UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO FÍSICA

Low Noise Power Supplies for the High Voltage Board of the TILECAL Calorimeter

Rui Paulo Serrano Fernandez

Mestrado Integrado em Engenharia Física

Dissertação orientada por:
Doutora Guiomar Gaspar de Andrade Evans

2019

Acknowledgments

Firstly, I want to thank my family for their understanding and support over the last five years, especially my parents who were always present when I needed them and my sister who brought me to places I didn't know, which allowed me forget a little of the pressure the university brings. To my classmates, who made the course not only a place to learn but to grow as a person. To Professor Guiomar Evans for giving me the opportunity to participate in this spectacular project, which allowed me to learn much more than I had expected, for supporting me since then, always available to help and for being my advisor.

I also want to thank Professor José Augusto for being always available to answer my questions. To Luis Gurriana who helped me a lot throughout the project in using a program I didn't know. To all the teachers who over the past 5 years have given me all the information I needed to be able to complete this thesis and be ready for the next step, the world of work. And to the University of Lisbon for providing a favourable environment for my development, both as a person and a student.

I want to leave a special thank you to my closest group of friends, who made my college life more bearable and fun and helped me when I needed it most, especially Inês Albuquerque who was present since the beginning of my academic life. She was always there to offer me help even when she needed it too, and in recent months especially where we both had a difficult time due to the thesis, but the fact that she was present made me believe that I could finish the course and push me to continue even when I no longer wanted to.

Finally, I want to thank all the people I could not refer to but who have been by my side for these 5 long years and have been my moral support. The university is not easy but it is still worth it and especially if we are lucky enough to know and accompany people who make their mark in our life, without them none of this would be possible.

A huge and sincere thank you to all of you from the deepest of my heart.

Resumo

O sistema atual de distribuição de alta tensão do calorímetro hadrónico central da experiência ATLAS do CERN, TileCal, foi fabricado no final dos anos 90. Este foi projetado para estar em funcionamento durante 10 anos, no entanto já se encontra em funcionamento há cerca de 20 anos. Atualmente, muitos dos componentes utilizados encontram-se obsoletos o que impossibilita a sua reparação e reutilização. Por outro lado, o sistema atual encontra-se no interior da caverna ATLAS, logo encontra-se exposto a altos níveis de radiação. Esta exposição contínua a altos níveis de radiação resultantes das colisões entre os feixes de partículas, que ocorrem no LHC (Large Hadron Collider), afeta todo o sistema. O facto de o sistema se encontrar na caverna suscita ainda outros problemas, tais como, a dificuldade de reparar ou mesmo substituir qualquer componente ou placa constituinte do sistema eletrónico, danificado pela radiação ou devido ao envelhecimento eletrónico. Para se efetuar a reparação ou a substituição de componentes ou placas é necessário que o LHC pare o seu funcionamento durante alguns meses, de modo a que os níveis de radiação diminuam o suficiente para permitir que um técnico possa entrar na caverna, porém esta pausa de meses só oferece um tempo muito limitado para executar esta tarefa.

Para além destes problemas tem-se ainda como um dos objetivos o aumento da luminosidade do LHC, o que vai implicar um aumento do nível de radiação na caverna ATLAS. Outro dos objetivos, é a diminuição do intervalo de tempo entre as colisões de partículas, levando à necessidade de eletrónica mais rápida. Todos estes problemas e novos objetivos fazem com que seja necessário atualizar e/ou modificar toda a electrónica do TileCal.

De forma a superar estes problemas, foi proposta uma atualização: um novo sistema de distribuição de alta tensão (HVDS) será colocado fora da caverna onde se encontra o detetor, passando este a ser um sistema remoto que não é afetado pela radiação, maximizando assim a fiabilidade e a robustez do sistema. A eletrónica deste novo sistema será colocada numa sala sem radiação, localizada 100 metros acima da caverna ATLAS, o que permitirá o acesso permanente ao sistema de distribuição de altas tensões. Assim, deixa de ser necessária a existência de uma paragem do funcionamento do LHC para executar reparações no sistema. Outra vantagem inerente ao sistema remoto é deixar de haver uma limitação de tempo disponível para realizar as reparações e/ou substituições, diminuindo também o risco a que o técnico está sujeito quando as executa. Para este novo sistema é necessário produzir uma placa dedicada que forneça as alimentações primárias necessárias, alta e baixa tensão, dado que no sistema atual as fontes de alimentação primárias de baixa tensão encontram-se na caverna ATLAS e as de alta tensão embora se encontrem na sala sem radiação já referida são fontes lineares de elevado custo.

O trabalho apresentado nesta dissertação insere-se na colaboração portuguesa no projeto ATLAS/CERN. Este consiste no desenvolvimento de uma placa de alimentação, designada por Power Supplies, capaz de fornecer tanto a alta tensão (HV), -830 V a -950 V, como as baixas tensões ,± 12 V e $3,3$ V, sendo imperativo que todas as tensões produzidas tenham baixo ruído. Para produzir estas tensões recorreu-se à utilização de quatro conversores DC/DC, sendo que

dois dos conversores DC/DC são utilizados para produzir a alta tensão, -830 V a -950 V @ 10 mA, e os restantes dois para as baixas tensões, um para os 3.3 V @ 0.8 A, e o outro que é um conversor DC/DC duplo para os ± 12 @ 2.5 A. Cada HVDS fornece a alimentação para 48 fotomultiplicadores (PMTs) do detetor. Devido à corrente necessária para alimentar todos os PMTs, é necessário recorrer ao uso de dois conversores DC/DC para produzir a alta tensão.

Os valores da tensão de saída dos conversores DC/DC de alta tensão são controlados digitalmente, podendo fornecer dois valores diferentes, -830 V ou -950 V. Estes valores de tensão distintos permitem que cada PMT do detetor possa receber a tensão adequada para funcionar corretamente. Dado que não existe um único componente que seja igual a outro, cada PMT terá as suas características próprias e, portanto, a sua tensão de alimentação deve ser ajustada para se obter o melhor desempenho do detetor. Estes dois valores de tensão permitem a correta calibração de todos os PMTs efetuada pelo sistema de distribuição das altas tensões.

A placa que fornecerá as alimentações ao HVDS, deverá ainda oferecer algumas funcionalidades extra, tais como: a possibilidade de uma monitorização em tempo real do consumo em tensão e corrente de cada conversor, a leitura da temperatura em dois pontos diferentes da placa, a capacidade de ligar/desligar digitalmente cada um conversores DC/DC individualmente e ligar/desligar manualmente todos os conversores DC/DC ao mesmo tempo, através de um interruptor. Este último servirá como medida de segurança caso o método digital não funcione ou em caso de substituição ou manutenção do sistema sem necessidade de recorrer ao sistema de controlo digital do ATLAS.

O controlo disgital da placa Power Supplies será baseado num protocolo de comunicação SPI e num expansor série/paralelo. O sinais de saída do referido expansor serão os sinais para ligar/desligar os conversores, os sinais de seleção de tensão de saída dos conversores de alta tensão e os sinais que permitem a leitura adequada e em tempo real dos consumos de tensão e de corrente e dos sensores de temperatura utilizados. Estas leituras são efetuadas recorrendo ao controlo digital de um multiplexador analógico e a um conversor analógico digital (ADC).

Ainda no âmbito desta tese, á apresentada a interface gráfica de utilizador (GUI) desenvolvida na linguagem de programação Python. Esta foi utilizada para facilitar a comunicação entre a placa Power Supplies e o utilizador. A interface gráfica está dividida em três secções diferentes, de forma a ser mais intuitiva para o utilizador. A primeira secção é a secção responsável por ligar/desligar os conversores DC/DC, sendo que esta apresenta quatro caixas de seleção, uma para cada conversor, que quando selecionadas pelo utilizador, executam o código responsável por enviar a instrução ao expansor para enviar o sinal de ligar/desligar para os conversores DC/DC selecionados. Na segunda secção encontram-se representadas duas barras deslizantes, às quais se encontra associado um cursor que se pode deslocar entre duas posições distintas, associadas à selecção da tensão de saída de cada um dos conversores DC/DC de alta tensão. Associada à posição do cursor encontra-se também um texto informativo que permite que o utilizador verifique se a tensão seleccionada é a pretendida.

A terceira e última secção é a da leitura dos consumos de tensão e de corrente assim como das duas temperaturas lidas por dois sensores de tempertura colocados em pontos disntintos da carta. Esta leitura pode ser feita de duas formas diferentes, pode ser feita uma única medida através da seleção de botões dedicados que apenas permitem selecionar uma opção de cada vez, sendo o resultado da leitura apresentado em duas caixas. A primeira caixa com a leitura em contagens do ADC, que é o valor que o ADC fornece diretamente, e a segunda caixa com a leitura do valor correspondente ao que se está efetivamente a medir com a respetiva unidade fisica. A outra forma envolve um conjunto várias medições continuas de uma das grandezas anteriormente referidas, sendo que o utilizador pode escolher o número de medições pretendidas e o intervalo de tempo entre cada medida. Os valores lidos/medidos através deste método são apresentados em gráficos diferentes em função do tempo atualizados em tempo real, sendo possível guardar estes dados num ficheiro do tipo *csv*.

O trabalho desta dissertação consistiu no desenvolvimento de uma placa que irá fornecer as alimentações primárias necessárias para o novo sistema de distribuição de alta tensão, e no desenvolvimento da interface gráfica de utilizador dedicada para esta placa que permitirá o seu teste funcional e que será mais tarde migrada para o teste de controlo digital do ATLAS.

Palavras-chave: CERN, ATLAS, Fonte de Alimentação, Baixo Ruído, TileCal, tubos fotomultiplicadores (PMTs), interface série (SPI), conversor analógico-digital (ADC).

Abstract

The current system that distributes high voltage to the hadronic calorimeter TileCal of the ATLAS experiment at CERN was manufactured in the late 1990s and now many of its components are obsolete. In addition to this, the continuous exposition to high levels of radiation that results from the LHC collision affects the whole system. The calorimeter itself will be upgraded and a faster and low noise electronic will be needed. Given this, an update was proposed to mitigate these problems: a new high voltage distribution system (HVDS) placed outside of the detector, a remote system which will not be affected by the radiation, that maximize the reliability and robustness of the system. For this new system it is necessary to produce a dedicated board that provides the necessary primary supplies. Therefore, the presented work consists in the development of a power supply board capable of providing both high voltage (HV), -830 V and -950 V, and low voltage, ± 12 V and 3.3 V, with low noise, resorting to DC/DC converters. Each HVDS provides the supply to 48 photomultipliers tubes (PMTs) of the detector. Due to the current needed, two high voltage sources are available, each one to supply just half of the PMTs. The values of the provided HV supplies are digitally controlled to one of the referred values, so each PMT of the detector can receive the right voltage to work correctly. Besides that, this board is controlled by a serial peripheral interface (SPI) communication protocol and has an analog to digital converter (ADC) and an analog multiplexer that are used to provide the user monitoring of all supply voltages and currents in real-time as well as the temperature, in two different positions of the board, in real time. A graphical user interface (GUI) has also been developed which allows easy communication between the power supply board and the user.

Keywords: CERN, ATLAS, Power Supply, Low Noise, TileCal, PMTs (acronym of photomultipliers tubes), SPI (acronym of serial peripheral interface), ADC (acronym of analog to digital converter).

Contents

List of Figures

List of Tables

Nomenclature

ADCs *Analog to Digital Converters*

AGND *Analog Ground*

ALICE *A Large Ion Collider*

ATLAS *A Toroidal LHC ApparatuS*

BJT *Bipolar Junction Transistors*

CERN *Conseil Européen pour la Recherche Nucléaire (In French)*

CMS *Compact Muon Solenoid*

CPHA *Clock Phase*

CPOL *Clock Polarity*

CS *Chip Select*

CSV *Comma-Separated Values*

DACs *Digital to Analog Converters*

DAQ *Data Acquisition System of the ATLAS Detector*

DCS *Detector Control Systems*

DGND *Digital Ground*

DNL *Differential Nonlinearity Error*

EM *Electromagnetic*

EMI *Electromagnetic Interference*

FPGA *Field Programmable Gate Array*

FSR *Full Scale Range*

GND *Ground*

GPIO *General Purpose Input/Output*

GUI *Graphical User Interface*

HV *High Voltage*

HVDS *High Voltage Distribution System*

INL *Integral Nonlinearity Error*

JFET *Junction Field Effect Transistor*

LHC *Large Hadron Collider*

LHCb *Large Hadron Collider beauty*

LSB *Least Significant Bit*

MISO *Master Input Slave Output*

MOSI *Master Output Slave Input*

MUX *Multiplexer*

NC *Not Connected*

PCB *Printed Circuit Board*

PMT *Photomultipliers tubes*

RF *Radio Frequency*

ROD *Read Out Driver*

SCLK *Clock*

SPI *Serial Peripheral Interface*

SS *Slave Select*

TileCal *Tile Calorimeter*

UI *User Interface*

VME *Versa Module Eurocards*

WLS *Wavelength Shifting*

Chapter 1

Introduction

The Large Hadron Collider (LHC) is the most powerful and largest particle accelerator in the world. The LHC consists of several superconducting magnets with various accelerator structures to increase particle energy along a 27-kilometre ring, which lies between 50 m and 175 m from the earth's surface [1].

Inside the accelerator are contained two beams of high energy particles that travel at a speed close to the speed of light before they are forced to collide. The beams travel in opposite directions in separate beam pipes, that are kept at ultrahigh vacuum produced by the vacuum system. This system ensures that the pressure in these pipes is in the order of 10^{-10} to 10^{-11} mbar [2]. A strong magnetic field is created and maintained by superconducting electromagnets, which is responsible for guiding these two particle beams along the accelerator ring. The electromagnets are built from coils of special electric cable that operates in a superconducting state, which means it requires chilling the magnets to 1.9 K (-271.3 ^{o}C), resorting to liquid helium, to cool the magnets, as well as other supply services.

The entire LHC is controlled by the CERN Control Center, where collisions between beams are also controlled to occur at four specific locations around the accelerator ring, corresponding to the positions of four particle detectors - ATLAS, CMS, ALICE and LHCb [3].

The main focus of this thesis is the development of a power supply board for the new HV remote distribution system for the ATLAS experiment. However, it is necessary to mention why the need for a new system and for a dedicated power supply board. This will be covered in this chapter, where a short presentation of the system that is currently implemented in CERN's ATLAS experiment and the reasons that led to the need for its upgrade, mentioning some of the most important aspects of the new system that is under development.

1.1 ATLAS Experiment

The LHC has several detectors, two of which are general purpose detectors, one of these two general purpose detectors being ATLAS. The main purpose of this detector covers a very wide range of physics, from the Higgs boson search to extra dimensions and particles that can make up dark matter. While having the same scientific goals as the CMS experiment, ATLAS uses

different technical solutions, a different magnet system design, and different types of detectors so that data is redundant, to ensure that when something is detected by both detectors, is in fact something relevant and not a mere casualty or malfunction of the detector [4]. ATLAS uses an advanced trigger system to tell the detector which events to record and which to ignore so this enormous flow of data can be processed.

In ATLAS there are six different detecting subsystems arranged in layers that record several data when beams of particles from the LHC collide at the center of ATLAS. The most important data that are record are the paths, momentum, and energy of the particles, allowing them to be individually identified, and with a help of a huge magnet system that bends the paths of charged particles it is possible to measure the momenta of these particles [4].

The TileCal is the central hadronic calorimeter of the ATLAS experiment, a cylindrical hadronic sampling detector with steel absorbent and scintillating plastic tiles, surrounding the electromagnetic calorimeter cryostat. It is located about 100 meters deep, all of its electronics are contained in the calorimeter itself and is built in three sections: a 6-meter-long central barrel detector divided into two partitions, within which collisions occur, and two 3-meter-long extended barrel detectors, as can be observed in the figure 1.1. Each section is divided, azimuthally into 64 modules [5].

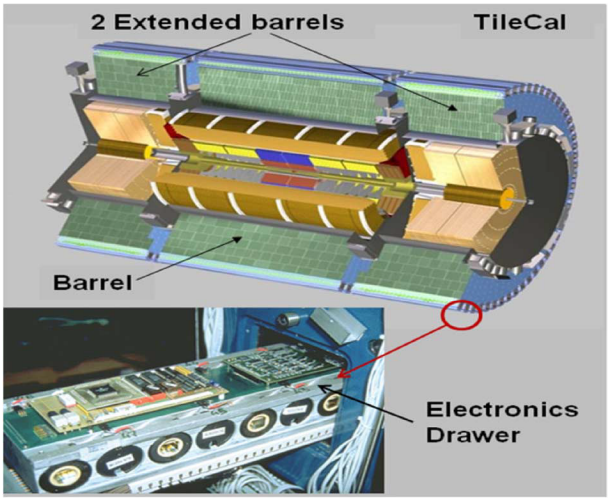


Figure 1.1: TileCal hadronic calorimeter structure [5].

The scintillating tiles are placed in the plane perpendicular to the colliding beams and are radially staggered in depth, as illustrated in figure 1.2. The scintillating tiles are read out by wavelength shifting (WLS) optical fibers that collect and deliver the light to the PMTs located in the outer radius iron structure that also houses the front-end electronics. Each cell is read out from both sides independently, resulting in a total of 9856 readout channels, allowing for redundancy in information gathering and increasing spatial uniformity. Adjacent tiles, together with the optical fibers, are grouped together to form TileCal cells. The analog signal from each PMT is processed by electronics at the detector end (in a module drawer, figure 1.1), which is responsible for signal conditioning and amplification, providing three analog output signals: two for the read out detector and one for the trigger [6][7].

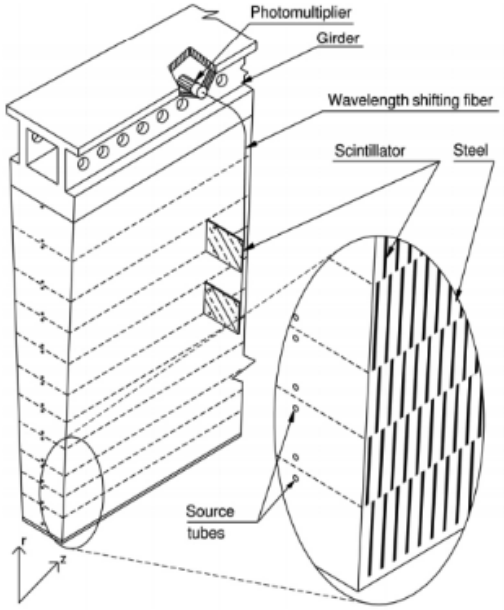


Figure 1.2: Schematic showing the various components of the optical readout, namely the tiles, the fibers and the photomultipliers. The trapezoidal scintillating tiles are oriented perpendicular to the colliding beam and are read out by fibers coupled to their non-parallel sides [7].

As the proton-proton interaction rate at the design luminosity of $10^{34}\ cm^{-2}s^{-1}$ is approximately 1 GHz and the event data recording, is limited to about 200 Hz, there has to be a system that selects which information is important and which is not, this system being the trigger system. The trigger system consists of two levels, the Level-1 (L1) trigger and the high level trigger. The decision maker of whether or not an event should continue to be processed is the L1 system, which uses a subset of the total detector information to do so, reducing the data rate to approximately 75 kHz, which is limited by the reading system bandwidth, however it can be upgraded to 100 kHz. While the high level trigger provides the reduction to a final data collection rate of approximately 400 Hz. [8].

The Read Out Driver (ROD) is the chain link between the front end electronics and the general data acquisition system of the ATLAS detector (DAQ). The TileCal ROD is responsible for reading and processing 9856 channel data every $10\,\mu$s, and it must also be able to work in real time. The data collected from these channels is digitized and transmitted to the RODs, using high-speed optical links [9].

In order to perform precision measurements of the Higgs boson and other particles properties and to search for new particles it is necessary to increase the luminosity of the LHC and this will have impact on the ATLAS TileCal electronic system because [10]:

- The system is located inside the detector, working under high doses of radiation;
- Current TileCal electronics, and in particular the high voltage electronics, has been designed to be radiation hard for 10 years of operation and it has been in operation for almost 20 years;

- The difficulty in maintaining and replacing any board that composes the electronic system. This is only possible when the LHC stops for at least a few months, reducing the radiation levels present in the cave and thus allowing an operator to enter the cave to make the reparations. However, even with this pause, the radiation levels remain very high and as a result the time an operator can be in the cave is very limited in addition to the reduced maneuvering space, which makes these operations very difficult.

Furthermore, increasing the luminosity would increase the amount of radiation and data that the PMTs would receive while the current electronic is not fast enough to process that increase of data. Thus, an upgrade is needed for all TileCal electronics and particularly for the HV distribution system, which is the subject of this work.

1.2 Current High Voltage Distribution System

The current High Voltage Distribution System (HVDS), designated by HV Opto, is responsible for the distribution of high voltage power to the TileCal PMTs, which are in the ATLAS cave, and since it was designed over 10 years ago, there are components that are currently obsolete. On the other hand, these are under high dose of radiation that affects the functioning of the components, and for this reason it is also complicated to repair or even replace the components as already mentioned.

All electronics are contained in 256 retractable drawers that can hold up to 48 PMTs each. Every drawer has the capacity to scan and process the signals, and these drawers have their own ”sub-drawer” that can contain up to 24 PMT blocks. All the digital information obtained from the reading and the control is transferred to control units that are placed in the calorimeter control room, that is located outside the ATLAS cave, by optical fibers [11].

An external source can provide two high voltage values, -830 V or -950 V, so each PMT can receive their individual high voltage to work correctly. The HV Opto board, resorting to these two tensions, can adjust locally up to 24 different voltages within a range of 350 V below the applied input voltage. There is a dedicated control card, the HV Micro, for every two HV Opto, and the main component of this board is a microcontroller, and more specifically the Motorola MC68376. Communication between HV Opto and HV Micro is via a VME (Versa Module Eurocards) bus [12].

The HV Opto offers several features such as:

- Individual adjustment, for each PMT, of the high input voltage using digital-to-analog converters (DACs) and control loops;
- Turn channels on/off for each set of 24 PMTs, but only have the ability to turn off/on all the 24 channels at a time;
- Reading a reference test voltage;

- Individual reading of the high voltage value applied to the PMT using analog multiplexers and an analog-to-digital converter (ADC);
- Reading two temperature sensors, this reading being done in 2 different positions on the HV Opto board, provides temperature flow information along the board.

1.3 High Voltage Distribution System Upgrade

As already mentioned above, an increase of the luminosity of the LHC is necessary to improve the measurements and to search for new particles. With such change it would be difficult to substitute or repair the electronics in use on the TileCal detector. Not to mention the fact that even with the regular updates realized on the LHC, many of the components that were used for the last 10 years are now obsolete and don't have the capacity to process all the data from the collisions after the upgrade.

With this in mind, two teams from different countries, were assigned to develop the new high voltage distribution system. The US[1] team proposed a board with new components capable of withstanding the high doses of radiation, and it would remain in the same place as its predecessor, while the Portuguese team proposed a remote option.

The new high voltage distribution card being developed by the Portuguese team, called HV Remote, aims to overcome some problems placing the HV Remote cards in a room 100 m away from the detector cave, designated by USA15 (low radiation environment). This will avoid the exposure to radiation, expanding the lifetime of the system and allowing for immediate maintenance and replacement when needed, overcoming two of the major problems of the actual system. In this case the high voltage distribution to PMTs is done with a cable. 256 HV Remote cards will be produced, each one will be able to supply 48 PMTs (having 48 channels). Each channel will be controlled from a dedicated control system on the HV Remote board and by a digital control system implemented on a Field Programmable Gate Array (FPGA). Sixteen HV Remote plus power supply boards will be controlled by one FPGA board. Finally, the FPGA boards will be controlled by the digital control system (DCS), which is TileCal's general control system, figure 1.3. But this system has one major disadvantage, which is that it requires a large number of multi-conductor cables.

The main goals that HV Remote aims to achieve are:

- Be able to turn on/off each channel of the PMTs individually;
- Individually adjust the high voltage supply value (between -360 V to -1000 V) for each PMT;
- Reading the high voltage value applied to each PMT;
- Reading of two temperature sensors;

[1] United States

- Reading a test reference voltage;
- Revision of the individual PMT HV control loop to mitigate the noise.

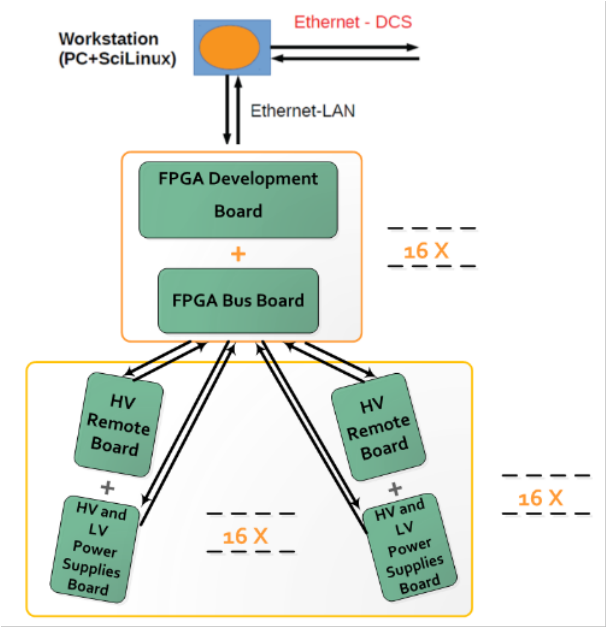


Figure 1.3: HV remote system control architecture.

As in this new system will be implemented a digital control, as well as the associated buses, not existing in its predecessor, the dedicated HVMicro control card will no longer be required. It should be noted that in this new system instead of primary high voltage sources will be developed dedicated high and low voltage power supplies. These will be the central point of this thesis, where will be presented all the steps taken in the development of dedicated sources as well as current state of the project and the work that is yet to be done.

1.4 Proposed Power Supplies Board

As mentioned earlier, the purpose of HV Remote is to distribute high voltage to PMTs. These individual voltages are obtained from a high voltage that must initially be supplied to HV Remote. In the current HV distribution system (HV Opto), the low power supplies are shared with the other ATLAS detectors, this means that the low power supplies are in the cave near the detector and a primary HV source is placed in the USA15 room. With this new HV Remote system, all the needed power supplies, will have to be installed also at the USA15 room, near to the HV Remote board. On the other hand, to mitigate the HV noise, for economic reasons and to facilitate their maintenance/replacement, low power printed circuit board (PCB) mount DC/DC converters to HV are used, instead of an expensive primary HV source. Thus, there is a need to project a board capable of producing the necessary power supply voltages for the HV Remote, the high and the low voltages, and that was the main objective of this thesis.

The Power Supplies board will be a dedicated power supply card that will be connected to the HV Remote board, and there will be one Power Supplies board for each HV Remote, so it will have to meet some criteria so that when both are connected they can work properly:

- The board must be capable of providing high voltage, with low noise, and be able to produce two alternative HV values, -830 V and -950 V both with 10 mA of output current, so each PMT can receive the necessary supply to work in perfect conditions;
- It has to provide the following low voltages: ± 12 V with 2.5 A of output current, and 3.3 V with 0.8 A of output current, to the analog and digital components of the HV Remote board;
- It shall be able to read and send to the FPGA (that resends to the DCS system) the power consumption of each power supply (current and tension);
- It should allow the temperature reading on 2 different places of the board;
- It must offer the possibility of turning off the power supplies individually by software (digital control system) and turn them off all together by hardware (manual switch).

This thesis will feature a more detailed description of the mentioned supply board, called Power Supplies board, of the interface that will be used with the board which allows the digital control system and the board tests that were made. Namely, to some of the used components and to ensure that it will work properly. The Power Supplies board will use the same protocol as the HV Remote board, as illustrated in figure 1.4, although it will be used a serial peripheral interface (SPI) bus independent from the one of the HV Remote board.

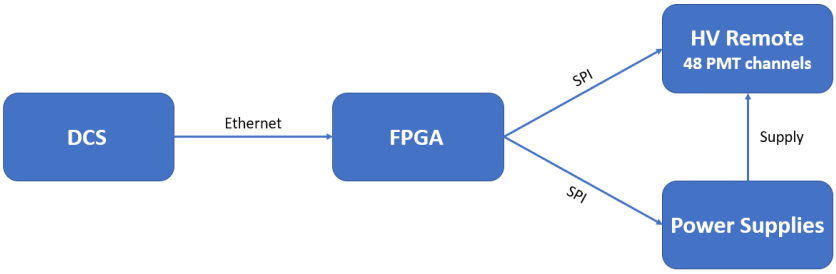


Figure 1.4: Diagram of HV Remote distribution system and dedicated power supply card.

The following chapters will describe in detail the Power Supplies board design process (Chapter 2) and how it works, followed by a description of the user interface that will control the Power Supplies board (Chapter 3). In chapter 4 a description of the tests that will have to be performed in order to ensure its proper functioning will be presented, culminating with the conclusions drawn from the work done and work to be done in the future (Chapter 5).

Chapter 2

Power Supplies Board

As already mentioned in the section 1.4, the purpose of this thesis was to develop the Power Supplies board for the HV Remote. This chapter will detail the main features, as well as, the development steps of this board.

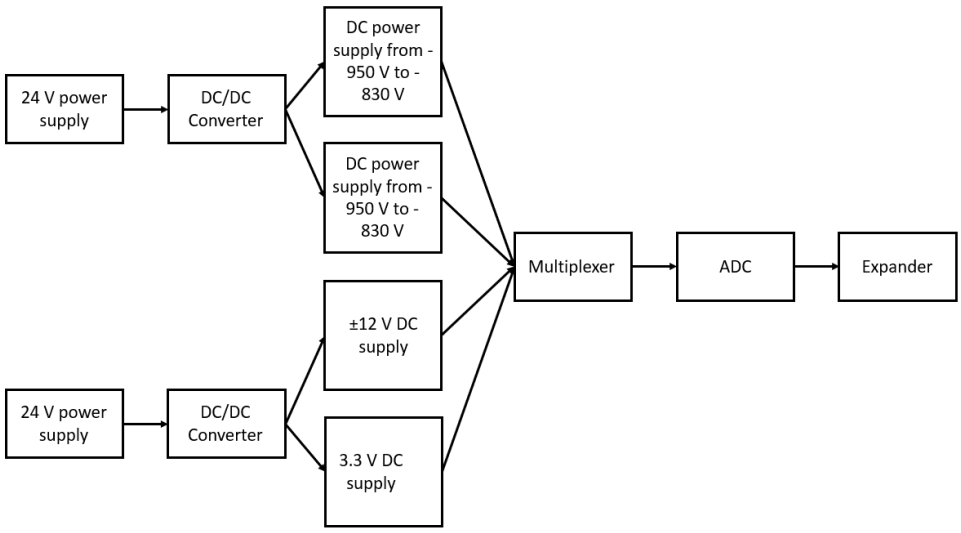


Figure 2.1: Power Supplies board scheme.

This board has to deliver all the voltages necessary to the HV Remote, with low noise. In the case of high voltages, need to be capable of producing two high voltage values, -830 V and -950 V @ 10 mA, because each PMT in TileCal is unique and during operation may behave differently. To properly calibrate the PMTs voltage, one of these two values need to be supplied to the HV Remote. The other voltages this board must supply are low voltage, ± 12 V @ 2.5 A and 3.3 V @ 0.8 A, for the analog and digital components of the HV Remote board. The Power Supplies board should be able to read the current and tension consumptions of the board. In order to be able to supply all the necessary voltages for the HV Remote, the Power Supplies board uses dedicated DC/DC converters, which convert the incoming voltage from 2 external 24 V @ 20 A sources to the desired power supply, as shown in figure 2.1. Each crate will be powered by 2 of these external 24 V @ 20 A sources, ie each 2 external sources will supply 32 boards since each crate will have 16 HV Remote boards plus 16 Power Supplies boards (figure 1.3).

To control the output voltage of each of the two HV supplies an appropriate digital signal must be send by the digital control system. This system is based in a SPI communication protocol and to separate the received serial data, a serial to parallel converter is needed. For that, a 16-bit port expander with SPI interface (MCP23S17) is used, this expander also controls the enable/disable of each DC/DC converter used to obtain the desired power supplies. In this board was also implemented a 16-bit analog multiplexer (MPC506) which is used together with a 12-bit ADC (MAX1240) to provide all consumptions as well as the output of two temperature sensors (TMP17), placed in two different positions, thus allowing a temperature flow monitoring.

2.1 Project

The first step in the development of the Power Supplies board was the project of all the circuits needed to fully fit all the board requirements. This is the most interesting part of the project. Not less important is the choice of the components (in Appendix B is presented the list of all used components), as this choice must respect certain parameters so that the Power Supplies board can perform the required features. One of the biggest problems in this step was the appropriate choice of DC/DC converters, since we can only use, for each set of 16 Power Supplies boards: two 24 V @ 20 A, one 3.3 V @ 0.8 A and one ± 12 @ 2.5 A power supplies. The 24 V @ 20 A power supplies are the primary power sources from which all others (the output power sources to HV Remote) are generated, being all DC/DC converters powered by these two power supplies. The input 3.3 V @ 0.8 A and ± 12 @ 2.5 A power supplies will be in charge of power all other components used on this board. Thus, we have a limitation of current that we can use versus the current that is required to be supplied to the HV Remote board.

In the first approach to the choice of components this problem was quite evident. The choice of DC/DC converters at an early stage had only counted the current required for the normal operation of board with all PMTs on and not for the case as they are off as required (in the worst case all the 48 PMTs are off). When a PMT is disabled, due to the implemented circuit in HV Remote, the current consumption increases. To solve this problem, new DC/DC converters had to be chosen but this time taking into account what the two 24 V @ 20 A are supplying in relation to the consumption of the Power Supplies board converters, not forgetting that the current supplied by the converters has to take into account the current consumption of the HV Remote board. A new choice of the two external power supplies (24 V) need also to be done, taking into account the fact that they are common to all the boards the crate and they supply 16 Power Supplies boards and not just one.

For each Power Supplies board, four DC/DC converters were chosen, two of them generate the primary HV, -830 V and -950 V @ 10 mA, (C12446-12 Hamamatsu), one for the HV Remote digital circuits, 3.3 V @ 0.8 A (TDK-Lambda CC1R5-2403SF-E), and other for the HV Remote analog circuits, ± 12 V @ 2.5 A (TDK-Lambda CCG304812D). It is necessary to use 2 HV DC/DC converters to increase the stability and to ensure that all PMTs get enough current. Another concern at this stage was the need for the voltages produced by the DC/DC converters to be as stable and low noise as possible. Therefore, there must be a careful choice of the capacitors used for the used filters, as well as the noise decoupling capacitors, the

latter require close attention to their PCB placement and should be as close as possible to the respective power supplies input pin.

