

**FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO**



# **Electronics Based VPGM Controller Board**

**Tiago José Ferreira Mendonça**

**FOR JURY EVALUATION**

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Advisor: Prof. Hélio Mendes de Sousa Mendonça

Co-Advisor: Prof. António Paulo Gomes Mendes Moreira

25 de Junho de 2018



# Resumo

O CERN centra a sua atividade na realização de pesquisas nucleares visando descobrir os alicerces que sustentam o Universo. Tais experiências decorrem em regime controlado com recurso a aceleradores de partículas. Um dos módulos fulcrais para o sucesso das mesmas é o sistema de vácuo. Este é assegurado por diversos equipamentos, incluindo bombas e válvulas, que são controlados por um PLC. Porém, apesar da sua fiabilidade, robustez e longevidade, uma arquitetura de controlo desta natureza acarreta alguns inconvenientes, tais como perda de suporte, dificuldade de integração com novos modelos e um considerável custo de aquisição.

Nesta dissertação é explorada a viabilidade e fiabilidade de substituir um PLC por uma arquitetura sustentada em SBC, nomeadamente a versão mais recente da *BeagleBone*. Esta unidade é usada como módulo de processamento, sendo a aquisição física dos sinais do grupo de vácuo feita através de uma placa protótipo concebida especificamente para o propósito. A interface entre esta e a unidade de processamento é feita por comunicação série, nomeadamente SPI. A arquitetura de controlo assenta num soft-PLC, CODESYS, e está de acordo com a norma IEC 61131. Um fator digno de destaque neste projeto prende-se com a utilização de princípios de memória partilhada para transmissão de informação entre o CODESYS e o programa inerente à configuração do protocolo SPI. O desempenho do sistema é testado tanto a nível funcional como temporal, comprovando-se a adequabilidade à sua aplicação final. Embora o projeto seja desenvolvido para um propósito específico, é aferida a viabilidade para que uma arquitetura desta natureza seja extrapolada para outros contextos de automação industrial, sendo também ponderada a eventual utilização de um microcontrolador em detrimento de uma SBC. Finalmente, são também desenvolvidos testes de comunicação com um escravo *Profibus*, de forma a corroborar o correto funcionamento de uma pilha protocolar oriunda de um projeto prévio e abrindo o horizonte para uma futura integração com o sistema.



# Abstract

CERN centers its activity on nuclear research aiming to discover the building blocks in which relies the universe. Such experiments occur on a controlled environment taking advantage of particles accelerators. Pumping system is one of most important modules to its operation. This relies on several devices, including valves and pumps, that are controlled by a PLC. Despite its reliability, robustness and long term operation, this kind of architecture implies some drawbacks, like loss of support, difficult integration with new models and high acquisition cost.

On this thesis it is explored the feasibility and reliability of replacing a PLC by a SBC-based architecture, namely the newer model of *BeagleBone*. This element is used as processing unit, being the physical acquisition of vacuum group signals performed by a prototype board design from scratch. The interface with core unit relies on serial communication, namely SPI protocol. The control architecture is developed on a soft-PLC, CODESYS, and is compliant with IEC 61131 standard. One remarkable feature of this project is that the data exchange between control program (running on CODESYS) and SPI interface script is made according with shared memory principles. System's performance is evaluated on functional and temporal points of view, being confirmed that fits system requirements. Although the project is adjusted for a single application, the final results support the possibility of extrapolating it for different industrial automation scenarios, being also performed a comparison with a system based on a microcontroller. Additionally, it is also showed some communication tests with a *Profibus* slave based on an open-source stack developed on a previous project, being check its correct operation and preparing future integration with the system.



# Acknowledgements

In first place, I want to thank to professors Hélio Mendonça and António Paulo Moreira for the support, helpfulness, recommendations and the knowledge transmitted during all thesis stages. I also thank the financial support provided by INESC TEC during the internship at CERN. Still on INESC crowd, a special acknowledgment to José Carlos Azevedo for all the effort invested on project hardware development.

I would like to say also thank you to Rodrigo Ferreira for taking care of me during my swiss adventure, for pushing me to go further even when everything was burnt, literally, and, not only for the technical, but also personal advices that keep me on the right track. Also, an honorable mention to remaining *Bâtiment 30* group for the hospitality and easy integration.

Lastly, but not least, I thank to my parents and brother for carrying me when I still was an open-loop system and for always providing the rights inputs and outputs that allow to tune my personal controller. I hope that the resulting impulse response makes them proud. And, of course, for being the second family, I appreciate friends company, namely the ones that always forced me to abandon my computer screen at Friday's nights and to go out from engineering ecosystem.

Tiago Mendonça



*“There are known knowns and known unknowns,  
but what we should be worried about most is the unknown unknowns.”*

Gary Marcus



# Contents

|          |                                                   |           |
|----------|---------------------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                               | <b>1</b>  |
| 1.1      | Problem statement, goals and motivation . . . . . | 1         |
| 1.1.1    | Project details and objectives . . . . .          | 2         |
| 1.2      | Document structure . . . . .                      | 3         |
| <b>2</b> | <b>State of the art</b>                           | <b>5</b>  |
| 2.1      | Particles accelerators . . . . .                  | 5         |
| 2.1.1    | Basic principles . . . . .                        | 5         |
| 2.1.2    | CERN accelerator complex . . . . .                | 7         |
| 2.2      | Vacuum system . . . . .                           | 8         |
| 2.2.1    | Vacuum principles and instrumentation . . . . .   | 8         |
| 2.2.2    | CERN pumping groups . . . . .                     | 11        |
| 2.2.3    | VPGM control . . . . .                            | 13        |
| 2.3      | Programmable logic controller (PLC) . . . . .     | 15        |
| 2.3.1    | PLC architecture . . . . .                        | 15        |
| 2.3.2    | IEC 61131 . . . . .                               | 16        |
| 2.3.3    | Input module . . . . .                            | 20        |
| 2.3.4    | Output module . . . . .                           | 21        |
| 2.3.5    | Source and sink . . . . .                         | 22        |
| 2.3.6    | Communication module . . . . .                    | 23        |
| 2.4      | Soft-PLC . . . . .                                | 28        |
| 2.5      | Single-board computer (SBC) . . . . .             | 29        |
| 2.6      | Conclusions . . . . .                             | 31        |
| <b>3</b> | <b>Hardware Architecture</b>                      | <b>33</b> |
| 3.1      | Hardware overview . . . . .                       | 33        |
| 3.2      | PCB modules design . . . . .                      | 36        |
| 3.2.1    | Digital inputs . . . . .                          | 37        |
| 3.2.2    | Analog inputs . . . . .                           | 41        |
| 3.2.3    | Digital outputs . . . . .                         | 46        |
| 3.2.4    | Power supply . . . . .                            | 49        |
| 3.3      | PCB considerations . . . . .                      | 51        |
| 3.4      | Serial interface . . . . .                        | 55        |
| 3.5      | <i>Profibus</i> interface . . . . .               | 57        |
| 3.5.1    | <i>Profibus</i> connector driver . . . . .        | 59        |
| 3.6      | Conclusions . . . . .                             | 60        |

|          |                                                |            |
|----------|------------------------------------------------|------------|
| <b>4</b> | <b>Software Architecture</b>                   | <b>63</b>  |
| 4.1      | Software overview . . . . .                    | 63         |
| 4.2      | Control process software . . . . .             | 65         |
| 4.2.1    | VPGM process . . . . .                         | 66         |
| 4.2.2    | Human-machine interface . . . . .              | 81         |
| 4.3      | SPI software . . . . .                         | 90         |
| 4.3.1    | <i>Serializer</i> serial interface . . . . .   | 90         |
| 4.3.2    | <i>Thermocouple</i> serial interface . . . . . | 91         |
| 4.3.3    | ADS serial interface . . . . .                 | 92         |
| 4.3.4    | Output driver serial interface . . . . .       | 94         |
| 4.3.5    | Implemented Script . . . . .                   | 95         |
| 4.4      | Scripts communication . . . . .                | 96         |
| 4.5      | <i>Profibus</i> stack . . . . .                | 98         |
| 4.6      | Conclusions . . . . .                          | 100        |
| <b>5</b> | <b>Tests and Results</b>                       | <b>101</b> |
| 5.1      | Process operation . . . . .                    | 101        |
| 5.2      | Time constraints . . . . .                     | 102        |
| 5.2.1    | Kernel latency . . . . .                       | 103        |
| 5.2.2    | System time requirements . . . . .             | 104        |
| 5.3      | Hardware robustness . . . . .                  | 107        |
| 5.4      | HMI . . . . .                                  | 110        |
| 5.5      | <i>Profibus</i> tests . . . . .                | 112        |
| 5.6      | Conclusions . . . . .                          | 114        |
| <b>6</b> | <b>Conclusions and future work</b>             | <b>117</b> |
| 6.1      | Conclusions . . . . .                          | 117        |
| 6.2      | Microcontroller vs SBC . . . . .               | 118        |
| 6.3      | Future work . . . . .                          | 119        |
| <b>A</b> | <b>Real time systems</b>                       | <b>121</b> |
| A.1      | Systems classification . . . . .               | 121        |
| A.2      | System latency measurement . . . . .           | 122        |
| <b>B</b> | <b>Shell script configuration</b>              | <b>123</b> |
| <b>C</b> | <b>Group I/O connections</b>                   | <b>125</b> |
| <b>D</b> | <b>Group control process test</b>              | <b>127</b> |
| <b>E</b> | <b>PCB electric schematic</b>                  | <b>129</b> |
|          | <b>References</b>                              | <b>143</b> |

# List of Figures

|      |                                                                         |    |
|------|-------------------------------------------------------------------------|----|
| 2.1  | Cavities on the beam pipe [1]. . . . .                                  | 6  |
| 2.2  | Magnetes arrangement to confine the particle beam [1]. . . . .          | 7  |
| 2.3  | CERN accelerator complex [2]. . . . .                                   | 8  |
| 2.4  | Pressure ranges classification [3]. . . . .                             | 9  |
| 2.5  | Useful range of standard pumps [4]. . . . .                             | 10 |
| 2.6  | Pressure gauges application [4]. . . . .                                | 11 |
| 2.7  | Basic structure of a turbo-molecular pumping group [5]. . . . .         | 12 |
| 2.8  | Front and back view of a CERN mobile pumping group. . . . .             | 13 |
| 2.9  | Operation modes of VPGM [5]. . . . .                                    | 14 |
| 2.10 | Hardware architecture [5]. . . . .                                      | 15 |
| 2.11 | PLC architecture [6]. . . . .                                           | 16 |
| 2.12 | IEC 61131-3 programming languages [7]. . . . .                          | 17 |
| 2.13 | PLC software architecture [8]. . . . .                                  | 18 |
| 2.14 | PLC scan cycle [9]. . . . .                                             | 20 |
| 2.15 | PLC DC input [10]. . . . .                                              | 21 |
| 2.16 | PLC output (adapted from [11]). . . . .                                 | 22 |
| 2.17 | Input and output sourcing unit [10]. . . . .                            | 23 |
| 2.18 | Input and output sinking unit [10]. . . . .                             | 23 |
| 2.19 | SPI multiple slaves communication [12]. . . . .                         | 25 |
| 2.20 | Daisy chain configuration [12]. . . . .                                 | 25 |
| 2.21 | Profibus protocol stack [13]. . . . .                                   | 26 |
| 2.22 | <i>Profibus</i> character codification [14]. . . . .                    | 27 |
| 2.23 | <i>Profibus DP</i> slave state machine [14]. . . . .                    | 27 |
| 3.1  | <i>BeagleBone Green</i> overview [15]. . . . .                          | 34 |
| 3.2  | <i>BeagleBone</i> LCD display cap [16]. . . . .                         | 35 |
| 3.3  | RS485-USB adapter. . . . .                                              | 35 |
| 3.4  | Hardware architecture. . . . .                                          | 36 |
| 3.5  | <i>Serializer</i> input protection circuit (adapted from [17]). . . . . | 39 |
| 3.6  | <i>Serializer</i> input isolation circuit. . . . .                      | 40 |
| 3.7  | ADS8665 pinout [18]. . . . .                                            | 41 |
| 3.8  | ADS8665 acquisition circuit. . . . .                                    | 44 |
| 3.9  | MAX31856 pinout [19]. . . . .                                           | 45 |
| 3.10 | MAX31856 acquisition circuit. . . . .                                   | 46 |
| 3.11 | VNI8200XP pinout [20]. . . . .                                          | 47 |
| 3.12 | VNI8200XP configuration. . . . .                                        | 49 |
| 3.13 | Power supply protection (adapted from [17]). . . . .                    | 50 |
| 3.14 | Power supply circuit. . . . .                                           | 51 |

|      |                                                                                                                                   |     |
|------|-----------------------------------------------------------------------------------------------------------------------------------|-----|
| 3.15 | PLC upper and side view with respective dimensions, in mm. . . . .                                                                | 52  |
| 3.16 | Analogical and digital grounds separation. . . . .                                                                                | 53  |
| 3.17 | Wires connectors [21]. . . . .                                                                                                    | 53  |
| 3.18 | LCD board. . . . .                                                                                                                | 54  |
| 3.19 | PCB board. . . . .                                                                                                                | 55  |
| 3.20 | Serial bus buffer configuration. . . . .                                                                                          | 57  |
| 3.21 | MAX485 internal configuration and typical application [22]. . . . .                                                               | 58  |
| 3.22 | Termination resistors and failsafe circuit (adapted from [23]). . . . .                                                           | 59  |
| 4.1  | Software architecture. . . . .                                                                                                    | 65  |
| 4.2  | Valve FB. . . . .                                                                                                                 | 68  |
| 4.3  | Turbo-molecular pump FB. . . . .                                                                                                  | 69  |
| 4.4  | Primary pump FB. . . . .                                                                                                          | 71  |
| 4.5  | UPS FB. . . . .                                                                                                                   | 71  |
| 4.6  | Vacuum gauge FB. . . . .                                                                                                          | 72  |
| 4.7  | Bake-out FB. . . . .                                                                                                              | 72  |
| 4.8  | Group state machine [24]. . . . .                                                                                                 | 73  |
| 4.9  | Transitions between modes [24]. . . . .                                                                                           | 74  |
| 4.10 | Devices condition on each step [24]. . . . .                                                                                      | 75  |
| 4.11 | Group errors hierarchy (adapted from [24]). . . . .                                                                               | 76  |
| 4.12 | Execution cycle. . . . .                                                                                                          | 81  |
| 4.13 | Synoptics graphical disposition. . . . .                                                                                          | 83  |
| 4.14 | Valves synoptic (label not included). . . . .                                                                                     | 83  |
| 4.15 | Valves command mode flags. . . . .                                                                                                | 84  |
| 4.16 | Primary pump synoptic on off state with error flag enabled (left) and turbo pump synoptic, when on nominal state (right). . . . . | 84  |
| 4.17 | Group state synoptic before and after an error occurrence. . . . .                                                                | 85  |
| 4.18 | <i>Basic Mode</i> frame. . . . .                                                                                                  | 86  |
| 4.19 | <i>Login</i> frame. . . . .                                                                                                       | 86  |
| 4.20 | <i>Expert Mode</i> frame. . . . .                                                                                                 | 87  |
| 4.21 | <i>Manual Control</i> frame. . . . .                                                                                              | 88  |
| 4.22 | <i>Settings</i> frame. . . . .                                                                                                    | 89  |
| 4.23 | Sequence of status bits in <i>Serializer</i> [17]. . . . .                                                                        | 90  |
| 4.24 | Calculation of parity bits [20]. . . . .                                                                                          | 95  |
| 4.25 | Script operation logic . . . . .                                                                                                  | 96  |
| 4.26 | Data transmission between programs over shared memory. . . . .                                                                    | 97  |
| 4.27 | <i>Profibus</i> stack architecture (adapted from [25]). . . . .                                                                   | 98  |
| 5.1  | Minimum system response. . . . .                                                                                                  | 106 |
| 5.2  | Maximum system response. . . . .                                                                                                  | 106 |
| 5.3  | System response time. . . . .                                                                                                     | 107 |
| 5.4  | PCB enclosure box design. . . . .                                                                                                 | 110 |
| 5.5  | Terminal block and screwed pluggable connector (adapted from [21]). . . . .                                                       | 110 |
| 5.6  | HMI display. . . . .                                                                                                              | 111 |
| 5.7  | 7" HDMI display [26]. . . . .                                                                                                     | 112 |
| 5.8  | Modular <i>Profibus</i> slave user for tests. . . . .                                                                             | 112 |
| 5.9  | Example of a successful communication with a <i>Profibus</i> slave. . . . .                                                       | 113 |

# List of Tables

|     |                                                                            |     |
|-----|----------------------------------------------------------------------------|-----|
| 2.1 | Action qualifiers [27]. . . . .                                            | 19  |
| 2.2 | SPI bus mode [28] . . . . .                                                | 24  |
| 2.3 | <i>BeagleBone Green</i> and <i>Raspberry Pi 3</i> specs [15][29] . . . . . | 30  |
| 3.1 | Electrical characteristics of PLC input module [30] . . . . .              | 37  |
| 3.2 | Electrical characteristics of <i>serializer</i> [17] . . . . .             | 37  |
| 3.3 | Input filter configuration [17] . . . . .                                  | 38  |
| 3.4 | PLC digital output module characteristics [30]. . . . .                    | 46  |
| 3.5 | VNI8200XP characteristics [20]. . . . .                                    | 47  |
| 4.1 | Primary pump faults hierarchy . . . . .                                    | 70  |
| 4.2 | <i>Thermocouple</i> registers configuration. . . . .                       | 92  |
| 4.3 | ADS8665 registers configuration. . . . .                                   | 93  |
| 4.4 | Master-to-slave 16-bit frame [20]. . . . .                                 | 94  |
| 4.5 | Slave-to-master 16-bit frame [20]. . . . .                                 | 94  |
| 5.1 | System latency measures. . . . .                                           | 103 |
| 5.2 | System latency measures with preemptive patch. . . . .                     | 104 |
| 5.3 | Process cycle time and <i>jitter</i> measures. . . . .                     | 105 |
| 5.4 | Devices tests checklist. . . . .                                           | 108 |
| 5.5 | Master-to-slave 8-bit frame [20]. . . . .                                  | 108 |
| 5.6 | Slave-to-master 8-bit frame [20]. . . . .                                  | 109 |
| 5.7 | Minimum timeout required per baud rate. . . . .                            | 114 |
| A.1 | <i>BeagleBone</i> system latency [31]. . . . .                             | 121 |
| C.1 | Output commands to control group devices. . . . .                          | 125 |
| C.2 | Input signals provided by group devices . . . . .                          | 126 |
| D.1 | Control process checklist. . . . .                                         | 128 |





# Abbreviations

|       |                                             |
|-------|---------------------------------------------|
| ADC   | Analog-to-Digital Converter                 |
| ARM   | Advanced RISC Machine                       |
| ASIC  | Application Specific Integrated Circuit     |
| CPU   | Central Processing Unit                     |
| DC    | Direct Current                              |
| DP    | Decentralized Peripherals                   |
| EMF   | Electromotive Force                         |
| FB    | Function Block                              |
| FBD   | Function Block Diagram                      |
| FDL   | Fieldbus Data Link                          |
| FMS   | Fieldbus Message Specification              |
| GPIO  | General Purpose Input/Output                |
| GSD   | General Station Description                 |
| HDMI  | High-Definition Multimedia Interface        |
| HMI   | Human-Machine Interface                     |
| I2C   | Inter-Integrated Circuit                    |
| IC    | Integrated Circuit                          |
| IEC   | International Electrotechnical Commission   |
| IL    | Instruction List                            |
| LCD   | Liquid Crystal Display                      |
| LHC   | Large Hadron Collider                       |
| MSB   | Most Significant Bit                        |
| OS    | Operating System                            |
| OSI   | Open System Interconnection                 |
| PA    | Process Automation                          |
| PC    | Personal Computer                           |
| PLC   | Programmable Logic Controller               |
| PRU   | Programmable Real-time Unit                 |
| RAM   | Random Access Memory                        |
| ROM   | Read-Only Memory                            |
| SBC   | Single-Board Computer                       |
| SCADA | Supervisory Control and Data Acquisition    |
| SDN   | Send Data with No acknowledge               |
| SFC   | Sequential Function Chart                   |
| SOC   | System On Chip                              |
| SPI   | Serial Peripheral Interface                 |
| SRD   | Send and Request Data with reply            |
| ST    | Structured Text                             |
| TMP   | Turbo-Molecular Pump                        |
| UART  | Universal Asynchronous Receiver-Transmitter |
| UHV   | Ultra-High Vacuum                           |
| UPS   | Uninterruptible Power Supply                |
| USB   | Universal Serial Bus                        |
| VPGM  | Mobile Vacuum Pumping Group                 |

# Chapter 1

## Introduction

The present chapter has the purpose to introduce the thesis "Electronics Based VPGM Controller Board". Then, on a first stage, [1.1](#), it is explained the context in which it is proposed, the imposed goals and motivation. On [1.2](#), it is exposed the document frame, structuring each one of the following chapters.

### 1.1 Problem statement, goals and motivation

At CERN, the European Organization for Nuclear Research, the particles' experiments are characterized by the increase of beams' speed on circulating chambers along large kilometers. In specific points of their path, the beams circulating in opposite directions are forced to collide and many particles and radiation are released. These products are registered by detectors and posteriorly analyzed. With these events, the scientists aim to go back in time and recreate the conditions that rule the space matter immediately after the Big Bang occurrence.

During their trajectory, particles describe a specific path and collide at specific points. There, to ensure the expected behavior, the beam pipe must be properly isolated from the outside and free of any suspended air molecule inside, since, if not verified, the beam would collide with undesired particles and diverge from its path. So, the chamber stays at very low-pressure state, called Ultra High Vacuum (UHV). This condition is result of successive pumping stages and the first one is assured by turbo-molecular pumping groups. Depending of particles accelerator, they can be fixed or mobile. In last case, the mobile pumping groups (VPGMs) are composed by a turbo-molecular pump, a primary pump, vacuum gauges and one or more valves. The combined operation of all these devices must be carefully controlled. The control procedure is implemented on a programmable logic controller (PLC) allocated on a crate fitted on the trolley. This kind of core solution is reliable, safe and ready to use. However, as stated in [\[2\]](#), at long term, the use of a PLC may be associated to some drawbacks. Due to the constant release of new models, an older one will lose, gradually, its support, on both hardware, with no availability of spares, and software sides, due to the no release of development platform updates and loses compatibility with new operating systems, and the problems already faced on the past will become again a reality. In

addition (and although partially justified by the features offered), a PLC is not the cheapest option and its dissemination along almost two hundred mobile pumping groups implies a significant investment.

To suppress such shortcomings, the evaluation of the reliability and feasibility of an electronic card, whose core unit is micro-controller (or similar technology) based, to replace PLC use is of particular interest.

### 1.1.1 Project details and objectives

Following the showed road implies the consideration of two major concerns: ensures electrical compatibility with all the devices that compose the pumping group and implements the control process that supports the group operation.

Regarding electrical specifications, distinct kinds of interfaces must be supported. The large part of connections is of discrete kind, 0-24V. They are associated to feedback signals, which indicates the current state or operation mode of the devices, such as, for example, valves and pumps, and control signals, actuation orders executed on the devices. There are also analogical input connections that shall be conditioned, namely 0-10V signals and a type-k *thermocouple* connection. A full list of group connections is attached on C. Additionally, the card must be designed to be properly fed by the available power supply, a 24V DC UPS.

Furthermore, a core architecture is required to implement the control process. This element must acquire input data, process the information and generate a suitable output response, according with the group operation scheme.

In addition, a HMI must be included to allow user monitor and interaction, ordering process changes, configuring its parameters and handling devices operation.

The combination of all topics explained represents the main objective of the project. However, this not fully replace the current PLC-based control system. Therefore, additional features are indicated below but their contemplation is optional.

The connections described are I/O based. However, some devices may be interfaced through *Profibus*. This type of communication provides a more complete diagnosis than the previous one. Here, the PLC acts as master and the field devices as slaves, but, on a separate *Profibus* network, it is integrated on Vacuum SCADA, acting as a slave and exchanging data with a central master PLC.

In short, the project objectives are summarized on next topics:

- **Mandatory**

- > Design of an hardware architecture, including acquisition circuit and processing unit.
- > Implementation of pumping group control process with additional interlocks, error flags and respective handling action and possibility of devices manual control.
- > Implementation of an HMI to user interaction, receiving user inputs and providing a complete group diagnosis.

- **Additional**

-> *Profibus* support:

Acting as a master, interacting with turbo-molecular pump controller and vacuum gauge.

Acting as a slave in order to be integrated on vacuum group SCADA.

## 1.2 Document structure

The present report is composed by six chapters. Firstly, on the current one, it is introduced the subject in study as well as the respective objectives, providing some guidelines for the technical features aimed. On 2, it is presented the state of the art, being explored the useful topics to posterior steps of project development, namely the utility of pumping groups, the control process inherent to it, hardware support and other relevant group features. On chapters 3 and 4, both hardware and software architectures, respectively, are detailed from bottom to top, referring specific design details and the tools that support development stages. On chapter 5, the tests and results developed on the system to evaluate its performance are presented. Finally, on chapter 6, all work developed is analyzed, being enumerated the remarkable achievements and the tasks not so well succeed, being suggested further actions that might be done in the future. Additionally, it is also included on A, B, C and D some additional information and guidelines to help the user through software packages installation.



## Chapter 2

# State of the art

In this chapter are explored the concepts that may be relevant on project development, creating a basis knowledge that will support the posterior stages.

Thus, firstly, it is contextualized the particles accelerators operation, 2.1, and why it implies the existence of vacuum groups, 2.2, how they should be handled to be efficient on their task, leading to PLC application and what are the characteristics that it offers that make it so widely used on automation environment, 2.3, being, then, compared, software, 2.4, and hardware, 2.5, options, aiming to satisfy the same purposes.

### 2.1 Particles accelerators

The possibility of knowing the history of the Universe is a common wish shared among all humanity. “What we are made for?” or “Where we came?” are questions that easily across our mind. However, their answer is not immediate and can cause some confusion. To solve these issues exists the particle physics. This area tries to understand the base laws that support the Nature and its elements through the study of fundamental particles of matter. Since they were born at the Big Bang, their structure preserves the timeline of the Universe and reflects its evolution. With this purpose in mind, the scientists drive their experiments to pursuit the recreation of the conditions that were present at the primordial explosion and identify the subatomic particles associated. This is only possible due to particle accelerator, the 21<sup>st</sup> century time machine.

#### 2.1.1 Basic principles

A particle accelerator is defined as “a device that provides forces on charged particles by some combination of electric and magnetic fields and brings the ions to high speed and kinetic energy” [32]. In other words, it’s a machine that allows the gradual acceleration of electrical particles and, consequently, to reach successive higher levels of energy along a specific path. When a desired state of energy is obtained, they are forced to collide and the results (small size particles and radiation) are collected to posterior analysis. The parameters of the accelerator, like size, type of trajectory and energy level, are correlated and dependent from the end application. Although

the most well-known usage is the research area, these principles are also widely disseminated in medicine practices, namely radiography and radiation therapy, and in technological equipment, as, per example, old televisions and computers monitor based on cathode rays [33].

Usually, the accelerators are classified in two types. In linear accelerators, particles travel down on a long and straight track, colliding with a target, while, in circular models, travel around a variable number of times in a circle until they collide with the target [32][34]. Nevertheless, in general, all kinds of accelerators share the same operation principles.

In first place, it is mandatory a particles source. Commonly, the circulated elements are protons or electrons. These particles are present in any material. However, they only assume a single number on hydrogen nuclei. So, this is the gas selected to provide the particles. The principle of extraction depends the type of the accelerator, but, generally, implies the bombardment of the gas with energetic electrons that results in the ionization of the atoms [33]. As consequence, it is obtained a beam of particles that is injected in the circulation channel, designated beam pipe.

Once in circulation, the particles must be gradually accelerated in order to achieve the high states of energy. This goal is accomplished due to the presence of electromagnetic fields along the trajectory. The beam is properly segmented in sectors denominated cavities. In each one exists a microwave generator. When a passing particle interacts with these waves, it is created an electric force that, depending of the orientation, may push the particle forward, increasing the speed, or difficult the movement. This process repeated along all beam length results in a cumulative arising of the particles energy. However, this only happens if the interaction between the beam and waves occurs at a specific timing (the moment when the electric force impulses the movement). For a precisely coordination, the particles are aggregated in bunches. The spacing between cavities and circulating bunches is proportional to the wavelength [33][32][35].

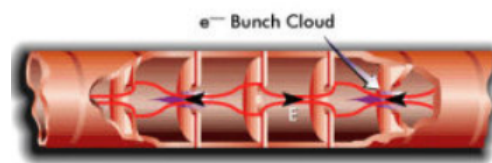


Figure 2.1: Cavities on the beam pipe [1].

Additionally, to assure a uniform distribution of the energy and to avoid collisions with the vessel's walls, the bunches must be kept focused and concentrated at the center. That's why, these same walls are complemented with dipoles and quadrupoles. They generate a magnetic field that conditions the particles' trajectory. The dipoles are responsible for the concentration of the beam while the quadrupoles assure that the particles remain at the center, especially in non-linear paths [35]. Also, the intensity of the field must increase with the energy of the beam. So, due to the large current implied, it must be dimensioned an adequated cooling system.

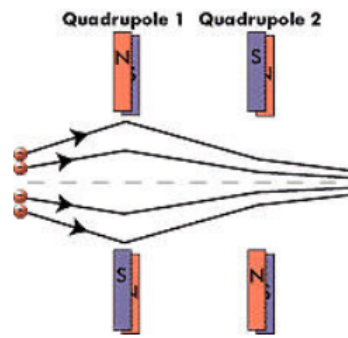


Figure 2.2: Magnets arrangement to confine the particle beam [1].

Finally, when the pretended state of energy is reached, the particles are forced to collide. In research area, to monitor and track properly all the process, this event occurs in specific points called detectors [34]. Here, all the information obtained is recorded and analyzed. The collision may happen with a fixed target or between bunches circulating in opposite directions.

An additional condition to the above procedure is that the all beam pipe must remain at vacuum. This state is essential to avoid the collision of the particles with the molecules suspended in the air that would cause significant losses of energy and the deviation of the beam. These situations unfeasible all the process. So, a pumping group is a mandatory system. The low pressure associated depends from the level of energy inherent to the accelerator.

### 2.1.2 CERN accelerator complex

At CERN, the particles experiences are developed by a complex of machines. These elements are connected in chain. So, the output beam of one accelerator constitutes the input of the next and so on. This configuration allows successive levels of higher energy that culminate at LHC. The previous elements are necessary to inject the particles into the chambers and to accelerate them properly to the next stages. The *Large Hadron Collider* is an accelerator that has two beams with bunches circulating in opposite directions along, approximately, 27 km in a circumference path. Along this length, there are four experiments, points where, after several passages, occur collisions as result of particles interaction, each one associated to a specific purpose of research [35][2].

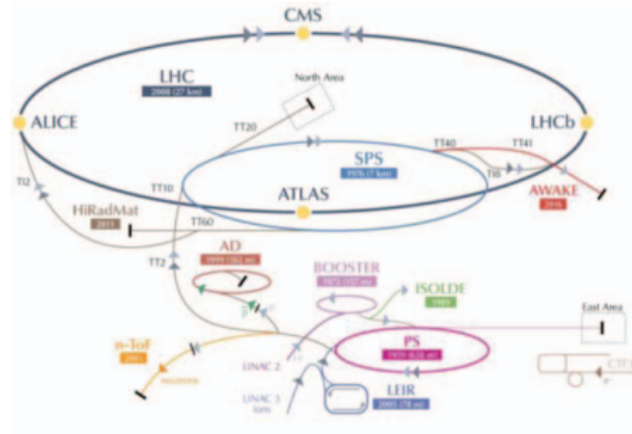


Figure 2.3: CERN accelerator complex [2].

The high stage of energy associated to the LHC depends substantially from both magnets cooling and vacuum systems. For one side, it is more challenging bend the particles trajectory to the beam pipe. Therefore, a higher excitation current at the magnets is needed. To avoid power dissipation, it is implemented a cryogenic system, based on superfluid helium, which allows the operation at temperatures as low as 1.9k. The thermal insulation with the surrounding area is only possible due to a vacuum barrier. Besides, the beam pipe must also be composed with the minimal concentration of gas in order to avoid undesirable collisions with residual particles. So, the vacuum system plays a dual key role in the accelerator's activity [34][35][2].

## 2.2 Vacuum system

As stated previously, the vacuum in beam pipe is an essential ingredient to a correct full operation. If this condition isn't verified, the beam particles interact with the residual gas present in circulating chambers. As result, they are scattered and diverge from the planning path and may also collide with chamber's walls. Therefore, the lifetime of the beam is reduced as well as the luminosity produced, and the ionization occurred on the interaction induces instability in the circulating beam [4]. In synthesis, the overall performance of the machine is compromised. This way, the pressure at the chamber must remain at very low value to assure the efficient of the process.

### 2.2.1 Vacuum principles and instrumentation

Pressure is defined as the force applied per unit of surface inside a restricted space. In other words, when a gas is contained in a vessel, molecules describe thermal movements and interact with each other and with the walls of the surface [3]. Consequently, this results in partial forces that are applied over the particles and limit their behavior. The sum of all these forces determines the pressure inside the recipient. The S.I. unit is Pascal,  $Pa$ . However, it is also common to express in *torr*, *bar* or *atm*, whose reference, in last one, is the atmospheric pressure at sea-level, 101.325  $Pa$ .

In literature, vacuum is described as “the state of a gas at which its pressure in a vessel and therefore its particle density is lower than that of the ambient surrounding atmosphere or in which the pressure of the gas is lower than 300 mbar, i.e. lower than the pressure of the atmosphere on the Earth’s surface” [3]. It is possible to identify distinct levels of vacuum according with the pressure range considered.

| Pressure range              | Pressure hPa                          | Pressure Pa                           |
|-----------------------------|---------------------------------------|---------------------------------------|
| Atmospheric pressure        | 1,013.25                              | 101,325                               |
| Low vacuum (LV)             | 300...1                               | 30,000...100                          |
| Medium vacuum (MV)          | 1...10 <sup>-3</sup>                  | 100...10 <sup>-1</sup>                |
| High vacuum (HV)            | 10 <sup>-3</sup> ...10 <sup>-7</sup>  | 10 <sup>-1</sup> ...10 <sup>-5</sup>  |
| Ultra-high vacuum (UHV)     | 10 <sup>-7</sup> ...10 <sup>-12</sup> | 10 <sup>-5</sup> ...10 <sup>-10</sup> |
| Extremely high vacuum (XHV) | <10 <sup>-12</sup>                    | <10 <sup>-10</sup>                    |

Figure 2.4: Pressure ranges classification [3].

In particles accelerators, vacuum chamber is at ultra-high vacuum. To achieve and maintain this condition, it is mandatory a special vacuum system capable of generating this kind of pressure. However, instrumentation and pumping by themselves aren’t enough. It is fundamental a proper treatment and handling of material, previously and during the operations, to minimize the residual gas inside the beam pipe [4][36][3].

Generally, gas loads can have one of the following origins [37]:

- **Outgassing loads:** residual particles released by the vessel walls;
- **Permeation loads:** gases originated outside of vessel walls;
- **Diffusion loads:** gases originated inside of vessel walls as consequence of ionization of particles.

Firstly, outgassing loads constitute the most significant part. These particles accumulate at the surface of material during the production phase as result of dust inherent to industrial processes. To inhibit their presence, the material suffers a chemical clean before installation to remove them. However, some of particles still resist, mostly moisture. The best way to remove water is through heat. As result, the surface is heated by a bake-out system before the assembling and, periodically, when in operation. By other side, diffusion loads are consequence of the material type. In this field, copper represents the best compromise. Assembling process also affects the concentration of the residual charges. Therefore, the join areas and the screw needs are reduced as possible to minimize the probability of contamination. Finally, the permeation loads are consequence of the above two enumerated [36][37].

Once the vacuum chamber is properly settled, it is ready to be evacuated. Here, the key elements are the pumping mechanisms. They are responsible to lower the pipe pressure since the atmospheric to UHV condition. However, due to the roughly range variation that it implies, the

process is separated in stages, each of them assured by an adequate pump [4]. To this purpose, there are many pumps, diverging in the principle and conditions of operation. At a high-level, they can be separated in two major groups: gas transfer pumps and gas binding pumps. The first ones take advantage of the kinetic energy and the compression effects to prompt the particles to the atmosphere along extraction lines while the second ones capture the particles through absorption, condensation or chemical combination, this last one commonly associated to a limited capacity of extraction [3].

In picture 2.5, it is presented the pressure range of application of common pumps.

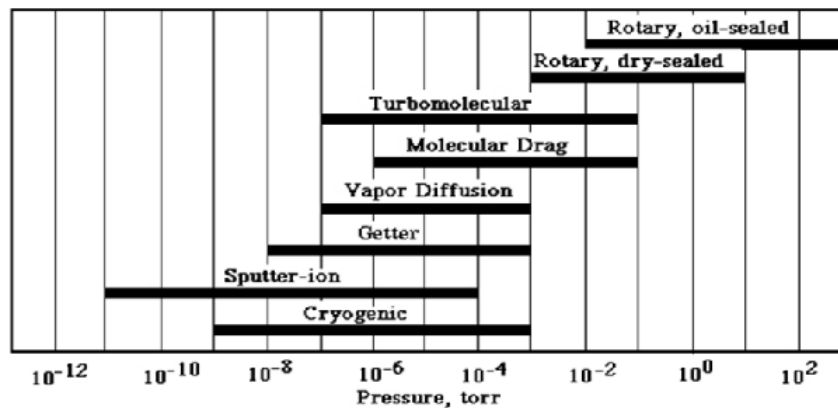


Figure 2.5: Useful range of standard pumps [4].

To a precise control of the chamber vacuum, the system needs to know the actual pressure at each moment. So, it must be installed vacuum gauges along the pipe line. Also, here, distinct types of mechanisms should be conciliated to allow the measure along all range of values. To low vacuum conditions, it is typical the application of instruments composed by an elastic membrane whose deflection is proportional to the pressure level. Under that, a force cannot be induced in sensing element. So, the measure of physical properties that are function of the gas density represents a more feasible approach [4].

It is represented on 2.6 the different vacuum gauge options and respective application range.

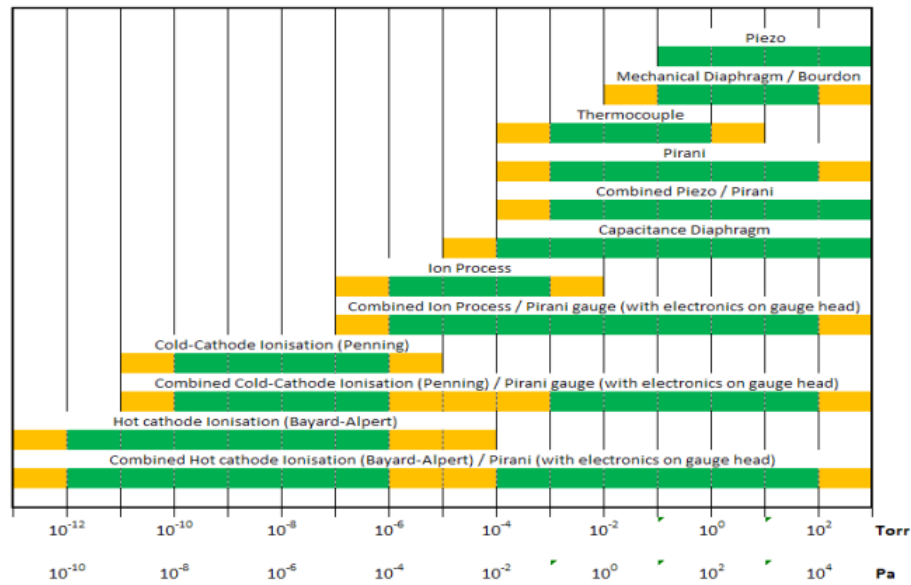


Figure 2.6: Pressure gauges application [4].

Finally, there are also valves. Manually operated and/or electro-pneumatic actuated, they regulate the state of aperture between the vacuum chamber and pumping stations or atmospheric environment access [36].

### 2.2.2 CERN pumping groups

At CERN, particles activity is performed, as already expressed, on an ultra high vacuum environment. In order to achieve this level, pumping operations must be realized gradually through several stages. The mechanisms involved are denominated pumping groups. Generally, they include a pump, vacuum gauges, valves and, eventually, leak detectors [5].

The first phase of pumping, i.e., the lowering of pressure since atmospheric until  $10^{-9}$  torr, is guaranteed by turbo-molecular pumping groups [2]. They are composed by a primary pump, turbo-molecular pump and penning/pirani gauges. The layout is illustrated on picture 2.7.

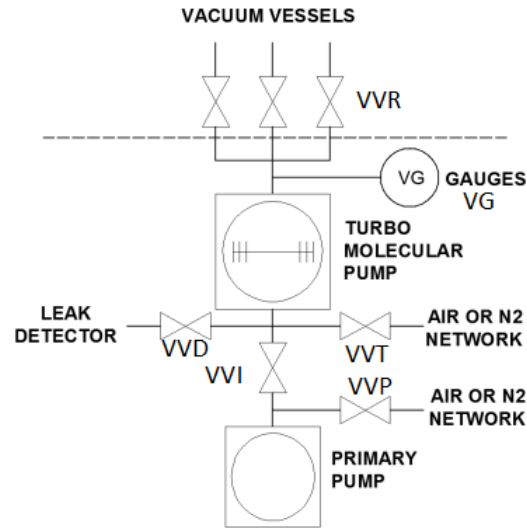


Figure 2.7: Basic structure of a turbo-molecular pumping group [5].

The operation principle of turbo-molecular pump relies on the transfer of particles' momentum. An internal rotor with blades make the particles flow in axial direction with a successive higher compression, only possible due to the decreasing spacing among blades. The compression rating at exhaust side will depend of rotor speed, but, at nominal values, it's never higher than  $10^{-2}$  torr [4]. In order to conclude the evacuation process, a series disposed primary pump is required to raise the pressure to atmospheric parameters and, this way, expel the gases. Typically, it is a rotary van-based pump [2].

In addition, the vacuum vessels must also be vented when needed. So, this reverse process need to be properly controlled in order to avoid the contamination of the chamber and compromise the posteriors events. So, instead of air, it is commonly the utilization of nitrogen and, this way, minimize the deposition of residual gas present at the air on vessels surface [2].

The additional valves present are necessary to regulate the communication among the circulating lines on the different states of the pumping and venting processes. Therefore, VVR valves establish the communication between turbo-molecular pump and one or plus vacuum vessels, VVT and VVP are used to venting process and VVD is applicable on leak detection. Besides, there is also a vacuum gauge on the inlet side of TMP to measure the pressure level [2][5].

The referred pumping groups are inherent to all machines integrated on vacuum system at CERN's complex. However, there are two possible configurations. In some cases, they can be fixed to the machine and, on other side, mobile groups may assure the pumping process. The last ones are distributed along the accelerators and are utilized where they are required. Once pumping is ceased, they are free to be used elsewhere [2].

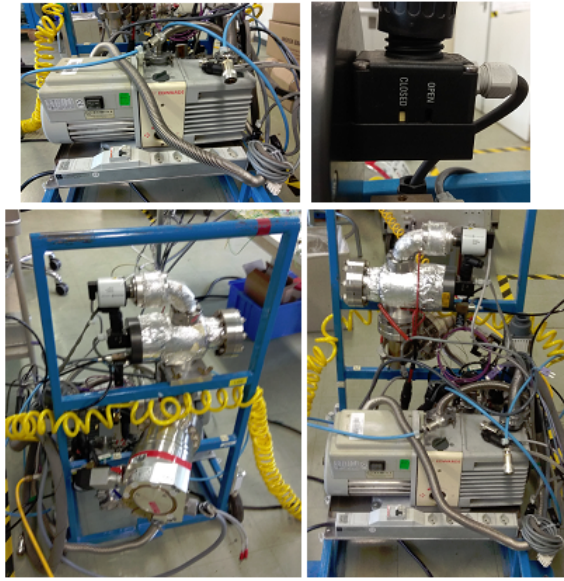


Figure 2.8: Front and back view of a CERN mobile pumping group.

### 2.2.3 VPGM control

The group during pumping and venting processes goes through different operational stages and, in each one of them, the coordination between all devices shall be a major concern.

The control process running in background on turbo-molecular pumping group may be described by a state machine diagram. In off mode, the automatic control is shutted down and all the devices remain in the current state in a way that can be manually operated. This condition, however, may cause hazardous consequences, thus, only the most experienced ones are able to manage the group. On other states, the machines are automatically controlled by the process and it includes phases of pumping, venting, leak detection and even no operation [5]. Each one is yet decomposed in several steps. The transition between states may occur automatically or may be requested by the operator. Although, in some operational conditions, the action indicated may be blocked depending the status of the devices or the states involved [2]. In the figure 2.9, it is present a high-level representation of the process control state diagram.

