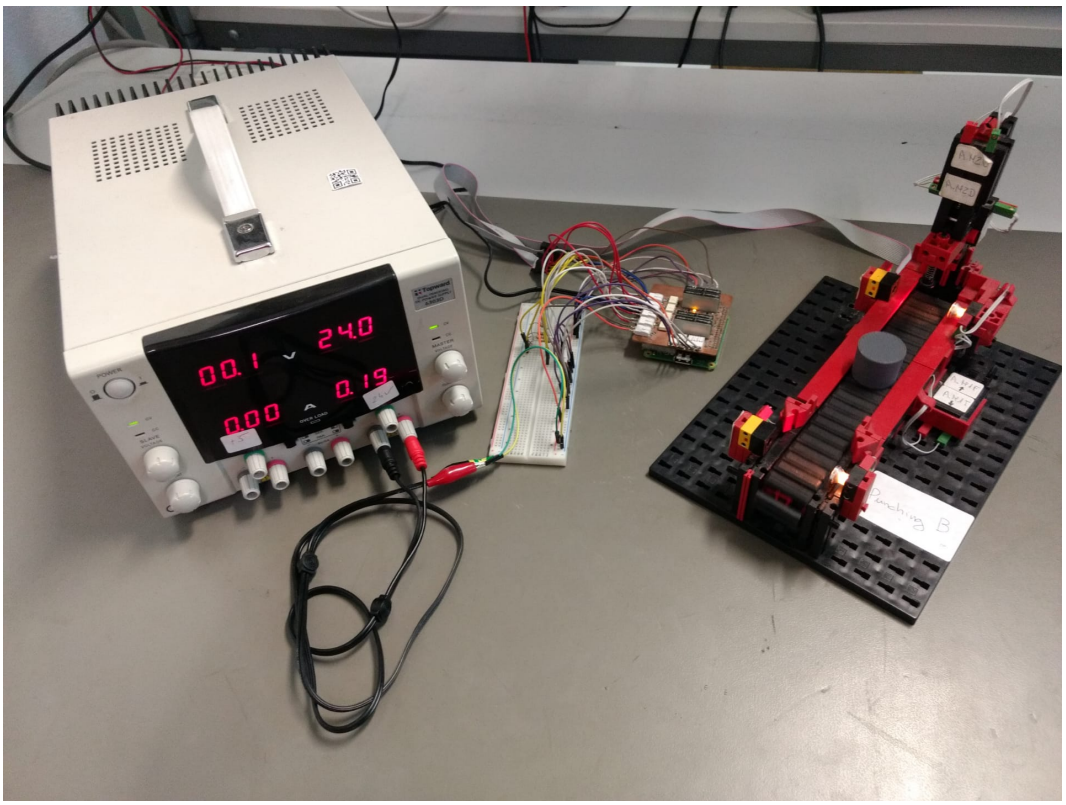

Figure 5.5: Automation process being controlled.

Chapter 6

Results and Discussion

In the Case Study it is possible to observe the functioning and behavior of the proposed Industrial Controller controlling efficiently an automation process under two frameworks that follow different IEC standards. The first one, OpenPLC provided the behavior expected by use of Function Block Diagram. The logic was developed taking in consideration every step forward in a temporal perspective. Each section was developed taking into account the value of its data at every time the control flows, so a beginning point of the operation must be duly observed and defined. Meanwhile, when controlled by 4DIAC, even when the operation is activated on an input located in the middle of the process, each step further is performed seamlessly until the defined end. This behavior was accomplished due to the event-driven approach of 4DIAC that not only encapsulates logic in Function Blocks, but also separates data from events. Such implementation generates a local cascade of control sequences. What matters for adequate control performance are the previous and next function block data values adequacy when the an event input is triggered. This is a very efficient distributive technique implemented where the Industrial Controller perform precisely.

Additionally, the scan time was decided to be 20 ms due to the application demand that does not require a much higher response time. This value also contributes to the reduction of processor heating even when the scalability of the system to be controlled increases.

From this work, it was possible to observe the complementarity between the standards used in the execution of plant control carried out in the Case Study. Therefore, the analysis of this characteristic was carried out and how this impacts the operation of the control system.

IEC 61499 defines event-driven function blocks which can be distributed to field devices and interconnected across multiple controllers. This standard is characterized by software portability between tools of several suppliers, interoperability among different devices and distributed configuration possibilities. Such traits add flexibility in a manner that empower the system to adapt and reconfigure based on environment changes. This architecture reduces the cost and complexity of industrial automation processes.

Complementing IEC 61131-3 programming languages, IEC 61499 is developed to support distributed control system not only encapsulating algorithms but also system applications and sub applications so the system designer is able to design applications and reuse them to simplify design procedures whilst the older standard allows only whole changes of the application, not gradually [28]. Code is encapsulated allowing operators who are not programming experts to understand and implement the graphical code through blocks. The function block structure goals the reuse of control algorithms.

Extending the concept developed in the previous standard, the newer norm facilitates control decentralization since its function blocks can be mapped to specific devices by the choice of the designer. Moreover, the control is performed asynchronously. Each block is triggered from its predecessor decreasing connection interdependence among the function blocks. It provides a useful and effective distributed control. This reduces the number of connections between function blocks making debugging easier what in turn diminish chances of making an error. It contributes to make a flexible code which is easy to maintain and reconfigure in the face of changing conditions reducing the complexity of distributed systems. It rids the need of redesigning the entire scheme one more time.

Therefore, distributed control is achieved by direct link to devices spread throughout the manufacturing plant. Decision processes associated with an application are not running under a single processor. They are divided among several processors, each having

their own thread of control [8].

The release of the IEC 61131-3 standard covered very well the programming languages aimed for controller code development. Its hierarchical design techniques make plant controls easy to design, they also result in designs that are very useful for plant maintenance. However, as the control program grows due to the addition of new components to be controlled, the scalability of control does not grow proportionally. The IEC 61131-3 employs the data-driven approach meaning that the data connections between FBs define the overall execution order which denotes no direct control over the processing sequence by the user [28]. Much uniformity with respect to the architecture of the PLC controllers was established through this standard, although requirements relative to updated control nature arose over time and needed to be complemented.

Central control systems are being replaced by smart connected devices in distributed controlled environments. Heterogeneous systems with components from different hardware and software manufacturers are becoming effectively accomplished [8].

The Standard IEC 61131-3 was developed with the primary objective of unifying the way of programming PLCs. Additionally, it introduced concepts from the domain of software engineering into the industrial control field. It improves the quality, modularity, and reusability of control applications when compared to the previous pure electromechanical control systems.

A minor yet important feature missed was the ability to exchange scripts, libraries and projects between development softwares compliant with the IEC 61131-3. IEC 61131-10 defined an open interface which can be used by different kinds of software tools to transfer the information to other platforms. XML (eXtensible Mark-up Language) accomplished interoperability exchange of created programs.

The concept of decentralized control is not such a recent perspective. However, it is only after the development and integration of technology in the field of telecommunications, electronics and computation at the current level that it is possible to develop distributed control systems that are efficient while at the same time accessible in the financial sphere. This becomes the most targeted approach to the control of extensive

manufacturing plants since every device have intelligence and is able to communicate with each other efficiently making the system work collectively. IEC 61499 proposes through up-to-date tools based on contemporary technology interesting solutions in the field of distributed control, extending the old and still highly applicable standard IEC 61131-3 to current requirements.

The raise of competition between manufacturing companies of this new globalized era has increased the requirements in relation to the capacity of reconfiguration of the chain of production. A broad expansion of the production lines diversity is necessary in order to keep the competitive power at the highest. Hence, improving the flexibility of system design and maintenance costs drive the automation control field to bigger challenges.