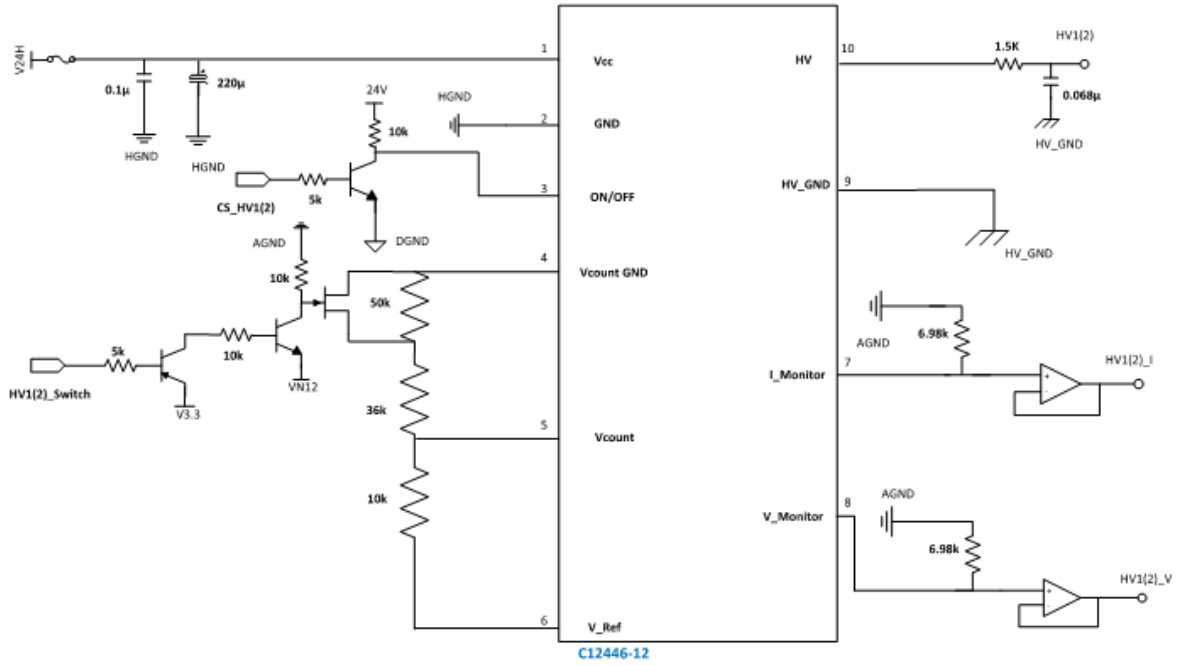
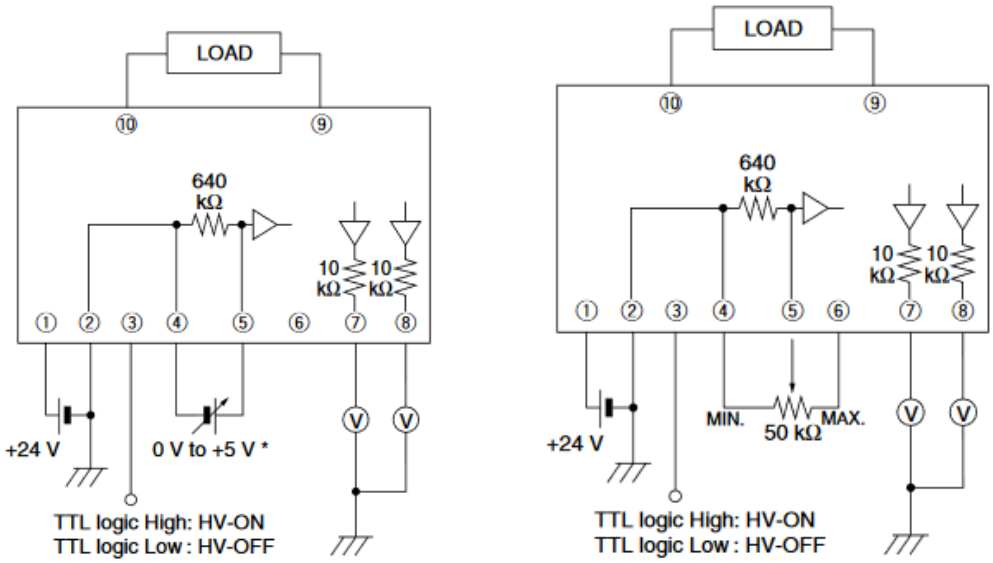


Figure 2.2: Electrical scheme of the electronics associated with the DC/DC converter (C12446-12 Hamamatsu 0 V to -1000 V @ 10 mA).

In the case of the HV converter, figure 2.2, it was necessary to use some components to select the voltage that the converter would be outputing, since this converter offers the ability to produce voltages between 0 and -1000 V. To do this, we first looked at its data sheet [13], the manufacturer recommended two different circuits to do so. The first was using an external voltage setting an output voltage value, the second circuit made use of a potentiometer to regulate the output voltage, as represented in figure 2.3. Both circuits were unsuitable for what was intended but the first was not useful as it did not allow changing by software the output voltage and the HV Remote needs 2 different HV voltage values. The second was not convenient either because although it allows changing the output voltage it could only be changed manually. In order to be able to change the voltage by user in the DCS system, it was required to create a new circuit that would allow this. This circuit is shown on figure 2.4, and uses two bipolar junction transistors, BJT, one PNP (BC559BTA) and one NPN (BC549B), one Junction Field Effect Transistor, JFET, N-Channel Switch (J111) and a resistors chain. The transistors change state, cut-off or saturation, are depending on a digital signal sent by the DCS. When the JFET is in cut state -950 V output (signals HV1_Switch and HV2_Switch for each HV output, table 3.1, with low) and when it is in saturation state -830 V output (signals HV1_Switch and HV2_Switch with high).



(a) Output voltage control by external voltage, using pins 4 and 5 from the converter[13].

(b) Output voltage control by external potentiometer, using pins 4, 5 and 6 from the converter[13].

Figure 2.3: Circuits recommended by the manufacturer to select the output voltage of HV converters.

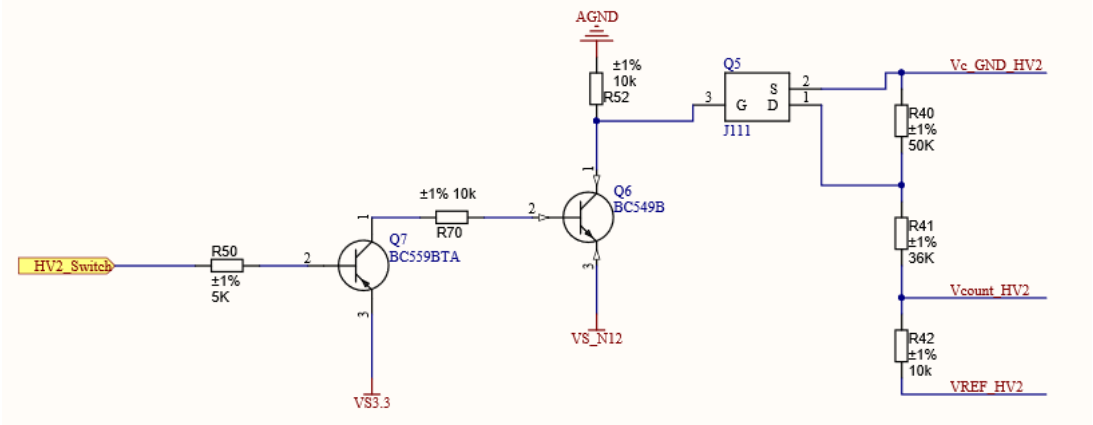


Figure 2.4: Circuit design to to select the output voltage of HV converters by software.

The implemented circuits associated to the low voltage DC/DC converters are shown in figures 2.5 and 2.6. Some of these circuits, as the filters, are recommended by manufacturer but others are needed to full fit the specific requirements of DCS, to ensure that it would be able to do the enable/disable of the converters and made the real time reading of consumptions. A more detailed description of this circuits is made in 2.1.1 and 2.1.2.

Another required feature is the temperature reading into different places (2.1.3) of the Power Supplies board. These measurements are important to know the temperature flux in operation and to design a proper cooling system.

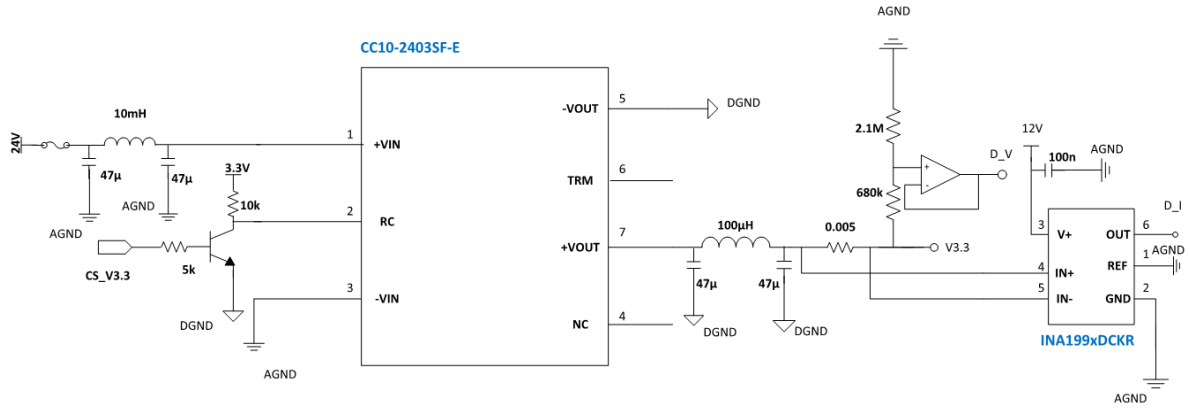


Figure 2.5: Electrical scheme of the electronics associated with the DC/DC converter (TDK-Lambda CC1R5-2403SF-E 3.3 V @ 0.8 A).

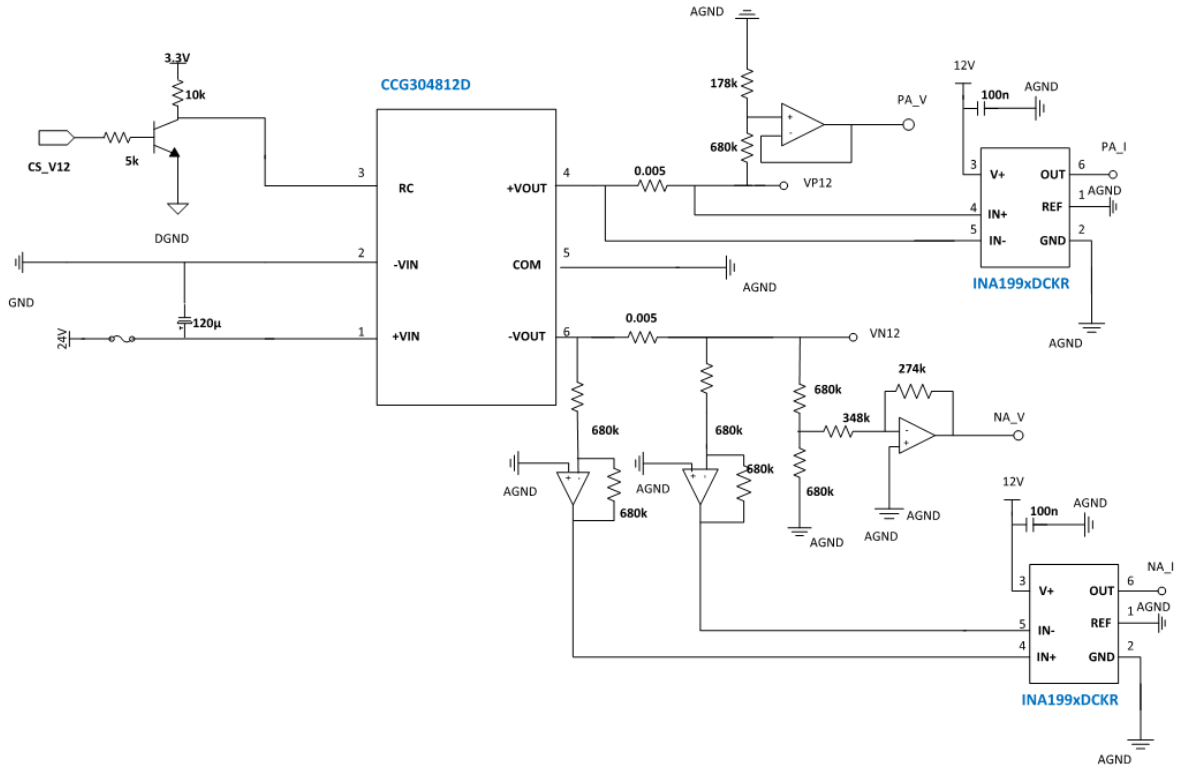


Figure 2.6: Electrical scheme of the electronics associated with the DC/DC converter (TDK-Lambda CCG304812D ± 12 V @ 2.5 A).

The second step of this project consisted in the design of the Power Supplies board PCB, to do so it was used the program called Altium Designer 16.0 from Altium Limited. This step will de developed in 2.2.

2.1.1 Enable/Disable of the DC/DC Converters

In the Power Supplies board each DC/DC converter can be enabled/disabled in two different ways, one strictly digital and its command can be remotely executed via computer (DCS), and another manual that allows to turn off all the converters at the same time with a hardware switch implemented as a safety issue and for board maintenance/substitution.

The digital enable/disable uses one of the pins of each converter, specific for this task. This pin is connected a general purpose transistor with junction NPN (2N3904) and two resistors, one of 5 kΩ and one of 10 kΩ, in the base and in the collector respectively. It's possible to see this circuit connected with pin 3, called on/off, in figure 2.2. In figure 2.5 the referred circuit is connected with pin 2, called RC, and in figure 2.6 it's connected with pin 3, also called RC.

To control the state of each converter the output values of the 16-bit port expander with SPI interface (MCP23S17) are used, figure 2.7. This component communicates directly with the computer through the SPI communication protocol. This protocol allows sending a command to the port expander, using the Graphical User Interface (GUI), which was developed in this thesis specifically for the Power Supplies board, to activate the converters. In the output registers of the port expander a low or high digital signal (CS_HV1, CS_HV2, CS_V12, CS_V3.3, figure 2.7) are connected to the resistors at the base of transistors 2N3904, causing them to change state. As an example, when a low digital signal is sent the transistor will be in cut-off state, which means that in the pin RC of the DC/DC converter is applied a voltage of 3.3 V, then the converter will be activated, and deactivated when the digital signal sent is high, figure 2.5. In the case of the HV converter the activation method is similar with only a difference in the voltage applied to the collector resistor, since to activate this one a 2.4 mA current is required which is obtained by applying 24 V instead of 3.3 V, figure 2.2.

Note that during component testing and the development of the GUI, instead of the FPGA, a Raspberry Pi 3 B+ card was used, in conjunction with a code specially developed for this project, which will be presented in Chapter 3.

As you can see in the figure 2.7, the output signals of the port expander also controls the output voltages of HV converters, as well as the multiplexer and the ADC components. This will be further explained in section 3.2 where the code developed for the GUI will be presented.

The other way to disable the converters (an hardware one) is to cut off the voltage supply to the Power Supply board, thus disabling not only the converters but all other components that need an external voltage to work. This is done by placing three switches, U18, U19 and IC4 represented in figure 2.8, just after the socket where the voltage supplies are provided by the external sources (24 V @ 20 A, sources). There are four different external power supplies

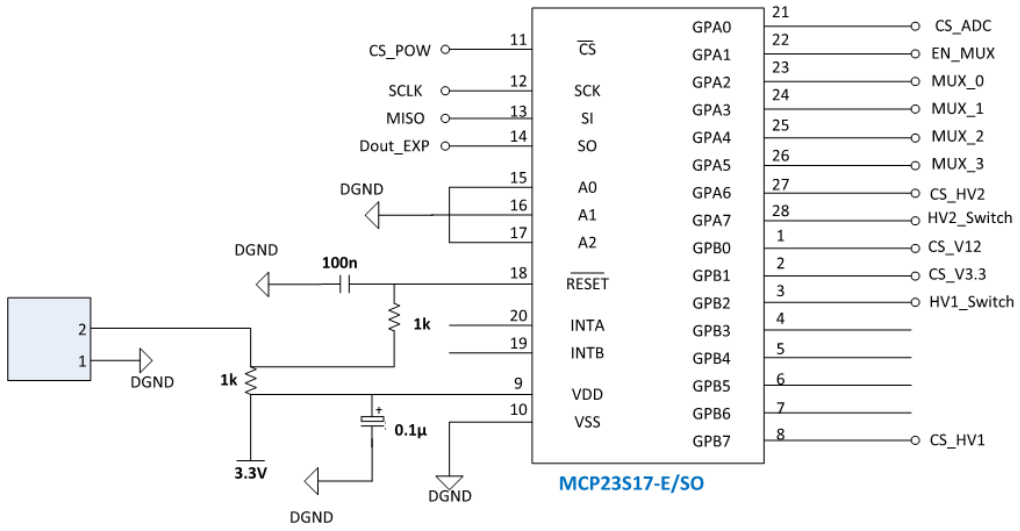
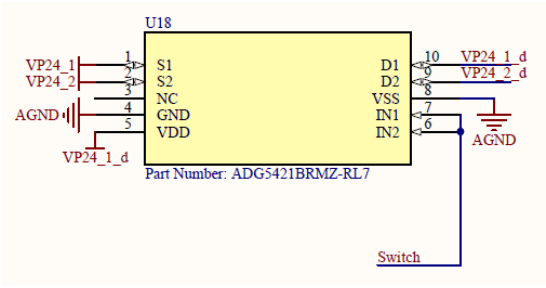


Figure 2.7: Electrical scheme of the electronics associated with the 16-bit port expander with SPI interface - MCP23S17.

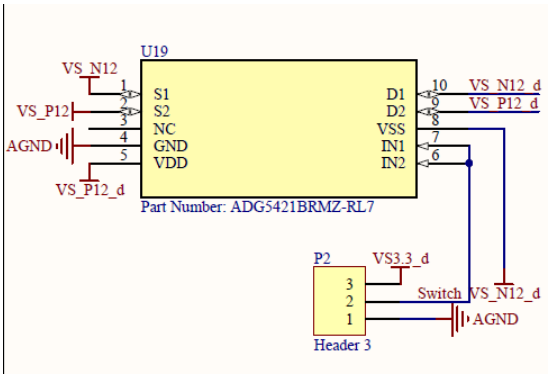
that are used to power this board, two 24 V @ 20 A, one 3.3 V @ 0.8 A and one ± 12 V @ 2.5 A, noticing that the last one have two different tension values that can be provide at the same time thus counting as 2 different power supplies. It's used a switch for the two of 24 V (ADG5421BRMZ-RL7), other for the ± 12 V (ADG5421BRMZ-RL7) and other to the 3.3 V (ADG601BRTZ-REEL7), as seen in the figures 2.8a, 2.8b, 2.8c respectively. After that, the control pins of the three switches are connected to one of the pins of a connector of 3 pins, that is displayed in figure 2.8b designated as P2 (Header 3). The other 2 pins of this connector are wired, one to the ground (0 V) and the other to the 3.3 V external source. This connector (P2, figure 2.8b) will be substituted by an analog switch, when the board is produced.

Figure 2.8d shows the schematic of the ADG601BRTZ-REEL7 switch, and through this it is possible to see that the operation of this type of components is not very complex, having only 3 pins, besides the power supply, one of them being pin D where a signal is input, pin S where the signal goes out if pin D is connected to S and pin IN which controls whether the two previous pins are connected or not. The ADG5421BRMZ-RL7 switch works the same way as ADG601BRTZ-REEL7 with the only difference being that the circuit is duplicated allowing two signals to be input and output at the same time.

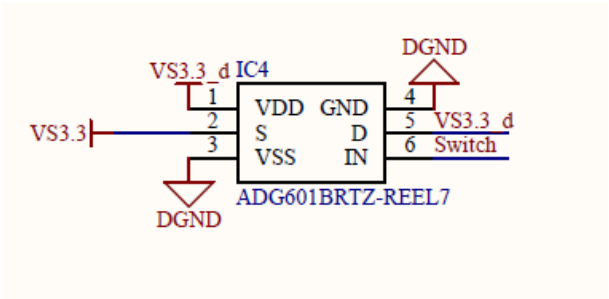
As already mentioned, the P2 connector will then be replaced by a manual switch, which will allow to disable all converters at the same time, this is done by breaking the connection between the Power Supply board and external power supplies. This substitution is shown in figure 2.9, and in this case, when the manual switch is in the off position a digital signal will be sent to all IN pins of the three switches (U18, U19 and IC4), breaking the connection between pins D and S, thus disrupting power to the Power Supply board, which will only be restored when the position of the manual switch changes to on.



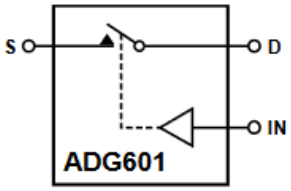
(a) Low voltage external power supply (24 V @ 20 A) switch (ADG5421BRMZ-RL7).



(b) Low voltage external power supply (± 12 V @ 2.5 A) switch (ADG5421BRMZ-RL7).



(c) Low voltage external power supply (3.3 V @ 0.8 A) switch (ADG601BRTZ-REEL7).



(d) Schematic figure of the switches.

Figure 2.8: Switches used to turn off sources in case or digital mode does not work.

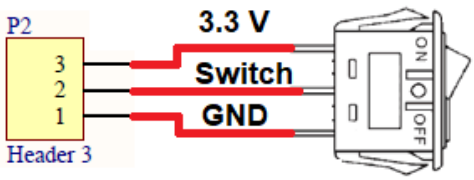


Figure 2.9: Manual switches connected to the connector P2.

2.1.2 Real-Time Consumption Readings

Real-time DC/DC converters consumption readings, both current and voltage, are taken using a 16-bit analog multiplexer (MUX36S16) and a 12-bit ADC (MAX1240), both represented in figure 2.11. The last one is connected to the MISO[1] Serial Peripheral Interface (SPI) signal, which is read by Raspberry board to the GUI that will run on a computer, transmitting the information to the user. However, not all converters used have a pin that provides this information, as is the case with 3.3 V and ± 12 V converters, CC1R5-2403SF-E in figure 2.5 and CCG304812D in figure 2.6 respectively, so it is necessary to implement other components to make this possible.

The HV converters are the only ones that have output pins that allow reading current and voltage consumption directly. Since these pins provide voltages between 0 and 5 V depending on the current and voltage the converter has on the output, it is also necessary to perform a conversion of these values to obtain the actual consumption, which is given by 0.5 mV / V for voltage and 0.5 V / mA for current. However, due to the limitations of the analog multiplexer, the maximum input voltage is 2.5 V, it is not possible to use the values directly from HV converters. This limitation is easily overcome by using a passive linear circuit that produces an output voltage that is a fraction of its input voltage, is called a voltage divider, and the simplest way to implement this circuit is to use two resistors in series, figure 2.2.

In figure 2.10a it's represented the referred voltage divider, and the output tension is calculated by the equation 2.1, were V_{out} is the output voltage and V_{in} is the input voltage.

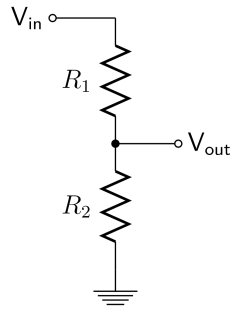
$$V_{out} = \frac{R2}{R1 + R2} \times V_{in} \tag{2.1}$$

As in this case it is only necessary that the maximum reading output of the reading pins is 2.5 V and since these pins have an output impedance of 10 kΩ, a resistor close to this value is used. In addition to that is also used a buffer, figure 2.10b, so that there is no influence of the input impedance of the multiplexer, which could cause an error in the division of voltage giving at the end a wrong value. In fact, a smaller resistor is used (6.98 kΩ), as shown in figure 2.2 in pins 7 and 8, because the maximum output that is taken from this converter is -930 V and this is not the maximum this converter can provide, thus the equation 2.1 would look like:

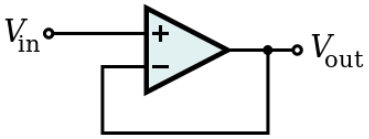
$$V_{out} = \frac{6980\Omega}{10000\Omega + 6980\Omega} \times V_{in} \tag{2.2}$$

For the others, there were some differences to obtain the consumption values. Since this reading pin does not exist, the voltage output pin was used directly. For the ± 12 V DC/DC converter, special attention must be paid to the fact that it has two different outputs, one positive and one negative, which implies that the way each reading is made is different, since the multiplexer only accepts positive values. Therefore, a voltage divider is also used in this output but instead of the buffer an inverter voltage amplifier circuit, figure 2.10c, is used to invert the signal so it can then be supplied to the multiplexer.

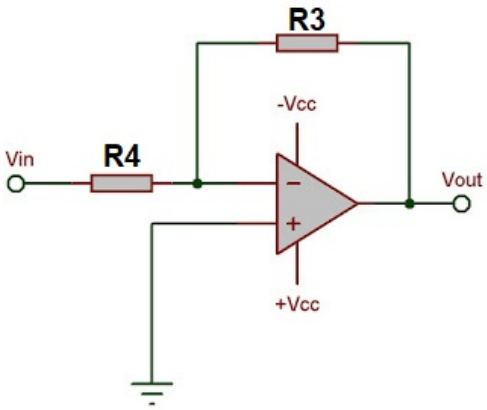
[1] Master Input Slave Output - data output from slave



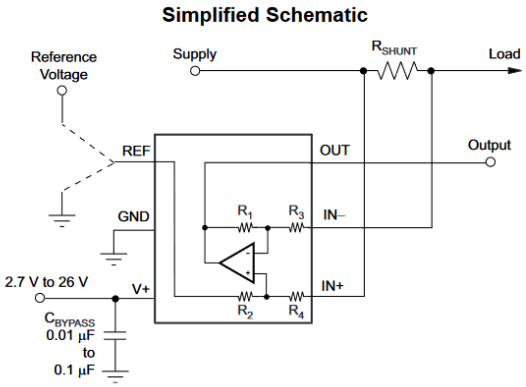
(a) Voltage divider scheme [14].



(b) Buffer scheme [15].



(c) Inverter voltage amplifier circuit scheme [16].



(d) Current sense amplifier ina199 [17].

Figure 2.10: Additional circuits used to allow consumptions reading.

$$V_{out} = \frac{-R3}{R4} \times V_{in} \tag{2.3}$$

In the equation 2.3, it can be seen that this circuit may have a voltage gain of less than one, making the voltage divider circuit unnecessary at first glance, but this is not quite true. As mentioned above, because there is a very tight margin in relation to the voltage that is produced by the converter and the one that needs to be supplied, it is imperative to avoid losses. As the readings of the consumptions are being taken directly at the output pin of the converters, there will be a loss of current, however it can be minimized using resistors with a high impedance, which is why voltage divider are used, not only to reduce voltage but also to prevent current losses, as possible to see in figure 2.6 in the pin 6.

To read the current consumptions it is not necessary to use a voltage divider or a buffer as referred to the voltage readings, but it is still necessary to use an inverter voltage amplifier circuit in the negative output, we need to use a special type of resistors to mitigate the output voltage loss, current sense resistors, which are used to monitor the current in a circuit and translate the amount of current in that circuit into voltage that can easily measured and monitored. But because of the small value of the resistor ($0.005\ \Omega$), the voltage drop is too small so, a dedicated amplifier is needed, a current sense amplifier (or current shunt amplifier, figure 2.10d). These special differential amplifiers give an output voltage proportional to the current flowing in sense resistor. The current sense resistor value used is $0.005\ \Omega$ and the current sense amplifier gain is 200, so the current value is given by the equation 2.4, where ΔV is the output voltage of the current sense amplifier.

$$I = \frac{\Delta V}{0.005 \times 200} \tag{2.4}$$

For the 3.3 V converter, as for the ± 12 V converter, a voltage divider and a buffer are used to read the voltage consumption, and a current sense resistor is used to read the current consumption along with a current sense amplifier, where the values of these two last ones are the same ones used in the ± 12 V converter, meaning that the current it's obtained using the equation 2.4 as well.

It is important to note that because only one ADC is used, only one value can be read at a time, this value being in this case the output value of the multiplexer. The description of the read values is defined in table 3.4 and the value read is defined by the user. This will be further explored in Chapter 3, where the definition of these readings as well as the multiplexer channel to which they are assigned will be relevant for GUI development.

2.1.3 Temperature Reading

As mentioned earlier, two 24 V power supplies, which are used to power all DC/DC converters (one for HV converters and the other for low voltage converters), have a current of 20 A, and due to this factor there may be overheating in the components that are supplied by them. However, it is not the only source of heat, all the components in operation release heat and

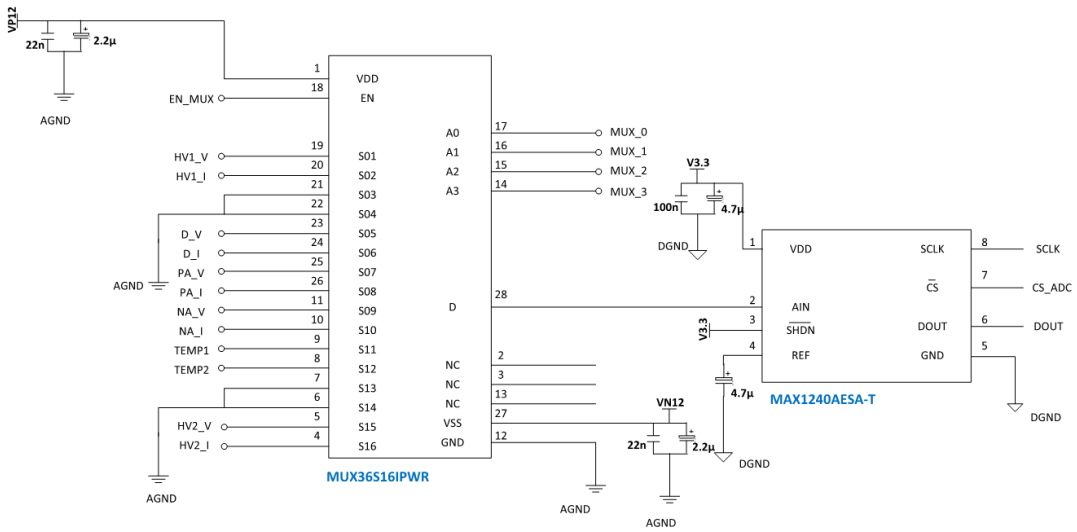


Figure 2.11: Electrical scheme of the electronics associated with the 16-bit analog multiplexer MUX36S16 and the 12-bit ADC MAX1240.

once the board is finished and implemented in a crate, it will be surrounded by many other boards. Although they may not produce much heat on their own, when together this heat can be greatly increased. Because of this the temperature reading at two different points on the board is very important, although it is not very relevant at the prototype stage this board is in, as only one board will be used. Reading the temperature at two different points on the board will provide information on the temperature flow and type of cooling system that will need to be implemented in the crate to prevent component overheating, resulting in malfunction or even total breakdown of the Power Supply boards.

For this, two monolithic integrated circuit temperature transducer, TMP17, were used, figure 2.12. This circuit provides an output current that is proportional to absolute temperature. The output of the TMP17 is converted in a voltage and is one of the multiplexer analog inputs. In the figure 2.13 the graph of the transfer characteristics of the TMP17 is represented. It is through this that it is possible to obtain the temperature value from the signal send by the ADC, presented at last in GUI.

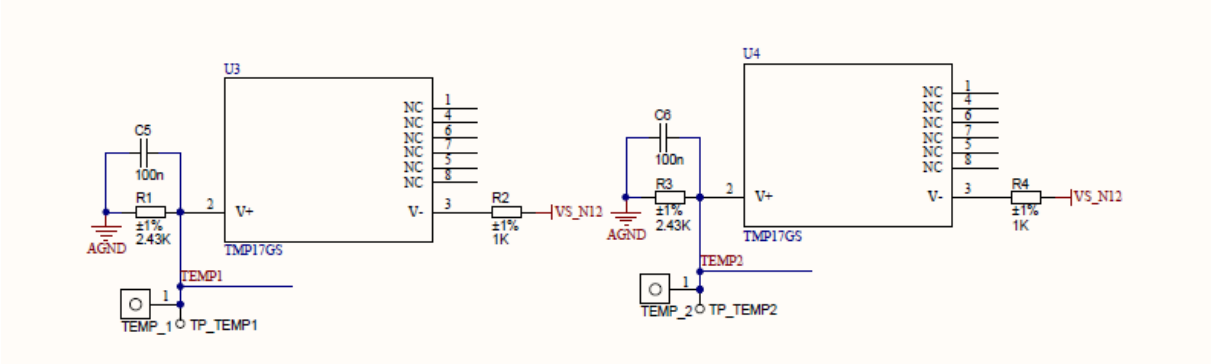


Figure 2.12: Electrical scheme of the electronics associated with the temperature transducer, TMP17.

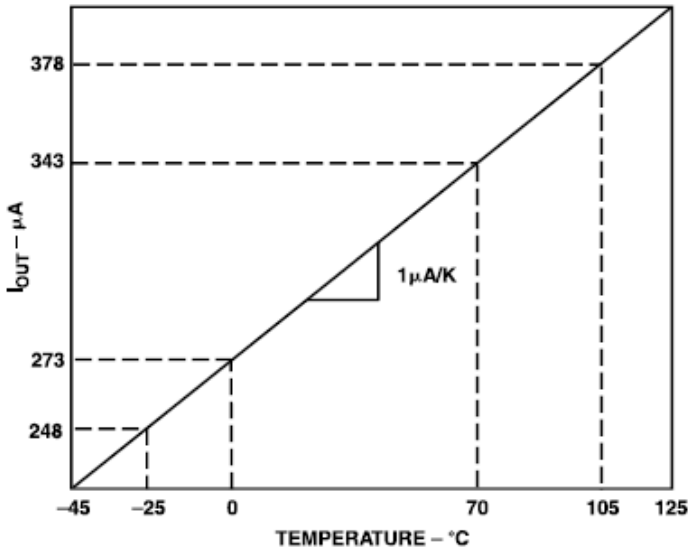


Figure 2.13: Transfer characteristics of TMP17 [18].

2.2 Printed Circuit Board

After the schematic designs are completed (presented in Appendix A), the next step is the printed circuit board (PCB) design. It is at this stage that all circuits, presented in the schematic designs, are translated into an actual physical design. But when it comes to designing high voltage PCBs, it's important to ensure that a good layout can optimize the strengths of the electric field to optimize the operation and lifetime of the PCB. Although the process of manufacturing high voltage PCBs does not differ greatly from that of normal PCBs, more careful attention to the materials used and their properties are critical to success. A thorough understanding of spacing rules, clearances, operating frequencies, and design techniques will greatly help you avoid setbacks and potential challenges, and there are some things that need special attention, such as [19]:

- Isolating high voltage areas - it's very important to group the high voltage circuitry together in order to minimize the impact that it will have on the rest of the board, and by doing this it's possible decrease the risk of arcing on the board [19];

- Decreasing voltage across the board gradually - even if isolation is done, by doing this, the roughness of the entire board is increased, preventing potential problems that could arise even with isolation [19];

- Isolating noise sources - if this is not done the noise can couple through parasitic capacitance on the board or in the insulation, this will allow an easy propagation to areas of the board that are very sensitive, compromising its integrity [19];

- Minimizing interconnects - if fewer interconnects are used, the odds of happen transient generation in the design are reduced, if not, it can damage sensitive components or reduce the performance of the board. This will also minimize the propagation of high voltage across the board. Special attention must be taken in the connections between

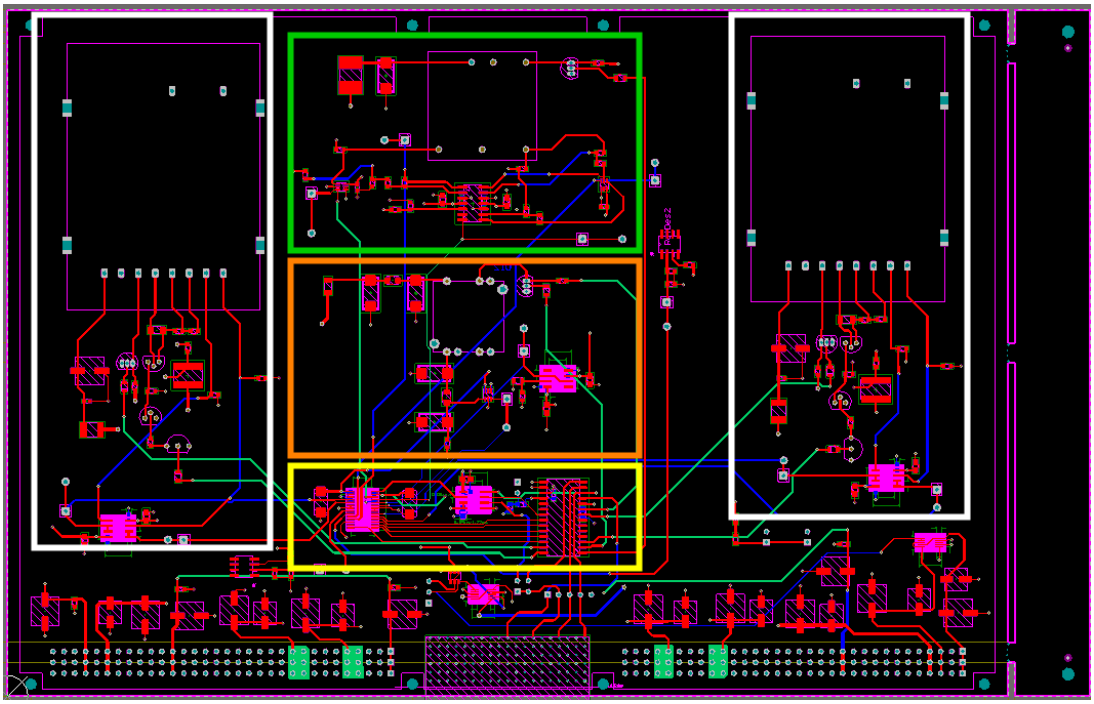


Figure 2.14: 2D printed circuit board of the Power Supplies board.

