UNIVERSITAT DE BARCELONA

Projecte fi de carrera
ENGINYERIA EN ELECTRÒNICA

Facultat de Física

RADIATION TOLERANT AND CONTROLABLE POWER SUPPLY DESIGN AND IMPLEMENTATION FOR THE VFE OF SPD AT LHCB DETECTOR

Barcelona, Setembre 2005

Autor: Albert Comerma Montells
Director: Atilà Herms i Berenguer

Realitzat a: Departament d'E.C.M. UB

PROJECTE FINAL DE CARRERA D'ENGINYERIA EN ELECTRÒNICA

Títol	Radiation Tolerant and controlable power supply design and implementation for the VFE of SPD at LHCb detector
Autor	Albert Comerma Montells
Director	Dr. Atilà Herms i Berenguer
Data	Setembre 2005
Descriptors	SPD, Reguladors lineals, FPGA, VHDL

Resum del treball:

En el present projecte, emmarcat en el disseny electrònic de circuits de control, es desenvolupa la font d'alimentació per als circuits de lectura de dades (VFE) del subdetector SPD en el detector LHCb que s'està construint al CERN. Per tant és una de les parts principals que formen aquest subdetector i s'ha d'assegurar la seva robustesa amb el màxim de funcionalitat implementada possible. Cal tenir en compte que tota la electrònica d'aquest subdetector treballarà sota un entorn en radiació i, per tant, s'han de prendre totes les mesures necessàries per assegurar-ne un correcte funcionament en aquest entorn.

El sistema de regulació de tensió es basa en reguladors lineals dissenyats de forma específica per a una empresa privada per tal d'assegurar la resistència a la radiació esmentada anteriorment. Pel que respecta el control d'aquesta font s'utilitzen FPGA's també tolerants a la radiació. La flexibilitat d'aquests dispositius ens permeten implementar tots els elements lògics de control, comunicació i lectura de dades necessaris per a l'aplicació.

El disseny del sistema es divideix bàsicament en les següents tasques; definició dels requeriments (que ens vindràn limitats en la seva majoria per les característiques de consum del VFE i de control en el subdetector), selecció dels components a utilitzar en la implementació electrònica, disseny dels esquemes necessaris, disseny del circuit impres, prototipatge del circuit i posta en marxa, codificació en un llenguatge de descripció de hardware (VHDL) de les funcions de control i finalment la comprovació del correcte funcionament del hardware. Degut a les necessitats de control d'aquesta aplicació també s'hauran de realitzar dissenys de circuits auxiliars i de software per tal de comprovar el correcte funcionament del prototip.

Aquest projecte representa un bon exemple del desenvolupament complet d'un disseny de hardware mixte digital/analògic basat en una FPGA en l'àmbit de la recerca.

PROJECTE FINAL DE CARRERA D'ENGINYERIA EN ELECTRÒNICA

Title	Radiation Tolerant and controlable power supply design and implementation for the VFE of SPD at LHCb detector
Author	Albert Comerma Montells
Directors	Dr. Atilà Herms i Berenguer
Date	Setembre 2005
Descriptors	SPD, Linear Regulators, FPGA, VHDL

Summary:

In the present project we present the developement of the electronic system of the low voltage power supply of the VFE board of the SPD subdetector in the LHCb detector that is being built at CERN. This is a fundamental part to assure the correct behaviour of the subdetector system. In this design we must take into account that all the electronics of the subdetector will be exposed to relevant levels of radiation and use the components most suituable to work in this environment.

The voltage regulator system is based in linear regulators designed by a comercial manufacturer specially for this purpose of working under radiation conditions. In the control part we will use FPGA's wick are also radiation tolerant. The flexibility of this devices will perit the implementation of all the logic elements of control, comunications and data readout necessary for this aplication.

The design of the whole system will follow this steps; defintiton of the specifications (basically determined by the power consumption of the VFE and the comunications system used at the subdetector), selection of the components wich will be used, schematics design, printed circuit board design, prototiping, coding of the FPGA funcionality in VHDL and finally checking of the complete design behaviour. In addition and because of the control nature of this design there would be necessary to design some auxiliar hardware and to code some software in order to check the correct behaviour of the prototype.

This project represents a good example of a complete hardware developement of a mixed signal system (analog and digital) in the research scope.

GLOSSARY

ALICE: A Large Ion Collider Experiment
ASIC: Application Specific Integrated Circuit
ATLAS: A Toroidal LHC AparatuS
CERN: *Centre Européen pour la Recherche Nucléaire*
CMS: Compact Muon Solenoid
ECAL: Electromagnetic Calorimeter
ECS: Experiment Control System
FE: Front End electronics
FIFO: First In First Out
FPGA: Field Programmable Gate Array
FSM: Finite State Machine
HCAL: Hadronic Calorimeter
HDL: Hardware Description Language
I2C: Inter Integrated Circuit comunications protocol
LHC: Large Hadron Collider
LHCb: Large Hadron Collider beauty experiment
LVDS: Low Voltage Differential Signaling
LVPS: Low Voltage Power Supply
MAPMT: Multi-Anode PhotoMultiplyer Tube
MIP: Minimum Ionizing Particle
NIEL: Non-Ionizing Energy Losses
PCB: Printed Circuit Board
PLL: Phase Locked Loop
PMT: PhotoMultiplyer Tube
PS: PreShower
SEE: Single Event Effects
SEL: Single Event Latchup
SEU: Single Event Upset
SPD: Scintillator Pad Detector
SPECS: Serial Protocol for the Experiment Control System
TID: Total Ionizing Dose
TOTEM: Total Cross Section, Elastic Scattering and Diffraction Dissociation
VFE: Very Front End electronics
VHDL: VHSIC (Very High Speed Integrated Circuit) HDL.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In the context of high energy physics experiments it's always necessary the design and improvement of instrumentation equipment with special power consumption requirements. This project is aimed to the development of the hardware which will supply power to the main electronics system, and perform tasks of monitoring current consumption and voltage as well as temperature. For these monitoring purposes there will be necessary to introduce some kind of programmable device, and the firmware will be developed as well in this project.

1.1 Objectives

The purpose of this project is to design the Low Voltage Power Supply for the Very Front End of the Scintillator Pad Detector. The required tasks include:

- Definition of the requirements and specifications of the hardware.
- Definition of the requirements and specifications of the firmware.
- Identification of the hardware components to be used.
- First prototype design.
- Test functionality of the hardware in the prototype.
- Firmware development (first simulations and then over the hardware prototype).
- Normal condition work tests over the working hardware prototype.
- Design of final hardware, and production.

The power supply will be integrated in the system designed by the LHCb group in Barcelona, so it must meet all the requirements specified by the rest of hardware. For this reason the hardware will be designed close to the rest of the electronics in this part of the experiment.

1.2 Document Organitzation

This manuscript is organized in chapters that cover the different steps of the work arround this system. These steps include:

- Introduction; first of all we explain the context in wich this project is developed. We show a brief description of the complete detector where this system must integrate and concretely a more detailed explanation about the function and how it works de SPD.
- System architecture; in this chapter there is a more detailed description about the SPD electronics and we sum up all the especifications that must meet our hardware in order to achieve all the funcionalities. Most of them are derived directly from the SPD VFE caracteristics, for this reason in this section there is a more complete explanation about this part.
- Hardware design; in this chapter there is a concrete explanation of the decisions taken in all the aspects of the design of the hardware, including the component selection and the final specifications wich will meet de power supply.
- Firmware; in this chapter we describe the basic structure of the vhdl code generated for the FPGA onboard, wich does all the interface and control functions. The basic information are the FSM's diagrams.
- Test Setup; in this section we describe other things that had to be developed in order to be ready to test the board functionality. This part has been nearly as time consuming as the rest of the design because of the needs of the design of auxiliary hardware and to code the simulation of the comunications interface.
- Performance Tests; in this chapter there is a short description of all the tests done so far with the prototypes.
- Cronogram prevision; in this section there is the schedule wich was decided before the start of the project and we explain at wich status is the project and what is left.
- Bibliogaphy and references; some of the material needed to understand parts or the whole of the things wich include this project.

1.3 The Large Hadron Collider (LHC)

CERN is placed between France and Switzerland, with facilities at both countries, the flagship of the complex will be the Large Hadron Collider (LHC) wich is scheduled to switch on in 2007. It is a particle accelerator formed by a succession of machines, small accelerators, wich increase the energy of the particles and inject de resulting beam of particles to the next machine, resulting into a very high energy beam. Once at a high energy particles collide on a target that may be other particles or materials, generating other particles with a very high energy wich would leave traces in the detectors of the accelerator. The study of these traces will give information about the subatomic particles (wich often decay into other particles) and the forces wich keep them together in the atoms nuclei. As we increase in the energy of a particle we decrease the wavelenght of it permiting the study of phenomena wich occurs in a smaller scale. At 1 Tev we could study phenomena in a scale of 10^{-19} meters (tree orders of magnitude smaler than atoms nuclei).LHC is expected that will collide beams of protons at energy of 14TeV.