Chapter 7

Conclusions and Future Work

Low-cost PLC is not a new technology. However, they are outdated. As technology evolves, while taking advantage of the benefits of this evolution, technological investment must keep being made for the continuous improvement of these important techniques that raise the automation control field to ever more fascinating levels of efficient and inexpensive automation. Continuing improvement includes renewing already successfully established technologies in the industry through an approach that updates it to the new perspective of the current technological reality. Every machine is becoming smarter. From device to device, more than ever, the importance of enabling communication between control devices, even more in a decentralized way, is the natural direction of technology development from now on in the face of the enormous benefits of greater flexibility, reconfiguration and low cost from this new reality being established through Industry 4.0.

In this work, it was proposed to develop an automaton capable of controlling an automation process. For that, the concept of galvanic isolation was established in order to separate and consequently protect the low voltage CPU from the external voltage of 24 V standardized in the automation field for outputs and inputs. Then, after the development of the electronic circuits that are part of the input and output sections, the way the Industrial Shields would connect was discussed. The number of 4 inputs and 4 outputs per board was established in order to fit the space granted by the area superior to Raspberry Pi. The alternative of using flat ribbon cables to connect the Industrial Shields

with each other was brought up. However, a more direct and space-saving solution was chosen. Stackable Shields connected through 40-pin extended sockets makes the controller more compact as well as makes handling and fixing more robust. It was also observed that the Raspberry Pi 3 A+, released during the execution of this work, presents the same processing power while having a smaller size than the 3 B+ version what optimizes the Industrial Controller volume.

The main difficulties encountered in carrying out the work were related to the PCB making process. The method, although accessible, is not compatible with the better standardization offered by PCB manufacturing methods today. The final version of the board took up more space than needed. If it were possible to manufacture by the latest chemical methods with mask application in order to protect both sides of the plate, use of plated vias in addition to the high isolation provided between tracks within a little but proper and adequate clearance, an optimized layout would be achieved that would use less space and make the Industrial Shields more reliable without worrying about possible short circuits that exposed copper PCBs present.

The Industrial Controller developed in this work proved to be an efficient and highly desirable solution from the perspective of low-cost Programmable Logic Controllers achieving an adequate behavior under the desired scan time. Raspberry Pi plays the main role as the inexpensive and efficient processor of the system. Providing enormous processing power, connectivity and reliability at a low cost makes it an excellent choice implemented in the controller. The choice of the small components added to the use of the 40-pin GPIO that the Raspberry provides for direct connection, offers enough space to allocate the Industrial Shields one over another. In this way, the use of space is greatly optimized while allowing the controller to be modular. According to the need for inputs and outputs to be controlled, Shields can be withdrawn and added as needed.

Central control systems are being replaced by smart connected devices in distributed controlled environments. Heterogeneous systems with components from different hardware and software manufacturers are becoming effectively accomplished. For that purpose focus was placed on implementing a cyber-physical system compliant with centralized IEC

61131-3 with the objective to deliver multiple well established languages of the automation controlling field and decentralized IEC 61499 that was employed to offer a programming solution based on event-driven function blocks responsible to take advantage of distributive features and deliver it as an innovative and very useful approach of automation process controlling. For the logic control development and implementation, open source frameworks were employed in order to offer the most accessible and unbounded vendor-proprietary environment.

Future work includes a modern chemical method of PCB manufacturing that allows proper copper isolation and plated vias in order to achieve a better quality and finished version of the PCB alongside a new solution for powering up the controller. Integration with Supervisory Control and Data Acquisition (SCADA) systems is an interesting point worth getting through since it enables a monitoring environment capable of monitoring the system behavior. It's an important step further that makes the control system robust and up to date with current technologies. ScadaBR is a solution in that concern supported by OpenPLC which consequently shows itself as the first path to be taken in that direction. Shields that are capable of working with analog signal are also an interesting proposal that aims to diversify the type of input and output connectivity to the controller. For logic control development purposes, agents approach is an opportunity of deployment very useful for autonomous and reliable control. Using the Java Agent Development Framework (JADE), the agents approach is highly capable of handling the automation process in an efficient and modern way. Developing aspects concerning the mechanical robustness of this controller requires a deeper study regarding the mechanical component in which the IEC 61131-3 standard is responsible for establishing parameters.

Bibliography

[1] L. A. Bryan and E. A. Bryan, *Programmable Controllers: Theory and Implementation*, 2nd. Industrial Text Company, 1997, p. 1035.

[2] H.-H. Erbe, "Manufacturing with Low Cost Automation", vol. 36, no. 7, pp. 95–100, 2003.

[3] International Electrotechnical Commission, *IEC 61131-3:2013*, `https://webstore.iec.ch/publication/4552`.

[4] ——, *IEC 61499-1:2012*, `https://webstore.iec.ch/publication/5506`.

[5] T. R. Alves, M. Buratto, F. M. de Souza, and T. V. Rodrigues, "OpenPLC: An open source alternative to automation", in *IEEE Global Humanitarian Technology Conference (GHTC 2014)*, 2014, pp. 585–589.

[6] U. technology, *Unipi technology*, `https://www.unipi.technology`.

[7] S. Labs, *Sfera labs*, `https://www.sferalabs.cc`.

[8] M. Sadeghi, "Developing IEC 61499 in Industrial Processes, Measurement and Control Systems (IPMCS)", *WSEAS Transactions on Systems and Control*, vol. 5, 2010.

[9] Vishay, *Datahseet 22 nF and 100 nF 0805 SMD Ceramic Capacitors*. [Online]. Available: `http://www.vishay.com/docs/49653/49653.pdf`.

[10] Wago, *Datahseet Wago Connectors 2060-452/998-404 and 2060-453/998-404*. [Online]. Available: `https://www.wago.com/infomaterial/pdf/51300133.pdf`.

[11] Vishay, *Datahseet DIP 4N26 Optocoupler*. [Online]. Available: `http://www.vishay.com/docs/83725/4n25.pdf`.

[12] ——, *Datahseet 0805 SMD Power Resistor 330 Ohm / 0.5 W and 1200 Ohm / 0.5 W*. [Online]. Available: https://www.vishay.com/docs/20043/crcwhpe3.pdf.

[13] Fairchild, *Datahseet SOT-23 SMD 2N7002 N-Channel Enhancement Mode Field Effect Transistor*. [Online]. Available: https://www.mouser.com/ds/2/149/2N7002-8405.pdf.

[14] Vishay, *Datahseet 0805 SMD 1000 Ohm / 0.125 W Thin Resistor*. [Online]. Available: https://www.vishay.com/docs/28705/mcx0x0xpro.pdf.

[15] ——, *Datahseet SOD-323 SMD 1N4148WS Small Signal Fast Switching Diode*. [Online]. Available: https://www.vishay.com/docs/85751/1n4148ws.pdf.

[16] Kingbright, *Datahseet Yellow 0805 SMD LED*. [Online]. Available: http://www.us.kingbright.com/images/catalog/SPEC/APT3216SYCK.pdf.

[17] E. Tisserant, L. Bessard, and M. de Sousa, "An Open Source IEC 61131-3 Integrated Development Environment", in *2007 5th IEEE International Conference on Industrial Informatics*, IEEE, vol. 1, 2007, pp. 183–187.

[18] International Electrotechnical Commission, *IEC 61131-10:2019*, https://webstore.iec.ch/publication/33034.

[19] M. de Sousa and A. Carvalho, "An IEC 61131-3 compiler for the MatPLC", in *EFTA 2003. 2003 IEEE Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No.03TH8696)*, vol. 1, 2003, 485–490 vol.1.

[20] T. Alves and T. Morris, "OpenPLC: An IEC 61,131–3 compliant open source industrial controller for cyber security research", *Computers & Security*, vol. 78, pp. 364–379, 2018.

[21] T. Alves, *OpenPLC on Raspberry Pi*. [Online]. Available: https://www.openplcproject.com/getting-started-rpi.

[22] T. R. Alves, *OpenPLC Editor*. [Online]. Available: https://www.openplcproject.com/plcopen-editor.

[23] K. Thramboulidis, "IEC 61499 in factory automation", in. 2007, pp. 115–124.

[24] J. Christensen, T. Strasser, A. Valentini, V. Vyatkin, and A. Zoitl, "The IEC 61499 Function Block Standard: Overview of the Second Edition", 2012.