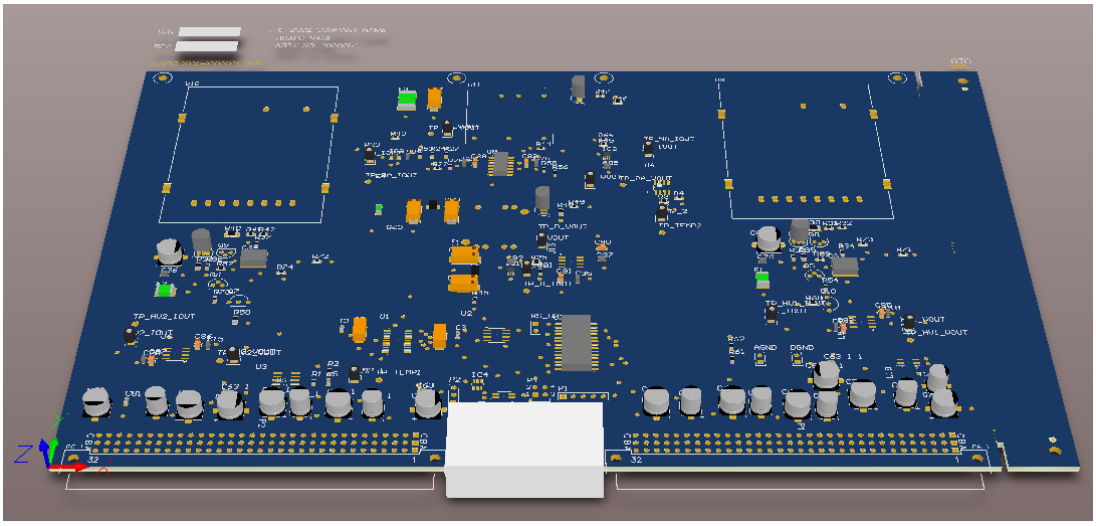


Figure 2.15: 3D printed circuit board of the Power Supplies board.

components and its noise decoupling capacitors. Some of them should be placed in the bottom side of the board to achieve that [19].

Figure 2.14 shows the final appearance of the PCB in the Altium environment. Within the white rectangles is the area of the board that is dedicated to the high voltage, isolated from the others, in the green and orange rectangles is the ± 12 V DC/DC converter and the 3.3 V DC/DC converter, respectively, as well as all the circuits associated with them, and finally inside the yellow rectangle are the digital circuits, thus being separated from the analog ones. The figure 2.15 shows the same PCB but it is in 3D. The board consists of 8 layers, as shown in figure 2.16, of which 3 are mass planes, 1 an output power plane, 1 an input power plane and 1 a layer for signals. There are only components in the top layer and bottom layer, the others are internal layers. But this is just the end product, before the PCB looked like this there was a lot of work invested along the steps required to design a good performance PCB. The first thing to do when starting to design the PCB is the placement of components, this is a very important step as it affects all future work, so it has to be done carefully and is a lengthy process.

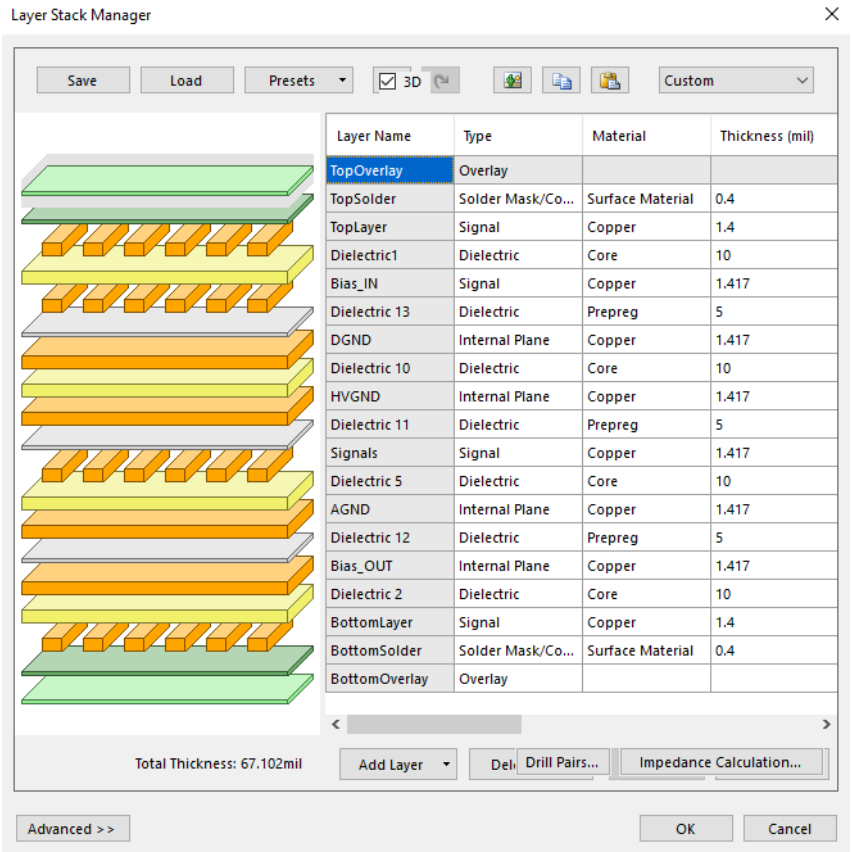


Figure 2.16: Layer stack of the Power Supplies board.

During the placement phase, there are some things that need to be present in the mind of the person performing this process. For instance, it's necessary to think about manufacturing, this means that several considerations must be taken into account during the design. The first is component orientation, if all components are always facing the same direction, when the board is produced, installation, testing and inspection will be much easier [20].

The second point to consider is how components are placed relative to each other. When the board is complete and ready for manufacturing, it will go through a solder oven to connect all parts to the empty circuit board. If big components block the small ones, they are likely to be poorly soldered. Another aspect that can be very crucial, and in the case of power supplies is very important, is when placing the components, always pay attention to size, not only in two-dimensional space, but also the height, width and weight, as is the case with HV converters that have to be placed at each end of the board so that there is no overweight on one side which can cause a break in the board [20].

Another thing to keep in mind during this step is to think about routing, although we are not in this step yet, it is important to take this into account so as not to make the mistake of placing components too close, causing a lack of space when routing starts, or the fact that if the components are too close this may heat up more [20]. Attention should also be paid to signal induction, which may be caused by the proximity of the components, as well as extra caution in separating the digital from the analog, since if they mix together the board will not work as intended. Finally, it is very relevant to have good mass planes as well as power plans, since they allow a great attenuation of the fluctuation of the voltages that may exist in both planes, thus allowing a noise reduction that is crucial in this project.

After these was taken into consideration, the placement was made, the HV converters were placed at the end of the board, one on the right and one on the left, and underneath were all the components associated with filters and consumption measurements, no others components were placed in these two areas, isolating the high voltage from the others components. The port expander, the multiplexer and the ADC, as these must communicate with all DC/DC converters, need to be in the middle so that the path through which the information travels is similar. They have been placed on the bottom of the card in the central zone near to the connector dedicated solely to signals such as the used SPI signal (MOSI[2], MISO[3] and SCLK[4]) and the chip select signal of the board. This region deals with all the digital circuitry of the Power Supplies board. The 3.3 V and ± 12 V converters were placed on the top of the board in the central area, so that they did not interfere with the other components, along with all components associated with the filters and its consumptions readings circuitry.

One very important thing I learned from doing this thesis was that when we are placing the noise decoupling capacitors on the PCB, we want them to be as close as possible to their respective components. One of the most effective methods to do this is to put the components on the top layer of the PCB and the capacitors immediately below them on the bottom layer connecting them with a via. This method saves a great deal of space and greatly reduces the noise that could be introduced by the distance between the components and their recommended capacitors, which could be quite critical for this board since one of the requirements is to be able to produce low noise voltages.

[2]Master Output Slave Input - data output from master
[3]Master Input Slave Output - data output from slave
[4]clock signal

After placement is completed, the next step is routing. In this step, the worst thing that can happen is to connect the networks incorrectly, but with the help of the Altium designer program this is not likely to happen. The other concerns at this stage are the crossing between nets and the width of the nets, the latter being quite relevant in this board, since there are high currents running through the nets and if the width is not adequate it will cause nets to overheat and consequently their rupture. In figures 2.17 and 2.18 are represented a table and a graphic of the trace width versus the current carrying capacity of a net that was taken in account during the routing process, it is important to note that in these figures the net used is only 1 oz. of thickness.

IPC Recommended Track Width For 1 oz cooper PCB and 10 °C Temperature Rise

Current/A	Track Width(mil)	Track Width(mm)
1	10	0.25
2	30	0.76
3	50	1.27
4	80	2.03
5	110	2.79
6	150	3.81
7	180	4.57
8	220	5.59
9	260	6.60
10	300	7.62

Figure 2.17: PCB trace width vs. current carrying capacity table [21].

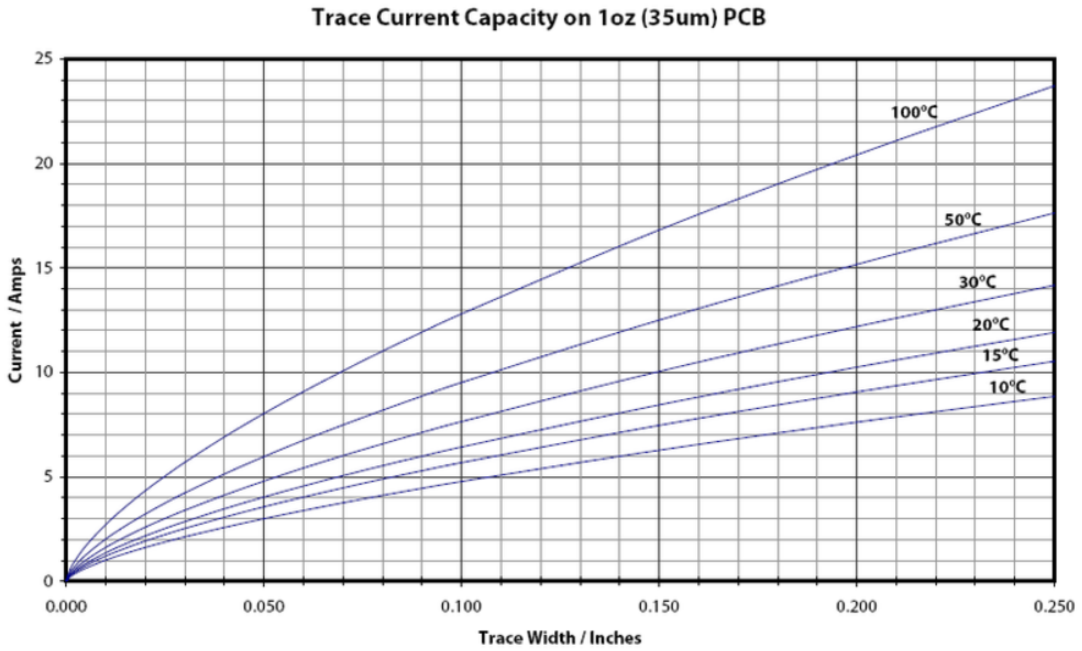


Figure 2.18: PCB trace width vs. current carrying capacity graphic in different temperatures [21].

The first thing to do in the beginning of PCB design, was the creation of the 3 mass planes, one for high voltage, HV ground, one for analog voltage, analog ground, and another for digital voltage. These are isolated from each other except for a single point where the three are connect. By doing this in the first place it is possible to simplify the connections that have to be made because the ground line connections of each component disappear (as these are made directly), and the fact that we have three separate planes in different layers allows us to have a very stable mass planes which reduces noise produced by the currents return.

Following the same principle of stability and low noise, polygons were used to make the supply and output voltages net planes, by doing this the voltages become more stable and are not affected as much by noise, as well as avoiding the problem of nets width. But there has to be extra care for the high voltage plane that is made using an exception to define the minimum clearance allowed between any two primitive objects on a copper layer, this allows a good isolation of the HV net with every other nets. In Appendix C it's possible to see the routing done by layer, however the mass planes are not represented since there are no nets in these layers.

After this is done, only the nets that connect the components to each other and the nets of the control signals remain, and to do this routing, there are 3 different layers, namely the top, the bottom and an internal signal layer. In the end, all the project properties were validated with the help of the Altium designer program, and all errors were corrected. After that a bill of material was also generated through the Altium designer program (presented on Appendix B), as well as a file of board plans and perforations and one of the position of the components. All of that are also done by the Altium designer program and a set of layer planes, called the gerber files, with all the information need are generated. These are files needed to the PCB fabrication. Only after this is complete is the board finally ready to be shipped for fabrication. Right now the Power Supplies board is in fabrication in one of the two different companies, the first is EUROCIRCUITS that is in charge of manufacturing the board without the components, and to do so they need the file of board plans and perforations previously generated. The other is Systion Electronics Lda, that is in charge of the assembly of the components, so they need the board already manufactured as well as the information of where to place the components, that can be found in the file of the position of the components.

Chapter 3

User Interface and Communication with Power Supplies board

In this chapter a detailed description will be given of the graphical user interface developed to facilitate communication between the Power Supplies board and the user, as well as how communication between them is performed. In figure 3.1 we can see the block diagram of the Power Supplies board digital control. The control master is a PC/workstation configured as a node of the DCS (Detector Control System) of ATLAS. The DCS will use an interface specifically created for this board, developed in WinCC programming language, in order to facilitate the communications that occur between the DCS and the Power Supplies board, which takes place with an FPGA as an intermediate. The DCS will control the port expander, and the outputs registers of the port expander will be connected to the multiplexer, the ADC and to the enable/disable pins of each DC/DC converter, allowing its digital control. The multiplexer was used to permit to read the analog signals from two temperature sensors, current and voltage consumptions of the DC/DC converters, one at a time. It's output value, with the value selected by the user, is the input signal of a 12-bit ADC that will convert these signals into digital ones.

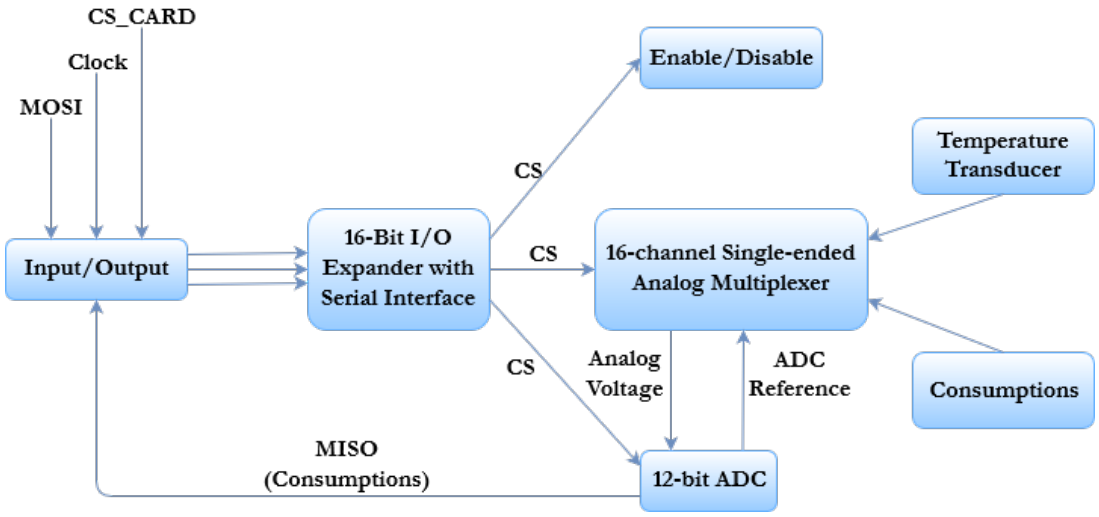


Figure 3.1: Block Diagram of the Power Supplies board control system.

To test the implemented board without the need of the ATLAS DCS system, a dedicated user interface (GUI) was implemented in the scope of this work in Python language. Instead of a FPGA, this interface allows the communication between the Power Supplies board and a Raspberry Pi 3 B+, via the Serial/SPI communication protocol, and the Raspberry Pi will also work as the computer since this is a low cost, credit-card sized computer that has its own operating system, and to work with it you just need to connect a computer monitor, a computer mouse and a keyboard. More specifically, this interface is intended to allow the functional test of the developed board during the production of more than one board, as well as the functional test of some of the components used as the DC/DC converters, the port expander and the ADC. Only at a later stage and for operation tests with all the boards of a crate will the interface for the FPGA be developed.

In this chapter will present the type of communication used in the communication between Raspberry Pi and the Power Supplies board, the already developed GUI and a more detailed explanation of the code behind this interface as well as the functions used in the implemented scripts. All scripts made for this project are in Appendix D.

3.1 Communication Protocol

The Serial Peripheral Interface (SPI) is a synchronous serial communication interface that was developed by Motorola in the mid-1980s and since then it as became one of the most widely used interfaces between microcontroller and peripheral ICs such as sensors, ADCs, DACs, shift registers, SRAM, and others [22].

SPI devices communicate in full duplex mode using a master-slave architecture and can only have a single master [23]. This protocol involves 4 separate lines to communicate as represented in figure 3.2, the clock (SCLK), so the data from the master or the slave is synchronized, the Master Output Slave Input (MOSI), the Master Input Slave Output (MISO), that allow to both master and slave can transmit data at the same time, and a select line to choose which device to communicate with, designated by slave select (SS), also called chip select (CS).

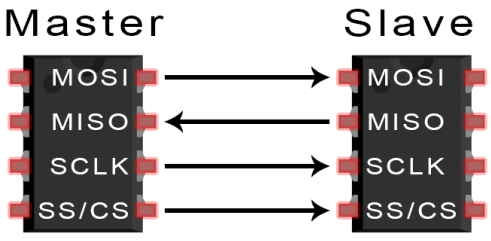


Figure 3.2: Correspondence between master and slave data lines in the SPI protocol [24].

There are several types of communication protocols, however, the SPI protocol offers several advantages over others. In the case of a common asynchronous serial port, if we only send

data, there is no control over it or guarantee that both sides are running at the same rate. Because computers rely only on a single ”clock”, that is the main crystal attached to a computer that drives everything, when two systems with slightly different clocks try to communicate, communication problems can occur. Therefore, to solve this problem, it's needed to add extra bits at the beginning and at the ending to each byte, allowing the receiver to synchronize data as it arrives. This makes the asynchronous series work relatively well but has a lot of overhead on the extra start and stop bits sent each byte and the complex hardware needed to send and receive data [25].

So, to overcome that issue, it is possible to use the SPI protocol. This is a ”synchronous” data bus, this means that the protocol uses separate data lines (MISO and MOSI) and a ”clock” that allows both sides (Master and Slave) to be in perfect sync. The clock is an oscillating signal that transmits to the receiver the exact time to sample the bits in the data line. When the receiver picks up this signal, it immediately knows that it has to look at the data line to read the next bit. Because the clock is shipped with the data, speed specification is not important, although devices have a maximum speed at which they can operate [25].

The figure 3.3 shows a schematic of how communication via SPI protocol happens. This communication can be divided into 4 different steps, the first being when the Master switches the SS/CS pin to a low voltage state, which activates the slave, thus selecting which device to communicate. The second step is for the Master to emit the clock signal so that the Slave can know when data transmission begins and when it ends. Depending on the type of logic used by the Slave, the rising or falling clock edge is used to sample and/or shift the data. It is finally in the third step, when the data is sent from the Master to the Slave using the MOSI line. In this step, one bit at a time is send and the slave reads the bits as they are received. The fourth and final step, it's not always needed, but if a response is necessary, the slave returns data one bit at a time to the master along the MISO line. This data line can also be used to read other kind of data from the slave as we will do in this project. The master reads the bits as they are received, only after that the SS/CS returns to a high state, ending the communication [24].

The SPI communication protocol offers the possibility for a Master to communicate with several Slaves, however in this thesis only one Slave was used, the Power Supplies board [23].

In SPI protocol, the master can select the clock polarity (CPOL) and clock phase (CPHA) to correspond to slave device requirements. Depending on the CPHA bit, the rising or falling clock edge is used to sample and/or shift the data. The CPOL parameter assigns the clock level when the clock is not active. The SCLK signal may be inverted (CPOL=1) or non-inverted (CPOL=0). For the inverted clock signal, the first clock edge is falling, and as for the non-inverted clock signal, the first clock edge is rising. If CPHA=0, the data is sampled on the leading clock edge, regardless of whether that clock edge is rising or falling. If CPHA=1, the data is sampled on the trailing clock edge. In figure 3.4 it's represented the 4 possible modes for the clock polarity and the clock phase [23][26][27].

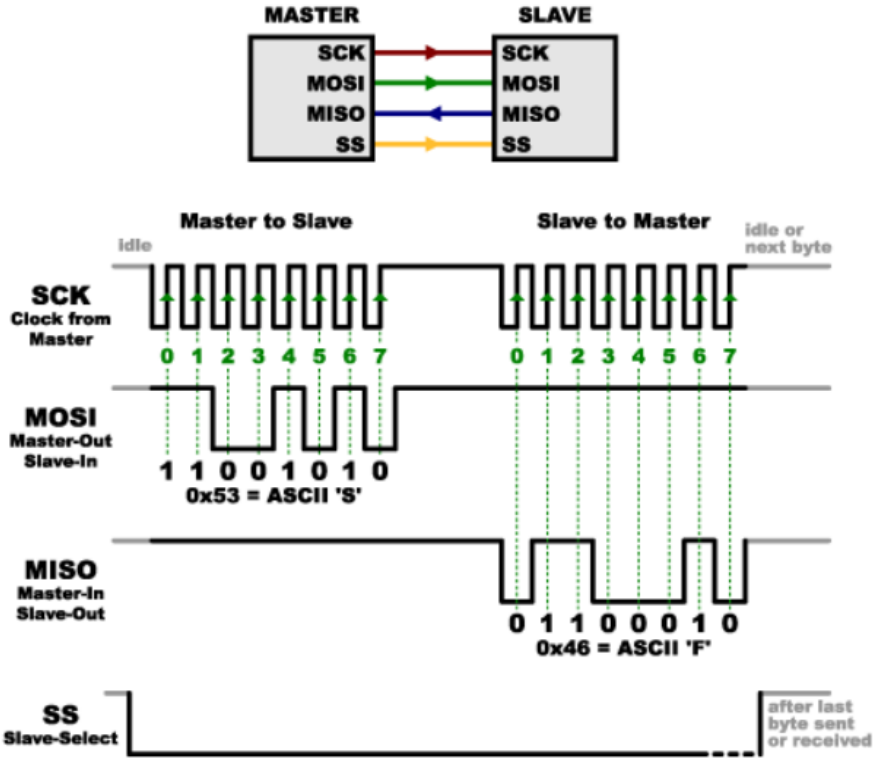


Figure 3.3: SPI Data Transmission steps [25].

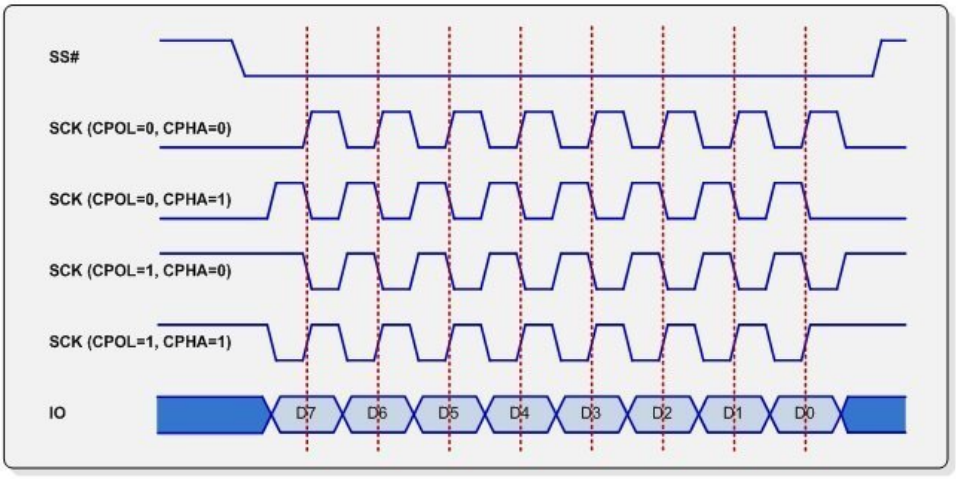


Figure 3.4: SPI clock polarity and clock phase modes [26].

3.2 Power Supplies Graphic User Interface

As mentioned in the previous chapters, Power Supplies board has several functionalities, such as the consumption and temperature reading, the selection of HV DC/DC converters output voltages and the enable/disable of the all DC/DC converters. These functions are all digitally controlled, which means it's needed to send a command each time one of these functions are performed. Without a GUI, both working with this board and testing it will become a tedious and exhausting work. In this context, a GUI was developed using Python version 3.7.3 as a programming language. This programming language offers a wide variety of packages available specifically for graphical development, as well as a wide variety of libraries that allow communication with various types of devices, such as the *spidev* library for interface communication with Raspberry Pi and RPi.GPIO. This library provides a class to control the General Purpose Input/Output (GPIO), which are programmable data input and output ports that are used to provide an interface between peripherals and microcontrollers/microprocessors, on a Raspberry Pi [28]. For the graphical part of the interface, the *PyQt 5* library was chosen.

Other library used was Qt, which is a set of cross-platform C++ libraries that implement high level APIs in order to access many aspects of the most modern systems, including mobile and desktop systems. PyQt5 is a broad set of Python bindings for Qt version 5. PyQt5 has over 35 extension modules implemented allowing Python to be used as an alternative application development language commonly developed in C++ language, in all supported platforms, including iOS and Android, thus allowing to build applications more quickly while not sacrificing much of the speed of C++ language [29].

The developed interface has three main sections: the enable/disable of the DC/DC converters, the selection of the output voltage of the HV DC/DC converters and, finally, the consumption and temperature reading. You can see how they are organized in figure 3.5, where it is also possible to observe that in the reading section there are two ways to read the consumptions as well as the temperatures, being possible to make only a single immediate reading or a set of readings that will be presented in a graph in function of the time this measurement was taken.

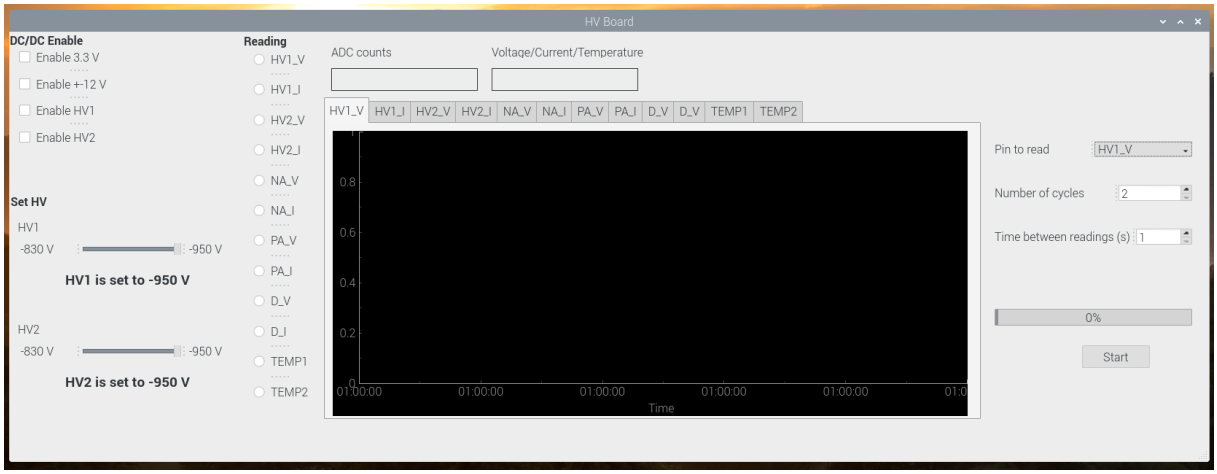


Figure 3.5: Power Supplies board graphical user interface.

In the enable/disable section of the converters, four check boxes are used, one for each converter, and to activate any of these converters it is only necessary to select the corresponding check box, being possible all the converters to be activated simultaneously. In the output voltage selection section of the HV converters, two sliders are used, one for each of the HV converters, in which case the output voltage selection is chosen by slide the sliders from one position to the other, since there are only two possible positions, -830 V and -950 V. These bars are further accompanied by text that makes it easier to understand the output voltage that is selected for a particular HV converter.

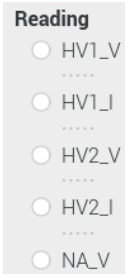


Figure 3.6: Five radio buttons of the twelve radio buttons used in the reading section of the GUI of the Power Supplies board.

In the reading section, for an immediate reading, twelve radio buttons are used to choose which measurement is desired, figure 3.6, and the measurements that can be made are the same as those presented in table 3.4, and it's only possible to select one of these radio buttons at a time. The result of this measurement will be presented in two boxes, one in real units, i.e. if it is the measurement a voltage consumption, the value will be displayed in volts, if it is a current consumption, it will be in ampere and if it is temperature it will be in Celsius. The other box displays the values in ADC counts, which are the values read by the ADC. An ADC converts an analog voltage to a binary number (a series of 1 and 0) and the interface convert the binary number into a decimal number. The number of binary digits (bits) representing the digital number determines the ADC resolution. In this case as a 12-bit ADC is used the resolution of this is 2^n being $n = 12$, thus the number of different output codes are $2^{12} = 4096$, and the value read by the ADC, the ADC counts, will be between 0 and 4095.

For the set of measurements, two spin boxes and a combo box are used so that the user can control what he wants to measure, how many measurements are intended and the interval between these measurements. The combo box allows the user to choose which measurement to take from those in table 3.4. The first spin box, which is accompanied by the text "Number of cycles", allows the user to define the number of desired measurements, and the minimum possible is two, otherwise it would be the same as taking an immediate measurement. The other spin box is used to define the time interval between measurements, this is defined in seconds and the minimum interval is one second.

3.2.1 Connection Between User Interface and Raspberry Pi

In order to be able to communicate between the Raspberry Pi and the Power Supplies board they must be connected and this is done via the connector, called P1 and shown in figure 3.7,

this connector is used only for the input of SPI communication signals of the Power Supplies board. Figure 3.8 shows the Raspberry Pi pin out, and for communication between the Power Supplies board and the Raspberry Pi, four of the pins on the latter must be connected to connector P1. The pins used in Raspberry Pi are: pin 13 (BCM 27) which will be connected to pin 1 of connector P1, which will serve to control the Power Supplies board select chip, pin 21 (BCM 9) that will be connected to pin 2 P1, this is a Raspberry Pi pin specially dedicated to send data and will be the SPI protocol MISO line, pin 19 (BCM 10) which will be connected to pin 3 of connector P1, which like the previous one is a dedicated pin, but in this case is to receive data, this will be the SPI's MOSI line, finally the pin 23 (BCM 11) will be connected to the pin 4 of connector P1, this being also a dedicated pin, but for the clock, being this the last line of SPI protocol, the SCLK.

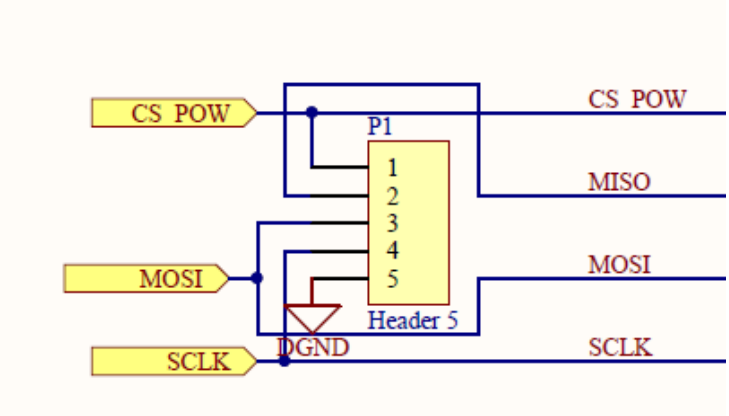


Figure 3.7: Connector used for the input of SPI communication signals.

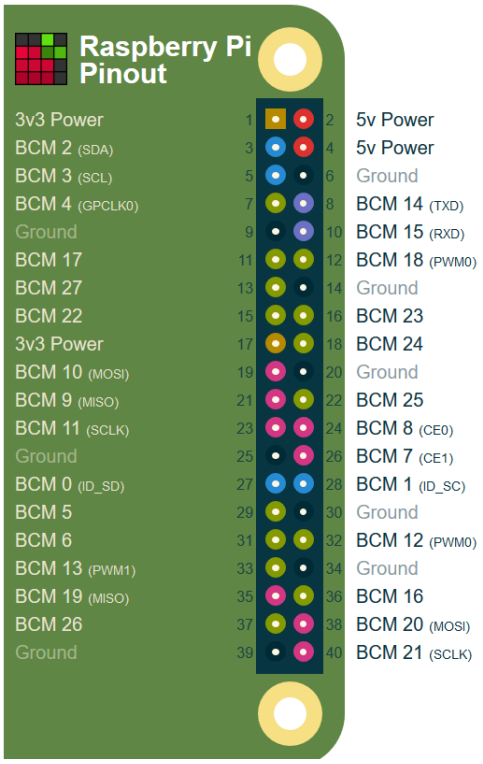


Figure 3.8: Pin out of the Raspberry Pi [30].

Once the connections are made it is then possible to use the Python GUI to control and test the Power Supplies board. The *SpiDev* library allows interfacing with SPI devices to configure the connection between the board and Raspberry Pi. In the beginning of the interface code, this library must import and declare an object, to be used throughout the code, that references the connection. The initialization of this object is made using instruction 3.1. After that it's necessary to select one set of the SPI pins that Raspberry Pi 3 B+ has, possible to see in figure 3.8 these pins are identified as having a pink color. The selection is made by using instruction 3.2, this will activate pins 19, 21 and 23. Then it's necessary to set the communication speed, that it's done using instruction 3.3.

$$spi = spidev.SpiDev() \tag{3.1}$$

$$spi.open(0, 0) \tag{3.2}$$

$$spi.max\_speed\_hz = 976000 \tag{3.3}$$

With the line of code 3.1 is created an SPI object that is designated by *spi*, and in this line of code *SpiDev* represents the use of the imported library and *SpiDev* is one of the inherent methods of this library, which does not need any parameters.