It will become the world's largest particle accelerator. It uses the 27 km circumference tunnel created for the Large Electron Positron (LEP) collider. Five experiments (detectors) will be built to utilitze the LHC; ATLAS, CMS, LHCb, ALICE and TOTEM. The two first are large general purpose particle detectors, while the others are smaller and more specialized. In figure 1 can be seen the placement of the CERN facilities, accelerators and detectors.

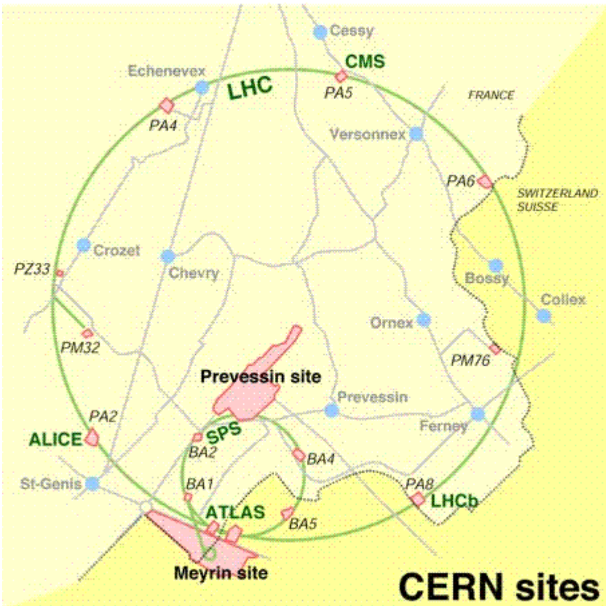


Figure 1: Cern Sites

1.3.1 Physics at LHC

Particle physics or high energy physics is based in the study of elemental particles and it's interactions and forces wich keep them together. Matter is formed by nuclei, wich are electrons, protons and neutrons, wich are formed by smaller particles called quarks. Matter is subject to forces wich are weak, strong, electromagnetic and gravitatory. The electromagnetic force acts between charged particles like electrons or protons, gravitatory force acts between particles with mass, the weak force is responsible of the unstability of the particles with bigger mass and the strong force is the one responsible to keep together the particles wich form the atom's nuclei. The elemental particles wich form all the matter are; quarks, leptons, photons, gluons and bosons W and Z, as shown in figure 2.

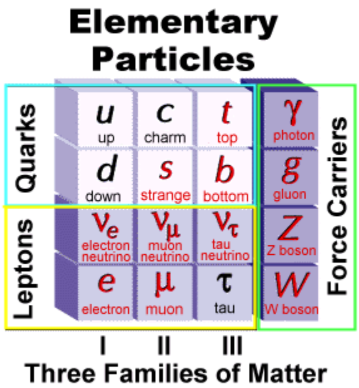


Figure 2: Elementary particles

With this classification we can distinguish between Hadrons and Leptons, and the first ones can be classified taking into account the number of quarks wich form the particles; barions, formed for three quarks, and mesons, with two quarks. Particles transform from one to another spontaneously for radioactive decays, generating in the process leptons and force carriers.

The beta decay is the process in wich a neutron (formet for two quarks up and two quarks down) is transformed into a proton (two quarks up and one quark down), generating one electron and one electronic neutrino. This process can be represented using Feynmann diagrams like figure 3.

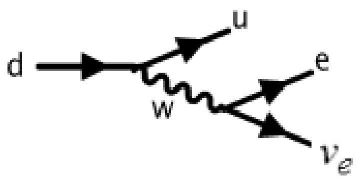


Figure 3 . Feynmann diagram. d=quark down, u=quark up, w=Wboson (wich decays to electron and neutrino), e=electron, V_e=electronic neutrino.

Disintegration of quarks always happens in the heavy quarks to the less heavy ones direction, otherwise there would be no energy conservation. Figure 4 shows the most frecuent decays.

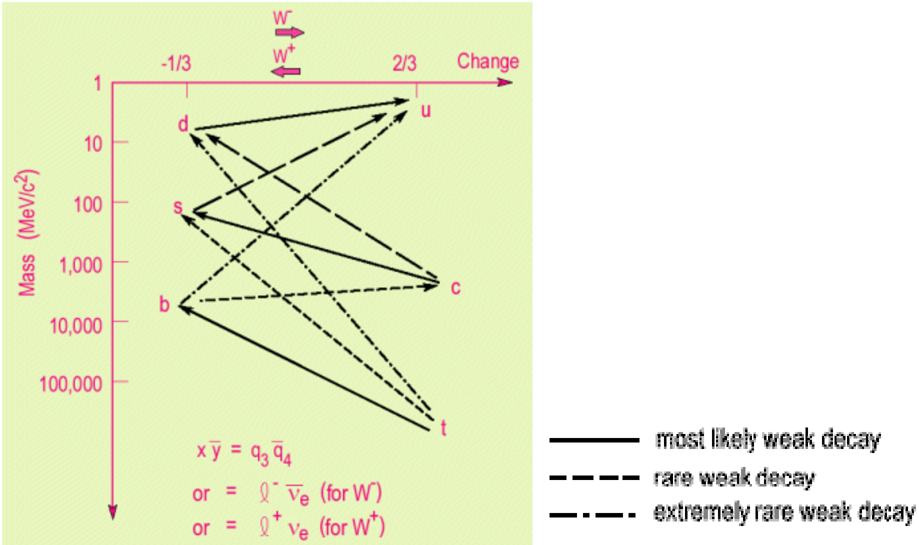


Figure 4. Most frecuently decays.

The particle behaviour is explained well with the Standard Model, proposed in 1968 for Weinberg, and wich has been completed with the physics evolution. This model descrives some particles as the Higgs, wich has not been proved to exist by the moment because of it's high energy (~115 GeV), but does not descrive particles like the graviton wich causes the gravitational force. The Standard Model is based in the laws of the Simmetry conservation, wich affect the interactions between particles. The simmetry conservation assure the conservation of the basic properties of mater, charge (C), parity (P) and time (T). The charge simmetry implies the transformation of one particle to it's antiparticle (with oposite charge). The parity simmetry implies the change in behaviour of the particle when we change one of the references. Finally, temporal simmetry permits that time goes in a determinate direction.

The particle accelerator at LHCb is designed in order to study the disintegration of one type of heavy particles, mesons B, wich will be produced by proton-proton collisions. In this decay there is a CP (charge-parity) violation wich is also descrived in the Standard Model, in a certain possibility. This probability comes from the so called "angles" in the CKM matrix [1]. The Standard Model gives some theoretical values of those angles wich will be tested experimentally within the LHCb experiments. The experimental verification of those values could explain why there is much more matter than antimatter in the universe, or why particles have different quantity of mass.

1.3.2 LHCb detector design

The LHCb detector is designed to make precise studies of CP asymmetries and of rare decays in the B-mesons systems in the LHC proton-proton collider at CERN. The detector can reconstruct a decay vertex with very good resolution and provide charged partice identification. The layout of the LHCb spectrometer is shown in figure 5. It comprises:

- A vertex detector whose main task is to provide information on the production and decay vertices of b-hadrons and the trajectories of the particels close to the interaction point. It is formed by 17 silicon discs.
- A tracking system. That provides reconstruction and precise momentum measurement of charged tracks and information for the triggers. It's formed by two elements the iner tracker and de outter tracker and has a resolution of 0.3% in momentum range from 1-200 GeV/c.
- Rich detectors (gas Rich1 and aerogel Rich2) to identify charged particles with a momentum over a threshold. In function of the range of the momentum particles can be classified. It also can be detected if it follows the ascendent flux (going through RICH1) or descendent (going through RICH2).
- An electromagnetic calorimeter system to provide identification of electrons and hadrons for trigger analysis with measurements of position and energy.
- A hadronic calorimeter wich will perimt the detection of photons.
- A muon detector wich identifies muons for the trigger information.