[25] E. Foundation, *4DIAC*, `https://www.eclipse.org/4diac`.

[26] M. V. García, F. Pérez, I. Calvo, and G. Morán, "Building industrial CPS with the IEC 61499 standard on low-cost hardware platforms", in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 2014, pp. 1–4.

[27] Eclipse Foundation, *Eclipse FORTE installation*. [Online]. Available: `https://www.eclipse.org/4diac/documentation/html/installation/raspi.html`.

[28] W. Lepuschitz, A. Zoitl, M. Vallée, and M. Merdan, "Toward Self-Reconfiguration of Manufacturing Systems Using Automation Agents", *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, pp. 52–69, 2011.

Appendix A

Apendix A

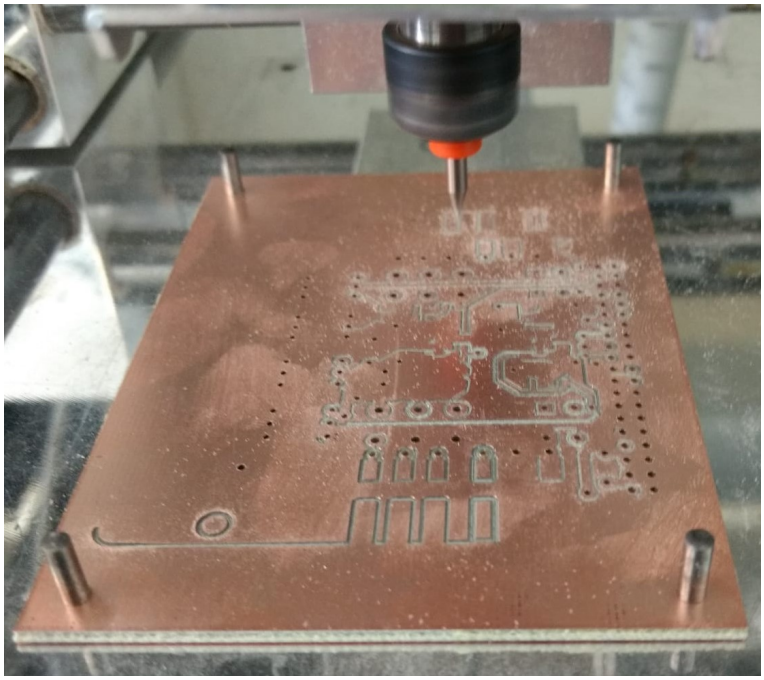

Figure A.1: Board milling process.

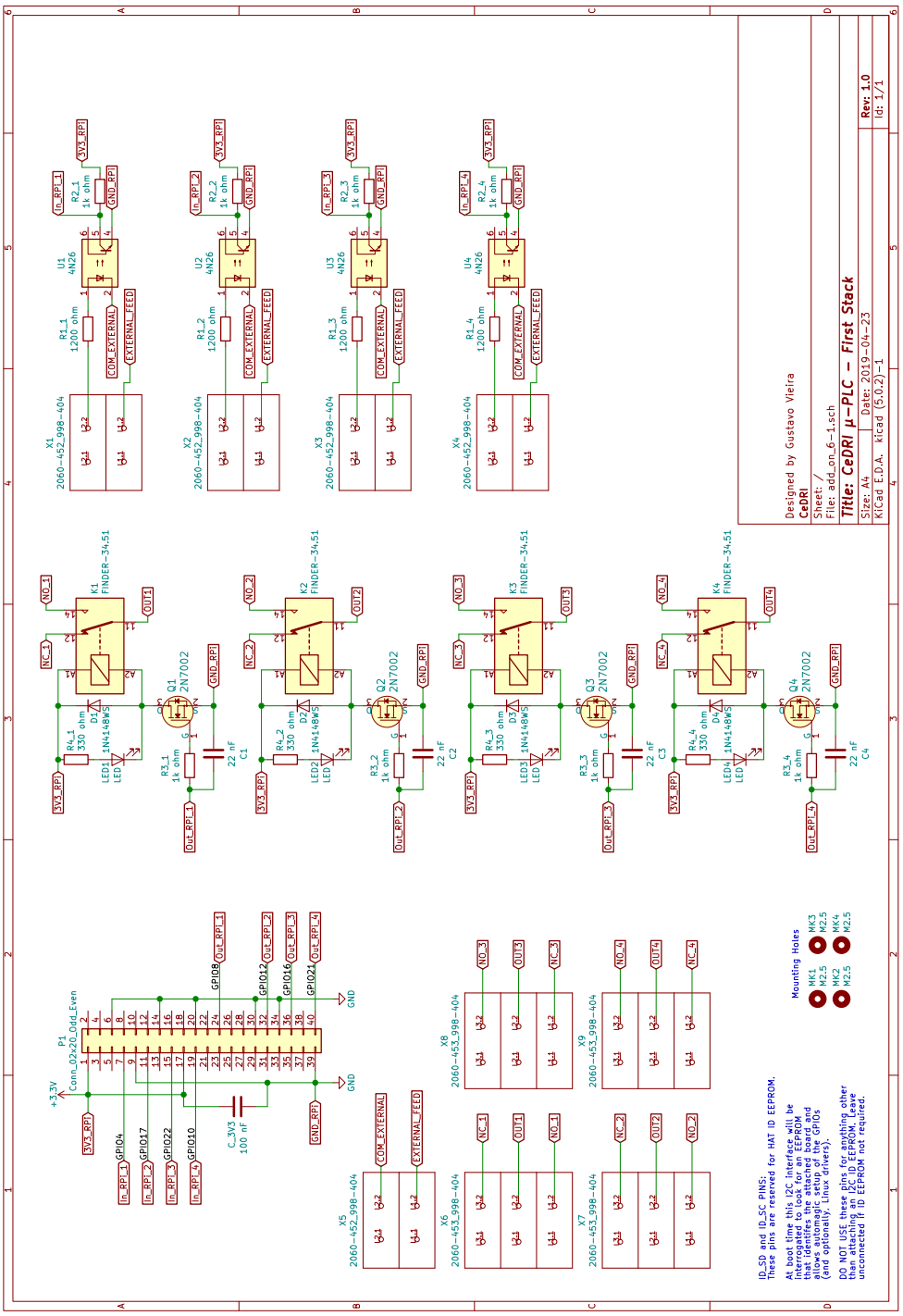

Figure A.2: First stack schematic.