With this done, it is then possible to create an object of type expander, which will be further explored in section 3.2.2, for such is used the instruction 3.4. It creates an object with the name *mcp1*, this will need to receive 5 parameters being them the SPI object that was created on line 3.1, the device ID of the component, the Raspberry Pi pin that will be used as reset, the Raspberry Pi chip select pin and the class *GPIO*. It will be through this object that all features of the Power Supplies board will be controlled.

$$mcp1 = MCP23S17(spi, 0b000, -1, 13, GPIO) \tag{3.4}$$

The steps just described are essential for the developed GUI to work, but none of them are the ones that actually create the interface, so you need to add a few more lines of code to do this task. In figure 3.9, it's possible to see these lines of code, in the first line it is necessary to import the *sys* module, that provides access to some variables used or maintained by the interpreter and functions that interact strongly with the interpreter. In the second line is created an object called *app*, which uses the *QtWidgets* module, which provides a set of User Interface (UI) elements to create classic desktop-style user interfaces, these widgets are the primary elements for creating user interfaces in PyQt5. The *app* object need to receive some parameters, and to do so, it's used the *sys.argv* function which is a Python list, that contains the command line arguments passed to the script, this will allow widgets to use system arguments so that they can work. Next is created another object of the same type as the *app* object, called *MainWindow*, this is responsible for creating the main interface window, however if only this command existed, only a blank window would be shown to the user, thus to add the widgets to the window it's necessary to use the first two lines of the code presented in figure 3.9. In

the fourth line of code this object is assigned to a variable, denominated *ui*, a class called *Ui_MainWindow*, this is where all the widgets used in the interface are, as well as the functions assigned to them, the fact that this class exists greatly facilitates the task of organize the code. However, this class is independent of the *MainWindow* so there has to be a command that connects these two and that's what is done on line 5. With this all done it is possible to execute line 6 that displays the main window with all widgets in it. The code will stay in this line while the main window is open executing only the functions present in *Ui_MainWindow*, and as soon as the main window is closed the code will execute the last line of code that will terminate the program.

```
import sys
app = QtWidgets.QApplication(sys.argv)
MainWindow = QtWidgets.QMainWindow()
ui = Ui_MainWindow()
ui.setupUi(MainWindow)
MainWindow.show()
sys.exit(app.exec_())
```

Figure 3.9: Main code of the graphical user interface.

The *Ui_MainWindow* class has two functions that are always executed, the first is *setupUi*, this function defines the whole aspect of what is presented to the user, stipulating the size of the main window, the widgets that are present, the name, the features, their size and position. The second function is *retranslateUi*, this function is what makes the interface work, since it is responsible for assigning the functions to the widgets and doing so it makes each widget work as intended. The other functions will only be executed when the user interacts with the widget that has been assigned to it. In the later subsections, these functions will be presented by section of the interface in detail.

3.2.2 Port Expander

Although the port expander receives the serial information sent by the SPI bus it is necessary to convert it to a parallel bus, as shown in figure 3.10, remembering that on the Power Supplies board the expander that is used is the MCP23S17. In this figure it is possible to notice that the information sent by the SPI bus is converted into a parallel bus with 16 communication lines, and this process occurs inside the MCP23S17 itself.

Table 3.1 shows all signals, as well as their descriptions, which are connected to the port expander. In this it is possible to see that it is through the port expander that most of the functions present in the GUI will be performed, such as the enable/disable of the DC/DC converters and the selection of the output voltage of the HV DC/DC converters.

Communication with the port expander has been facilitated by the existence of a class specific to it and ready for use in Python. This class was developed by Filipe Cuim, who is another Master student involved in the ATLAS upgrade project, this was adapted from RPiMCP23S17

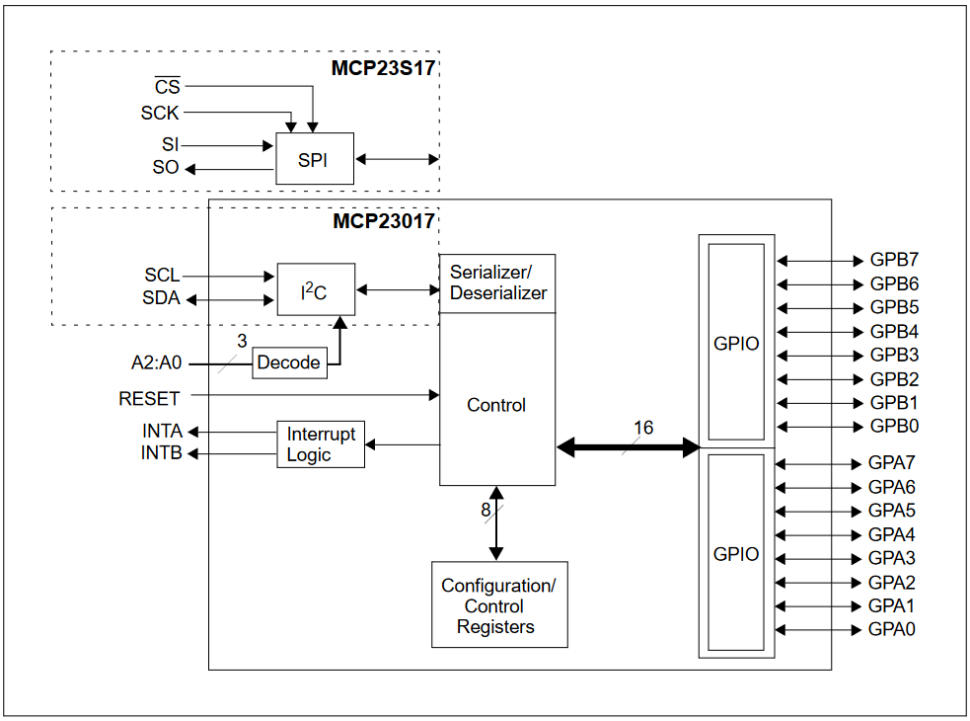


Figure 3.10: Functional block diagram of the MCP23S17 [31].

Pin number	Name of the pin	Name	Descreption
21	GPA0	CS_ADC	Active-Low Chip Select pin of the ADC.
22	GPA1	EN_MUX	Active high digital input pin of the multiplexer.
23	GPA2	MUX_0	Address line A0 of the multiplexer.
24	GPA3	MUX_1	Address line A1 of the multiplexer.
25	GPA4	MUX_2	Address line A2 of the multiplexer.
26	GPA5	MUX_3	Address line A3 of the multiplexer.
27	GPA6	CS_HV2	ON/OFF input pin of the HV2 DC/DC converter.
28	GPA7	HV2_Switch	HV output selection pin of the HV2 DC/DC converter.
1	GPB0	CS_V12	RC pin of the $\pm$12 V DC/DC converter (On/Off function).
2	GPB1	CS_V3.3	RC pin of the 3.3 V DC/DC converter (On/Off function).
3	GPB2	HV1_Switch	HV output selection pin of the HV1 DC/DC converter.
4	GPB3	NC	Not connected
5	GPB4	NC	Not connected
6	GPB5	NC	Not connected
7	GPB6	NC	Not connected
8	GPB7	CS_HV1	ON/OFF input pin of the HV1 DC/DC converter.

Table 3.1: Pin out of the expander with correspondent signals and their description.

which is an open source module to access GPIO expanders MCP23S17 from a Raspberry Pi. This class provides an abstraction of the GPIO expander for the Raspberry Pi. It is dependent on the Python packages *spidev* and RPi.GPIO, and each object receives the SPI connection and GPIO lines for the chip selects. Table 3.2 below describes the methods used to perform the interface.

MCP23S17()	
Description	Expander initialization as an object.
Syntax	object_name = MCP23S17(spidev.SpiDev(), device_id, pin_reset, chip_select, GPIO)
Keyword arguments	device_id: The device ID of the component, i.e., the hardware address (default 0), pin_cs: The pin of the Raspberry Pi that will be used as chip select of the MCP, pin_reset: The pin of the Raspberry Pi that will be used as reset of the MCP
Return	none
setDirection()	
Description	Sets the direction for a given pin.
Syntax	object_name.setDirection(pin, direction)
Keyword arguments	pin: The pin index (0 - 15) direction: The direction of the pin (Input = 0, Output = 1)
digitalWrite()	
Description	Sets the level of a given pin.
Syntax	object_name = digitalWrite(pin, level)
Keyword arguments	pin: The pin index (0 - 15) level: The logical level to be set (LEVEL_LOW = 0, LEVEL_HIGH = 1)
readGPIO()	
Description	Reads the data port value of all pins.
Syntax	object_name.readGPIO()
Keyword arguments	none
Return	The 16-bit data port value.
reset()	
Description	Sets the object parameters to their default values.
Syntax	object_name.reset()
Keyword arguments	none

Table 3.2: Description of methods inherent to the expander class.

To work with this class, it's necessary to follow a few steps first such as opening the SPI communication line and set of communication polarity as well as the phase. It is also necessary to initialize the GPIOs and select which object we want to communicate with. Without these steps being performed, communication is impossible, since the class MCP23S17 is dependent on them, however as soon as they are executed the port expander is ready to start commu-

nication. These steps will be best presented during the description of the code used in the GUI.

3.2.3 ADC and Multiplexer

In the case of communication with ADC, the code used was also authored by Filipe Cuim, however this one is much simpler than the previous one having only one function, besides the constructor, which is used in the graphic interface development, represented in table 3.3.

MAX1240()	
Description	ADC initialization as an object.
Syntax	object_name = MAX1240(spidev.SpiDev(), SHDN, chip_select, MCP)
Keyword arguments	MCP: Expander object that must be previously created. SHDN: three-level shutdown input. When using external reference, must be open. chip_select: pin of the MCP23S17 port expander, to be used as chip select.
Return	none
readVoltage()	
Description	Reads the voltage as a word of 12 bits. Bytes are sent with MSB first, and bits are clocked at the rising edge o SCLK.
Syntax	object_name.readVoltage()
Keyword arguments	none
Return	Returns an integer less than 4096.

Table 3.3: Description of methods inherent to the ADC class.

The Multiplexer does not need a special class because the programming in which it is involved is quite simple. Programming the multiplexer only involves activating the address line pins and the active high digital input pin, which will allow reading of a specific multiplexer pin. However, this component will be of utmost importance in the Power Supplies board since this is where all current and voltage consumptions as well as the two temperatures voltage signals are connected, because the use of this component reduces the cost of implementing separate channels for each data source. The activation of the referred pins will be done through the port expander outputs, which will be connected to the multiplexer by the same pins as shown in table 3.1.

3.2.4 Enable/Disable of DC/DC Converters

The section called DC/DC Enable, represented in figure 3.11, is responsible, as the name suggested, of the digital enable/disable of the DC/DC converters. In this section, four check boxes are used to do this, one for the 3.3 V converter, one for the ± 12 V converter and another for each HV converter. To activate a converter the user only has to select the respective checkbox, and to deactivate he just needs to perform the inverse process. Interacting with these check boxes will execute the function that is connected to them, which is represented in figure 3.12.

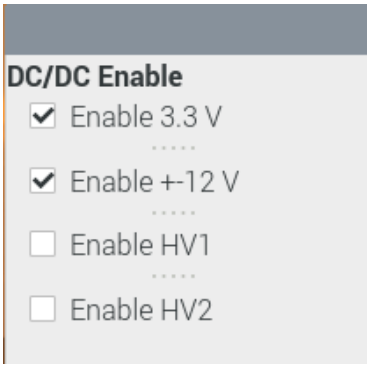


Figure 3.11: Section responsible for enable/disable of DC/DC converters.

```
def Enable(self):

    if self.checkBox.isChecked() == True:
        mcp1.digitalWrite(2, MCP23S17.LEVEL_HIGH)
        time.sleep(0.5)
    else:
        mcp1.digitalWrite(2, MCP23S17.LEVEL_LOW)
        time.sleep(0.5)

    if self.checkBox_2.isChecked() == True:
        mcp1.digitalWrite(1, MCP23S17.LEVEL_HIGH)
        time.sleep(0.5)
    else:
        mcp1.digitalWrite(1, MCP23S17.LEVEL_LOW)
        time.sleep(0.5)

    if self.checkBox_3.isChecked() == True:
        mcp1.digitalWrite(8, MCP23S17.LEVEL_HIGH)
        time.sleep(0.5)
    else:
        mcp1.digitalWrite(8, MCP23S17.LEVEL_LOW)
        time.sleep(0.5)

    if self.checkBox_4.isChecked() == True:
        mcp1.digitalWrite(27, MCP23S17.LEVEL_HIGH)
        time.sleep(0.5)
    else:
        mcp1.digitalWrite(27, MCP23S17.LEVEL_LOW)
        time.sleep(0.5)
```

Figure 3.12: Enable/disable DC/DC section code.

This function is neither too complicated nor too long, as it just consists in 6 lines of code that are repeated, with their appropriate changes, for each checkbox. The first line of this code consists of an *if* statement that checks whether or not a checkbox is selected. For this the *isChecked()* function is used which returns *True* if the checkbox is selected and *False* if not. If the checkbox is selected, the *digitalWrite()* function inherent to the expander object will be executed, but as referred in table 3.2, it will need to receive the pin corresponding to the associated DC/DC converter and the variable LEVEL_HIGH of class MCP23S17, this will make the referred pin have a voltage of 3.3 V thus activating the converter connected to it. Next, because this step is not immediate, the sleep function is used causing the code to stop for 0.5 seconds, making this it is possible to prevent any conflicts that could be caused by sending too many instructions to the expander without it having time to finish the ones that were already running.

When it is desired to disable a DC/DC converter, the user has only to deselect a previously selected checkbox, doing this the *isChecked()* function will return *False* thus entering in the else case, shown if figure 3.12. In this case what happens is similar to the enable case but instead of using the LEVEL_HIGH variable, it's used the LEVEL_LOW variable, also from the class MCP23S17, that makes the selected pin have a voltage of 0 V, ending again with the sleep function, after that.

3.2.5 Set the Output Voltage of HV Converters

As already mentioned, it is in this section that it is possible to control the voltage of the two HV DC/DC converters that are used on the Power Supplies board. In figure 3.13 shows the set HV section where it's used two sliders, one for each HV converter, to control these voltages, so it's only necessary to slide the slider to change the output voltage. These sliders have only two positions, one for -830 V and one for -930 V, and are accompanied by informative text that changes according to the position of the slider to make this section easier to understand. The function that is used in this section is represented in figure 3.14.

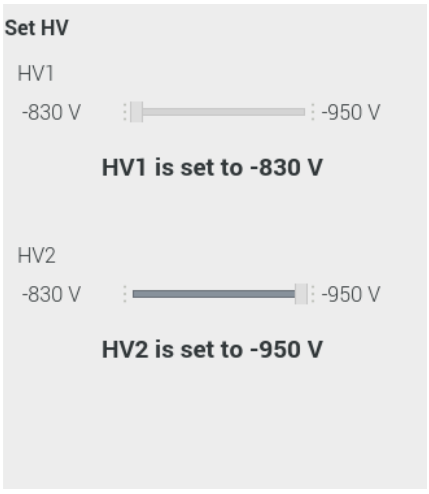


Figure 3.13: Section responsible for selecting the output voltage of HV converters.

The *HVSel* function, like the function mentioned earlier, is not very extensive, it also starts with an *if* statement. However as sliders are used in this section instead of check boxes, it's needed to use a different function to determine the cursor position, this is done by using the *value()* function that returns the value of the slider cursor position. In this case the position 0 corresponds to -830 V and the position 1 to -950 V, so it's only needed to use this information with the *if* statement. Inside the *if* statement, it is necessary to perform two different actions. The first one is to put in the respective port expander output the corresponding signal to change the output voltage of the respective HV DC/DC converter and the other is to change the informational text on the corresponding label, thus providing the correct information more explicitly to the user. The latter one is performed in two steps, the first is to erase the label information and the second is to write the correct information to that same label, so to do that it's used the *setText(" ")* function that programmatically sets the caption in the label. To perform the HV DC/DC converter output voltage selection task, the *digitalWrite()* function is used, as in the subsection 3.2.4, but in this case the LEVEL_HIGH variable is for when the output voltage is -830 V and LEVEL_LOW for when the output voltage is -950 V.

```
def HVSel(self):

    if self.HV1.value() == 0:
        self.label_14.setText("")
        mcp1.digitalWrite(3, MCP23S17.LEVEL_HIGH)
        self.label_14.setText("HV1 is set to -830 V")
        time.sleep(0.5)
    else:
        self.label_14.setText("")
        mcp1.digitalWrite(3, MCP23S17.LEVEL_LOW)
        self.label_14.setText("HV1 is set to -950 V")
        time.sleep(0.5)

    if self.HV2.value() == 0:
        self.label_15.setText("")
        mcp1.digitalWrite(28, MCP23S17.LEVEL_HIGH)
        self.label_15.setText("HV2 is set to -830 V")
        time.sleep(0.5)
    else:
        self.label_15.setText("")
        mcp1.digitalWrite(28, MCP23S17.LEVEL_LOW)
        self.label_15.setText("HV2 is set to -950 V")
        time.sleep(0.5)
```

Figure 3.14: Set HV section code.

3.2.6 Reading Functionality

It is in this section that measurements of current and voltage consumptions as well as temperatures can be performed. As it's possible to see in figure 3.15, these can be done in two different ways, a single and immediate reading or a set of readings that are presented in a graph. Immediate measurements are made through 12 different radio buttons, a more detailed explanation of what each radio button reads is found in table 3.4. The use of this type of button only allows reading one quantity at a time, which is one of the limitations of this board since only one ADC is used. The result of these measurements is presented in two boxes, the leftmost one shows the ADC counts directly and the other box shows the value in the corresponding units, i.e. the voltage consumptions are presented in units of Volt, current in Amperes and the temperatures in Celsius.

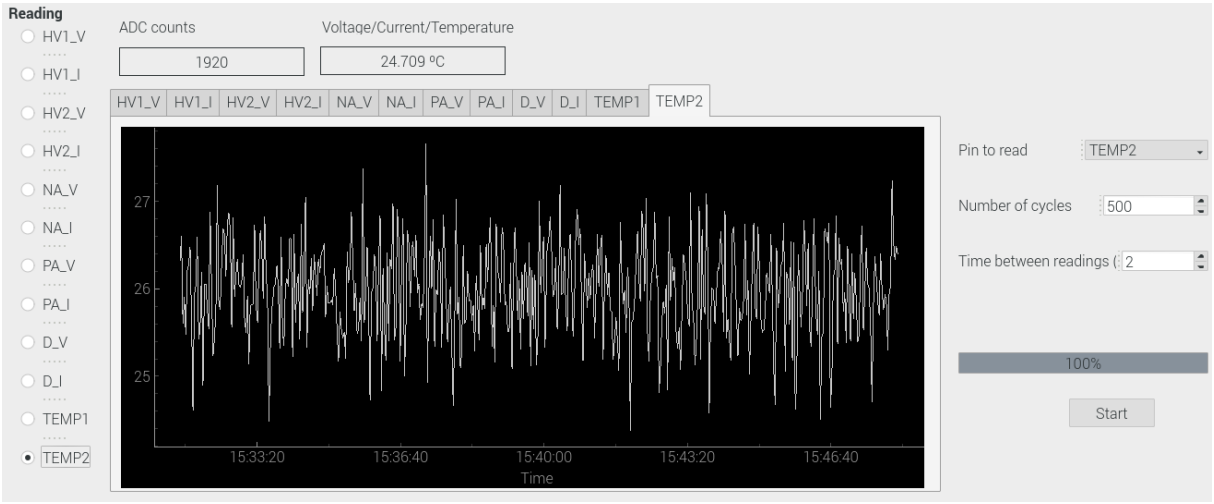
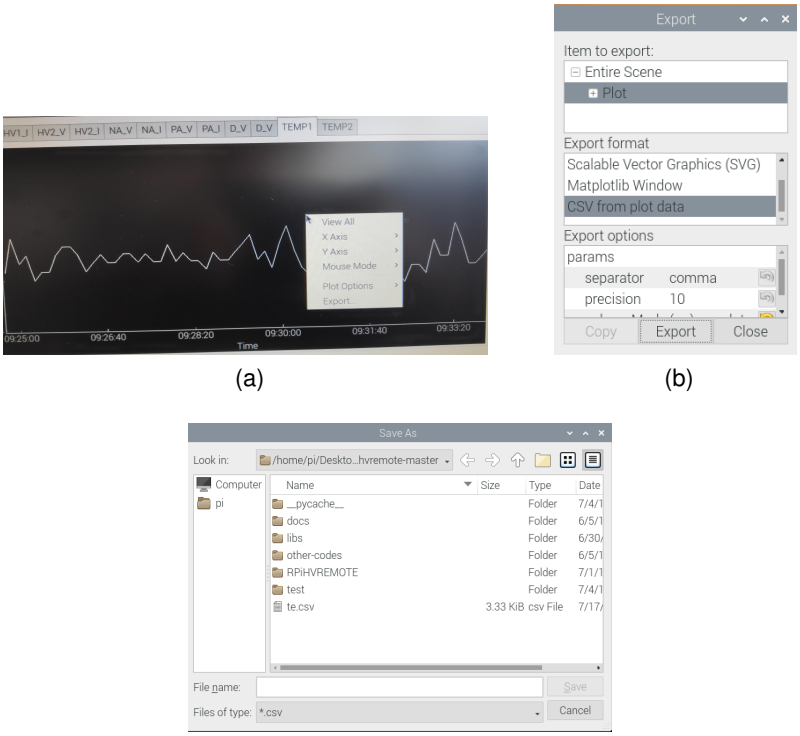


Figure 3.15: Section responsible for reading DC/DC converter consumptions and temperatures.

For the set of measurements, it's possible to choose what it's wanted to measure with a combo box, and the options that can be chosen in this are those in table 3.4. It is also possible to establish how many measurements are made using a spin box, that corresponds to the number of cycles in the interface, and the minimum number of iterations is two, otherwise it would be the same as taking an immediate reading. The time interval between measurements is also defined by the user using a spin box, which defines in seconds the interval between one measurement and the next one. In this spin box, the shortest interval between measurements that can be set is 1 second, so there is time for each measurement to be performed successfully. In order to present the measurements made, a graph is used, each quantity has its own graph and they are all grouped in different separators, as you can see in figure 3.15, they are presented in the unit of the measured quantity as a function of the date of the measurement. Each time a set of measurements is taken, the graph is automatically selected, i.e. the separator is selected according to the measurement made. However, the user can choose which tab to see after it is automatically changed to the one of the chosen measurements.

This graph also has other advantages for the user, such as the possibility for the user to be able to control the position of the axis, the axis scale and even offers the possibility to save the data in a *csv* format file. To save the data in this type of file, the user just has to click no the graph with the right mouse button, then choose to export which will open a new window. This new window will allow the user to choose the file type in which he want the data to be stored, then it's only necessary to choose the name of the file and location where it's intended to save it, these steps are shown in figure 3.16. This will allow a deeper analysis of the data obtained since this interface is not equipped with the ability to do so, however this analysis will be extremely important on the board's forehead.

The functions that are performed in this section are quite extensive to be presented in their entirety, so they will be divided to facilitate their explanation. Before starting with the code, there is one thing to keep in mind, as previously mentioned, the measured values of consumptions of the DC/DC converters and the temperatures, before being read by the ADC, are connected to an analog multiplexer as such to make a measurement it's needed to choose the correct



(a) (b)

(c)

Figure 3.16: Steps required for the user to save the data in a csv file. (a): First step: Right-click on the graph. (b): Second step: Select the file type as csv and click on export. (c): Third step: Choose the location and file name.

channel of the multiplexer. This choice is made through address line pins of the multiplexer that are directly connected to the port expander output signals, A0, A1, A2 and A3, and it is the combination of these signals that allows the choice of channels, as can be seen in the table 3.4. In addition to these 4 pins, there is yet another multiplexing pin, the active high digital input pin, which must also be activated for the multiplexer to be used.

In the first part of the immediate read function, called *read*, the radio buttons are interconnected with the signal that must be sent from the expander to the multiplexer, according to table 3.4. For this some *if* statements are used, as shown in figure 3.17 where an excerpt of this code is found, within these *if*'s statements is also defined the units to be represented as well as a variable named *r*, which will be used later to define what is the appropriate equation to obtain the real values.

The second part consists of activating the port expander output pin that is responsible for the multiplexer activation signal and the signals to the pins that will allow the choice of the channel that is intended to be read, for this *if* statements are used again, as well as the MCP23S17 *digitalWrite()* function, as possible to see in figure 3.18.

Finally, the value is read, although it is said to be an immediate measurement in reality each of these values are an average of 10 sequential readings. This allow to minimize the

[1] Active high digital input pin

Name	Pin number	ON-Channel	A0	A1	A2	A3	EN[1]	Description
		All channels are off	x	x	x	x	0	
HV1_V	19	S01	0	0	0	0	1	Voltage consumption reading of HV converter 1.
HV1_I	20	S02	0	0	0	1	1	Current consumption reading of HV converter 1.
NC	21	S03	0	0	1	0	1	Not connected
NC	22	S04	0	0	1	1	1	Not connected
D_V	23	S05	0	1	0	0	1	Voltage reading of the digital converter.
D_I	24	S06	0	1	0	1	1	Current reading of the digital converter.
PA_V	25	S07	0	1	1	0	1	Voltage reading of the positive output of the analog converter.
PA_I	26	S08	0	1	1	1	1	Current reading of the positive output of the analog converter.
NA_V	11	S09	1	0	0	0	1	Voltage reading of the negative output of the analog converter.
NA_I	10	S10	1	0	0	1	1	Current reading of the negative output of the analog converter.
TEMP1	09	S11	1	0	1	0	1	TMP17 temperature reading 1.
TEMP2	08	S12	1	0	1	1	1	TMP17 temperature reading 2.
NC	07	S13	1	1	0	0	1	Not connected
NC	06	S14	1	1	0	1	1	Not connected
HV2_V	05	S15	1	1	1	0	1	Voltage consumption reading of HV converter 2.
HV2_I	04	S16	1	1	1	1	1	Current consumption reading of HV converter 2.

Table 3.4: MUX36S16 multiplexer truth table associated with the measurement's options of the Power Supplies board GUI and the description of the signals.

```
def read(self):
    if self.radioButton.isChecked() == True:
        a = "0000"
        s = " V"
        r = 0
    if self.radioButton_2.isChecked() == True:
        a = "0001"
        s = " mA"
        r = 1
    if self.radioButton_3.isChecked() == True:
        a = "1110"
        s = " V"
        r = 0
    if self.radioButton_4.isChecked() == True:
        a = "1111"
        s = " mA"
        r = 1
    if self.radioButton_5.isChecked() == True:
        a = "1000"
        s = " V"
        r = 2
    if self.radioButton_6.isChecked() == True:
        a = "1001"
        s = " mA"
        r = 3
    if self.radioButton_7.isChecked() == True:
        a = "0110"
        s = " V"
        r = 4
```

Figure 3.17: Selection of measure to be performed of the immediate read code.

```
mcp1.digitalWrite(22, MCP23S17.LEVEL_HIGH)

if int(a[3]) == 0:
    mcp1.digitalWrite(23, MCP23S17.LEVEL_LOW)
    time.sleep(0.5)
else:
    mcp1.digitalWrite(23, MCP23S17.LEVEL_HIGH)
    time.sleep(0.5)
if int(a[2]) == 0:
    mcp1.digitalWrite(24, MCP23S17.LEVEL_LOW)
    time.sleep(0.5)
else:
    mcp1.digitalWrite(24, MCP23S17.LEVEL_HIGH)
    time.sleep(0.5)
if int(a[1]) == 0:
    mcp1.digitalWrite(25, MCP23S17.LEVEL_LOW)
    time.sleep(0.5)
else:
    mcp1.digitalWrite(25, MCP23S17.LEVEL_HIGH)
    time.sleep(0.5)
if int(a[0]) == 0:
    mcp1.digitalWrite(26, MCP23S17.LEVEL_LOW)
    time.sleep(0.5)
else:
    mcp1.digitalWrite(26, MCP23S17.LEVEL_HIGH)
    time.sleep(0.5)
```

Figure 3.18: Activation of the address pin and the enable pin used to activate in the immediate read code.

noise associated with the measurement of the reading values. In this part of the code it is necessary to create two arrays that served to store the data obtained from the ten successive measurements. These arrays will be called *madc*, which will be used to store the data read directly from the ADC, that is, the ADC counts, and *m*, which will store the real values of the measurements, and these real values will be obtained through the formulas presented in the table 3.5 and later presented with the appropriate units. Then it is defined the *for* cycle, responsible for the ten successive measurements. Within this *for* cycle the function that reads the ADC, *readVoltage()* function from the class MAX1240, is then executed, this measurement will be done in ADC counts but can easily be converted to a voltage by dividing this value by the maximum number of bytes, 4096, and multiplying by the reference voltage, 3.3 V as possible to see in table 3.5. After this and with the variable *r* defined above, it is possible to use the appropriate equation to obtain the appropriate value of the measurement being taken, getting the correspondence between measure and equation as it is represented in the table 3.5. This value will be assigned to one of the arrays created at the beginning of this part and the other will contain the value read directly from the ADC. Before the end of the cycle, the program must deactivate the pins used, for that is used an *if* statement that is only executed in the last iteration of the cycle, in this, the address line pins and the enable pin used to activate and choose the channel of the multiplexer will be deactivated, resorting to the *digitalWrite()* function.

To finish this function just need to have the average of the measured values printed in the text boxes, this is done with the *setText* instruction, previously used in the HV DC/DC converter output voltage selection code. The average is done using the *mean* function of the statistics library together with the *round* function to limit the number of decimal places (digits) to be printed, and in the second box is where the value with physical units will be displayed is necessary to add to this box the unit of the value and for that is used the variable *s* defined at the beginning.

Name	Equation	
ADC counts to voltage	$Volt = \frac{adc.readVoltage() \times 3.3}{4096}$	(3.5)
HV1_V (V)	$value = \frac{Volt}{0.005}$	(3.6)
HV2_V (V)		
HV1_I (A)	$value = \frac{Volt}{0.5}$	(3.7)
HV2_I (A)		
NA_V (V)	$value = \frac{43 \times Volt}{2979750}$	(3.8)
NA_I (A)	$value = \frac{Volt}{5 \times 200}$	(3.9)
PA_I (A)		
D_I (A)		
PA_V (V)	$value = \frac{Volt \times 429}{89}$	(3.10)
D_V (V)	$value = \frac{Volt \times 139}{105}$	(3.11)
TEMP1 (ºC)	$value = \frac{Volt * 1000000}{1000} - 273.15$	(3.12)
TEMP2 (ºC)		

Table 3.5: Table of the equation that are used in the GUI code to convert the ADC counts in the real value.

To make a set of measurements, the code used has some differences from the previous one, and these are mainly due to the fact that we want to do more than one measure. Figure 3.19 shows a flowchart that explains the operation of the *plotting* function that is used to make these sets of measurements.

In the *plotting* function the way the measurement we want to make is different, since it is used a combo box instead of multiple radio buttons, which means that the signal that must be sent to the multiplexer will be given by the current index selected in the combo box, as shown in figure 3.20, then the address lines and the chip select pins of the multiplexer are activated as in the *read* function, figure 3.18.

The core of the measurement is also the same as the *read* function, however in this case it is within another *for* cycle and there is a need to create other variables to control the number of iterations of this cycle, the time between each iteration as well as two arrays to store the measured values, as represented in figure 3.21. It is possible to observe here that the first step is to assign to two variables the values defined by the user in the spin box that controls the number of cycles, i.e., the number of measurements that will be made and in the spin box that controls the time interval between iterations. Two arrays are created to store the data and the progress bar is set, which will inform the user of the percentage completed of the measurements. After the first cycle is defined, it's assigned to the *x* array the current time, this will be the x axis of the graph. It's also created another array, *madc*, that will be used to store the values that will be measured by the ADC. The value that the *madc* will store are the 10 measurements already converted in volt using again the equation on table 3.5. After these 10 measurements are made, it will be used the other equations in table 3.5 to obtain the real values of the measurements. These values will be stored in the variable k, which will be the y

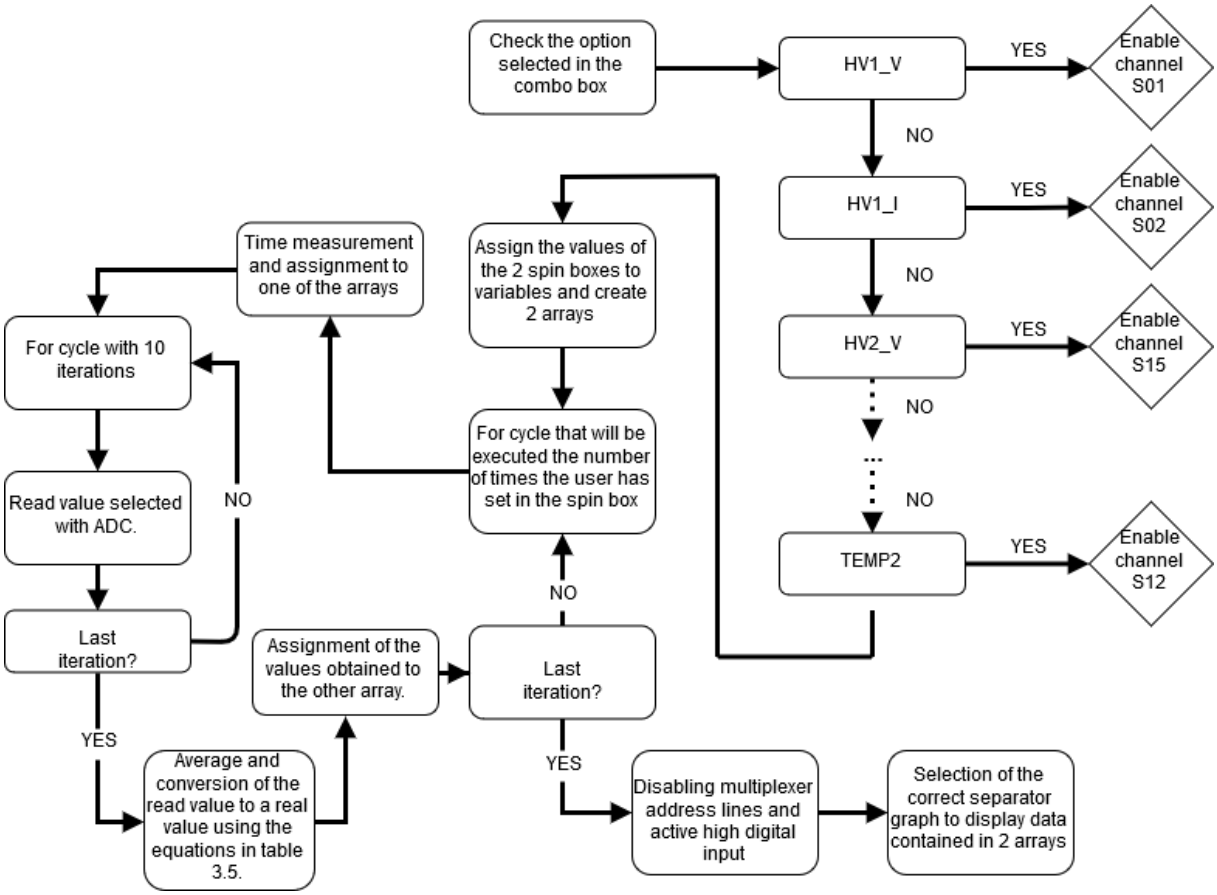


Figure 3.19: Flowchart of the plotting function used for reading and plotting a set of measurements.

```
def plotting(self):
    t = self.comboBox.currentIndex()
    if t == 0:
        a = "0000"
    if t == 1:
        a = "0001"
    if t == 2:
        a = "1110"
```

Figure 3.20: Selecting the multiplexer channel to be activated, depending on the desired measurement, in the measurement set reading code.

axis of the graph. Then the progress bar will be updated, and the *sleep* function is used which will stop the code for the amount of time that has been set by the user in the spin box. In the last iteration of the first *for* cycle, represented in figure 3.20, the address lines and the chip select pins of the multiplexer are then deactivated.

```
u = self.spinBox.value()
s = self.spinBox_2.value()
k = np.zeros(u)
x = np.zeros(u)
self.progressBar.setValue(0)
self.progressBar.setMaximum(u)

for i in range(u):
    x[i] = int(time.mktime(datetime.datetime.now().timetuple()))
    madc = np.zeros(10)
```

Figure 3.21: Assigning values in spin boxes to variables, creating arrays to store values, defining the first for cycle and measuring of the time within the last one.