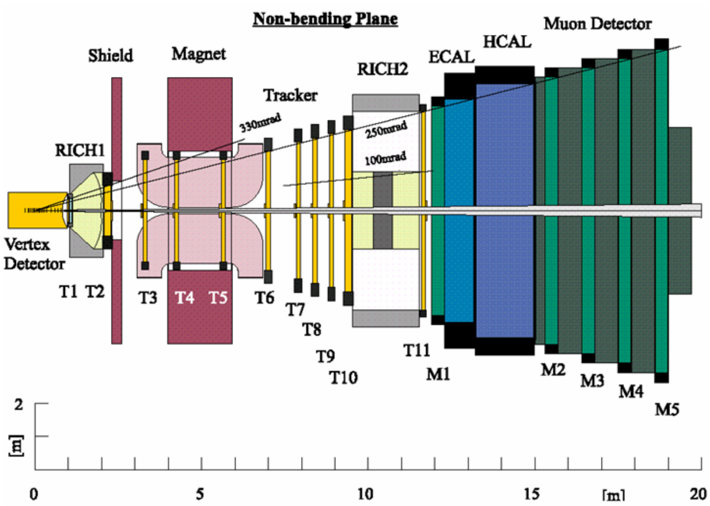


Figure 5. LHCb detector parts.

The research group formed by the *Universitat de Barcelona* and *Escola d'Enginyeria i Arquitectura La Salle* are in charge of the development and construction of the SPD (Scintillator Pad Detector), part of the trigger system of ECAL, wich will distinguish between electrons and photons.

1.3.3 The SPD (Scintillator Pad Detector)

The LHCb experiment [1] is designed to study B meson physics in the LHC proton-proton collider at CERN. LHCb calorimetry [2] is based on four elements: a hadronic calorimeter (HCAL), an electromagnetic calorimeter (ECAL), a Preshower detector (PS) and a Scintillator Pad Detector (SPD) (see figure 5 and 6). Such system is responsible to provide high energy hadron, electron and photon candidates to the level-0 trigger.

Specifically, the SPD is designed to distinguish between electrons and photons. This detector consists in a layer of plastic Scintillator, divided in about 6000 cells of different size, in order to obtain a better granularity near the beam [2].

Charged particles will produce ionization in the Scintillator while neutral particles won't. This ionization generates scintillating light pulses that are collected by a Wave Length Shifting (WLS) fibre coiled inside the Scintillator cell.

The light is transmitted through a clear fibre to the readout system. The signal of the Scintillator pads is processed in a Very Front End (VFE) unit. The VFE includes a photomultiplier (PMT) to convert the collected light into a charge pulse, the electronics to perform the discrimination between electron and photon signals, a beam bunch crossing clock receiver, a control unit and a LVDS (Low Voltage Differential Signalling) serialiser to send the information to the PS Front End cards.

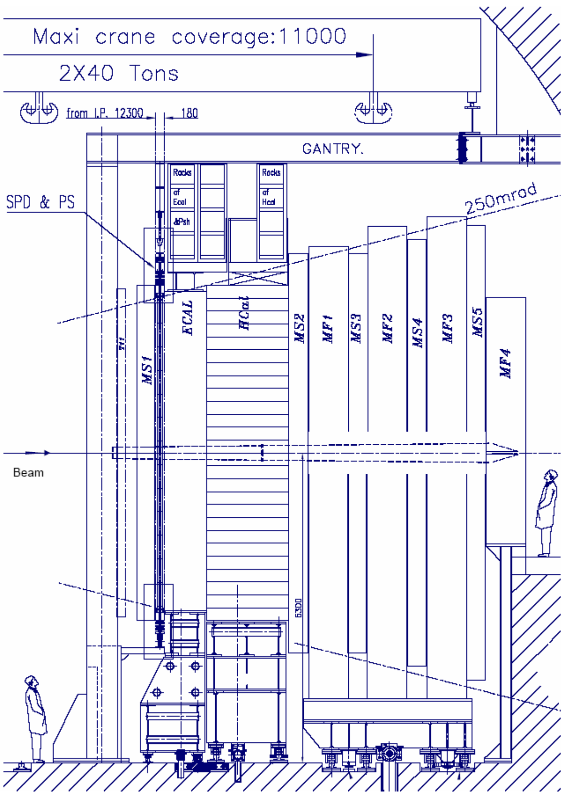


Figure 6. LHCb elements and mechanics.

Among several PMT solutions, multi-anode tube (MaPMT) candidates were considered in order to reduce the cost per channel. Baseline choice was made for a 12 stage, 64 channels PMT with Metal Channel Dynodes, bualkaly photocathode and pixel size of $2x2mm^2$. The parameterization of the MaPMT is being performed in parallel with the construction of the VFE electronics by the Barcelona group [3].

The SPD is the basis for the selection of events (trigger system) containing information to study B physics and enables the reconstruction of this interactions produced in the collider. The separation between electrons an photons performed by the SPD is called the level 0 of trigger.

Both the SPD and PS uses scintillator pads readout by wavelenth-shifting(WLS) fibres that are coupled to multi-anode photomultiplier tubes (MAPMT) via clear plastic fibres. The specific features of the SPD/PS detector are the rather hifh granularity in the inner part of the detector, and the use of 64 channel photomultiplier tubes with small pixel dimension of $2x2mm^2$. The choice of a MAPMT allows the design of a fast, multi-channel pad detector with a reduced cost per channel. The layout of the SPD/PS detector is shown in figure 7. It consists of a lead converter that is sandwiched between two almost identical planes of square scintillator pads of high granularity with a total of 11904 detection channels. The sensitive area of the detector is 7.6m in width and 6.2m in height. The lead converter in between the SPD and PS detector planes has a tickness of 12 mm and it can not be moved. The distance along the beam axis between the centres of the PS and the SPD scintillator planes is 56 mm. The arrangement of cells is a one to one projective correspondence with the ECAL segmentation. Therefore each PS and SPD plane is subdivided into inner, middle and outer sections with pad dimensions of approximately $4x4cm^2$, $6x6cm^2$ and $12x12cm^2$. The scintillator tickness is 15 mm. Figure 8 shows an individual scintillator pad with the WLS fibre. The basic plastic component is polystyrene to wich primary and secondary wave-length shifting dopants are added. The square structure of a pad is cut out from a 15 mm tick scintillator plate, and the scintillator surface is polished to reach the necessary optical quality. In order to maximise the light collection efficiency WLS fibres are coiled and placed into a ring groove that is milled in the body of the cell. The groove is filled with 3.5 loops of WLS fibre. The number of loops is chosen to achieve an overall optimisation of the light collection efficiency and the duration of the time response. Two additional grooves are milled to the scintillator to allow the WLS fibre to enter and exit the plate.

The clear fibres connecting the scintillator to the MAPMT have different lengths according to the pad position in the detector. This introduces different delays in the signal arrival to the readout system. To simplify the system, every MAPMT is connected to neighbouring cells and the corresponding clear fibers ar cut to the same length. In this manner, all signals arriving to a given MAPMT share the same delay.