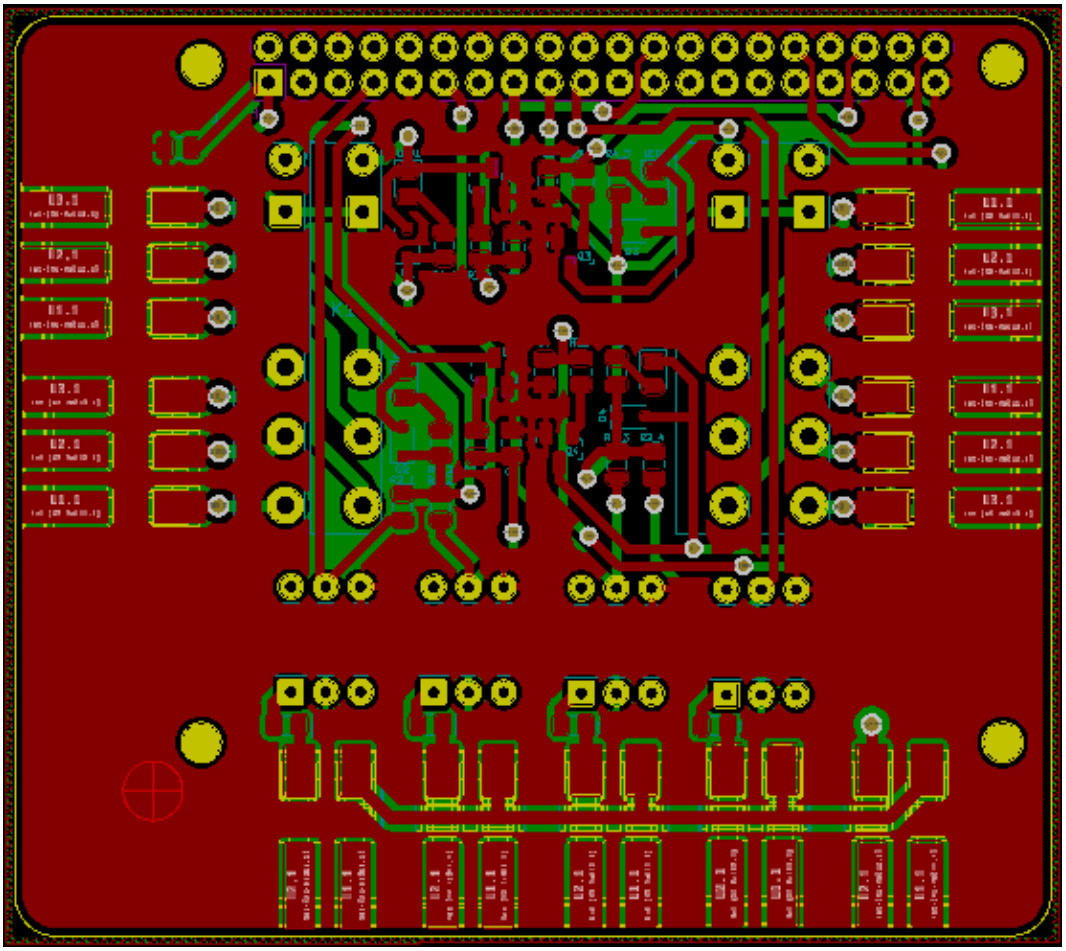

Figure A.3: Top view of the first Shield

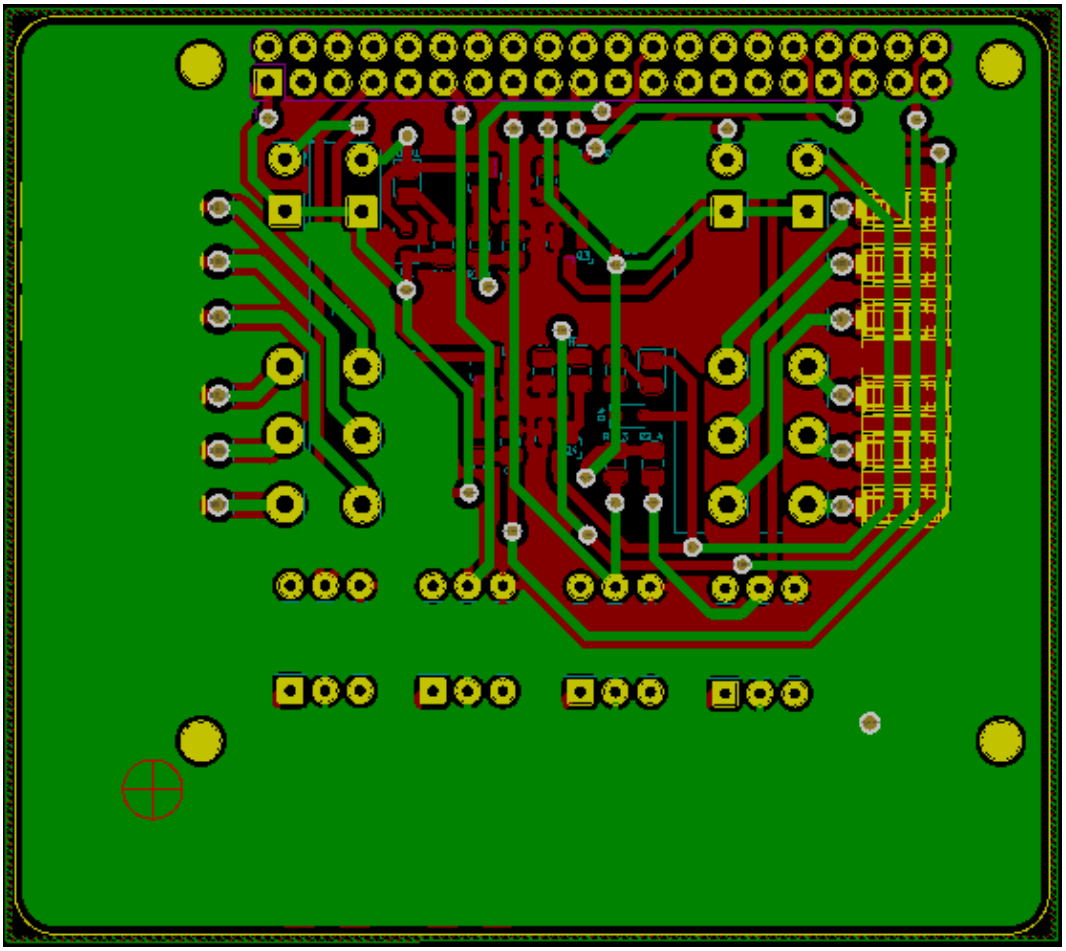

Figure A.4: Bottom view of the first Shield.

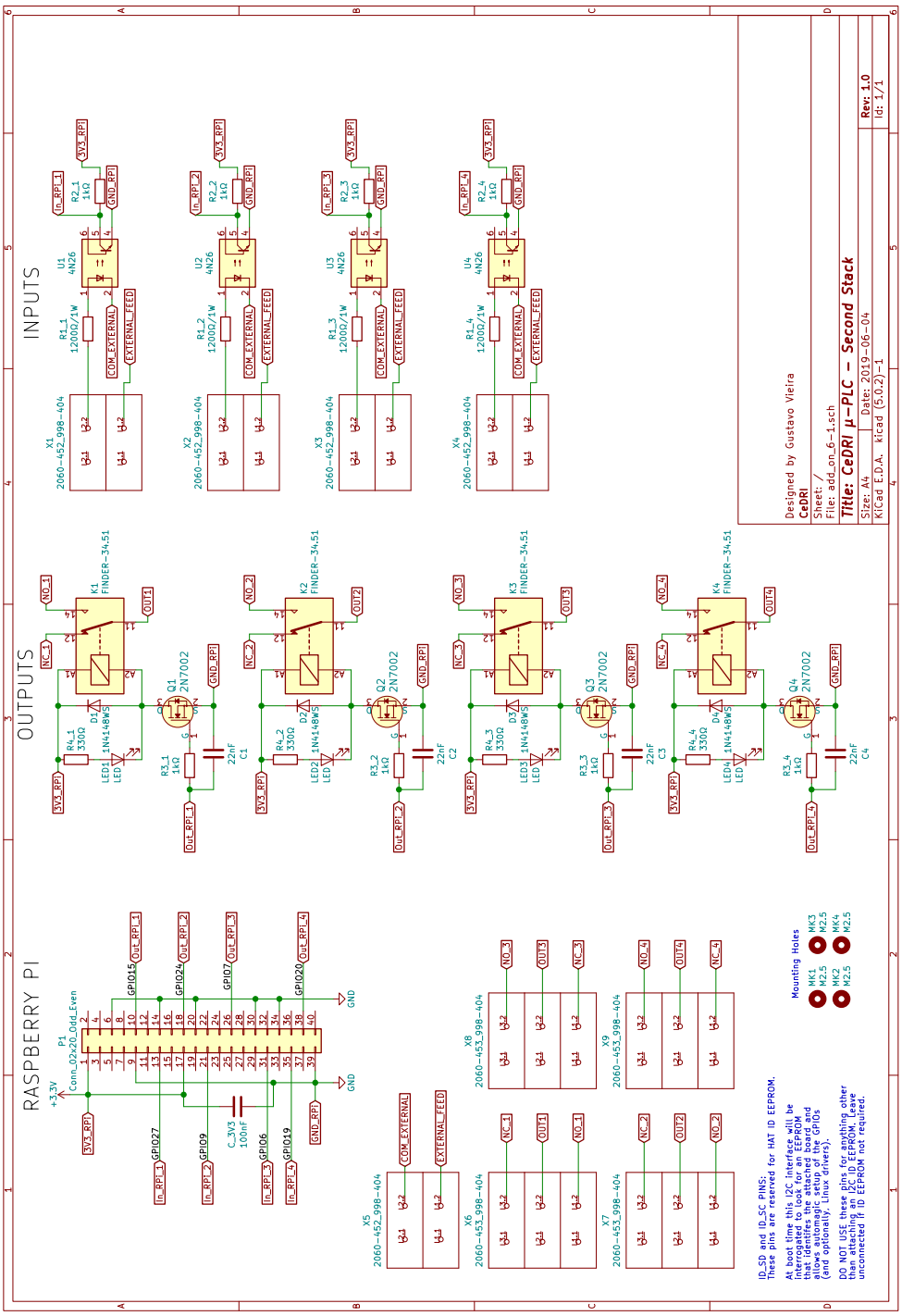

Figure A.5: Second stack schematic.