Finally, in the figure 3.22 is represented the code used for the measured values to be displayed in the respective graphs. For this, the combo box index is used to select the correct separator graph to display the data, then the *plot*(x, k) function is used displaying the measured quantity as a function of the time it was measure.

```
if t == 0:
    self.graphicsView.clear()
    self.graphicsView.plot(x,k)
if t == 1:
    self.graphicsView_2.clear()
    self.graphicsView_2.plot(x,k)
if t == 2:
    self.graphicsView_3.clear()
    self.graphicsView_3.plot(x,k)
```

Figure 3.22: Code lines responsible for plotting the data in the corresponding graphs in the measurement set reading code.

Chapter 4

Functional Tests

As previously mentioned, the purpose of this thesis is to develop a power supply card dedicated to HV Remote board that will produce both high and low voltage supplies with low noise. Therefore, it is important to perform some tests on the Power Supplies board once it is produced in order to ensure that the voltages produced are stable and not too noisy. Such tests will be executed as soon as the first prototype of the Power Supplies board is fabricated, and they will evaluate the following aspects:

- Stability of the produced output voltages – DC/DC converters performance;
- ADC converter performance;
- Output voltage noise;
- Burn-in effect - the board will be placed in an oven at 65 ^{o}C for one week, during which the functional tests will be performed.

The main objectives of these tests are to verify whether the generated voltages are the intended, to quantify the voltages noise and to verify that the board can withstand continuous operation, guaranteeing stability and specified parameters.

However, DC/DC converters are not the only components that will have to be tested. Components such as the port expander, the multiplexer and the ADC used also need to be tested as it is through these that the Power Supplies board will be controlled and the consumptions and temperatures will be read. There is one other thing that needs to be tested, the graphical user interface developed in Python specifically for this board, as it will allow you to test the aforementioned components faster and easier.

This chapter will present the process performed to test the developed GUI, and consequently the multiplexer, the port expander and the ADC used in this process. The tests that will be performed to the ADC will also be presented, and these will be more complex and more accurate than those already performed during the GUI test. Finally, it will be detailed the tests that will be performed on the Power Supplies board as soon as its manufacture is complete.

4.1 Graphical User Interface Test

Once the GUI is finished, the next logical step will be to test it. To do this, a port expander, an ADC and a multiplexer were mounted on a breadboard as shown in figure 4.1.

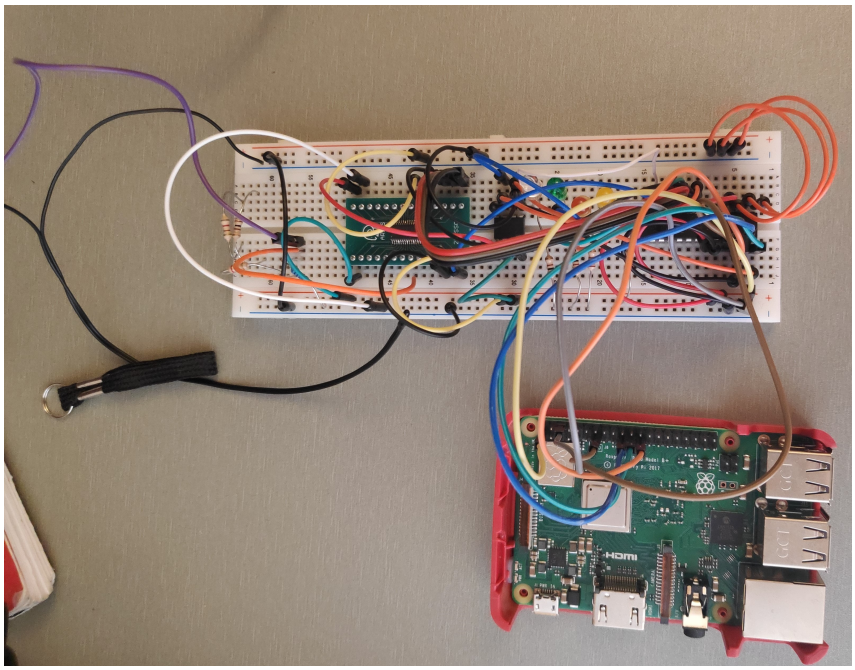
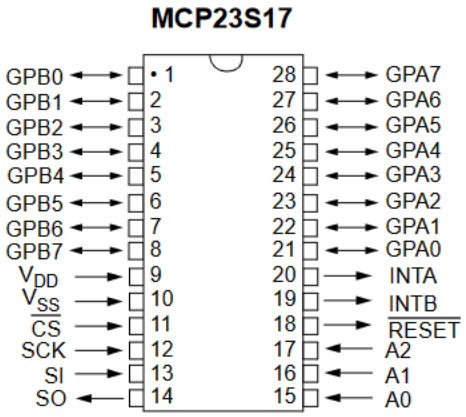


Figure 4.1: Assembly used for GUI, multiplexer, port expander, and ADC testing.

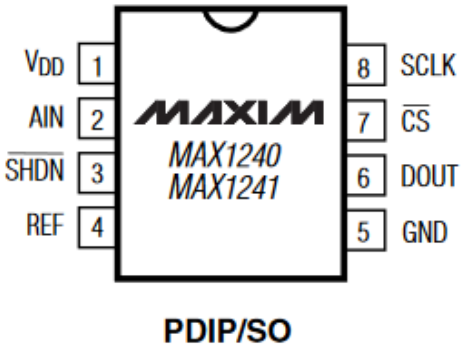
In the figure 4.1 it is also possible to observe a Raspberry Pi 3 B+ that was also used during the test of the GUI developed. Raspberry Pi was connected directly to the port expander, as there is no specific control signal connector on the breadboard as there is on the Power Supplies board, meaning that the connections will be made differently from what was presented in section 3.2.1 of Chapter 3. The pins of each component used are shown in figure 4.2 and the pin out of Raspberry Pi in figure 3.8 of Chapter 3.

The connections made on the breadboard are shown in figure 4.3 and these are divided by color, where the connections that are connected to the Raspberry Pi are the yellow ones that correspond to the SPI protocol SCLK line, the orange connections correspond to the MISO line of the SPI, the green that corresponds to the SPI MOSI line and the pink that corresponds to the chip select of the port expander. In this case, as there is no external power supply, it was necessary to use one of the Raspberry Pi 3.3 V pins, represented in figure 4.3 by red lines, as well as a ground pin, represented by black lines. The connections between the components are also represented in this figure, and the dark blue connections are the connections between the expander and the address lines of the multiplexer, the light blue the connection of the port expander to the chip select pin of the multiplexer and the purple the link between the port expander and the chip select of pin of the ADC

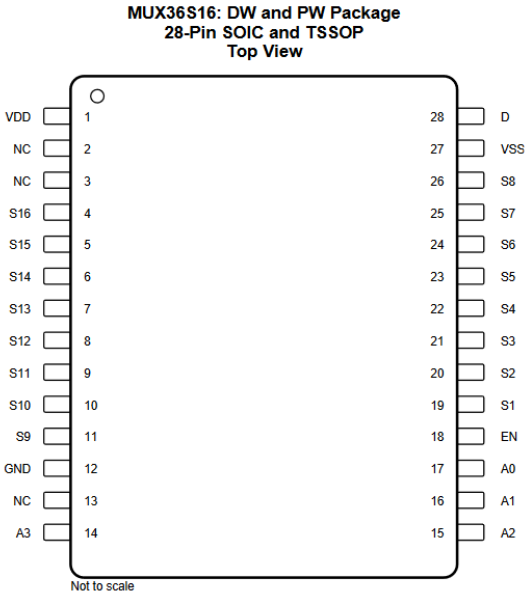
It is also possible to notice that in this figure no capacitor coupled to any component is found, this was mainly due to the unavailability of suitable capacitors during the testing phase. This



(a) Expander MCP23S17 pin out [32].



(b) ADC MAX1240 pin out [33].



(c) Multiplexer MUX36S16 pin out [31].

Figure 4.2: Pin out of the components used in conjunction with Raspberry Pi in the GUI test.

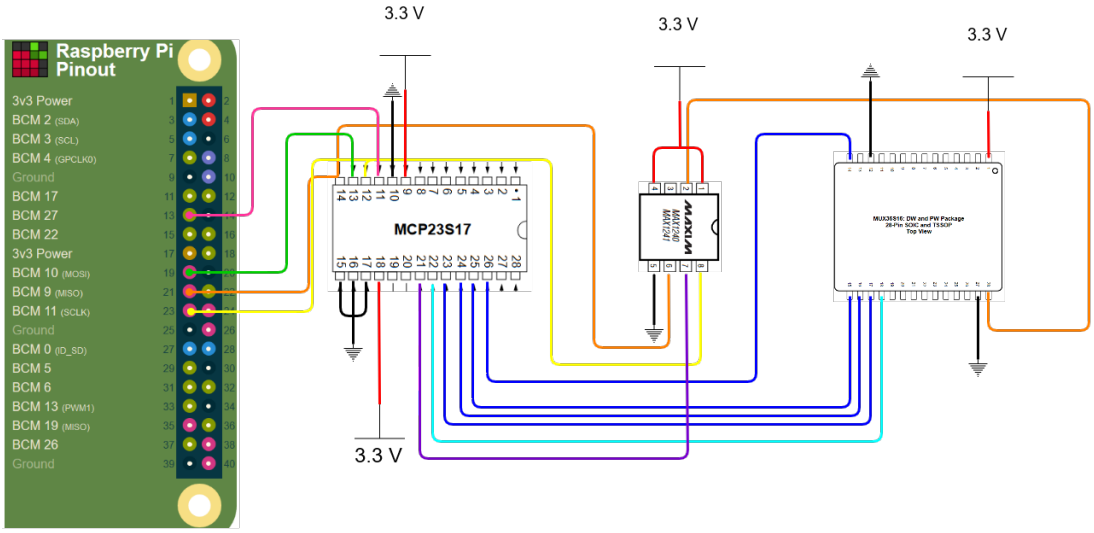


Figure 4.3: Connections of the assembly used for GUI, multiplexer, port expander, and ADC testing.

caused ADC readings to be much noisier than desired, however as the main objective of this phase was to test the GUI and some basic functionalities of the other 3 components, the end result is not greatly affected by this factor, this setback also showed the importance of capacitors for noise minimization.

The first sections of the GUI to be tested were the section responsible for the enable/disable of the DC/DC converters and the section for selecting the output voltage of the HV DC/DC converters. These two sections allow the port expander (MCP23S17) to be tested, since it consists of making the output register of port expander to had the correct chip select signals to activate the DC/DC converters, according to table 3.1 in Chapter 3, through the SPI protocol, and confirming that the signal was sent using a digital multimeter. This process was repeated several times, sending signals to enable and disable the DC/DC converters as well as signals to change its output voltage, always confirming the correct execution of these instructions.

The consumption and temperature reading section allows you to test basic ADC and multiplexer functions that have not been tested in the other two sections, however the communication with the port expander is also retested in this section. In order to be able to read any value by the ADC it is necessary that in the multiplexer channels there is something connected, for this a thermistor was used in pin 8, which corresponds to the channel S12. It was used as a substitute of the TMP27, and also used to test the other multiplexer input signals, presented in table 3.4 of the Chapter 3, with different voltage values, as can be seen as an example in figure 4.4.

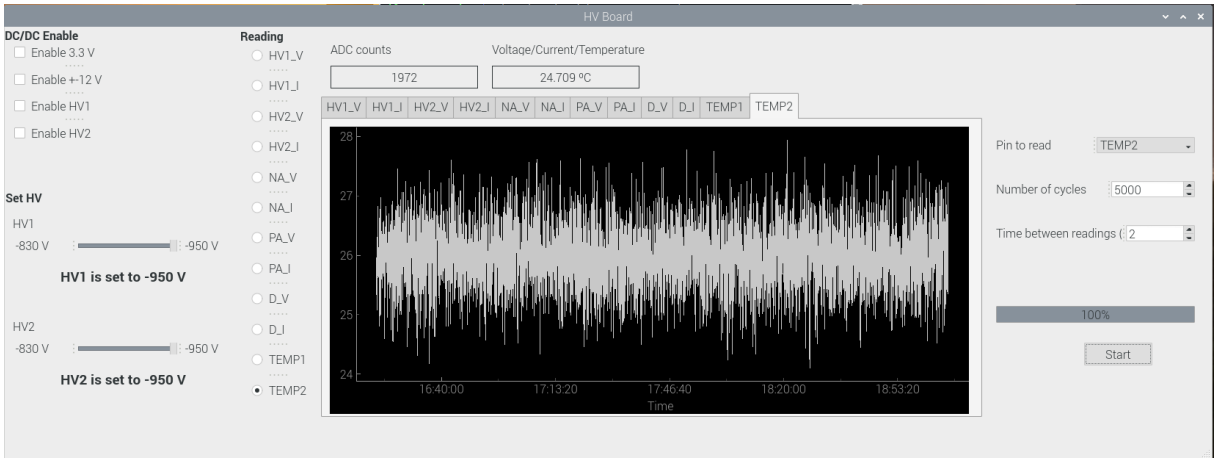


Figure 4.4: Temperature measurements for testing the graphical user interface performed on a day with an average temperature of 26.49 ºC.

With this done, it was possible to take measurements through the immediate measurement option obtaining values that were within the expected. Temperature measurements were obtained on a day when the average temperature was 26.48 ºC, and according to the data obtained by the thermistor the temperature was 24.709 ºC, as observed in figure 4.4. Has these measurements were repeated sometimes it was observed a fluctuation in the readings even when the voltage consumptions were measured, which should be stable once were obtained from a constant voltage, however as already mentioned this problem was already expected since no capacitors were used to minimize noise but it was still possible to test that the im-

mediate read command worked as it was expected as well as multiplexer channel selection and the ADC's ability to convert to a digital one the input analog value.

To test the option of reading a set of measurements, only the information from the thermistor was used, since the voltage consumption reading's information come from a stable voltage, this means that making a measurement over time of these values would be meaningless. Figure 4.4 shows the data obtained from a set of measurements of the values obtained by the thermistor, with 5000 points measured at intervals of 2 seconds between each measurement. This data was later stored in a CSV file to enable further analysis of the data obtained, as shown in figure 4.5. In this data it was possible to see that there is a clear fluctuation of temperatures from 24.101 ºC to 27.938 ºC, however it is possible to observe that the command responsible for executing the instructions that will make a set of measurements works as intended.

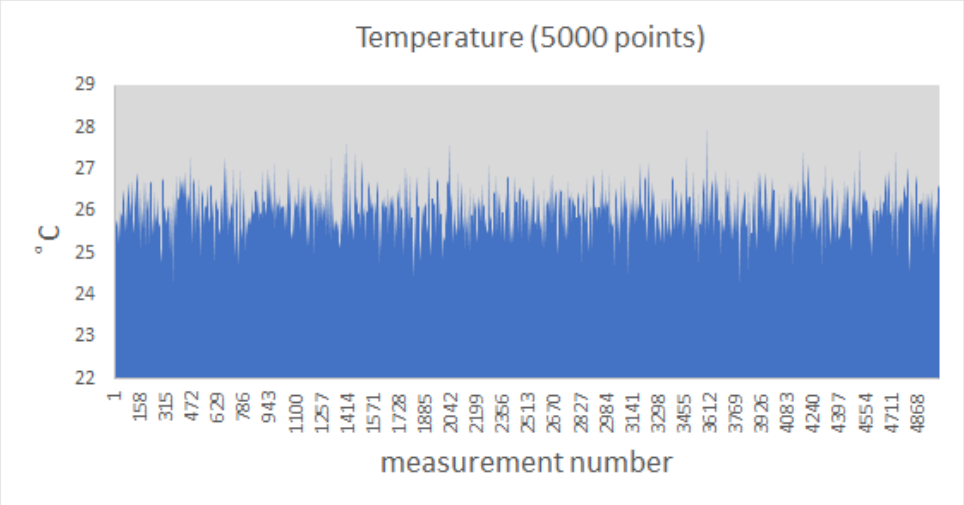


Figure 4.5: Data obtained by a thermistor that performed 5000 cycles at intervals of 2 seconds.

4.2 ADC Static Tests

The ADCs are components that have several factors that can cause errors during their use, however there are ways to mitigate or even correct these errors, but first it is necessary to identify and quantify these. For this, static tests should be performed on the ADC (MAX1240), which will allow this quantification and subsequently the minimization of errors presented by the ADC in order to obtain more accurate measurements. A test is static when the input signal to the ADC under test is varied slowly and a ramp is normally chosen as the input signal for this type of tests. As the ADC chosen just had a surface mount version it was not possible to make these tests in the breadboard in time to the presentation of this work. By the other hand, the important is to test this component in the Power Supplies board to evaluate if its noise decoupling capacitors are close enough and if ground and power planes are sufficient, and that will be made as soon as the board arrives.

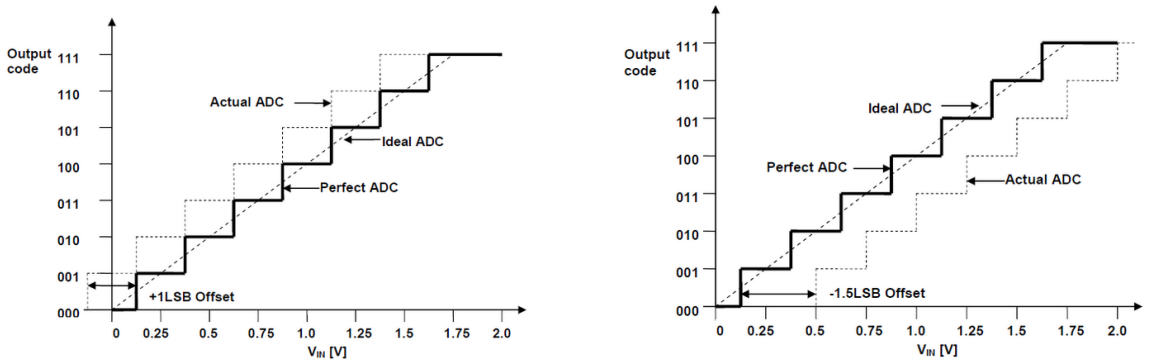
4.2.1 ADC Offset Error

The ADC offset error is given by the deviation between the ADC transfer function in use and the perfect zero point ADC transfer function for the transition measured in the Least Significant Bit (LSB), expressed in voltage in equation 4.1, where N is the number of bits of the ADC. Offset error occurs when the transition from 0 to 1 output value does not occur with an input value of 0.5 LSB, and this offset error can be positive or negative. It is positive when the output value is greater than 0 and the input voltage is less than 0.5 LSB, as shown in figure 4.6a, and is positive or negative when the input value is bigger or smaller than 0.5 LSB when the first transition occurs. This error can be easily compensated once it is quantified. In expressions 4.2 and 4.3 the offset errors of the figures 4.6a and 4.6b, respectively, are quantified [34][35].

$$1LSB = \frac{V_{Ref+} - V_{Ref-}}{2^N} \tag{4.1}$$

$$ADC_{Perfect}–ADC_{Real} = 1.5 \text{ LSB}–0.5 \text{ LSB} = +1 \text{ LSB} \tag{4.2}$$

$$ADC_{Perfect}–ADC_{Real} = 0.5 \text{ LSB}–2 \text{ LSB} = -1.5 \text{ LSB} \tag{4.3}$$



(a) ADC positive offset error [34].

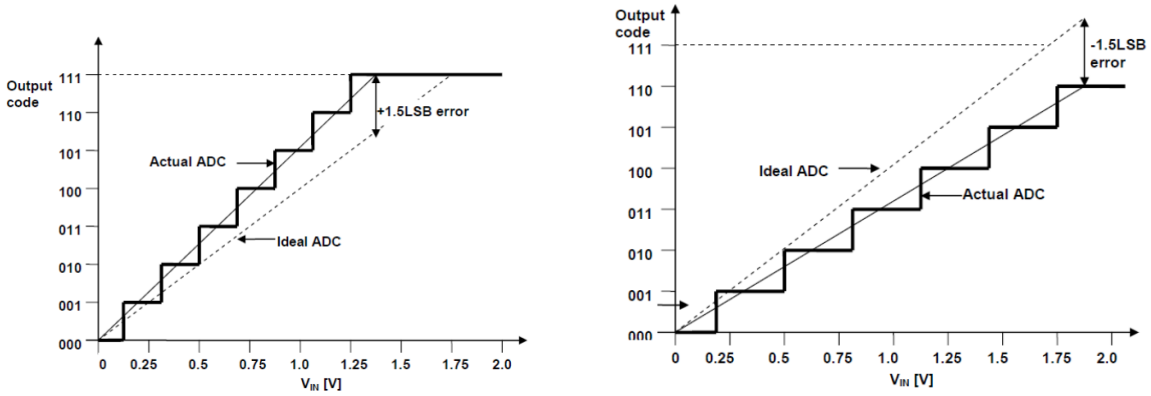
(b) ADC negative offset error [34].

Figure 4.6: ADC offset error.

4.2.2 ADC Gain Error

The ADC gain error is given by the deviation between the midpoint of the last step of the actual ADC and the midpoint of the last step of the ideal ADC, after the offset error compensation. Applying an input voltage 0 will result in an output value 0, however, the gain errors cause the actual transfer function to tilt causing it to deviate from the ideal slope. It is possible to measure and compensate for this error by scaling the output values [36][35].

In 4.7 two examples of a 3-bit ADC transfer function with gain errors are shown, and in figure 4.7a it has a positive gain because the actual ADC transfer function is above the ideal straight line and in figure 4.7b a negative gain error is represented as the actual ADC transfer function is below the ideal straight line. The gain error is calculated by the number of LSBs of a vertical



(a) ADC positive gain error [36].

(b) ADC negative gain error [36].

Figure 4.7: ADC gain error.

straight line drawn between the midpoint of the last step of the transfer curve of an actual ADC and the ideal straight line [36].

4.2.3 ADC Differential Nonlinearity Error

The differential nonlinearity error (DNL) of an ADC is defined as the difference between the width of a step presented by the transfer function of a real ADC, $L(i)$ as a function of the transfer function of a perfect ADC [37]. For an optimal ADC, DNL = 0 LSB, i.e. each analog step is equal to 1 LSB with all transition values having a width of exactly 1 LSB. The nonlinearity produces steps with varying widths, some of which may be narrower and others wider, as shown in figure 4.8[38]. DNL can only be specified after the ADC gain error is corrected, which is given by equation 4.4, where G is the gain error, $L(i)$ is the width of the quantization levels and V_{LSB} is the ideal spacing for two adjacent digital codes.

$$DNL(i) = \frac{G \times L(i) - V_{LSB}}{V_{LSB}} \tag{4.4}$$

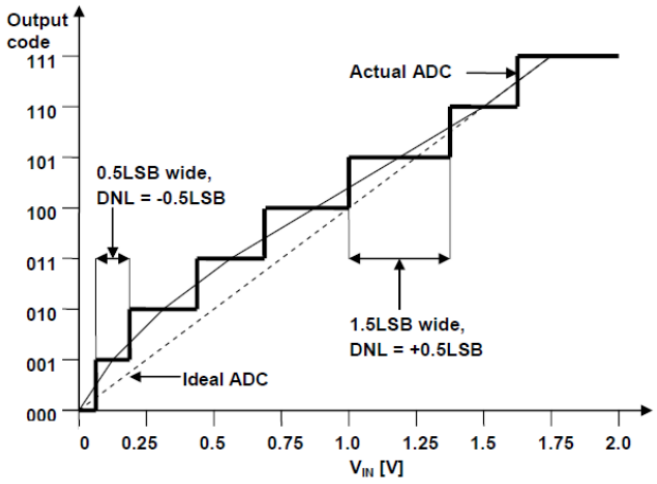


Figure 4.8: ADC differential nonlinearity error [38].

4.2.4 ADC Integral Nonlinearity Error

The integral nonlinearity (INL) error of an ADC is described as the deviation, in LSB or percentage of full scale range (FSR), of a transfer function of a real ADC of a straight line. The INL can be interpreted as a sum of DNLs, since several consecutive negative DNLs increase the actual curve above the ideal curve (positive INL) and several consecutive positive DNLs decrease the real curve below the ideal curve (negative INL). Being this possible to determine it using equation 4.5, where V_{OS} is the deviation error, G is the gain error, V_{LSB} is the ideal spacing for two adjacent digital codes, V_t is the actual value of the transition voltage and V_{ideal} is the value of the transition voltage in an ideal converter [39][37].

The INL can be measured by two different methods, the best-fit method and the end-point method. With the best-fit method it is possible to obtain information about the first two errors referred to in this section, offset and gain errors, as well as the position of the transfer function, which are compared with a linear transfer function and balancing the total of positive and negative deviations. This method can still determine, in the form of a straight line, the closest approximation of the transfer function of an actual ADC, however the exact position of the line is not clearly defined, yet this method generates the best repeatability and can be used as a true representation of linearity. The end-point method is measured by connecting the midpoints of all output steps of the real converter transfer function, thus defining the precise position of the line. Thus, the straight line for an N-bit ADC is defined by its zero outputs (all zeros) and its full output scale, as denoted in figure 4.9. It is also important to note that INL can only be quantified after static shift and gain errors have been overridden [39][37].

$$INL(i) = \frac{G \times V_t(i) + V_{OS} - V_{ideal}(i)}{V_{LSB}}$$
$$INL(i) = \sum_{j=1}^{i} DNL(j) \tag{4.5}$$

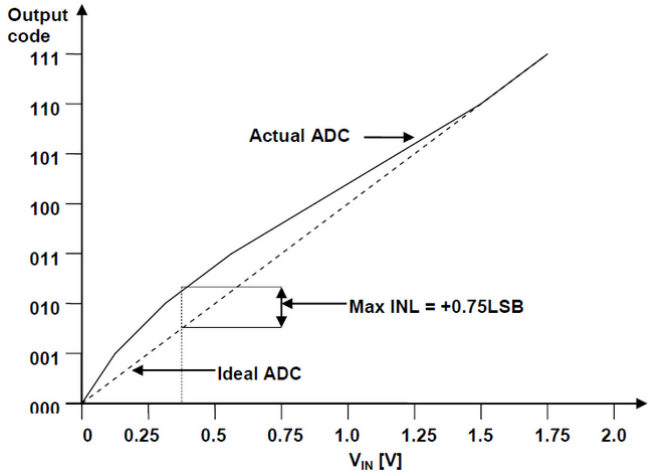


Figure 4.9: ADC integral nonlinearity error [39].

An ADC with INL values outside the range $\pm$ 1 LSB are not suitable to be used, so this functional test is quite important.

4.3 Power Supplies Board Tests

As previously mentioned it is important to do a thorough study of the developed Power Supplies board to ensure that it meets the requirements to be able to work together with HV Remote, otherwise this first prototype of the Power Supplies board will need to be modified to correct the problems encountered, this should be done until all the requirements are fulfilled. For this purpose specific tests will have to be developed, more specifically the parameters previously presented in at the beginning of this chapter will be tested to ensure that the voltages generated are in fact the intended ones, to quantify the noise produced by the board and to verify that the board can withstand continuous operation even in some overheating conditions, while ensuring stability of the specified parameters.

Since it is necessary for the Power Supplies board to produce voltages with as little noise as possible, the stability of the DC/DC converters becomes quite relevant, since if the voltage produced is not stable enough it will only aggravate the interference noise, which could even be small if it wasn't for the oscillating voltage generated by the DC/DC converter. The overall performance of the DC/DC converters will also be tested in order not only to ensure that the voltage generated by them is stable, but also that it is intended to be stable and at the desired voltage for as long as the board is in operation.

There are several types of noise that can affect DC/DC converters, especially the switched source converters such as those used on this board as they are subject to unwanted electrical and electromagnetic signal noise generated by switching artifacts. Other types of noise that can affect DC/DC converters are RF noise, input noise, and output noise. RF noise is caused by the radio waves that switching converters inherently emit and produce electromagnetic interference (EMI).

The input noise, as the name implies, comes from the input voltage supply, which is not negligible as it can change the value of the output voltage of DC/DC converter in addition to producing even more RF noise. The output noise is associated with the inherent noise of the DC/DC converter output signal, which is aggravated by the existence of other types of noise, that is, a DC/DC converter produces a voltage at which some level of electrical noise overlaps, switching noise, produced by the commutation of the converter and some thermal noise present in all types of circuits. Both output and input noise can only be mitigated using filters and noise decoupling capacitors, so it is necessary to ensure that the ones used are the correct ones.

Lastly, the burn-in test will ensure that the Power Supply board will be able to work at its maximum capabilities even when implemented in a crate, where the temperature will rise due to the presence of several other boards that will also be in that same crate.

Chapter 5

Conclusions and Future Work

Testing the components that make up the Power Supplies board is critical to ensuring its proper operation before it is coupled to the HV Remote board. The objective of this work was the development of a high and low voltage supplies board, with little noise, that could supply the whole HV Remote board, the reading of current and voltage consumption, as well as the development of an interface that would allow communication with the Power Supplies board allowing its functional tests.

The first prototype of the Power Supplies board is already made. However, we are already thinking about the improvements that will have to be implemented in the second version of the prototype, one of them being the new placement of components, which is not completely wrong, but it is not completely optimized. Given this, in the near future it is intended to redesign the PCB of this board so as to save space and reduce the number of layers if possible, thus reducing the cost of producing this board. For that, noise measurements and tests stability test should be made.

At the level of the graphical user interface, it will also need to be improved since the last tests performed detected an error in the code, which is not very critical at this phase but can cause a serious problem in the future. This error affects the DC/DC converters enable/disable section and the output voltage selection section of the HV DC/DC converters. This error is only evident once the user interacts with these sections by enabling the converters and/or selecting a output voltage for the HV converters, then closes the interface and reopens the interface. What happens is that once the user enables the DC/DC converters, they will remain enable even after the interface has been closed, however when the interface is restarted even though the converters are still enable the interface provides the opposite information, the same being said to the HV DC/DC converters output voltage selection section. However, a solution to this flaw has already been devised but unfortunately at the time this solution was found, time did not allow its implementation in this thesis.

Much work remains to be done in the future. The most complex ADC tests have not yet been performed and all the control system must be tested with the Power Supplies board. Although the graphical user interface has already been tested, it will still be tested when the first prototype is ready, so that we can test the interface in the environment for which it was designed

to work and not only with three components assembled on a breadboard, thus ensuring that it really works. No tests were performed on the DC/DC converters either, which will also have to be tested when the prototype is finished. As soon as the components are assembled, the necessary tests will be performed to verify the general functioning of the board as well as its components.

Once this is done, the Power Supplies board prototype will then be linked to the HV Remote board. This set will need to be tested to ensure that both boards function properly and if the work presented in this thesis fulfill all the necessary requirements. If necessary, some changes will be made, and a second prototype of this board will be implemented.

Bibliography

[1] G. F. Giudice. *A Zeptospace Odyssey - A Journey into the Physics of the LHC*. Oxford University Press, Oxford New York, 2010.

[2] C. O'Luanaigh. A vacuum as empty as interstellar space. `https://home.cern/science/engineering/vacuum-empty-interstellar-space`, January 2014.

[3] C. O'Luanaigh. The large hadron collider. `https://home.cern/topics/large-hadron-collider`, January 2014.

[4] C. O'Luanaigh. Atlas. `https://home.cern/science/accelerators/large-hadron-collider`, January 2014.

[5] F. Tang, K. Anderson, G. Drake, J. Genat, M. Oreglia, J. Pilcher, and L. Price. Design of the front-end readout electronics for atlas tile calorimeter at the slhc. *IEEE Transactions on Nuclear Science*, 60(vol. 2):1255 – 1259, April 2013. DOI: 10.1109/TNS.2012.2215053.

[6] L. Plazak. The atlas hadronic calorimeter at the lhc and the phase iiupgrade program. *The XXII International Workshop High Energy Physics and Quantum Field Theory, Samara, Russia, 24 Jun - 1 Jul 2015*, page 6, November 2015. Preliminary results.

[7] A. Henriques. 2015 4th international conference on advancements in nuclear instrumentation measurement methods and their applications (animma). *IEEE*, April 2015.

[8] A. Collaboration. The atlas experiment at the cern large hadron collider. *Institute of physics publishing and SISSA*, page 437, August 2008.

[9] J. Valls and J. Castelo. Tilecal read-out driver. `http://ific.uv.es/tical/old/rod/index.html`, April 2005. IFIC, Universitat de València.

[10] F. Vazeille. Performance of a remote high voltage power supply for the phase ii upgrade of the atlas tile calorimeter. *Topical Workshop on Electronics for Particle Physics*, October 2015.

[11] S. Muschter, H. Aakerstedt, K. Anderson, C. Bohm, M. Oreglia, and F. Tang. Development of a digital readout board for theatlas tile calorimeter upgrade demonstrator. *Journal of Instrumentation*, vol. 9, January 2014.

[12] J. Alves. Interface ethernet para um testador de sistemas electrónicos do tileca. Master's thesis, Faculdade de ciências da Universidade de Lisboa, 2012.

[13] H. P. K.K. C12446 series datasheet, April 2016.

[14] Wikipedia. Voltage divider. https://en.wikipedia.org/wiki/Voltage_divider, .

[15] Wikiwand. Buffer amplifie. https://www.wikiwand.com/en/Buffer_amplifier.

[16] Ângelo Meneguini. Amplificador operacional - amplificador inversor. https://eletronicagaragem.blogspot.com/2017/03/amplificador-operacional-amplificador_30.html, March 2017.

[17] T. I. I. Copyright © 2009–2014. Ina199 datasheet, April 2009. Revised February 2017.

[18] A. D. I. © 2003. Tmp17 datasheet.

[19] Altium. *HIGH-VOLTAGE PCB DESIGN*. Altium Designer, September 2017.

[20] S. Sattel. Pcb layout basics part 1: How to place your components. https://www.autodesk.com/products/eagle/blog/pcb-layout-basics-component-placement/.

[21] M. C. Lim. Relationship trace width and current carrying capacity. https://www.mclpcb.com/pcb-trace-width-vs-current-table/.

[22] Wikipedia. Serial peripheral interface. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface, .

[23] P. Dhaker. Introduction to spi interface. https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html#, March 2017.

[24] C. Basics. Basics of the spi communication protocol. http://www.circuitbasics.com/basics-of-the-spi-communication-protocol/.

[25] M. Grusin. Serial peripheral interface (spi). https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all#introduction.

[26] Corelis. Spi tutorial. https://www.corelis.com/education/tutorials/spi-tutorial/.

[27] D. Ware. Clock phase and polarity. http://dlnware.com/dll/Clock-Phase-and-Polarity.

[28] T. Baumann. Spidev documentation. http://tightdev.net/SpiDev_Doc.pdf.

[29] R. C. Limited. Pyqt5 5.13.1. https://pypi.org/project/PyQt5/, September 2019.