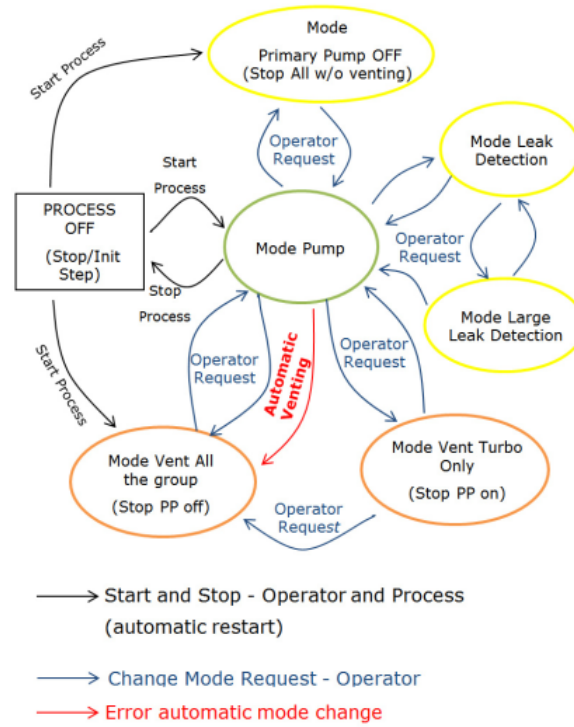


Figure 2.9: Operation modes of VPGM [5].

As additional safety measure, there are also top-level interlocks [2]. They are implemented to each valve, independently of the state machine, and unable manipulating actions when some conditions are satisfied. They can be of two kinds. Start-interlock maintain the current state of the valve when occurred. Full-interlock close the valve, if it is not, and remains at that state.

The whole control process of the pumping group is implemented on a PLC [5]. So, it shall interact with all the devices that compose the pumping group. These interactions resume to receive feedback signals and send action commands. The electrical connections include six electronic-pneumatic valves, primary and turbo-molecular pumps, the bake-out system and, of course, a power supply. The type of interface is either digital, 0-24V, or analogical, 0-10V.

Although all the connections may be I/O based, some of them are based on *Profibus*, namely with the TMP controller and the vacuum gauge. In addition, the PLC also acts as *Profibus* master in the interface with a local touch panel, which represents human-machine interface.

Besides that, the pumping group is also integrated in an upper-level control system, vacuum SCADA. This network relies on *Profibus*, but, here, local PLC acts as a slave device [2].

In picture 2.10, it is represented the typical hardware architecture.

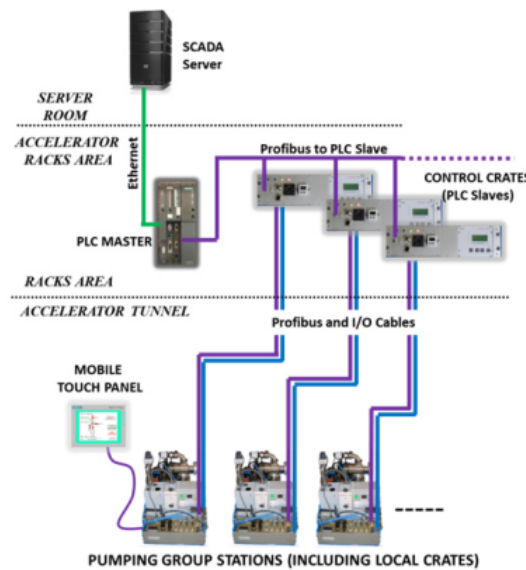


Figure 2.10: Hardware architecture [5].

Once the purpose of this thesis it is to develop an electronic card that can replace the PLC application, all the electrical compatibility with the connections previously described shall be assured. In order to guarantee, at least, the same level of computational capacity and safety, it is convenient the study of typical PLC characteristics as well as appropriated core solutions to the card.

## 2.3 Programmable logic controller (PLC)

A PLC is an industrial computer designed to control manufacturing processes. The first device surged at 1960's years and the main purpose was replacing the relays switches, which dominate the industrial controlling scenarios until then. The relays application required the electrical wiring of these elements along the whole circuit of the system to be controlled in such a way that a specific sensor position triggers a proper action on output actuators. Therefore, a particular solution should be devised to each control application [6]. With the advent of PLCs, those problems were overcome.

### 2.3.1 PLC architecture

PLC offers a control system that can be reusable in distinct control contexts. The control rules are set through a microprocessor-based system and accordingly with a logic view of the process, i.e. if the state A was observed the action B must be taken [38]. These industrial devices rely on operational principles like ones that support a typical personal computer. However, the first ones diverge by a more limited computational capacity, restricted to arithmetic and logic operations, a real-time system and, of course, a more robust hardware since they must support dust, vibrations,

noises and other adverse conditions inherent to industrial environments [6][39]. The advantages highlighted make the PLC a typical presence at industry nowadays.

In a simply perspective, PLC operation resumes to acquire inputs provided by external field devices (sensors, for instance), process them according with the instructions programmed at memory and provide the respective outputs, which are sent to the actuators. To perform these sequence of operations, they must be composed by the following elements. Firstly, a CPU is required to compute the information based on the instructions configured at memory. Thus, a unit memory is also essential to register instructions, variables, timers and other parameters relevant to process. The configuration of memory implies a programming device, typically assured by an external PC running a software developed by the manufacturer. An additional communication module is also present to interact and exchange data with another PLC. Finally, input and output modules are a must to interface with field devices and a power supply to feed the entire system [38].

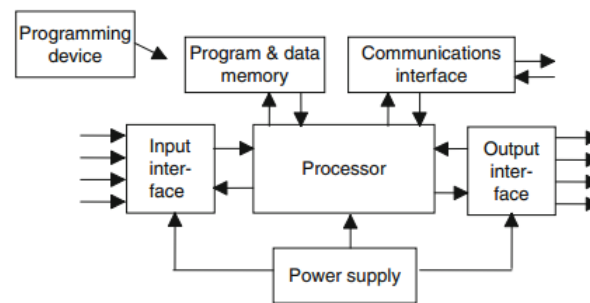


Figure 2.11: PLC architecture [6].

The previous elements are necessarily integrated in the architecture of any device. However, depending the modularity associated to them, there are two distinct mechanical types of PLC. For one side, there is the single box (or brick) type. In such case, the units are all fixed in a monolithic structure and the number of interfaces available to connect to inputs and outputs are static and cannot be extended. Therefore, it is more suited for small applications. On the other side, there are the modular ones. Here, the units are all mounted at the same rack but independently. The rack is composed by several sockets where the units can be plugged. Therefore, the modules can be added or removed according with the application and so distinct elements can be mixed. The connection among them is assured by connectors at the backplane [6].

### 2.3.2 IEC 61131

IEC 61131 is a standard developed to PLCs. It establishes the principles that regulate the whole lifecycle of these devices as well as the peripheral instruments that can be connected to them. It is divided in several parts and, among other aspects, includes the construction and functional requirements, tests and validations applied, operational conditions, safety issues and user guidelines [40].

Among all parts, there is one that deserves special emphasis, IEC 61131-3. This section specifies syntax and semantics of programming languages for programmable controllers aiming

to unify the software implementations through all manufacturers. The standard describes five possible languages: Instruction List (IL), Structured Text (ST), Sequential Function Chart (SFC), Ladder Diagram (LD) and Function Block Diagram (FBD) [40].

It is possible to distinguish two groups. For one side, there are textual languages represented by ST and IL. ST allows a programming paradigm very similar to Pascal and IL is more logic and intuitive, directed for those who not feel too comfortable in programming. However, in both options following the process flow is not the easiest task. Thus, targeting a better system overview, there are graphical languages. LD is suitable for electrical engineers who prefer a graphical wiring representation of the installation, but any complex implementation beyond arithmetic operations won't be feasible. SFC emphasizes the process states and the transitions conditions and provides a very clear representation. FBD is well fitted for repeated tasks because it allows the easy reuse of code fragments. Based on the aspects referred, it is easy to realize that the implementation of a control process on a PLC corresponds to a task that is not uniquely restricted to electrical engineers or programming crew [38][7]. This is the purpose of the standard, user-friendly configuration of programmable controllers.

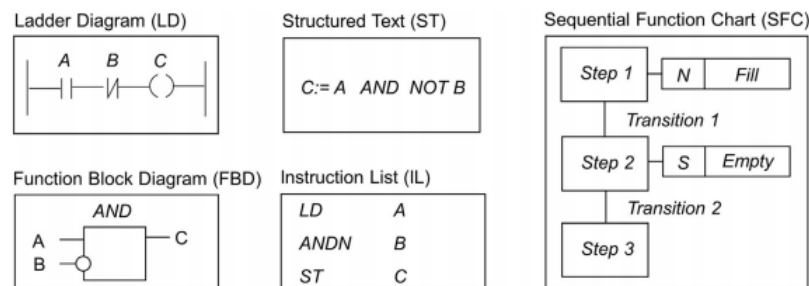


Figure 2.12: IEC 61131-3 programming languages [7].

### 2.3.2.1 Software architecture

The standard establishes a hierarchical approach to programming structure on PLCs.

On the top position, there is the configuration block. It defines the PLC function on a plant where can exists multiple PLCs with distinct purposes [8], being each one related to a specific operation, in other words, represents the entire PLC system. A configuration is composed by resources. According with the standard, a resource allows all kind of interfaces, namely with I/O devices and even human-machine interface. So, a resource is a mandatory element and generally corresponds to a runtime application for instance, which can be allocated not just on PLC as well as on personal computer that is used to programming work [41]. Generally, a resource may include multiple tasks. A task defines the execution of specific actions and has some attributes, namely name, type (defines the execution behavior, i.e., if it is executed continuously, cyclically or triggered by an event) and the priority (to define execution order) [8]. A task is related to a program, whose implementation relies on function blocks and functions. A function block is located at a higher position than functions. The main difference is that the first ones can preserve

its internal state between executions and, that's why, the result of each invocation is not always the same even if the input arguments don't change, while on functions they return always the same output value for one specific input array [8]. Due this, they occupy the bottom position of the hierarchy.

The picture 2.13 depicts the previous ideas.

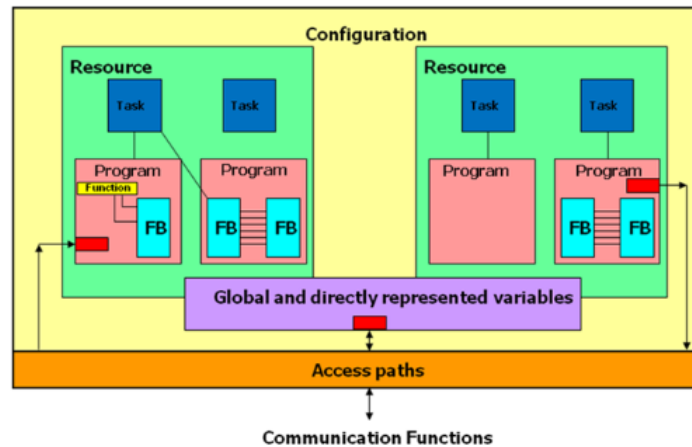


Figure 2.13: PLC software architecture [8].

### 2.3.2.2 Sequential Function Chart (SFC)

The operation of almost engineering systems is characterized by several states that represent distinct systems conditions occupied by process while running. The state change is launched by a specific condition inherent to the system being modulated.

In engineer, the simpler approach to design such behavior is through state diagrams. They are composed by states, represented by circles, and linked to each other in a way that recreates the expected system flow, being associated a transition condition to each arrow. This is the most primitive tool to describe a process.

A standard and more sustainable design methodology is GRAFCET. The principle on it relies is the same, but, here, a more clear and structured solution is obtained [42]. It is recreated reliably the system flow from top to bottom and a functional overview of the whole control process is obtained.

To modulation purposes, three main elements must be considered: steps, actions and transitions. A step is the state associated to the system in one specific moment. When starting up, it must be assigned a singular step to the process, flagged by a double line around that [43]. While at a step, some tasks may be performed on the process. For that reason, a step may have linked to it actions. Such actions, however, can, for instance, being executed not during all step activation, but just at a specific time range, while activated or even after its deactivation. Such representation is obtained using action qualifiers. This is a letter which indicates what type of action is been dealing. The code established is depicted on table 2.1.

Table 2.1: Action qualifiers [27].

| Qualifier | Description      | Qualifier | Description             |
|-----------|------------------|-----------|-------------------------|
| N         | Non-stored       | SD        | Stored and time Delayed |
| R         | overriding Reset | DS        | Delayed and Stored      |
| S         | Stored           | SL        | Stored and time Limited |
| L         | time Limited     | P0        | Pulse, falling edge     |
| D         | time Delayed     | P1        | Pulse, rising edge      |
| P         | Pulse            | PN        | Pulse, Non-stored       |

Finally, there are transitions. They modulate a condition, passible to be evaluated as true or false, that must be verified to allow system evolution, i.e., step change. Between two consecutive steps shall always exist a transition.

In addition, several rules must be respected during design procedure. The whole process might be represented by one or several branches. Branches bifurcation is accomplished making use of specific structures, alternative and parallel branching. In first case, just one of the possible several branches included may be followed, beginning and finishing with a transition, while on parallel case all branches are executed simultaneously, beginning and finishing each branch with a step, considering that the process will only evolve when all of them are on last step. In both cases, the junction of branches is possible through joint structures [43].

A process may end at a specific step or be described in loop. The last scenario is possible taking advantage of jumps, providing a transition between nonconsecutive steps. Its application, however, should be cautious, since an overuse may difficult following process logic [43].

The SFC language is result of integration of GRAFCET modelling tool on IEC 61131 standard as a graphical programming language [42]. For that reason, its support is common among PLC manufacturers. The implementation is made respecting the rules previously defined and, consequently, it is just to migrate the process design to development environment almost with no change, preserving the intuitive overview and facilitating the debug procedure once running.

### 2.3.2.3 PLC scan cycle

Generally, when working with PLCs, the operator, through a specific manufacturer software, develops the code that reproduces the control action required to be performed. The program is then transferred to internal memory and, once running, will assure the intended process rules. To operate, several inputs are required, provided by sensors for example, and another amount of outputs is generated, and sent to the actuators. The steps that occurs between these two stages depend from the code implemented.

Although the program located on memory set how occurs the processing step, the logic that supports the PLC operation is always the same. In first place, the inputs are acquired. All of them are registered simultaneously and uploaded to the input process image. This is important to assure

consistency among all input data and to safeguard that the signals are related to the same process moment. Once acquired, the data is processed and the outputs returned are saved on the output process image. Again, here, the commands are transferred simultaneously to the physical devices at the end of the process. Once concluded, the cycle is repeated [44][9].

The presented behavior is inherent to any PLC and is called the scan cycle, as illustrated on picture 2.14.

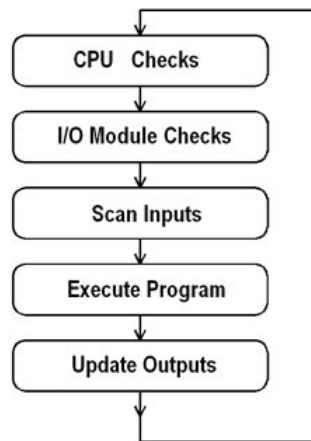


Figure 2.14: PLC scan cycle [9].

### 2.3.3 Input module

The input module of a PLC allows the electrical connection with field devices such as sensors or switches. Depending the device considered, signals generated may have distinct characteristics, i.e. discrete/digital signals where only two levels of voltage are possible or analog signals where the voltage or current varies continuously within possible parameters range, for instance 0-10V or 4-20mA signals, respectively [10]. Despite that, a PLC is a digital device and internally operates at voltage mode, between 0V and internal voltage level, normally 5V. Externally, it is necessary to adjust the outside signals to these specifications. This way, there are distinct types of input modules, diverging not only in the number of connections allowed as well as in the interface supported.

On digital/discrete connections, the principal caution is related with voltage levels. The inputs may have one of the various ranges that are possible, such as 5V, 24V, 110V and 240V [6]. When selecting a module, this aspect must be considered in order to make it corresponds the input signals expected. Typically, these devices present internally an *optocoupler* to isolate the internal circuit from the outside device. Thus, the *optocoupler* plays a dual role, facilitating also the adjustment of voltages levels [6][39]. In addition, to prevent an eventual connection of the input in reverse mode and its consequent destruction, a diode is included in parallel with *optocoupler*, as exposed in picture 2.15.

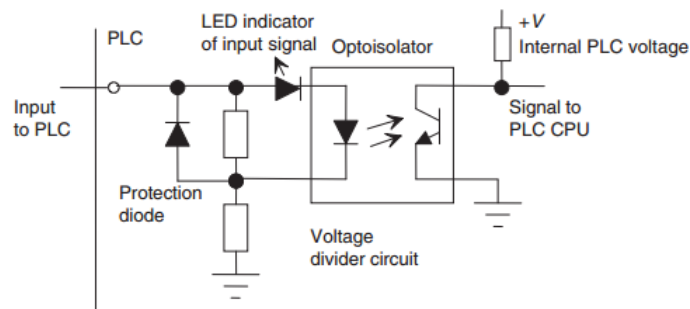


Figure 2.15: PLC DC input [10].

With analog signals, more aspects shall be considered. Since the internal operation of PLC relies on digital logic, the input signal must be acquired by an analog-digital converter (ADC). Here, the number of bits assumes special relevance. The choice will depend of the desired minimum input variation that causes a change in one bit (called resolution) [10]. In addition, a module commonly includes a unique ADC and several inputs whereby a multiplexer is also included, being the channel selection internally controlled. Besides all that, it is convenient a previous filter to prevent aliasing effects due to the possible high frequency of the process being sensed [39].

In some cases, a mix of analog signals may have to be connected to PLC (for instance, 4-20mA and 0-10V). To avoid the use of different modules, an external conditioning circuit may assure the compatibility. Typical topologies may be the use of a simple resistor (converting current in voltage), voltage divider or an op-amp (to assure compatible voltage levels) [10].

### 2.3.4 Output module

The outputs of a PLC may be of distinct kinds depending of the load to be controlled and the behavior intended: relay, triac and transistor.

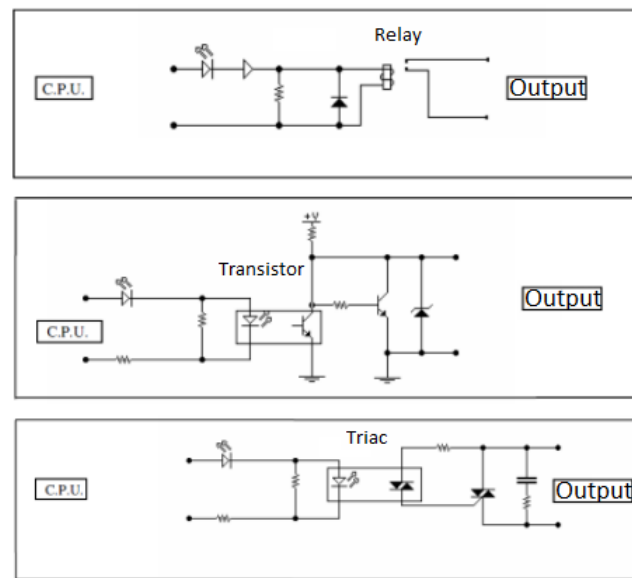


Figure 2.16: PLC output (adapted from [11]).

A relay output, as the denomination suggest, act as a typical relay allowing the switching operation with AC and DC signals and supporting large currents as well as significant voltage levels (up to 30V DC/2A and 240V AC/2A). It represents the more versatile option but supports a low switching frequency. To overcome such disadvantage, there is the transistor type. This provides a very high switching rating but it is only suitable for DC applications and lower currents, typically not much higher than 300mA [10][38][10]. Finally, the triac output is equivalent to the last but for AC applications.

The transistor and triac types are isolated from the internal circuit of PLC by *optocouplers*. In addition, it is also included in parallel with the output a protection diode. This device (also called *flyback* diode) is used to protect the output driver when switching inductive loads. In such case, the permanent commutation induces back EMF as consequence of magnetic phenomenon and the diode provides an alternative path to the current flow in such situation, avoiding the destruction of the output [10].

### 2.3.5 Source and sink

When dealing with digital I/O, two concepts should be clarified: sourcing and sinking. These classifications are strictly related with the direction in which flow the current relatively to the I/O module [38].

An I/O is considered of sourcing type when provides the voltage to the circuit with the external device, i.e. the outside element is supplied by the interface module.

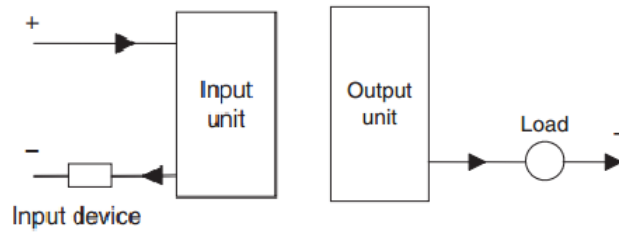


Figure 2.17: Input and output sourcing unit [10].

On the other side, a sinking I/O provides the ground to the circuit and so it is fed by the external device.

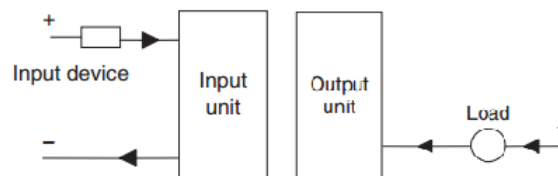


Figure 2.18: Input and output sinking unit [10].

### 2.3.6 Communication module

On an industrial scenario, a PLC might need to connect with many peripheral devices. For instance, to transfer the control program to memory unit, it is necessary the interface with a computer. Additionally, it is also common the communication with local human-machine interface (HMI) or remote PLCs to data exchange purposes. Thus, to assure the enumerated interfaces, a communication module is integrated in hardware architecture.

The communication module may provide diverse kinds of interaction. The simplest that is typically available is serial communication. It includes several standards, namely RS232, RS422, RS423, RS485 [10]. The choice will depend of the distance implied and the characteristics of surrounding environment. For example, a connection established through RS232 is not practicable to distances larger than 15m due to noise that corrupts the transmitted signal. On the other hand, RS485 represents a balanced transmission, i.e. based on the voltage difference between signals, whereby higher speeds and distances are feasible [10].

However, in some situations, the simplicity inherent to serial interface may not be enough. To a more complex processing of data flowing and control of transmission, are used more robust systems of communication. Among all the industrial network systems available, the most known are the Fieldbuses and *Ethernet*.

### 2.3.6.1 Serial Peripheral Interface (SPI)

SPI is a protocol defined for serial communication. It is a synchronized approach intended for integrated circuits/chips and its main purpose is to provide a simple interface, overcoming the synchronization shortcomings and complexity inherent to UART based protocols [12].

Considering the premises stated, any data exchange between devices is complemented by an external clock signal. The composition of such signal will dictate when the receptor must sample the acquired or to be sent data. Normally, it is characterized by several pulses and the sampling moment is dependent from clock polarity and phase, parameters that may be configured or fixed for each device. Regarding to this question, four SPI bus modes are valid.

Table 2.2: SPI bus mode [28]

| SPI bus mode | Control Bit States |             | Description                                                                             |
|--------------|--------------------|-------------|-----------------------------------------------------------------------------------------|
|              | Clock Polarity     | Clock Phase |                                                                                         |
| 0            | 0                  | 1           | Clock idle state = High, Transmit occurs on transition from active to idle clock state. |
| 1            | 0                  | 0           | Clock idle state = High, Transmit occurs on transition from idle to active clock state. |
| 2            | 1                  | 1           | Clock idle state = Low, Transmit occurs on transition from active to idle clock state.  |
| 3            | 1                  | 0           | Clock idle state = Low, Transmit occurs on transition from idle to active clock state.  |

Considering the clock source, two types of devices can be defined. The master is the one that generates the signal transmitted, existing just one per bus, while the slave receives and sends the data according with the rate imposed by it, being possible multiple presences of such stations on the same bus [12]. So, there is no need to configure previously the transmission frequency. Despite that, normally the value must within a specific range, provided on devices *datasheet*, as well as the number of pulses must be proportional to the amount of data (bits) being transmitted in both sides, implying a prior set of the bus. The master station defines the bus rules, being commonly associated to microcontrollers or similar platforms, while the slaves are more related to sensors and equivalent devices [12].

SPI is a full-duplex protocol and, for such reason, there are two distinct data lines per bus. One carries the data sent by master to slave, being called MOSI (which stands for *Master Output Slave Input*), and the another one returns slave response, now called MISO, *Master Input Slave Output* [12].

When there are two or more slaves communicating with the same master, SPI bus must be shared. In this case, a device need to know when it is its time to exchange data, i.e., the information must be addressed. This is possible through a *Select Slave* line. In absence of communication, this

signal stands high, falling to low level at the beginning of transmission and remaining there until the end [12]. To apply this approach to several slaves, it is needed a signal per slave.

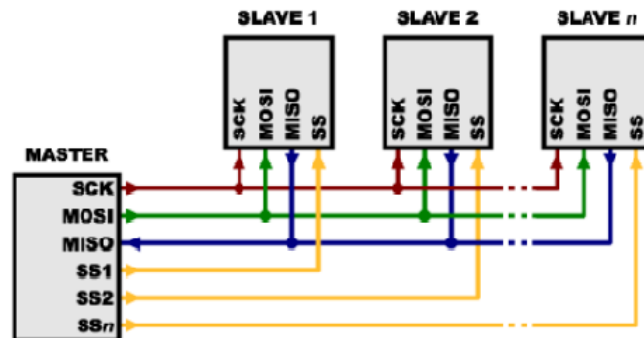


Figure 2.19: SPI multiple slaves communication [12].

In last case, If the amount of slaves is huge, that solution may not be feasible or, at least, too confuse. This can be overcome putting the stations on daisy chain configuration [12]. This represents a ring topology where the output signal of the prior slave is connected to the input of next one and so one, being the last one of the chain connected to master. Here, one select line is enough, activating all devices at once. However, this kind of application requires the device support as well as a proper disposition of data being transmitted in order to be delivered to intended slave.

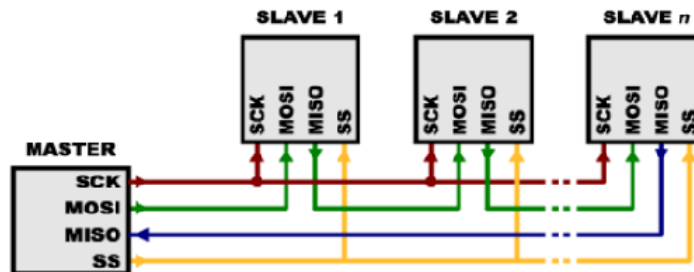


Figure 2.20: Daisy chain configuration [12].

### 2.3.6.2 Profibus

*Profibus* represents a standard communication largely widespread in automation applications. It defines a format messaging that allows a high speed I/O communication, in such a way that is considered the fastest Fieldbus. It is characterized by its interoperability, allowing the interface among devices from distinct manufactures, and time reliability, enabling the prediction of time range inherent to transmission. A transmission occurs in a speed range extended since 9600Kbps until 12Mbps, depending from the distance intended, and it is established between two devices, master and slave [45].

A master corresponds to an active station, i.e. when it has the permission to access the bus, may communicate without restrictions and outside permissions, prompting the interaction with the

slaves associated. Due this, a slave is considered a passive station, it is only allowed to respond to its master requests or acknowledge a reception. Generally, this role is played by peripheral devices such as sensors and actuators. The maximum number of devices permitted on a bus is limited to 126 [13].

In order to assure its interoperability, *Profibus* is supported by OSI model. This conceptual model groups the functions inherent to a network in layers, being each one related to a specific task and operating independently of upper and lower ones [14]. To full support a telecommunication system operation, there are seven layers, as below represented. *Profibus* standard only uses three of them: application layer, data link layer and physical layer [45].

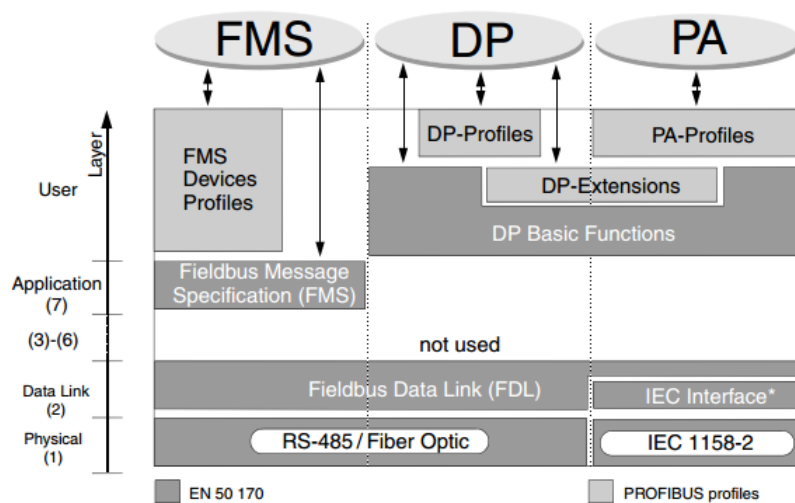


Figure 2.21: Profibus protocol stack [13].

At the application layer, it is possible to distinguish three versions of the standard: *Profibus FMS*, *Profibus DP*, *Profibus PA*. For one side, the Field Message Specification (FMS) was designed to allow the complex communication between PLCs and PCs whereby it is more suitable for the exchange of complex data and with less time restrictions. It is more applied on control level of automation, but, due to its low flexibility, it is currently abandoned and being replaced by *Ethernet*. The *Profibus Decentralized Peripherals (DP)* is the most common one. It assures high speed and time reliability in the communication between centralized equipment and distributed devices at factory floor, such as sensors and actuators. There are three variants of this version (DP-V0, DP-V1, DP-V2) with a successive increasing in available communication features, but accomplished by a proportional complexity rising. Finally, the *Profibus Process Automation (PA)* is similar to the last one, however, more suitable to operate on hazardous environments, due to the power limitation associated to physical transmission [13].

The data link layer is supported by the Fieldbus Data Link (FDL). The main functions are the control of the access to the communication bus and the monitoring of structure, integrity and safety of frames in transmission. As stated earlier, there are two types of devices associated to a *Profibus* network, masters and slaves. However, the communication is established by a common

bus and, therefore, this channel must be properly shared in order to avoid collisions. In this case, this regulation is obtained through a hybrid mechanism, combining token passing with master-slave technique. Firstly, one master device only has the right to access the bus when it holds a software token, which is circulated in a ring topology among all masters. The communication between masters is not allowed, being uniquely restricted to token circulation. Once holding the token, its applicable a master-slave method in which a master station requests all its slaves in a limited period. This parameter is configured by user and it is the reason of *Profibus* deterministic behavior. Master-slave interaction may be of two kinds, namely SRD (send and request data with acknowledge) or SDN (send data with no acknowledge). The frame transmitted will depend of information and its purpose, but the full size never exceeds 255 bytes, a maximum of 244 bytes to payload and a header of 11 bytes. Each character is codified with 11 bits [13].

| Start | D0  | D1 | D2 | D3 | D4 | D5 | D6 | D7  | Parity | Stop |
|-------|-----|----|----|----|----|----|----|-----|--------|------|
| "0"   | 0   | 1  | 2  | 3  | 4  | 5  | 6  | 7   | even   | "1"  |
| ←     | LSB | ←  | ←  | ←  | ←  | ←  | ←  | MSB | ←      | ←    |

Figure 2.22: *Profibus* character codification [14].

Finally, the physical layer is inherent to electrical transmission of the data through the bus. In *Profibus DP*, the communication relies on RS485 standard [14].

Additionally, the interaction between master and slave stations is only possible after an initial configuration in order to assign a valid address to each device and set the interaction parameters. The identification and specifications of a station (baud rate, data length, number of inputs and outputs) are stated in GSD file, a *datasheet* of an electronic device. This setting state is assured by another type of master station, a class 2 master. It acts as a supervisory element, playing configuration actions and diagnosis purposes. After this initial validation step, the devices can communicate and operate at the normal state [14].

In picture 2.23, it is represented a state diagram which illustrates the operation of *Profibus DP* slave.

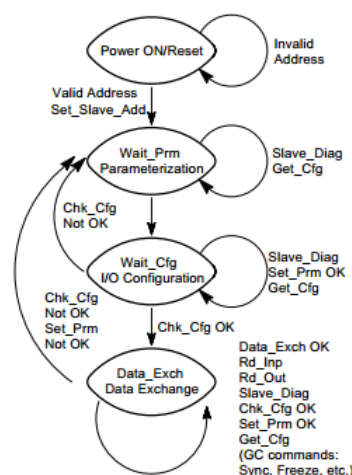


Figure 2.23: *Profibus DP* slave state machine [14].

### 2.3.6.3 *Profibus* implementation

The *Profibus DP* offers a fast data exchange combined with a deterministic medium access. Despite these appealing facts, the protocol implementation requires specific hardware and software with a high acquisition cost. Thus, the application domain is commonly limited to professional and commercial ends [46].

Traditionally, the interface with a *Profibus* network is made through specific interface modules. These elements assume a node role on fieldbus connection, described by an address, and allows the *Profibus* coupling among remote field devices. They implement the protocol layers and structure the exchanged information in both ways according with that. The module must be chosen considering the type of station desired, master or slave.

Another kind of support is based on application specific integrated circuits (ASICs). In this case, dedicated chips implement the lower two layers of *Profibus* stack and handle the medium access, requiring, however, a RS485 transceiver to interface with the fieldbus physical layer. Additionally, a host controller, typically microcontroller based, implements the application level layer and interacts, through serial or parallel interface, with ASIC [47]. This option represents a small size and not quite expensive alternative, allowing a tailor-made solution to the user who can build his own hardware architecture [48].

In order to offer a cheaper alternative than the *Profibus* market, there are some developments of open-source solutions [49][46]. The main purpose of these projects is to minimize the need of hardware and build a software-based solution to protocol stack, supported by a wide range of systems platforms. In this context, it is remarked the project indicated at [50]. This presents an implementation of a *Profibus* master. Currently, it is on development state, thus, the full implementation of protocol stack wasn't yet achieved. However, it is indicated that the communication with a *Profibus* slave was made successfully and, consequently, physical layer is functional and represents a good basis to posterior work. With this solution, instead of expensive hardware, it is just required a RS-485 interface and the costs would be minimal.

## 2.4 Soft-PLC

The PLC represents the most widespread solution in industry control scenarios. A hardware suitable for the most adverse environment, a hard real-time software with a wide range of programmable languages and the possibility of connection with diverse peripheral devices make them the most preferable choice on automation context. However, its use also implies some drawbacks. These result from the increasing inconsistency among the manufactures, specific software associated to each company and expensive acquisition cost [51]. To avoid these disadvantages and replace its utilization, the use of soft-PLC is becoming a reality.

Soft-PLC consists of a control software engine designed to the conventional CPUs that equips a PC and implements the characteristic features of a traditional PLC, i.e. using the hardware

and processing unit of a PC it is possible to suppress the basic function of a PLC. This PC-based solution allows a higher computation capability and a versatile support of communication interface. In addition, the portability of these devices permits higher flexibility and small size regarding to space requirements [52][53]. Besides all these facts, a soft-PLC is based on the IEC 61131 standard whereby the programming paradigm inherent to a PLC is also supported and facilitates program development. The associated cost-effective and openness architecture make this a very feasible alternative. Although the advantages stated, the software running on a PC is not suitable for real-time operation [51]. So, a higher reaction time shall be expected whereby special careful must be taken to ensure that the normal operation of controlled process is not significantly affected.

CODESYS is a software platform for industrial automation technology. It offers a full support of all levels of automation control processes. At engineering level, it possibilities the configuration and programming of automation devices as well as appropriated tools to a correct debug and visualization of data. At device level, it provides a running system to converts a PC-based or embedded system into an industrial controller including optional add-on modules to interact with any I/O device. In addition, all the development software supports IEC 61131 standard. Therefore, the large support and the possibility of running on diverse platforms make this a common choice to soft-PLC role. The company provides a trial version that allows all functionalities, but with time limitations. A premium license has a cost depending from desired platform [54].

CODESYS also offers a wide support of fieldbus. However, to interface with industrial networks additional modules may be required. In *Profibus* case, the integration implies the installation of physical *Profibus* terminals [55] and the acquisition of supplementary software controller to register the device as a station. Still, different packages are required to act as master or slave station [56][57]. The purchase value of such gateways combined with extra software licenses cost make this interface approach too expensive on a low-cost project.

Furthermore, another option may be the TwinCAT software released by Beckhoff company [58]. Here, the automation control experience is very similar to the last one, including the support of engineering and device levels with diverse functions available to the user, but with more restrictions in platform application, being limited to windows-based environments. A full function software is freely available for 30 days. For a further use, a premium license must be acquired.

Additionally, there are also open-source options such as Beremiz [59]. However, generally, they represent projects in progress that may not achieve a sufficient stable solution for a real automation application.

## 2.5 Single-board computer (SBC)

The main element of a PLC operation is the processing unit. This module assures the execution of program instructions implemented at memory unit, the acquisition of inputs and the generation of respective outputs. So, an architecture to automation process control non-based on PLC implies

necessarily a core device with, at least, similar computational capability. A solution that may fit the required conditions is a single-board computer (SBC).

A SBC corresponds to a computer built on a single circuit board and, thus, composed by all the elements associated to this class, namely microprocessor, memory, I/O connections and eventually other kinds of interfaces. Although less powerful, these boards are characterized by hardware specifications (RAM, ROM, CPU speed) that can equate them to conventional PC and ensure similar operation principles. They are designed to educational purposes. However, the versatility combined with small size and low cost make them a very appealing solution to other application fields, such as automation industry [60].

At the market, a vast selection is available. Although the wide offer, *Raspberry Pi* and *BeagleBone* are the dominant options. Both present several versions launched, being the most recent ones the *Raspberry Pi 3* and the *Beaglebone Green*. The characteristics of each one are exposed on table 2.3.

Table 2.3: *BeagleBone Green* and *Raspberry Pi 3* specs [15][29]

|                              | <b>BeagleBone Green</b>                                                           | <b>Raspberry Pi 3</b>                                                                     |
|------------------------------|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| <b>Price (euros)</b>         | 45                                                                                | 35                                                                                        |
| <b>Processor</b>             | 1GHz TI Sitara AM3359<br>ARM Cortex A8                                            | Quad Core 1.2GHz Broadcom<br>BCM2837/64bit CPU                                            |
| <b>RAM</b>                   | 512MB                                                                             | 1GB                                                                                       |
| <b>Storage</b>               | 4GB eMMC on-board storage, MicroSD                                                | MicroSD                                                                                   |
| <b>Operating System (OS)</b> | Debian, Ubuntu, Android, Android, Cloud9 IDE on node.js, and others.              | Raspbian (recommended), Ubuntu, Android, ArchLinux, FreeBSD, Fedora, RISC OS, and others. |
| <b>Supply</b>                | 210-460mA @ 5V                                                                    | 150-350mA @ 5V                                                                            |
| <b>GPIO</b>                  | 2 46 pin headers                                                                  | 40 pin extended                                                                           |
| <b>Connectivity</b>          | 1 USB port<br>1 Micro-USB port<br>1 Ethernet<br>2 Grove connectors (I2C and UART) | 4 USB port<br>1 Micro-USB Power<br>1 Ethernet<br>1 HDMI<br>CSI camera<br>DSI display      |

In a brief inspection, it is possible to conclude that none of the two options dominates above other. *BeagleBone* provides a larger number of GPIO pins and an internal memory storage while the *Raspberry Pi* disposes of additional onboard connectors to interface with its own camera and LCD display. To support this kind of interfaces, the *Beagleboard* might be expanded through additional capes [16]. Besides the connections referred, both options also support UART, I2C and SPI interactions. On the software topic, they support several programming languages, including C, C++, Java and Python. Typically, *Raspberry Pi* is the preferable choice among the developers and hobbyists and therefore the online support is vaster [60].

In automation control, their presence has been increasing. Not only they have suitable connections to interact with field devices (*Ethernet* and serial communication) as well as there are soft-PLC solutions to ARM architectures. Specifically, CODESYS develops running environments to each one of these devices [61][62]. But, a SBC is not based on a real-time operating system. So, its performance must be carefully analyzed, as there isn't any sustainable data related to this topic. However, in this area, the *BeagleBone* may take advantage. The onboard processor is composed by two subsystems. A microprocessor unit (MPU) based on ARM architecture and, separated from this core, a programmable real-time unit (PRU). It is an independent processor with its own memory and instruction set, offering flexibility and high-speed response, suitable to real-time scenarios [63]. This unit eliminates the need of an external microcontroller to handle with such operations but requiring a learning curve and some additional libraries and tools.

## 2.6 Conclusions

On this chapter, it was explored the background concepts that must be clarified and understood due to the importance that they assume on next development stages.

Firstly, some base knowledge was provided, stating that the vacuum system is utterly mandatory to CERN particles experiments, allowing to reach an enormous high level of energy inside of accelerators. Pipelines vessel is characterized by an UHV, requiring a lot of niggling techniques and procedures as well as a proper instrumentation to maintain such low pressure. A vacuum group is composed by pumps and valves, being the combined operation assured by a PLC, through a state machine based process.

Justified by the relevance assumed on project context, PLC characteristics were studied. First, a brief reference to IEC 61131 standard was provided. This is like a manual for whole PLC lifecycle, defining the hardware robustness and software architecture. A PLC may have a fixed number of modules, single box, or, on opposite, being modular, built through the connection of several and distinct blocks. Included on this group, there are inputs and outputs ones. In case of inputs, they can be digital or analog while the outputs may be classified in transistor, relay or triac types depending from electrical characteristics of manipulated signals. Also, an I/O may be categorized as sink or source type, according with the direction in which flows the current. On software side, it was presented the hierarchical architecture, starting from configurations, at top, and ending on functions, located at the lowest level. This was complemented with stipulated programming languages, with more emphasis to SFC.

Another approached aspect is related with industrial communication. Between simple ICs, SPI is a common protocol, allowing synchronized serial data transaction over a full-duplex bus. On of the chip must act as master and, on the other pole, a slave is necessary. To set the devices communicating, select signals are used. For a more structured data exchange, it is used *Profibus* network, which also operates with master and slave stations and being compliant with seven layers

OSI model. Its implementation is ensured by industrial modules, being more guided for commercial purposes. However, there are some open-source projects that represent a legitimate path to be paved.

Finally, SBC architecture was explored aiming to choose a model for core unit role. *BeagleBone* and *Raspberry Pi* were compared, having, in general, similar specs with just some differences to available connections. The *BeagleBone* has the particularity of disposing of PRU that provides an operation equivalent to a pure microcontroller. Besides these considerations, soft-PLC solutions were identified to run on ARM core unit, being the most notable and compatible one the CODESYS software.

## Chapter 3

# Hardware Architecture

On this chapter, it is presented the hardware architecture. Firstly, on 3.1, it is provided an overview of physical system, discriminating the used devices and the premises beyond its application. On 3.2, it is detailed the design of each module of electrical board, the assumptions made during this process and the reason why. On 3.3, it is considered the physical support of the board, devices disposition and electrical robustness of hardware. After that, on 3.4, it is discussed aspects related with serial communication between the board and core unit, ending the chapter, 3.5, with some considerations associated to *Profibus* support.

### 3.1 Hardware overview

To satisfy project electrical specifications, it was necessary to develop an acquisition board whose purpose is to interface with pumping group through the several connections available. This assures electrical isolation, protecting the devices from each other and from core module, and exchanges data with core unit, in both sides.

According with the signal connections inherent to group, the board is composed by four main modules: power supply, digital input, analogical input and digital output. The respective design was made from scratch and supported on *EAGLE* software, from *AUTODESK*.

Regarding to core unit, the choice fell over the *BeagleBone*. When compared to its more close competitor, *Raspberry Pi*, it benefits from having an internal storage and, mainly, onboard real time units that complement microprocessor operation. This last consideration is the main reason behind that option, since it could be a major spec if the main CPU by itself wasn't enough to guarantee a timely response to the system. The model in use on the project is the *Green* one for being the most recent. However, the others are still compatible.