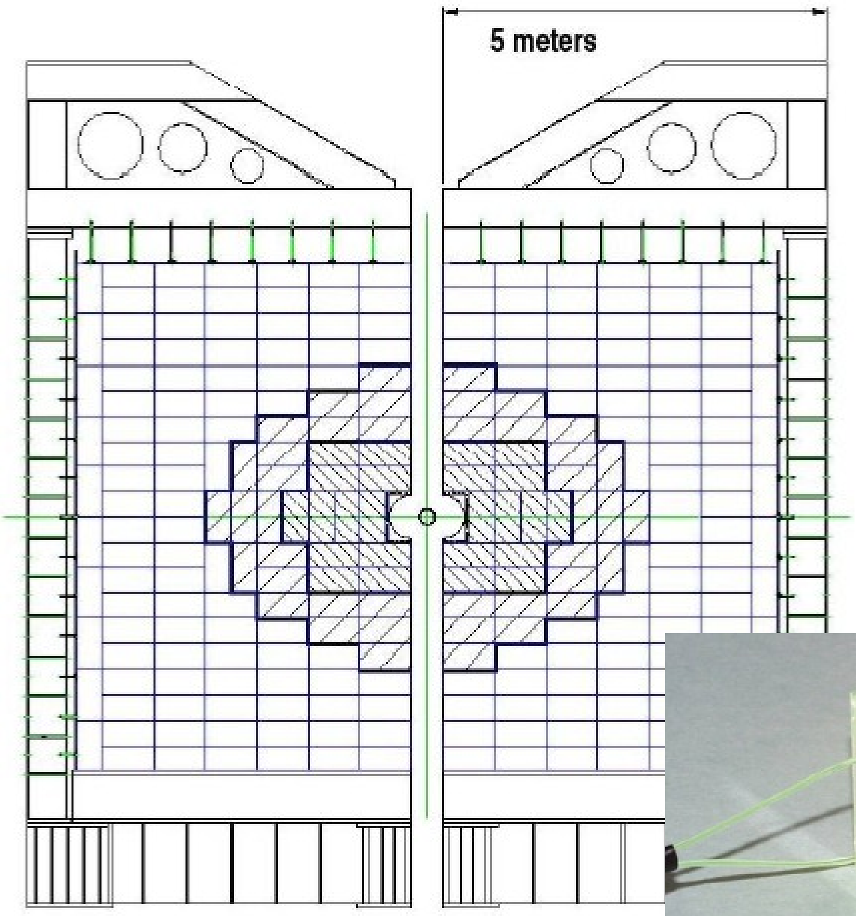


Figure 7. SPD/PS scintillators layout

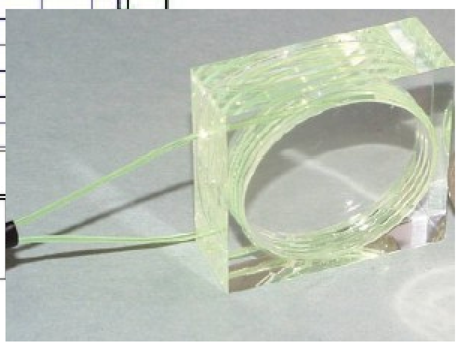


Figure 8. Individual scintillator pad

Charged particles crossing the scintillator produce ionisation, while neutrals, in first aproximation, do not. So the collection and detection of scintillation light seems to be suficient to assign a 1-bit logical signal to the 0-MIP (Minimum Ionizing Particle) / 1-MIP case, according to the requirement for the SPD [1][2].

However, some processes can cause photons to deposit indirectly energy in the scintillator so that they might be taken for a charged particle. From the physical point of view, photons deposit energy in the scintillator via the charged products of their interaction with the material of the detector.

These interactions may occur:

- On the material before the SPD in LHCb.
- Inside the SPD
- After the SPD, where charged particles generated at the photon-electromagnetic shower in the PS/ECAL trabal backwards, hitting the SPD (called backsplash).

Figure 9 shows the distribution of deposited energies in the SPD cell by photons and electrons obtained from Monte Carlo simulations. In the light of these plots, the assignment of the charged particle bit is done by comparing the deposited energy against a threshold. This threshold is fixed around 0.5 MIP. To allow for a wider range, the upper exploration energy is fixed to 5 MIPs. Finally the resolution of the SPD electronics must be better than 0.05 MIPs.

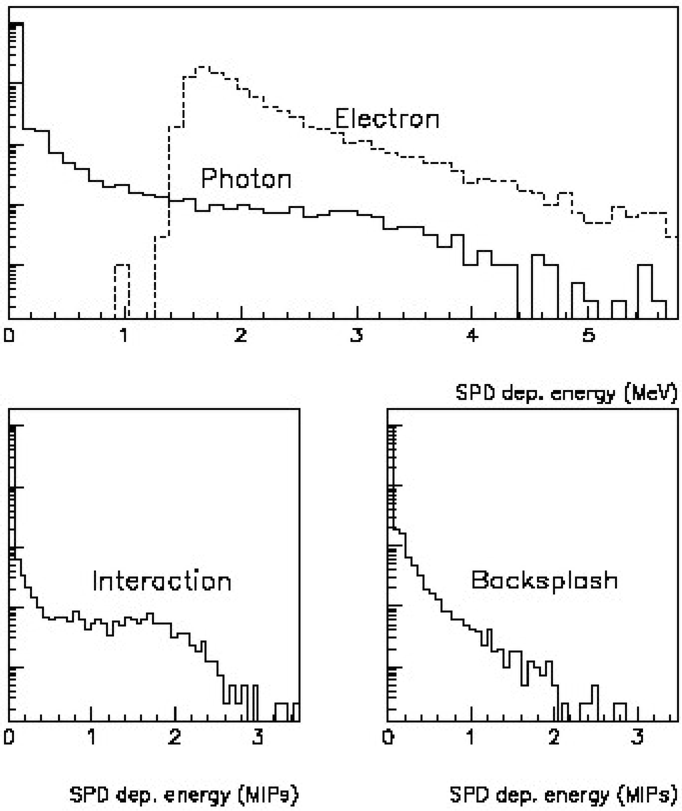


Figure 9. The upper plot shows the normalised Monte Carlo distributions of energies deposited in the SPD by 20 GeV electrons (dashed histogram) and by 14 to 18 GeV photons (full histogram). The lower plots show, for these photons, the energy deposited due to interactions with the SPD (left) and due to backsplash (right).

2. SYSTEM ARCHITECTURE

In this chapter there is a more detailed explanation about the SPD electronics in order to sum up all the caracteristics importants to take into account in the Power Supply Design.

2.1 SPD electronics

The SPD electronics system [5] is formed by tree elements; Control Board (CB), Very Front End (VFE) and Low Voltage Power Supply (LVPS). There is also another element; the High Voltage Power Supply that will feed the PMT's, this part is being developed by the Russian group. There's also an other part wich is in charge to make an active voltage division of the high voltage wich will feed the PMT's, this part is all a divisor board wich will be plugged between the VFE and the PMT's and has been designed by Clermont Ferrand's group. The third part, Low Voltage Power Supply is the purpose of this project.

2.1.1 Control Board

Squematically the control board is the electronics part that controls the performance of all the Very Front End Boards. It receives control parameters from a high-speed bus SPECS [4]. These control parameters are sent from the main control of the experiment, the ECS (Electronics Control System). The main purpose of the control board is to become a bridge between the SPECS bus to the I2C protocol used to control the VFE. It also brings the experiment clock to the VFE with a programable delay and does some calculus for trigger purposes. In figure 10 there is a block diagram of the Control Board electronics. The SPECS comunication is translated into I2C and other comunication protocols using a SPECS mezzanine board [4].

In order to improve the signal between the long cable connections from the Control Board to the VFE or the LVPS a differential transceiver is used wich will make the signal more immune to noise. This transceiver is the National Semiconductor's DS92LV010 which converts from a CMOS/TTL input to an LVDS output, with data direction control capabilities. The resulting I2C bus can be extended in terms of distance and velocity. For the 40MHz experimental clock a special circuitry with an amplifier is used in order to have a signal without jitter and with clear edges at the VFE.

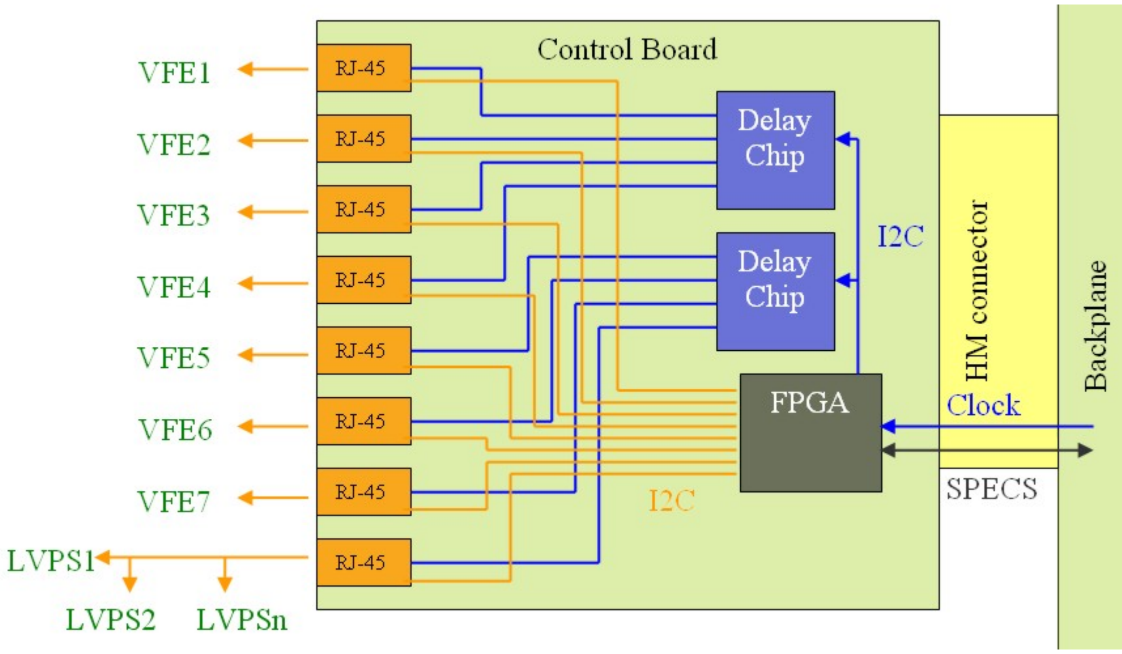


Figure 10. Control Board bloc diagram.