[30] P. Howard. Raspberry pi pinout. https://pinout.xyz/pinout/#.

[31] T. I. Incorporated. Mux36s16 datasheet, November 2016. Revised April 2018.

[32] M. T. I. © 2007. Mcp23s17 datasheet, June 2005. Revision C July 2016.

[33] M. I. Products. Max1240 datasheet, August 2010.

[34] I. © 2019 Microchip Technology. Adc offset error. https://microchipdeveloper.com/adc:adc-offset-error#top-of-page, .

[35] I. Freescale Semiconductor. How to increase the analog-to-digital converteraccuracy in an application. January 2016. Application note.

[36] I. © 2019 Microchip Technology. Adc gain error. https://microchipdeveloper.com/adc:adc-gain-error, .

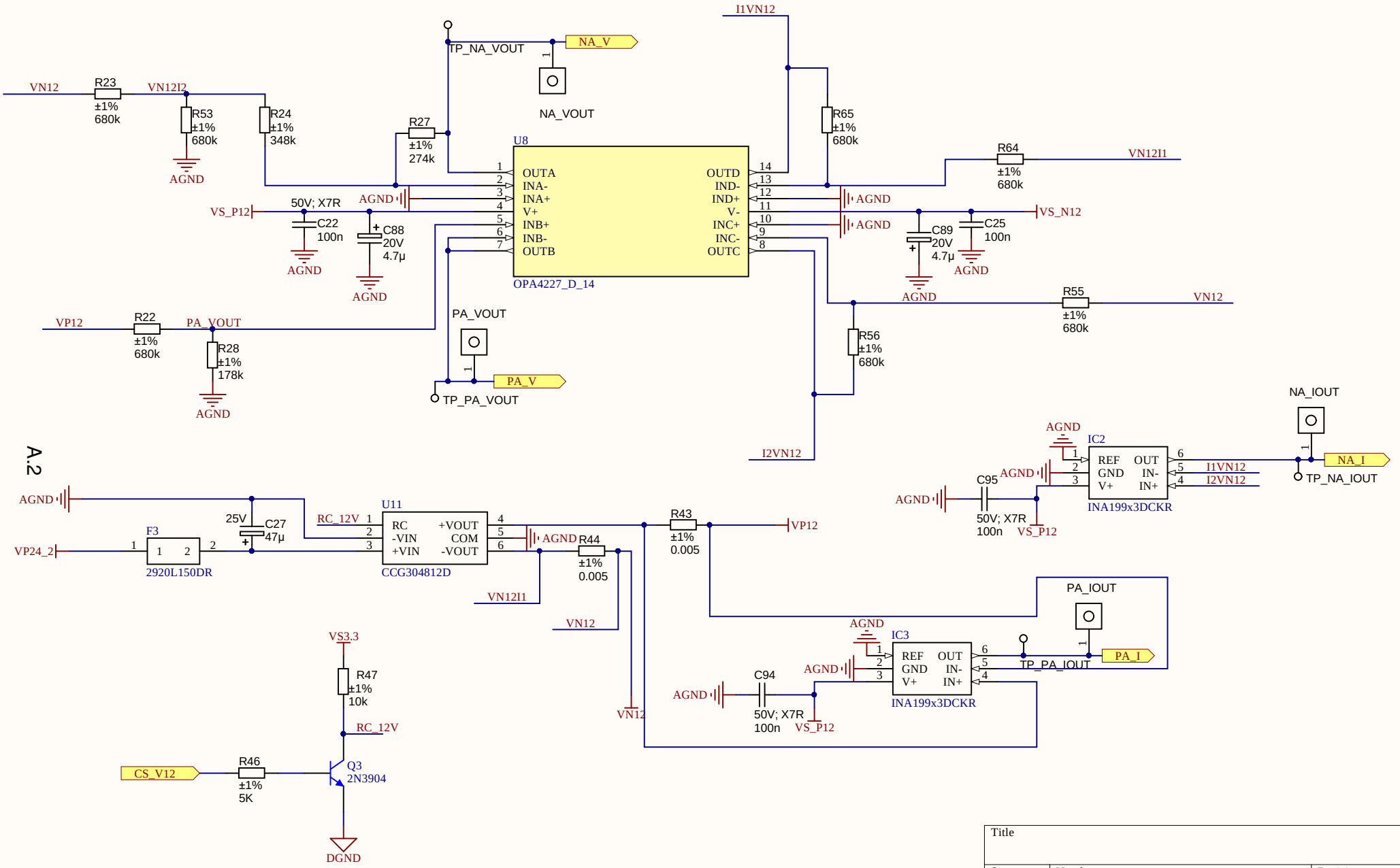
[37] I. ©2001, Maxim Integrated Products. Adc integral nonlinearity (inl). https://www.maximintegrated.com/en/design/technical-documents/tutorials/2/283.html, November .

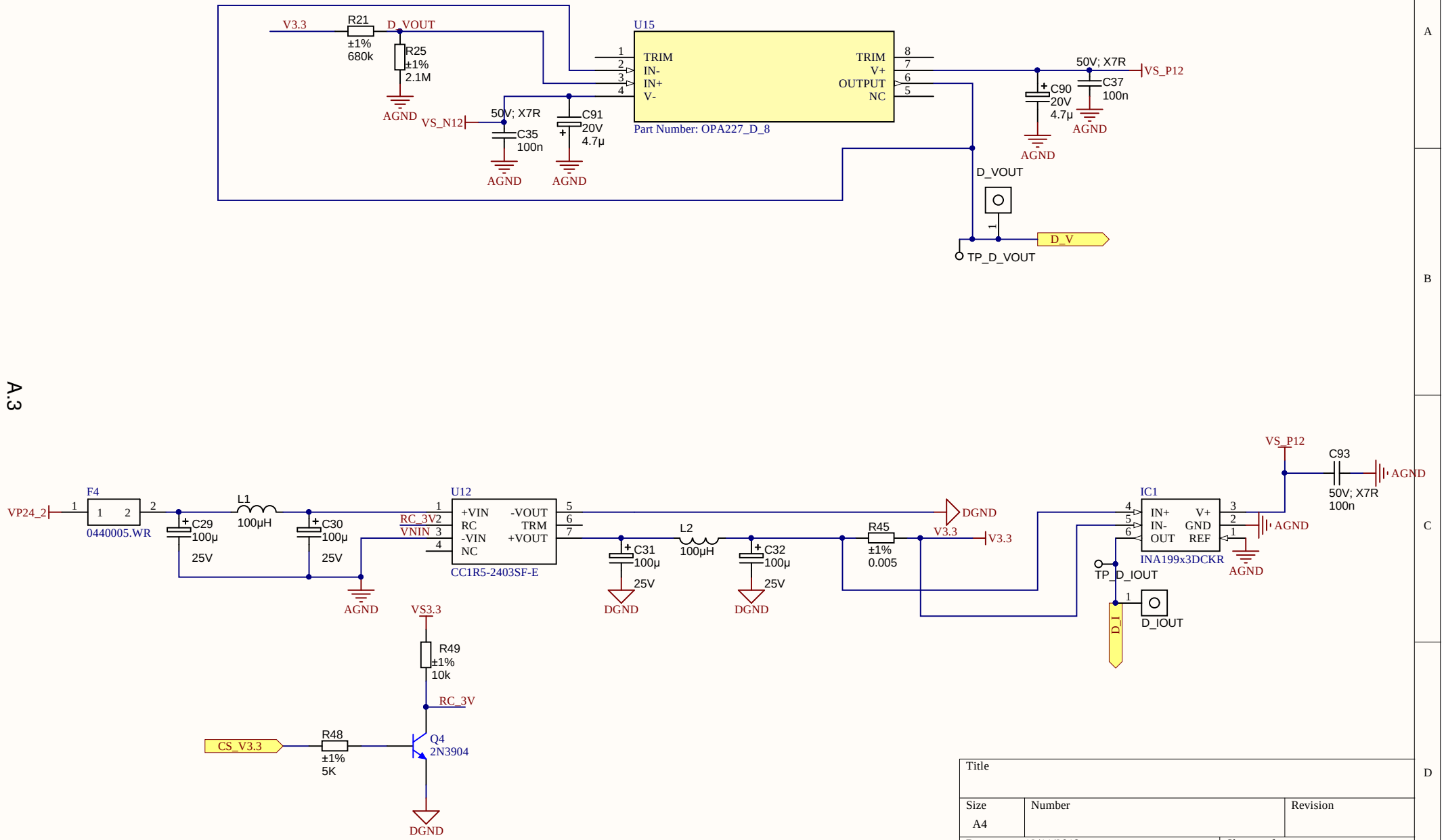
[38] I. © 2019 Microchip Technology. Adc differential nonlinearity. https://microchipdeveloper.com/adc:adc-differential-nonlinearity, .

[39] I. © 2019 Microchip Technology. Adc integral nonlinearity (inl). https://microchipdeveloper.com/adc:adc-inl, .

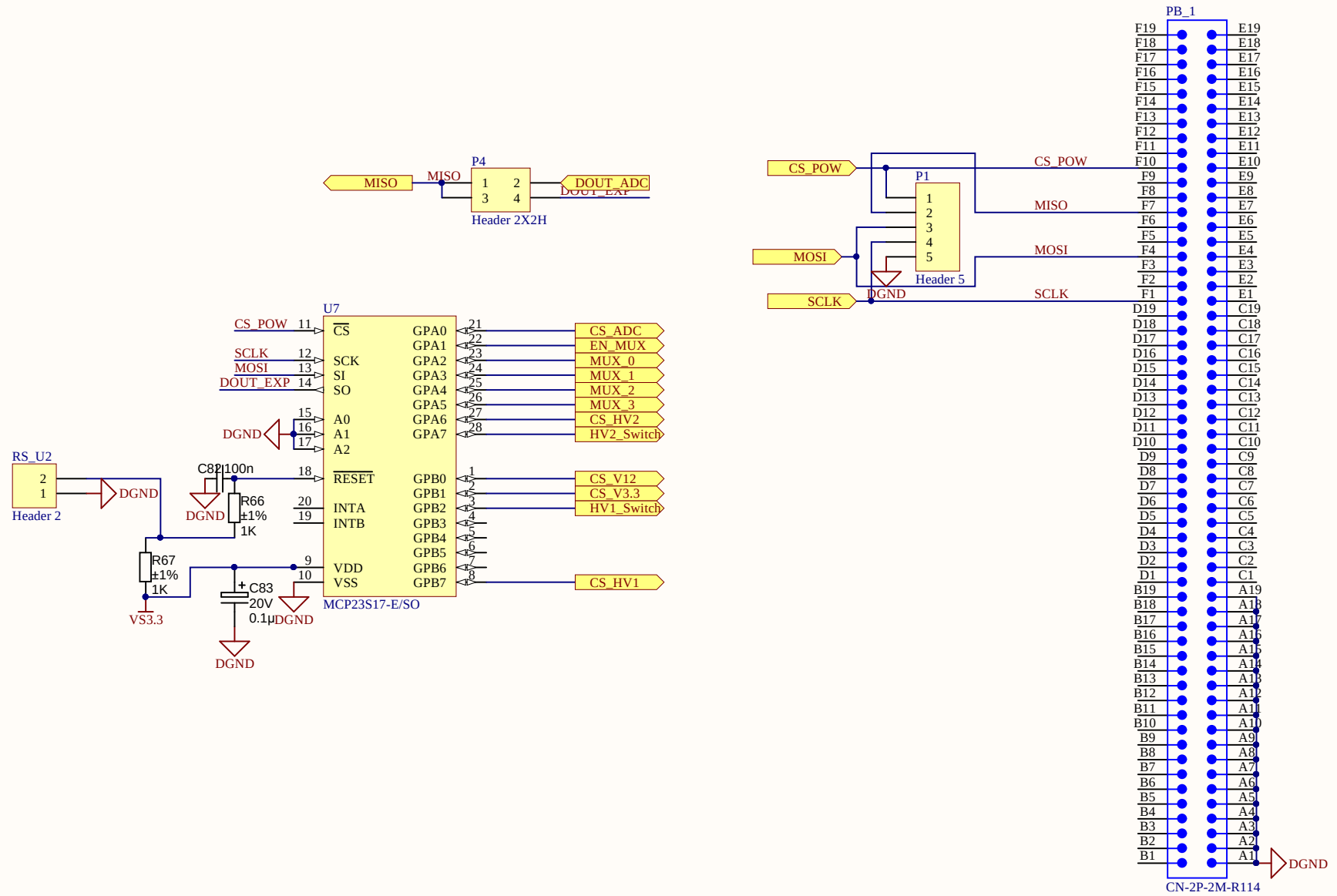
Appendix A

Schematic Designs of the Power Supplies Board

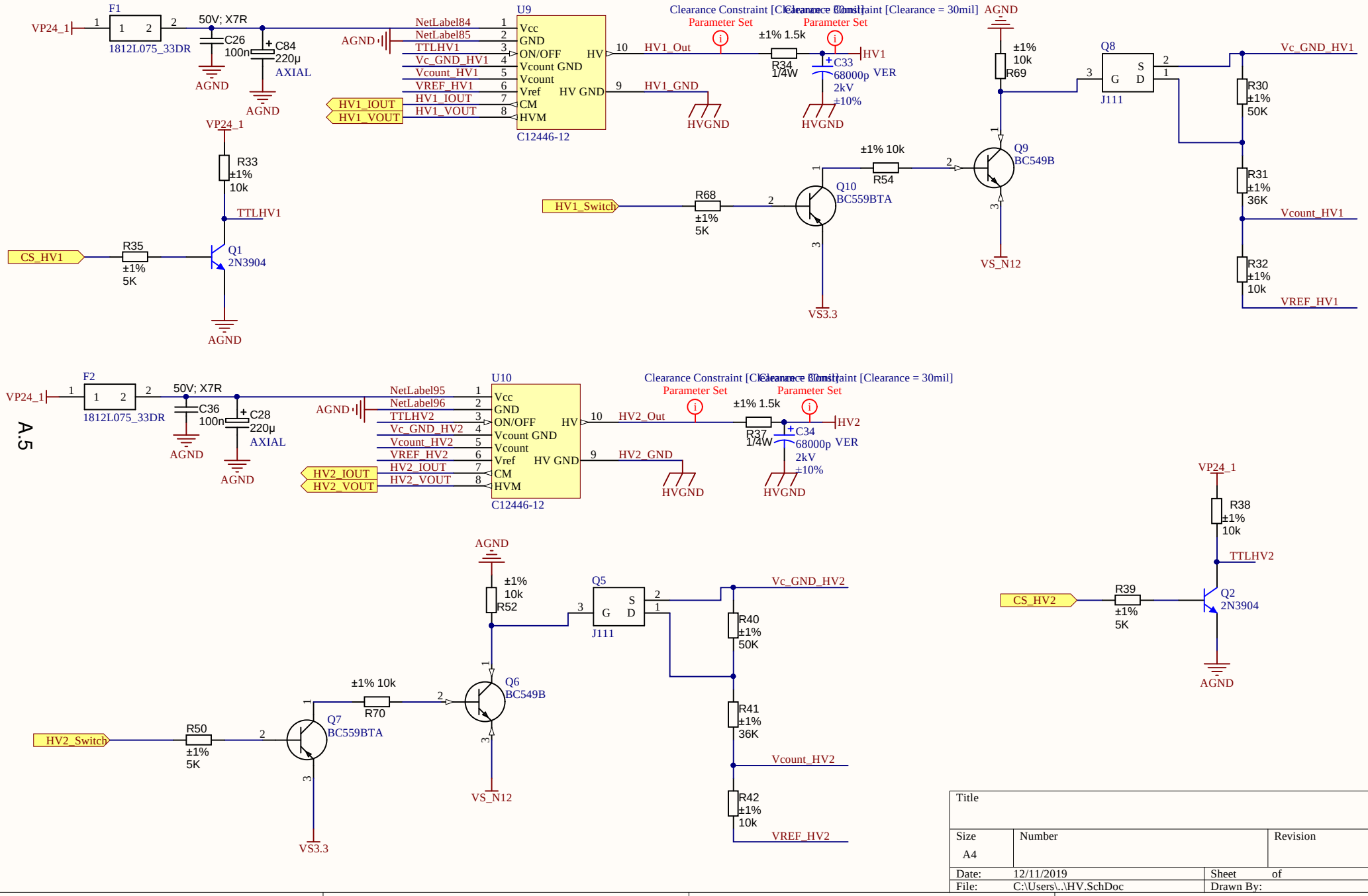

A.2
VN12
R23
±1%
680k
VN12I2
R53
±1%
680k
AGND
R24
±1%
348k
TP_NA_VOUT
NA_V
NA_VOUT
R27
±1%
274k
I1VN12
U8
OUTA
INA-
INA+
V+
INB+
INB-
OUTB
OUTD
IND-
IND+
V-
INC+
INC-
OUTC
OPA4227_D_14
R65
±1%
680k
R64
±1%
680k
VN12I1
VS_P12
50V; X7R
C22
100n
C88
20V
4.7µ
C89
20V
4.7µ
C25
100n
VS_N12
R55
±1%
680k
VN12
VP12
R22
±1%
680k
PA_VOUT
R28
±1%
178k
R56
±1%
680k
PA_V
TP_PA_VOUT
I2VN12
NA_IOUT
NA_I
TP_NA_IOUT
IC2
REF
GND
V+
OUT
IN-
IN+
INA199x3DCKR
C95
50V; X7R
100n
VS_P12
VP24_2
F3
2920L150DR
25V
C27
47µ
RC_12V
U11
RC
-VIN
+VIN
+VOUT
COM
-VOUT
CCG304812D
R44
±1%
0.005
R43
±1%
0.005
VP12
VN12I1
VN12
PA_IOUT
PA_I
TP_PA_IOUT
IC3
C94
50V; X7R
100n
VS_P12
VS3.3
R47
±1%
10k
RC_12V
CS_V12
R46
±1%
5K
Q3
2N3904
DGND
Title
Size
A4
Number
Revision
Date: 12/11/2019
Sheet of
File: C:\Users\..\AV.SchDoc
Drawn By:

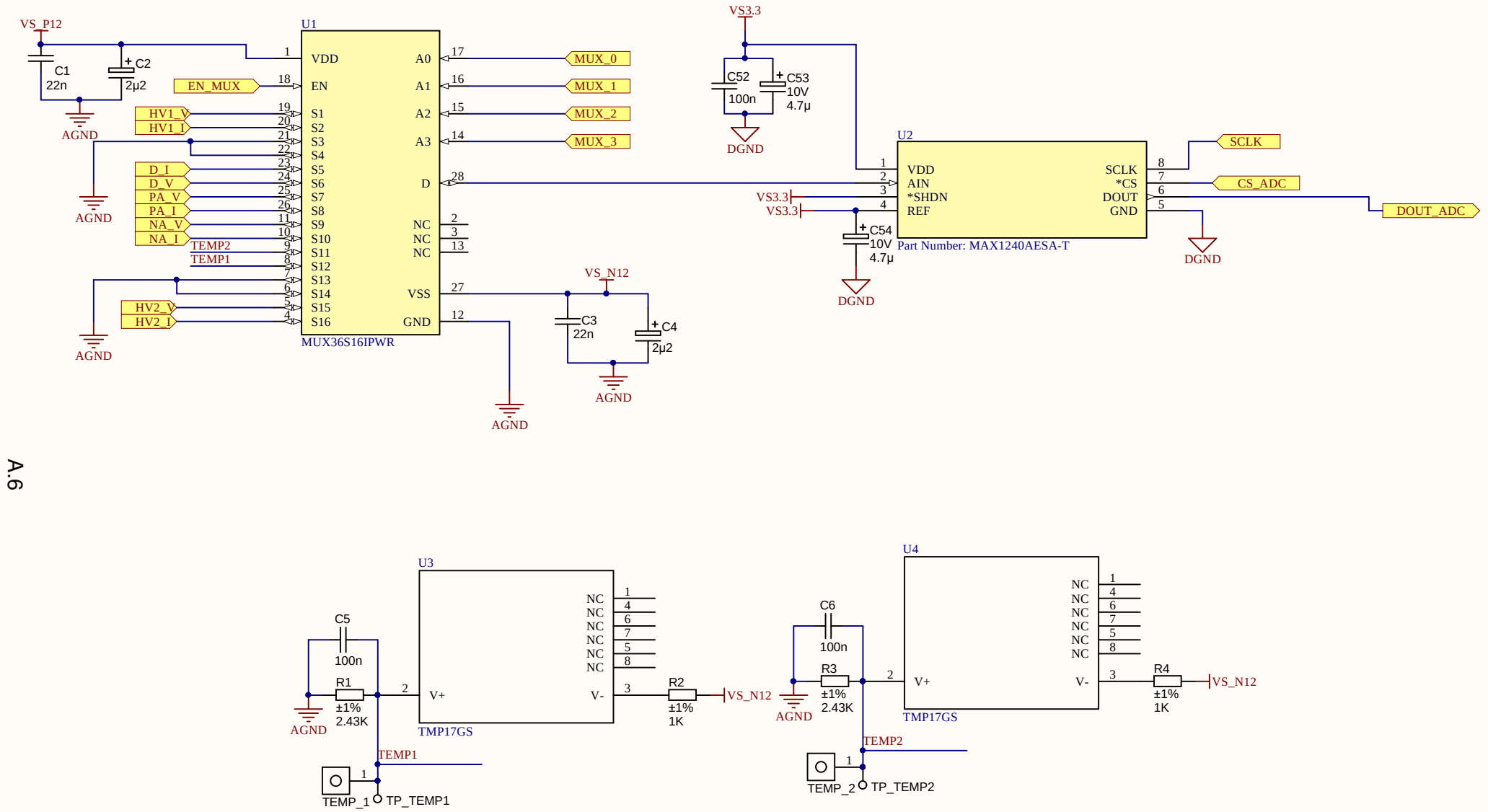


Title		
Size: A4	Number	Revision
Date: 12/11/2019		Sheet of
File: C:\Users\..\DV.SchDoc		Drawn By:

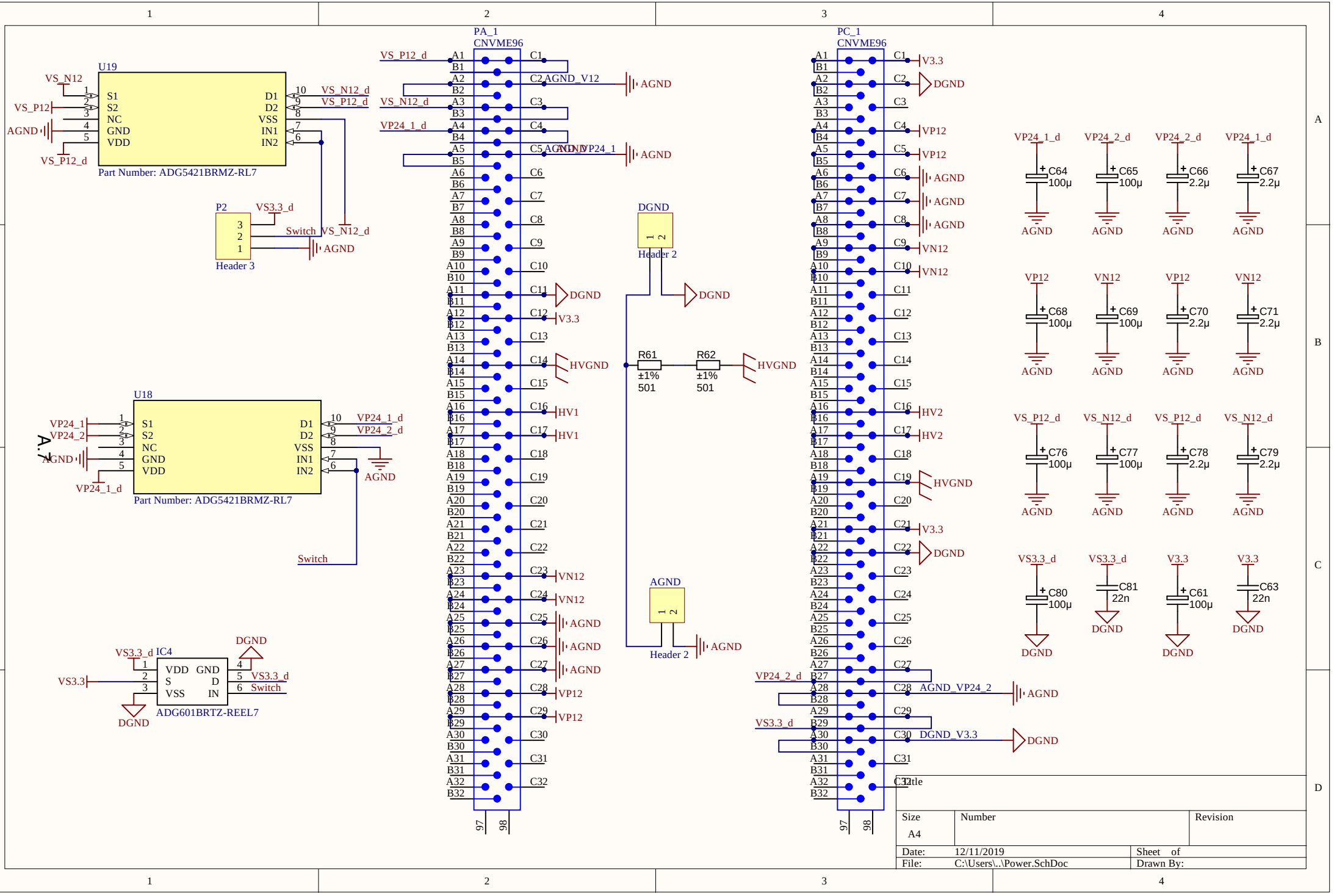


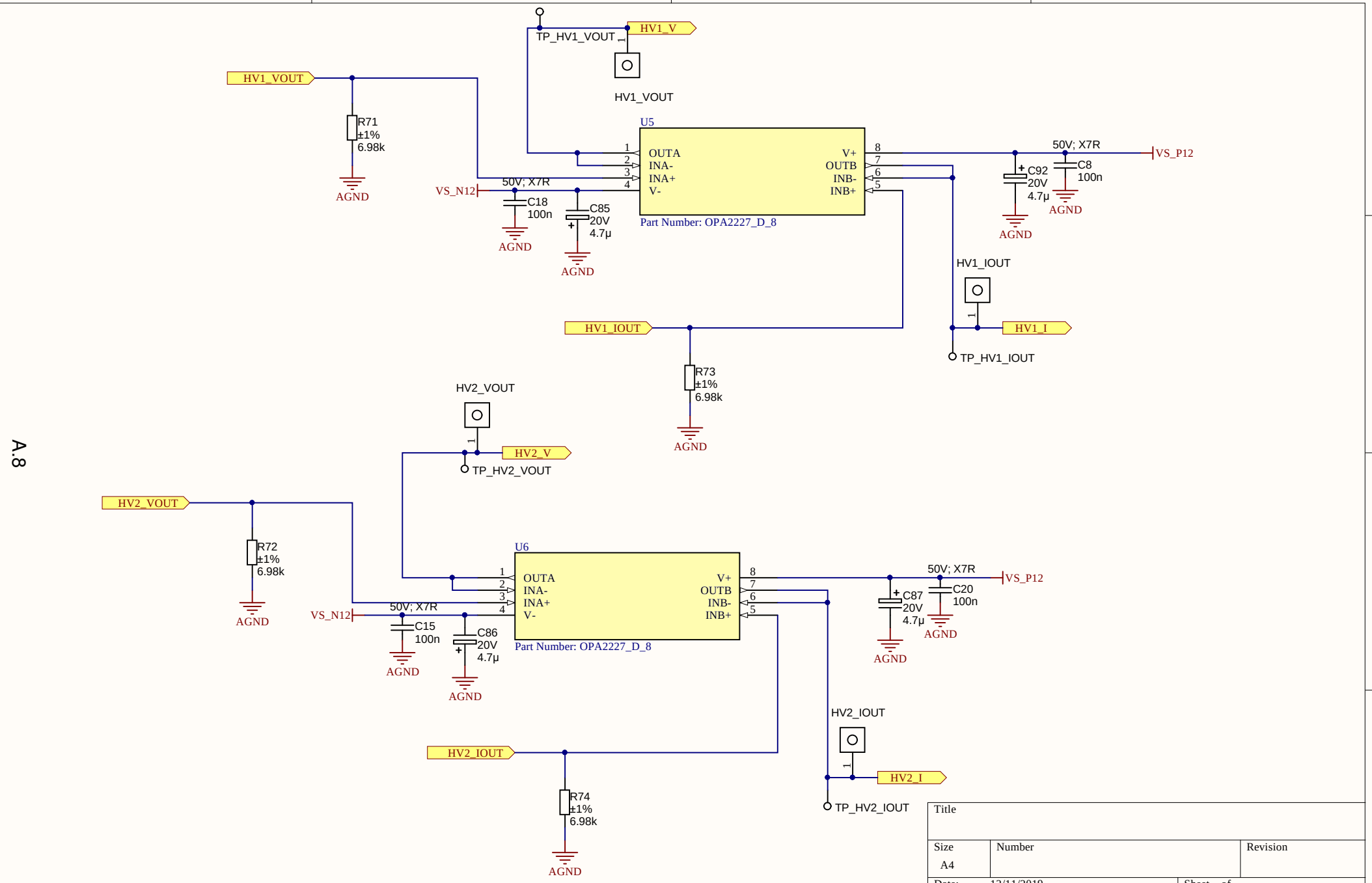
Title		
Size A4	Number	Revision
Date: 12/11/2019		Sheet of
File: C:\Users\..\Expansor.SchDoc		Drawn By:

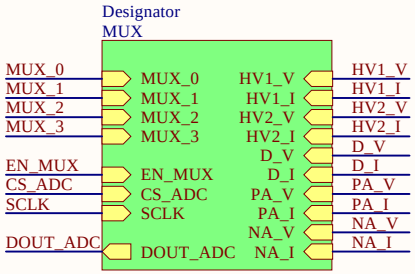

F1
VP24_1
1812L075_33DR
50V; X7R
C26
100n
C84
220µ
AXIAL
AGND
U9
NetLabel84
NetLabel85
TTLHV1
Vc_GND_HV1
Vcount_HV1
VREF_HV1
HV1_IOUT
HV1_VOUT
Vcc
GND
ON/OFF
Vcount GND
Vcount
Vref
CM
HVM
HV
HV GND
C12446-12
HV1_Out
HV1_GND
HVGND
Clearance Constraint [Clearance = 30mil]
Parameter Set
±1% 1.5k
R34
1/4W
C33
68000p
2kV
VER
±10%
HV1
AGND
±1%
10k
R69
Q8
J111
Vc_GND_HV1
R30
±1%
50K
R31
±1%
36K
Vcount_HV1
R32
±1%
10k
VREF_HV1
VP24_1
R33
±1%
10k
TTLHV1
CS_HV1
R35
±1%
5K
Q1
2N3904
HV1_Switch
R68
±1%
5K
Q10
BC559BTA
±1% 10k
R54
Q9
BC549B
VS_N12
VS3.3
F2
U10
NetLabel95
NetLabel96
TTLHV2
Vc_GND_HV2
Vcount_HV2
VREF_HV2
HV2_IOUT
HV2_VOUT
HV2_Out
HV2_GND
HV2
C34
R37
R38
R39
R40
R41
R42
R50
R52
R70
Q2
Q5
Q6
Q7
A.5
Title
Size
A4
Number
Revision
Date:
12/11/2019
Sheet
of
File:
C:\Users\..\HV.SchDoc
Drawn By:

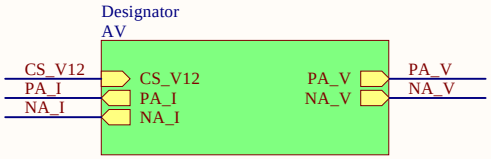


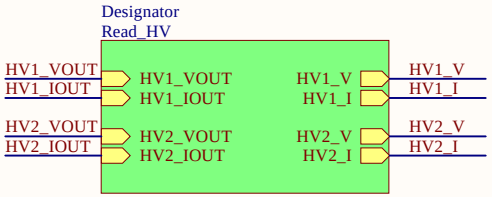
Title		
Size A4	Number	Revision
Date: 12/11/2019		Sheet of
File: C:\Users\..\MUX.SchDoc		Drawn By:

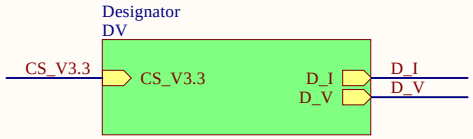

U19
VS_N12
VS_P12
AGND
VS_P12_d
S1
S2
NC
GND
VDD
D1
D2
VSS
IN1
IN2
VS_N12_d
VS_P12_d
Part Number: ADG5421BRMZ-RL7
P2
VS3.3_d
Switch
VS_N12_d
AGND
Header 3
U18
VP24_1
VP24_2
VP24_1_d
VP24_2_d
Part Number: ADG5421BRMZ-RL7
Switch
IC4
VS3.3_d
VS3.3
DGND
VDD
GND
S
D
VSS
IN
Switch
ADG601BRTZ-REEL7
PA_1
CNVME96
VS_P12_d
VS_N12_d
VP24_1_d
AGND_V12
AGND_VP24_1
DGND
V3.3
HVGND
HV1
VN12
VP12
PC_1
CNVME96
V3.3
DGND
VP12
VN12
AGND
HV2
HVGND
VP24_2_d
AGND_VP24_2
VS3.3_d
DGND_V3.3
Header 2
R61
R62
±1%
501
C64 100µ
C65 100µ
C66 2.2µ
C67 2.2µ
C68 100µ
C69 100µ
C70 2.2µ
C71 2.2µ
C76 100µ
C77 100µ
C78 2.2µ
C79 2.2µ
C80 100µ
C81 22n
C61 100µ
C63 22n
Title
Size
A4
Number
Revision
Date: 12/11/2019
File: C:\Users\..\Power.SchDoc
Sheet of
Drawn By:

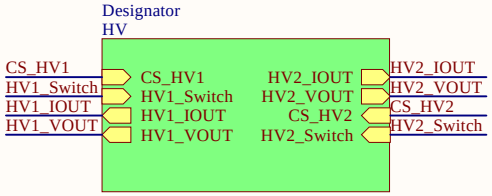

TP_HV1_VOUT
HV1_V
HV1_VOUT
HV1_VOUT
R71
±1%
6.98k
AGND
U5
OUTA
INA-
INA+
V-
V+
OUTB
INB-
INB+
Part Number: OPA2227_D_8
VS_N12
50V; X7R
C18
100n
C85
20V
4.7µ
AGND
AGND
50V; X7R
C92
20V
4.7µ
C8
100n
VS_P12
AGND
AGND
HV1_IOUT
HV1_IOUT
HV1_I
TP_HV1_IOUT
R73
±1%
6.98k
AGND
HV2_VOUT
HV2_V
TP_HV2_VOUT
HV2_VOUT
R72
±1%
6.98k
AGND
U6
OUTA
INA-
INA+
V-
V+
OUTB
INB-
INB+
Part Number: OPA2227_D_8
VS_N12
50V; X7R
C15
100n
C86
20V
4.7µ
AGND
AGND
50V; X7R
C87
20V
4.7µ
C20
100n
VS_P12
AGND
AGND
HV2_IOUT
HV2_IOUT
HV2_I
TP_HV2_IOUT
R74
±1%
6.98k
AGND
A.8
Title
Size
A4
Number
Revision
Date: 12/11/2019
Sheet of
File: C:\Users\..\Read_HV.SchDoc
Drawn By:

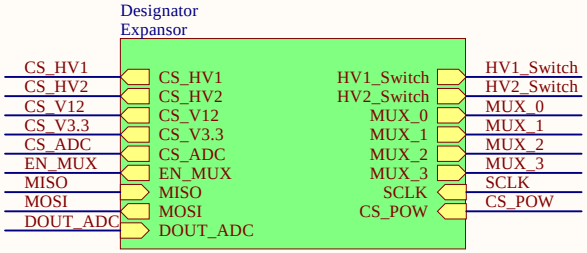


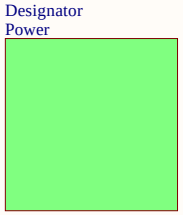












A.9

Title		
Size A4	Number	Revision
Date: 12/11/2019		Sheet of
File: C:\Users\..\Shc_HVLV.SchDoc		Drawn By:

Appendix B

List of Components

Component list

Bill of Materials For PCB Document [Shc_HVLV.PcbDoc]

Source Data From: Shc_HVLV.PcbDoc
Project: PCB_Project_1.PrjPcb
Variant: None

Report Date: 04/06/2019 16:55:13
Print Date: 18-Sep-19 6:14:26 PM

#	LibRef	Description	Footprint	Quantity
1	Header 2	Header, 2-Pin	HDR1X2	3
2	CAPC_22n_0603_X7R_50V_10%	Capacitor SMD Cer 22nF 10% 50V X7R 0603	CAPC1608X90N	8
3	CAPT_2.2u_6032_50V	Capacitor SMD Tantalum 2.2uF 10% 50V EIA 6032-28	CAPMP6032X280N	2
4	CAPC_100n_0603_X7R_50V_10%	Capacitor SMD Cer 100nF 10% 50V X7R 0603	CAPC1608X90N	4
5	CAPC_100n_0603_X7R_50V_10%	Capacitor SMD Cer 100nF 10% 50V X7R 0603	1608[0603]	13
6	CAPT_47u_2917_25V	Capacitor SMD Tant 47uF 20% 25V 2917	CAPMP7343X310N	1
7	CAPE_220u_35V_SMD	Capacitor SMD Electrolytic 220uF 20% 35V Radial	CAP_ELEC_6.3x4.2	2
8	CAPT_100u_2917_25V	Capacitor SMD Tant 100uF 20% 25V 2917	CAPMP7343X310N	4
9	CAPC_68n_4540_2kV_10%	Capacitor SMD Cer 68nF 10% 2kV 4540	CAPC4564AM	2
10	CAPT_4.7u_0805_20V	Capacitor SMD Tantalum 4.7uF 20% 20V 0805	CAPMP2012X150USR	8
11	CAPT_4.7u_0805_10V	Capacitor SMD Tantalum 4.7uF 20% 10V 0805	CAPMP2012X150USR	2
12	CAPE_100u_100V_SMD	Capacitor SMD Electrolytic 100uF 20% 100V Radial	CAP_ELEC_6.3x4.2	12
13	CAPE_2.2u_100V_SMD	Capacitor SMD Electrolytic 2.2uF 20% 100V Radial	CAP_ELEC_5x5.7	8
14	CAPT_0.1u_0805_20V	Capacitor SMD Tantalum 0.1uF 20% 20V 0805	CAPMP2012X150USR	1
15	CONN_SOCKET_1x1P_0.100in	Socket header THT 1-Pos Low profile Single-row 2.54mm	1x1POS_SOCKET_LOW	12
16	1812L075_33DR	Fuse LITTELFUSE 1812L075/33DR	FUSC4632X155N	2
17	2920L150DR	Fuse LITTELFUSE 2920L150DR	FUSC7451X125N	1
18	0440005.WR	Fuse LITTELFUSE 0440005.WR	FUSC3216X86N	1
19	INA199A1DCKR	INA199x3DCKR	SOT65P210X110-6N	3
20	ADG601BRTZ-REEL7	ADG601BRTZ-REEL7	SOT95P280X145-6N	1
21	INDC_100u_1210_220mA	Inductor 100uH 10% 0.22A 1.82ohms 1210 SMD chip	1210	2
22	Header 5	Header, 5-Pin	HDR1X5	1
23	Header 3	Header, 3-Pin	HDR1X3	1
24	Header 2X2H	Header, 2-Pin, Dual row, Right Angle	HDR2X2H	1
25	CNVME96	96-Pin Connector (3x32 2.54mm grid)	CNVME96	2
26	CN-2P-2M-R114	Board-to-Board Connector, Vertical, 6-Row, Pitch 1.27mm,	BTB2-6H114M	1
27	2N3904	NPN General Purpose Amplifier 2N3904	TO-92A	4
28	J111	Transistor J111	SS9018HBU	2
29	BC549B	Transistor BJT NPN BC549B	TO-92_BULK	2
30	BC559BTA	Transistor BJT PNP BC559BTA	TO-92_3L_(AMMO)	2
31	RES_2.43K_0603_1%	Resistor SMD chip 2.43k Ohm 0.1W 1% 0603	RESC1608X55N	2
32	RES_1K_0603_1%	Resistor SMD chip 1k Ohm 0.1W 1% 0603	RESC1608X55N	4
33	RES_680K_0603_1%	Resistor SMD chip 680k Ohm 0.1W 1% 0603	RESC1608X55N	8
34	RES_348K_0603_1%	Resistor SMD chip 348k Ohm 0.1W 1% 0603	RESC1608X55N	1

Figure B.1: List of components of the Power Supplies board, part A.