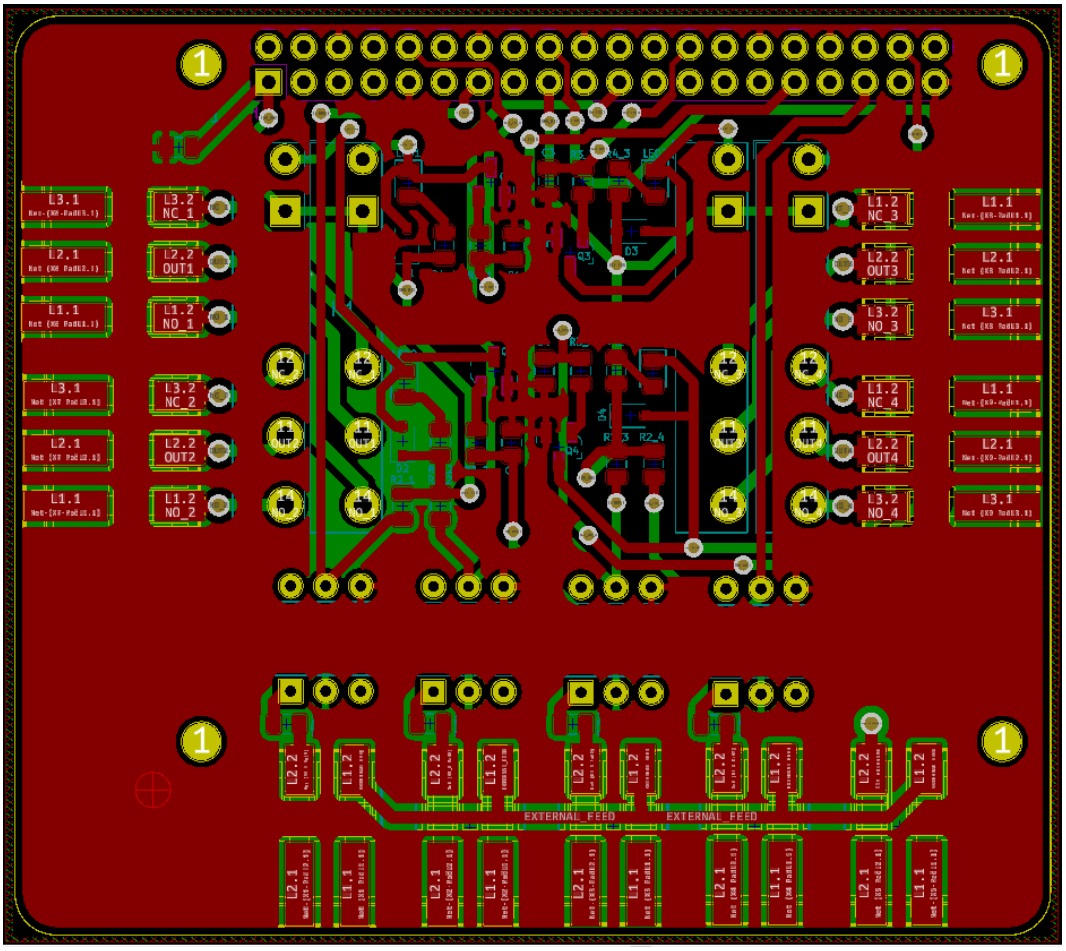

Figure A.6: Top view of the second Shield

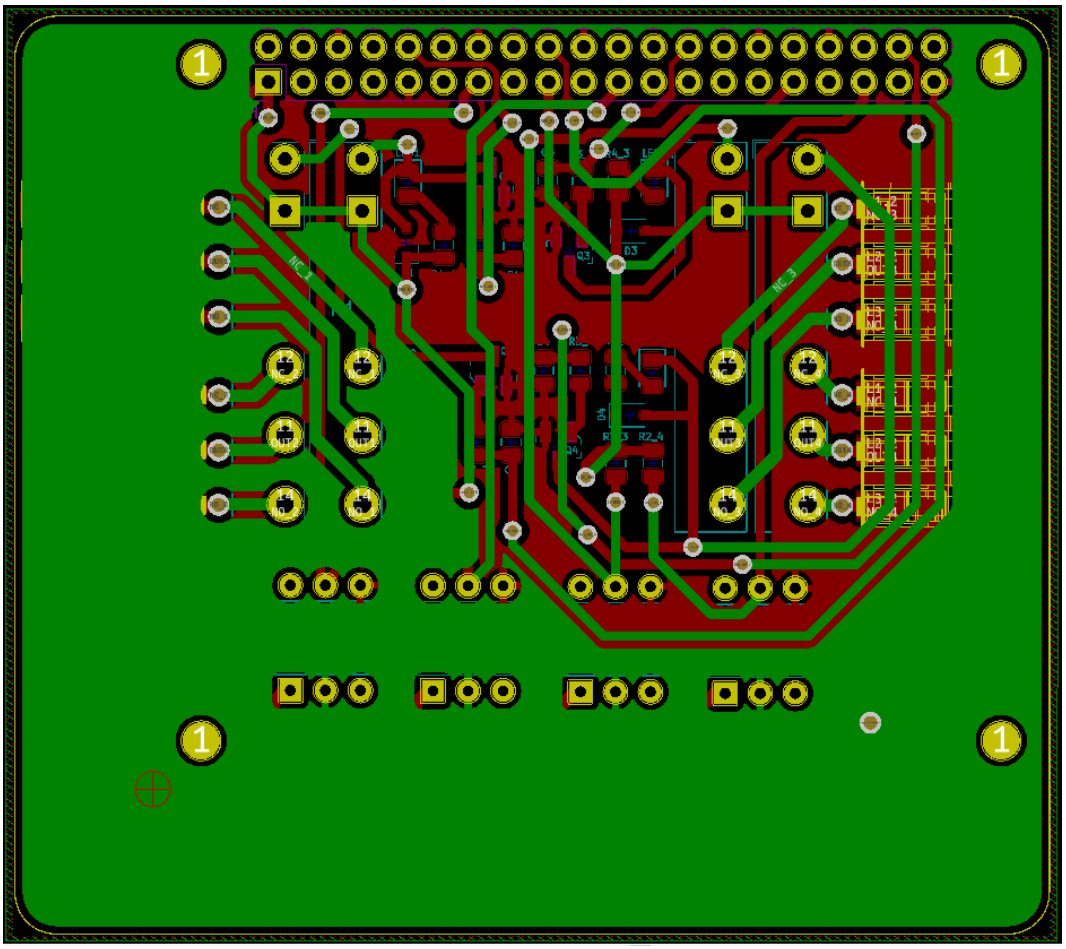

Figure A.7: Bottom view of the second Shield.