2.1.2 Very Front End

The Very Front End Electronics receive the Clock and control bus (differential I2C) from the Control Board. The base of the VFE is an ASIC (Application Specific Integrated Circuit) [6][7] designed by the Barcelona group wich processes the analogue signal from de MAPMT and outputs a digital signal per channel. Each ASIC has eight channels. In order to process a complete 64 channels MAPMT there are eight ASICS per VFE. The complete design of the VFE comprises the MAPMT with its voltage division board, the ASICS board and an output to a serialisers board which convert the 64 channels digital output to a 16 pair LVDS serialized signal. All of this electronics must work at the experimental clock rate (40MHz), so the multiplexed signal which comprises 7 data bits goes at 280Mbits. This conversion is done by the National Semiconductor's chip DS90CR215. A complete VFE would be like the one in figure 11, the electronics printed circuit boards are assembled in order to take the minimum space in the system. In the front of the VFE there is a MAPMT with a bundle of optic fibres positioned through a small piece of metal at the centre of every channel. The MAPMT is covered, once the fibres are subjected, with a piece of μmetal to minimize the transversal magnetic field. The data output and clock signals come from the Control Board and are connected in the opposite side of the VFE. Finally, the cables with the power supply are connected at the back of the VFE also.

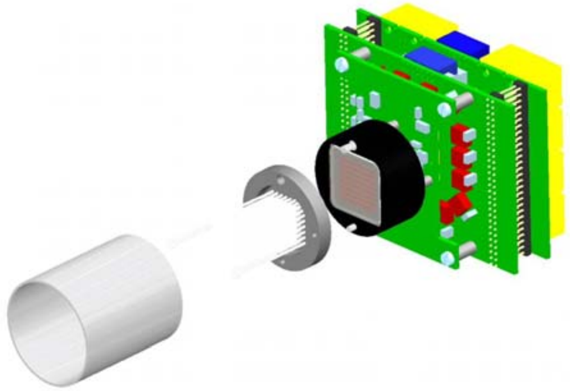


Figure 11. VFE Electronics assembly

2.2 Radiation Hardness

The calorimeter Front End electronics will be located in a region which is not protected from radiations (as seen in figures 5 and 6). Therefore all these electronics circuits must be qualified to stand some defined radiation levels. In general the radiation effects in electronic devices can be divided in two main categories; cumulative Effects and Single Event Effects (SEE).

Cumulative effects are due to the creation or activation of microscopic defects in the device, the effect of each individual defect not affecting significantly the device characteristics. Their steady accumulation causes measurable effects, which can ultimately lead to device failure. Cumulative effects are classified, according to their cause as Total Ionizing Dose (TID) and Non-Ionizing Energy Losses (NIEL).

Total Ionizing Dose effects, are due to the energy deposited in the electronics by radiation in the form of ionization. The unit for TID in the International System is the Gray (Gy). Ionization in the silicon dioxide, used in semiconductor devices for isolation purposes, generates electron-hole pairs that can be separated by a local electric field. The consequent macroscopic effect varies with the technology. In CMOS technologies the threshold voltage of transistors shifts, their mobility and transconductance decrease, their noise and matching performance degrades, and leakage currents appear. In bipolar technologies, transistor gain decreases and leakage currents appear.

Non-Ionizing Energy Losses (NIEL) in silicon cause atoms to be displaced from their normal lattice sites, seriously degrading the electrical characteristics of semiconductor devices. For displacement damage, it is common practice to express the radiation environment in terms of the particle fluency (particles/cm^2). The induced damage is a function of the particle nature and energy, and the Non-Ionizing Energy Loss is used as a parameter to correlate the effects observed in different radiation environments. Though this correlation is not free from uncertainties and fails in some cases, it is still commonly used to translate a complex radiation environment into a simpler mono-energetic equivalent, namely 1 MeV neutrons. CMOS transistors are practically unaffected up to particle fluency much higher than those expected at LHC. In bipolar technologies, displacement damage increases the bulk component of the transistor base current, leading to a decrease in gain. Other devices being sensitive to displacement damage are some types of light sources, photo detectors and optocouplers.

Single Event Effects (SEE) are due to the direct ionization of a single particle, able to deposit sufficient energy in the ionization process to disturb the operation of the device. In the LHC the charged hadrons and the neutrons of the primary particle environment do not directly deposit enough energy to generate a SEE. Nevertheless, they might induce a SEE through nuclear interaction in the semiconductor device or in its close proximity. The recoils from the interaction are indeed often capable of a sufficient energy deposition. Due to their statistical nature, it is possible to speak of SEEs only in terms of their probability to occur, which will depend on the device, and the flux and nature of particles. Therefore, the best one can do is to estimate their rate in the expected radiation environment. The family of SEE is very wide, but the main members are Permanent SEEs, Static SEEs and Transient SEEs. Permanent SEEs also known as “Hard errors” may be destructive. The best known is Single Event Latchup (SEL) which occurs in CMOS technologies, when the parasitic npnp thyristor is triggered by the ionizing energy deposition in a sensitive region of the circuit. This leads to an almost short-circuit current on the power lines, which can permanently damage the device. Sometimes, this condition can be local and the current limited (microlatchup), the effect can still be destructive. Some others effects like the Single Event Burnout (SEB) and the Single Event Gate Rupture (SEGR) are typical for power devices and can also lead to permanent damage. Stuck bits have been observed in SRAM and DRAM circuits irradiated with heavy ions [9]. The state of the memory point is permanently changed to a logic value, without the possibility to rewrite the correct value.

Static SEEs are not destructive, and happen whenever one or more bits of information stored by a logic circuit are overwritten by the charge collection following the ionization event. These effects are known as Single Event Upsets (SEU).

Transient SEEs happen when charge collection from an ionization event creates a spurious signal that can propagate in the circuit. This may happen in most technologies, and its effects vary significantly with the device, the amplitude of the initial current pulse, and the time of the event with respect to the circuit.

Radiation hardness of every electronic component in the VFE has been tested[8]. The components have been characterized for cumulative effects, Total Ionizing Dose and None Ionizing Energy Losses, and for single event effects, Single Event Upset and Single Event Latchup. The ASIC has been qualified in terms of expected total dose; the rate of SEU is acceptable while no SEL are foreseen. The best suited DAC has been selected, which shows no problems for the accumulated dose and SEL. However a low rate of SEU is to be expected, both soft and hard level, which will need dedicated additional control logic. Other tested components have been shown neither performance degradation nor SELs and can also be considered as qualified. In the case that any Latch Up may occur the Low Voltage Power Supply must monitor the current of every VFE and avoid any exceptional increase in current.

For the Low Voltage Power Supply components it's foreseen the use of the same operational amplifiers wich are tested under radiation and used in the VFE. Other components must be tested or documented to have been tested under similar conditions in order to be used. An other factor to take into account is the use of commonly used components in order to have the minimum number of diferent parts used, permiting changes and stocks more easily, for this reason a good candidate for the analog to digital converter used in the LVPS would be the converter used in the PS, the AD9203 wich has been tested and a few SEL's are expected, so it would need some kind of delatching circuitry. This converter can go up to 40MHz with an internal pipeline, but since it has a clock input it could be used at a slower sampling rate.

2.3 Cabling and grounding

Power supplies for sensitive analogue front-end electronics is a delicate point that requires significant attention in large scale systems. Power supplies must be used in a fashion compatible to the defined grounding scheme. Analogue front-end electronics is in many cases extremely sensitive to noise in the power supply because of the single ended nature of detector signals. It is in most cases required to have separate power supplies for analogue and digital front-end electronics to prevent noise from digital circuitry to disturb the analogue part. The cuestion of location of the power supply units and the related transfer of power on cables often have significant problems. Low frequency voltage drops on cables can to a large degree be compensated for by the use of remote sensing to compensate for cable losses. Remote sense can on the other hand also pose stability problems as the compensation loop must be stable under large load variations and with different cable characteristics. In special cases the remote sense circuit must be specially adapted to the final working conditions. High frequency components can only be handled with the use of local decoupling capacitors. In most cases a difficult choice must be made between using normal commercial power supplies in the counting room with long cables (~80m) to the front-end electronics in the detector, or having power supplies in the cavern close to the experiment but then having to deal with radiation and magnetic fields. Power supplies for the use in the cavern must be specially designed to handle the radiation effects and the magnetic fields. For the general LHC experiments only few companies are being considered to be capable of supplying power supplies that can handle the environments of the experimental caverns. In the power supplies side the recommendation of the colaboration is to use twisted, shielded and halogen free cables. An other factor in the detector grounding scheme is that all the mecanical structure will be attached to a good ground, permiting to have the electronics conected to that ground. This does not grant that it will be a ground good enough for our electronics, so it may not be taken as a solution for the grounding and it's recommended to have cabling for that purpose with a section big enough to deliver all the power needed in the electronics.