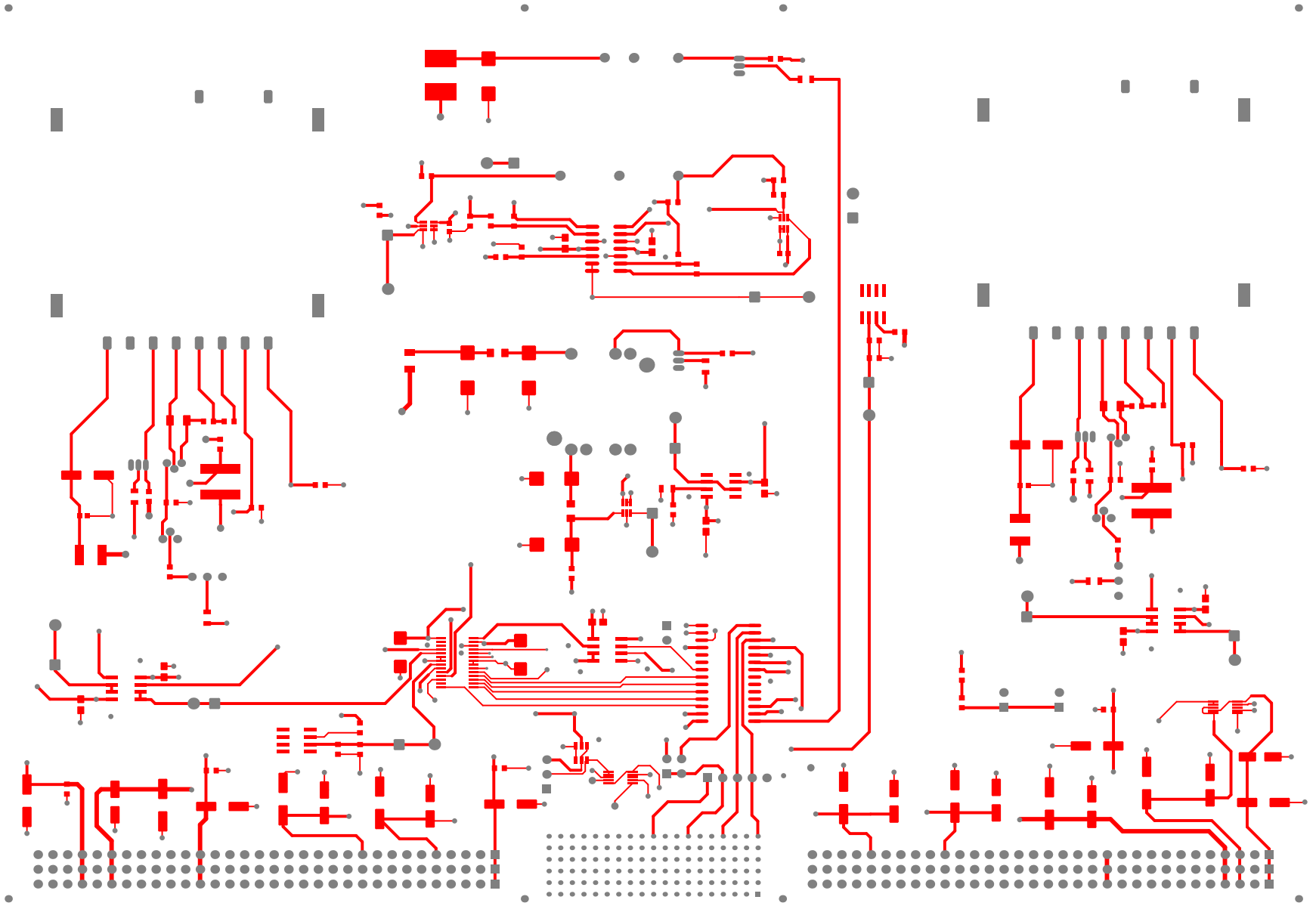
35	RES_2M1_0805_1%	'Resistor SMD chip 2.1M Ohm 0.125W 1% 0805	RESC2012X70N	1
36	RES_5k_0805_1%	Resistor SMD chip 5k Ohm 0.125W 1% 0805	RESC2012X70N	6
37	RES_274K_0603_1%	Resistor SMD chip 274k Ohm 0.1W 1% 0603	RESC1608X55N	1
38	RES_178K_0603_1%	Resistor SMD chip 178k Ohm 0.1W 1% 0603	RESC1608X55N	1
39	RES_50K_1206_1%	Resistor SMD chip 50K Ohm 0.25W 1% 1206	RESC3216X70N	2
40	RES_36K_0603_1%	Resistor SMD chip 36k Ohm 0.1W 1% 0603	RESC1608X55N	2
41	RES_6K98_0603_1%	Resistor SMD chip 6.98k Ohm 0.1W 1% 0602	RESC1608X55N	2
42	RES_10K_0603_1%	Resistor SMD chip 10k Ohm 0.1W 1% 0603	RESC1608X55N	12
43	RES_1K5_0603_1%	Resistor SMD chip 1.5k Ohm 0.25W 1% 0603	RESC1608X55N	2
44	RES_0.005_0603_1%	Resistor SMD chip 0.005 Ohm 0.1W 1% 0603	RESC1608X55N	3
45	RES_501_0603_1%	Resistor SMD chip 501 Ohm 0.1W 1% 0603	RESC1608X55N	2
46	MUX36S16IPWR	MUX36S16IPWR	PW0028A_N	1
47	MAX1240AESA-T	MAX1240AESA-T	21-0041B_8	1
48	TMP17GS	TMP17GS	R_8	2
49	OPA2227_D_8	High Precision, Low Noise Operational Amplifiers	D8	2
50	MCP23S17-E/SO	16-Bit I/O Expander with Serial Interface, 28-Pin SOIC,	SOIC-SO28_N	1
51	OPA4227_D_14	OPA4227_D_14	TI-D14_M	1
52	C12446-12	DC DC CONVERTER 0 to -1000 V Hamamatsu C12446-12	C12446-12	2
53	CCG304812D	DC DC CONVERTER +/-12V 30W TDK CCG304812D	CCG30	1
54	CC1R5-2403SF-E	DC DC CONVERTER 3.3V TDK CC1R5-2403SF-E	CC1R5-XXXXXF-E	1
55	OPA227_D_8	High Precision, Low Noise Operational Amplifiers	D8	1
56	ADG5421BRMZ-RL7	ADG5421BRMZ-RL7	RM_10	2
Approved				**172**

Figure B.2: List of components of the Power Supplies board, part B.

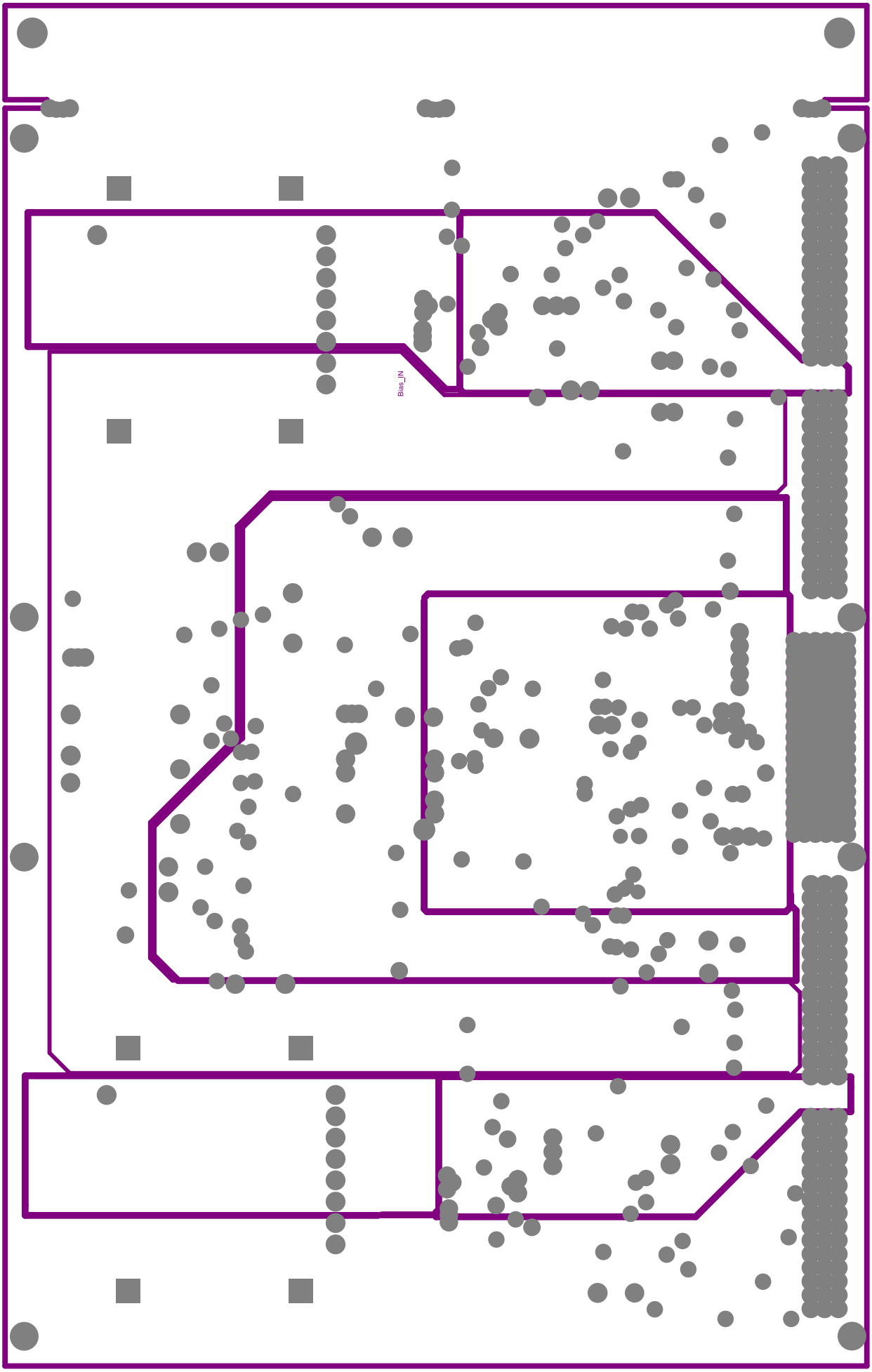
Appendix C

Layers of Power Supplies Board

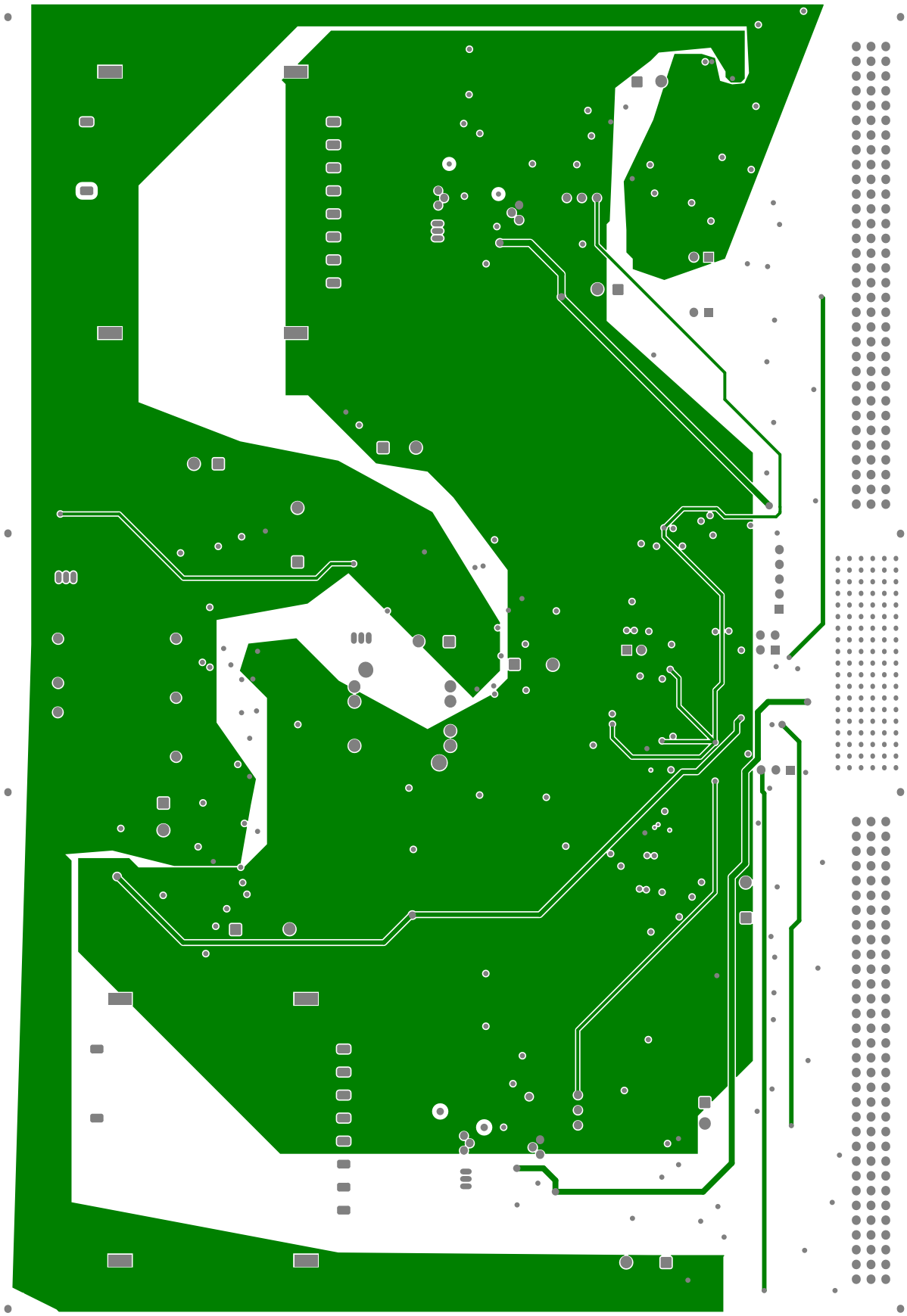
C.1 Top Layer



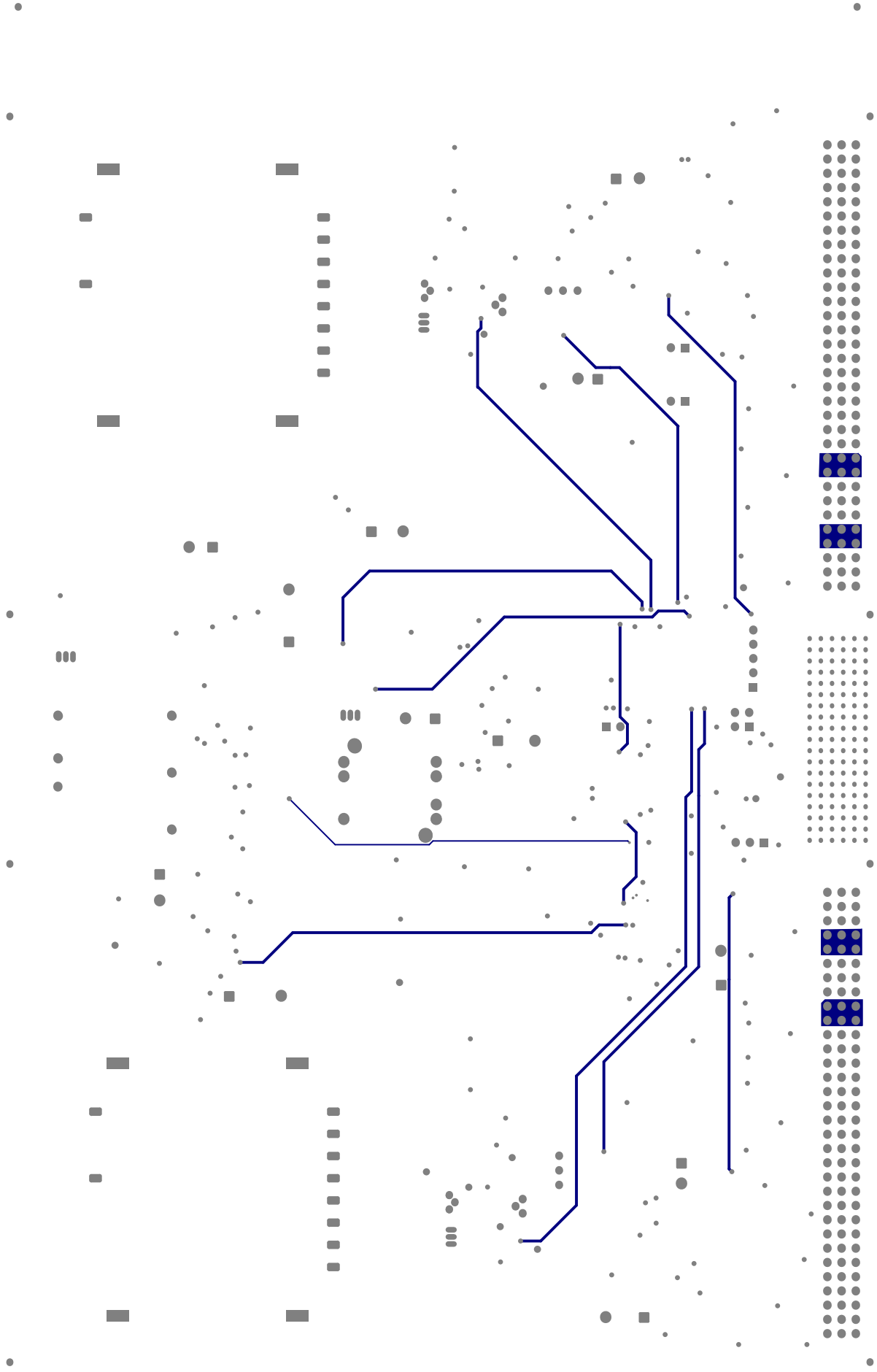
C.2 Bias_OUT Layer


Bias_IN

C.3 Bias_IN Layer



C.4 Signals Layer


.G3

C.5 Bottom Layer

PN XXXXXX-XXX REV A
.GBL
U12

Appendix D

Implemented Scripts

```
import time

import RPi.GPIO as GPIO

import spidev


class MCP23S17(object):
    """This class provides an abstraction of the GPIO expander MCP23S17
    for the Raspberry Pi.
    It is depndent on the Python packages spidev and RPi.GPIO, which can
    be get from https://pypi.python.org/pypi/RPi.GPIO/0.5.11 and
    https://pypi.python.org/pypi/spidev.
    """
    PULLUP_ENABLED = 0
    PULLUP_DISABLED = 1

    DIR_INPUT = 0
    DIR_OUTPUT = 1

    LEVEL_LOW = 0
    LEVEL_HIGH = 1

    """Register addresses as documentined in the technical data sheet at
    http://ww1.microchip.com/downloads/en/DeviceDoc/21952b.pdf
    """
    MCP23S17_IODIRA = 0x00
    MCP23S17_IODIRB = 0x01
    MCP23S17_IPOLA = 0x2
    MCP23S17_IPOLB = 0x3
    MCP23S17_GPIOA = 0x12
    MCP23S17_GPIOB = 0x13
    MCP23S17_OLATA = 0x14
    MCP23S17_OLATB = 0x15
    MCP23S17_IOCON = 0x0A
    MCP23S17_GPPUA = 0x0C
    MCP23S17_GPPUB = 0x0D

    """Bit field flags as documentined in the technical data sheet at
    http://ww1.microchip.com/downloads/en/DeviceDoc/21952b.pdf
```

```
    """
    IOCON_UNUSED = 0x01
    IOCON_INTPOL = 0x02
    IOCON_ODR = 0x04
    IOCON_HAEN = 0x08
    IOCON_DISSLW = 0x10
    IOCON_SEQOP = 0x20
    IOCON_MIRROR = 0x40
    IOCON_BANK_MODE = 0x80

    IOCON_INIT = 0x28  # IOCON_SEQOP and IOCON_HAEN from above

    MCP23S17_CMD_WRITE = 0x40
    MCP23S17_CMD_READ = 0x41

    def __init__(self, spiDevice=spidev.SpiDev(), device_id=0x00, pin_reset=-1, chip_select=-1, iGPIO=GPIO):
        """
        Constructor
        Initializes all attributes with 0.

        Keyword arguments:
        device_id -- The device ID of the component, i.e., the hardware address (default 0)
        pin_cs -- The pin of the Raspberry Pi that will be used as chip select of the MCP
        pin_reset -- The pin of the Raspberry Pi that will be used as reset of the MCP
        """
        self.device_id = device_id
        self.__pin_reset = pin_reset
        self.__chip_select = chip_select
        self.__GPIO = iGPIO
        self.__spiMode = 0b00
        self.GPIOA = 0
        self.GPIOB = 0
        self.IODIRA = 0
        self.IODIRB = 0
        self.GPPUA = 0
        self.GPPUB = 0
        self.__spi = spiDevice
        self.isInitialized = False

        self.__setupGPIO()
        self.isInitialized = True
        self.__writeRegister(MCP23S17.MCP23S17_IOCON, MCP23S17.IOCON_INIT)

        # set the pins as outputs
        for index in range(21, 29):
            self.setDirection(index, MCP23S17.DIR_OUTPUT)
        for index in range(1, 9):
            self.setDirection(index, MCP23S17.DIR_OUTPUT)

    def reset(self):
        """
```

```
        Sets the registers IODIR, GPPU AND GPIO to theire default values
        """
        if self.__pin_reset != -1:
            self.__GPIO.output(self.__pin_reset, self.__GPIO.LOW)
        else:
            self.__writeRegisterWord(MCP23S17.MCP23S17_GPIOA, 0x00)
            self.__writeRegisterWord(MCP23S17.MCP23S17_IODIRA, 0xFFFF)
            self.__writeRegisterWord(MCP23S17.MCP23S17_GPPUA, 0x00)

        self.GPIOA = 0
        self.GPIOB = 0
        self.IODIRA = 0
        self.IODIRB = 0
        self.GPPUA = 0
        self.GPPUB = 0

    def setPullupMode(self, pin, mode):
        """Enables or disables the pull-up mode for input pins.

        Parameters:
        pin -- The pin index (1 - 8 for BANK B and 21 - 28 for the BANK A)
        mode -- The pull-up mode (MCP23S17.PULLUP_ENABLED, MCP23S17.
    PULLUP_DISABLED)
        """

        assert pin in range(1, 9) or pin in range(21, 29)
        assert (mode == MCP23S17.PULLUP_ENABLED) or (mode == MCP23S17.
    PULLUP_DISABLED)
        assert self.isInitialized

        if pin in range(21, 29):
            register = MCP23S17.MCP23S17_GPPUA
            data = self.GPPUA
            noshifts = pin - 21
        else:
            register = MCP23S17.MCP23S17_GPPUB
            noshifts = (pin + 7) & 0x07
            data = self.GPPUB

        if mode == MCP23S17.PULLUP_ENABLED:
            data |= (1 << noshifts)
        else:
            data &= (~(1 << noshifts))

        self.__writeRegister(register, data)

        if pin in range(21, 29):
            self.GPPUA = data
        else:
            self.GPPUB = data

    def setDirection(self, pin, direction):
        """Sets the direction for a given pin.

        Parameters:
```

```
145         pin -- The pin index (0 - 15)
146         direction -- The direction of the pin (MCP23S17.DIR_INPUT, MCP23S17.
    DIR_OUTPUT)
147         """
148 
149         assert pin in range(1, 9) or pin in range(21, 29)
150         assert direction == MCP23S17.DIR_INPUT or direction == MCP23S17.
    DIR_OUTPUT
151         assert self.isInitialized
152 
153         if pin in range(21, 29):
154             register = MCP23S17.MCP23S17_IODIRA
155             data = self.IODIRA
156             noshifts = pin - 21
157         else:
158             register = MCP23S17.MCP23S17_IODIRB
159             noshifts = (pin + 7) & 0x07
160             data = self.IODIRB
161 
162         if direction == MCP23S17.DIR_INPUT:
163             data |= (1 << noshifts)
164         else:
165             data &= (~(1 << noshifts))
166 
167         self.__writeRegister(register, data)
168 
169         if pin in range(21, 29):
170             self.IODIRA = data
171         else:
172             self.IODIRB = data
173 
174     def digitalRead(self, pin):
175         """Reads the logical level of a given pin.
176 
177         Parameters:
178         pin -- The pin index (0 - 9) for GPIOB and (21 - 29) for GPIOA
179         Returns:
180          - MCP23S17.LEVEL_LOW, if the logical level of the pin is low,
181          - MCP23S17.LEVEL_HIGH, otherwise.
182         """
183 
184         assert pin in range(1, 9) or pin in range(21, 29)
185         assert self.isInitialized
186 
187         if pin in range(21, 29):
188             self.GPIOA = self.__readRegister(MCP23S17.MCP23S17_GPIOA)
189             if self.GPIOA & (1 << (pin - 21)) != 0:
190                 return MCP23S17.LEVEL_HIGH
191             else:
192                 return MCP23S17.LEVEL_LOW
193         else:
194             self.GPIOB = self.__readRegister(MCP23S17.MCP23S17_GPIOB)
195             pin += 7
196             pin &= 0x07
197             if (self.GPIOB & (1 << pin)) != 0:
```

```
                return MCP23S17.LEVEL_HIGH
            else:
                return MCP23S17.LEVEL_LOW

    def digitalWrite(self, pin, level):
        """Sets the level of a given pin.
        Parameters:
        pin -- The pin index (0 - 15)
        level -- The logical level to be set (LEVEL_LOW, LEVEL_HIGH)
        """

        assert self.isInitialized
        assert pin in range(1, 9) or pin in range(21, 29)
        assert (level == MCP23S17.LEVEL_HIGH) or (level == MCP23S17.LEVEL_LOW)

        if pin in range(21, 29):
            register = MCP23S17.MCP23S17_GPIOA
            data = self.GPIOA
            noshifts = pin - 21
        else:
            register = MCP23S17.MCP23S17_GPIOB
            noshifts = (pin + 7) & 0x07
            data = self.GPIOB

        if level == MCP23S17.LEVEL_HIGH:
            data |= (1 << noshifts)
        else:
            data &= (~(1 << noshifts))

        self.__writeRegister(register, data)

        if pin in range(21, 29):
            self.GPIOA = data
        else:
            self.GPIOB = data

    def writeGPIO(self, data):
        """Sets the data port value for all pins.
        Parameters:
        data - The 16-bit value to be set.
        """

        assert self.isInitialized

        self.GPIOA = (data & 0xFF)
        self.GPIOB = (data >> 8)
        self.__writeRegisterWord(MCP23S17.MCP23S17_GPIOA, data)

    def readGPIO(self):
        """Reads the data port value of all pins.
        Returns:
         - The 16-bit data port value        """

        assert self.isInitialized
        data = self.__readRegisterWord(MCP23S17.MCP23S17_GPIOA)
```

```
253         self.GPIOA = (data & 0xFF)
254         self.GPIOB = (data >> 8)
255         return data
256 
257     def __writeRegister(self, register, value):
258         assert self.isInitialized
259         command = MCP23S17.MCP23S17_CMD_WRITE | (self.device_id << 1)
260         self.__setupSPI()
261         self.__GPIO.output(self.__chip_select, self.__GPIO.LOW)
262         self.__spi.xfer2([command, register, value])
263         self.__GPIO.output(self.__chip_select, self.__GPIO.HIGH)
264 
265     def __readRegister(self, register):
266         assert self.isInitialized
267         command = MCP23S17.MCP23S17_CMD_READ | (self.device_id << 1)
268         self.__setupSPI()
269         self.__GPIO.output(self.__chip_select, self.__GPIO.LOW)
270         data = self.__spi.xfer2([command, register, 0])
271         self.__GPIO.output(self.__chip_select, self.__GPIO.HIGH)
272         return data[2]
273 
274     def __readRegisterWord(self, register):
275         assert self.isInitialized
276         buffer = [0, 0]
277         buffer[0] = self.__readRegister(register)
278         buffer[1] = self.__readRegister(register + 1)
279         return (buffer[1] << 8) | buffer[0]
280 
281     def __writeRegisterWord(self, register, data):
282         assert self.isInitialized
283         self.__writeRegister(register, data & 0xFF)
284         self.__writeRegister(register + 1, data >> 8)
285 
286     def __setupGPIO(self):
287         self.__GPIO.setup(self.__chip_select, self.__GPIO.OUT)
288         self.__GPIO.output(self.__chip_select, self.__GPIO.HIGH)
289         if self.__pin_reset != -1:
290             GPIO.setup(self.__pin_reset, self.__GPIO.OUT)
291             GPIO.output(self.__pin_reset, self.__GPIO.HIGH)
292 
293     def __setupSPI(self):
294         self.__spi.mode = self.__spiMode
295         time.sleep(0.01)
```

Listing D.1: MCP23S17 class code.

```
1  import time
2  import spidev
3  from RPiHVREMOTE.MCP23S17_v3 import MCP23S17
4 
5  class MAX1240(object):
6      """This class provides an abstraction of the 12-bit
7      ADC MAX1240 for the Raspberry Pi.
8      It is depndent on the Python packages spidev and RPi.GPIO, which can
9      be get from https://pypi.python.org/pypi/RPi.GPIO/0.5.11 and
10     https://pypi.python.org/pypi/spidev.
```

```
    """

    def __init__(self, spiDevice = spidev.SpiDev(), SHDN = -1, chip_select = -1,
     MCP = MCP23S17):
        """
        Constructor
        Initializes all attributes

        Keyword arguments:
        vref -- reference voltage for voltage calculations
        SHDN -- three-level shutdown input. When using external reference, must
    be open
        chip_select -- pin of the MCP23S17 port expander, to be used as chip
    select
        """

        self.__SHDN = SHDN
        self.__chip_select = chip_select
        self.__MCP = MCP
        self.__spi = spiDevice
        self.__spiMode = 0b00
        self.__voltageBits = 0
        self.__isInitialized = False

        self.__setupGPIO()
        self.__isInitialized = True

    def readVoltage(self):
        """
        Reads the voltage as a word of 12 bits. Bytes are sent with MSB first,
    and bits are clocked
        at the rising edge o SCLK
        Returns and integer less then 4096
        """
        assert self.__isInitialized == True
        if self.__SHDN != -1:
            self.__MCP.digitalWrite(self.__SHDN, self.__MCP.LEVEL_HIGH)

        self.__MCP.digitalWrite(self.__chip_select, self.__MCP.LEVEL_LOW)
        time.sleep(0.01) #wait 9 us for conversion to complete (time specified
    is actually 7.5 us)
        self.__setupSPI()
        data = self.__spi.xfer2([0, 0])
        self.__MCP.digitalWrite(self.__chip_select, self.__MCP.LEVEL_HIGH)

        first_byte = data[0]
        second_byte = data[1]
        # The first bit is '1' and only represents the end of conversion
        # so it must be deleted
        first_byte = (first_byte & 0x7F) << 5
        second_byte = second_byte >> 3
        self.__voltageBits = first_byte | second_byte

        if self.__SHDN != -1:
            self.__MCP.digitalWrite(self.__SHDN, self.__MCP.LEVEL_LOW)
```

```
61
62          return self.__voltageBits
63
64      def __setupGPIO(self):
65          IODIR = (self.__MCP.IODIRB << 8) | self.__MCP.IODIRA
66          if self.__chip_select in range(1, 9):
67              noshifts = self.__chip_select + 7
68          else:
69              noshifts = self.__chip_select - 21
70          if IODIR & (1 << noshifts) != 0x00:
71              self.__MCP.setDirection(self.__chip_select, self.__MCP.DIR_OUTPUT)
72          self.__MCP.digitalWrite(self.__chip_select, self.__MCP.LEVEL_HIGH)
73          if self.__SHDN != -1:
74              self.__MCP.setDirection(self.__SHDN, self.__MCP.DIR_OUTPUT)
75              self.__MCP.digitalWrite(self.__SHDN, self.__MCP.LEVEL_LOW)
76
77      def __setupSPI(self):
78          self.__spi.mode = self.__spiMode
79          time.sleep(0.01)
```