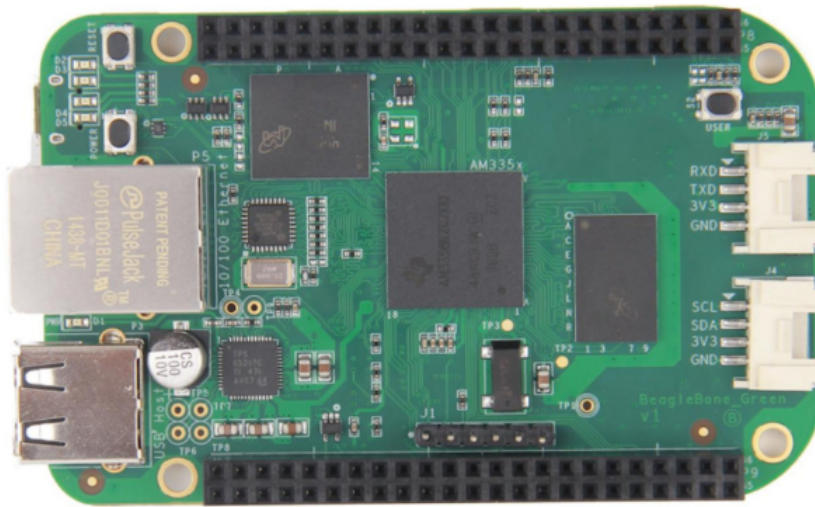


Figure 3.1: *BeagleBone Green* overview [15].

The platform includes, from default, an operating system on internal ROM. However, it can be flashed and the OS replaced by other version, or, even, an external microSD with an installed software may be used to boot the board. On this case, the device includes the Debian 9.3 version. But, for being too recent, it wasn't compatible with all used hardware and software libraries. For that reason, an older but more stable option was considered, Debian 8.7 with LXQT. This last package is recommended to have a full graphical desktop, accessible over virtual network computing (VNC).

In addition, to communication purposes with the electrical modules, it is used the SPI feature of *BeagleBoard*. The board pinout includes support for two SPI bus. By default, however, all the pins are set as GPIO. This is because the hardware architecture relies on devices trees, i.e., a device tree is a file that describes a particular hardware configuration, existing several ones already compiled one kernel, but not enable at boot moment. Being the hardware connected not automatically detected, the intended device tree needs to be manually loaded, otherwise the pins will just operate as normal GPIO. In this case, just one of the SPI bus is necessary to be set (see instructions on [64]). Besides that, to make the bridge between hardware and software a library is recommended to be installed. From default, pins control/manipulation is possible but through a low-level programming context. The installation of a proper library offers a higher level interaction, being just necessary to call some specific functions and provide the required arguments. The one selected is the *libsoc*. This is specifically developed for this kind of processor architecture, system on chip (SoC), representing a well known option among *BeagleBone* users, combining fast response with easy interface, with a smooth learning curve. The native library development is implemented on C language, but it is also offered an API to python compatibility. Anyway, in this case, it is privileged the first option, being the code written in C. The installation procedure is detailed on [65].

In order to support the HMI and aiming to be compatible with the already exposed resources, it is considered a *BeagleBone* cape (i.e. a plug-in board that may be connected to main board

providing an extension of its functionalities) composed by a 7" LCD display [16]. The board connects behind the cape and the running desktop is immediately projected (if LXQT is supported by the installed OS). It offers a touch interface and its operation relies on resistive principles. The cape selected is specifically indicated to work only with *Green* model and the operating system must be the one indicated. With newer kernel versions, it is not guaranteed compatibility.



Figure 3.2: *BeagleBone* LCD display cap [16].

Regarding to *Profibus*, the physical layer of protocol is based on RS-485 serial communication. Since the *BeagleBone* doesn't include the required port, it is taken advantage from USB host connector. Thus, it is utilized a RS485-USB converter.



Figure 3.3: RS485-USB adapter.

The articulated integration of all indicated tools support the hardware architecture. On picture 3.4, it is presented a symbolical representation of its schematic.

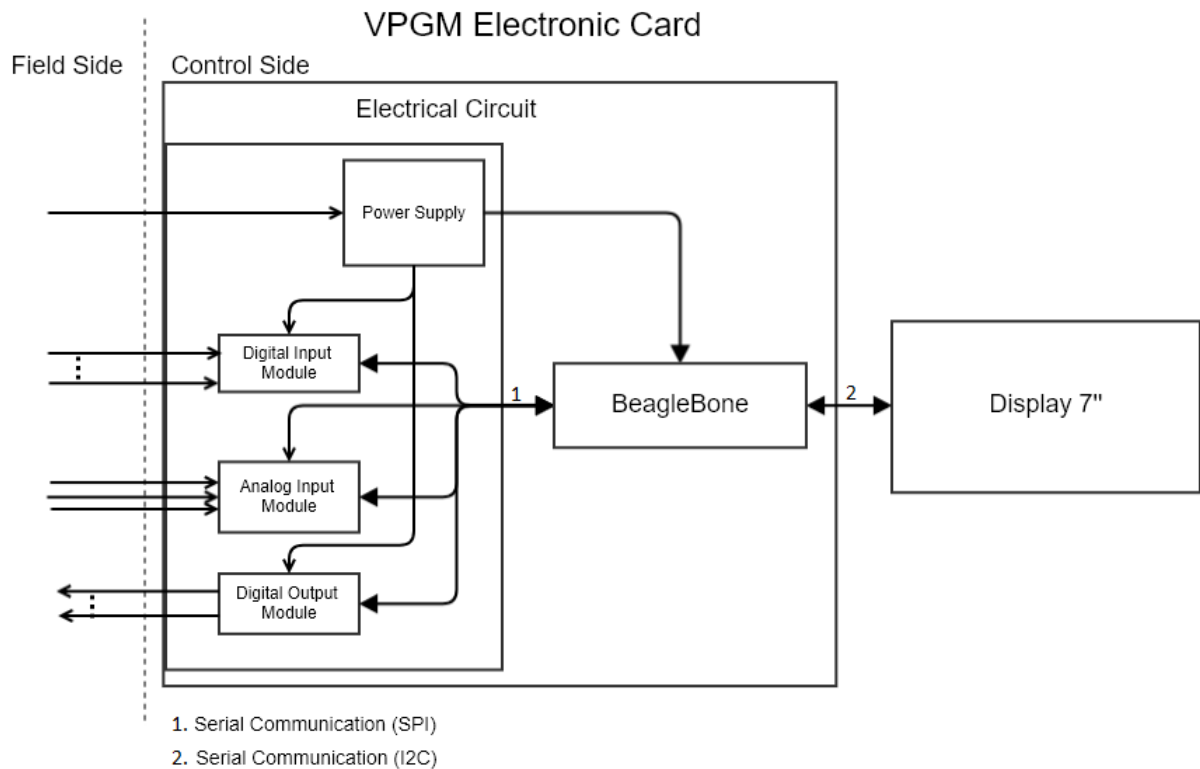


Figure 3.4: Hardware architecture.

Briefly, the whole system is supplied by group UPS, providing 24V-2A. The voltage regulation occurs at power supply module, being responsible to provide all the required voltage levels to properly feed the board loads, including *BeagleBone*. The digital and analogical modules are associated to connection with field devices signals, being the information shared with the SBC through a SPI bus, where the devices act as slaves and the *BeagleBoard* as master.

Relatively to display, it is linked to *BeagleBone* through a flat cable that carries all the required signals and the connection relies on serial interface, namely I2C.

## 3.2 PCB modules design

The card is composed by several modules, being each one linked to a specific function as already identified. In the next topics, it will be described more carefully each one of them, indicating the electrical components required to the intended operation, their sized, the principles inherent to that choice and, finally, the electrical schematic. The link between the card and the SBC is established through SPI interface. Thus, the integrated circuits shall be selected according with this assumption.

### 3.2.1 Digital inputs

One kind of inputs signals is the digital group. These are voltage signals that, at a specific moment, only assume one of two possible voltage levels. A low logical level, zero, is related to 0V and an high logical level is associated to 24V. So, it's not only necessary to get the signal information as well as adjust the voltage level to *BeagleBone* range, 0-3.3V.

To fulfill announced requirements, it is used *serializer* ICs. This kind of devices allows to acquire multiple parallel signals on input side and export them on a serial way. Then, it is possible to reduce the number of input channels to just a serial interface. The quantity of input connections and the serial type are parameters that vary with the model considered.

In this case, the digital inputs represent a total of 29 connections. To assure an interface of this kind, multiple devices must be used. Normally, they offer eight input signals, what implies the use of four ICs, being the remaining connections available for additional applications. The selection of a specific model among the whole *serializer* offer was supported on the characteristics of the I/O PLC module that currently assures the acquisition of digital signals on the pumping group. So, the main goal was to assure, at least, the same electrical characteristics of the module to guarantee a full compatibility with physical devices.

Consulting the *datasheet* of PLC module, the relevant electrical parameters are registered on table 3.1.

Table 3.1: Electrical characteristics of PLC input module [30]

| Logical Level (voltage) | 0 (0V)     | 1 (24V)      |
|-------------------------|------------|--------------|
| Input Voltage           | 5V (max.)  | 15V (min.)   |
| Input Current           | 1mA (max.) | 2.5mA (min.) |

The selected device must match these specifications in order to a logical level not be misinterpreted and cause an erroneous device status acquisition. Therefore, one model that fulfills the desired requirements is: SN65HVS881 [17]. This is a *serializer* with SPI serial communication that supports digital input signals from a range between 10V and 34V. Among all electrical characteristics, the most relevant for this application are represented on table 3.2.

Table 3.2: Electrical characteristics of *serializer* [17]

| Logical Level (voltage)                         | 0 (0V)    | 1 (10-34V). |
|-------------------------------------------------|-----------|-------------|
| Field input voltage ( $V_{IPL}$ and $V_{IPH}$ ) | 4V (max.) | 5.5V (min.) |

Comparing these aspects with the PLC ones, it is possible to conclude that the voltage range inherent to I/O module are included on the IC selected, existing, however, one slight difference on low logic level. Here, there is 1v of offset between the thresholds of the two devices, and, since it is less than the *serializer* voltage hysteresis, the output logic level may be unpredictable if the device operates to near from its limits. However, this issue is mitigated by the prior processing signal circuit.

On the other hand, the input current limit is customizable and established by an external resistor, that must be dimensioned between  $17\text{K}\Omega$  and  $500\text{K}\Omega$ , with a nominal value of  $25\text{k}\Omega$ . Based on the device *datasheet*, this parameter calculation relies on the following formula:

$$R_{Lim} = \frac{90}{I_{Lim}}$$

To the nominal value, the current limit is  $3.6\text{mA}$ . In this case, it is expected a maximum current of  $4\text{mA}$ . This corresponds to a  $22.5\text{K}\Omega$  resistor. Despite this limit, a current higher than  $2.5\text{mA}$  (I/O module) generates a voltage drop enough to be classified as a logical '1'. For that reason and to not operate the component near to its limit conditions, the nominal value is used.

The device also provides additional functionalities that can be explored.

In first place, it disposes of two configuration pins for an internal debouncing filter, whose purpose is to suppress unintended signal transitions due to contact bounce or other mechanical effects. These two pins allow four possible configurations, that diverge on the filter time constant.

Table 3.3: Input filter configuration [17]

| DB0  | DB1  | Function  |
|------|------|-----------|
| Open | Open | 3ms delay |
| GND  | Open | 1ms delay |
| Open | GND  | Bypassed  |

Vacuum process is characterized by its slow motion and, consequently, the time constants are not too strict, with a magnitude near to seconds. With this in mind, the two pins are left open and, therefore, the input signals will be exposed to a low pass filter with a cutoff frequency of, approximately,  $80\text{Hz}$ .

The *serializer* also has incorporated a DC-DC regulator. It provides fixed  $5\text{V}$  as output to supply not only internal elements as well as some external ICs. However, as it is referred on 3.2.4, since there is a  $5\text{V}$  regulator, this extra supply isn't necessary and it isn't used. Nevertheless, to not left the pin floating and following the *datasheet* recommendations, it is placed two capacitors, whose purpose is to stabilize the voltage and avoid fluctuations. The values indicated are  $1\mu\text{F}$  and  $0.1\mu\text{F}$  on parallel configuration.

Furthermore, the internal circuit includes some interface pins that are connected to inputs and allows to observe the current state of each one. So, a LED may be linked and, this way, it is possible to the user to visually identify the current state of each channel. However, to the present purpose, it is not considered a mandatory functionality and it is discarded. But, it can be reused in the future if facilitates the debug process.

To increase the device robustness and reliability when operating in adverse industrial conditions, it is recommended on *datasheet* two additional measures besides the ones already addressed. Firstly, regarding to power supply, an external circuit should be used in order to provide a supply

voltage so stable as possible, suppressing burst and surge-transients. and to ensure device protection against electrostatic discharges. Indeed, internally, the *serializer* includes such kind of circuit, but, with this approach, it is offered protection redundancy and, providing a conditioned supply, the device immunity is increased. On the other hand, on the input signal, the situation is similar. Despite of the internal filter to the input signal, external components are recommended, namely a low-pass filter. Therefore, each signal will be processed along two filtration stages, being expected more accurate results. Based on these assumptions, the configuration of picture 3.5 shall be considered.

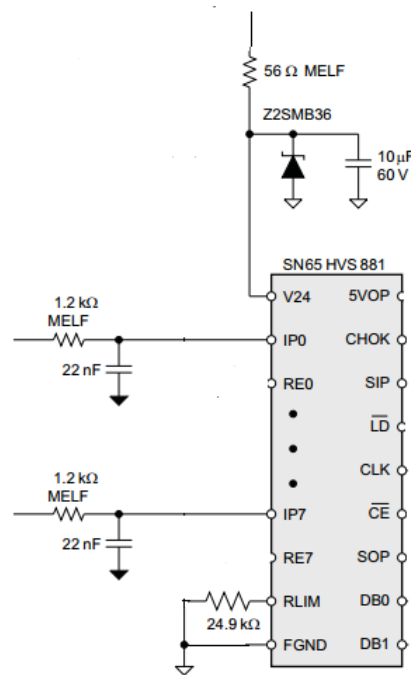


Figure 3.5: *Serializer* input protection circuit (adapted from [17]).

The previous ideas have the purpose to raise the IC robustness, supporting abnormal conditions without damaging any internal element. However, there isn't still electrical isolation not only with the core unit as well as among channels. So, in the intended application, such kind of gap can generate chaotic results. If it is allowed current paths between inputs, it means that, for instance, a problem with the turbo pump can be propagated to the primary pump either, and, in the worst scenario, it can cause the destruction of the whole pumping group. Also, the possible destruction of the core unit would not be nice either, because the control of group would be compromised and its behavior unpredictable with possible dangerous consequences. Thus, these two kinds of isolation are a must.

To fulfill the previous requirements, each channel is preceded by an *optocoupler*. This breaks the current loop which could eventually be described between inputs and, electrically, isolates the field side from the control one, where is located the core unit. The device selection shall be made according with the required voltage levels. Here, the field devices provide 24V. On the isolated side, the voltage is assured by the power supply module of the card. Since the *serializers* only

work with voltage levels above 10V, 24V seems the natural choice. Supported on these facts, it is selected the TCMT4100 IC [66].

The interface circuit exposed on picture 3.6 is considered.

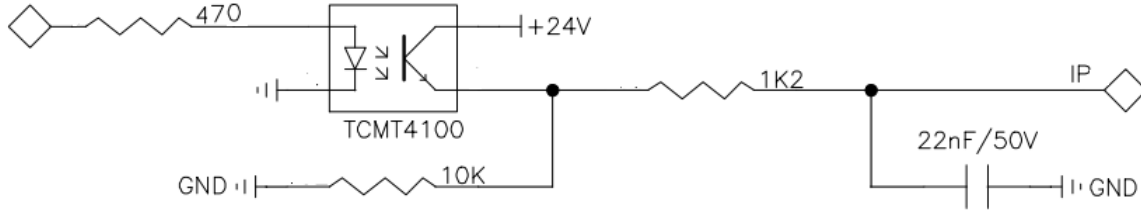


Figure 3.6: *Serializer* input isolation circuit.

So, in a no conduction state, the input voltage is 0V. There is no current path through the *photodiode* and it won't be polarized. Consequently, the transistor is on cutoff mode. Since the *serializer* input pin is pulled-down, the voltage will be zero and, consequently, a low logic level will be generated on the internal shift register. On the opposite scenario, when the field input is supplied with 24V, the transistor is at conduction mode and 24V will feed the input and be interpreted as a high logical level. Additionally, as the *photodiode* only support a limit reverse polarization voltage, an anti-parallel diode is included to prevent the circuit damage in case of accidental reverse connections.

The resistors calculation was made according with the electrical characteristics inherent to I/O PLC module and already presented on the beginning of this topic. Moreover, it is based on the *optocoupler datasheet* to set the intended point of polarization.

In this application, the input voltage is 24V and the IC LED has a typical forward voltage of 1.35V. Polarizing the device to be able to provide the maximum current on collector side, state at which corresponds a forward current of 50mA, the input resistance is given by:

$$R_{in} = \frac{24 - 1.35}{0.050} \cong 470\Omega$$

The output resistance should be dimensioned in such a way that, when not conducting, the leakage voltage is low enough to the input be classified as a low logical level and, when on, 24V are acquired. That's why, it should be a pull-down configuration. Regarding to device, to a forward current of 50mA and and output voltage of 24V, the collector current may achieve 70mA, approximately. On off mode, the dark current is at maximum 100nA [66]. A pull-down resistor of 10KΩ guarantees that, when not conducting, the output voltage is not enough to be interpreted as a logical true value (in such case the voltage is 1mV, being below the *serializer* threshold). On the opposite scenario, a current of 2.4mA ( $24 \div 10K$ ) passes through resistor and the remaining, almost 70mA, is more than enough to fed the *serializer's* input, here preceded by a RC filter.

One *optocoupler* is applied per each input channel along the four *serializers* required for the inputs signals presented on the vacuum group.

The communication of this module with the *BeagleBone* is made through SPI interface as already said. A more detailed explanation related with the serial communication between the core and the card ICs is provided on [3.4](#).

### 3.2.2 Analog inputs

Besides the digital inputs, there are also analog input signals present in the group that must be properly conditioned and acquired by the card to posterior software processing. These signals are related to the vacuum gauge pressure, primary pump current and the bake-out temperature. The two first ones are 0-10V signals, while the temperature is provided by a thermocouple type-k. Thus, an acquisition circuit shall be dimensioned for each kind of interface.

#### 3.2.2.1 0-10V inputs

The primary pump and the vacuum gauge provide analog feedback signals of 0-10V type. To a correct acquisition, an ADC based component is necessary with the support of serial stream through SPI of the captured data.

Currently, the acquisition is assured by a PLC analog module. This has 10 bits of resolution and a conversion time of  $625\mu\text{s}$  [67]. So, the searching process for an IC to replace its functionality must be led by this specs, in order to assure, at least, an equal resolution and conversion performance, with the additional support of 0-10V input range and serial interface.

One component that has the desired characteristics is the ADS8665 [18]. This IC offers a 12 bit successive approximation based ADC with a sampling frequency of 500SKPS and a maximum conversion time of  $1\mu\text{s}$ . The maximum voltage range is  $\pm 12.288\text{V}$ . However, due to an integrated PGA, its gain can be adjusted and, then, distinct acquisition ranges can be configured, both unipolar and bipolar ones, to obtain a higher resolution as possible. One of that ranges is 0-10.24V, matching the required application.

The utilization of the device implies a proper external circuit. This sets the voltage supply, reference voltage to internal operation and the conditioning circuit to the input signals. Also, some software actions through the serial interface are required to set shift registers configuration.

Regarding to hardware, the component has the pinout depicted on the picture [3.7](#).

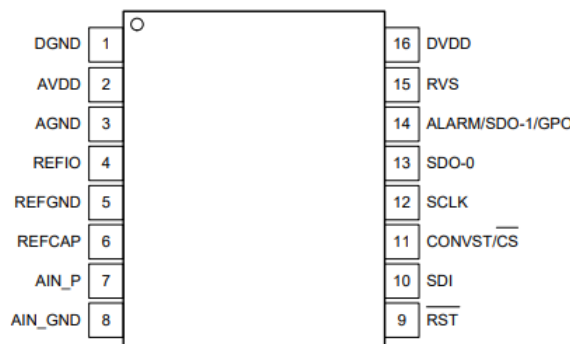


Figure 3.7: ADS8665 pinout [18].

It is possible to distinguish two supply pins, being one to analogical part and the other related with the digital circuit and the logic module.

According with the *datasheet* [18], the analog side has a nominal value of 5V. This is used to feed the acquisition components and it is recommended a decoupling capacitor to remove unwanted transient behavior. However, due to the noisy behavior characteristic of analog devices, an additional low-pass filter, based on a RC circuit, is configured to raise the device immunity and assures a supply as stable as possible. On the digital side, 3.3V represents the nominal value and a  $10\mu\text{F}$  capacitor should be used to a more stable supply, not being necessary a so cautious solution like the one adopted on analog side. Both voltage levels are assured by the power module of the card, implying, consequently, the voltage adjustment from 24V (provided by the UPS) to 5V and 3.3V, respectively.

Besides the supply of internal components, additional connections are necessary to a correct acquisition. To its internal operation, the ADC requires a voltage reference which is by default 4.096V [18]. However, this value can be provided externally through the REFIO pin. In this case, the internal reference is applicable whereby this pin is configured as output of the internal voltage but must not be used to supply loads. It should be connected through a capacitor to the REFGND pin. This acts as a low-pass filter and suppress the internal noise, increasing acquisition accuracy. A  $4.7\mu\text{F}$  is the minimum value recommended. The REFCAP pin has a similar purpose and it should be connected to two capacitors ( $1\mu\text{F}$  and  $10\mu\text{F}$ ) on parallel configuration, aiming to reduce the voltage ripple of the ADC reference.

After that, it remains the input signals connections. These pins are internally connected to a proper protection circuit that precedes the PGA. This protection circuit is composed by ESD protection, input current limitation and over-voltage withstand. Each input has, in series, a high value resistor that, in series with the PGA level adjust resistances, causes a limitation of input current, assuring the electrical integrity of the device. In addition, there are also two extra diodes setting the low and high level of voltage allowed and, so, the input saturates out of this range. The combined action of these features enables a protection from input voltage up to  $\pm 20\text{V}$ . After that, the component is in risk. Besides the protection topic, there is also a conditioning signal module whose purpose is to clean the acquired data from unintended disturbances. With this in mind, a low-pass filter is included on the digital domain, removing any oscillation above 15kHz.

Despite of the internal protection and signal processing circuit described, this should be faced as a backup measure. It means that, externally, a similar configuration must be considered and enable the same features, being the inside one just for use in case of external failure and not for continuous application. So, an equivalent external circuit shall be dimensioned.

The first step consists in protect the device from a high input current. This feature can be easily enabled through a resistor. If the input resistance is sufficiently high for the operation voltage, the current will be limited to values that cannot put the device in dangerous conditions of acquisition. Regardless of this, the ADC already includes a high input resistance whereby it is established a voltage divider. That's why, the external one value must be large enough to protect the device but also not too high, because an elevated voltage drop might cause an erroneous interpretation.

Normally, a good value is between  $100\Omega$  and  $10k\Omega$  [68].

The model selected has an internal resistance of  $1.2M\Omega$  as nominal value and a resolution of 12 bits. Additionally, the LSB has a value of  $2.5mV$  [18].

So, for an input resistor of  $1k\Omega$ :

$$\Delta V = \frac{1K}{1K + 1.2M} \times 10V \cong 8.3mV > \frac{1}{2}LSB$$

On the other side, for  $100\Omega$ :

$$\Delta V = \frac{100}{100 + 1.2M} \times 10V \cong 0.83mV < \frac{1}{2}LSB$$

The resistor value must be chosen in a way that the largest voltage drop is less than an half of the LSB. Just respecting this assumption, it won't occur misinterpretations. Being the highest voltage drop the one related with  $10V$  as input, with an input resistor of  $1k\Omega$ , this value is  $8.3mV$ , not matching the required criterion. In this case, it is lost, at least, 3 bits of resolution. Since the ADC is over-sized when compared to the PLC analog input model, this condition might not be too critical and the results still acceptable. But, if the resistor value is  $100\Omega$ , the worst case scenario respecting to voltage drop is within of the intended range. For that reason, this is the preferred choice.

Additionally, the circuit must also execute some processing over the signal in order to remove high frequency noise. This can be achieved through a simple first order low-pass filter. To this purpose, a RC circuit is enough. Thus, it is just needed to add a capacitor after the previous resistor. The capacitor value should be calculated according with the desired cutoff frequency. Here, the internal parameters of the ADC are maintained and, therefore, a value of, approximately,  $15KHz$  is considered.

$$R \cdot C = \frac{1}{2 \cdot \pi \cdot f} \Leftrightarrow 100 \cdot C = \frac{1}{2 \cdot \pi \cdot 15K} \Leftrightarrow C \cong 0.1\mu F$$

Thus, a  $0.1\mu F$  capacitor is used.

Finally, the missing module is the protection against over-voltage. This circuit allows an input voltage within a configured range and, above its limits, saturate the signal, safeguarding the IC. In this topic, two possible paths may be followed. For one side, a *zener* diode may be used, clipping the input signals between the threshold levels, or two separated diodes, one pulled-up and the other one pulled-down. Aiming to approach the device internal configuration, the last option is used. Besides that, the goal is that the external protection overlaps the internal one. For that, the diodes shall actuate previous than the internal circuit whereby *schottky* diodes must be considered. This kind is characterized by a lower forward bias, approximately  $0.3V$ , compared to the common  $0.7V$  and, consequently, starts to conduct first. Lastly, but not least, it is recommended to precede them by a series resistor in order to limit the current circulation when conducting and to not exceed the diode power dissipation limit. A  $10\Omega$  resistor is suggested [68].

The whole configuration described must be applied to both input pins to assure the symmetry among signals. The combination of all previous ideas results on pciture 3.8 acquisition circuit.

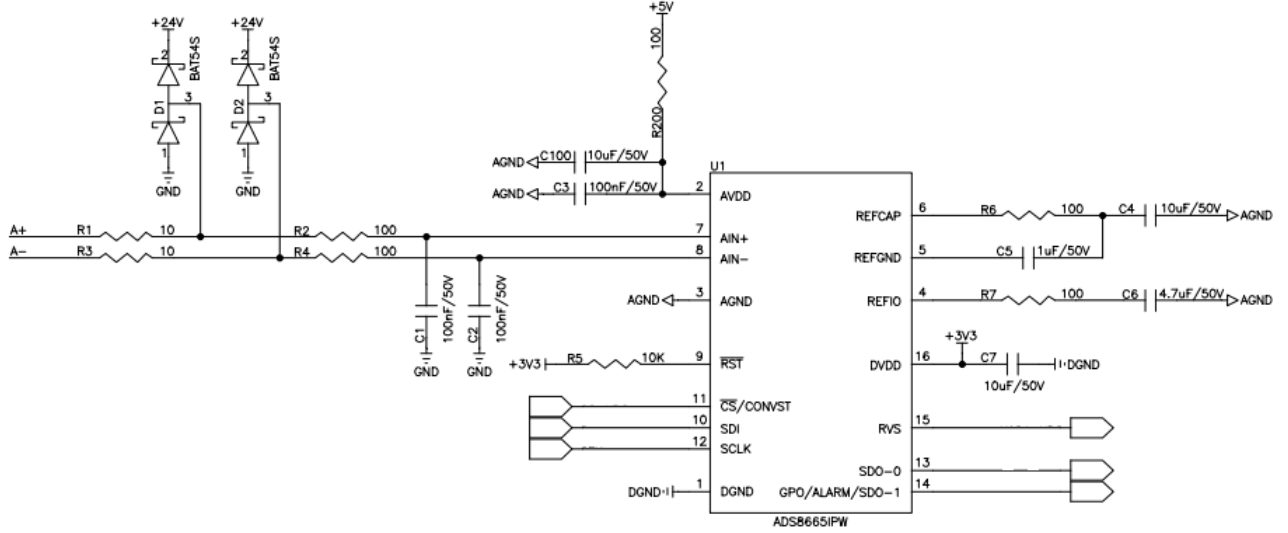


Figure 3.8: ADS8665 acquisition circuit.

The remaining pins not referred are related to serial interface and alarm features. They are all connected to *BeagleBone* GPIOs and the last ones act as mean of diagnosis, flagging abnormal conditions of operation. These aspects are more detailed on 4.3.

In this application, it is intended to acquire two analog signals. So, two circuits equal to the previous configuration are included on the electronic board. Instead of this solution, it could be chosen an IC that had two or more inputs and, this way, allow the connection of multiple devices and save space on the board. However, since the signals are provided by different physical devices, vacuum gauge and primary pump, the use of distinct acquisition circuits represents a more robust approach, ensuring isolation between group equipment.

### 3.2.2.2 Thermocouple input

The bake-out temperature is sensed by a *thermocouple*. This kind of transducer generates an output voltage signal that provides the analog temperature value from a range between -80mV and 80mV. There are several types of *thermocouples*, diverging on the composition material and the temperature range in which should be applied. In this case, it is used a type K one that is suitable for temperatures above -200°C and below 1260°C.

Currently, the acquisition and conversion of *thermocouple* data is assured by a PLC module specifically adapted to this temperature sensors [69]. This module supports the interface with almost all types of *thermocouples* and there is no characteristic that deserves mention except the input voltage range as previously referred and the number of bits which is 15 plus the sign one.

To replace the previous module operation, a generic ADC can be used since the acquisition voltage range is compatible with the provided. However, as it assumes a very low amplitude, it

would be hard to find one option with a resolution equivalent to the present solution. That's why, it is selected a tailor-made IC for *thermocouple* interface, MAX31856 [19]. Analogous to the PLC module, it allows the connection with distinct types, including the k one, has a resolution of 19 bits and export the information through SPI bus, being compatible with the communication protocol inherent to the remaining ICs of electronic board. Through this interface, it is possible to configure several internal registers and read data on the opposite direction.

The device pinout is presented on picture 3.9.

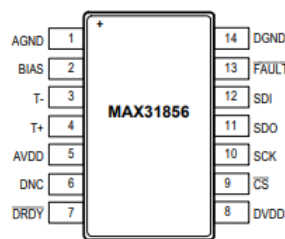


Figure 3.9: MAX31856 pinout [19].

Regarding to supply connections, the device must be fed in both analog and digital sides. Both have 3.3V as nominal value. On the analog side, like already described on the previous analogical signals, there is a low-pass filter before the supply pin, due to the noisy and disturbances inherent to analogical application. With this procedure, the circuit immunity and supply stability are raised.

The remaining connections were made following the *datasheet* suggestions. Firstly, a capacitor should be connected between the two input lines acting as a filter of the noise inherent to channels. A value of  $100\mu\text{F}$  is recommended. This procedure should be complemented with additional capacitors between T+ and  $V_{CC}$  and T- and GND pins. In both cases, it is used a value of  $10\mu\text{F}$ . Additionally, a series resistor might be connected in each input signal. Like in previous analog acquisitions, it develops a dual role creating a low-pass filter and limiting the input current. A  $100\Omega$  resistor combined with the capacitors assumptions ensures a higher noise rejection robustness and a device protection over  $\pm 45\text{V}$  [19].

The other pins are related to serial communication and device diagnosis and are directly connected to *BeagleBone* GPIOs. The DRDY pin signals the end of conversion of a new acquired value and the alarm one flags a value outside the temperature expected limits. Besides that, additional configurations through software are possible and are more detailed on 3.4.

Hereupon, it is obtained the final configuration represented on 3.10.

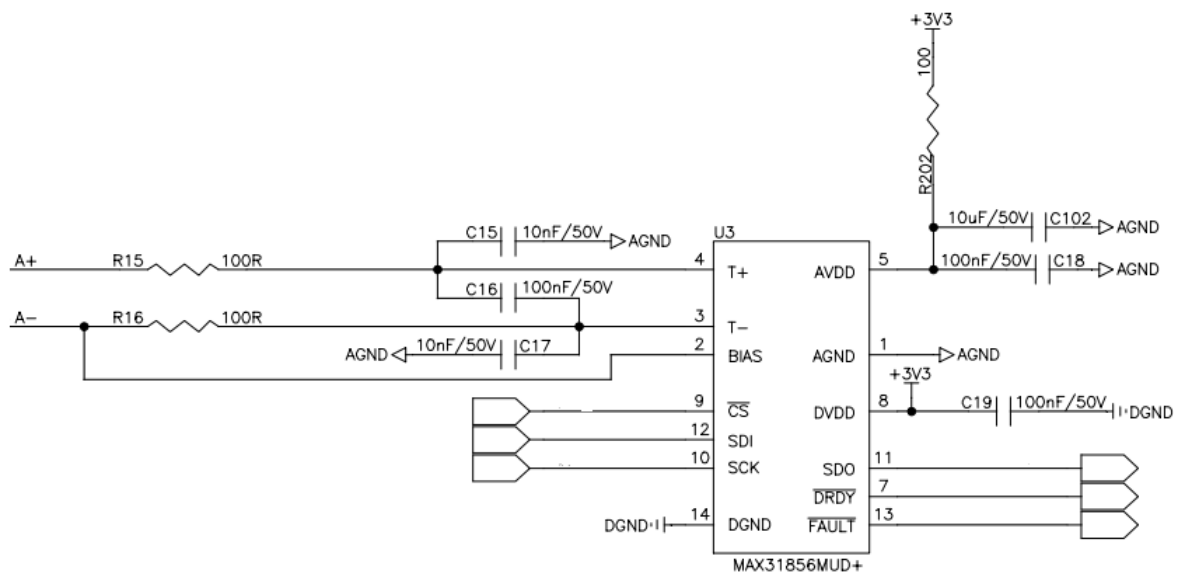


Figure 3.10: MAX31856 acquisition circuit.

### 3.2.3 Digital outputs

The output signals represent actions sent by the user to actuate on field equipment. Regarding to valves, these orders allow to control the open and closed state, while on pumps side, it is permitted to manipulate their operation mode (on/off, reset or standby mode). These interactions are based on digital signals, with 0-24V voltage level, analogous to digital inputs. In total, there are 17 output signals.

To build a solution able to satisfy the requirements implied for such application, the current PLC module was studied and its principal characteristics registered. It is transistor high side output driver and, consequently, on a source configuration, feeding the external circuit, and with the electrical characteristics indicated on table 3.4.

Table 3.4: PLC digital output module characteristics [30].

|                         |             |
|-------------------------|-------------|
| Voltage Level           | 24V         |
| Max. Output Current on  | 500mA       |
| Max. Output Current off | 10 $\mu$ A  |
| Turn-on Delay (0->1)    | 50 $\mu$ s  |
| Turn-off Delay (1->0)   | 200 $\mu$ s |

The new solution dimensioned must be suitable with the present specifications to assure full compatibility with the group. Besides that, other aspects shall be considered, namely the interface with the host controller. The approach used in the other chips should be maintained, promoting a serial communication instead of a parallel one. Here, this criterion assumes a considerable weight, since, due to the large amount of connections, a parallel interaction would imply an equivalent number of GPIOs accessible on the *BeagleBone* and wouldn't be a feasible option.

Taking into account the explained premises, the model VNI8200XP is selected [20]. This chip consists in a transistor high side driver that allows the connection of eight output parallel loads. Additionally, it supports both serial and parallel interface and offers several diagnosis features to monitor the device operation. The main electrical parameters, checked on *datasheet*, are represented on table 3.5.

Table 3.5: VNI8200XP characteristics [20].

|                        |            |
|------------------------|------------|
| Voltage Level          | 10.5-36V   |
| Nominal Output Current | 500mA      |
| Turn-on Delay (0->1)   | 10 $\mu$ s |
| Turn-off Delay (1->0)  | 15 $\mu$ s |

Comparing these values with the PLC module, it is verified immediately that the device performance is similar to this last one and, therefore, it can replace its functionalities with full compatibility.

Hereupon, a proper external circuit must be dimensioned. On picture 3.11, it is presented the device pinout.

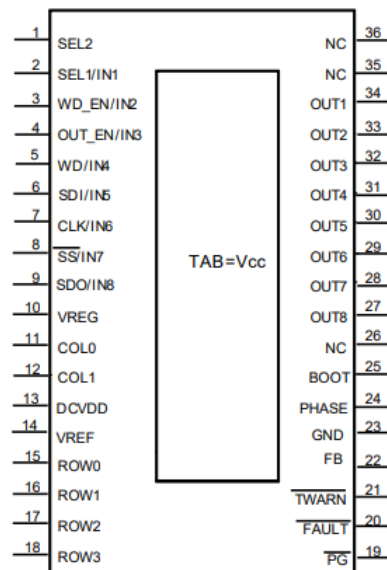


Figure 3.11: VNI8200XP pinout [20].

Related to device power supply, a lot of aspects should be mentioned. In first place, the main supply is connected to  $V_{cc}$  pin and corresponds to a voltage level of 24V. Once it is dealing with digital signals, no particular care is necessary, whereby the pin can be fed directly, being suggested just a single capacitor to decoupling effects. The device still disposes of other supply paths. The VREG connection is related to digital logic operations, including SPI communication. A voltage range between 0 and 6V is allowed, using, in this case, 3.3V for being the voltage level associated to *BeagleBone* GPIOs. The VREF pin is an output and is related with an internal DC-DC converter.

The device is composed by an internal step-down regulator whose purpose is to feed a LED matrix that indicates the current state of each output (accessible through COL and ROW pins) or external devices, for instance, *optocouplers*. The output voltage of such regulator is settled on the feedback pin, FB, through a resistor divider or linking directly the regulator output, assuming, on this last scenario, a fixed value of 3.3V. Here, this functionality is not essential and can be suppressed connecting the feedback to the VDCC. The remain pins associated to step-down converter are left open.

The chip includes some diagnosis functionalities. The first one is a watchdog to monitor the communication with the master unit. When enabled, this functionality implies a periodic negative pulse on WD pin to reset an internal watchdog. If this action doesn't happen within the time range, the outputs are putted on safety state, i.e, are all turned-off. This functionality is deactivated setting the WD-EN to ground. Secondly, there is a Power good (PG) pin either. This is flagged when the supply voltage is below of a fixed threshold, 9.5V. The TWARN signals an over-temperature condition of the IC case. The fault one is activated when occurs a parity check fail on serial communication or a channel over-temperature. The output enable pin provides a fast way to disable all outputs at once, what can be useful in an emergency situation, bypassing the delay inherent to serial communication. All these pins are connected directly to *BeagleBone* GPIOs, being processed on software side. Finally, there is yet a connection to switch between serial and parallel interface, being the serial one configured. If parallel mode is chosen, almost all the diagnosis functionalities will be overlapped and won't be available to use.

The output pins should be connected to the intended loads. However, to raise the output protection and safeguard the driver, two diodes shall be included. This circuit aims to protect the internal circuit against back EMF when switching inductive loads and, additionally, when the load is accidentally reverse connected, providing an alternative path instead of destroying the IC.

According with all the assumptions made, the final schematic is presented on [3.12](#).

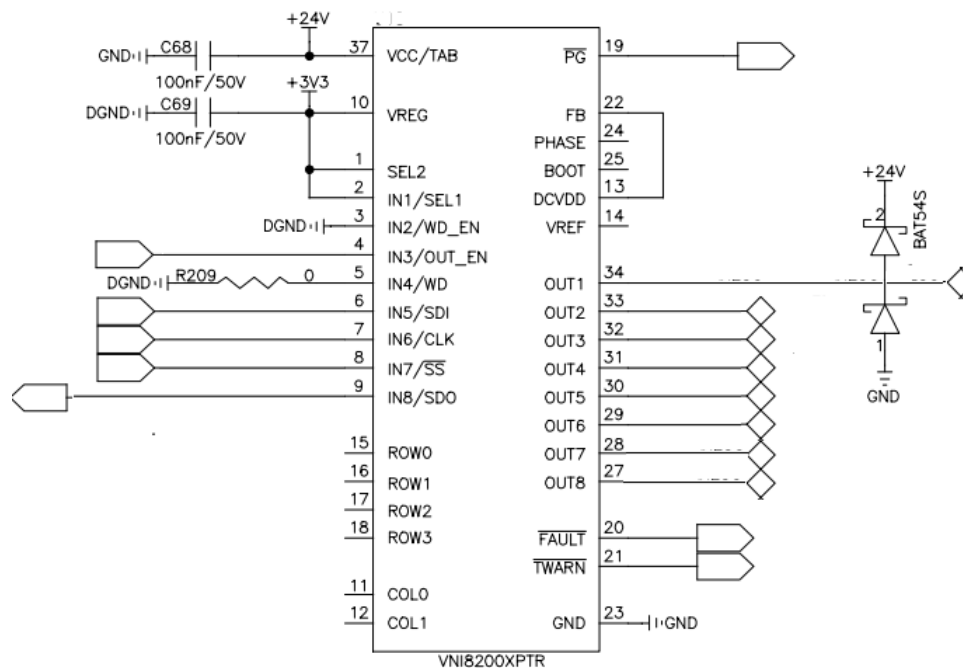


Figure 3.12: VNI8200XP configuration.

There is a total of 17 outputs. To develop a solution suited to this requirement, three similar circuits shall be assembled, being the remaining free outputs available to interface any extra load.

### 3.2.4 Power supply

The power supply module is intended to feed all the integrated circuits present all over the board and should be compatible with the mains power source of the group, 24V/2A. So, starting from this voltage level, the necessary adjustments shall be made. Paying attention to the information previous referred to each device, it is possible to identify three distinct supply contexts: 24V, 5V, 3.3V.

The 24V are directly provided by the group and are also required for the digital inputs and outputs. For one side, the *serializers* are exposed to input signals until 24V and, for that reason, must be fed with an equal voltage level. On the other hand, the outputs drivers must set 24V to feed properly the loads. The 24V sourced on the group are the output of an UPS included for emergency conditions in which the mains connection fail. Therefore, the resultant signal is already incipiently filtered, but additional care is recommended to be taken. Despite the no need of isolation, since it occurs at a different level, a condition circuit to protect against electrostatic discharges and remove the noise inherent to harsh industrial environments is a convenient procedure to guarantee a clean and stable power supply without compromising the devices operation and internal circuit. Aiming to satisfy this purpose, a circuit is dimensioned based on the indications present on [17].

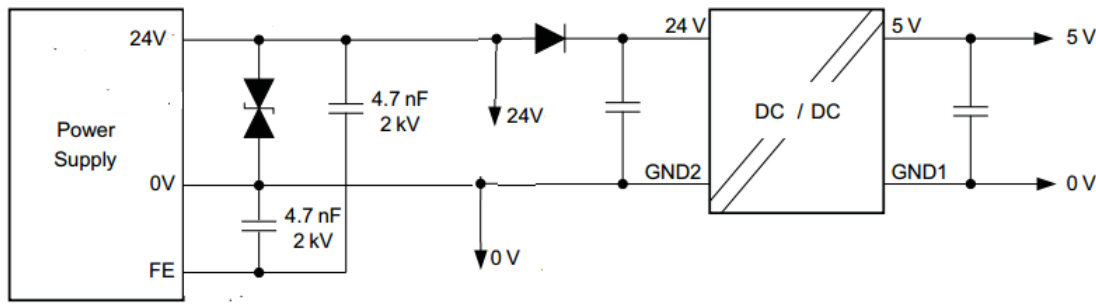


Figure 3.13: Power supply protection (adapted from [17]).

After this, a voltage drop from 24V to 5V must be performed, being this last one the level required for ADCs analog side operation and to supply the core unit, *BeagleBone*. However, since it is a considerable difference between the levels involved, the use of a simple low drop-out wouldn't be energetically efficient. A low drop-out corresponds to a voltage regulator in which the output is obtained through a linear component, such as a resistor. So, despite its simplicity, it isn't profitable, generating a lot of heat due to the power dissipation implied when the in and output voltage are too far. Instead, a regulator based on switching mode operation is preferred. This kind is characterized for transform the input into a pulsed voltage and then, through capacitors, inductors and a switch component, the output is fixed in the value configured. For this, the additional complexity allows better performance regarding to energetic criteria and high speeds of operation. But, the high frequencies related to that last behavior are a common source of noise and, for that reason, the regulator must be allocated on the circuit in such a way that not compromises the surrounding elements operation, namely the ADCs for being the most sensible ones regarding to this issue.

For fulfill the position considered, the TRACO power offer range was searched. Two high level types are accessible, the isolated and non-isolated ones. As previous referred, in this case a non-isolated solution is enough. Inside this ones, the selection relies on the power intended. To calculate this parameter, the downstream loads to be fed need to be analyzed. The most significant is the *BeagleBone*. The recommended supply current is at least 1.2A that corresponds to a consumption power of 6W [15]. The remaining devices are the ADC, *thermocouple* acquisition IC and logic supply of output drivers. All of them have a supply current in order of some mA, whereby it is insignificant when compared to core unit. Hereupon, a 10W regulator is chosen. On the component *datasheet*, it is suggested to put a  $22\mu\text{F}$  capacitor before the regulator input. This acts as a bulk capacitor and has the purpose to reduce the input ripple voltage to a level that not affects output stability.

Finally, to obtain the 3.3V, it is used a low drop-out regulator. Its application on this scenario is feasible since the voltage levels are sufficiently close to the drawbacks of a linear regulator represent a threat to the operation. For this application, since it is only necessary to provide a few mA of current, it was over-sized a maximum current value of 2A. The NCP5562 may be used. Also, here, the *datasheet* recommends to use an input capacitor of, at least, of  $150\mu\text{F}$  and output

one of  $15\mu\text{F}$ .