2.4 Technical characteristics and Specifications

In this section the concrete requirements of the system are exposed. All this requirements would be the specifications of the forthcoming prototype. The main purpose is to start the dimensioning of the power supply and its mechanical characteristics.

2.4.1 Placement

The mechanical system where the SPD will fit is already designed. Due to the asymmetry in the cells the result of the SPD are boxes with different numbers of VFE in each one. The idea is to fit the regulators board in the spare space of the VFE boxes. The distribution of the VFE cards is as shown in figure 12. This figure shows the half of the SPD boxes, the other half is symmetric with the centre. The boxes where would fit the power supplies are highlighted.

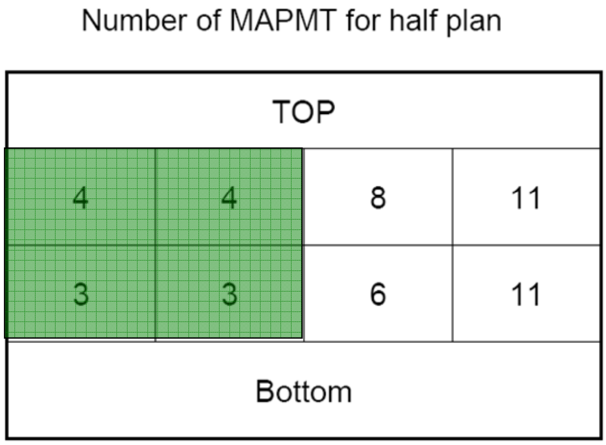


Figure 12. VFE distribution in SPD.

This distribution implies that in the worst case 27 VFE must be powered by two power supplies (one in each box). The cabling would be horizontal in every group of boxes. In conclusion every power supply must give enough power for 14VFE. This would mean an enormous board for fitting all the regulators, so the power supply could be divided in two identical boards which would feed 7 VFE. This division would permit an easier cooling system avoiding the hot spot resulting in a complete regulators board for all the VFE.

2.4.2 Size

The main part of the boxes would be occupied by the VFE boards, so we must take the spare space left in a side. Figure 13 shows the layout of the VFE in the boxes with less quantity of boards.

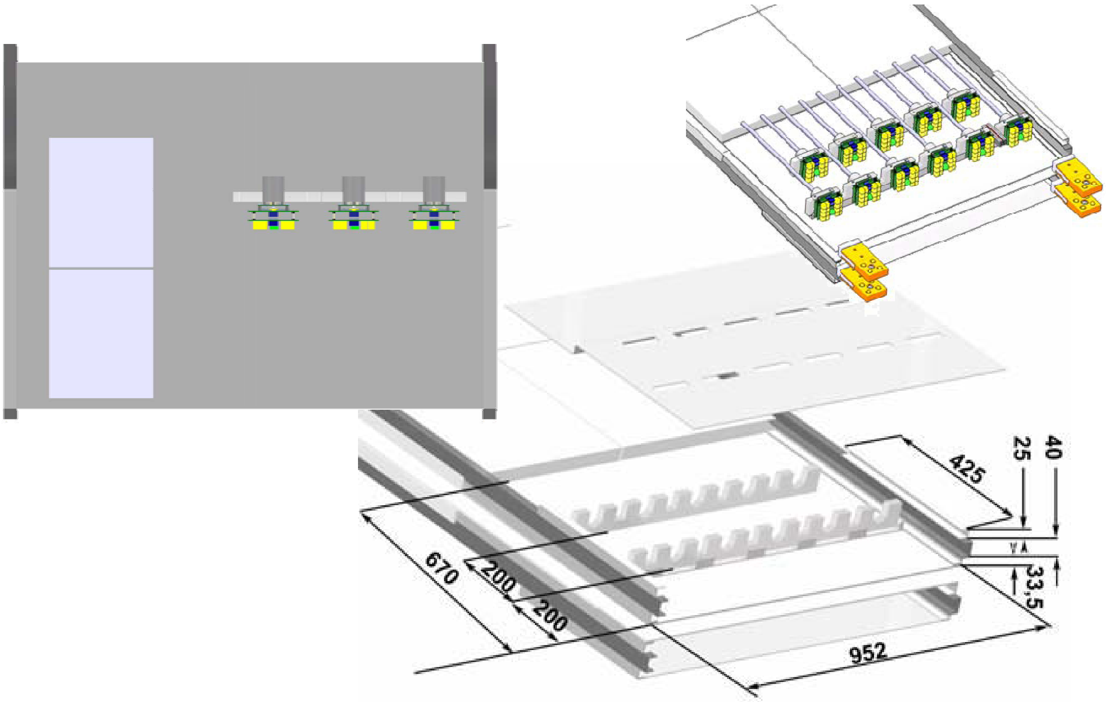


Figure 13. Power supplies position and maximum size

This layout results in a maximum size of $476x670mm^2$ with a maximum height of 98mm. The orientation of the boards could be vertical in order to provide a better cooling and an easier connection. See cooling section for more information.

2.4.3 Cooling

The cooling system proposed at first consisted in compressed air (cold) flowing from the bottom part of the electronics. The heated air would go out the box by special holes over the electronics, but finally the group decided to use a water cooling system, because it's more efficient and does not heat the cavern air.

The electronics must be designed in a way that makes easy the insertion of heatsinks wich would be connected to water pipes and go to the general cooling system.

2.4.4 VFE Power requirements

Next table summarizes the power requirements of every VFE board, which will be the basis of the output power necessary in the Power Supply. Note the capital letters next to the voltage rating, they mean analogue and digital power (A or D).

Voltage (V)	Current (mA)
+1’65A	1408
-1’65A	1280
+1’65D	200
-1’65D	200
+0’85D	50
+3’3A	256
3’3D	200
GND	200

Table 1: VFE power consumption

Basically the ±1’65A power consumption is from the ASIC’s, each of them consumes arround 180 mA, most of it static due to the diferential bipolar tecnology used. The ±1’65D feeds the FPGA I/O and the level shifters between the ASIC’s and the serializers. The +0’85 is the FPGA Core power (wich is 2’5V over -1’65V wich is it’s reference). The 3’3V analog is used in the reference DAC’s that control the ASIC’s and are buffered using operational amplifiers. Finally there is a small current between +3’3V digital and ground wich corresponds to the serializers consumption.

3. HARDWARE DESIGN

3.1 Proposed solution

In this section is explained the proposal of the hardware distribution and components to meet the system specifications. Due to all the constraints referring the previous sections, there have been a few decisions that have been done at an early step of the project; the most important one is the regulators to be used.

3.1.1 Hardware blocs

The bloc diagram of the full hardware is resumed in figure 14. There are three kinds of blocs; the first one would be the most important part of the system, the regulators which will feed the VFE from an input voltage. The second kind of bloc would be the analogue one, which converts currents measured through protection fuses (if fuses are well matched in resistance there is no need to add a shunt resistor), scaling it to fit de ADC specifications, and the reading of temperature sensors (basically NTC resistors connected in a Wheatstone bridge configuration). And finally the third part would be the digital control electronics, formed by the FPGA and all the communications system. It must control the output current and voltage levels and in case there is any channel with too much current or improper voltage it can switch off the corresponding regulator. All this functionality must be accessed from main control system ECS, so all control registers must be read/write by the local I2C through CB.

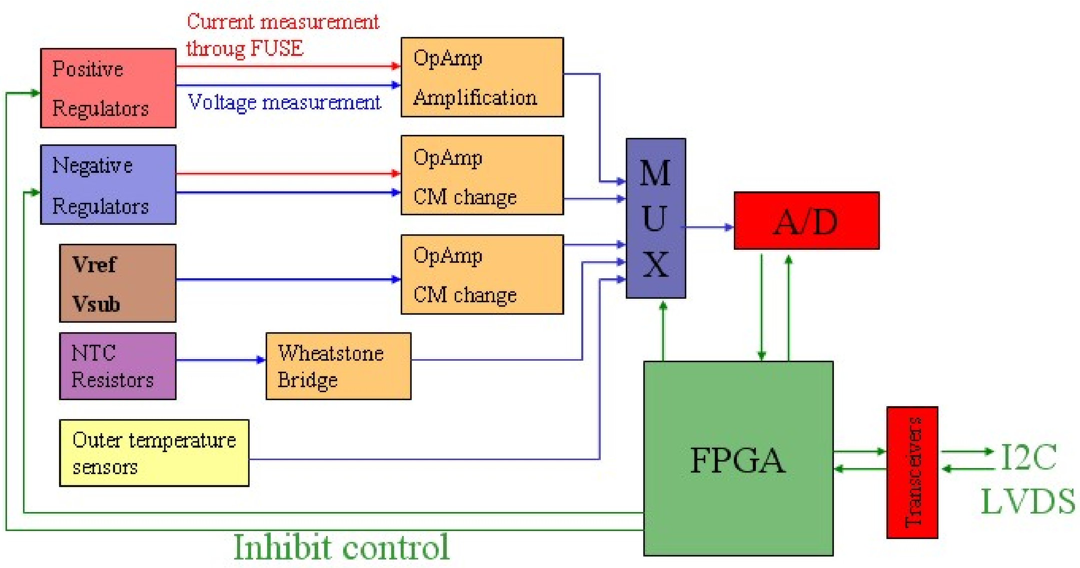


Figure 14. Hardware bloc diagram

3.1.2 Low Voltage Power Supplies boards location

The distribution of the boards would be te one seen in figure 15. This distribution is for half detector: 4 boxes top and 4 boxes bottom, the other half is symmetric to this one.