Listing D.2: MAX1240 class code.

```
1  from PyQt5 import QtCore, QtGui, QtWidgets
2  import spidev
3  import time
4  import datetime
5  import statistics
6  import numpy as np
7  import pyqtgraph as pg
8  import RPi.GPIO as GPIO
9  from RPiHVREMOTE.MAX1240_v2 import MAX1240
10 from RPiHVREMOTE.MCP23S17_v3 import MCP23S17
11
12
13 MCP_CHIP_SELECT = 13
14 VREF = 3.3
15 spi = spidev.SpiDev()
16 spi.open(0, 0)
17 spi.max_speed_hz = 976000
18 spi.no_cs = True
19
20 GPIO.setwarnings(False)
21 GPIO.setmode(GPIO.BOARD)
22
23 mcp1 = MCP23S17(spi, 0b000, -1, MCP_CHIP_SELECT, GPIO)
24
25 class TimeAxisItem(pg.AxisItem):
26     def __init__(self, *args, **kwargs):
27         super().__init__(*args, **kwargs)
28         self.setLabel(text='Time', units=None)
29         self.enableAutoSIPrefix(False)
30
31     def tickStrings(self, values, scale, spacing):
32         return [datetime.datetime.fromtimestamp(value).strftime("%H:%M:%S") for
    value in values]
33
```

```
34
35 class Ui_MainWindow(object):
36     def setupUi(self, MainWindow):
37         MainWindow.setObjectName("MainWindow")
38         MainWindow.resize(1643, 588)
39         self.centralwidget = QtWidgets.QWidget(MainWindow)
40         self.centralwidget.setObjectName("centralwidget")
41         self.groupBox = QtWidgets.QGroupBox(self.centralwidget)
42         self.groupBox.setGeometry(QtCore.QRect(320, 0, 1321, 551))
43         self.groupBox.setObjectName("groupBox")
44         self.label_2 = QtWidgets.QLabel(self.groupBox)
45         self.label_2.setGeometry(QtCore.QRect(120, 50, 201, 31))
46         self.label_2.setFrameShape(QtWidgets.QFrame.Box)
47         self.label_2.setAlignment(QtCore.Qt.AlignCenter)
48         self.label_2.setText("")
49         self.label_2.setObjectName("label_2")
50         self.label_3 = QtWidgets.QLabel(self.groupBox)
51         self.label_3.setGeometry(QtCore.QRect(340, 50, 201, 31))
52         self.label_3.setFrameShape(QtWidgets.QFrame.Box)
53         self.label_3.setAlignment(QtCore.Qt.AlignCenter)
54         self.label_3.setText("")
55         self.label_3.setObjectName("label_3")
56         self.splitter = QtWidgets.QSplitter(self.groupBox)
57         self.splitter.setGeometry(QtCore.QRect(10, 20, 81, 491))
58         self.splitter.setOrientation(QtCore.Qt.Vertical)
59         self.splitter.setObjectName("splitter")
60         self.radioButton = QtWidgets.QRadioButton(self.splitter)
61         self.radioButton.setObjectName("radioButton")
62         self.radioButton_2 = QtWidgets.QRadioButton(self.splitter)
63         self.radioButton_2.setObjectName("radioButton_2")
64         self.radioButton_3 = QtWidgets.QRadioButton(self.splitter)
65         self.radioButton_3.setObjectName("radioButton_3")
66         self.radioButton_4 = QtWidgets.QRadioButton(self.splitter)
67         self.radioButton_4.setObjectName("radioButton_4")
68         self.radioButton_5 = QtWidgets.QRadioButton(self.splitter)
69         self.radioButton_5.setObjectName("radioButton_5")
70         self.radioButton_6 = QtWidgets.QRadioButton(self.splitter)
71         self.radioButton_6.setObjectName("radioButton_6")
72         self.radioButton_7 = QtWidgets.QRadioButton(self.splitter)
73         self.radioButton_7.setObjectName("radioButton_7")
74         self.radioButton_8 = QtWidgets.QRadioButton(self.splitter)
75         self.radioButton_8.setObjectName("radioButton_8")
76         self.radioButton_9 = QtWidgets.QRadioButton(self.splitter)
77         self.radioButton_9.setObjectName("radioButton_9")
78         self.radioButton_10 = QtWidgets.QRadioButton(self.splitter)
79         self.radioButton_10.setObjectName("radioButton_10")
80         self.radioButton_11 = QtWidgets.QRadioButton(self.splitter)
81         self.radioButton_11.setObjectName("radioButton_11")
82         self.radioButton_12 = QtWidgets.QRadioButton(self.splitter)
83         self.radioButton_12.setObjectName("radioButton_12")
84         self.tabWidget = QtWidgets.QTabWidget(self.groupBox)
85         self.tabWidget.setGeometry(QtCore.QRect(110, 90, 901, 441))
86         self.tabWidget.setObjectName("tabWidget")
87         self.tab = QtWidgets.QWidget()
88         self.tab.setObjectName("tab")
```

```
89          self.graphicsView = pg.PlotWidget(self.tab,
90              #labels={'left': 'Reading / mV'},
91              axisItems={'bottom': TimeAxisItem(orientation='bottom')})
92          self.graphicsView.setGeometry(QtCore.QRect(10, 10, 871, 391))
93          self.graphicsView.setObjectName("graphicsView")
94          self.tabWidget.addTab(self.tab, "")
95          self.tab_2 = QtWidgets.QWidget()
96          self.tab_2.setObjectName("tab_2")
97          self.graphicsView_2 = pg.PlotWidget(self.tab_2,
98              #labels={'left': 'Reading / mV'},
99              axisItems={'bottom': TimeAxisItem(orientation='bottom')})
100         self.graphicsView_2.setGeometry(QtCore.QRect(10, 10, 871, 391))
101         self.graphicsView_2.setObjectName("graphicsView_2")
102         self.tabWidget.addTab(self.tab_2, "")
103         self.tab_3 = QtWidgets.QWidget()
104         self.tab_3.setObjectName("tab_3")
105         self.graphicsView_3 = pg.PlotWidget(self.tab_3,
106             #labels={'left': 'Reading / mV'},
107             axisItems={'bottom': TimeAxisItem(orientation='bottom')})
108         self.graphicsView_3.setGeometry(QtCore.QRect(10, 10, 871, 391))
109         self.graphicsView_3.setObjectName("graphicsView_3")
110         self.tabWidget.addTab(self.tab_3, "")
111         self.tab_4 = QtWidgets.QWidget()
112         self.tab_4.setObjectName("tab_4")
113         self.graphicsView_4 = pg.PlotWidget(self.tab_4,
114             #labels={'left': 'Reading / mV'},
115             axisItems={'bottom': TimeAxisItem(orientation='bottom')})
116         self.graphicsView_4.setGeometry(QtCore.QRect(10, 10, 871, 391))
117         self.graphicsView_4.setObjectName("graphicsView_4")
118         self.tabWidget.addTab(self.tab_4, "")
119         self.tab_5 = QtWidgets.QWidget()
120         self.tab_5.setObjectName("tab_5")
121         self.graphicsView_5 = pg.PlotWidget(self.tab_5,
122             #labels={'left': 'Reading / mV'},
123             axisItems={'bottom': TimeAxisItem(orientation='bottom')})
124         self.graphicsView_5.setGeometry(QtCore.QRect(10, 10, 871, 391))
125         self.graphicsView_5.setObjectName("graphicsView_5")
126         self.tabWidget.addTab(self.tab_5, "")
127         self.tab_6 = QtWidgets.QWidget()
128         self.tab_6.setObjectName("tab_6")
129         self.graphicsView_6 = pg.PlotWidget(self.tab_6,
130             #labels={'left': 'Reading / mV'},
131             axisItems={'bottom': TimeAxisItem(orientation='bottom')})
132         self.graphicsView_6.setGeometry(QtCore.QRect(10, 10, 871, 391))
133         self.graphicsView_6.setObjectName("graphicsView_6")
134         self.tabWidget.addTab(self.tab_6, "")
135         self.tab_7 = QtWidgets.QWidget()
136         self.tab_7.setObjectName("tab_7")
137         self.graphicsView_7 = pg.PlotWidget(self.tab_7,
138             #labels={'left': 'Reading / mV'},
139             axisItems={'bottom': TimeAxisItem(orientation='bottom')})
140         self.graphicsView_7.setGeometry(QtCore.QRect(10, 10, 871, 391))
141         self.graphicsView_7.setObjectName("graphicsView_7")
142         self.tabWidget.addTab(self.tab_7, "")
143         self.tab_8 = QtWidgets.QWidget()
```

```
144         self.tab_8.setObjectName("tab_8")
145         self.graphicsView_8 = pg.PlotWidget(self.tab_8,
146             #labels={'left': 'Reading / mV'},
147             axisItems={'bottom': TimeAxisItem(orientation='bottom')})
148         self.graphicsView_8.setGeometry(QtCore.QRect(10, 10, 871, 391))
149         self.graphicsView_8.setObjectName("graphicsView_8")
150         self.tabWidget.addTab(self.tab_8, "")
151         self.tab_9 = QtWidgets.QWidget()
152         self.tab_9.setObjectName("tab_9")
153         self.graphicsView_9 = pg.PlotWidget(self.tab_9,
154             #labels={'left': 'Reading / mV'},
155             axisItems={'bottom': TimeAxisItem(orientation='bottom')})
156         self.graphicsView_9.setGeometry(QtCore.QRect(10, 10, 871, 391))
157         self.graphicsView_9.setObjectName("graphicsView_9")
158         self.tabWidget.addTab(self.tab_9, "")
159         self.tab_10 = QtWidgets.QWidget()
160         self.tab_10.setObjectName("tab_10")
161         self.graphicsView_10 = pg.PlotWidget(self.tab_10,
162             #labels={'left': 'Reading / mV'},
163             axisItems={'bottom': TimeAxisItem(orientation='bottom')})
164         self.graphicsView_10.setGeometry(QtCore.QRect(10, 10, 871, 391))
165         self.graphicsView_10.setObjectName("graphicsView_10")
166         self.tabWidget.addTab(self.tab_10, "")
167         self.tab_11 = QtWidgets.QWidget()
168         self.tab_11.setObjectName("tab_11")
169         self.graphicsView_11 = pg.PlotWidget(self.tab_11,
170             #labels={'left': 'Reading / mV'},
171             axisItems={'bottom': TimeAxisItem(orientation='bottom')})
172         self.graphicsView_11.setGeometry(QtCore.QRect(10, 10, 871, 391))
173         self.graphicsView_11.setObjectName("graphicsView_11")
174         self.tabWidget.addTab(self.tab_11, "")
175         self.tab_12 = QtWidgets.QWidget()
176         self.tab_12.setObjectName("tab_12")
177         self.graphicsView_12 = pg.PlotWidget(self.tab_12,
178             #labels={'left': 'Reading / mV'},
179             axisItems={'bottom': TimeAxisItem(orientation='bottom')})
180         self.graphicsView_12.setGeometry(QtCore.QRect(10, 10, 871, 391))
181         self.graphicsView_12.setObjectName("graphicsView_12")
182         self.tabWidget.addTab(self.tab_12, "")
183         self.progressBar = QtWidgets.QProgressBar(self.groupBox)
184         self.progressBar.setGeometry(QtCore.QRect(1030, 380, 271, 23))
185         self.progressBar.setProperty("value", 0)
186         self.progressBar.setObjectName("progressBar")
187         self.pushButton = QtWidgets.QPushButton(self.groupBox)
188         self.pushButton.setGeometry(QtCore.QRect(1150, 430, 93, 31))
189         self.pushButton.setObjectName("pushButton")
190         self.label_10 = QtWidgets.QLabel(self.groupBox)
191         self.label_10.setGeometry(QtCore.QRect(120, 20, 201, 16))
192         self.label_10.setObjectName("label_10")
193         self.label_11 = QtWidgets.QLabel(self.groupBox)
194         self.label_11.setGeometry(QtCore.QRect(340, 20, 221, 16))
195         self.label_11.setObjectName("label_11")
196         self.splitter_5 = QtWidgets.QSplitter(self.groupBox)
197         self.splitter_5.setGeometry(QtCore.QRect(1030, 210, 271, 22))
198         self.splitter_5.setOrientation(QtCore.Qt.Horizontal)
```

```
        self.splitter_5.setObjectName("splitter_5")
        self.label_5 = QtWidgets.QLabel(self.splitter_5)
        self.label_5.setObjectName("label_5")
        self.spinBox = QtWidgets.QSpinBox(self.splitter_5)
        self.spinBox.setObjectName("spinBox")
        self.splitter_6 = QtWidgets.QSplitter(self.groupBox)
        self.splitter_6.setGeometry(QtCore.QRect(1030, 150, 271, 22))
        self.splitter_6.setOrientation(QtCore.Qt.Horizontal)
        self.splitter_6.setObjectName("splitter_6")
        self.label_12 = QtWidgets.QLabel(self.splitter_6)
        self.label_12.setObjectName("label_12")
        self.comboBox = QtWidgets.QComboBox(self.splitter_6)
        self.comboBox.setObjectName("comboBox")
        self.splitter_7 = QtWidgets.QSplitter(self.groupBox)
        self.splitter_7.setGeometry(QtCore.QRect(1030, 270, 271, 22))
        self.splitter_7.setOrientation(QtCore.Qt.Horizontal)
        self.splitter_7.setObjectName("splitter_7")
        self.label_13 = QtWidgets.QLabel(self.splitter_7)
        self.label_13.setObjectName("label_13")
        self.spinBox_2 = QtWidgets.QSpinBox(self.splitter_7)
        self.spinBox_2.setObjectName("spinBox_2")
        self.splitter_5.raise_()
        self.splitter.raise_()
        self.label_2.raise_()
        self.label_3.raise_()
        self.tabWidget.raise_()
        self.progressBar.raise_()
        self.pushButton.raise_()
        self.label_10.raise_()
        self.label_11.raise_()
        self.splitter_6.raise_()
        self.splitter_7.raise_()
        self.groupBox_2 = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox_2.setGeometry(QtCore.QRect(-1, -1, 211, 201))
        self.groupBox_2.setObjectName("groupBox_2")
        self.splitter_2 = QtWidgets.QSplitter(self.groupBox_2)
        self.splitter_2.setGeometry(QtCore.QRect(10, 20, 171, 141))
        self.splitter_2.setOrientation(QtCore.Qt.Vertical)
        self.splitter_2.setObjectName("splitter_2")
        self.checkBox = QtWidgets.QCheckBox(self.splitter_2)
        self.checkBox.setObjectName("checkBox")
        self.checkBox_2 = QtWidgets.QCheckBox(self.splitter_2)
        self.checkBox_2.setObjectName("checkBox_2")
        self.checkBox_3 = QtWidgets.QCheckBox(self.splitter_2)
        self.checkBox_3.setObjectName("checkBox_3")
        self.checkBox_4 = QtWidgets.QCheckBox(self.splitter_2)
        self.checkBox_4.setObjectName("checkBox_4")
        self.groupBox_3 = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox_3.setGeometry(QtCore.QRect(0, 220, 311, 301))
        self.groupBox_3.setObjectName("groupBox_3")
        self.splitter_4 = QtWidgets.QSplitter(self.groupBox_3)
        self.splitter_4.setGeometry(QtCore.QRect(10, 70, 311, 16))
        self.splitter_4.setOrientation(QtCore.Qt.Horizontal)
        self.splitter_4.setObjectName("splitter_4")
        self.label_8 = QtWidgets.QLabel(self.splitter_4)
```

```
        self.label_8.setObjectName("label_8")
        self.HV1 = QtWidgets.QSlider(self.splitter_4)
        self.HV1.setOrientation(QtCore.Qt.Horizontal)
        self.HV1.setTickPosition(QtWidgets.QSlider.NoTicks)
        self.HV1.setObjectName("HV1")
        self.label_9 = QtWidgets.QLabel(self.splitter_4)
        self.label_9.setScaledContents(False)
        self.label_9.setObjectName("label_9")
        self.splitter_3 = QtWidgets.QSplitter(self.groupBox_3)
        self.splitter_3.setGeometry(QtCore.QRect(10, 210, 311, 16))
        self.splitter_3.setOrientation(QtCore.Qt.Horizontal)
        self.splitter_3.setObjectName("splitter_3")
        self.label_6 = QtWidgets.QLabel(self.splitter_3)
        self.label_6.setObjectName("label_6")
        self.HV2 = QtWidgets.QSlider(self.splitter_3)
        self.HV2.setOrientation(QtCore.Qt.Horizontal)
        self.HV2.setObjectName("HV2")
        self.label_7 = QtWidgets.QLabel(self.splitter_3)
        self.label_7.setObjectName("label_7")
        self.label = QtWidgets.QLabel(self.groupBox_3)
        self.label.setGeometry(QtCore.QRect(10, 40, 55, 16))
        self.label.setObjectName("label")
        self.label_4 = QtWidgets.QLabel(self.groupBox_3)
        self.label_4.setGeometry(QtCore.QRect(10, 180, 55, 16))
        self.label_4.setObjectName("label_4")
        self.label_14 = QtWidgets.QLabel(self.groupBox_3)
        self.label_14.setGeometry(QtCore.QRect(10, 110, 301, 20))
        font = QtGui.QFont()
        font.setPointSize(14)
        font.setBold(True)
        font.setUnderline(False)
        font.setWeight(75)
        font.setStrikeOut(False)
        font.setKerning(False)
        self.label_14.setFont(font)
        self.label_14.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.label_14.setAlignment(QtCore.Qt.AlignCenter)
        self.label_14.setObjectName("label_14")
        self.label_15 = QtWidgets.QLabel(self.groupBox_3)
        self.label_15.setGeometry(QtCore.QRect(10, 250, 301, 20))
        font = QtGui.QFont()
        font.setPointSize(14)
        font.setBold(True)
        font.setUnderline(False)
        font.setWeight(75)
        font.setStrikeOut(False)
        font.setKerning(False)
        self.label_15.setFont(font)
        self.label_15.setAlignment(QtCore.Qt.AlignCenter)
        self.label_15.setObjectName("label_15")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 1643, 26))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
```

```
309         self.statusbar = QtWidgets.QStatusBar(MainWindow)
310         self.statusbar.setObjectName("statusbar")
311         MainWindow.setStatusBar(self.statusbar)
312 
313         self.retranslateUi(MainWindow)
314         self.tabWidget.setCurrentIndex(0)
315         QtCore.QMetaObject.connectSlotsByName(MainWindow)
316 
317     def retranslateUi(self, MainWindow):
318         _translate = QtCore.QCoreApplication.translate
319         MainWindow.setWindowTitle(_translate("MainWindow", "HV Board"))
320         self.groupBox.setTitle(_translate("MainWindow", "Reading"))
321         self.radioButton.setText(_translate("MainWindow", "HV1_V"))
322         self.radioButton_2.setText(_translate("MainWindow", "HV1_I"))
323         self.radioButton_3.setText(_translate("MainWindow", "HV2_V"))
324         self.radioButton_4.setText(_translate("MainWindow", "HV2_I"))
325         self.radioButton_5.setText(_translate("MainWindow", "NA_V"))
326         self.radioButton_6.setText(_translate("MainWindow", "NA_I"))
327         self.radioButton_7.setText(_translate("MainWindow", "PA_V"))
328         self.radioButton_8.setText(_translate("MainWindow", "PA_I"))
329         self.radioButton_9.setText(_translate("MainWindow", "D_V"))
330         self.radioButton_10.setText(_translate("MainWindow", "D_I"))
331         self.radioButton_11.setText(_translate("MainWindow", "TEMP1"))
332         self.radioButton_12.setText(_translate("MainWindow", "TEMP2"))
333         self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab), _translate("
    MainWindow", "HV1_V"))
334         self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_2), _translate
    ("MainWindow", "HV1_I"))
335         self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_3), _translate
    ("MainWindow", "HV2_V"))
336         self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_4), _translate
    ("MainWindow", "HV2_I"))
337         self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_5), _translate
    ("MainWindow", "NA_V"))
338         self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_6), _translate
    ("MainWindow", "NA_I"))
339         self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_7), _translate
    ("MainWindow", "PA_V"))
340         self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_8), _translate
    ("MainWindow", "PA_I"))
341         self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_9), _translate
    ("MainWindow", "D_V"))
342         self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_10),
    _translate("MainWindow", "D_I"))
343         self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_11),
    _translate("MainWindow", "TEMP1"))
344         self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_12),
    _translate("MainWindow", "TEMP2"))
345         self.pushButton.setText(_translate("MainWindow", "Start"))
346         self.label_10.setText(_translate("MainWindow", "ADC counts"))
347         self.label_11.setText(_translate("MainWindow", "Voltage/Current/
    Temperature"))
348         self.label_5.setText(_translate("MainWindow", "Number of cycles"))
349         self.label_12.setText(_translate("MainWindow", "Pin to read"))
350         self.label_13.setText(_translate("MainWindow", "Time between readings (s
```

```
    )"))
        self.groupBox_2.setTitle(_translate("MainWindow", "DC/DC Enable"))
        self.checkBox.setText(_translate("MainWindow", "Enable 3.3 V"))
        self.checkBox_2.setText(_translate("MainWindow", "Enable +-12 V"))
        self.checkBox_3.setText(_translate("MainWindow", "Enable HV1"))
        self.checkBox_4.setText(_translate("MainWindow", "Enable HV2"))
        self.groupBox_3.setTitle(_translate("MainWindow", "Set HV"))
        self.label_8.setText(_translate("MainWindow", " -830 V"))
        self.label_9.setText(_translate("MainWindow", " -950 V"))
        self.label_6.setText(_translate("MainWindow", " -830 V"))
        self.label_7.setText(_translate("MainWindow", " -950 V"))
        self.label.setText(_translate("MainWindow", "HV1"))
        self.label_4.setText(_translate("MainWindow", "HV2"))
        self.label_14.setText(_translate("MainWindow", "HV1 is set to -950 V"))
        self.label_15.setText(_translate("MainWindow", "HV2 is set to -950 V"))


        self.comboBox.addItems(["HV1_V","HV1_I","HV2_V","HV2_I","NA_V","NA_I","
    PA_V","PA_I","D_V","D_I","TEMP1","TEMP2"])
        self.pushButton.clicked.connect(self.plotting)
        self.spinBox.setMinimum(2)
        self.spinBox.setMaximum(99999)
        self.spinBox.setValue(2)
        self.spinBox_2.setMinimum(1)
        self.spinBox_2.setMaximum(99999)
        self.spinBox_2.setValue(1)

        self.radioButton.toggled.connect(self.read)
        self.radioButton_2.toggled.connect(self.read)
        self.radioButton_3.toggled.connect(self.read)
        self.radioButton_4.toggled.connect(self.read)
        self.radioButton_5.toggled.connect(self.read)
        self.radioButton_6.toggled.connect(self.read)
        self.radioButton_7.toggled.connect(self.read)
        self.radioButton_8.toggled.connect(self.read)
        self.radioButton_9.toggled.connect(self.read)
        self.radioButton_10.toggled.connect(self.read)
        self.radioButton_11.toggled.connect(self.read)
        self.radioButton_12.toggled.connect(self.read)


        self.checkBox.toggled.connect(self.Enable)
        self.checkBox_2.toggled.connect(self.Enable)
        self.checkBox_3.toggled.connect(self.Enable)
        self.checkBox_4.toggled.connect(self.Enable)

        self.HV1.setMinimum(0)
        self.HV1.setMaximum(1)
        self.HV1.setValue(1)


        self.HV2.setMinimum(0)
        self.HV2.setMaximum(1)
        self.HV2.setValue(1)

```

```
404
405         self.HV1.valueChanged.connect(self.HVSel)
406         self.HV2.valueChanged.connect(self.HVSel)
407
408
409
410     def read(self):
411         if self.radioButton.isChecked() == True:
412             a = "0000"
413             s = " V"
414             r = 0
415         if self.radioButton_2.isChecked() == True:
416             a = "0001"
417             s = " mA"
418             r = 1
419         if self.radioButton_3.isChecked() == True:
420             a = "1110"
421             s = " V"
422             r = 0
423         if self.radioButton_4.isChecked() == True:
424             a = "1111"
425             s = " mA"
426             r = 1
427         if self.radioButton_5.isChecked() == True:
428             a = "1000"
429             s = " V"
430             r = 2
431         if self.radioButton_6.isChecked() == True:
432             a = "1001"
433             s = " mA"
434             r = 3
435         if self.radioButton_7.isChecked() == True:
436             a = "0110"
437             s = " V"
438             r = 4
439         if self.radioButton_8.isChecked() == True:
440             a = "0111"
441             s = " mA"
442             r = 3
443         if self.radioButton_9.isChecked() == True:
444             a = "0101"
445             s = " V"
446             r = 5
447         if self.radioButton_10.isChecked() == True:
448             a = "0100"
449             s = " mA"
450             r = 3
451         if self.radioButton_11.isChecked() == True:
452             a = "1010"
453             s = " C"
454             r = 6
455         if self.radioButton_12.isChecked() == True:
456             a = "1011"
457             s = " C"
458             r = 6
```

```
459
460
461         adc = MAX1240(spi, -1, 21, mcp1)
462
463         mcp1.digitalWrite(22, MCP23S17.LEVEL_HIGH)
464
465         if int(a[3]) == 0:
466             mcp1.digitalWrite(23, MCP23S17.LEVEL_LOW)
467             time.sleep(0.5)
468         else:
469             mcp1.digitalWrite(23, MCP23S17.LEVEL_HIGH)
470             time.sleep(0.5)
471         if int(a[2]) == 0:
472             mcp1.digitalWrite(24, MCP23S17.LEVEL_LOW)
473             time.sleep(0.5)
474         else:
475             mcp1.digitalWrite(24, MCP23S17.LEVEL_HIGH)
476             time.sleep(0.5)
477         if int(a[1]) == 0:
478             mcp1.digitalWrite(25, MCP23S17.LEVEL_LOW)
479             time.sleep(0.5)
480         else:
481             mcp1.digitalWrite(25, MCP23S17.LEVEL_HIGH)
482             time.sleep(0.5)
483         if int(a[0]) == 0:
484             mcp1.digitalWrite(26, MCP23S17.LEVEL_LOW)
485             time.sleep(0.5)
486         else:
487             mcp1.digitalWrite(26, MCP23S17.LEVEL_HIGH)
488             time.sleep(0.5)
489
490         madc = np.zeros(10)
491         m = np.zeros(10)
492         for i in range(0,10):
493             Voltb=adc.readVoltage()
494             Volt = Voltb*VREF/4096
495             if r == 0:
496                 Val = Volt/0.005
497             if r == 1:
498                 Val = Volt/0.5
499             if r == 2:
500                 Val = 43*Volt/2979750
501             if r == 3:
502                 Val = Volt/(5*200)
503             if r == 4:
504                 Val = 429*Volt/89
505             if r == 5:
506                 Val = 139*Volt/105
507             if r == 6:
508                 Val = ((Volt*1000000)/1000)-273.15
509             madc[i] = Voltb
510             m[i] = Val
511             time.sleep(0.5)
512             if i == 9:
513                 mcp1.digitalWrite(22, MCP23S17.LEVEL_LOW)
```

```
                mcp1.digitalWrite(23, MCP23S17.LEVEL_LOW)
                mcp1.digitalWrite(24, MCP23S17.LEVEL_LOW)
                mcp1.digitalWrite(25, MCP23S17.LEVEL_LOW)
                mcp1.digitalWrite(26, MCP23S17.LEVEL_LOW)
                mcp1.digitalWrite(21, MCP23S17.LEVEL_LOW)

        self.label_2.setText(str(int(statistics.mean(madc))))
        self.label_3.setText(str(round(statistics.mean(m),3)) + s)


    def Enable(self):

        if self.checkBox.isChecked() == True:
            mcp1.digitalWrite(2, MCP23S17.LEVEL_HIGH)
            time.sleep(0.5)
        else:
            mcp1.digitalWrite(2, MCP23S17.LEVEL_LOW)
            time.sleep(0.5)

        if self.checkBox_2.isChecked() == True:
            mcp1.digitalWrite(1, MCP23S17.LEVEL_HIGH)
            time.sleep(0.5)
        else:
            mcp1.digitalWrite(1, MCP23S17.LEVEL_LOW)
            time.sleep(0.5)

        if self.checkBox_3.isChecked() == True:
            mcp1.digitalWrite(8, MCP23S17.LEVEL_HIGH)
            time.sleep(0.5)
        else:
            mcp1.digitalWrite(8, MCP23S17.LEVEL_LOW)
            time.sleep(0.5)

        if self.checkBox_4.isChecked() == True:
            mcp1.digitalWrite(27, MCP23S17.LEVEL_HIGH)
            time.sleep(0.5)
        else:
            mcp1.digitalWrite(27, MCP23S17.LEVEL_LOW)
            time.sleep(0.5)


    def HVSel(self):

        if self.HV1.value() == 0:
            self.label_14.setText("")
            mcp1.digitalWrite(3, MCP23S17.LEVEL_HIGH)
            self.label_14.setText("HV1 is set to -830 V")
            time.sleep(0.5)
        else:
            self.label_14.setText("")
            mcp1.digitalWrite(3, MCP23S17.LEVEL_LOW)
            self.label_14.setText("HV1 is set to -950 V")
            time.sleep(0.5)

        if self.HV2.value() == 0:
```

```
            self.label_15.setText("")
            mcp1.digitalWrite(28, MCP23S17.LEVEL_HIGH)
            self.label_15.setText("HV2 is set to -830 V")
            time.sleep(0.5)
        else:
            self.label_15.setText("")
            mcp1.digitalWrite(28, MCP23S17.LEVEL_LOW)
            self.label_15.setText("HV2 is set to -950 V")
            time.sleep(0.5)


    def plotting(self):
        t = self.comboBox.currentIndex()
        if t == 0:
            a = "0000"
        if t == 1:
            a = "0001"
        if t == 2:
            a = "1110"
        if t == 3:
            a = "1111"
        if t == 4:
            a = "1000"
        if t == 5:
            a = "1001"
        if t == 6:
            a = "0110"
        if t == 7:
            a = "0111"
        if t == 8:
            a = "0101"
        if t == 9:
            a = "0100"
        if t == 10:
            a = "1010"
        if t == 11:
            a = "1011"


        VREF = 3.3

        adc = MAX1240(spi, -1, 21, mcp1)

        mcp1.digitalWrite(22, MCP23S17.LEVEL_HIGH)


        if int(a[3]) == 0:
            mcp1.digitalWrite(23, MCP23S17.LEVEL_LOW)
            time.sleep(0.5)
        else:
            mcp1.digitalWrite(23, MCP23S17.LEVEL_HIGH)
            time.sleep(0.5)
        if int(a[2]) == 0:
            mcp1.digitalWrite(24, MCP23S17.LEVEL_LOW)
            time.sleep(0.5)
```

```
        else:
            mcp1.digitalWrite(24, MCP23S17.LEVEL_HIGH)
            time.sleep(0.5)
        if int(a[1]) == 0:
            mcp1.digitalWrite(25, MCP23S17.LEVEL_LOW)
            time.sleep(0.5)
        else:
            mcp1.digitalWrite(25, MCP23S17.LEVEL_HIGH)
            time.sleep(0.5)
        if int(a[0]) == 0:
            mcp1.digitalWrite(26, MCP23S17.LEVEL_LOW)
            time.sleep(0.5)
        else:
            mcp1.digitalWrite(26, MCP23S17.LEVEL_HIGH)
            time.sleep(0.5)


        u = self.spinBox.value()
        s = self.spinBox_2.value()
        k = np.zeros(u)
        x = np.zeros(u)
        self.progressBar.setValue(0)
        self.progressBar.setMaximum(u)

        for i in range(u):
            x[i] = int(time.mktime(datetime.datetime.now().timetuple()))
            madc = np.zeros(10)

            for j in range(10):
                Voltb = adc.readVoltage()
                Volt = Voltb*VREF/4096
                madc[j] = Volt
                time.sleep(0.5)

            if r == 0:
                Val = statistics.mean(madc)/0.005
            if r == 1:
                Val = statistics.mean(madc)/0.5
            if r == 2:
                Val = 43*statistics.mean(madc)2979750
            if r == 3:
                Val = statistics.mean(madc)/(5*200)
            if r == 4:
                Val = 429*statistics.mean(madc)/89
            if r == 5:
                Val = 139*statistics.mean(madc)/105
            if r == 6:
                Val = ((statistics.mean(madc)*1000000)/1000)-273.15
            k[i] = Val
            self.progressBar.setValue(i+1)
            time.sleep(s)

            if i == u-1:
                mcp1.digitalWrite(22, MCP23S17.LEVEL_LOW)
                mcp1.digitalWrite(23, MCP23S17.LEVEL_LOW)
```

```
                mcp1.digitalWrite(24, MCP23S17.LEVEL_LOW)
                mcp1.digitalWrite(25, MCP23S17.LEVEL_LOW)
                mcp1.digitalWrite(26, MCP23S17.LEVEL_LOW)
                mcp1.digitalWrite(21, MCP23S17.LEVEL_LOW)

        if t == 0:
            self.graphicsView.clear()
            self.graphicsView.plot(x,k)
        if t == 1:
            self.graphicsView_2.clear()
            self.graphicsView_2.plot(x,k)
        if t == 2:
            self.graphicsView_3.clear()
            self.graphicsView_3.plot(x,k)
        if t == 3:
            self.graphicsView_4.clear()
            self.graphicsView_4.plot(x,k)
        if t == 4:
            self.graphicsView_5.clear()
            self.graphicsView_5.plot(x,k)
        if t == 5:
            self.graphicsView_6.clear()
            self.graphicsView_6.plot(x,k)
        if t == 6:
            self.graphicsView_7.clear()
            self.graphicsView_7.plot(x,k)
        if t == 7:
            self.graphicsView_8.clear()
            self.graphicsView_8.plot(x,k)
        if t == 8:
            self.graphicsView_9.clear()
            self.graphicsView_9.plot(x,k)
        if t == 9:
            self.graphicsView_10.clear()
            self.graphicsView_10.plot(x,k)
        if t == 10:
            self.graphicsView_11.clear()
            self.graphicsView_11.plot(x,k)
        if t == 11:
            self.graphicsView_12.clear()
            self.graphicsView_12.plot(x,k)
        self.tabWidget.setCurrentIndex(t)



if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Listing D.3: Graphical User Interface code of the Power Supply board.