Based on these facts, the final circuit is the following on picture 3.14.

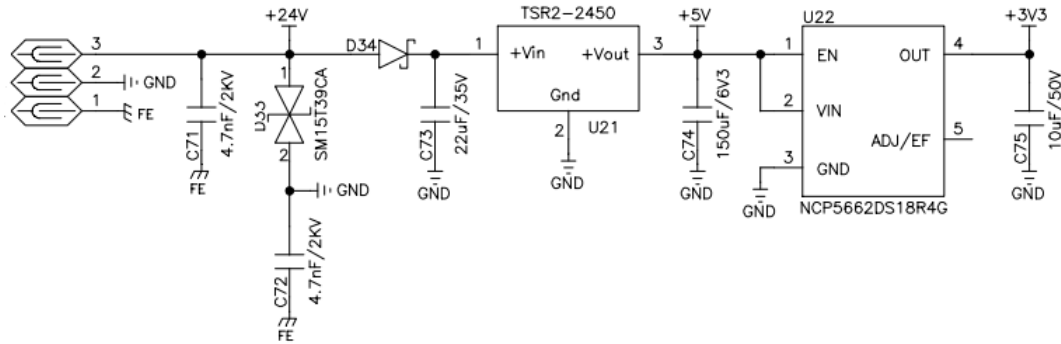


Figure 3.14: Power supply circuit.

As said, the 5v and 3.3v are used to supply simultaneously both analog and digital side of *thermocouple* and ADCs. Although, the analog signals are generally characterized to be too noisy and, when sharing the supply line, it can interfere with digital side. To avoid this phenomenon, the analog and digital supply lines (even associated to the same voltage level) as well as the ground reference are bifurcated as near as possible from the power source. With this measure, it is intended to improve circuit robustness and signal quality on the digital side and circumscribe high frequency noise to analog one.

### 3.3 PCB considerations

To validate the circuit designed, testing on the physical vacuum group, a PCB was built. This building process was guided by several constraints and assumptions that were made.

The first important condition is the size. The PCB must be allocated on the crate in the position that is currently filled by the PLC. Therefore, at maximum, the PLC dimensions can't be overpassed or, otherwise, the space available won't be enough to fit the acquisition board. The measures considered are  $270 \times 75 \times 100$  mm, distributed according with schematic 3.15.

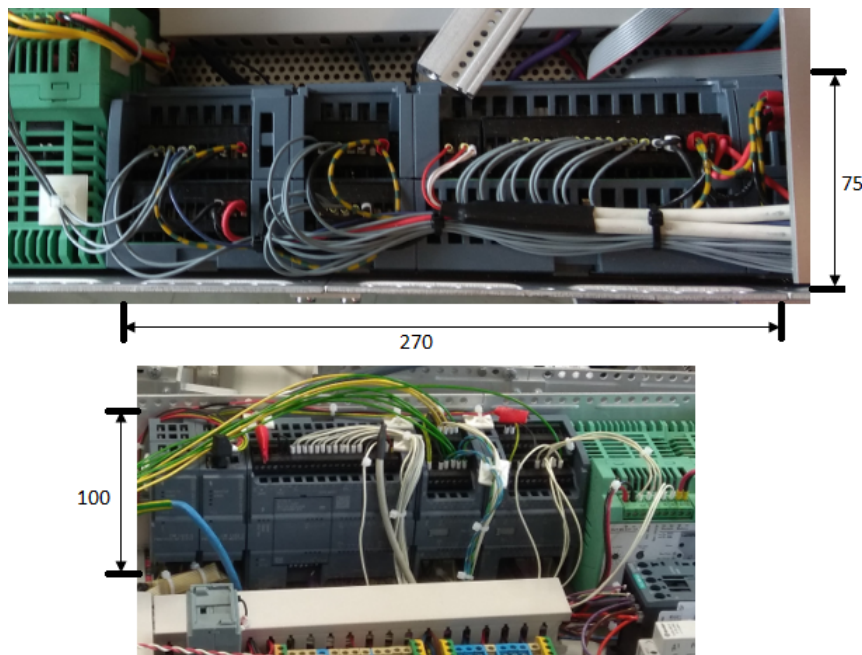


Figure 3.15: PLC upper and side view with respective dimensions, in mm.

This restriction limits the next steps to be taken. The first consequence resulting from it is the packaging type of components. Here, it is considered superficial mounting devices (SMD). Several benefits are obtained through this technique. For one side, the components associated to this construction technology have lower dimensions when compared to pin through hole (PTH) ones and, naturally, it is possible to save PCB area. On other hand, as the name suggest, these components are only mounted in one face of the board whereby both sides of PCB may be used to solder chips. As consequence of both previous facts, this method is characterized by a low devices density per area. This is a major advantage in applications dealing with high values of frequency, reducing the possibility of interference between devices. On this usage scenario, the signals acquired are relatively slow, but the serial communication bus implies a clock signal whose frequency has a value in order of MHz. So, it is recommended a significant trail distance to the other ones, what is directly related with devices concentration.

Still on the noise topic, another important consideration is the distinction between analog and digital sides. As already referred on [3.2.2](#), they need to be fed on analog and digital side, being the voltage level the same or not. However, even if the voltage level is the same, it is important to separate the supply trails in order to avoid distorting the digital data with the noise that is commonly inherent to analog signals. With that purpose in mind, the respective trails are separated on the PCB, diverging as near as possible to the main supply to minimize, at maximum, the signal interaction.

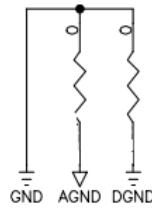


Figure 3.16: Analogical and digital grounds separation.

Other important issue is related with board connectors, namely their type and disposition. Including analog and digital signals, there is a total of 52 connections plus the power supply and grounds. To support such amount of links, a considerable number of connectors and, consequently, PCB area is required. According with current crate disposition, the PLC is allocated on a lateral DIN rail and the wires are connected on the upper side. So, the ideal disposition would be mount all the connectors longitudinally along all the upper side of the board. However, the length required is larger than the available. So, to mitigate this problem, they are still disposed longitudinally, only in one side of the board, but, instead of just one long row, there are several ones with a small number of connectors, organized in two groups on the left and right sides of the board. The connectors are placed in such a way that the connection is made by the upper side. Since they are placed in layers, the upper ones would block the access to the lower ones. This can be solved by using connectors with a certain degree of inclination and, instead of being straights, they are disposed in angle and don't overlap. Finally, regarding to the type of connection, once the board represents a prototype to be tested, the use of plug-in spring connections are preferable over the screw ones, because it allows a quick way to plug and unplug signals during the test phase.



Figure 3.17: Wires connectors [21].

Additionally, to achieve an integrated hardware solution, the core unit, represented by a *BeagleBone*, shall also be included on the board. The SBC interface is made through two female connectors with fifty pins each one. Here, are included serial pins, GPIOs and even power ones when it is intended to supply the board externally and not from the micro USB connector, as in this application. So, to assure a full support, two similar male connectors are soldered on the center of the board and, thus, the *BeagleBoard* can fit perfectly on the PCB.

Despite previous considerations, according with 3.1, the LCD display that assures the user interface requires a connection like the one here referred, the unit fits on the LCD back pins and no connections are let accessible, but not all the pins have a functionality associated. That's why,

to assure the compatibility and combination of both solutions, it is built a second circuit board, with a shape of a *BeagleBone* frame, and the required signals for LCD operation are grouped in one connector, disposed on a parallel fashion to the other ones, and a flat cable link the two boards.

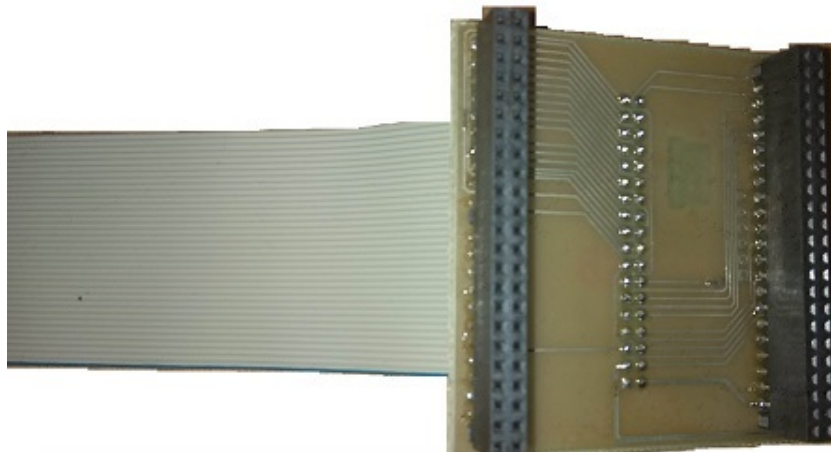


Figure 3.18: LCD board.

To prevent a possible incompatibility of such solution, the male connectors on the main board are extended to the back side by female ones and, in case of need, the whole card may be connected directly to the LCD. Of course, in a final solution this wouldn't be possible since the card must be fixed on the crate. But, on an experimental phase and as it is dealing with a prototype, this is a legitimate approach to avoid unexpected future issues.

Still on the road of unpredicted problems, an additional connector for a flat cable is included if, in the future, arises the necessity to use an external *microcontroller*. Here, are accessible the main signals, namely power supply, serial bus and diagnosis signals from chips.

Finally, to fix the PCB to the crate side, it is reserved on the board space for holes that will allow to screw a DIN rail connector. In total, there are available four fixing connectors, one in each extreme of the board and two on the center, in order to guarantee a good physical support.

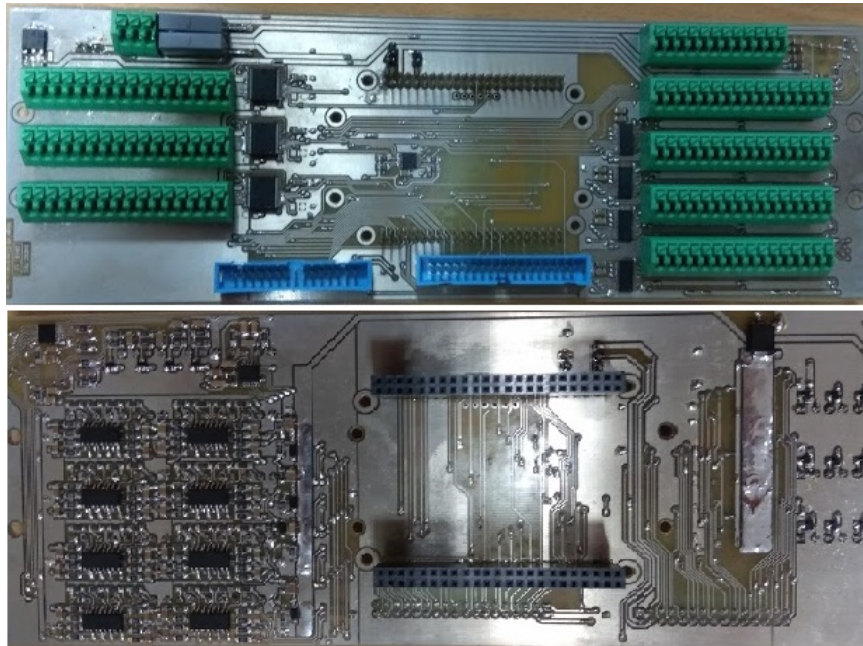


Figure 3.19: PCB board.

### 3.4 Serial interface

The communication protocol that supports chips-*BeagleBoard* link is based on a serial interface, being assured by a SPI bus. As indicated on previous topics, each device was selected according with that premise whereby all of them support such kind of interaction.

In order to highlight the most relevant aspects, SPI bus is characterized by four signals, being one the clock, other the chip select and two data lines. When exchanging data with a specific element, the respective *Select Slave* signal goes low, one of data lines carry the data transmitted by master to slave and the another one returns the response data that flow on the opposite direction. In applications where are several slaves, two approaches are susceptible to be applied. A select chip signal might be used per device and the remaining shared, operating independently. On other side, a daisy chain configuration is possible, being the bus signals, inclusive *Select Slave*, common to all devices. The slaves, however, are all connected on a ring topology, since the output of prior one is connected to the input of next one. So, the master, as response, receives a frame with the data returned by all slaves, disposed according with their position on the chain.

Here, on this application, there all ten integrated circuits that will act as slaves, being the *BeagleBone* the master station.

The SPI bus configured includes a select slave signal by default. If all the slaves would operate independently, it would be required ten signals of this kind. This would be possible since that any *BeagleBone* GPIO may be handled manually on software side and achieves the same behavior of that pins. But, once the chips were selected to be compatible with daisy chain mode and as

there are multiple applications of the same model, the devices may be grouped according with that criterion.

Based on previous ideas, there are three parallel daisy chains (ADS, *serializers* and output drivers) and one device that operates separated, the *thermocouple*. Eventually, all devices could be configured as on large chain. But, since the data associated, configured and returned, to each model diverges, this would imply a significant increase of complexity on data processing side and difficult the debug procedure in case of occurrence of any fault. To avoid that, it is preferable to join on the same chain only the similar devices. This implies the configuration of four *Select Slave* signals, one per group, while the clock and data ones are common to every chips.

Besides that, there are two additional considerations that deserve a careful analysis. For one side, since output data line, MISO, is shared, when a device is deactivated, the respective output must be in a high impedance state, otherwise it will cause a short circuit due the information provided by the chip current in use. Some devices, namely the *serializer* ones, don't go to tri-state when not in use. Then, this inconvenient must be overpassed. On other hand, the *BeagleBone* pins have a voltage range of 0-3.3V. Though, these voltage levels are not compatible with every ICs. Once again, the *serializers* have an internal logic operating between 0V and 5V, not compatible with the intended range.

In order to overcome the enumerated issues, it is used buffer gates. They are composed by sets of two inputs and one output. Briefly, one of the inputs is the data line that it is desired to put on the output side and the another one is a enable signal, blocking in one logic level, generally true, the transmission of the signal, and allowing, on the opposite scenario, commonly high level, the communication.

Since there are four sets, a quadruple bus buffer is required, being chosen the SN74AHC125 model. This IC is compliant with the rules presented. The output enable pin offers a negated logic operation, blocking the signal when at true voltage level. When a slave is enabled, its select line is at low logic level. That's why, these signals are used as the output enable signal, being the output of a chips set putted on high impedance, i.e., tri-state when not in use. Additionally, the buffer supports an input voltage until 7V and provides an output range according with the supply voltage. Then, feeding it with 3.3V, the output range is 0-3.3V independently of the input levels. It acts, then, like a voltage regulator. Finally, to avoid any unwanted short circuit during system start-up, the enable lines are pulled-up by 10K $\Omega$  resistors.

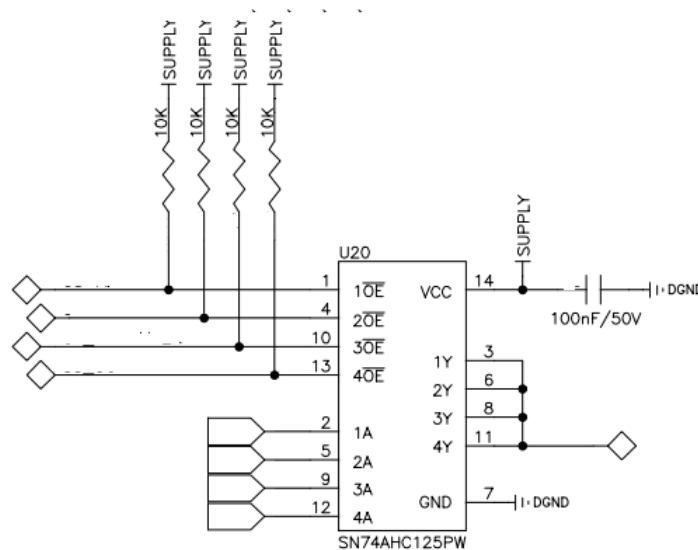


Figure 3.20: Serial bus buffer configuration.

### 3.5 Profibus interface

The *Profibus* support is an additional but relevant feature, since it provides a more complete data exchange with turbo pump and vacuum gauges. To be included on group acquisition, an electrical conditioning circuit is required to assure compatibility with this fieldbus connectivity.

*Profibus* physical layer relies on a serial communication, namely RS485 protocol. This is characterized for being associated to an electrical balanced transmission, i.e., there are two signals that transmit data flow, one of them carries the information on a non-inverting topology and the other one transmits the same bits but inverted. This configuration is used since the noise affects equally both signals and, measuring the voltage difference on the opposite pole, the effects eventually induced will be subtracted, remaining the initial load absence of any distortion.

The protocol requires a serial port to connect the device to the bus network. The *BeagleBone*, however, doesn't bring on its PCB an interface of this kind. One way to overpass this question, it is making use of USB port included. Through proper integrated circuits, it is possible to convert one type into another.

To not develop a solution of that kind from scratch, designing each individual component, there are also final solutions that can be used. A USB-RS485 converter is a device that supports, from one side, a connection with a RS485 serial interface and, on the other side, a USB support. They represent an option simply understandable, accessible and ready to use, not requiring any additional effort. On this project, it is used the adapter displayed on the picture 3.3.

The indicated device is composed, internally, by two main chips. It includes a MAX485 IC [22], that is a RS485 transceiver containing one driver and one receiver to serial communication, and a CH340 [70], which is a bus converter from USB to serial communication and vice-versa. Externally, it is accessible one USB connector and the two data lines of balanced serial interface. However, on serial side some premises must be considered before connecting the cable.

On adapter surface, it is possible to identify two labels that identify each line, 'A/D+' and 'B/D-'. On RS485 standard, it is defined that A represents the non-inverting line and B the inverting one [71]. But, several manufacturers adopt the opposite assumption and, that's why, this nomenclature is ambiguous, requiring always a confirmation on device *datasheet*. In this case, consulting the MAX485 specs [22], it is easily concluded that the A pin represents the non-inverting line and B the opposite.

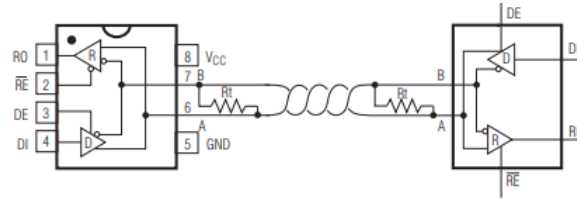


Figure 3.21: MAX485 internal configuration and typical application [22].

Additionally, RS485 interface implies other cares. As indicated, in each end side of transmission, it is measured the voltage difference, being the data classified in 0 and 1 logic level according with the value acquired. To not generate misinterpretations, this difference must be higher than 200mV or lower than -200mV. In a conduction state, this criterion is achieved, but, in some specific conditions, this might not be verified. For instance, if the line is in open circuit or the bus is in idle state (when stations are not “talking”), the value will be zero and an unpredictable behavior will be observable as well as the generation of spurious bits. One way to overcome that is pulling-up the non-inverting line and pulling-down the other one. This is only necessary in one point of the bus. The resistors values should be sized in order to guarantee the minimal threshold, assuming a standard value of  $390\Omega$ .

Besides that, the bus also has other particularity. When the signal is being transmitted on the cable, there is some impedance of the cable during its path. Once arrived the end, the resistance assumes an infinity level and the signal is sent back, due to an echo effect. The noise obtained is mixed with intended data, causing distortions and erroneous messages. To suppress this phenomenon, resistors must be putted on terminal points in order to avoid reflections [23]. Since RS485 serial bus is linear having just two end points, the resistors shall be located on each one of that points. The respective value should be similar to cable impedance. Normally, this parameter is proportional to its length. But, in *Profibus* usage, the cable utilized is marked for having a constant impedance regardless of length. That value is, approximately,  $150\Omega$ , being, although, used a resistor of  $220\Omega$  to assure an operation sufficiently far from voltage threshold limits [23].

In compliance with the exposed arguments, and despite of the adapter used includes an internal pull-up and pull-down of lines, the converter must be complemented with the circuit presented on picture 3.22.

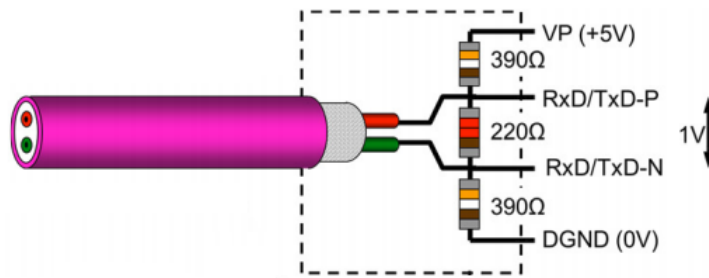


Figure 3.22: Termination resistors and failsafe circuit (adapted from [23]).

The used adapter is only for test purposes, since, on a final solution, it is required isolation between the slave stations and the master in order to assure a more robust and safe electrical operation.

### 3.5.1 Profibus connector driver

The *BeagleBone* installed operating system includes, by default, on its kernel, a driver that is compatible with the previous indicated USB-RS485 converter model. Once the device is connected, it is possible to confirm the proper driver load, through the terminal using the command line *dmesg*, being listed all load capes and drivers by a chronological order, or *lsmod*, presenting also a list of used drivers. Additionally, to identify the device name, it shall be executed the command *ls /dev/tty\**, being expected the device identification as *ttyUSB0*.

After the first tests with serial port, it was verified that the transmitted characters didn't match the intended settings. As stated on 2.3.6.2, each *Profibus* character is codified with 11 bits, eight of them related with data byte, one start bit, one parity bit and also a stop bit. To assure a transmission according with this codification, the serial port must be configured before start frames transmission. It was observed, however, that even when imposing those signal specs, the data sent didn't include a parity bit. After some research, it was discovered that the default driver doesn't support signal parity.

One way to solve the issue is to add the parity code patch to the original driver code. Since this fix was already released for other platforms, namely *Raspberry Pi*, it is only necessary to delimit the code fragment that was changed and perform the same action in this case. Consulting [72], it is marked the portion related to parity support:

---

```
if (C_PARENB(tty)) {
    if (C_PARODD(tty)) {
        if (tty->termios.c_cflag & CMSPAR) {
            dev_dbg(&port->dev, "parity = mark\n");
            par_flags = 0xeb;
        } else {
```

```

        dev_dbg(&port->dev, "parity = odd\n");
        par_flags = 0xcb;
    }
} else {
    if (tty->termios.c_cflag & CMSPAR) {
        dev_dbg(&port->dev, "parity = space\n");
        par_flags = 0xfb;
    } else {
        dev_dbg(&port->dev, "parity = even\n");
        par_flags = 0xdb;
    }
}
} else {
    dev_dbg(&port->dev, "parity = none\n");
    par_flags = 0xc3;
}
ch341_control_out(port->serial->dev, 0x9a, 0x2518, par_flags);

```

---

So, the driver compiled by default on OS, here 4.4 kernel version [73], was edited, added the code and finally this file was grouped with driver make file, downloaded from manufacturer site [74]. Then, it is necessary to re-install the driver. Though, this last step implies the compilation of whole operating system. To avoid that, the driver needs to be compiled apart and manually loaded. To compile singular system modules, it is necessary to download kernel headers, required to define system functions, otherwise the compilation process will fail. For that, the next steps must be executed:

```

root@beaglebone:~# sudo apt-get clean all
root@beaglebone:~# sudo apt-get update
root@beaglebone:~# sudo install linux-headers-$(uname -r)

```

After this, the driver is compiled through command *make*. If well succeed, it is ready to be loaded. It should be done before the adapter connection in order to overlap the default one, installed on kernel. This can be achieved executing the command *make load* inside of respective folder. Once finished it, the adapter can be connected and serial parity is now possible to be settled, including even and odd signals or no parity if not applicable.

### 3.6 Conclusions

On this chapter, it was presented the hardware architecture, detailing used devices and technologies as well as some additional considerations regarding to hardware robustness.

Firstly, the core unit role is played by a *BeagleBone Green*. The physical interface with group signals is made through an intermediate board, in order to assure compatibility with SBC pinout. The board is composed by several modules, each one related to a specific kind of electrical signal. The digital inputs are acquired by a chain of *serializers* that streams on a serial fashion the data acquired on parallel interface. On input side, they are isolated through *optocouplers* in order to avoid electrical interaction between group equipment. The analog signals are read through analogic-digital converter based chips, selected to operate on 0-10V range and another one to thermocouple. The data is also streamed on a serial way. On output, a chain of drivers, high side configuration, is used to send commands to VPGM. The interface of acquisition board with core unit is over a SPI bus, acting this last one as master and the board chips as slaves, communicating with each module at time.

Beside this I/O approach, *Profibus* interface is also considered. The hardware support behind it is a RS485-USB converter, being used for experimental purposes. The driver connector included by default on OS kernel doesn't support parity, not be compliant with *Profibus* requirements. So, the driver was updated with parity setting code, compiled separated and is ready to be loaded.

Also on hardware road, the interface with user through HMI is possible with an official 7" *BeagleBone* touch display. This communicates with the core unit over I2C, receiving image data and sending touch inputs, and projects the OS desktop.



## Chapter 4

# Software Architecture

On this chapter it is presented the software architecture behind the whole VPGM operation. On [4.1](#), it is identified the technologies chosen, the reason related to such choice and how they are integrated in order to suppress the intended requirements. Then, on [4.2](#), it is detailed the system process control, explaining the architecture underlying its development and relevant implementation details. The main screens associated to HMI are also showed as well as the ground rules that guide its utilization. After that, on topic [4.3](#), it is introduced the background software regulating SPI interface, being described on [4.4](#) how it is established the bridge between SPI and CODESYS runtime. Finally, on [4.5](#), it is detailed the *Profibus* stack architecture.

### 4.1 Software overview

Vacuum pumping groups are composed by several vacuum valves, a primary pump, a turbo-molecular pump, a bake-out and a vacuum gauge. The operation of each individual device must be combined with the other ones in order to achieve the expected group behavior. For that, a suitable software support is required.

In this case, all processing operation occurs on core unit, being the data exchange with field devices performed through the electrical board. Thus, on software side, there are two relevant parts. From one side, exists the acquisition program related to SPI interface and, on other hand, the pumping group control process.

The link ICs-*BeagleBone* is achieved through a SPI bus. This implies the host device, master station, configuration with the protocol specs related to each one of the slaves, namely SPI bus mode as well as data exchange size. In addition, some devices require a previous register configuration before being ready to use. These aspects are all combined on a program built on C language. This is the most natural choice for being in accordance with *libsoc* library and a native language of the operating system, not requiring additional compilers. The program must be deployed on a shell script and this executed at SO boot, in order to operate uninterruptedly (for further instructions see [B](#)).

On other side, the process development is supported on IEC 61131 standard. This solution is preferable for several reasons. Firstly, allows to maintain a solution equivalent to the one currently implemented, being easily understood by anyone used to the previous one. On other hand, preserves all the advantages of the standard (already enumerated on [2.3.2](#)), namely the user-friendly development environment, not requiring too much programming skills or knowledge to configure a controller device, and the integration in the industrial automation scenario, sharing the same programming paradigm that characterizes the devices applied in this field. Finally, and as consequence of previous premises, the code migration between PLC and this kind of architecture is almost immediate.

The process implementation is then performed on CODESYS runtime. Currently, it is one of the most stable and professional soft-PLC accessible, taking some leverage for offering solutions to this kind of core architecture. The software available is not totally free, being the demo version released timely limited to a consecutive range of operation until two hours. After that, it is required a device restart. On a final solution, this constraint is unacceptable and the license must be acquired. In this project, however, since its main goal is to provide a proof of concept, this not represents a significant issue.

The development of process on CODESYS must be performed on the desktop application installed on windows personal computer. Once developed, it is deployed on *BeagleBone* runtime, starting run as soon as the compilation step is finished. Before that, although, the runtime needs to be properly installed and configured, as discriminated on [\[61\]](#).

The CODESYS also includes an add-on related to visualization tasks. This allows the building of graphical interfaces inside of a standard program, including a library with diverse predefined visual objects to support it. Furthermore, it is possible to configure a web server and widespread the interface through a local area network, being accessible for any device connected to it. Such features led to its use to develop group HMI.

According with the previous selected technologies, it is presented the software architecture on picture [4.1](#).

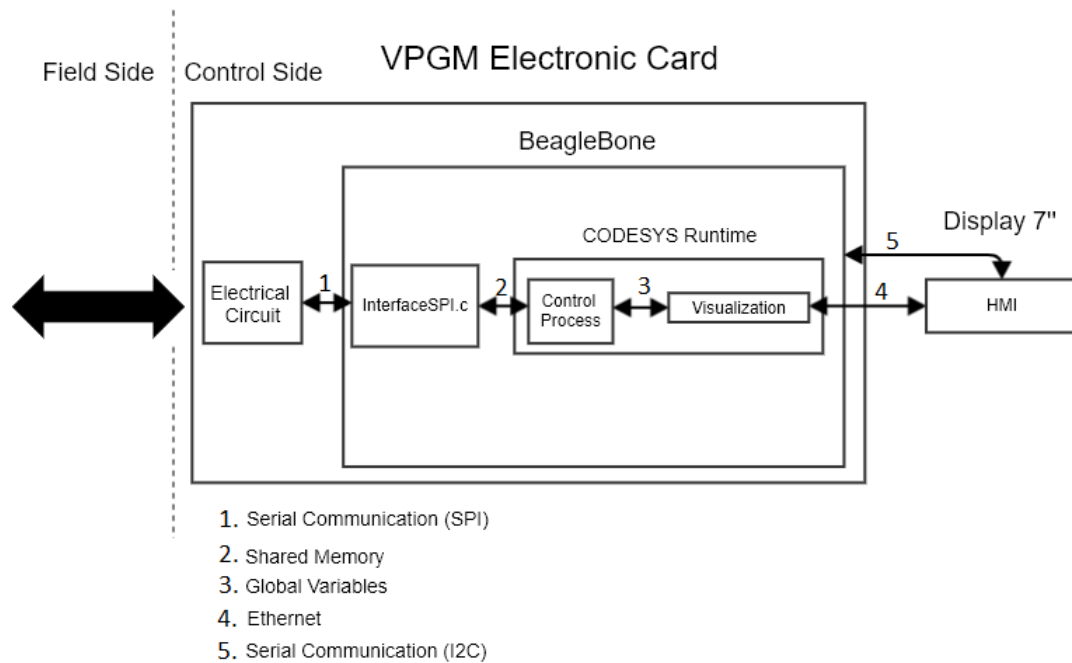


Figure 4.1: Software architecture.

On a brief analysis, and according with the aspects already referred on 3.1, the communication between electrical components and core unit is made through SPI. This implies a script running on background to set communication rules, sending and receiving data. Then, all the information needs to be exchanged with CODESYS runtime, that executes the control procedure. This task is based on POSIX shared memory API, associating a object descriptor to a memory area, being this accessible on both poles of communication.

Finally, the *BeagleBone* interaction with the LCD cape is made through I2C protocol. But, the access to HMI is achieved opening the browser and connecting to core unit local IP address, being then loaded the graphical interface.

## 4.2 Control process software

As said, the software architecture is according to the IEC 61131 standard. Then, the software development shall be compliant with the model proposed by the standard. It establishes that the software architecture relies on hierarchical elements, namely configurations, resources, tasks, programs, function blocks and functions. The articulation of all them allows the building of a control process.

In this case, the project implementation relies on only one resource supported by two main tasks. One of them is related to the effective process control, state machine, devices and group diagnosis, and the other one is linked to the graphical interface, allowing the user interaction on the process and showing, on the opposite direction, the group status and the flowing of process. The tasks have distinct levels of priority and this is determined according the time requirements

inherent to the action that each one performs. The communication and data exchange between them is achieved through global variables.

Regarding to global variables, they are grouped in four main structs. For one side, there is the inputs struct. This acts as the input process image and it is registered all group output signals, i.e, the feedback information sent by the devices, including valves, pumps, UPS and bake-out feedback signals, that represent the input data to the system control procedure. On other hand, the outputs struct represents, symbolically, the output process image, registering, at a specific moment, the last commands sent (or that will be send) to the physical equipment.

Additionally, there is also other struct related to the registration of the current state of control procedure and devices. Its purpose is to share the group data with graphical interface. Besides that, it also preserves several parameters that flag actuation orders sent by the user to manipulate process or equipment.

Finally, the last one is the configuration struct. Since the control process includes some parameters susceptible to be configured by user, namely some state timers, timeouts, enable/disable process auto restart and other actions, this registers the last configuration loaded or saved by the operator.

From now on, it will be described in detail each task on the process as well as the programs and function blocks that belongs to it.

#### **4.2.1 VPGM process**

The VPGM process task is the one with highest priority and supports the implementation of the entire process required to operate the vacuum group. Its architecture is intended to be as modular as possible and easily understood. Further, to achieve a standard solution that meets the industrial requirements imposed to PLC's operation, the process is compliant with the PLC scan cycle. It means that in every cycle the inputs are acquired, the processing is performed over that data array and, then, the outputs are set. It is mandatory following this principle in order to predict system behaviour as well as bound its temporal response. In this application, to aim a standard and competitive solution, the imposed rules are respected.

Additionally, regarding to modularity criterion, the implementation is supported on function blocks. Since the group is composed by several elements, some of them equal (valves, for instance, there are, at least, four of them, with identical behaviour), each type of device is implemented on a function block. Thus, in case of multiple presences, the code implementation is just done once, being defined several invocations of the same function block and providing the required input and output arguments. Not only this path saves programming effort, requiring less code and facilitating debug procedure in case of eventual malfunctions, as well as can be easily extrapolated for other applications or groups with the same devices, being only necessary to change the implementation related to group process.

In the next steps, it is explained each device functionalities and how the implementation achieves them.

#### 4.2.1.1 Valves

The valves are present on the group to control the air flow to and from the vacuum vessels, the conduction path between pumps and, also, additional ones might be included to leaking detection purposes. The number of valves diverges among groups but, at maximum, it is six: VVR1, VVR2, VVI, VVD, VVT, VVP. They are all equal and share the same principle of operation, based on compressed air. A valve disposes of one actuation and two feedback signals. The whole interface is digital 0-24V. Regarding to actuation, the signal is used to electrify a solenoid that will redirect the compressed air and force valve open, remaining, thus, closed when not actuated. The feedback signals are related, respectively, with opened and closed states. So, in a normal operation, they should be characterized, at each moment, for opposite logic levels. However, when this is not verified, the condition must be signalized, launching an error or warning flag. If both signals are at 24V, it is considered an error. When they are both off, it can be result of some electrical problem, whereby after 5s without communication a warning is generated.

Regarding to implementation, the respective function block logic is divided in two parts. Firstly, there is a diagnosis step where the feedback information is analyzed and set the valve state with the respective description, being the possible states {0, 1, 2, 3} and the description {open, closed, warning, error}, respectively. Before being classified as in warning condition, a timer, with a preset value of 5s, is started and only after its expiration the state change. The second portion is related to valve actuation. Here, the valve may be interlocked, controlled by process or operated manually, being the priority exactly on this order. There are two input variables for each one of these modes, one indicating that is on/off and the other one indicates the command, open/close, dictated by that mode to apply to valve. At a specific instant of the process, the three modes may be simultaneously applicable but just the one with higher priority will be considered. Despite this, an interlock may be forced and the valve manually manipulated by user, whereby a forced flag is included as input, being such hierarchy ignored when this is armed. At the output side, it is provided the command to send to the valve and a variable that preserves the current mode, being {0, 1, 2, 3} that corresponds to {manual, interlocked, forced, process}, respectively.

Some valves, however, may only be present in some groups. With that hypothesis on mind, an additional parameter is also considered on the input side and, in false scenarios, the respective FB logic is blocked, being the valve disconsidered from the process.

Additionally, when occurs a state change, i.e., switching off to on for example, the device takes some time to react and, in the meanwhile, the feedback information might be unpredictable and isn't reliable. To avoid the generation of any false alarm, after every switch, there is a blind gap of 2s, where the valve diagnosis is still performed but not considered for error purposes. Two timers are implemented to measure how long a valve is on the current state, being cleared at each change.

According with the information exposed, the function block interface can be symbolical represented by the picture [4.2](#).

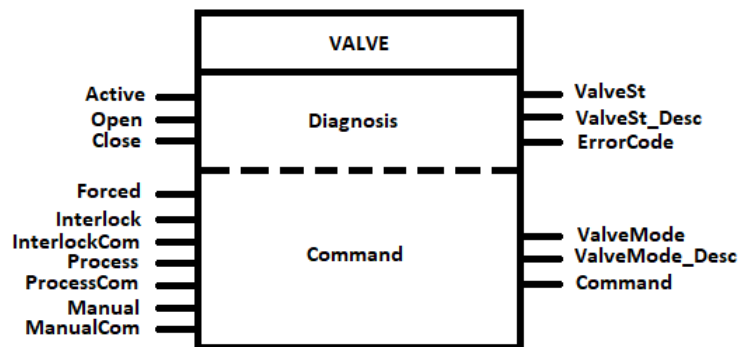


Figure 4.2: Valve FB.

Except the input signals provided by the device and output command, the remaining parameters, since they preserve the current state and mode of the device, they are shared with HMI task.

#### 4.2.1.2 Turbo-molecular pump

The Turbo pump contacts directly with the vacuum vessel and is responsible for the achievement of the lowest levels of pressure. However, the pump cannot be directly interfaced by the card. It disposes of a controller that assures such communication and only this may interact with the card. This link may be supported on *Profibus* or direct I/O communication. In the first scenario, a complete diagnosis is possible, allowing the verification of a large amount of error conditions. But, without protocol stack, the only one remaining is the I/O option. Here, few feedback signals are accessible, namely on/off, standby, nominal speed and pump condition. All of them are digital signals and, due to the poor information that provide, only allow a basic diagnosis. The last one flags any abnormal condition associated to the pump but not discriminate which, making difficult the debug procedure.

On the implementation side, the same approach as the one described for the valves is followed. Equally, there is a portion of code related with diagnosis and another one to set the output command. Here, the diagnosis signals are not mutually exclusive and all the states may coexist at the same time, i.e., the pump might be, for instance, turned on, on standby mode and even in an abnormal condition. That's why, there is one output variable to flag each one of these conditions, being, then, processed on next verification steps during the process execution as well as transmit the device condition to graphical interface. This is achieved by just exporting the incoming feedback signals, without the need of any processing inside of FB. The output commands may be ordered by the process or manually by user when the first one is not applied. On this application, it is not foreseen interlock conditions for turbo pump operation. Therefore, there are four input parameters, two for the process, indicating one if the pump is controlled by the process or not and the other one the state forced by it. On the manual context, it is similar. The process overlaps any order sent by user when applicable. So, the command that prevails is sent to the pump as actuation order.

When switching between off and on mode, or the opposite, the feedback signals might take a while to be updated and reflect the real state of the pump, like described on valves. To not generate false errors, again one timer per state is implemented and just after the respective timer expires a group error condition is considered. The timers are configured with a preset value of 5s, being an enough value to assure a proper operation. The timers are reset at each state switch.

The pump also offers other possibilities of actuation besides on/off command. It is allowed to set the standby mode, being the pump rotation limited to a customizable percentage of the nominal speed (66% by default), and to reset the pump, erasing error messages. Both of signals are not controlled by process and the user can manipulate them at any time. The first one is a static signal and can be activated and deactivated as intended. The second is a pulse signal and so, when activated for the first time, a timer of 1s is started and, once elapsed that time, the command is turned off.

Hereupon, the FB interface looks like represented on picture 4.3.

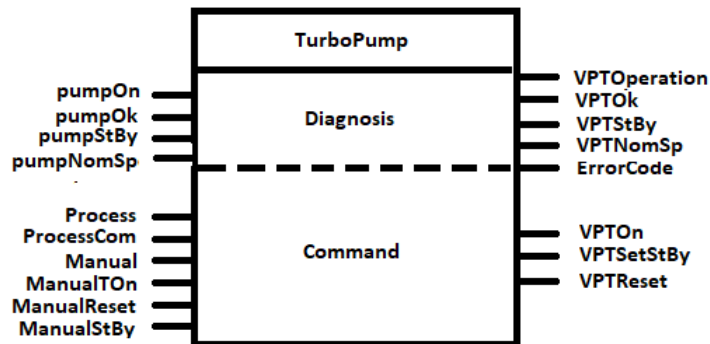


Figure 4.3: Turbo-molecular pump FB.

#### 4.2.1.3 Primary Pump

The turbo pump, in one of the sides, contacts directly with the vacuum vessel. The other one can't be left at atmospheric pressure, otherwise the lowering pressure procedure won't be effective and the ultra-high vacuum won't be achieved. To suppress that, a back pump between turbo one and atmospheric environment is placed to ensure a proper pressure on the turbo outer vessel. A primary pump plays such role.

The interface with the primary pump is I/O based. As feedback signals, a large amount of connections is accessible. It is not only reported the present state, the cable connection status, thermal protection as well as recognized a phase error, when on three-phase mode. Furthermore, there is an analog signal informing the current consumption at a given moment. On the opposite direction, the pump may be turned on/off and the thermal protection reset, actuating on a relay during a brief time period.

Once again, the implementation aspects are based on the premises defined for the previous elements. The function block receives the feedback signals provided by pump as input arguments

and they are then processed. The higher number of connections compared with the turbo pump facilitates a more exhaustive diagnostic, being verified the existence of several warnings and errors. The result is exported through several parameters, indicating on/off state (*VPPOperation*), if the pump is in valid, warning or error (*VPPState*) with the respective description (*VPPStateDesc*), and, if applicable, the error code is saved (*errorCode*) and the details related to it (*errorDesc*). All data generated is saved and showed on the graphical interface. The errors related to pump may occur simultaneously but just one of them is registered. Thus, the verification process respects a hierarchy which is represented on table 4.1.

Table 4.1: Primary pump faults hierarchy

| Primary Pump Faults |      |                                    |            |
|---------------------|------|------------------------------------|------------|
| Type                | Code | Description                        |            |
| Error               | 129  | Cable disconnected                 | l          |
|                     | 127  | No current in pump                 | l          |
|                     | 130  | Thermal relay tripped              | l Lower    |
|                     | 131  | Phase error (only on 3 phase mode) | l Priority |
|                     | 128  | Current critically high in pump    | l          |
| Warning             | 103  | High current                       | l          |
|                     | 102  | Low current                        | v          |

The actuation order is dictated by the process or according with operator instructions. The first one overlaps the second and is signalized through two variables that indicate order status and the command to set (*Process* and *ProcessCommand*, respectively). When not applied, the manual command is privileged and registered on pump actuation command (*VPPTOn*). The reset of thermal protection is free of any process order and just controlled by human interaction. The signal sent is a pulse with the duration of 1s, triggered by the reset order (*ManualReset*) and cleared after that interval.

In the same way as already exposed, timers for on/off operation mode are included to cause a delay on group generation error after commute the pump. The preset value is 5s.

The FB appearance is exposed on picture 4.4.

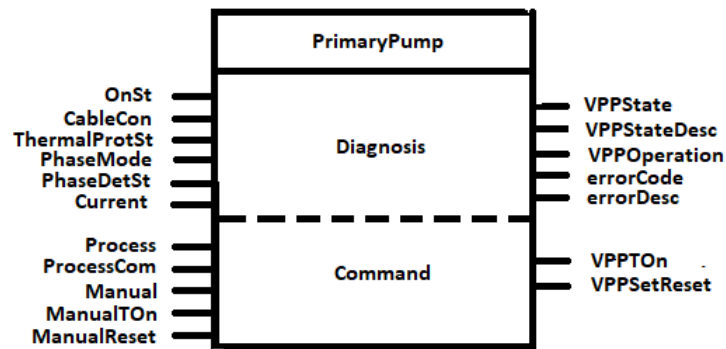


Figure 4.4: Primary pump FB.

#### 4.2.1.4 UPS

The group has an UPS to assure, temporarily, the maintenance of group operation in case of power failure. The device interface enables three feedback signals related with the current state of operation, state of charge and also an alarm flag. No action is needed to be executed on the equipment, since, in case of a power cut, the UPS is automatically turned on.

On the implementation side and regarding to process, the only relevant diagnosis parameter is the one that indicates the on/off state (*UPSSSt*). When in on state, an error should be generated to notify the group supply by UPS. Based on this, the FB, despite of receiving as input arguments the three feedback signals, only the state is verified and, if true, the output state parameter (*State*) is equally set to true and the error code indicated (*errorCode*). After that, no more verification steps are required.

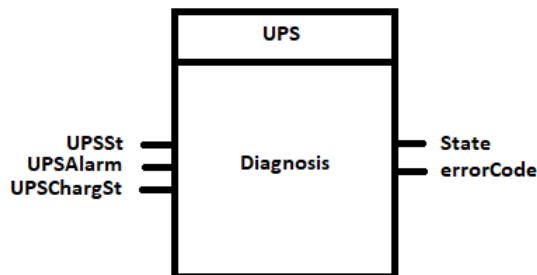


Figure 4.5: UPS FB.