TOP			
1 Regulator	2 Regulator	1 Regulators	-
1 Regulator	2 Regulator	1 Regulators	-
BOTTOM			

Figure 15. Low Voltage Power Supplies location

In figure 16 can be seen a simulation with the mechanical size of all the boards that must fit the box with most ocupation of VFE and Regulators board. The size of the regulators board is the one of the first prototype.

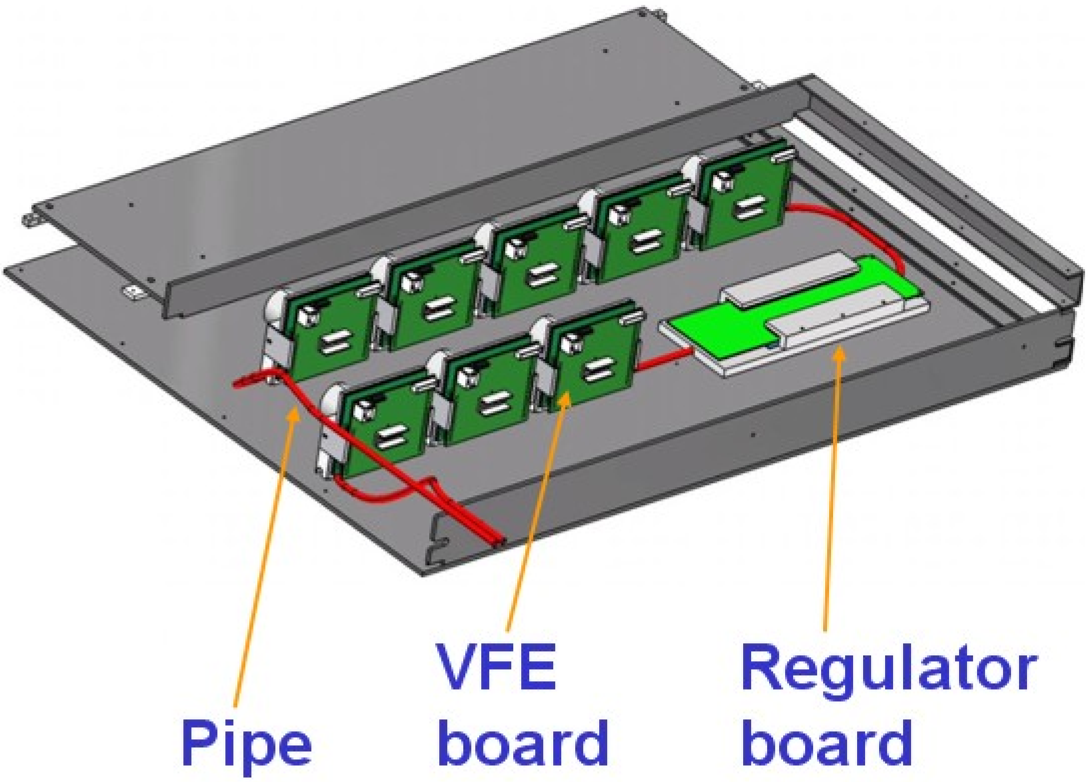


Figure 16. Mechanical view of a box with one regulators board and 8 VFE

3.1.3 Regulators

A special set of radiation tolerant linear regulators have been developed for the LHC experiments. This set consists in a pair of devices; one for positive voltages and one for negative, of Low Drop Out (LDO) regulators with inhibit input and adjustable output voltage. Those regulators also permit a remote voltage sensing terminal which can be used to avoid the voltage drop in the cabling. The option of remote sensing was discarded due to stability problems reported by the Clermont Ferrand group. Other functions of these regulators are over current protection, over temperature protection and output short circuit monitoring, signalled by CMOS output. Housed into SO-20 slug-up package with stand off zero, they are specifically intended for applications in rugged environments, such as Nuclear Physics, in which they have to withstand large amounts of radiation doses during operating life. Input voltage ranges from 3 to 12V in positive regulator L4913 [10] and from -3 to -12V in negative regulator L7913 [11]. These regulators are available in fixed voltage output, but due to the power requirements, the adjustable version is used in this project.

3.1.4 Regulators per board

Table 1 implies that each regulator at least must deliver 1408 mA. This way we could feed one board per regulator. In order to avoid excessive power consumption in each regulator (hot spots) this is the limit we will give at each regulator (in theory they could deliver 3A). Using a maximum of 1’5A per regulator we have less power dissipation in each one. To keep the number of regulators as low as possible there are VFE wich will share power coming from a common regulator in the digital part wich has less consumption. Following this limitations we result in the next table of regulators per board (which will feed 7 VFE). We decided to split the power supply in circuits wich only feed those 7 VFE so it’s easy for the cooling point of view and permits to use a smaller FPGA package. Those numbers are without the needed regulators onboard wich will feed the FPGA and the conditioning electronics wich will be (1 for +3’3V, 1 for +2’5V and 1 for -1’65V). The power consumption of those regulators is small due to the low current consumption of the FPGA core and the signal conditioning electronics, wich must feed just the small current of the ADC sample & hold.

Voltage (V)	Regulators
+1’65A	7
-1’65A	7
+1’65D	3
-1’65D	3
+0’85D	1
+3’3A	1
3’3D	1

Table 2: Regulators per board

3.1.5 Power consumption

According to regulators datasheet 1’5V would be necessary at the regulators input to achieve the desired voltage, leading to an important power consumption done in each regulator. To minimize this power consumption we can provide an input fixed voltage that would be exactly the regulators output voltage plus the necessary voltage drop for the desired current. This results in a main power supply that would deliver (without taking into account the voltage drop in long cabling); 4’8V, 3’15V, -3’15V. Table 3 summarizes the power dissipation in each power supply board.

Voltage (V)	Power(W)
+1’65	15’75
-1’65	15’75
+3’3	6’75
TOTAL	**38’25**

Table 3: Regulators power consumption

3.1.6 Grounding

As stated before grounding is a very important issue in this kind of Analogue and Digital mixed systems to avoid noise and to keep current consumption in the desired electronics and not flowing between power supplies for grounding problems. For this reason the basic scheme of grounding is to keep the VFE with a common ground and the regulators board also with a common ground. A good section of cabling would deliver ground to the regulator board where a unique ground plane would be used for analog and digital voltages, the same that in the VFE. All the electronic cards also would be screwed to the detector box that will be well grounded and the screws will conduct ground to the internal ground planes.

3.1.7 Cabling

Using the fixed voltages described in section 3.1.3 the resulting cabling is resumed in figure 17. Note the important current that must go through the cables. In order to avoid important voltage drops it will be important to increase the cables section.