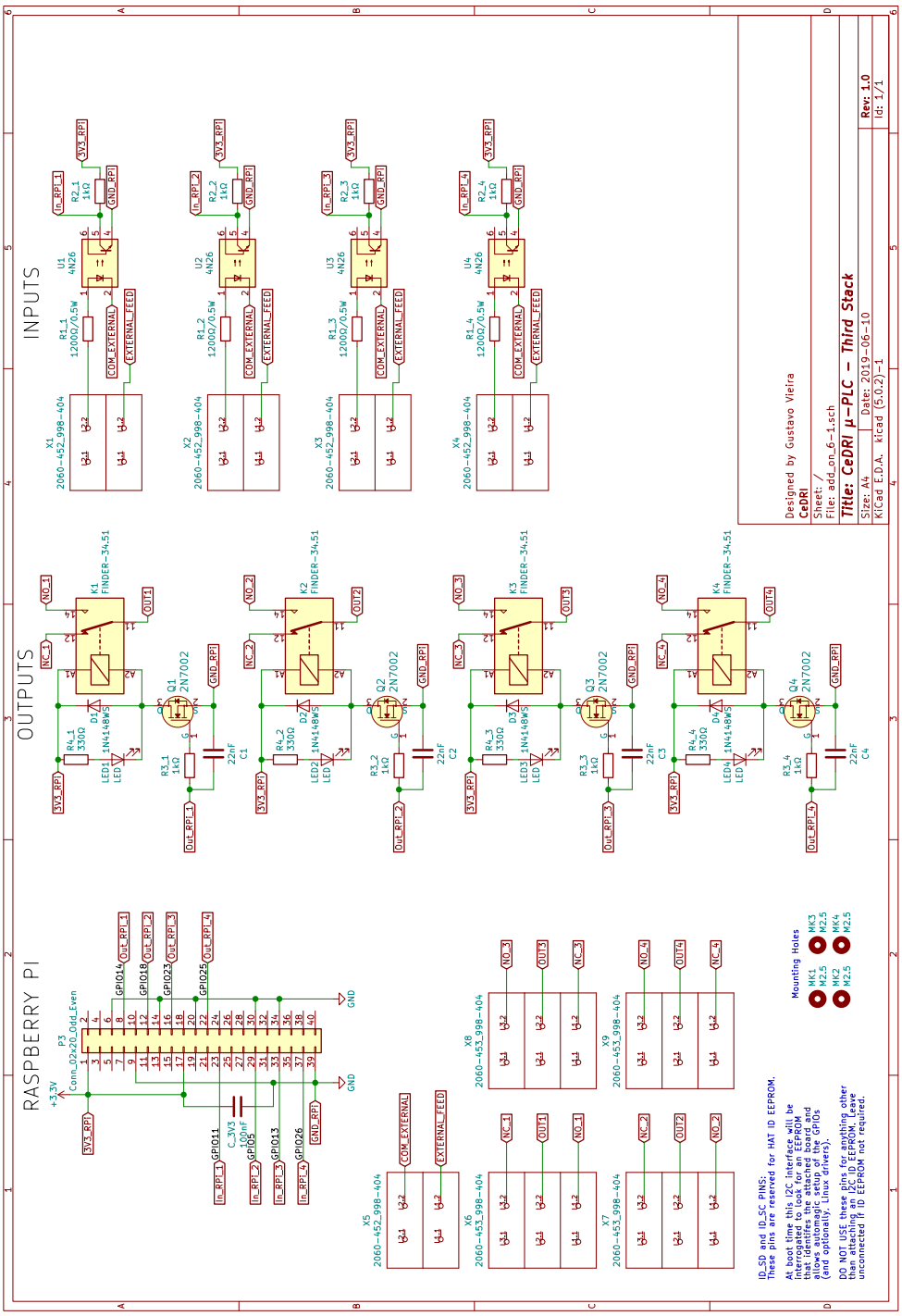

Figure A.8: Third stack schematic.