#### 4.2.1.5 Vacuum gauge

The vacuum gauges present on the group allows to monitor the pressure of the system being pumped. Two kinds of active pressure gauges are considered, namely *Pirani* and *Penning* ones. These, although of diverging on principle of operation, offer a similar interface, that can be of two types. The simplest is I/O connections. Here, it is possible to turn on/off the gauge and read the vessel pressure through an analogous signal. Another way is through *Profibus* network, providing

a large amount of data. On this case, since protocol stack isn't integrated, it is just considered the first approach.

The function block associated to the device only includes one input parameter, pressure read, and the activation command as output. Additionally, the pressure is shared with HMI whereby the respective value is saved on internal process variables.

The implementation is fairly simple. But, this modular approach relied on FB leaves the implementation opened for futures additions, namely the inclusion of the *Profibus* stack, being just necessary to add the intended verification steps inside and, then, proliferate the code through several calls according with the number of gauges on the group.

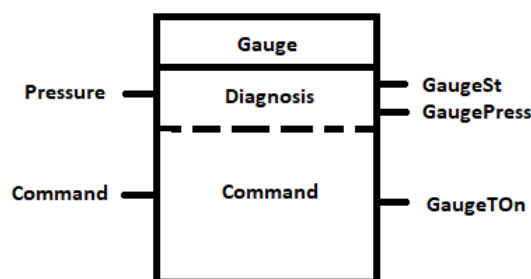


Figure 4.6: Vacuum gauge FB.

#### 4.2.1.6 Bake-out

In some cases, the presence of a bake-out system may be required to heat the vacuum vessel and reduce the concentration of certain impurities. Although not always this kind of devices is included on the group, the crate must assure the support for it. The interface follows the principle described for the gauge, except that only I/O signals are available. Then the system might be turned on/off according with user order, controlling a relay state for that purpose, and the feedback is a temperature value, provided by a *thermocouple* sensor.

The FB implementation is characterized by one input signal and an output action. The temperature is also shared with HMI.

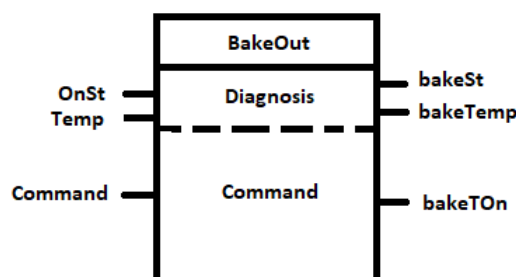


Figure 4.7: Bake-out FB.

### 4.2.1.7 State machine

The group behavior is characterized by several modes of operation, existing several steps inside each mode and being the field devices in specific conditions in each one. In each mode, a certain task is performed and transition between modes shall be solicited by operator. Furthermore, some of them as well as transitions might not be satisfied if the user doesn't have permission or if it can put the system on a dangerous condition, respectively.

On a functional perspective, the main procedure that must be executed on the vessel is the pumping of the pipelines. Taking advantage of primary and turbo pumps, the pressure is lowered until UHV level and maintained. Either the reverse behavior is considered, i.e., a controlled venting avoiding the contamination of internal surface of vessels. The venting may mobilize both pumps, venting all, or just to the turbo-molecular one, venting turbo. Furthermore, it is also possible to verify the vacuum isolation through the leak or large detection modes. The first one is applied only to the turbo-molecular side while on the second the intermediate valve is open and the whole system is tested. Finally, there are two additional modes, the stop mode, being all valves closed and pumps turned off, and the off mode, where the devices are not controlled by process but manually by the user.

The state machine that modulates the control process may be described by picture 4.8.

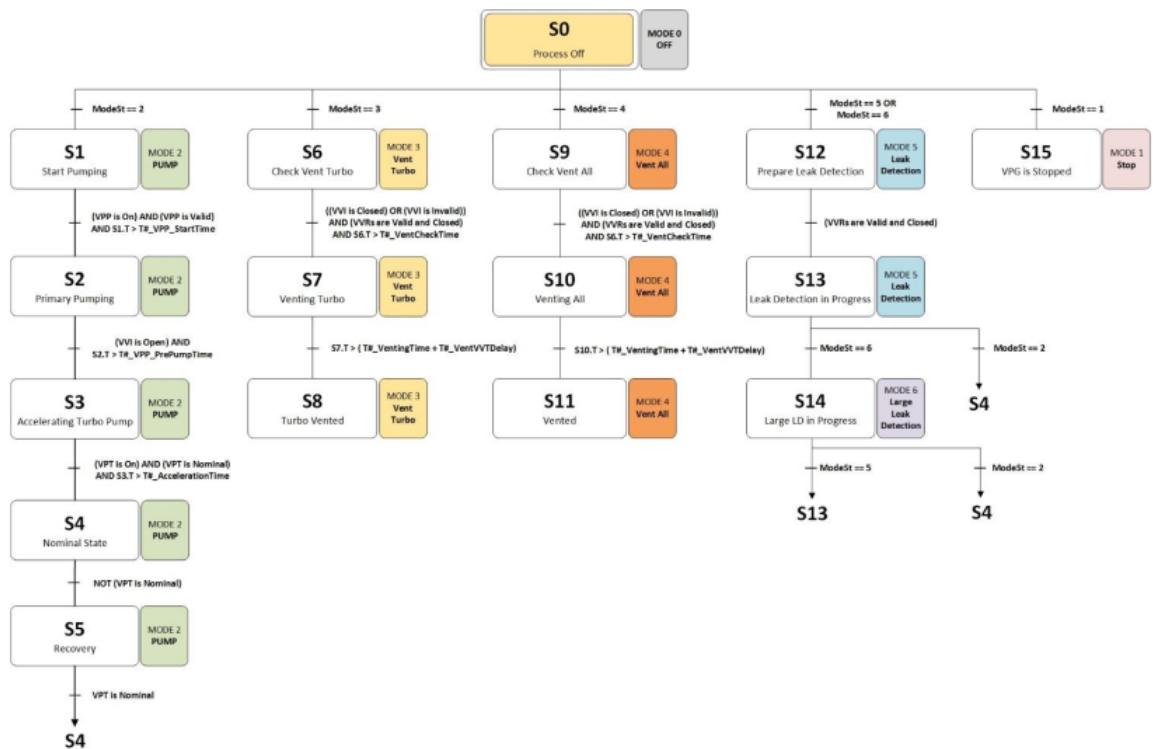


Figure 4.8: Group state machine [24].

As exposed, there are five parallel branches, being each one related to a certain mode. When a mode change is requested, what can happen in any step inside that mode, the condition required is verified and, if valid, the transition occurs to off mode and, then, to the intended branch. From this

behavior are excluded the transitions to large leak detection and from leak and large leak detection. To enable the large mode the process must be on leak detection since these two are executed on a sequential fashion. That's why, the transition is direct, not requiring reset the process to off mode. On the other side, when leaving the main steps of these modes, the system should return to nominal state of pumping mode without executing its previous steps whereby the transition is also performed directly, not passing by off mode.

The transitions permitted between modes are represented on picture 4.9

| Current Mode |                      | Requested Mode |      |                                                    |            |          |                                   |                      |
|--------------|----------------------|----------------|------|----------------------------------------------------|------------|----------|-----------------------------------|----------------------|
|              |                      | 0              | 1    | 2                                                  | 3          | 4        | 5                                 | 6                    |
|              |                      | Off            | Stop | Pump                                               | Vent Turbo | Vent All | Leak Detection                    | Large Leak Detection |
| 0            | Off                  |                | Yes  | Yes                                                | Yes        | Yes      | No                                | No                   |
| 1            | Stop                 | Yes            |      | Yes                                                | Yes        | Yes      | No                                | No                   |
| 2            | Pump                 | Yes            | Yes  |                                                    | Yes        | Yes      | Yes<br>(only from states 3, 4, 5) | No                   |
| 3            | Vent Turbo           | Yes            | Yes  | Yes                                                |            | Yes      | No                                | No                   |
| 4            | Vent All             | Yes            | Yes  | Yes                                                | Yes        |          | No                                | No                   |
| 5            | Leak Detection       | Yes            | Yes  | Yes<br>(only from state 13,<br>go back to state 4) | No         | No       |                                   | Yes                  |
| 6            | Large Leak Detection | Yes            | Yes  | Yes<br>(only from state 14,<br>go back to state 4) | No         | No       | Yes                               |                      |

Figure 4.9: Transitions between modes [24].

Besides these permissions, additional conditions must be considered when validating a transition. For one side, leak detection and Large Leak Detection modes are only allowed if valve VVD exists, fact that might not be observed in some group configurations, and if VVR valves are in valid and closed state, being also a mandatory condition to enter on Vent Turbo and Vent All modes. Regarding to Vent Turbo, it is also required the presence of VVT valve, optional in some cases. Finally, setting any other mode when the system is on pumping one implies that the VVR valves are in valid state in order to prevent an accidental venting.

In each step of the state machine, the field devices might or not be controlled by the process. On the first case, the operation state is dictated by the process, according with picture 4.10 correspondence.

|      | Mode 0<br>Off | Mode 2<br>Pump |     |     |     |        | Mode 3<br>Vent Turbo |      |     | Mode 4<br>Vent All |       |     | Mode 5 + 6<br>Leak Detection<br>Leak Detection<br>Large |     |     | Mode 1<br>Stop |
|------|---------------|----------------|-----|-----|-----|--------|----------------------|------|-----|--------------------|-------|-----|---------------------------------------------------------|-----|-----|----------------|
| Step | 0             | 1              | 2   | 3   | 4   | 5      | 6                    | 7    | 8   | 9                  | 10    | 11  | 12                                                      | 13  | 14  | 15             |
| VPP  | Man           | On             | On  | On  | On  | On     | On                   | On   | On  | Off                | Off   | Off | On                                                      | On  | On  | Off            |
| VPT  | Man           | Off            | Off | On  | On  | On     | Off                  | Off  | Off | Off                | Off   | Off | On                                                      | On  | On  | Off            |
| VVI  | Man           | Off            | On  | On  | On  | On     | Off                  | Off  | Off | Off                | Off   | Off | Off                                                     | Off | On  | Off            |
| VVP  | Man           | Off            | Off | Off | Off | Off    | Off                  | Off  | Off | Off                | On*   | Off | Off                                                     | Off | Off | Off            |
| VVD  | Man           | Off            | Off | Off | Off | Off    | Off                  | Off  | Off | Off                | Off   | Off | Off                                                     | On  | On  | Off            |
| VVT  | Man           | Off            | Off | Off | Off | Off    | Off                  | On** | Off | Off                | On*** | Off | Off                                                     | Off | Off | Off            |
| VVR1 | ITL           | ITL            | ITL | ITL | Man | St ITL | ITL                  | ITL  | ITL | ITL                | ITL   | ITL | ITL                                                     | Man | Man | ITL            |
| VVR2 | ITL           | ITL            | ITL | ITL | Man | St ITL | ITL                  | ITL  | ITL | ITL                | ITL   | ITL | ITL                                                     | Man | Man | ITL            |
| VG   | Man           | Man            | Man | Man | Man | Man    | Man                  | Man  | Man | Man                | Man   | Man | Man                                                     | Man | Man | Man            |

Figure 4.10: Devices condition on each step [24].

#### 4.2.1.8 Interlocks

Referring to VVR valves, these are not controlled by the process at any moment. The valves are interlocked by conditions external to the state machine, or let manual, being in such case sensible to operator orders. In some cases, the interlocks might be also extended to other valves. The interlock typically forces the valve closed state. There are, however, some conditions where valve is locked and remains on the state verified on interlock occurrence. The interlock conditions, separated by valves, are the following ones:

- **VVR1/VVR2**

**Interlock:** NOT (Step == 4 OR Step == 5 OR Step == 13 OR Step == 14) OR (VPG in Error State) OR (UPS in Battery Mode)

**Start interlock:** Step == 5

- **VVD**

**Interlock:** NOT ((VVR1 in Valid State) AND (NOT (VVR2 Exists) OR (VVR2 in Valid State)))

- **VVP/VVT**

**Interlock:** NOT ((VVR1 in Valid State) AND (VVR1 is Closed)) AND (NOT (VVR2 Exists) OR ((VVR2 in Valid State) AND (VVR2 is Closed)))

When active, an interlock may be forced by the user and, in such condition, the device is manually operated. This scenario, however, is only possible during a limited time, customizable, and, after the timeout, the valve is again interlocked. Besides that, when entering in some particular steps and modes, namely off, stop, vent and vent all modes or nominal step of pumping mode, the forced mode must be cleared and the interlock reset.

#### 4.2.1.9 Group errors

The individual devices may be in valid, warning or error states, like explained in the previous description of each one. In some cases of these last two situations, a group error is generated and, consequently, the VPGM operation is compromised. These errors are not just inherent to devices as well as may result from unwanted behaviors related with state machine. When they are observed, it is necessary to launch a handle action with the purpose to mitigate such condition or, at least, put the system on a safe status.

The procedure executed when an abnormal condition is detected consists in change the system process to a specific mode. This depends from several factors. For one side, it is conditioned by the error detected and, on the other hand, it also depends from auto-vent permission, customizable by the operator. If it is allowed, the system will switch to Vent All or Vent Turbo mode. If not, the Stop mode will be set. Once the error disappears, two paths are available. If the auto-restart is permitted and the actual number of restart times is lower than the limit imposed, the process returns to Pump mode, if it was previous there at moment of error occurrence, and to Stop mode on other cases. If auto-restart is forbidden or the limit is expired, the process remains on the recovery mode until new order.

On picture 4.11, it is described the group errors as well as the reaction expected from the system, depending from parameters configuration.

| Error No. | Description                                                        | Action (AutoVent = TRUE) | Action (AutoVent = FALSE) | Reset                                    |                     |
|-----------|--------------------------------------------------------------------|--------------------------|---------------------------|------------------------------------------|---------------------|
| 113       | Power Cut (running on UPS)                                         | VENT_ALL                 | VPG_STOP                  | When condition disappears                | Lower priority<br>↓ |
| 118       | VVR Error status                                                   | Nothing, ITL Valves      | Nothing, ITL Valves       | When condition disappears                |                     |
| 110       | VPT Error Status                                                   | VENT_TURBO               | VPG_STOP                  | When condition disappears                |                     |
| 114       | VPP Error Status                                                   | VENT_ALL                 | VPG_STOP                  | When condition disappears                |                     |
| 112       | VVI Error or Warning Status                                        | VENT_TURBO               | VPG_STOP                  | When condition disappears                |                     |
| 115       | VVP Error or Warning Status                                        | VENT_ALL                 | VPG_STOP                  | When condition disappears                |                     |
| 116       | VVT Error or Warning Status                                        | VENT_TURBO               | VPG_STOP                  | When condition disappears                |                     |
| 117       | VVD Error or Warning Status                                        | VENT_ALL                 | VPG_STOP                  | When condition disappears                |                     |
| 105       | VPP NOT pumping                                                    | VENT_ALL                 | VPG_STOP                  | By Auto-Restart or when VPG is restarted |                     |
| 109       | Accelerating (VPT) TimeOut [T#_ErrAccelTO]                         | VENT_TURBO               | VPG_STOP                  | By Auto-Restart or when VPG is restarted |                     |
| 108       | VPT not On Status after [T#_ErrVPTNonTO] in Accelerating Step      | VENT_TURBO               | VPG_STOP                  | By Auto-Restart or when VPG is restarted |                     |
| 103       | Recover TimeOut [T#_ErrRecoverTO]                                  | VENT_ALL                 | VPG_STOP                  | By Auto-Restart or when VPG is restarted |                     |
| 107       | VPT not On Status in Nominal or Recovering and Leak Detection Step | VENT_ALL                 | VPG_STOP                  | By Auto-Restart or when VPG is restarted |                     |
| 106       | VVT or VVP or VVD not closed in Nominal or Recovering Step         | VENT_ALL                 | VPG_STOP                  | By Auto-Restart or when VPG is restarted |                     |
| 104       | VVI not opened in PrimaryP, Accel, Nominal or Recovering Step      | VENT_ALL                 | VPG_STOP                  | By Auto-Restart or when VPG is restarted |                     |

Figure 4.11: Group errors hierarchy (adapted from [24]).

Several errors might occur at the same time. In that case, however, there is no need to signalize all of them and a hierarchy should be considered. The picture is ordered according with that, considering a successive decreasing priority.

#### 4.2.1.10 Implementation details

The implementation of the state machine is intended to be intuitive, following the diagram structure that facilitates the process debug and understanding by project outside elements, and matching once again a modular approach. To fulfill such requirements, the implementation relies on a function block supported by sequential function chart (SFC) programming language.

In order to approximate as much as possible the theoretical design from the practical implementation, the codification set on the diagram is maintained, meaning that the step's numbers match, and the actions and transitions are implemented on ST. Regarding to transitions, they are named according with the two steps between which they are located,  $TX\_Y$ , being  $X$  the number of previous step and  $Y$  of the next one. Therefore, for instance, the transition between  $S2$  and  $S3$  is codified as  $T2\_3$ . The actions included are of three types: entry, exit and continuous actions. The first ones are executed as soon as the step is activated and just once. Their nomenclature is  $SX\_setSt$ . The continuous actions are executed as long as the step is active and are associated to a  $N$  qualifier, as predicted on the IEC 61131 standard. In this case, they are named  $SX\_act$ . An exit action is executed only once before the step is deactivated and is labeled  $SX\_exit$ . In the previous references,  $X$  represents the step's number.

On the entry actions, the main purpose is to update the process variables according with the current step and set the devices that are controlled by process and the ones that aren't. So, firstly, the state timer is restarted, the process mode is updated to the one that belongs the step and the step number is registered. Furthermore, it is, thus, established the elements controlled by process. This is achieved configuring the *Process* variable inherent to each valve and pump. According with these premises, it is showed the entry action of step  $S1$ :

---

```

GLOBAL_VAR.Utills.stepStMachine := 1;
GLOBAL_VAR.Utills.modeST_curr := 2;
GLOBAL_VAR.Utills.StateTime(IN:=TRUE);
GLOBAL_VAR.Utills.skipStep := FALSE;
GLOBAL_VAR.Utills.vvi_process := TRUE;
GLOBAL_VAR.Utills.vvd_process := TRUE;
GLOBAL_VAR.Utills.vvp_process := TRUE;
GLOBAL_VAR.Utills.vvt_process := TRUE;
GLOBAL_VAR.Utills.vvr1_process := FALSE;
GLOBAL_VAR.Utills.vvr2_process := FALSE;
GLOBAL_VAR.Utills.tmp_process := TRUE;
GLOBAL_VAR.Utills.vpp_process := TRUE;

```

---

Besides the parameters referred, it is possible to notice the settling of a skip-step parameter. This is associated to a user order, accessible on HMI, to skip a step even if the transition condition is still not verified. Since the user might have forced the previous transition, it is recommended

to clear the variable every step change. In addition, regarding to mode codification, its number is according with the one present on figure 4.8.

The continuous action set the state to be applied to the devices controlled by process. The *ProcessCommand* parameter of each component is configured according with the intended action to be applied on it.

Finally, the exit action is only necessary to reset the state timer.

Hereupon, the final graphical coding is almost a copy of the theoretical process design. The main difference is related with an additional step included on an extra parallel branch, *S16*. The intention behind that is to modulate the situation where the process should not be considered, i.e., when the VPGM is on and in operation, the normal modes, from 0 to 6, are applicable; however, if the VPGM is off, the user may still want to control the equipment, but without any restrictions and group errors. So, this step has an implementation similar to Off mode, being the whole group on manual mode, with the difference that, here, there is no supervision over devices and the process is indeed off.

According with the previous details, when the execution starts the process is off. As soon the user turned it on, the Stop mode is configured and the system stays there until new order. Once that happens and if allowed, the state machine is reset, *S0*, and the process flows to the branch corresponding to requested mode. This is the logic inherent to the entire process operation. The last mode ordered by user is registered as well as the current one. The first one is used in transition state verification to evaluate if any request was ordered. But, if the change is not allowed, the two records will diverge and, in a future moment where the previous order mode might be permitted, since the order stayed registered, the system will indeed transit. However, this is not the pretended behavior, because at that moment, the previous order might be outdated. That's why, a validation timer is implemented, meaning that when the user set a change if it is allowed the two values won't diverge, but if after 1s they are different, the current mode is maintained.

Finally, the user is permitted to turn off the process operation in any step, being set on such situation the *S16* state.

The valves interlock conditions are independent from state machine and, for that reason, are implemented outside of its function block. Due to be the most intuitive and understandable way to evaluate the conditions, the programming code relies on ST language. The verification steps are simply done migrating the pseudo-logical description to ST format almost with no change, just matching the parameters referred with the variables implemented. There is, however, one aspect that deserves a mention. On interlock cases, the valves must remain closed. So, to the interlock valve flag is attributed a high logic value and, on the interlock command, a close action is ordered, low logic value. These two parameters belong to input arguments of valve FB and will overlap the process or manual commands. When the device is start interlocked, its state is analyzed and the interlock command will be equal to that. The fragment code is, then, included on the process main program. Since it is related to specific VPGM process aspects and doesn't correspond to a module that requires several invocations, it doesn't make sense create its own FB.

Despite the aspects considered, another important issue is the forced mode. This is required through HMI. In such scenario and if the device is interlocked, a boolean flag is putted on high logic level and a timer is started. After, at each program cycle, it is confirmed the elapsed time and, when expired, the flag is cleared. This variable is one of the input arguments of valves FB and, then, the internal implemented logic assures that, when high, the interlock is ignored and the user command prevails.

Referring to group errors, several aspects should be highlighted. The verification procedure is made through a chain of *if...else* statements with the order indicated on picture 4.11. Regarding to errors related with conditions inherent to a specific state machine's step, the verification is done inside its continuous action and, when detected, a flag raises. Then, at the diagnosis moment, it is enough to analyze that flag. When an error is detected, a parameter is set, the error code is registered and the respective description either. Additionally, if it implies the system restart, another boolean variable is used to inform that need. After that, it is determined the mode to which state machine shall transit, supporting that verification on the premises previously indicated.

Besides from what was said, when a command is sent to a device and the operation mode required is set, the feedback signals won't reflect immediately such state, existing some delay. In the meanwhile, false error events would be generated, interrupting the system operation. To avoid that, inside of the FB of each device, there are timers which registers how long the device is on that state (on or off). Despite of being informed to the user that the device is indeed in error, the group alarm only will be generated if the situation persists when the referred timers expire. To abnormal conditions associated to the state machine, a similar approach is implemented. Here, it is registered how long that step is active and only after 2s after the entree the errors are signalized.

#### 4.2.1.11 VPGM

All the previous defined function blocks and ST code fragments are combined on the process main program. A program is a composition of FB and is integrated on an embracing element, a task. Here are defined all the logic aspects strictly related to this kind of mobile groups and declared all the FB needed to represent the physical devices that are included on the system hardware. So, for instance, if it is necessary to design a process control procedure to another group with a similar equipment, all the FB implementation can be reused, since the devices are equal, and only this program must be adjusted to the new control rules. This approach promotes a modular structure and facilitates the code reutilization and its extrapolation for other applications, not limiting it to this kind of groups.

On this particular application are declared six valves, one primary pump, one turbo pump, one gauge, one UPS and, finally, one bake-out. The valves, however, might not be all present and, thus, the configuration should be chosen on HMI and, if not applicable, the active signal associated to a valve is reset.

This module implementation has the purpose to follow the IEC 61131 standard guidelines. So, this is done in such a way that the PLC scan cycle is respected. First, the physical inputs are

acquired at same time to process input image. All the subsequent processing activity is applied over that input set.

Immediately after the acquisition moment, it happens the devices diagnosis. The function blocks representing each one of them are invoked providing only the arguments required to diagnosis purposes. Once verified the equipment condition, it is executed the state machine starting procedure. This code fragment satisfies two needs. For one side, it is used to start the process when is off and the user requires such action. On that situation, a flag is abled true, through the HMI, and the start procedure runs. On first order, the group change from off condition, *S16*, to Stop mode, *S15*. This a safety measure to safeguard any accidental or no intended order. To enter on Pump mode, it is required another start command by user. The second need reflected on this procedure is the restart procedure that should be done when the group enter on an error state. If the group may return to normal operation after the condition disappears, the procedure clean all errors and put the group on Stop mode or Pump mode, if it was there when the alarm was launched. If the error implies the group restart, two ways may be followed. If the automatic restart parameter is configured true and the number of restart times is less than the allowed, the same procedure applied for the last case will be followed. On other scenarios, the group restart is not allowed and it will stay on the handle action mode until new order.

Once done the (re)start procedure (or bypassed if not applicable), the state machine is executed.

Passed the state machine execution, it is time for group errors diagnosis. This step is only executed at this moment because some errors are related with the process operation and not just with the devices. One important aspect is that some errors may occur and not be detected by user if the process restart autonomously. If the error is launched and mitigated while the operator is not near to the system, he won't understand the reason behind the process interruption. To avoid doubts, even if the group is, presently, in normal operation, the last error is always preserved and showed, being only cleared when VPG goes off and on again.

After group error diagnosis and respective handle action, it surges the evaluation of valve interlocks and forced mode. First, it is verified if the forced mode orders are expired and after that are checked the interlock conditions.

Being executed all the verification steps, including devices and group errors, and determined the interlocks, are achieved the requisites to set the devices output commands. That's why, their FB are invoked once again, but this time passing as input arguments the manual, process and interlock status and order and obtaining on the output side the action that prevails. This is done for all valves and pumps.

Additionally, according with the results returned by the whole program execution, the last step to be executed is update the HMI with that information, namely current mode, step and the overall condition of the group. The remaining indications, namely device status, not require this manual update, since the HMI task assures that by itself.

Finally, the process output image is ready to be transferred to physical side.

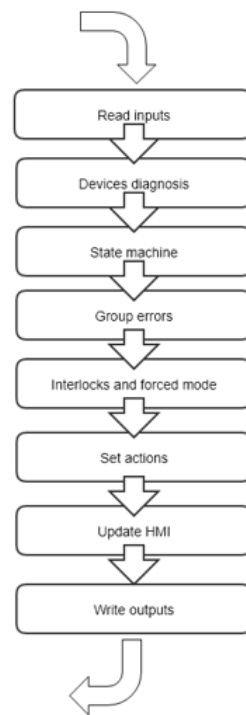


Figure 4.12: Execution cycle.

#### 4.2.1.12 Task configuration

A task is a composition of programs. On the present application, the control process is supported on only one program, like already referred. That program must be included on the task and then its attributes are configured. Here, the task has the highest priority and runs cyclically with intervals of 20ms. The period calculation was supported by the process specifications initially established. Vacuum is a slow process and time requirements are generally not too urgent. The only constraint is to ensure that when an error occurs, the system reacts in a time range inferior to 150ms. The current configuration is properly justified on [5.2](#).

#### 4.2.2 Human-machine interface

The human-machine interface allows the user to interact with the process, monitoring its flow, seeing at any moment the state of each individual device and the whole group condition, and set his intentions, namely turn on/off the group, ask mode change and, eventually, manual control of equipment. The interface should be as much intuitive as possible in order to not cause misinterpretation on operator side and refresh periodically with a time constraint that guarantees that the data showed is always up-to-date.

Regarding to available operations, the following functionalities are included to offer a quality and complete interaction human-machine:

- Monitor the pumping process including all process variables, modes, steps and state machines;
- Control the pumping process (start and stop pumping, change modes, among others);
- Control and monitor the bake-out process;
- Manual and individual control of all VPGM devices;
- Configure process settings (delays, timeout times and others).

Besides the exposed conditions, it is also important to consider the existence of different user's status. Since some actions may put the system on a dangerous condition that might cause the destruction of the circulating beam, compromising, consequently, all accelerator operation, it is mandatory to make a distinction on the privileges inherent to each kind of operator. The standard user can only turn on/off the group process, alternating between Stop and Pump modes, manipulating VVR valves, bake-out and vacuum gauge and, on the opposite direction, monitoring each device, vessel temperature and pressure and group errors. The privileged one (expert mode) needs to log on the system to access to reserved functionalities, including full permission to explore the remaining process modes, previously reserved, control, individually, the available equipment, set group configuration, namely activate or deactivate valves, since some of them are not always present, and process parameters. Relatively to diagnosis topic, the information provided is equal to the standard user one.

The control process implementation is based on CODESYS runtime. With a more integrated solution in mind, the HMI is developed on the same platform environment. CODESYS offers a visualization package that supports the development of graphical interfaces on the same application inherent to control process. This implementation scenario is characterized by drag and drop blocks, included by default on library. These may be of distinct kind, namely buttons, labels, text area introduction and predefined synoptics, being possible to the developer to submit his own pictures. Additionally, alarm flags may also be configured to each block. With these available features, a graphical interface may be easily assembled.

Besides that, CODESYS also offers another relevant feature related to visualization implementations. It is possible to include an interface on a webvisu package. This add-on creates a web server, spreading the implementation through all the devices connected on the LAN where it is inserted the running device. To access the interface, it is only required a web browser with HTML5 support and, once inserted the local address and respective port, the main screen will be immediately showed. If a project includes several graphical screens, it is defined the first one and set a proper linkage between pages.

On this project, there is one main screen composed by graphical diagnosis of the whole group and a frame section. This last field is related with available interactions and user inputs, being, at start-up, fulfilled by basic mode interface. According with the requests made, the frames switch one after other, including log in, expert mode, manual control and settings frames.

### 4.2.2.1 Graphical diagnosis

The devices and group diagnosis is a relevant section present in the main interface screen. The page is separated in two vertical sides, being the left half related to graphical diagnosis and the right one associated to user inputs, being associated to one of already announced frames.

The diagnosis interface is intended to be an intuitive representation of the VPGM and, with that purpose, the devices have a disposition like what should be expected on a real group, trying to create the flow since vacuum vessel, at top, until the atmospheric environment, at bottom.

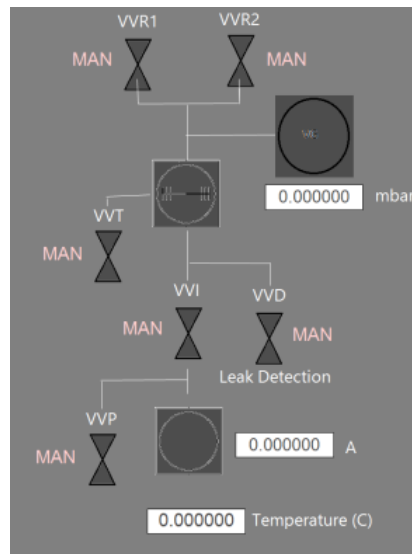


Figure 4.13: Synoptics graphical disposition.

As exposed, there are six valves, two pumps and a vacuum gauge. Some of the valves, however, are optional and, for that reason, at first system start-up only the mandatory ones (VVI in this case) are showed, being the others hidden. It is user responsibility to set a software configuration compatible to hardware on the manual mode page.

Regarding to valves, according with what was said on 4.2.1.1, they may be in three distinct states: valid, warning and error. Still, when in valid state, a valve can be opened or closed. So, the synoptic must be compatible with these conditions and inform clearly to the user the current state/mode applicable. To achieve that goal, a valve representation has associated four distinct colors, as represented on picture 4.14.



Figure 4.14: Valves synoptic (label not included).

The four configurations are assembled on an overlapped way on the interface and the one showed is dictated controlling blocks visibility. This can be done through process variables, namely the ones representing valve state. So, each picture is only visible when the valve state

is the one that it represents. Additionally, in optional valves, their synoptics are configured to be also invisible when they are deactivated.

Finally, valve's synoptic is complemented by a textual information associated with the kind of manipulation, i.e., if the valve is interlocked or in force mode or if it controlled by process or still manually.



Figure 4.15: Valves command mode flags.

The pumps follow the previous implementation approach. In this case, however, when the device is in warning or error conditions, it can still be on or off, since the conditions are not mutually exclusive. To represent such situation is necessary to include extra characters to represent both state and operation mode. The synoptic may be fulfilled with two colors whose purpose is to indicate the operation mode, on/off. The warning and error states are flagged with a yellow or red character, respectively, located outside, although near, to synoptic frame. In a normal device operation, they are hidden. On the turbo molecular pump, due to the relevance of such step on process context, there is still an additional parameter, situated below the previous flags, to announce when the pump achieves nominal speed. Once again, in implementation side, the objects visibility is controlled through process variables.

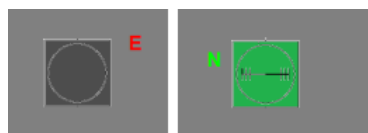


Figure 4.16: Primary pump synoptic on off state with error flag enabled (left) and turbo pump synoptic, when on nominal state (right).

There are also some boxes to inform the measured temperature, pumps current consumption and vessel's pressure, as showed on [4.13](#).

The diagnosis section also includes a field related with the group errors occurrence. There is, at the top, a text area signaling any group abnormal condition. At start-up and considering that was not launched any group error, the user is informed that it is in normal condition and the frame is characterized by a green color. Once an error is verified and while stays active, the color turns into red and a warning signal appears, being the text area filled by the error code and respective description. As soon as the error condition disappears, the object returns to normal mode again, green color, but preserving the last error occurred, shown to operator in order to be aware of any eventual disguised error. This field is only cleared when the process is fully shutted down, transiting to process off step and turning on again.

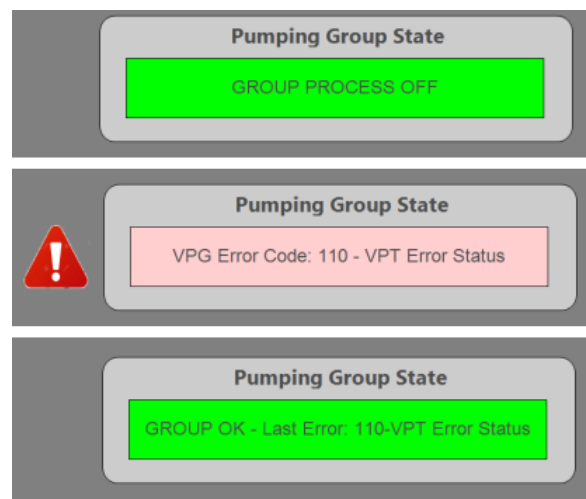


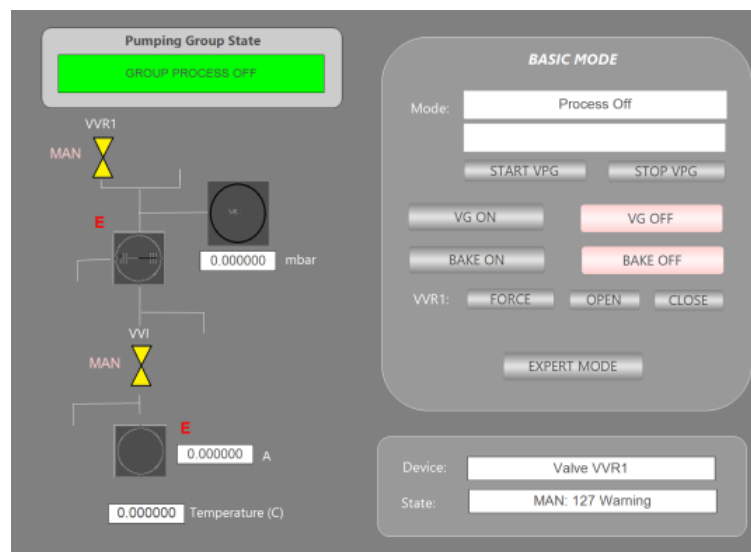
Figure 4.17: Group state synoptic before and after an error occurrence.

#### 4.2.2.2 Basic Mode frame

The basic mode is directed to the standard user and is the first frame showed once the soft-PLC is putted to run. Its purpose is to be used by generic operators who don't have too much technical knowledge and, thus, the array of available actions is too short. Inside this context, it is allowed to start the VPG process, namely the pumping mode, following the current mode and step inside that, and to interrupt its operation changing to process off. Regarding to manual control, the VVR valves may be manipulated (VVR2 must be configured as active on the group to be showed), being offered the possibility of force an interlock and ordering an open or close action. The bake-out and vacuum gauge control is also accessible, namely alternating between switch on and off.

Below the control section, there is an additional diagnosis area. If the mouse pointer is pressed over one of the synoptics, the information related with the respective device will be printed on this field, indicating device name, manipulation context, current state and warning or error code and respective description, if applicable. This is a more detailed description of the information provided by synoptic.

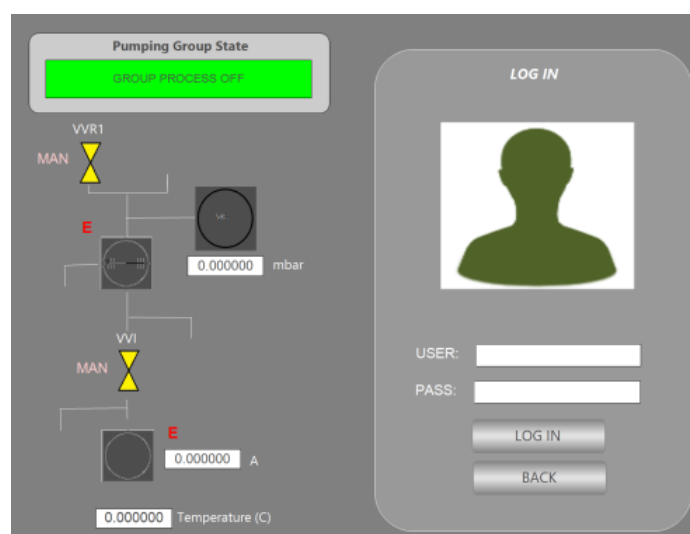
Regarding to implementation, the exchange of HMI orders and process operation is achieved through global variables. For instance, when a manual command is ordered a boolean variable reflects such intention. This parameter, on the process side, will be configured as an input of the respective device FB and, after that, the internal logic will determine if that command prevails or if it is overlapped by process or interlocks. On the other side, to force an interlock, an interface click enables it and starts a timer to measure the elapsed time on forced mode. On the process side, it is verified the permissiveness of such action and, if not approved, the forced mode is canceled and the timer is reset. To the remaining operations the method followed is equivalent, being used variables to flag commands and its validation (confirmed or denied) evaluated on the main process.

Figure 4.18: *Basic Mode* frame.

When the expert mode option is pressed, a login frame comes up, since that mode is reserved to a specific range of users.

#### 4.2.2.3 Login frame

The login window is an imperative step to enter on expert mode. Here, it is showed a typical access interface where it is asked to the user to fill the *username* and respective *password*. Once the *login* button is pressed, the inserted information is compared with the data preserved on the system, currently saved on two string arrays. If the pair user-pass matches with that register, the user is allowed to proceed and the expert mode interface is set. On the opposite case, if the verification fails, it is required to fill the fields again. Instead, the user can go back again to basic mode.

Figure 4.19: *Login* frame.

#### 4.2.2.4 Expert Mode frame

Once overpassed the sign in step, the user has full access to expert mode. Contrasting with the basic one, the whole process is accessible and, if not violate the set rules, almost there are no restrictions to user actions.

Firstly, like on basic mode, it is possible to turn on the process as well as interrupt it, being above that buttons printed the current mode and the step being executed. Besides this, when the process is already on, the user may require a mode transition, selecting the one to which the process should go. This is done on set mode section. Pressing the arrow, it surges a list of all available modes and it is only necessary to select the one intended. After that, it is just pressed the *set* button and, if permitted, the system transit to that mode. If not allowed, the process remains on the current one. Additionally, when the process is one specific step and the condition inherent to the next transition is still not true, the user may advance that state pressing the *skip step* button.

To configure process parameters, such as timers and other process behaviors, it is available the *settings* section.

Finally, to manipulate each device individually, it is also accessible *manual control* window.

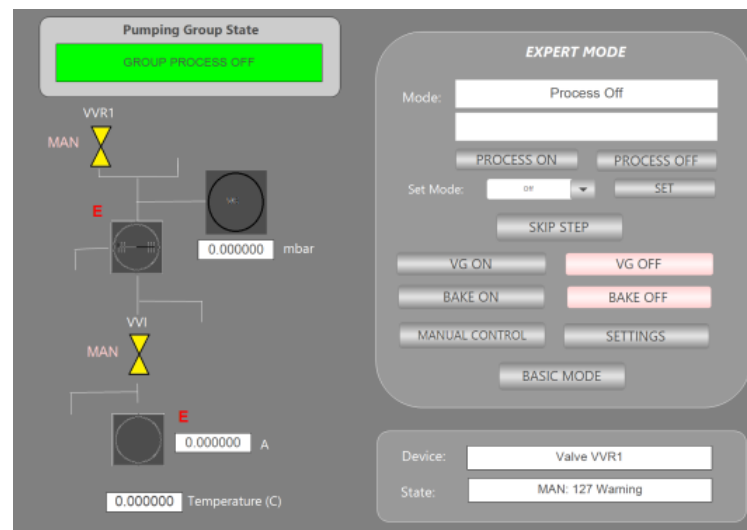


Figure 4.20: *Expert Mode* frame.

#### 4.2.2.5 Manual Control frame

On this screen, it is listed all the possible devices that can compose the VPGM and the respective available commands. These commands may be pressed but, if there is a process or interlock action that overlaps such command, it won't be applied. If the process is off, all devices are free to be controlled through HMI.

Regarding to valves, there are six of them. For each one, it is presented the respective name and the available commands in front, namely force, open and close actions. If the valve label box is fulfilled by a green color means that the valve is active and considered on process operation. If such box is red, however, the valve is not activated whereby the respective buttons are hidden. To

active that valve, if desired, the user should click inside the red area and immediately the valve synoptic will appear on diagnosis side and the respective buttons. Of course, this option is only available on optional valves. At first process start-up, only the mandatory valves are activated whereby the user must select, here, the appropriated configuration and that will be preserved on the subsequent executions.

Relatively to pumps, also several commands are available. First, the simpler actions common to both are turn on and off the pumps. Besides that, there are other specific commands for each one. The turbo pump also offers the option to put the pump in standby mode, where the speed is limited on a fixed percentage of the nominal value. This is a static command whereby once turned on only be turned off if required by user. This can be done pressing again the button. The reset option is associated to a pulse signal and, consequently, when pressed, will only be active for the time range necessary to reset the pump, 1s in this case, and after that will be automatically turned off. On the primary pump, the thermal protection reset has an equivalent operation, being the command switched off after the same period.

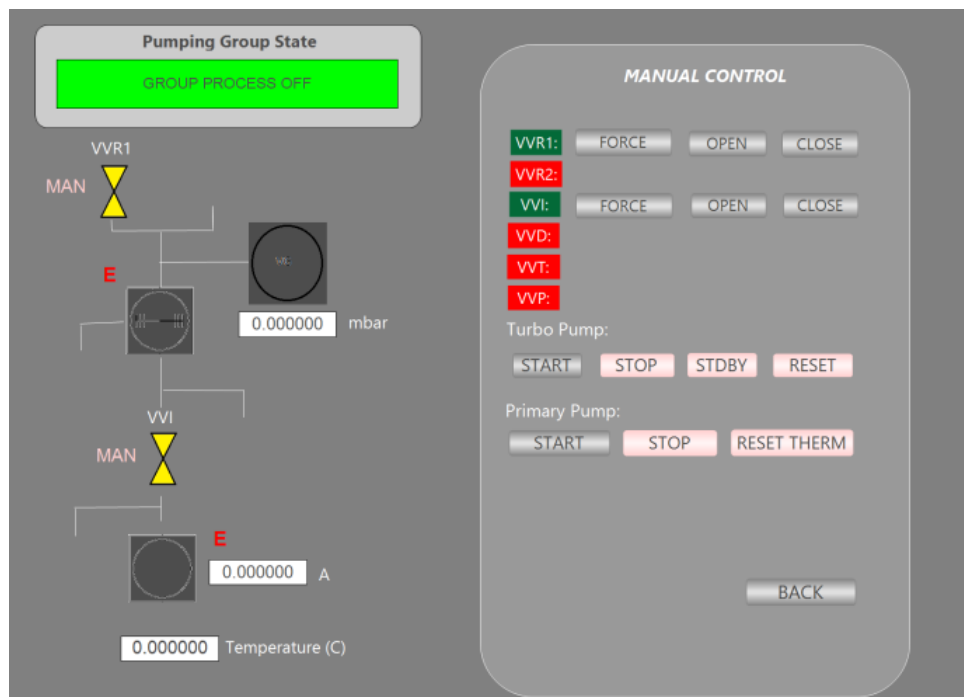


Figure 4.21: *Manual Control* frame.

#### 4.2.2.6 Settings frame

The settings interface allows the configuration of all customizable parameters inherent to process. These modifications can be done at any moment of the process. Among these, it is included pump start and acceleration times and specific timeouts, other temporizations related to venting process and still parameters associated to group errors handling, namely auto-vent and auto-restart and, in this last case, the limit number of restart times. The numerical fields are configured through a numerical keyboard that comes up when user presses over the intended variable. Once configured,

the respective *set* button must be pressed, otherwise the action will have no effect. The current configuration is, then, saved. The same action can be performed using the saving icon on the second page of this section. Here, it is also possible to update all the fields with the last saved configuration parameters pressing the refresh button on the top of the page. At first process start-up, the configuration saved on the last group operation is loaded, being that preserved on a text file on *BeagleBone* memory.



Figure 4.22: *Settings* frame.

#### 4.2.2.7 Task configuration

As stated on 4.2.1.12, the control procedure implementation is supported in only one program, VPGM, that is included on the main task. This runs cyclically with an interval of 20ms due to the urgent response required when an error is launched.

In this case, the main purpose of interface is to show group information and set user action on opposite direction. The commands sent are related with manual operations on devices and not with error handling actions, since these ones are generated by the own process. Justified by these facts, the time constraints are more relaxed and should only assure that the graphical data provided to user is up-to-date. So, there is no need to join visualization program to process task. The graphical interface has its own task. The execution mode is cyclic and the task interval is 100ms.

Besides the task parameters, it is also necessary to customize some aspects regarding to visualization manager and web server. It must be defined the start screen, *hmt* file name, refresh rate and resolution. The basic mode screen is configured as the main window and the remaining values are maintained with the default configuration. As consequence and since the port 90 is the default one, to access HMI remotely on network, it is just writing the running device IP followed by port and file name, like: *IP\_ADDRESS:9090/FILE\_NAME*.

### 4.3 SPI software

As briefly already suggested, the data exchange between ICs and core unit is established through SPI protocol. The amount of information transmitted on both sides, to and from *BeagleBone*, is fixed, but the respective content should be defined on a previous configuration state. Generally, each device is composed by several internal registers, that might support read and/or write operations, related with device configurations, conditions of operation and control of data flow, selecting the ones that are sent to master. In other cases, this procedure may not be possible and the data is not susceptible of customization.

#### 4.3.1 Serializer serial interface

The digital input *serializer* chips offer a SPI interface with no option of customization. Internally, they are composed by only one shift register with 11 bits. This range includes not only one bit per input state as well as three diagnostic bits, as represented on picture 4.23.

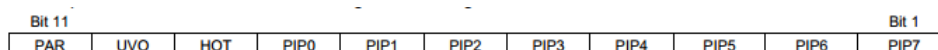


Figure 4.23: Sequence of status bits in *Serializer* [17].

The HOT bit has the purpose to flag an over-temperature condition. The bit is set when the device internal temperature reaches 150°C, offering an alarm indicator similar to HOT pin, accessible externally on IC package.

For other side, the UVO bit is related with under-voltage device condition. That's why, when the power supply is lower than the imposed internal threshold, the situation is properly marked setting the bit. Once it disappears, the bit returns to normal state, low level.

Finally, the parity signal state is set according with the others, i.e., this bit is flagged always that it happened some change in one of the previous bits, namely a change on input status, under voltage or over-temperature conditions.

The indicated array of bits is transmitted to SPI master as soon as the select slave signal is putted on low state, requiring a total of eleven clock cycles to stream all bits. Since there is no need of configuration or address of exchange data, there is no information flowing from master to slave, MOSI line. However, before each communication, to ensure that the data is not outdated, the register should be updated with the input current status, lowering the *Load* line during, at least, one clock cycle, independently of *Select Slave* one.

Additionally, it is also necessary to configure SPI mode as well as clock frequency. According with *datasheet*, the frequency might go up to 100MHz. To assure bus compatibility with remaining nodes, a value of 1MHz is configured. Regarding to SPI mode, there is no explicit information, but, taking into account waveforms showed on the document, the device supports mode 0.

In this application, there are three *serializers* arranged on daisy chain configuration. For that reason, the output data is composed by three sets of 11 bits. The first is related to last device on chain and so on. This is important to decode the received data, on master side, and associate each bit to respective physical signal.

### 4.3.2 Thermocouple serial interface

Unlike previous IC, the *thermocouple* disposes of sixteen internal registers. Among all of them, there are registers related to device operation mode, cold-junction measurements, temperature acquisition and alarm conditions customization. Each one is composed by 8 bits and has associated an address. A register may support read and/or write operations. In cases where both actions are allowed, to distinguish which one is intended to be performed, there are two distinct addresses, one to write operations (with MSB set high) and another to read the register (MSB set low). Thus, when executing one of these operations, in first place it is sent the register address and just after the respective data, if applicable. If several consecutive registers are intended to be read, it is just necessary to indicate the address of first one and define an adequate number of clock cycles and, then, the data byte contained in each register will be streamed one behind the other.

Before initiating its normal operation, the device must be configured according with the application requirements. In this case, it is set the register values showed on table 4.2.

Firstly, the alarm flags are by default masked. These include high and low *thermocouple* temperature, high and low cold-junction temperature, over and under-voltage and open-circuit detection. So, they are enabled on fault mask register and the respective thresholds configured in different registers. In this utilization, they are all enabled but the thresholds remain with the default values. Although, if eventually they won't be suitable, it is possible to change them. The

Table 4.2: *Thermocouple* registers configuration.

| Register (Address)             | Value | Description                                                                                               |
|--------------------------------|-------|-----------------------------------------------------------------------------------------------------------|
| Configuration 0 Register (80h) | A0h   | Automatic conversion mode (each 100ms), open circuit detection, CJ sensor enabled, comparator fault mode. |
| Fault Mask Register (82h)      | C0h   | Enable all fault detections.                                                                              |

device package also includes an external FAULT pin flagging the same situations, being, however, not possible to know the reason behind that, unless accessing the fault register.

The configuration registers allow to set device operation and define acquisition aspects. On register 0, it is enabled the automatic conversion mode (one acquisition every 100ms), the internal cold-junction temperature sensor (since it is not justified the use of an external one) and it is also activated the open circuit detection. On other hand, the register 1 is associated to *thermocouple* type selection, being by default set type-k, and the number of samples acquired per conversion. On this last scenario, it is defined just one sample, because conversion time increase with the number of samples. However, if the results won't sufficiently accurate, this parameter might be increased to achieve better performance.

The previous steps are executed just once on device initialization. After registers configuration, the device is ready to operate and, periodically, it is only necessary to read the *thermocouple* temperature value. This value is codified in 19 bits and retained on three consecutive registers, being 0Ch the address of first one. The read operation, however, should be executed at once and just after of a conversion to assure consistency among all bytes. In order to know when a conversion is over, the DRDY pin shall be considered. This signal goes low when a conversion result is ready to be read, returning to high state when such action is performed. After every acquisition, it is also necessary to configure again automatic mode, otherwise the device will stay on normally off mode. Finally, regarding to clock frequency, the same value previously used is maintained, 1MHz. Relatively to SPI mode, the device supports any configuration, but, since it is the most common among used ICs, mode 3 is selected.

### 4.3.3 ADS serial interface

THE ADS8665 offers a SPI interface like the ones previous described. The device is composed by nine configuration registers. Each one of them is result of four data byte registers, resulting in a total of 32 bits. Among other options, they are related to input and output data configurations, SPI interface definitions and alarm settings. Besides the normal read and write operations, the registers also support the write action of just some bits instead the whole word, set or clear a bit. Since in some registers there are reserved fractions of bits that shouldn't be overlapped, this kind of register manipulation is preferable. A command sent by master is characterized by 32 bits, being most significant 5 bits related with intended action and the remaining ones with register address and data to be written, if applicable.

Before entering on operation, the device must be configured according with table 4.3.

Table 4.3: ADS8665 registers configuration.

| Register (Address)    | Command sent                           | Description                 |
|-----------------------|----------------------------------------|-----------------------------|
| Device_ID_Reg (02h)   | D4020001h (Dev 1)<br>D4020002h (Dev 2) | Define the ID of each chip. |
| SDO_Ctl_Reg (0Dh)     | D00C0100h                              | Configure SPI protocol      |
| DataOut_Ctl_Reg (11h) | D0107D00h                              | Configure output data.      |
| Range_Sel_Reg (14h)   | D4140009h                              | Set PGA gain.               |

On a brief explanation, the first step consists in define device ID. Since it is being used a daisy chain configuration with two devices, this operation is important to identify the IC related to some set of returned data, in case of need. This is the only configuration step in which data diverges between chips, being the next steps executed equally on both elements.

Other required configuration is related with SPI protocol. Besides the conventional SPI interface, it is also supported a multi SPI protocol, which, instead of one, has available two output data lines. Since this feature it is not required for the used communication protocol, this is ignored and the second data line is used as an alarm output pin. Such definition is done on register SDO\_CTL\_REG.

After that, it is necessary to set the output data that is sent to master. This array is composed by 32 bits, including acquisition values as well as diagnostic data. The information included, however, is susceptible to be changed. It means that some bits may be discarded and others added. Here, the register is configured to enable the presence of alarm flags, namely AVDD and input alarm signals, device ID, acting as data label, and, of course, input signal measure. The internal ADC range as well as parity bits are disabled for not being relevant its inclusion.

Besides the previous aspects, it is necessary to set the internal PGA gain in order to achieve the desired input range. According with the input signals (0-10V), a gain of 2.5 is associated to the best ADC resolution. By default, a gain of  $\pm 3$  is defined, being necessary to change RANGE\_SEL\_REG register.

The remaining configuration registers stay with default settings.

Once done the device configuration, it begins the acquisition process. The control and monitor of operation might be complemented taking advantage of external package pins. The conversion starts as soon as the select slave signal goes to high state. During the process, RVS output is on low level, switching to high when it finishes. In this moment, the data is ready to be read. This can be done lowering the *Select Slave* line and, unless some of the previous commands is sent, read or write action on registers, it is initiated the stream of output data compliant with the format specified on respective customization register. Since the devices are in daisy chain, the bytes related to each other are sent one behind other and special care must be taken on decode procedure to not confuse data and physical signals association.

Regarding to SPI bus mode, this can be set on a specific register, `SDI_CTL_REG`. By default, mode 0 is configured and, once it doesn't cause any compatibility issue, it is maintained. The clock frequency, as on other cases, is 1MHz, being included on the range supported by the IC.

#### 4.3.4 Output driver serial interface

The SPI interface of output drivers depends from external circuit. The SEL1 and SEL2 pins are used to select the kind of interface desired. The SEL2 pin must be at high logic level, otherwise parallel communication would overlap serial interaction. If SEL1 is low, 8 bit SPI mode is used. Here, the data from and to master is grouped in sets of 8 bits, being the master allowed to set the state of each output and receiving the respective faulty state of them as feedback information. If SEL2 is high, 16 bit mode is considered. This is preferred when compared with the first because it offers a more complete diagnosis, providing bits related with power supply and package condition. Such information is also available directly on package pins, but, this way, it is ensured some redundancy.

This device doesn't require any previous configuration before starts its operation. So, when exchanging data between master and slave, it is not necessary to specify the operation (write or read) or the address, sending just the desired state to each output. A high logic value is associated to on state and the opposite turn off the respective output. On the opposite direction, it is sent to master current condition of each output, being a high logic value associated to a faulty state, related to output over-temperature. Besides this, it is included a power good bit and a case temperature warning indicator (like external package pins). There is also a bit related with internal voltage regulator, but not used on this application. The input and output frames are represented on tables 4.4 and 4.5, respectively.

Table 4.4: Master-to-slave 16-bit frame [20].

| MSB |     |     |     |     |     |     |     |  |  |  |  | LSB |    |    |     |
|-----|-----|-----|-----|-----|-----|-----|-----|--|--|--|--|-----|----|----|-----|
| IN7 | IN6 | IN5 | IN4 | IN3 | IN2 | IN1 | IN0 |  |  |  |  | P2  | P1 | P0 | nP0 |

Table 4.5: Slave-to-master 16-bit frame [20].

| MSB |    |    |    |    |    |    |    |       |        |    |     | LSB |    |    |     |
|-----|----|----|----|----|----|----|----|-------|--------|----|-----|-----|----|----|-----|
| F7  | F6 | F5 | F4 | F3 | F2 | F1 | F0 | FB_Ok | /TWARN | PC | /PG | P2  | P1 | P0 | nP0 |

Additionally, as observed, there are some bits related to parity check. This feature has the purpose to detect any communication error. Therefore, for one side, it is necessary to verify, on master side, if the data received matches the parity bits match with the remaining bits and, on other side, process the data that will be sent and calculate the respective parity. Such calculation shall be done according with the following rules:

$$\begin{aligned}
P0 &= F0 + F1 + F2 + F3 + F4 + F5 + F6 + F7 \\
P1 &= PC + FB\_OK + F1 + F3 + F5 + F7 \\
P2 &= \overline{PG} + \overline{TWARN} + F0 + F2 + F4 + F6 \\
nP0 &= \text{not } P0
\end{aligned}$$

Figure 4.24: Calculation of parity bits [20].

In this case, there are three equal devices connected on daisy chain configuration. Then, once again, the in and output data processing must be made according with their disposition on the chain.

Finally, regarding to SPI bus mode, according with waveforms on device *datasheet*, it is supported the mode 0. The clock frequency follows the previous cases, being used 1MHz.

#### 4.3.5 Implemented Script

All the previous steps must be included on a routine, implemented and executed on master side. This script should not only run the initial devices configuration and control the data exchange during their operation as well as perform a proper setup of *BeagleBone* pins, enabling and define the SPI bus settings.

As referred on the beginning of the chapter, the code is developed on C language and relies on *libsoc* library.

Regarding to routine structure, the initial instructions are related with GPIO configuration and SPI bus. The pins are defined as input or output according with the signal connected, including mainly the devices diagnosis pins and respective *Slave Select* signals. This last aspect is required because the SPI bus includes only one signal of this kind and, since it is used one per each group of chips, it is needed at least four of them. Then, GPIOs are configured as outputs and, when communicating with a device, the respective *Slave Select* line is putted down manually immediately before of first transmission. On this initial step, the inputs are pulled-up, since almost of them are active on low state, and the outputs pulled-down. Finished pins definitions, it is called the configuration function of each device, being performed the actions already indicated.

After the configuration moment, it is started cyclic data exchange with the devices, acquiring the digital and analogic inputs and defining the state of outputs. In each iteration, it is monitored the condition of chips diagnosis pins and, unless the device returns with no request the diagnosis data through serial interface, this interaction is only used for that purpose when one of the external pins flags an abnormal status, because, this way, it is possible to obtain a more detailed description and understand the reason causing the alarm (this approach has the purpose to allow better time reaction of the system when compared with a hypothetic situation where alarm routines are called every iteration). The diagnostic result is, then, registered on a text file in order to allow the user to access it and figure out the motives behind the system malfunction. This file permits the register of historic of group operation.

When the group operation ends, the stop procedures are done, releasing pins and bus configuration. Additionally, the current parameters related to group control process are registered in order to be persistently saved and loaded at the beginning of next execution.

The script structure may be described by fluxogram on picture 4.25.

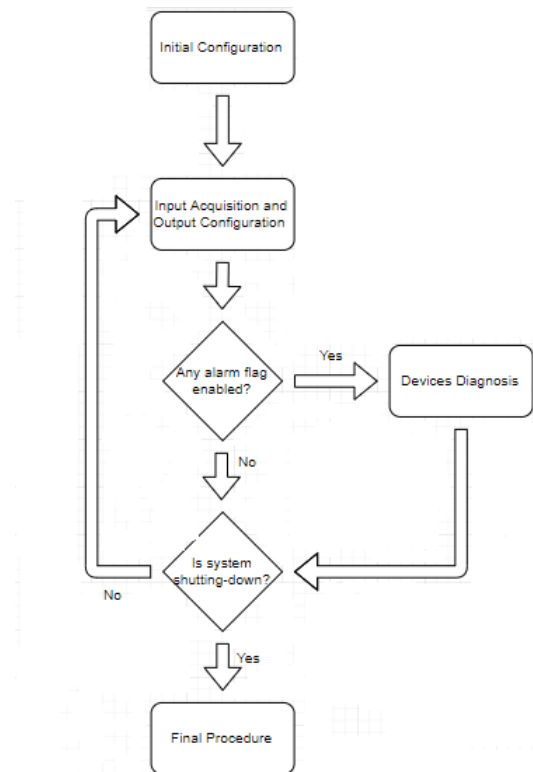


Figure 4.25: Script operation logic

## 4.4 Scripts communication

The interface with field devices is established through electrical card and the data exchange with control layer relies on SPI. The *BeagleBone* CODESYS runtime, however, doesn't offer a library support to this protocol. That reason justifies the development of a separated program to ensure such interaction. But, one problem arises from such solution, the mean of communication between it and control process.

On this application, that gap was suppressed taking advantage of shared memory principles. Since the amount of information transmitted on both sides is fixed and fairly small, a specific range of addresses on user space may be reserved for such purpose and accessible by both processes. Two independent segments are defined, one for input data, sent to CODESYS, and, on the opposite direction, another one for output data, used to send commands to group devices. With this approach, each set of data is written by just one process and read by the other, avoiding the need of process synchronization and, consequently, preventing issues that could arise from this, namely accidental simultaneous accesses. The premises behind this strategy are dictated by POSIX APIs.

Regarding to SPI side, several steps shall be followed, reminding that the program is developed in C and, then, the used functions are according with this. In first place, a memory area need to be reserved, assigning it to an address. This is achieved calling *shm\_open()* function, linking a file descriptor to a shared memory object. The name used to the file descriptor represents object ID and is the one that needs to be used on CODESYS side, when performing the same procedure, in order to assure that the addresses match. Additionally, as function argument, it is provided the type of operation to be performed on register (write/read), using for that specific macros (see [75]). After that, it is executed a memory mapping, *mmap()*, between a process object, with a fixed size that mentioned, and the memory object already defined. In this case, two separated structs are shared, being returned pointers to each one of them. The subsequent write and read operations might be performed directly on memory. However, this isn't a good methodology, since, in the middle of this process, some read action could be performed by the other process and, consequently, some parameters would be updated but other not. To ensure that all the data is referred to the same time instant, the acquisition is made to a local struct and, in the end of each iteration, is copied at once to shared address, ensuring time consistently among all parameters. Referring to outputs to be set, the previous ideas are common, on the beginning of each iteration the shared struct is loaded to a local parameter and the commands sent are based on that.

On CODESYS side, the procedure is very similar. CODESYS library includes support for handling with shared memory objects with a interface very similar to SPI side, when using ST. Initially, a shared object is created, indicating the respective file descriptor to be associated. If it already exists, there is no incompatibility and a proper handle is returned. After that, write or read operations may be performed, indicating the data struct and respective size. Following what was already said, the inputs acquisition is performed on the beginning of scan cycle and the outputs are written at the end, in order to be compliant with the IEC 61131 standard.

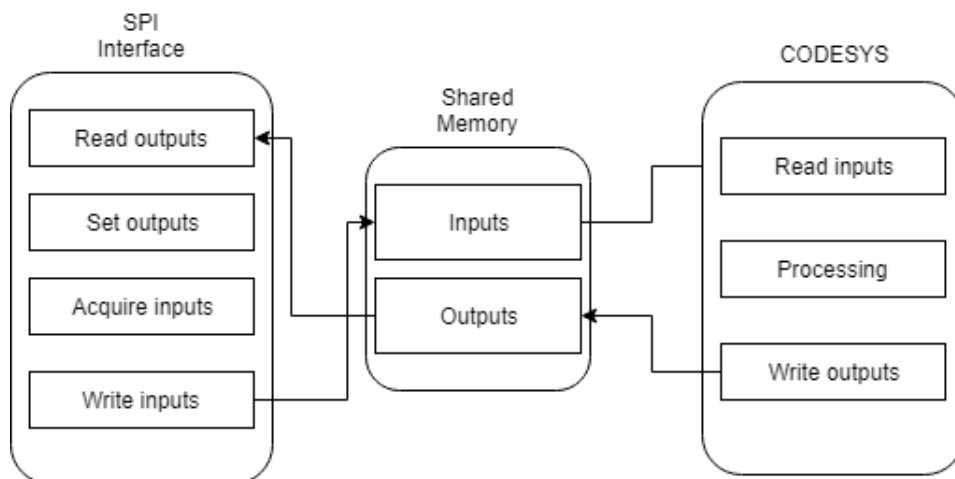


Figure 4.26: Data transmission between programs over shared memory.

The two previous processes are not synchronized. Although, the write and read operations are executed at once inside of respective running cycle, not compromising the data consistency

and ensuring a reliable transmission. Eventually, the only drawback that may be associated to this solution is a more difficult prediction related to time response. Those aspects are analyzed on chapter 5.

## 4.5 Profibus stack

The *Profibus* support is an additional feature required to full replace PLC operation. On this application, a dual role station is considered, acting as master when interacting with turbo-molecular pump controller and vacuum gauge and as slave when integrated on vacuum SCADA system. Like established on 2.3.6.2, the proposed approach is to test an already developed open-source stack, related with project [50].

Regarding to OSI model, *Profibus* operation relies on three of the standard seven layers, namely physical (layer 1), data link (layer 2) and application (layer 7). Therefore, the implementation is compliant with this, being developed a file per layer, configuring all the rules dictated by network premises. The code is written on python and may be described by the architecture represented on 4.27.

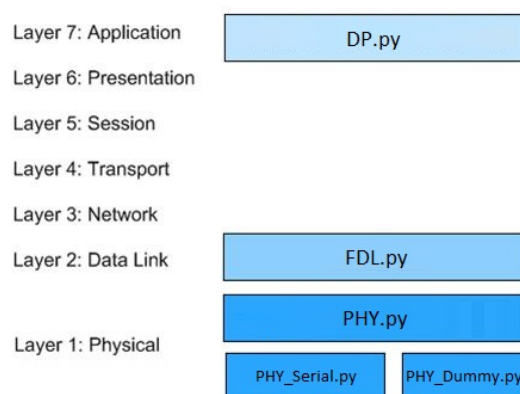


Figure 4.27: *Profibus* stack architecture (adapted from [25]).

Respecting to physical layer, this relies on three main files. The *phy.py* is the main one, defining all the generic aspects related to physical interaction, implementing send and poll operations and configuring the communication baud rate, that may be since 9.2Kbps until 12Mbps. In short, it configures the physical driver. On this case, the physical communication relies on RS485 protocol, being necessary to configure the serial port and control the data flow. The *phy\_serial.py* implements such methods. Alternatively, for testing purposes, a dummy slave may be created (*phy\_dummy.py*), simulating a physical station, providing an echo response when a request is sent, i.e., sent back the received information. This is just to confirm the proper stack operation without using a real slave.

The data link is codified by *fdl.py*, implementing the Fieldbus Data Link layer. This is responsible for all the aspects related with frame control. So, the data unit received from upper layer is

encapsulated on a larger frame. being calculated the start delimiter, that will depend from the telegram type, the respective length, the source and destination addresses, that also consider if service access points are or not included, the control function and bits and, finally, the checksum calculation. This actions are performed over several methods and, once the frame is built, it is sent the array to physical layer in order to be transmitted. On the opposite direction, the data received and sent from physical layer is decoded, checked its validity and, in affirmative case, each individual field is transmitted to application layer.

The application layer encodes each one of requests or commands that can be sent. Among these, there are, for instance, the diagnosis, check configuration, data exchange requests and others. Therefore, exists one class per request. including methods to compact the data and send to the lower layer and also other ones to decompress the information received on opposite direction. The ground rules are defined on *dp.py* file.

The group of files explained implements the three layers stack. However, the operating behaviour will vary with the kind of station considered. On this road, there is the *master\_dp.py* that is the implementation of a master station. Remembering what was already said on 2.3.6.2, a slave operation is described by a state machine, starting from slave initialization, passing over waiting diagnosis, waiting parameterization and waiting configuration steps until starting data exchange. Compliant with it, the master running operation passes through the same steps, sending in each one the frame that matches the slave characteristics. Each step is configured with a timeout whereby if the expected response is not received within this range, the state machine is reset. These values, however, may change if they not fit the intended communication needs (if they are too tight or too relaxed). When the data exchange is reached, a total of five faults is tolerated before a reset.

For a correct slave configuration, the master needs to know its specifications. These are provided on the respective GSD file. To interpret it, a parser module is also included, recording the slave identification, parameterization bytes and additional parameters that are mandatory to establish communication. Besides this, the user needs to create a *.conf* file where it is configured the master station settings. These include type of physical layer (serial or dummy), communication baud rate and the slave specs, including GSD file name, respective modules and additional settings like watchdog and freeze mode support. On the same file it is expected one configuration set per slave.

Finally, it is necessary to create the main program file. Here, it must be declared, at least, one master object, the respective configuration object and, after load all slaves specs, enters on a loop, establishing a transaction with a slave per time.

It is deducted that, currently, the stack only supports master stations, not implementing yet a slave one. Additionally, despite of includes almost every *Profibus* commands and requests, they still not operate on an autonomous way. That's why, the present implementation only assures a simple transaction between stations, passing over all state machine steps and reaching data exchange with success. Anyway, this operation will be properly confirmed on next chapter.

## 4.6 Conclusions

On this chapter, it was presented the software architecture. This is mainly supported by two programs, one related to SPI interface with ICs and another to control process.

The control procedure is implemented on CODESYS. A runtime is installed on *BeagleBone* and this acts as a PLC running continuously the program transferred. This solution is compliant with IEC 61131 standard and, for that reason, offers all the benefits inherent to it, namely the easy programming environment and a competitive industrial architecture. Also, the migration of the code from a previous PLC is also a quick and simple task. The program is built on a modular way, being each device operation implemented on function blocks and, in case of multiple occurrences, it is just to make a number of declarations proportional to it. This way, a more understandable architecture is reached, saving programming effort and facilitating the extrapolation for other application with same equipment. The specific group rules (interlocks, errors and other aspects) are defined on a separated function block. In addition, the HMI is also implemented on CODESYS, taking advantage of visual packages. A main screen is designed, where the user has access to visual and textual VPGM diagnosis. Then, there are several frames related to different levels of interaction (standard user, expert mode, manual control and process settings). The graphical interface is widespread through a web server and might be accessed by any device connected on local network.

The SPI interface program sets the bus rules and interact with the chips, making an initial configuration and, after that, cyclically, acquire the inputs and set outputs. The code is implemented on C language.

The previous two programs run independently. They need, however, to share data in both sides. This is made through shared memory, reserving two memory areas to inputs and outputs arrays.

Finally, a brief overview of *Profibus* stack was provided. This is implemented on python and a layer is implemented per file. It was concluded that all the network ground rules are set, but some effort needs to be performed on upper levels to reach an autonomous operation. Besides that, currently, the support is only targeted to master stations.

## Chapter 5

# Tests and Results

On this chapter, it is intended to evaluate the overall performance of system operation. This can be done according with two point of views. For one side, based on a functional perspective, the global operation of the system is analyzed, performing some specific tests and evaluating if the response generated by process is the one expected (5.1), as well as time constraints inherent to it (5.2) in order to verify if they are within the tolerance time range. After that, on 5.3, some hardware appreciations are also drawn. Finally, on 5.4, HMI suffers a quality appreciation based on what is desired on user side and, on 5.5, are detailed the procedures and conclusions achieved during *Profibus* tests.

### 5.1 Process operation

The implemented control process behavior shall be compliant with the operating rules defined on previous chapter. These combines the several modes, transitions, devices state in each step and the reaction from error conditions. Each one of these topics should be tested individually and, if well succeed, the whole operation is confirmed. The tests were performed in two stages. In first place, the verifications occurred on simulation mode and, only then, real experiments with a physical VPGM were executed.

Regarding to simulation side, first steps are related to soft-PLC configuration. The verification procedure is performed on desktop application putting the program on simulation mode. Once started the execution, it will run as it was actually on PLC. To interfere on the process flow, it is possible to define prepared values to program variables and write them to memory at once. This way, any physical condition is easily recreated, manipulating the input data array with the values expected to be acquired in a real application.

The tests were executed according with an incremental methodology, from bottom to top. They started on group state machine operation. Initially, all machine steps were visited one by one, the respective transitions blocked and the output data, supposed to send to devices, analyzed. For a correct operation, the output commands must match the expected devices state indicated on picture 4.10. When confirmed a specific step, the immediately subsequent transition was fired,

allowing the process flow step by step. After this, it was tested mode change. For that, in every step of one mode, a change was induced and observed its allowance, comparing with rules set by picture 4.9.

Once confirmed the expected behavior from state machine, the tests proceeded with aspects outside of it, namely interlocks. To achieve this, when the dummy group was running, the parameters were manipulated in order to enable interlock conditions, defined on 4.2.1.8 in pseudo-logical language, verifying if the system reacted interlocking the implied devices. Once interlocked, forced mode was confirmed, forcing an interlock and then waiting for forced mode timeout or requesting mode changes, namely for that ones which causes an interlock reset, also referred on 4.2.1.8.

Being succeed all previous stages, the basic operation of the system is achieved. However, the most important verification is still pendent, group errors. The confirmation of this supervisory layer is mandatory to guarantee that the process will react properly when an undesired condition happens. Such tests were performed forcing one error occurrence at a time and, according with present mode, registered the reaction generated. On D, it is registered all the cases that were tested.

Checked all conditions, the simulated operation is in accordance with the theoretical rules imposed to the process, being time to connect to physical group. Since at that moment the card was still not concluded, the interface circuit role was played by a temporary acquisition board with direct I/O connections, allowing to connect all digital signals, both inputs and outputs. This was only possible because the *BeagleBone* disposes of more than one hundred pins and the number of signals is lower than that. The available time to be in touch with the physical group wasn't enough to develop so detailed verifications as on simulation mode. Despite that, the most relevant ones were achieved. The major issue that arises immediately with the migration from simulation to real world was the devices feedback. Ideally, when a device command is set, the feedback is instantly updated, reflecting that new operation state. However, in a real context, the equipment takes a while to match the diagnosis data with the sent actions. For that reason, at first interaction with VPGM, the group was permanently in error condition. To solve that issue, and as detailed on implementation considerations, a time tolerance was implemented to each device, waiting few seconds after each state switch before generating an error. The range defined depends of considered equipment, since, for example, normally a valve responds more quickly than a pump. Overcame this question, the group was turned on and the process executed, running the system on pumping mode until achieve the nominal step. After that, according with previous error check list, some errors were forced and confirmed the equipment reaction.

## 5.2 Time constraints

On last topic, the process behavior was evaluated. Of course, it is important that the system sets the right action as response to an input event. However, it is also relevant that such reaction happens within the expected time range, otherwise will be classified as wrong as well. Therefore, when analyzing a program, the time perspective is so important as the functional one.

### 5.2.1 Kernel latency

One major factor that influences the system reaction time is the kernel behind the operating system that supports program execution. According with this behavior, a list of classifications arises, being, on a general view, composed by two main categories: real-time and non real-time systems. The first group is not only known for having a low delay between an event occurrence and handle action trigger but, mostly, for offering a constant delay even if this is not too low. This deterministic response is the reason that make them so appealing for industrial applications, because the operator always know what to expect. When this is not verified, the response delay is variable, causing kernel *jitter*. How much higher, less reliable the system is, since it is not possible to predict when the reaction will be generated, discouraging the use of non real-time software on critical operations, due the hazardous consequences that might be caused.

On this application, time constraints are not too tight. Vacuum is a relatively slow process, with temporal properties near from seconds, way far to the typical temporizations inherent to real-time. However, when an error occurs, it is expected a fast response, on worst-case scenario, in order of 100/150 ms.

The process execution relies on *linux* kernel, by nature a non real-time system. For that reason, the respective *jitter* must be measured and concluded if fits on application temporal range. To perform such test, several tools are accessible. One of them is called *cyclicttest* (see [A](#)). The program was running during several hours and, after one million of threads, executed with 1ms of interval, the results depicted on table 5.1 were obtained. An high priority was set to thread execution in order to overlap all other kernel processes.

Table 5.1: System latency measures.

| Thread  | Thread   | Thread   | System Latency |            |              |
|---------|----------|----------|----------------|------------|--------------|
| Cycles  | Priority | Interval | Min            | Avg        | Max          |
| 1723335 | 99%      | 1ms      | 15 $\mu$ s     | 41 $\mu$ s | 2600 $\mu$ s |

As observed, the average system latency is 41 $\mu$ s, being the worst case measured near from 3ms. On problem context, even this last value is still acceptable since the system constraints are almost one hundred times higher and a delay of that order wouldn't be meaning, unless the system operation is dimensioned too near from the imposed limit. However, if better performance is desired, other approaches may be followed.

By default, the OS kernel is characterized for being a non-preemptive system. On a short explanation, this means that when a process is ready and once assigned to CPU, its execution will be performed until the end. But, with 'selfish' processes, they may occupy the CPU for too long and the overall performance will be compromised. To overcome such drawbacks, a preemptive scaling algorithm may be promoted. Here, the processes, even if already in execution, can be interrupted and the CPU effort is periodically reallocated to high priority tasks. That reason make

it a suitable solution for critical applications, providing a response not so good as a real-time system but better than the default one (for further information see [A](#)).

After installing a preemptive patch on *linux* kernel, the system latency measures were performed again, on the same previous conditions, and the values achieved are showed on table [5.2](#).

Table 5.2: System latency measures with preemptive patch.

| Thread  | Thread   | Thread   | System Latency |            |            |
|---------|----------|----------|----------------|------------|------------|
| Cycles  | Priority | Interval | Min            | Avg        | Max        |
| 1179528 | 99%      | 1ms      | 9 $\mu$ s      | 20 $\mu$ s | 71 $\mu$ s |

On a brief analysis, the average system latency is similar to previous measures, being the differences more explicit on worst delay, that is, approximately, one hundred times lower. These aspects reinforce the premise that a real-time system (despite on this case not being a native one) is not necessarily faster, but, instead, more deterministic, providing a more narrowed response time.

## 5.2.2 System time requirements

Although the kernel *jitter* might cause some deviations, the system response is mainly dictated by program configurations. As stated earlier, the control process implementation relies on two main tasks. They have several attributes, being one of them the type of execution. In this case, they run cyclically with a task interval proportional to its priority. The task related to main program runs at each 20ms while the HMI is updated on intervals of 100ms. Since the first one has higher priority, it will interrupt the second one in case of need.

### 5.2.2.1 Task interval

To set the interval between task executions, it must be considered the reaction intended by process specifications. As earlier referred, the system reaction is slow and just in case of an error occurrence it is required a response time, in worst case scenario, of 150ms. To meet this requirement, any value below the limit that ensures that may be configured on task. However, decreasing execution interval increases CPU load, what, consequently, might result in a poor processing operation, with large delays.

To deal with the previous issue, a study of the relation between task time and processing load was developed with the purpose to find the best commitment between the two parameters involved. With that in mind, three different intervals were configured on task and measured the CPU load while running the program. The data obtained is only related with main program task, that includes errors verification and respective handling action, since the visual interface is not utterly urgent and only needs to ensure that the information showed is always up-to-date, being constrained by human time constants, too relaxed when compared with the ones here studied. For that reason, there is no need to take it into consideration.

Based on the exposed assumptions, the values measured are organized on table [5.3](#).

Table 5.3: Process cycle time and *jitter* measures.

| Task     | CPU   | Cycle  | Cycle Time (us) |     |       | Jitter (us) |       |
|----------|-------|--------|-----------------|-----|-------|-------------|-------|
| Interval | Load  | Count  | Min             | Avg | Max   | Min         | Max   |
| 1ms      | 71.9% | 142773 | 267             | 514 | 23694 | -682        | 22637 |
| 10ms     | 30.2% | 133780 | 593             | 976 | 14252 | -1922       | 1922  |
| 20ms     | 24.6% | 121727 | 560             | 934 | 11787 | -480        | 482   |

The measurements performed reinforce the ideas previously stated. The CPU load is strictly related with task's interval, increasing as this one decreases. With 1ms of interval, the CPU load is relatively high, almost 72%, and, below that, the program can't be executed, because the CPU reaches near to 100% of its capacity and an exception is generated, aborting the process. In addition, it is possible to confirm that lowering the interval increases the task *jitter*. This results from the fact that the maximum scan time available (which is the interval between tasks) might not be enough to its execution and, consequently, the next iteration start will be necessarily delayed. The accumulation of several delays causes a significant latency, as reflected on acquired data. Finally, it is not relevant to consider intervals above 20ms, since, on those cases, they would be probably too high in order to accomplish system time goal or, if possible, it would be operating near from the limit, not being recommendable.

From all said, a value of 20ms represents the most natural choice. It can combine a reasonable CPU load with an acceptable latency on problem context and still meet the system requirements.

### 5.2.2.2 System response

The system response time is calculated according with previous temporizations. However, it depends from the process moment at which happens an error whereby it is not possible to obtain a fixed value, but, instead, the limits of a range that contains all possible scenarios.

In first case, if an error happens immediately before the inputs acquisition, it is expected a reaction time equivalent to one scan cycle, being considered the average value previously measured with an interval of 20ms, that is, approximately, 1ms. Although, on process implementation, the error verification is made after state machine execution, being the abnormal condition flagged on current iteration, but just handled on the next one. Therefore, on this scenario, instead of one, two scan cycles are performed during system's reaction. Then, a reaction time of 22ms (two scan cycles plus one task interval) is pointed. Additionally, the soft-PLC doesn't interact directly with physical devices, existing an intermediate interface layer. This procedure has an operation equivalent to scan cycle, reading outputs from shared memory when beginning and writing the inputs at the end. Since processes are not synchronized, also here a similar approach is considered, being the best scenario the one where the processes are phased and the data is uploaded from and to physical group immediately. In such case, the 22ms are complemented with two cycle time of

SPI interface program, one to acquisition and another one to set process outputs, resulting in a total of  $22ms + 2 \times SPI\_Cycle$ .

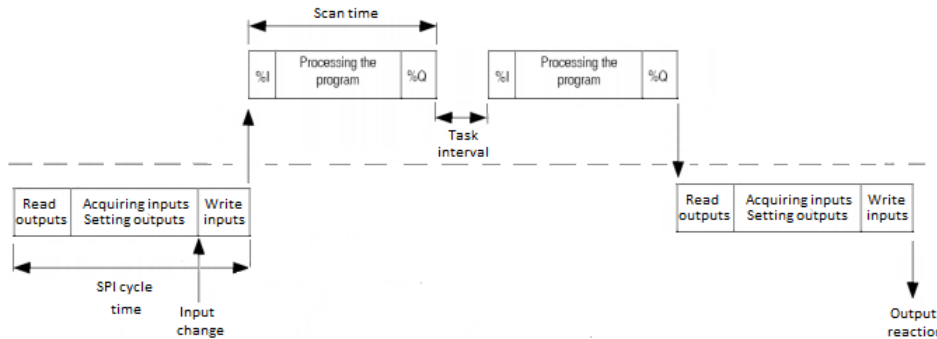


Figure 5.1: Minimum system response.

On worst case scenario, an error occurs on the first instant after the acquisition moment and the programs running are anti-phased. Here, it is predicted that the acquisition takes two cycles of SPI code, plus three cycles of CODESYS program (one to synchronization between processes and other two to processing error) and, finally, similar to acquisition, other two cycles to set outputs. Thus, the time response will be  $2 \times 20ms + 3 \times 1ms + 4 \times SPI\_Cycle$ , as represented on picture 5.2.

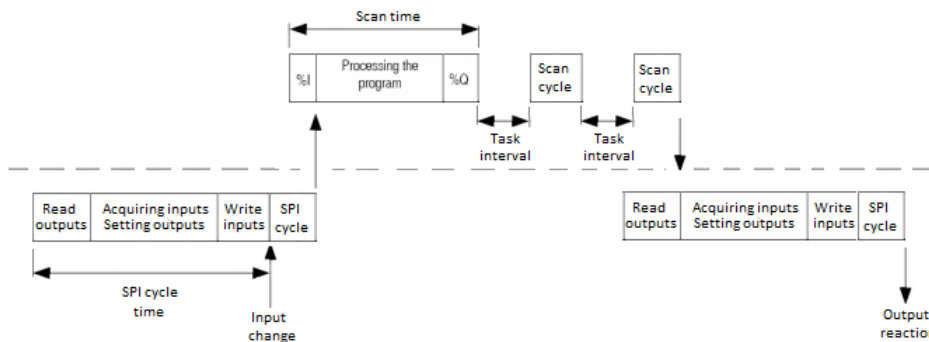


Figure 5.2: Maximum system response.

On previous calculations, it could be also included the exchange time related to data transmission over SPI bus. But, according with the transaction rate (1MHz) and the amount of data (at maximum, 64 bits per transaction), such values, for being in order of  $\mu s$ , are negligible on problem context.

To validate the theoretical values, measures must be performed on real system and compared. The best way to evaluate its reaction is forcing, on hardware side, an input signal level that represents an error on a specific moment of the process and measuring how much time elapses until being observed the supposed output. The procedure was done through an oscilloscope over a total of sixty measures, including reaction to several errors. After a proper statistical treatment, the data dispersion is exposed on graphic 5.3

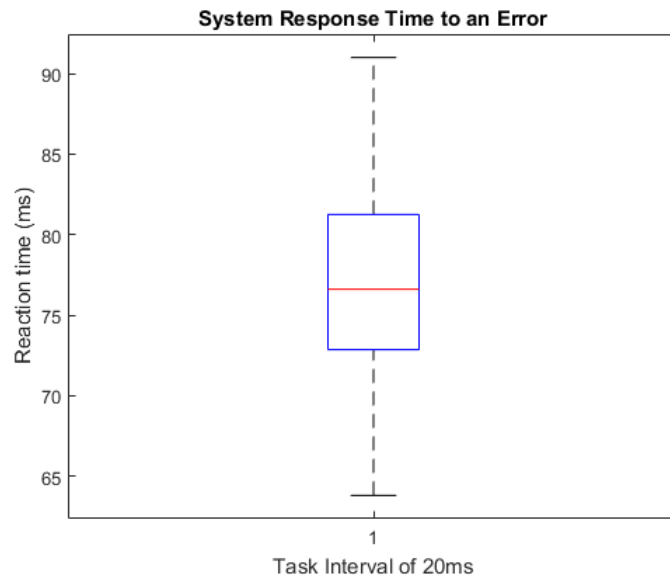


Figure 5.3: System response time.

Taking some attention to boxplot, it is possible to conclude that the response range is inside of imposed limits. By whiskers limits, the minimum reaction time is near from 60ms and the largest value is, approximately, 90ms, significantly below the intended 150ms. Regarding to remaining measures, the graphic suggests that they are uniformly spread along the whole range. Since the median value is more close to first quartile, there is a little higher concentration of values on this region, allowing to point a mean value near to 75ms. All these aspects corroborate the correct time behaviour of system operation.

### 5.3 Hardware robustness

The main hardware component is represented by the electrical board, that includes several modules to interact with physical equipment. Since all board design was made from scratch, before proceeding to building process it is recommendable to perform some individual tests with each component to prevent future issues. With this incremental integration it is intended to reduce as low as possible eventual bugs, what would imply to much effort to be diagnosed on a final solution, when the whole system was assembled, what would happen if a big bang integration testing was privileged instead.

Following the drawn path, a test code was implemented on core unit, not establishing at this point a link with control process. On a brief explanation, each module was mounted with the respective condition circuit and fed externally, using a controlled power supply instead of circuit supply module. After that, namely on acquisition side, the chip inputs were connected to dummy signals, with characteristics equivalents to the ones coming from VPGM equipment. Then, the data transmitted over SPI was decoded and compared to dummy information, being pretended that both values match. On output side, a test load was considered and one output was set at time,

being observed the load state. To evaluate ICs warning flags, some of the abnormal conditions were intentionally generated, causing, for instance, a faulty power supply voltage. Some of them, however, like package over-temperature and analogous flags, were not tested because it would put the component on a dangerous region of operation and, probably, destroy it. A list of evaluated situations is indicated on picture 5.4. On these functional assays, it was just used one IC per module. Despite some of them are mounted on daisy chain configuration, the principles of communication are the same and can be extrapolated, with the addition that, instead of one, there are two devices connected in series.