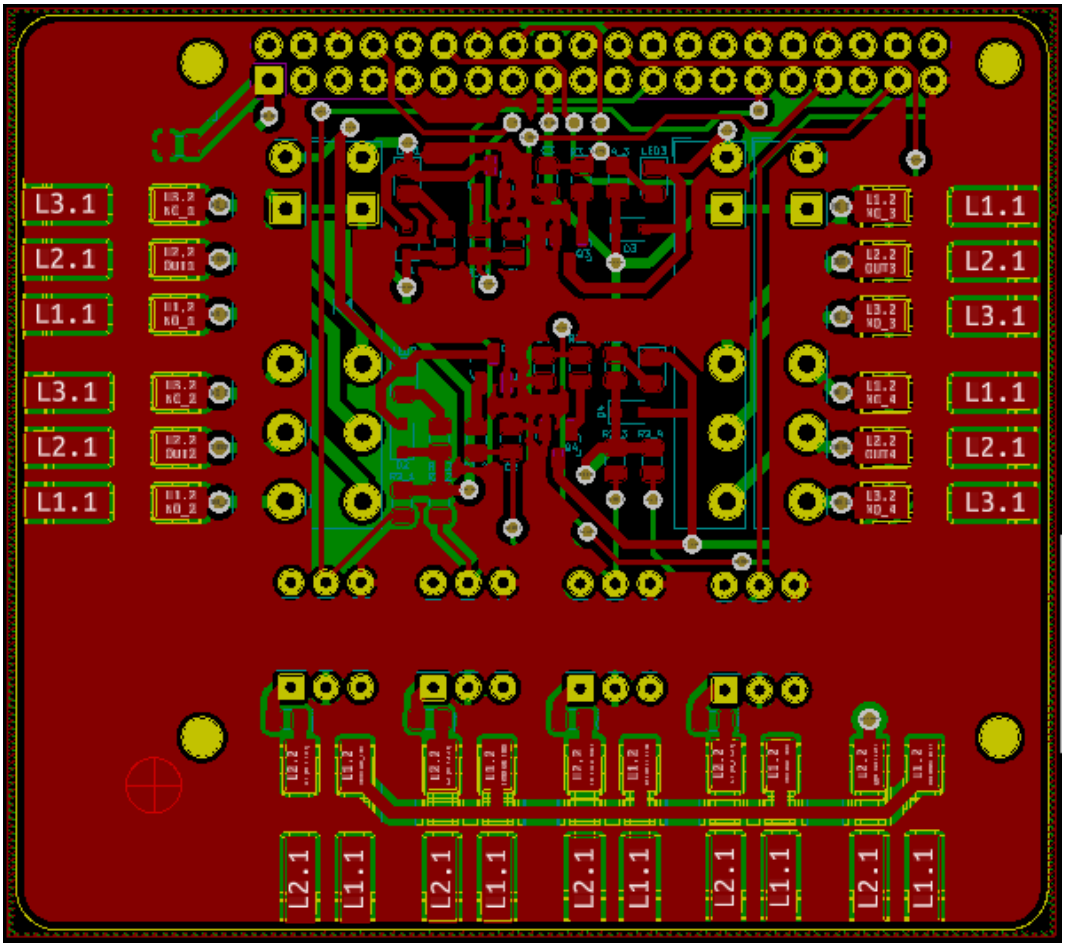

Figure A.9: Top view of the third Shield

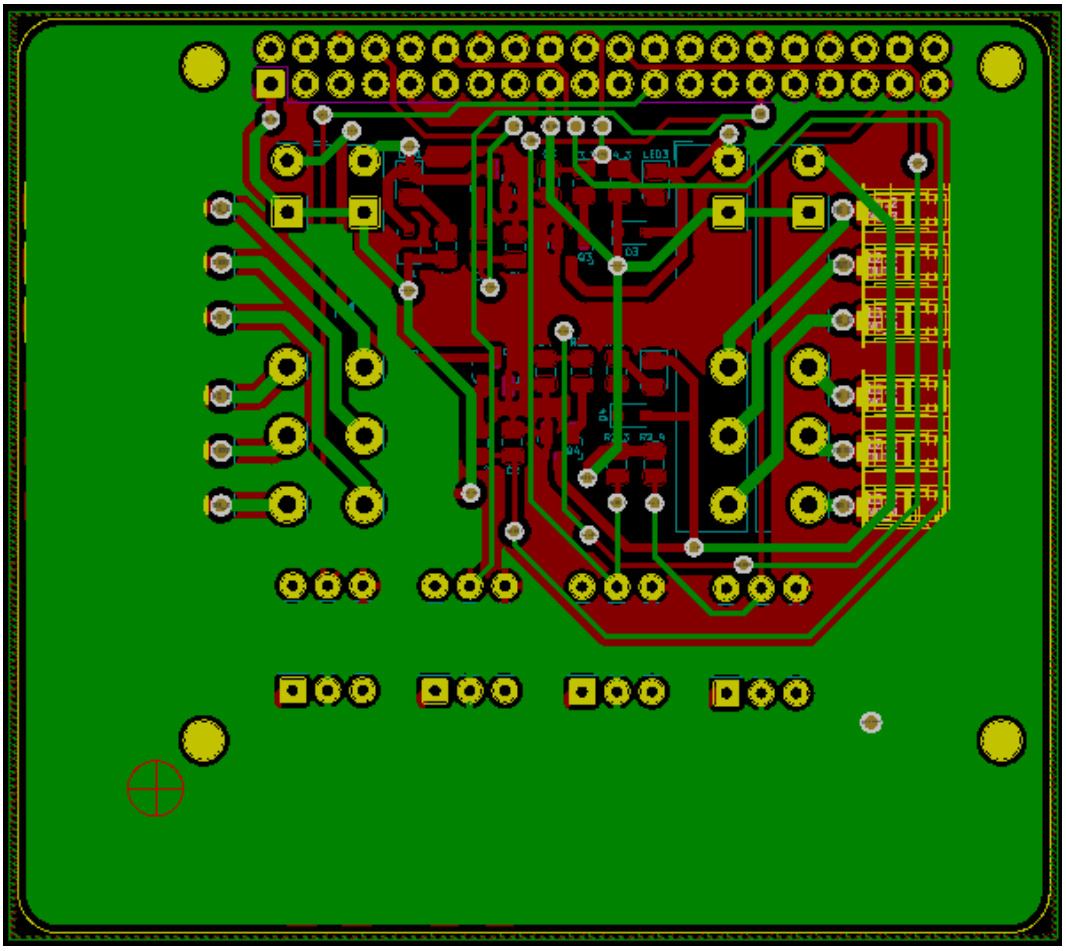

Figure A.10: Bottom view of the third Shield.

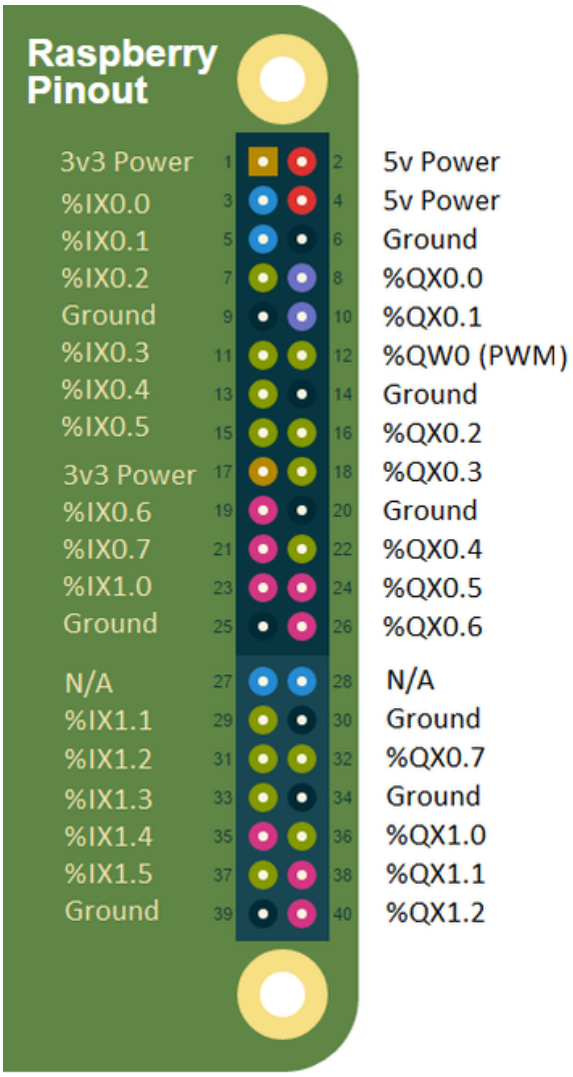

Figure A.11: OpenPLC pinout for the Raspberry Pi device.

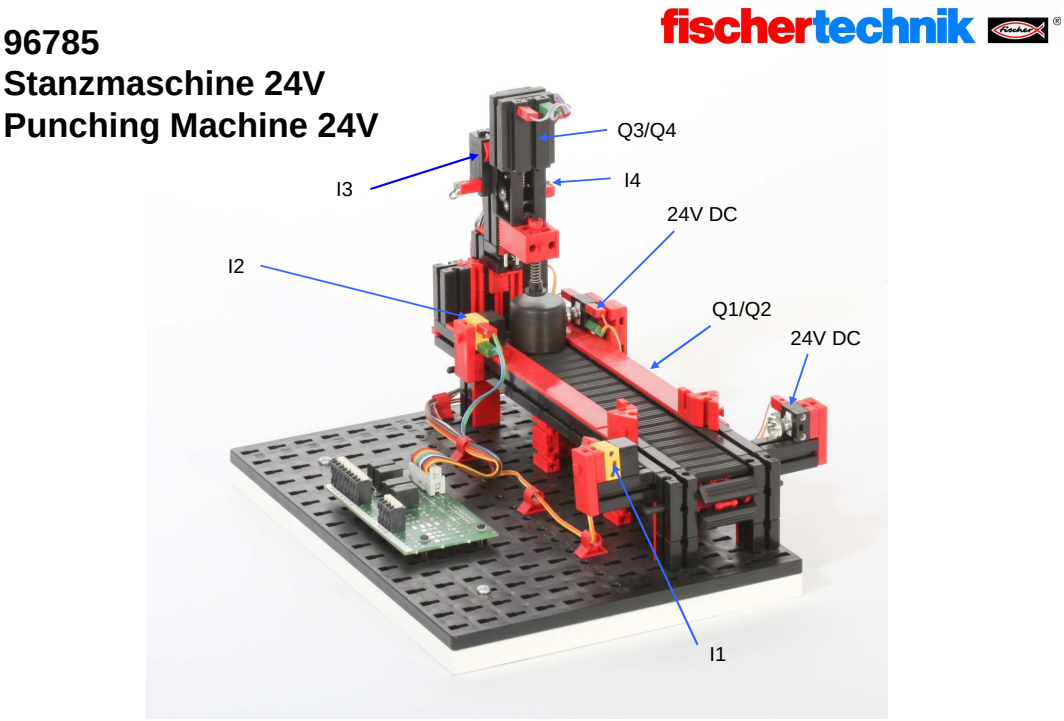

Figure A.12: I/O signaling on the Punching Machine industrial hardware simulation.

fischertechnik

Belegungsplan für Stanzmaschine mit Transportband 24V (Art.-Nr. 96785)
Circuit layout for Punching machine (item-no. 96785)

Klemme Nr. Terminal no.	Funktion Function	Eingang/Ausgang Input/Output
1	Stromversorgung (+) Aktoren power supply (+) actuators	24V DC
2	Stromversorgung (+) Sensoren power supply (+) sensors	24V DC
3	Stromversorgung (-) power supply (-)	0V
4	Stromversorgung (-) power supply (-)	0V
5	Fototransistor Ein/Auslagerungsstation Phototransistor goods in / goods out	I1
6	Fototransistor Stanze Phototransistor punching machine	I2
7	Taster Stanze oben Switch punching machine up	I3
8	Taster Stanze unten Switch punching machine down	I4
15	Motor Transportband vor Motor conveyor belt forward	Q1
16	Motor Transportband zurück Motor conveyor belt backward	Q2
17	Motor Stanze nach oben Motor punching machine up	Q3
18	Motor Stanze nach unten Motor punching machine down	Q4

Figure A.13: Punching machine I/O specifications.