Table 5.4: Devices tests checklist.

| Device                         | Action                                                                 | Expected Result                                                                                                        | Test       |
|--------------------------------|------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|------------|
| Output driver<br>(VNI8200XP)   | Setting one output at time                                             | Load is properly supplied                                                                                              | Ok         |
|                                | Setting all outputs at once                                            | All loads are powered on                                                                                               | Ok         |
|                                | Sending a frame with erroneous parity bits                             | PC bit is set high on next frame read and FAULT pin goes low                                                           | Ok         |
|                                | Supplying the IC with a low voltage level (below power good threshold) | PG bit is set on next frame read and PG pin goes low                                                                   | Ok         |
|                                | Setting low level on enable output pin                                 | All outputs are deactivated and loads powered off                                                                      | Ok         |
|                                | Heating device case until junction temperature                         | TWARN pin goes low and TWARN bit is set high                                                                           | Not tested |
|                                | Overheating on channel at time                                         | Fault pin goes low and the respective channel bit is set high                                                          | Not tested |
| Digital Inputs<br>(SN65HVS881) | Setting one field input at time                                        | Input bit is set high on serial interface and Parity bit is set high (if number of high state inputs is multiple of 2) | Ok         |
|                                | Supplying the IC with low voltage                                      | Under-voltage bit is set high as well as parity bit of first frame read                                                | Ok         |
|                                | Overheating the device above 150°C                                     | Hot bit is set high as well as parity bit of first frame read                                                          | Not tested |
| Analogic Inputs<br>(ADS8665)   | Acquiring dummy signals inserted on supposed input range (0-10V)       | Decode result matches the value expected from input voltage                                                            | Ok         |
|                                | Forcing an input signal above configured range                         | High input voltage flag set as well as Alarm pin                                                                       | Ok         |
|                                | Forcing an input signal below configured range                         | Low input voltage flag set as well as Alarm pin                                                                        | Ok         |
|                                | Supplying AVDD with a voltage above limit threshold (5.3V)             | High AVDD voltage flag set as well as Alarm pin                                                                        | Ok         |
|                                | Supplying AVDD with a voltage below limit threshold (4.7V)             | Low AVDD voltage flag set as well as Alarm pin                                                                         | Ok         |
| Thermocouple<br>(MAX31856)     | Connecting an experimental thermocouple sensor                         | Acquisition result matches sensor temperature                                                                          | Ok         |
|                                | Letting field connectors open                                          | Fault pin is putted low and Open-circuit flag is armed                                                                 | Ok         |
|                                | Acquiring a TC/CJ temperature higher than threshold register           | Fault pin is putted low and TCHigh/CJHigh flag is armed                                                                | Ok         |
|                                | Acquiring a TC/CJ temperature lower than threshold register            | Fault pin is putted low and TCLow/CJLow flag is armed                                                                  | Ok         |
|                                | Forcing a out-of-range TC/CJ value                                     | TCRange/CJRange flag is armed                                                                                          | Ok         |
|                                | Supplying the device with a negative voltage or above VDD              | Fault pin is putted low and OVUV flag is armed                                                                         | Ok         |

The previous tests allow to note a *datasheet* mistaken related to output drivers. According with the document, the device feedback data sent simultaneously on the opposite direction of an output configuration command is compliant with the format present on table 4.5. However, after some experimental communications, it was easily observed that the returned data, namely parity bit position, doesn't match with that configuration. For not being possible to confirm the right format and to avoid erroneous interpretations, a 8-bit serial interface replaces the initial 16-bit configuration. On master-to-slave data, the command is shorter, being removed the parity bits, table 5.5. On opposite direction, the diagnosis bits are not included now on the frame, being the checking device procedure made just through the external pins on IC package, table 5.6.

Table 5.5: Master-to-slave 8-bit frame [20].

| MSB |     |     |     |     |     |     | LSB |
|-----|-----|-----|-----|-----|-----|-----|-----|
| IN7 | IN6 | IN5 | IN4 | IN3 | IN2 | IN1 | IN0 |

Table 5.6: Slave-to-master 8-bit frame [20].

| MSB |    |    |    |    |    |    | LSB |
|-----|----|----|----|----|----|----|-----|
| F7  | F6 | F5 | F4 | F3 | F2 | F1 | F0  |

On performance field, there isn't a yelling criterion that could be used to evaluate the board operation. The only indicator that might be applicable is the one related to speed of communication between master and slaves. The SPI bus is configured to operate at a frequency of 1MHz. The time of communication will be dictated by the number of bits in circulation, being increased with daisy chain configurations. The largest amount of data is attached to ADS chips, with a total of 32 bits per device on each direction of communication, resulting in a total of 64 bits. This implies a communication time of  $64\mu\text{s}$ . This value, however, is insignificant when compared with the already mentioned temporal magnitudes, approximately one hundred times bigger.

Another source of PCB evaluation is its physical robustness. When in touch with the a VPGM exemplar, it was noted that the crate on which fits the control card is subjected to considerable adverse conditions, not only on operation, but also during the trolley transport, being required some mechanical resistance to support vibrations and physical shocks that can accidentally occurring. For that reason, no PCB area can be exposed, otherwise it will be easily damaged on such situations. Aiming a better compaction and physical protection and, consequently, a more stable package, the whole hardware should be involved by a hard case, letting just the wires connectors externally accessible. To play that role, a prototype was obtained using a 3D printer. As showed on picture 5.4, the model includes two main pieces, a box to save the board and the respective cover. The PCB fits inside on six supports, extruded from the bottom, being these perfurated along all length in order to screw the board on back DIN rail connectors, necessary to fix the whole package on group crate. Since the *BeagleBone* is disposed on a longitudinal fashion on the board and its ports (USB, HDMI, Ethernet and others) are located sideways, two side holes are included on the case, letting space to wires that can be used for such connections. Finally, the cover must protect all the board while allowing the access to signal connectors, being, then, designed four free regions, two of them to inputs and outputs signals, on left and right side, and the remaining ones to flat cable passing. Besides all that, to avoid that the *BeagleBone* is pulled off accidentally, its PCB is screwed on main board through four holes located on the limits of its area. With this, it is increased the hardware protection and robustness, preventing the disassembling of the package, due to vibrations or shocks, during group operation.

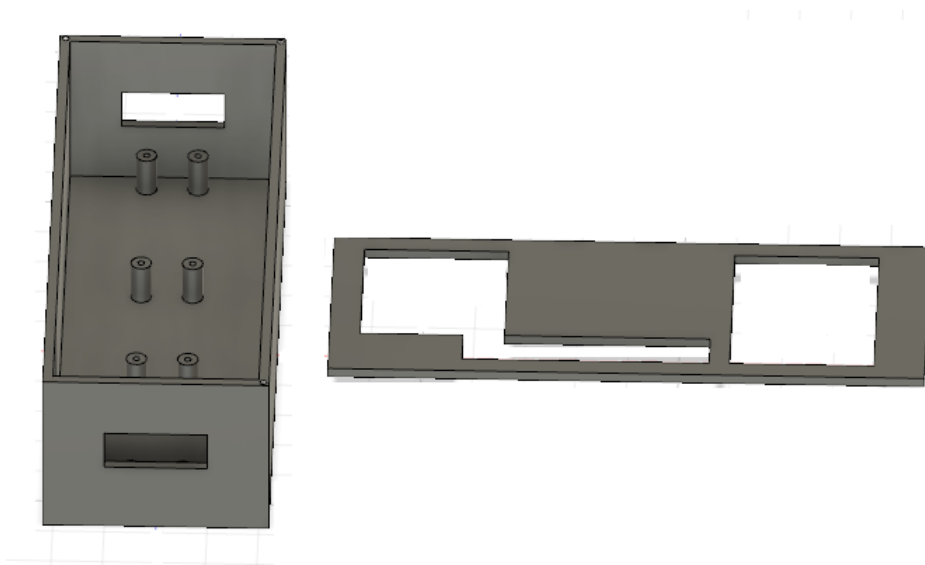


Figure 5.4: PCB enclosure box design.

Additionally, the first in loco connections showed that the spring connectors, initially chosen to facilitate wires installation, don't offer an enough strong hardware, breaking easily and not pressing the wires enough to prevent inadvertent disconnections. That's why, on an industrial environment, it is preferable a screw solution. With that in mind, the first ones were replaced by pluggable screw blocks, being fixed a terminal header on PCB, but the plug can be screwed externally on the wire and, then, docked on PCB terminals. This way, a easy to plug solution is obtain and, since the wire strength is controlled by screw, there isn't the risk of unwanted disconnections.



Figure 5.5: Terminal block and screwed pluggable connector (adapted from [21]).

## 5.4 HMI

The HMI relies on official LCD display released to *BeagleBone* interface. In this application, the connection between the electronic card and the display is established by a flat cable that contains all the signals, including supply and data transmission on both sides.

Referring to its evaluation, there is no quantitative indicator that could give an absolute classification of interface quality. The only numerical parameters that can be referred is that the display offers a 7" HMI distributed along a resolution of  $1024 \times 600$  pixels. Since it is a resistive touch, the user interactions sometimes may not be perceived at first, just requiring a second press on the screen.

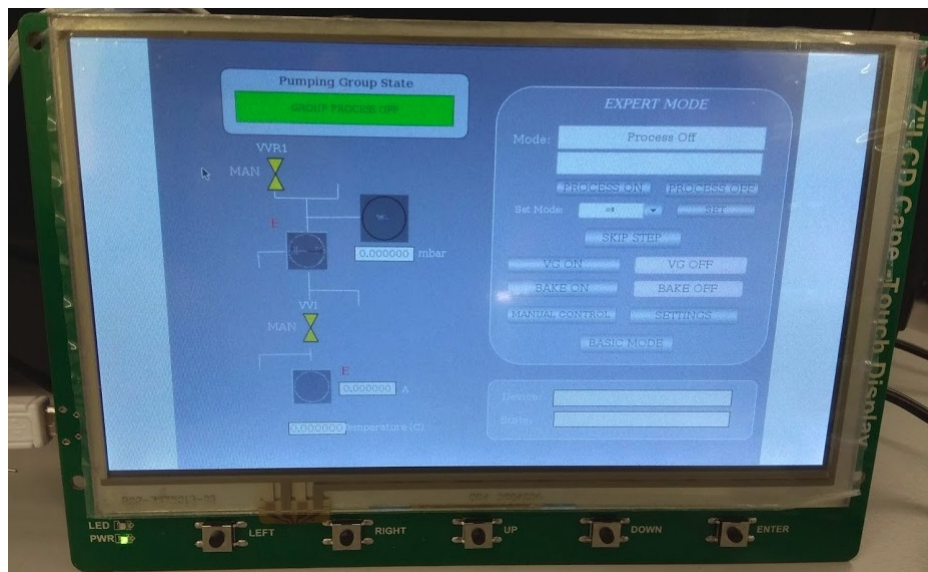


Figure 5.6: HMI display.

Besides these aspects, there is, however, an issue that unfeasible its use. When the *BeagleBone* is plugged directly on display back connector, everything work as expected. But, using the flat cable (the tests were performed with a length cable of 30cm), it is observable some noise and interferences on display. Once the data exchange is made over I2C protocol, the frequency inherent to such signals should interact with each other, corrupting the circulating data. Furthermore, on real group application, the display needs to be near to operator while the card is located at the bottom of the trolley, existing a separation distance around 150cm. Supported on these premises, the present solution is not feasible.

To overcome the listed drawbacks, a different approach might be explored. Some models of *BeagleBoards* (not the one used) include a HDMI connector onboard. This characteristic possibilities the expansion of supported displays, since, instead of being limited to the official one and serial interface, another ones could be used and taking advantage of that kind of interaction. Of course, the HDMI port is just to video output, being yet necessary to receive the information related to user touch. Such transmission might be done over USB. A solution based on these assumptions is presented on [26]. This road would require the adoption of a new *BeagleBone* model, that includes a HDMI port, but, being the code developed compatible with all models and easily migrated with no change, this wouldn't be a problem. Finally, because the board only disposes of one USB host, the connection of LCD would let no connector available to *Profibus* adapter. However, it is remembered that this represents a temporary approach, just to experimental purposes, and on a final solution it would be used not just an isolated converter as well as the connection would be established over the UART interface available on board connectors.



Figure 5.7: 7" HDMI display [26].

## 5.5 Profibus tests

To confirm that the *Profibus* stack operates correctly, it was necessary to make some experimental communications with the slave, acting the *BeagleBone* as master. With that in mind, a *Schneider* slave was used, namely the model *NDP2212*. This is a modular slave, being complemented with a power supply module, *PDT3100*, and also digital output one, *DDO3410*. For being modular, the GSD file is not available online because it depends from the elements connected. For that reason, it must be used a proper software. Taking advantage of *Unity* program, it was selected the slave configuration, generated and then exported the GSD.

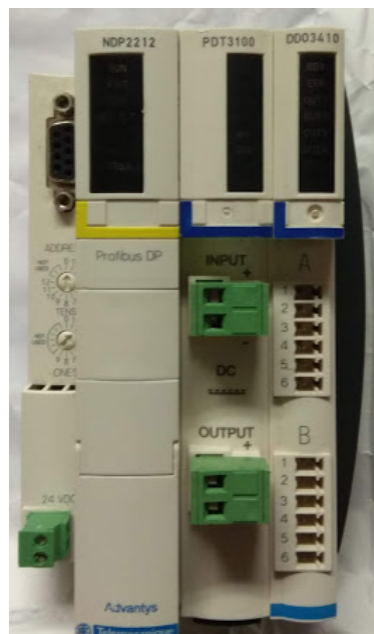


Figure 5.8: Modular *Profibus* slave user for tests.

After that, the configuration file must be created. As referred on 4.5, the stack implies a file of this kind in order to the master knows the configuration of the network, number of slaves, respective addresses and the data exchanged with each one. In this case, being the protocol established

between one master and one slave, the configuration is fairly simple.

Finally, a runnable file was implemented, defining configuration and master objects. A cyclic data exchange was performed, being, at each iteration, set one output and read the inputs on the opposite direction. But, since there is no inputs module, the data returned is related with the previous state of outputs set.

To execute the stack, a previous installation is required. This must be done through the command inside of package folder:

```
root@beaglebone/.../:~# python3 setup.py install
```

After that, the python library should be updated with *pyserial* package in order to provide the necessary interface to use serial port:

```
root@beaglebone/.../:~# pip install pyserial
```

Hereupon, the stack is ready to be executed:

```
root@beaglebone/.../:~# python runnable_file.py
```

With the default stack configuration, it is expected that the communication is established with success, passing over the initial state machine steps and entering on an endless loop of data exchange where the slave outputs are permanently switched, as observed on module lights.



Figure 5.9: Example of a successful communication with a *Profibus* slave.

The first test was performed with default state machine timeouts (1ms per step) and with the minimum baud rate, 9.6Kbps. Then, with a fixed rate, the timeout was gradually reduced until it expires. This test was done in every supported baud rate.

Table 5.7: Minimum timeout required per baud rate.

| Baud rate (Kbps) | Timeout (ms)  |
|------------------|---------------|
| 9.6              | 0.2           |
| 19.2             | 0.04          |
| 45.45            | 0.03          |
| 93.75            | 0.02          |
| 187.5            | No connection |
| 500              | No connection |
| 1500             | No connection |
| 3000             | No connection |
| 6000             | No connection |
| 12000            | No connection |

As expected, increasing the communication speed reduces the system reaction and, consequently, lower timeouts are susceptible to be configured, being, at higher tested speed, 93.75Kbps, a value in order of a couple of dozens of microseconds. This was the minimum value reached. With higher rates, the master wasn't able to start communication with the slave, performing several tries but with no success, not leaving initial state. The reason behind that is surely supported by hardware arguments. For one side, *BeagleBone* UART supports a communication speed up to, approximately, 2Mbps, but not above that. This justifies the absence of connection above 1.5Mbps. However, the failures begin on a frequency ten times below. One possible explanation is related with the USB-RS485 connector. This is composed by a MAX485 IC and CH340 chip, which, effectively, assures the conversion. The *datasheet* also suggests that a speed up to 2Mbps should be supported, but not clarifies the discrete levels allowed, because, commonly, not all values within that range are possible. On a final solution, a more reliable connector shall be selected, but, being aware that the maximum speed will be always limited by UART characteristics.

Despite previous aspects, it was possible to confirm the correct stack operation on master side, establishing with success data transaction with a slave.

## 5.6 Conclusions

On this chapter, it was evaluated the system performance on a temporal and functional perspectives.

In first place, since the *BeagleBone* OS is not suitable for real-time operations, a latency measure was performed. The first results allow to conclude that a maximum *jitter* in order of milliseconds units should be expected. This value is still within the tolerated range taking into account

process time requirements. To a better performance, a patch that implements a preemptive processes scaling algorithm was installed. After new measures, the obtained interval was much more bounded with a maximum value of dozens of microseconds. These values are not still adjusted to a hard real-time control, but fits the present application.

In second place, it was developed a procedure to find out the better task time interval. The tests confirmed that decreasing task interval increases the respective *jitter* as well as the CPU load. For values lower than 1ms, the program is even aborted due to CPU overload. After all, the conclusion reached is that a value of 20ms is the one that allows the best commitment CPU vs delay.

The previous temporal considerations were related with CODESYS program. There is, however, the intermediate layer (SPI program) that ensures the link between it and VPGM. To confirm that the whole data flow path (from SPI to CODESYS and the opposite direction) is within the range required, several measures were done with an oscilloscope. On acquisition side, a faulty signal was forced and, then, it was measured the time elapsed until the output reaction was observed. The tests showed that the worst case scenario is below 100ms (approximately, 95ms). That's why, the response is quick enough, since the initial requirements imposed a value of 150ms.

Besides this, several tests were developed to analyze system functional operation, being proved, for the tests performed (registered on [D](#)), that the control procedure follows the operation rules.

On hardware side, the ICs were interfaced individually in order to evaluate, independently, the behaviour of each one and create the knowledge for the interface with the full board. Regarding to robustness criteria, some updates were suggested, namely the use of an package to protect the board and not leaving PCB area exposed. The HMI display also proves not to be the most reliable option, being its operation characterized by some interferences, probably result of communication protocol (I2C) and the distance of transmission. Hereupon, it was suggested to use HDMI interface to image transmission and serial communication to send touch inputs to core unit. Then, it was suggested a new display supporting the enumerated connections.

Finally, the *Profibus* stack test reveal that is possible to establish with success the communication with a slave, corroborating implementation correctness. Regarding to speed aspects, the fastest communication was reached with a baud rate of 93.75Kbps. Above that, no connection was achieved, probably due to hardware limitations, namely RS485-USB quality of construction. Anyway, since *BeagleBone* UART speed is limited to 2Mbps, this is the highest value that could be configured.



## Chapter 6

# Conclusions and future work

### 6.1 Conclusions

Taking into account the considerations listed on [1](#), the project requirements are supported by hardware and software specifications. At the end, it was obtained a functional prototype, being legitimate to conclude that it was well succeed.

On hardware side, the main goals were related with the support of whole electrical interface with a VPGM. This includes the signals associated to pumps, valves and gauges with distinct electrical characteristics. With that in mind, a interface card was developed, being composed by different modules, one per each group of identical signals. This was designed from scratch and each set individually tested, being achieved the intended behaviour. Then, to process that information, a single-board computer, namely a *BeagleBone*, played that role. The data transaction relies on a SPI bus. This protocol is common to all integrated devices and allows a fast and simple way of transmission, acting the core unit as master and the remaining components as slaves.

On *BeagleBone* side, a fragmented software architecture is implemented. One of the blocks is affected to physical interface, running a program that configures SPI bus and read/write cyclically data exchanged between stations. Such information is processed on an upper layer, associated to process control procedure. This is implemented on CODESYS, i.e., on a soft-PLC that offers a programming paradigm equivalent to a PLC, being compliant with the IEC 61131 standard. The process is full functional, including all modes/steps of operation and not only flagging as well as setting the proper handling action for all listed group errors. This was tested on simulation environment and also on an VPGM exemplar. The temporal behaviour was also evaluated, being confirmed response correctness regarding with that prism, providing, on worst case scenario and considering the case tested, a value lower than 95ms, above the initial stipulated 150ms.

The communication between both layers (SPI interface and CODESYS runtime) is made through shared memory. Supported on POSIX API rules, a region of memory is reserved for saving the data that is accessible from both sides. This represents a easy, reliable and quick way of interaction. Alternatively, a network channel, like Modbus for instance, might be established, offering a more structured data exchange. However, this principle is suitable to communicate with

stations distant on space. In this case, since both communication poles are included on the same system, it makes more sense, at least on a technical point of view, the solution adopted.

The system HMI development also relies on CODESYS. The graphical interface is implemented on a visual package that is widespread through Ethernet, being accessible from any device connected on the core unit local area network. This includes all the features and functionalities ensured by the solution currently implemented on VPGM, offering a full experience to operator when interacting with the group. To display purposes, it is used a 7" touch display, an official *BeagleBone* LCD. However, first tests reveals that the data exchange, through a flat cable, is very susceptible to noise and interference when operating far away from the board. Aiming to achieve a better solution, it is then suggested the utilization of a different hardware, being the information sent over HDMI port to display and received back through USB. This way, it is expected to achieve the intended communication distance without any issue.

According with previous considerations, the next objectives were achieved:

- Building of an hardware architecture able to interface with all devices of a VPGM.
- Study and selection of a platform to act as processing unit.
- Implementation of full automatic vacuum group control process, including errors handling and safety interlocks.
- Development of an HMI with extended functionalities, including devices manual control, process interaction and customization and an extensive group diagnosis.

Therefore, the mandatory goals were all accomplished.

Additionally, although it can't be inserted on the same group, it was developed some work on card *Profibus* support. Remembering what was already said, the idea on this field was to use an open-source stack and integrate it on software architecture. The time available allow to test that stack, developing some experimental interactions with a *Profibus* slave. it was possible to write to outputs and read respective inputs, proving that the implementation is functional. As hardware support, a USB-RS485 converter was used, being putted some effort on changing the device driver compiled by default on kernel, that was not operating correctly.

## 6.2 Microcontroller vs SBC

Besides all the referred aspects, some considerations need to be weaved relatively to processing module. At the beginning, the project proposal suggested a microcontroller based solution. However, such kind of path would require the design of a whole architecture, having as start point the processing chip and defining the remaining hardware to obtain a functional device. In order to adjust the thesis duration to the project demand, an alternative branch was selected. Instead of a microcontroller, a single-board computer was used.

A SBC offers a device with internal microprocessor, memory and I/O interface as well as additional connections. The fact of being a ready to use platform easily plugged to external peripherals

make it very appealing. The additional aspect of relying on an operating system facilitates the installation of any software package, available for a typical PC. This is reflected on CODESYS runtime, allowing the maintenance of IEC standard application, providing a final solution near from conventional PLC one. This represents an advantage from implementation side, because a control process may be migrated to the new solution almost with no change, due to the identical programming environment. Despite these benefits, for having a kernel running behind, the system operation is not deterministic (although some procedures might be done to decrease its latency) and may not be compliant with some industrial applications.

For other side, the microcontroller implies more effort to reach a functional state, but this tailor made perspective allows more customization and the development of an architecture that fits exactly on project needs. In addition, for being free of a kernel, drivers and other background processes, running just the control cycle, a better time behaviour, regarding to real-time criteria, is expected. On the opposite side, the process implementation needs to be developed on native programming language (C, for instance), abdicating from IEC standard.

In short, it can be stated that the choice will always depend from the commitment desired and the final application. On this case, the SBC performance was evaluated to guarantee that it fits on project time constraints. However, it is always worth it to compare both architectures.

### 6.3 Future work

The delivered system is functional and offers an operation that fulfills the theoretical requirements. But, being aware of its final purpose, to be applied on vacuum groups used for pumping LHC chamber (a structure whose construction is in order of thousands of millions of euros), the consequences caused by a malfunction, for example an accidental venting, would include lost of weeks of work, invalidating all human work developed during that range and, on worst case scenario, could compromise/damage the accelerator. With that sense of responsibility, the system still needs to be more exhaustively tested on laboratory environment and just, then, this can be considered utterly reliable.

The *Profibus* road was partially explored, but, due to lack of time, an integrated solution was not reached. Once proved that the stack is functional, the next steps consist in integrating the protocol on system architecture. For that purpose, several paths are susceptible to be paved. The first suggestion points in using shared memory principles to share the protocol data with CODESYS program, promoting an approach identical to the one already followed for exchanging SPI values. Another alternative could be developing a CODESYS driver that would interact directly with hardware layer, providing a more high level interaction, being the network configuration responsibility from CODESYS side and the lower layers transparent to user. Regardless of adopted methodology, the protocol would only be developed for this particular application, not representing a full stack that might be extrapolated for any generic application. If the intention is that, it is preferable the use of FPGA that already implements master/slave stack, just needed to be integrated on a circuit and being ready to use.

In short, the delivered system proves that a solution of this kind is possible in the future, serving its feasibility to give the moto and encourages additional study on this field, exploring not only SBC as well as microcontroller solutions to fully assure PLC industrial applications, allowing a cheaper and more flexible architecture.

# Appendix A

## Real time systems

### A.1 Systems classification

The systems may be characterized according with the time constraints that they offer. They are selected according with the requirements implied by a final application, namely temporal demands. Following this path, a classification tree arises, existing, however, some variations on literature. A soft system is associated to relaxed time constraints and used for desktop applications, not including any critical operation (by critical it is intended the ones that can compromise humankind or the lost of huge amount of money). This represents the more typical kind, being the one where is inserted *linux* [31]. Climbing the stairs, it appears successive tight time constraints and, although the name of each step depends from editor, the last ones can be putted on hard real-time class. Here, not just the response of the system is fast (in order of microseconds) as well as too deterministic. So, it is possible to define a time range where, for sure, the delay of system reaction may be inserted. This is, indeed, the most appealing feature. But, obviously, it requires a considerable cost and hardware effort to achieve this level of performance [31]. That's why, it is necessary to define a level of commitment between application requirements and hardware complexity.

Natively, the *BeagleBone* OS is on bottom of hierarchy. There are, however, several ways to reduce system latency and achieve successive better levels of performance, regarding to a real-time point of view. On the following table, it is showed the latency inherent to each type of system.

Table A.1: *BeagleBone* system latency [31].

| System     | Typical instruction/interrupt cycle latency |
|------------|---------------------------------------------|
| Linux OS   | This specifies 1 second                     |
| PREEMPT_RT | This denotes 200 miliseconds                |
| Xenomai    | This denotes 20-50 miliseconds              |
| PRU        | This specifies 5-10 nanoseconds             |

The first two alternatives are both on software side. Respectively to *PREEMPT\_RT*, it consists on the installation of a patch over the kernel, changing the scaling process algorithm. By default,

*linux* has a non-preemptive mechanism where the execution of a process only ends when it is effectively ready, not being possible to interrupt its running cycle and allocate the CPU resources to other tasks. With a preemptive system, processes priorities are, periodically, evaluated and, if applicable, a low priority process might be interrupted and release the CPU to a higher priority one. This way, the real time performance is improved. However, to better results a *Xenomai* system can be considered. This is characterized by the existence of a co-kernel that is associated to the critical tasks and operates with a higher priority than the default one, which is related with the conventional processes.

Besides all that, to satisfy hard real-time needs, a PRU is the solution. This is a processing chip, located outside of main CPU, that can be compared to a pure microcontroller, except that it is onboard. For being devoid of a core kernel, it can avoid its slow cadency and achieve a significantly better performance. Since they are a separated hardware, these units need to be configured and properly programmed, task which might not be so easy to complex operations, requiring a setting period to allow user familiarization [31].

On current application, an installation of a preemptive system was promoted. For that, the next steps shall be executed:

```
root@beaglebone:~# cd /opt/scripts/tools
root@beaglebone:~# ./update_kernel.s --bone-rt-kernel --stable
root@beaglebone:~# reboot
```

These commands ensure the OS update to the most recent kernel version with a patch to preemptive processing.

## A.2 System latency measurement

To guarantee that the system fulfill application requirements, a measure of system latency must be performed. Several tools are susceptible to be used. The one recommended on [31] is the *cyclictest* framework. On a simple perspective, this program runs a thread which takes two snapshots at the begin and the end of a time interval. So, it is registered the start moment and, when the limit is reached, thread wakes up and catches that instant. The difference between the saved value and the theoretical one, obtained adding the configured range to the start moment, provides the kernel latency. Several cycles, depending of execution settings, are performed, being registered the minimum, average and maximum values.

To install the tool software package, it must be followed the procedure indicated on [31].

## Appendix B

# Shell script configuration

The interaction between user and operating system may rely on graphical user interface (GUI) or command line interface (CLI). This last one offers a low level interface where the commands are ordered through some specific 'words'. As example, there is the terminal window present in LINUX/UNIX systems. On a simple explanation, it is a textual interface that receives the inputs, processes it and returns an output. The commands interpretation is performed by a shell. Therefore, a shell is the main responsible for human-machine interface.

Normally, when on command line interface, this can be performed by one instruction at a time, over an interactive mode, or by providing several commands at once and, then, they are executed like a recipe, on batch mode. For reach that, a shell script needs to be implemented. The code includes the identification of shell (called shebang) followed by the commands to be executed. The created file has an `.sh` extension.

On current application, to avoid the need to open the terminal at every program execution, a script shell may be configured. Since there is only on executable file, the command that would be used on textual interface is written on the code. File content is just composed by the instructions below indicated:

```
root@beaglebone:~# nano script_name.sh
```

```
#!/bin/bash
sudo ./executable_name
```

The first line refers to interpreter, being used the bash for being the standard shell present on current LINUX/UNIX systems. The second one represents the command that executes the desired script. Since the linked file is the executable and not the source code, on posterior compilations, after shell script creation, it must be guaranteed that the executable name always matches the script and that they are on the same path. The execution may be done through two mouse clicks over the `.sh` icon.

Besides these aspects, it is also necessary to set the script permission. On this step, it is configured who can execute the file and the operations that they are allowed to do. This can be

done by a symbolical representation or an octal number, as detailed on [\[76\]](#). Here, it is suggested the following command on symbolical representation, that gives free execution to any user, but just the *root* may write on it:

```
root@beaglebone:~# chmod +x script_name.sh
```

## Appendix C

### Group I/O connections

Here, the system signals associated to group devices are all discriminated. For one side, there are the feedback signals provided by the physical equipment in order to evaluate its current condition and additional diagnosis information. These ones are the system inputs and are represented on table C.2. On another hand, device command signals constitute the system outputs and are detailed on table C.1.

Table C.1: Output commands to control group devices.

| I/O     | Type    | Description                  |
|---------|---------|------------------------------|
| Outputs | Digital | VVP actuator                 |
|         |         | VVD actuator                 |
|         |         | VVI actuator                 |
|         |         | VVT actuator                 |
|         |         | VVR1 actuator                |
|         |         | VVR2 actuator                |
|         |         | VPP turn on                  |
|         |         | VPP reset thermal protection |
|         |         | VPT turn on                  |
|         |         | VPT set standby              |
|         |         | VPT reset                    |
|         |         | Gauge power                  |
|         |         | Bakeout turn on              |
|         |         | Output 1 interlock           |
|         |         | Output 2 interlock           |
|         |         | Output 1 auxiliary device    |
|         |         | Output 2 auxiliary device    |

Table C.2: Input signals provided by group devices

| I/O    | Type    |              | Description              |
|--------|---------|--------------|--------------------------|
| Inputs | Analog  | Thermocouple | Bakeout temperature      |
|        |         | 0-10V        | Primary pump current     |
|        |         |              | Gauge pressure           |
|        | Digital | 0-24V        | VVP open                 |
|        |         |              | VVP closed               |
|        |         |              | VVD open                 |
|        |         |              | VVD closed               |
|        |         |              | VVI open                 |
|        |         |              | VVI closed               |
|        |         |              | VVT open                 |
|        |         |              | VVT closed               |
|        |         |              | VVR1 open                |
|        |         |              | VVR1 closed              |
|        |         |              | VVR2 open                |
|        |         |              | VVR2 closed              |
|        |         |              | VPP on state             |
|        |         |              | VPP cable connected      |
|        |         |              | VPP thermal protection   |
|        |         |              | VPP single/three-phase   |
|        |         |              | VPP phase detector state |
|        |         |              | VPT pump on              |
|        |         |              | VPT pump ok              |
|        |         |              | VPT standby              |
|        |         |              | VPT nominal speed        |
|        |         |              | Bakeout on               |
|        |         |              | Input 1 interlock        |
|        |         |              | Input 2 interlock        |
|        |         |              | Input 1 auxiliary device |
|        |         |              | Input 2 auxiliary device |
|        |         |              | UPS power cut            |
|        |         |              | UPS battery charging     |
|        |         |              | UPS alarm                |

## **Appendix D**

### **Group control process test**

Here, it is listed all the tests performed on system control procedure to evaluate its correct operation, from a functional point of view. So, a specific action is applied in order to force an error and the respective reaction is compared with the expected result, being indicated if it was or not well succeed. The cases analyzed are present on table [D.1](#).

Table D.1: Control process checklist.

| Mode                   | Group configuration  | Error       | Action                                                             | Result                                                                             | Test |
|------------------------|----------------------|-------------|--------------------------------------------------------------------|------------------------------------------------------------------------------------|------|
|                        |                      | <b>Code</b> | <b>Description</b>                                                 |                                                                                    |      |
|                        |                      | 108         | VPT not On Status after [#T_ErrVPTInOnTO] in Accelerating Step     | Goes to VENT_TURBO, Auto-Restart restarts error and pumping process                | Ok   |
|                        |                      | 109         | Accelerating (VPT) TimeOut [#T_ErrAcceTO]                          | Goes to VENT_TURBO, Auto-Restart restarts error and pumping process                | Ok   |
|                        |                      | 103         | Recover TimeOut [#T_ErrRecoverTO]                                  | Goes to VENT_ALL, Auto-Restart restarts error and pumping process                  | Ok   |
|                        |                      |             |                                                                    | Goes to VENT_ALL, Auto-Restart after power is back and restarts pumping process    | Ok   |
|                        |                      | 113         | Power Cut (running on UPS)                                         | Goes to STOP, Auto-Restart after power is back and restarts pumping process        | Ok   |
|                        |                      |             |                                                                    | Goes to STOP, Auto-Restart after power is back and restarts pumping process        | Ok   |
|                        |                      |             |                                                                    | Goes to VENT_ALL, Auto-Restart restarts error and pumping process                  | Ok   |
|                        |                      | 107         | VPT not On Status in Nominal or Recovering and Leak Detection Step | Goes to VENT_ALL, Auto-Restart restarts error and pumping process                  | Ok   |
|                        |                      |             |                                                                    | Goes to VENT_ALL, Auto-Restart restarts error and pumping process                  | Ok   |
|                        |                      |             |                                                                    | Goes to VENT_ALL, Auto-Restart restarts error and pumping process                  | Ok   |
|                        |                      |             |                                                                    | Goes to VENT_ALL, Auto-Restart restarts error and pumping process                  | Ok   |
|                        |                      | 106         | VVT or VVP or VVD not closed in Nominal or Recovering Step         | Goes to VENT_ALL, Auto-Restart restarts error and pumping process                  | Ok   |
|                        |                      |             |                                                                    | Goes to VENT_ALL, Auto-Restart restarts error and pumping process                  | Ok   |
|                        |                      |             |                                                                    | Goes to VENT_ALL, Auto-Restart restarts error and pumping process                  | Ok   |
|                        |                      | 104         | VVI not opened in PrimaryP, Accel, Nominal or Recovering Step      | Goes to VENT_ALL, Auto-Restart restarts error and pumping process                  | Ok   |
|                        |                      | 105         | VVP not pumping                                                    | Goes to VENT_ALL, Auto-Restart restarts error and pumping process                  | Ok   |
|                        |                      | 110         | VPT Error Status                                                   | Goes to VENT_ALL, Auto-Restart restarts process after error condition disappears   | Ok   |
| PUMP                   | Auto-Vent enabled    |             | Disconnected pump cable                                            | Goes to VENT_ALL, Auto-Restart restarts process after error condition disappears   | Ok   |
|                        | Auto-Restart enabled |             | Manually tripped thermal protection                                | Goes to VENT_ALL, Auto-Restart restarts process after error condition disappears   | Ok   |
|                        |                      |             | Disconnected pump cable                                            | Goes to VENT_ALL, Auto-Restart restarts process after error condition disappears   | Ok   |
|                        |                      | 114         | VPP Error Status                                                   | Goes to VENT_ALL, Auto-Restart restarts process after error condition disappears   | Ok   |
|                        |                      |             | Error forced by setting a higher current value than threshold      | Goes to VENT_ALL, Auto-Restart restarts process after error condition disappears   | Ok   |
|                        |                      |             | Set three-phase mode with single phase                             | Goes to VENT_ALL, Auto-Restart restarts process after error condition disappears   | Ok   |
|                        |                      |             | Error forced by setting a lower current value than threshold       | Goes to VENT_TURBO, Auto-Restart restarts process after error condition disappears | Ok   |
|                        |                      | 112         | VVI Error or Warning Status                                        | Goes to VENT_ALL, Auto-Restart restarts process after error condition disappears   | Ok   |
|                        |                      | 115         | VVP Error or Warning Status                                        | Goes to VENT_ALL, Auto-Restart restarts process after error condition disappears   | Ok   |
|                        |                      | 116         | VVT Error or Warning Status                                        | Goes to VENT_TURBO, Auto-Restart restarts process after error condition disappears | Ok   |
|                        |                      | 117         | VVD Error or Warning Status                                        | Goes to VENT_ALL, Auto-Restart restarts process after error condition disappears   | Ok   |
|                        |                      |             | Disconnected valve while PUMP - Primary Pumping                    | Pump process continues, VVP VVD and VVT are interlocked                            | Ok   |
|                        |                      |             | Disconnected valve while PUMP - Accelerating                       | Pump process continues, VVP VVD and VVT are interlocked                            | Ok   |
|                        |                      | 118         | VVR Error or Warning Status                                        | Pump process continues, VVP VVD and VVT are interlocked                            | Ok   |
|                        |                      |             | Disconnected valve while PUMP - Recovery                           | Pump process continues, VVP VVD and VVT are interlocked                            | Ok   |
|                        | Auto-Vent disabled   | -           | Any error                                                          | Goes to STOP mode, Auto-Restart still occurs                                       | Ok   |
|                        | Auto-Restart enabled | -           | Any error                                                          | Goes to proper handling mode and waits there until user order                      | Ok   |
| STOP                   | -                    | 118         | VVR Error Status                                                   | Goes to PUMP                                                                       | Ok   |
|                        | -                    | -           | Others                                                             | Remains on PUMP                                                                    | Ok   |
| LD and Large LD        | -                    | -           | Any error                                                          | Handling procedure equal to PUMP, but Auto-Restart is not allowed                  | Ok   |
| VENT ALL<br>VENT TURBO | -                    | -           | Any error                                                          | Stays on current mode                                                              | Ok   |

## **Appendix E**

### **PCB electric schematic**

On this appendix, it is included the electric schematic designed to PCB, namely interface board with physical group, being detailed the conditioning circuit related to each module.

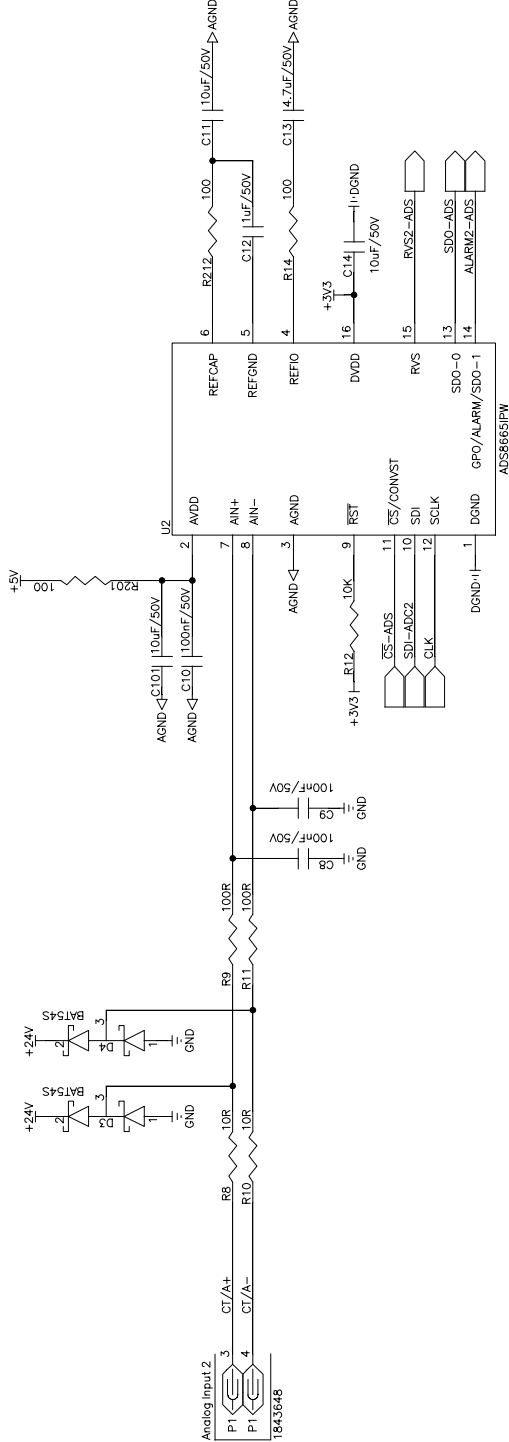


D

C

B

A



CENTRO DE CAD PARA ELECTRONICA – PORTO

Vpgrm V8 Electronic Card 2 ADC 2

PEDIDO: Tiago Mendonça DATA: 2018/05/15

EXECUTADO: Jose Carlos Azevedo DATA: 2018/06/17

PROJ. No: \_\_\_\_\_ REQ. No: \_\_\_\_\_ FOLHA: 2 de 12 VERSAO: 1.0

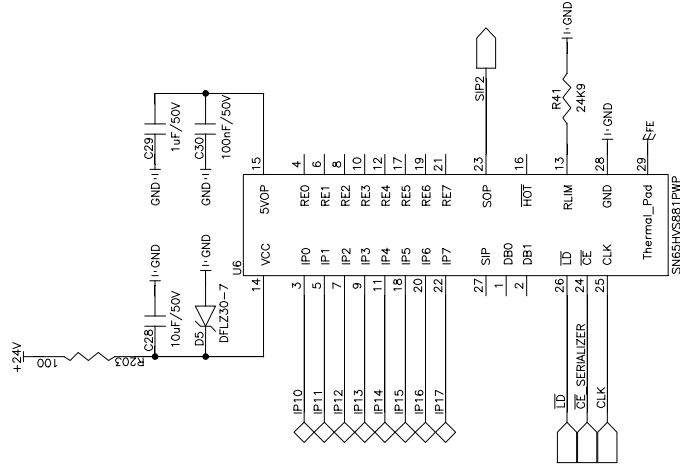
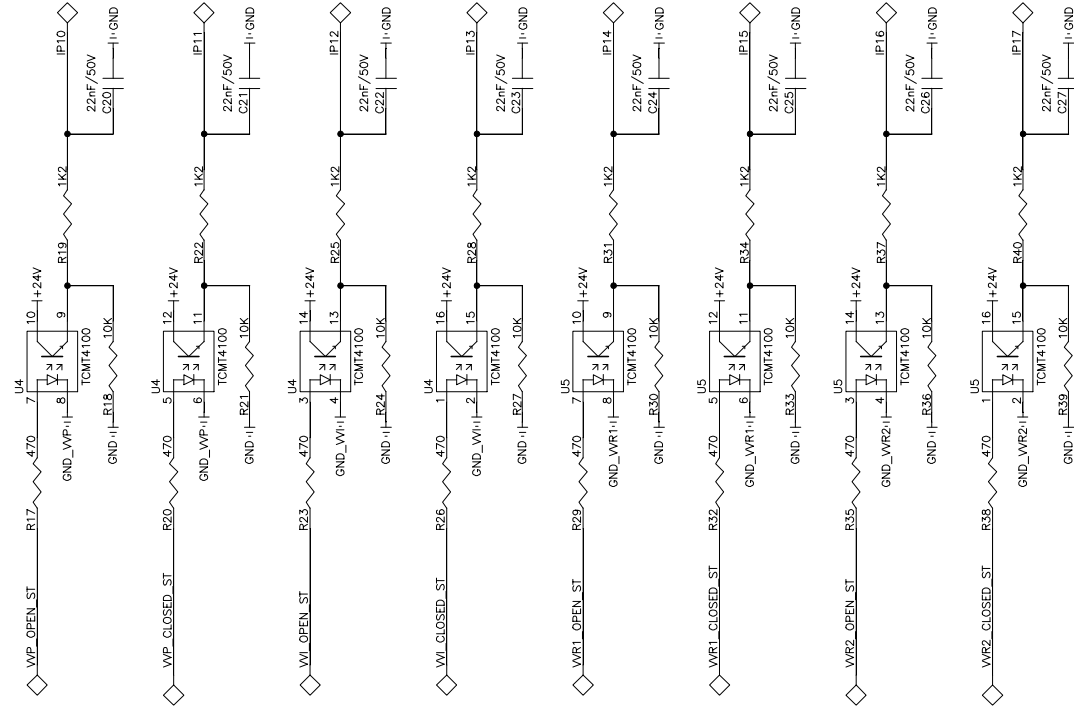
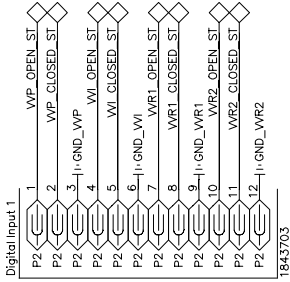
D