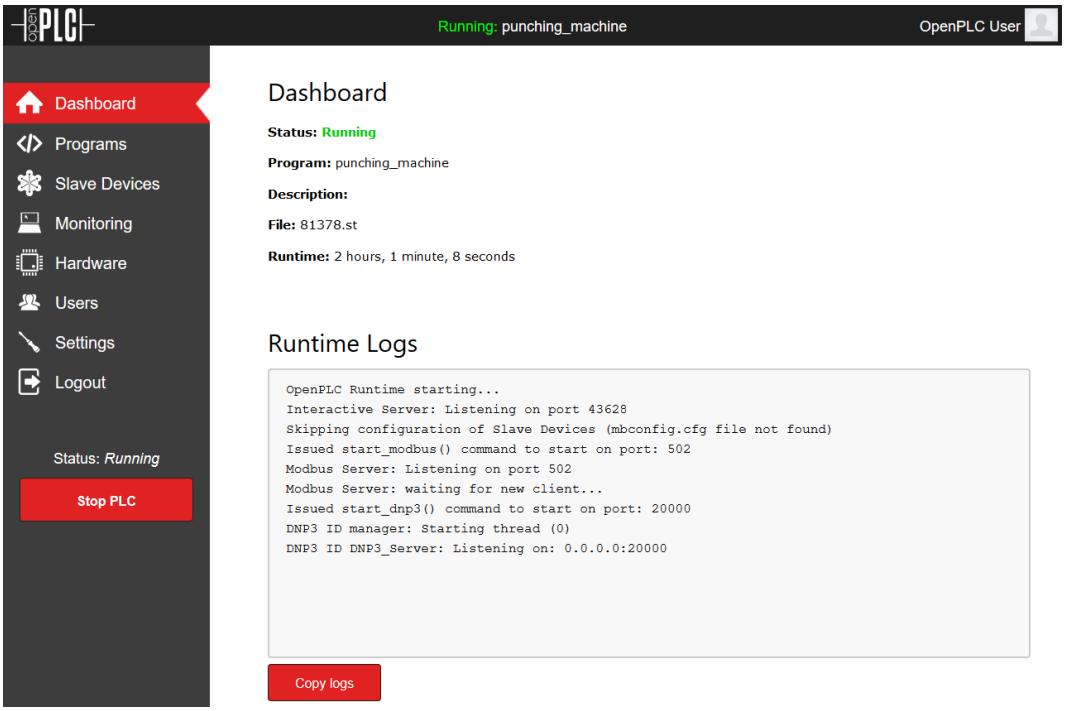

Figure A.14: The Industrial Controller kept operating successfully.

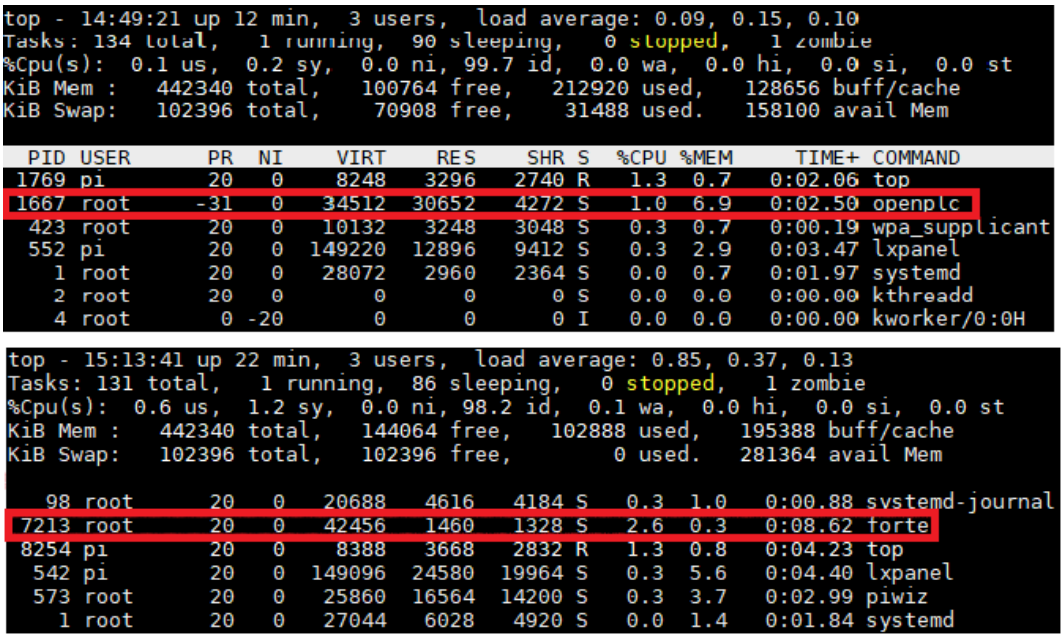

Figure A.15: Percentage of processor and RAM usage in Raspberry 2 A+.

Appendix B

Apendix B

Listing B.1: Testing the operation of Raspberry GPIOs via Python.

```
#import GPIO and time
import RPi.GPIO as GPIO
import time

# set GPIO numbering mode and define output pins
GPIO.setmode(GPIO.BOARD)
#GPIO.setup(22,GPIO.OUT, initial=GPIO.HIGH)
#GPIO.setup(26,GPIO.OUT, initial=GPIO.HIGH)
GPIO.setup(36,GPIO.OUT, initial=GPIO.HIGH)
GPIO.setup(40,GPIO.OUT, initial=GPIO.HIGH)
#GPIO.setup(12,GPIO.OUT, initial=GPIO.HIGH)
#GPIO.setup(8,GPIO.OUT, initial=GPIO.HIGH)
#GPIO.setup(10,GPIO.OUT, initial=GPIO.HIGH)
#GPIO.setup(16,GPIO.OUT, initial=GPIO.HIGH)
#GPIO.setup(18,GPIO.OUT, initial=GPIO.HIGH)
GPIO.setup(24,GPIO.OUT, initial=GPIO.HIGH)
GPIO.setup(26,GPIO.OUT, initial=GPIO.HIGH)
```

```
#GPIO.setup(32,GPIO.OUT, initial=GPIO.HIGH)
GPIO.setup(38,GPIO.OUT, initial=GPIO.HIGH)
#GPIO.setup(11,GPIO.IN)
#GPIO.setup(13,GPIO.IN)
#GPIO.setup(15,GPIO.IN)
#GPIO.setup(19,GPIO.IN)
#GPIO.setup(21,GPIO.IN)
#GPIO.setup(3,GPIO.IN)
#GPIO.setup(5,GPIO.IN)
#GPIO.setup(7,GPIO.IN)
#GPIO.setup(23,GPIO.IN)
GPIO.setup(29,GPIO.IN)
GPIO.setup(31,GPIO.IN)
GPIO.setup(33,GPIO.IN)
GPIO.setup(35,GPIO.IN)
GPIO.setup(37,GPIO.IN)


# cycle those relays

try:
    while True:
        if not(GPIO.input(29)):
            GPIO.output(24,GPIO.LOW)
            time.sleep(1)
            GPIO.output(24, GPIO.HIGH)
        elif not(GPIO.input(31)):
            GPIO.output(26,GPIO.LOW)
            time.sleep(1)
```

```
            GPIO.output(26, GPIO.HIGH)
        elif not(GPIO.input(33)):
            GPIO.output(36, GPIO.LOW)
            time.sleep(1)
            GPIO.output(36, GPIO.HIGH)
        elif not(GPIO.input(35)):
            GPIO.output(38, GPIO.LOW)
            time.sleep(1)
            GPIO.output(38, GPIO.HIGH)
        elif not(GPIO.input(37)):
            GPIO.output(40, GPIO.LOW)
            time.sleep(1)
            GPIO.output(40, GPIO.HIGH)
        time.sleep(.05)

finally:
#cleanup the GPIO before finishing
        GPIO.cleanup()
```