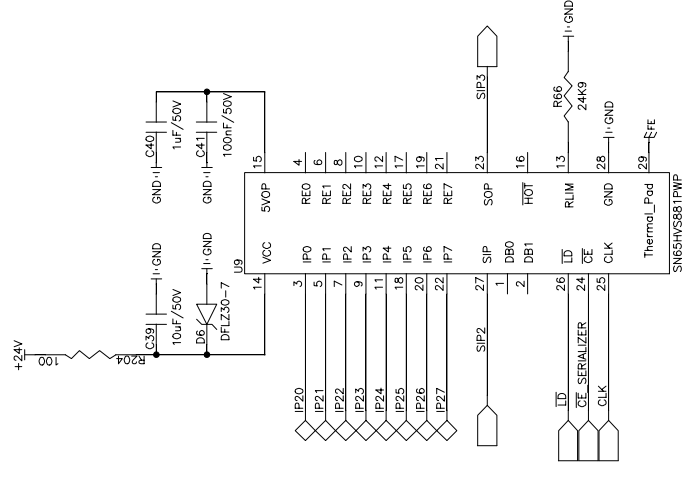
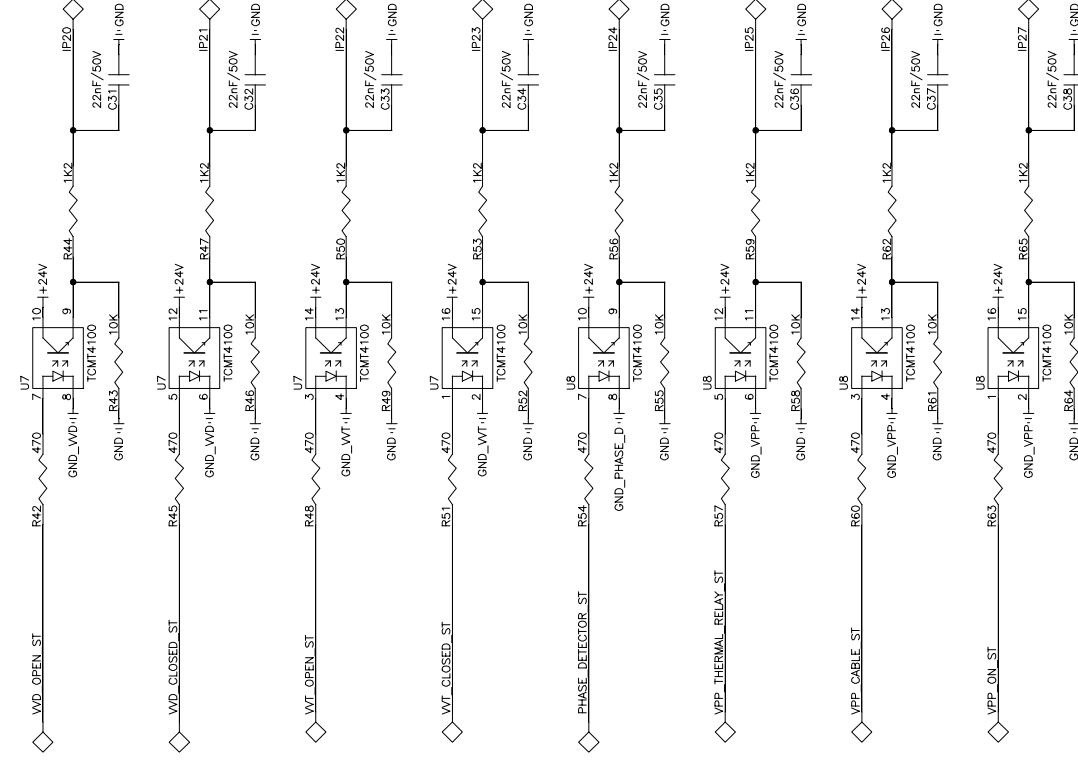
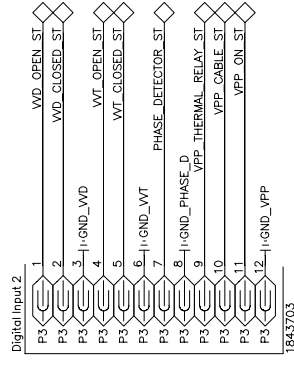
C

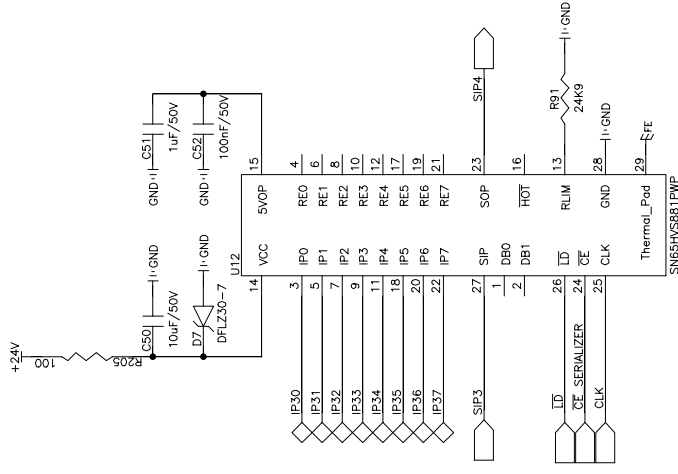
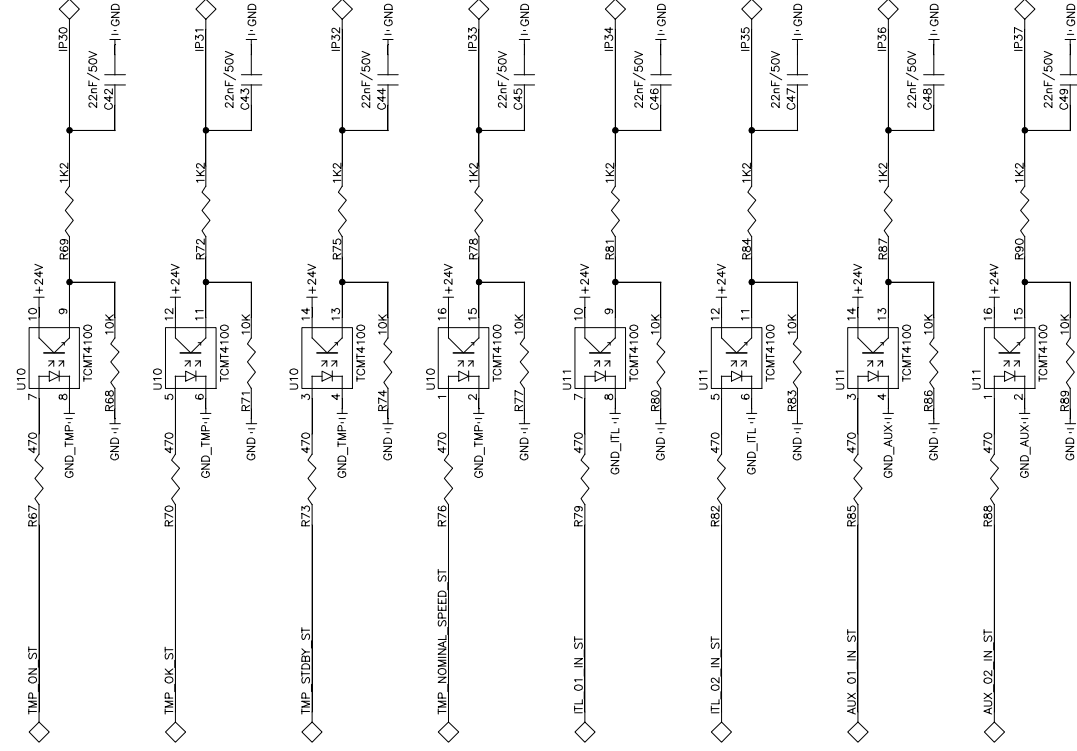
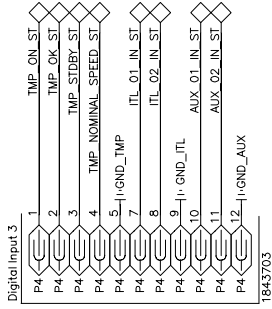
B

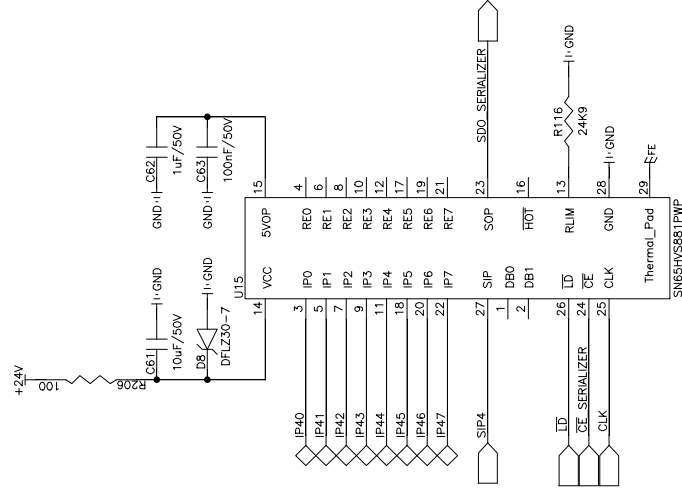
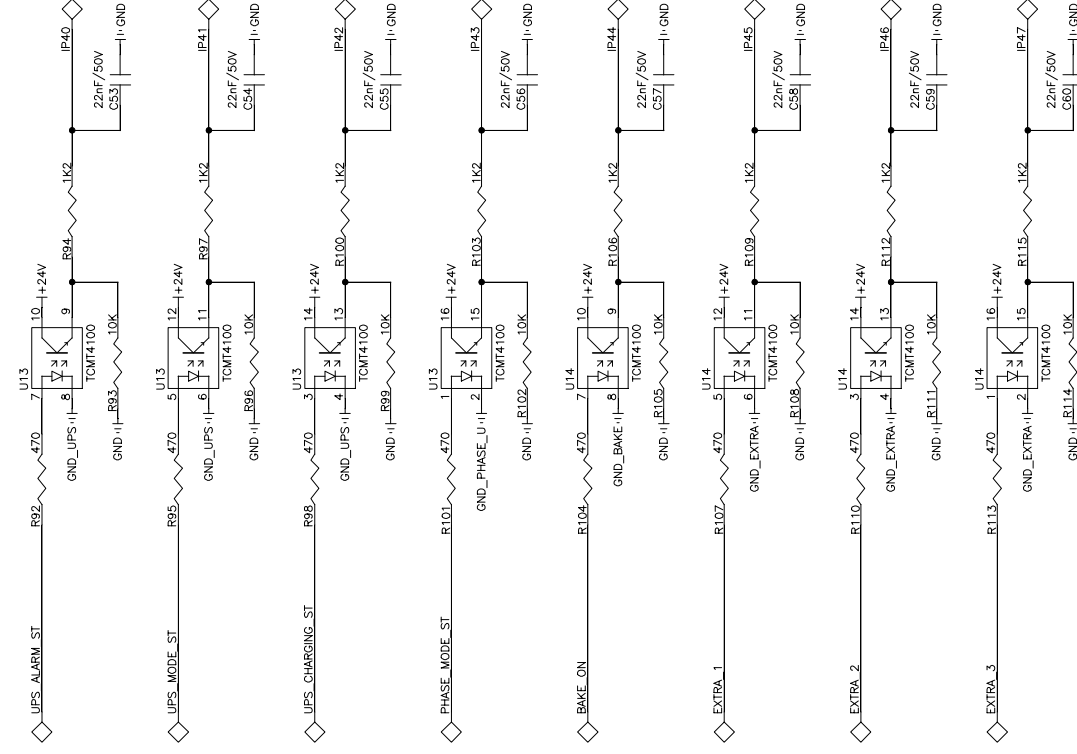
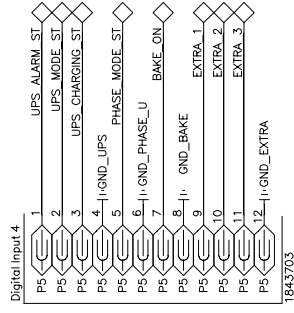
A

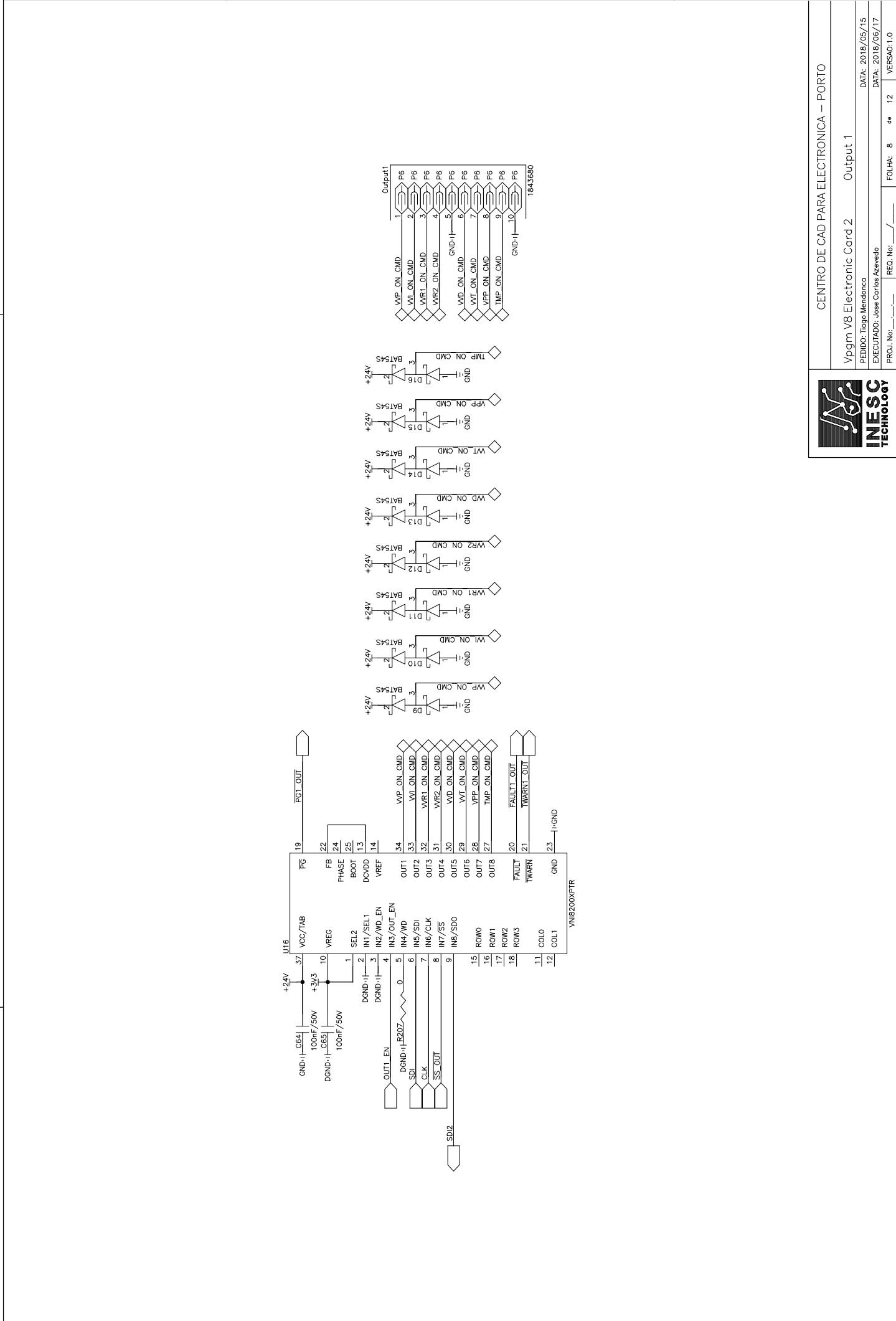






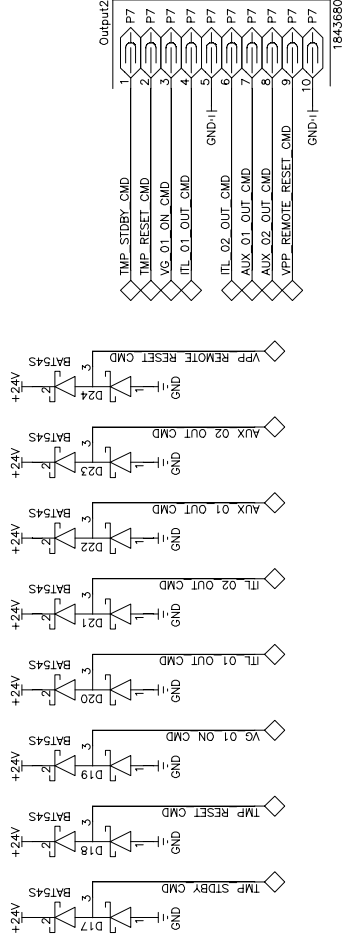
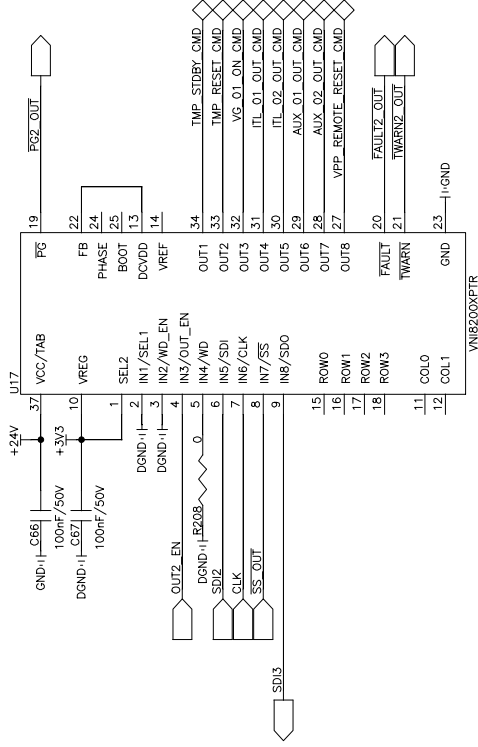






|                                                                                    |                |                                        |       |             |  |
|------------------------------------------------------------------------------------|----------------|----------------------------------------|-------|-------------|--|
|  |                | CENTRO DE CAD PARA ELECTRONICA – PORTO |       |             |  |
| Vpgrm V8 Electronic Card 2                                                         |                | Output 1                               |       |             |  |
| PEDIDO: Tiago Mendonça                                                             |                |                                        |       |             |  |
| EXECUTADO: Jose Carlos Azevedo                                                     |                |                                        |       |             |  |
| PROJ. No: _____                                                                    | REQ. No: _____ | FOLHA: 8                               | de 12 | VERSÃO: 1.0 |  |

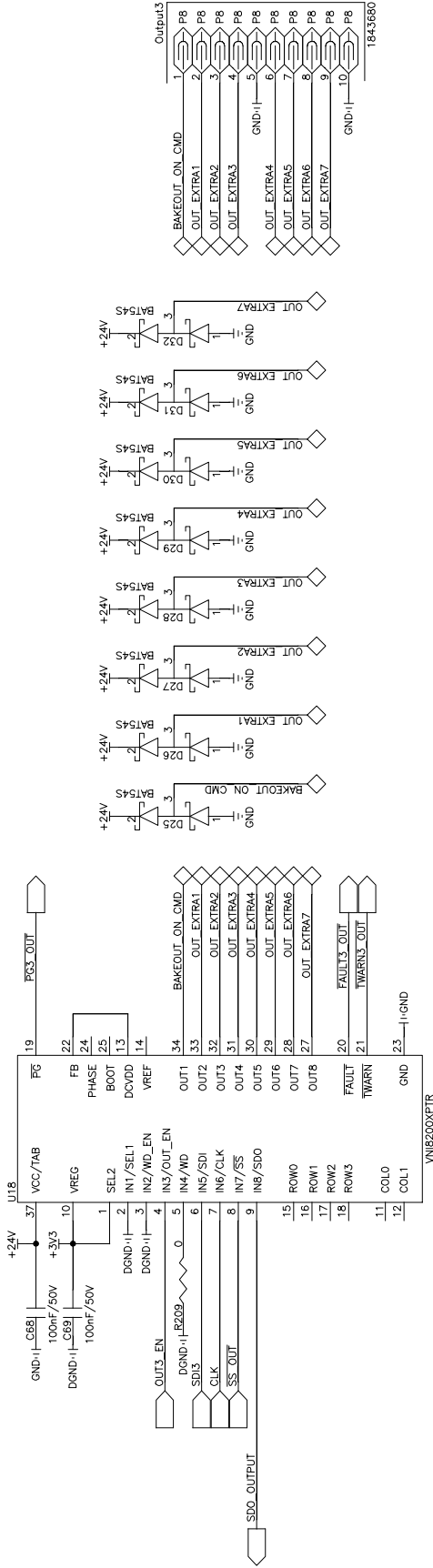




CENTRO DE CAD PARA ELECTRONICA – PORTO

Vpgrm V8 Electronic Card 2      Output 2

|                                |                |                  |       |
|--------------------------------|----------------|------------------|-------|
| PEDIDO: Tiago Mendonça         |                | DATA: 2018/05/15 |       |
| EXECUTADO: Jose Carlos Azevedo |                | DATA: 2018/06/17 |       |
| PROJ. No: _____                | REQ. No: _____ | FOLHA: 9         | d# 12 |
|                                |                | VERSAO:1.0       |       |



CENTRO DE CAD PARA ELECTRONICA – PORTO

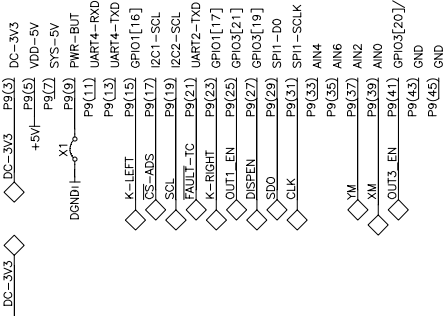
Vpgrm V8 Electronic Card 2 Output 3

PEDIDO: Tiago Mendonça DATA: 2018/05/15

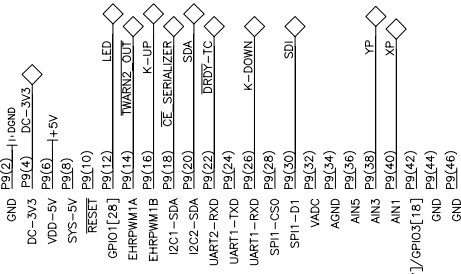
EXECUTADO: Jose Carlos Azevedo DATA: 2018/06/17

PROJ. No: \_\_\_\_\_ REQ. No: \_\_\_\_\_ FOLHA: 10 de 12 VERSAO:1.0

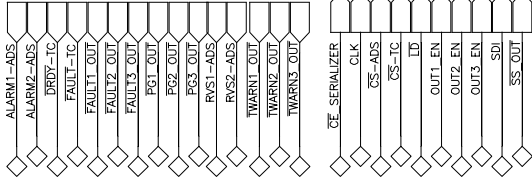
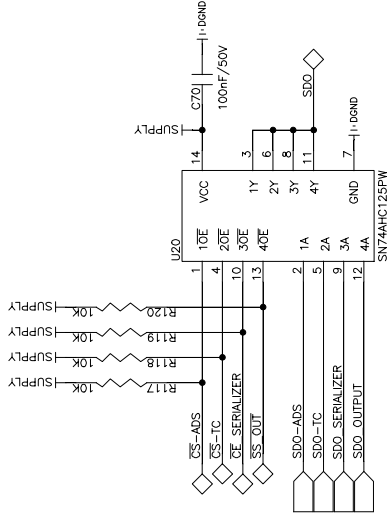
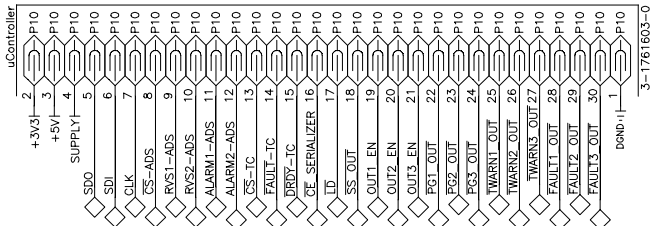
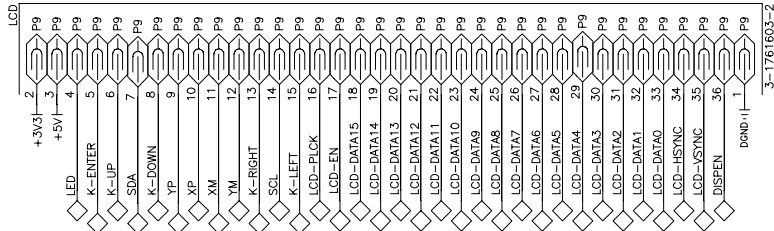
SUPPLY



U19



BeagleBoneBlack



CENTRO DE CAD PARA ELECTRONICA – PORTO

Vpgrm V8 Electronic Card 2 Beagle Bone Green / LCD

PEDIDO: Tiago Mendonça

EXECUTADO: Jose Carlos Azevedo

PROJ. No: \_\_\_\_\_

REQ. No: \_\_\_\_\_

FOLHA: 11

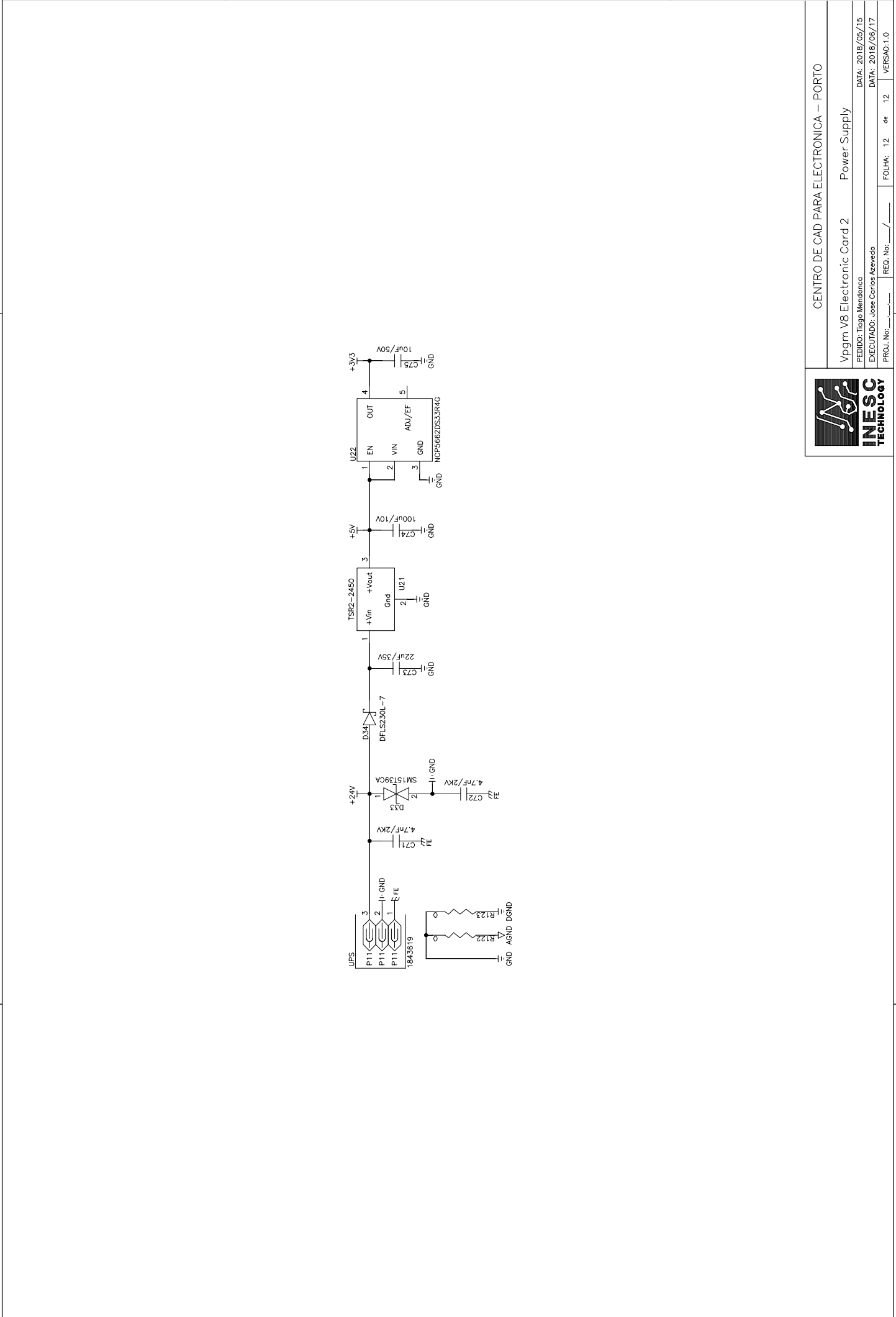
de

12

VERSÃO: 1.0

DATA: 2018/05/15

DATA: 2018/06/17



|                                                                                     |                |                                        |    |    |             |
|-------------------------------------------------------------------------------------|----------------|----------------------------------------|----|----|-------------|
|  |                | CENTRO DE CAD PARA ELECTRONICA – PORTO |    |    |             |
| Vpgrm V8 Electronic Card 2                                                          |                | Power Supply                           |    |    |             |
| PEDIDO: Tiago Mendonça                                                              |                |                                        |    |    |             |
| EXECUTADO: Jose Carlos Azevedo                                                      |                |                                        |    |    |             |
| PROJ. No: _____                                                                     | REQ. No: _____ | FOLHA: 12                              | de | 12 | VERSÃO: 1.0 |





# References

- [1] Particle accelerator. URL: <https://science.howstuffworks.com/atom-smasher4.htm>.
- [2] R. Ferreira, S. Blanchard, G. Pigny, H. Vestergard, and P. Gomes. Consolidation of the controls for turbo-molecular pumping groups at CERN's particle accelerator complex. In *2016 24th Mediterranean Conference on Control and Automation (MED)*, pages 961–966. IEEE, 6 2016. URL: <http://ieeexplore.ieee.org/document/7535968/>, doi: 10.1109/MED.2016.7535968.
- [3] Pfeiffer Vacuum GmbH. *The Vacuum Technology Book Volume II*. 2013. URL: [https://www.pfeiffer-vacuum.com/filepool/File/Vacuum-Technology-Book/Vacuum-Technology-Book-II-Part-2.pdf?referer=1456&request\\_locale=en\\_US](https://www.pfeiffer-vacuum.com/filepool/File/Vacuum-Technology-Book/Vacuum-Technology-Book-II-Part-2.pdf?referer=1456&request_locale=en_US).
- [4] José Miguel Jimenez. Vacuum Systems. URL: [https://indico.cern.ch/event/173359/contributions/276007/attachments/218832/306479/Vacuum\\_Systems\\_by\\_J.M.\\_Jimenez\\_-\\_CERN\\_v1.pdf](https://indico.cern.ch/event/173359/contributions/276007/attachments/218832/306479/Vacuum_Systems_by_J.M._Jimenez_-_CERN_v1.pdf).
- [5] S Blanchard, F Antoniotti, F Bellorini, JP Boivin, J Gama, P Gomes, H Pereira, G Pigny, B Rio, H Vestergard CERN, Switzerland L Kopylov, S Merker, and M Mikheev IHEP. Vacuum Pumping Group Controls Based On PLC. URL: <http://accelconf.web.cern.ch/AccelConf/PCaPAC2014/papers/wpo030.pdf>.
- [6] W. Bolton and W. Bolton. Programmable Logic Controllers. In *Programmable Logic Controllers*, pages 1–19. Elsevier, 2009. URL: <http://linkinghub.elsevier.com/retrieve/pii/B978185617751100001X>, doi:10.1016/B978-1-85617-751-1.00001-X.
- [7] K.L.S. Sharma and K.L.S. Sharma. Data Acquisition and Control Unit. In *Overview of Industrial Process Automation*, pages 125–149. Elsevier, 2017. URL: <http://linkinghub.elsevier.com/retrieve/pii/B9780128053546000098>, doi: 10.1016/B978-0-12-805354-6.00009-8.
- [8] Coder's Corner: PLCopen Standards Architecture; Data Typing. URL: <https://www.automation.com/library/articles-white-papers/programmable-control-plc-pac/coder146s-corner-plcopen-standards-architecture--data-typing>.
- [9] Programmable Logic Controller (PLC) - Wiki - odesie by Tech Transfer. URL: <https://www.myodesie.com/index.php/wiki/index/returnEntry/id/2962#The%20Scan%20Cycle>.

- [10] W. Bolton and W. Bolton. I/O Processing. In *Programmable Logic Controllers*, pages 79–113. Elsevier, 2015. URL: <http://linkinghub.elsevier.com/retrieve/pii/B9780128029299000042>, doi:10.1016/B978-0-12-802929-9.00004-2.
- [11] Trabalho CLP - Conceito sobre CLP. URL: <http://www.ebah.com.br/content/ABAAelZIAA/trabalho-sobre-clp?part=2>.
- [12] Serial Peripheral Interface (SPI). URL: <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi>.
- [13] Steve Mackay, Edwin Wright, Deon Reynders, John Park>, Steve Mackay, Edwin Wright, Deon Reynders, and John Park>. ProfiBus PA/DP/FMS overview. In *Practical Industrial Data Networks*, pages 181–199. Elsevier, 2004. URL: <http://linkinghub.elsevier.com/retrieve/pii/B9780750658072500376>, doi:10.1016/B978-075065807-2/50037-6.
- [14] Introduction To Profibus DP, 2002. URL: [https://www.acromag.com/sites/default/files/Acromag\\_Intro\\_ProfibusDP\\_698A.pdf](https://www.acromag.com/sites/default/files/Acromag_Intro_ProfibusDP_698A.pdf).
- [15] BeagleBoard - green. URL: <https://beagleboard.org/green>.
- [16] LCD Display Cape for Beaglebone Board. URL: <https://www.element14.com/community/docs/DOC-81966/1/43-lcd-display-cape-for-beaglebone-board-white-black-or-wireless>.
- [17] Texas Instruments. SN65HVS881 Datasheet. URL: <http://www.ti.com/lit/ds/symlink/sn65hvs881.pdf>.
- [18] ADS866x 12-Bit, High-Speed, Single-Supply, SAR ADC Data Acquisition System with Programmable, Bipolar Input Ranges. URL: <http://www.ti.com/lit/ds/symlink/ads8665.pdf>.
- [19] MAX31856 Datasheet. URL: <https://datasheets.maximintegrated.com/en/ds/MAX31856.pdf>.
- [20] Content Technologies. Octal high-side smart power solid-state relay with serial/parallel interface. 2015. URL: <http://www.st.com/content/ccc/resource/technical/document/datasheet/64/a2/80/0b/c0/c1/48/fd/CD00218854.pdf/files/CD00218854.pdf/jcr:content/translations/en.CD00218854.pdf>.
- [21] Mouser Portugal. URL: <https://pt.mouser.com/>.
- [22] Maxim Integrated. MAX485 Datasheet. URL: <https://datasheets.maximintegrated.com/en/ds/MAX1487-MAX491.pdf>.
- [23] PROFIBUS Installation Guidelines. 2016. URL: <http://verwertraining.com/wp-content/uploads/Installation-Guide-V13-small.pdf>.
- [24] CERN internal documentation.
- [25] PROFIBUS Protocol Overview. URL: <https://www.rtaautomation.com/technologies/profibus/>.
- [26] 7 inch 1024x600 Capacitive TouchScreen. URL: <https://www.seeedstudio.com/7-inch-1024x600-Capacitive-TouchScreen-p-2972.html>.

- [27] Action qualifiers for SFC elements. URL: <http://help.logicals.com/display/LC3UserDocuEN/Action+qualifiers+for+SFC+elements>.
- [28] Back to Basics: SPI (Serial Peripheral Interface). URL: <https://www.allaboutcircuits.com/technical-articles/spi-serial-peripheral-interface/>.
- [29] Raspberry Pi 3 Model B - Raspberry Pi. URL: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>.
- [30] SM 1223 digital input/output modules. URL: <https://mall.industry.siemens.com/mall/en/WW/Catalog/Product/6ES7223-1BH32-0XB0>.
- [31] Charles A. Hamilton. *BeagleBone Black Cookbook*. Packt, 2015.
- [32] Raymond L. Murray, Keith E. Holbert, Raymond L. Murray, and Keith E. Holbert. Particle accelerators. In *Nuclear Energy*, pages 123–137. Elsevier, 2015. URL: <http://linkinghub.elsevier.com/retrieve/pii/B9780124166547000095>, doi: 10.1016/B978-0-12-416654-7.00009-5.
- [33] Christine Sutton. Particle accelerator - Instrument, 1998. URL: <https://www.britannica.com/technology/particle-accelerator#ref255897>.
- [34] LHC Guide, 3 2017. URL: <https://cds.cern.ch/record/2255762>.
- [35] How an accelerator works, 1 2012. URL: <http://cds.cern.ch/record/1997220>.
- [36] Mexico Carlos Hernandez Garcia. Vacuum components. URL: [http://uspas.fnal.gov/materials/12UTA/Cathode\\_6.pdf](http://uspas.fnal.gov/materials/12UTA/Cathode_6.pdf).
- [37] Chad Folkman and Scott Rutherford. High and Ultra High Vacuum Systems. URL: <http://www.vacuumcursus.nl/pdf/highandultrahighvacuum.pdf>.
- [38] B.R. Mehta, Y.J. Reddy, B.R. Mehta, and Y.J. Reddy. The programmable logic controller. In *Industrial Process Automation Systems*, pages 37–74. Elsevier, 2015. URL: <http://linkinghub.elsevier.com/retrieve/pii/B9780128009390000024>, doi: 10.1016/B978-0-12-800939-0.00002-4.
- [39] PLCs and Other Logic Devices. In *Process Control*, pages 691–693. Elsevier, 1995. URL: <http://linkinghub.elsevier.com/retrieve/pii/B9780750622554500805>, doi: 10.1016/B978-0-7506-2255-4.50080-5.
- [40] IEC 61131 Standards. URL: [http://www.plcopen.org/pages/tcl\\_standards/](http://www.plcopen.org/pages/tcl_standards/).
- [41] Automation Basics: Programmable logic controllers: Hardware, software architecture - ISA. URL: <https://www.isa.org/standards-publications/isa-publications/intech-magazine/2010/december/automation-basics-programmable-logic-controllers-hardware-software-architect>.
- [42] Armando Sousa and José Faria. GRAFCET - UC sistemas e Automação, 2017.
- [43] Industry Automation and Drive Technologies - SCE SEQUENTIAL FUNCTION CHART. URL: [https://w3.siemens.com/mcms/sce/de/fortbildungen/ausbildungsunterlagen/hochschul-module/tabcardseiten/Documents/V7.0/eP01-07\\_sequential\\_function\\_chart\\_RC1012.pdf](https://w3.siemens.com/mcms/sce/de/fortbildungen/ausbildungsunterlagen/hochschul-module/tabcardseiten/Documents/V7.0/eP01-07_sequential_function_chart_RC1012.pdf).

- [44] Sequential Function Charts for All. URL: [http://www.plcdev.com/sequential\\_function\\_charts\\_all](http://www.plcdev.com/sequential_function_charts_all).
- [45] B.R. Mehta, Y.J. Reddy, B.R. Mehta, and Y.J. Reddy. PROFIBUS communication. In *Industrial Process Automation Systems*, pages 381–399. Elsevier, 2015. URL: <http://linkinghub.elsevier.com/retrieve/pii/B9780128009390000127>, doi: 10.1016/B978-0-12-800939-0.00012-7.
- [46] Tran Duy Khanh, Pavel Pisa, and Petr Smolik. An Open Implementation of Profibus DP. URL: <http://www.pbmaster.org>.
- [47] PROFIBUS on Sitara Processors. URL: <http://www.ti.com/lit/wp/spry155b/spry155b.pdf>.
- [48] PROFIBUS and PROFINET ASICs - Procentec. URL: <https://procentec.com/content/ordercodes-asics/>.
- [49] Pavel Trnka and Petr Smolík. Profibus DP Master for PC. URL: <http://profim.sourceforge.net/download/profim-doc.pdf>.
- [50] Michael Busch. PROFIBUS software stack. URL: [https://bues.ch/cms/automation/profibus.html#license\\_copyright](https://bues.ch/cms/automation/profibus.html#license_copyright).
- [51] Wenjun Zhou. Soft PLC Research And Development System Based On PC. In *Proceedings of the 2015 International Conference on Intelligent Systems Research and Mechatronics Engineering*, Paris, France, 2015. Atlantis Press. URL: <http://www.atlantis-press.com/php/paper-details.php?id=18540>, doi:10.2991/isrme-15.2015.352.
- [52] Quan Liang and Li Li. The Study of Soft PLC Running System. *Procedia Engineering*, 15:1234–1238, 1 2011. URL: <https://www.sciencedirect.com/science/article/pii/S1877705811017292>, doi:10.1016/J.PROENG.2011.08.228.
- [53] Fang Ding and Suzhuo Wu. The Research about Embedded Soft PLC Running System Based on ARM. *Sensors & Transducers*, 179(9):55–59, 2014.
- [54] CODESYS: The benefits for end-users and device manufacturers. URL: <https://www.codesys.com/the-system/why-codesys.html>.
- [55] EL6731 | PROFIBUS master/slave terminal. URL: <https://www.beckhoff.com/english.asp?ethercat/el6731.htm>.
- [56] CODESYS Profibus Master. URL: [https://store.codesys.com/codesys-profibus-master-el-6731-sl.html?\\_\\_store=en&\\_\\_from\\_store=default](https://store.codesys.com/codesys-profibus-master-el-6731-sl.html?__store=en&__from_store=default).
- [57] CODESYS Profibus Slave. URL: [https://store.codesys.com/codesys-profibus-slave-el-6731-0010-sl.html?\\_\\_store=en&\\_\\_from\\_store=default](https://store.codesys.com/codesys-profibus-slave-el-6731-0010-sl.html?__store=en&__from_store=default).
- [58] TwinCAT - PLC and Motion Control on the PC. URL: <http://www.beckhoff.com/twincat/>.
- [59] Beremiz. URL: <https://beremiz.org/>.

- [60] A. Valera, A. Soriano, and M. Vallés. Plataformas de Bajo Coste para la Realización de Trabajos Prácticos de Mecatrónica y Robótica. *Revista Iberoamericana de Automática e Informática Industrial RIAI*, 11(4):363–376, 10 2014. URL: <https://www.sciencedirect.com/science/article/pii/S1697791214000557>, doi: 10.1016/J.RIAI.2014.09.002.
- [61] CODESYS Control for BeagleBone. URL: <https://store.codesys.com/codesys-control-for-beaglebone-sl.html>.
- [62] CODESYS Control for Raspberry Pi. URL: <https://store.codesys.com/codesys-control-for-raspberry-pi-sl.html>.
- [63] AM335x Sitara™ Processors Silicon. 2011. URL: <http://www.ti.com/lit/er/sprz360i/sprz360i.pdf>.
- [64] BeagleBone Black Enable SPIDEV. URL: [https://elinux.org/BeagleBone\\_Black\\_Enable\\_SPIDEV#SPI0](https://elinux.org/BeagleBone_Black_Enable_SPIDEV#SPI0).
- [65] Libsoc Library. URL: <https://github.com/jackmitch/libsoc>.
- [66] TCMT 4100 Datasheet. URL: [www.vishay.com](http://www.vishay.com).
- [67] CPU 1214C. URL: <https://mall.industry.siemens.com/mall/pt/pt/Catalog/Product/6ES7214-1AG40-0XB0>.
- [68] Protecting Inputs in Digital Electronics | DigiKey. URL: <https://www.digikey.com/en/articles/techzone/2012/apr/protecting-inputs-in-digital-electronics>.
- [69] SM 1231 thermocouple module. URL: <https://mall.industry.siemens.com/mall/en/WW/Catalog/Product/6ES7231-5QD32-0XB0>.
- [70] CH340 datasheet. URL: <http://wch.cn>.
- [71] Texas Instruments. Polarity Conventions for RS-485 Transceivers. Technical report, 2003.
- [72] CH341 Parity Fix Patch. URL: <https://github.com/justinasjaronis/linux/commit/0866ccda19db50ec6886eb4e4425bf3293555e35>.
- [73] BeagleBoard 4.4 kernel CH341 driver. URL: <https://github.com/beagleboard/linux/blob/4.4/drivers/usb/serial/ch341.c>.
- [74] CH341 driver. URL: [http://www.wch.cn/download/CH341SER\\_ZIP.html](http://www.wch.cn/download/CH341SER_ZIP.html).
- [75] shm\_open - Linux manual page. URL: [http://man7.org/linux/man-pages/man3/shm\\_open.3.html](http://man7.org/linux/man-pages/man3/shm_open.3.html).
- [76] Learning the shell - Lesson 9: Permissions. URL: [http://linuxcommand.org/lc3\\_lts0090.php](http://linuxcommand.org/lc3_lts0090.php).