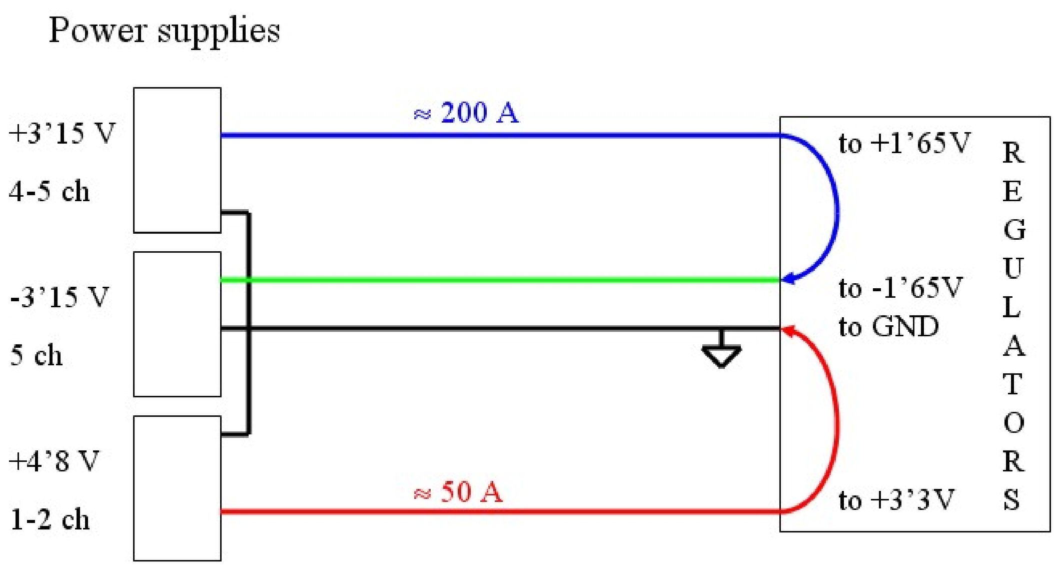


Figure 17. Cabling

Each regulator board would be powered through 2 screened twisted cables of 9 pairs (conductors have 1mm^2 section) CERN STORES SCEM 04.21.52.218.9. The outer diameter of each cable is 16,5 mm. The 2 cables will be connected to the regulator card through 1 single connector to avoid possible faults. The connector will be VLP-12V from JST, wich can stand 20 A per pin (the connectors were chosen to assure current, and in order to be low cost and easy to find in local stores). Each cable has one pair reserved for ground (the shield would also be connected to ground), this extra pair and the cable shield will be connected to ground using onboard fastons.

In table 4 there is a summary of the distribution of voltages among the 18 pairs, with the proposed pinout of the JST connector in table 5.

Supply [V]	Number of pairs	Number of conductors	Equivalent linear resistance [mℵ/m]	Max. Current [A]	Voltage drop [V] (20m: top)	Voltage drop [V] (30m: bottom)
+1.65 V	12	12	1,54	13,300	0,410	0,615
-1.65 V		12	1,54	13,300	0,410	0,615
Return for ±1.65V	1,5	3,00	6,17	0,007	0,001	0,001
3.3 V	3	3	6,17	2,100	0,259	0,389
Return for 3,3 V		3	6,17	2,100	0,259	0,389
GND	1,5	24,00	0,77			

Table 4: Voltage distribution in the power supplies cabling

+ 1.65 (3 c)	+ 1.65 (3 c)	+3.3 (3 C)	Ret1.65 (3C)	+ 1.65 (3 c)	+ 1.65 (3 c)
- 1.65 (3 c)	- 1.65 (3 c)	Ret3.3 (3 C)	GND (3C)	- 1.65 (3 c)	- 1.65 (3 c)

Table 5: Proposed pinout of the power input at regulators board

For the cabling between the regulators board and the VFE a smaller one has been chosed; screened twisted pair of 6 pairs of 0.5 mm^2 section, CERN STORES SCEM 04.21.52.124.4. The outer diameter of each cable is 12 mm. With this section the expected voltage drop in the cabling (wich would be 2m long maximum) is negligible (les than 100 mV).

For the data cabling it's expected the use of normal cat-5 or cat-5e data cables with RJ-45 connectors.

3.1.8 FPGA selection

As mentioned before, some kind of programmable device is needed in the Low Voltage Power Supply, in order to perform monitoring and control tasks to avoid permanent problems in the VFE hardware. Due to the radiation hardness requirement the decision of using FPGA's is the only valid. In this way and trying to be standard with the rest of the experiment, an ACTEL Accelerator FPGA was selected, but with the experience of other groups and specially with the new data of Flash FPGA's radiation tolerance [12], finally the family of ProASIC devices was selected, and in principle, the APA150 would fit in this design. This change was intended to achieve a better schedule in the development due to the flexibility that flash devices give to the developer. And it gives the possibility of hardware reconfiguration once the system is up due to its in circuit programming capabilities [13]. This device also can be upgraded up to the APA1000 with the same package, allowing to use the same PCB's if we can not fit de design in APA150.

3.2 Components list

In table 6 there is a summary of the components selected to be used in the Low Voltage Power Supply.

Type	Component	Comments	Related documents
Regulator	L4913		[10]
Regulator	L7913		[11]
FPGA	APA150		[12], datasheet
Op Amp	TLV2462		[8], datasheet
Delatcher	MAX891	Tested by Clermont Ferrand group	datasheet
Analog Switch	MAX4581	MAX4583 tested	[14], datasheet
Transistor	BFT93	Tested by Orsay group	datasheet
Transceiver	DS92LV010		datasheet
ADC	AD9203	Tested by Clermont Ferrand group	datasheet
Oscillator	EQXO-1050BMH		datasheet

Table 6: Components selected

Among the components in table 6 we must comment the case of AD9203 wich is expected to have a few SEL in the 10 years of the experiment, for this reason a delatcher has been included in it's power circuit (MAX891). A different case is the one of MAX4581 wich has not been tested under radiation but MAX4583 yes, wich is a component of the same family with the same transistor count and same technology, the only diference is how are connected inputs and outputs of it's pass-gates.

3.3 Blocs implementation

In this section there is a brief description of every bloc implementation that has been done in the design of the prototype. For a full view of the schematics and PCB layout see **APPENDIX 1: LVPS SCHEMATICS AND PCB**.

3.3.1 Positive Regulators

The basic schematic of the positive regulator is the one in figure 18. It follows the usual protection scheme; reverse diode at the output, filter capacitors at input and output, and fuse in series. The fuse is a protection in order to avoid possible fire causes, because of the change in section of the conductors could be possible to catch a fire for excessive current in one of the VFE power cabling. This fuse has also been used for monitoring purposes.

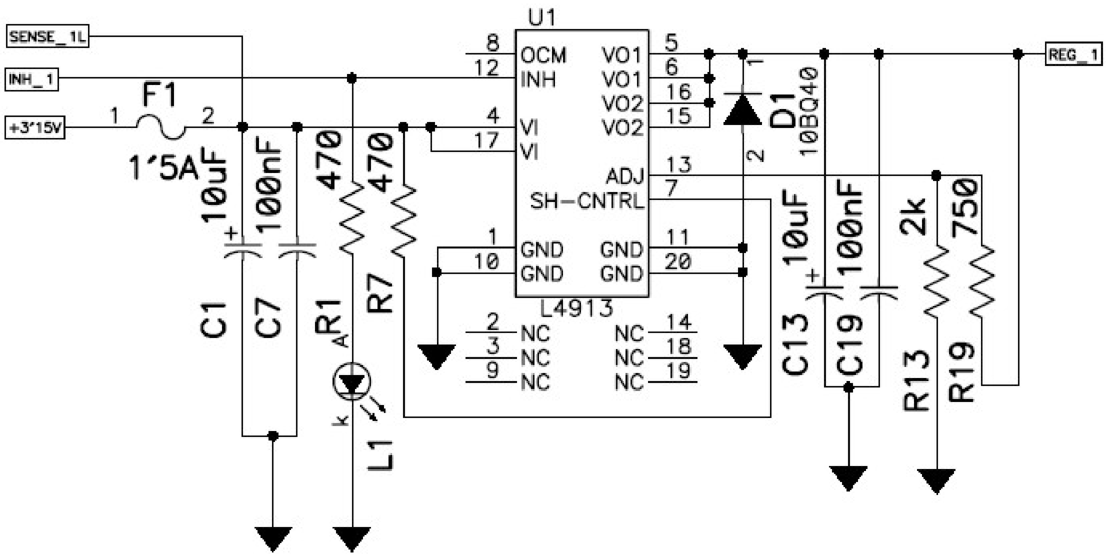


Figure 18. Positive regulators

In the prototipe there has been aded a led to see the status of the regulator inhibit pin and a current limiting resistor (R7), wich may be removed in the final version. The voltage output is controlled with the ratio between R13 and R19 as follows;

$$V_O = 1.225 * \left(1 + \frac{R19}{R13}\right)$$

Figure 19. Output voltage calculus

If we fix R13 below 2k5 (2k) wich is recommended by the datasheet we can calculate R19 easily and would result in about a 700 Ohm resistor. In order to permit a litlle of margin with the voltage drop in the cabling we choose a 750 Ohm resistor wich would give about 1’69 V output.

All the positive regulators are between a power input and ground except the ones wich may deliver +0’85 V digital. In principle the minimum voltage that can give this adjustable regulator is it’s reference voltage (1’225V) so in order to decrease this we use as a reference the -1’65V digital for those regulators.

3.3.2 Negative Regulators

The schematics of the negative regulators are quite similar to the ones in the positive, just changing polarity and including digital interface power supply. In this case we have the fuse at the output of the regulator, in order not to have problems in the common mode of the sensing circuitry. In this schematics we also have the limiting resistor and the led diode to indicate the status of the regulators, all of them may be removed or not soldered in the final version.

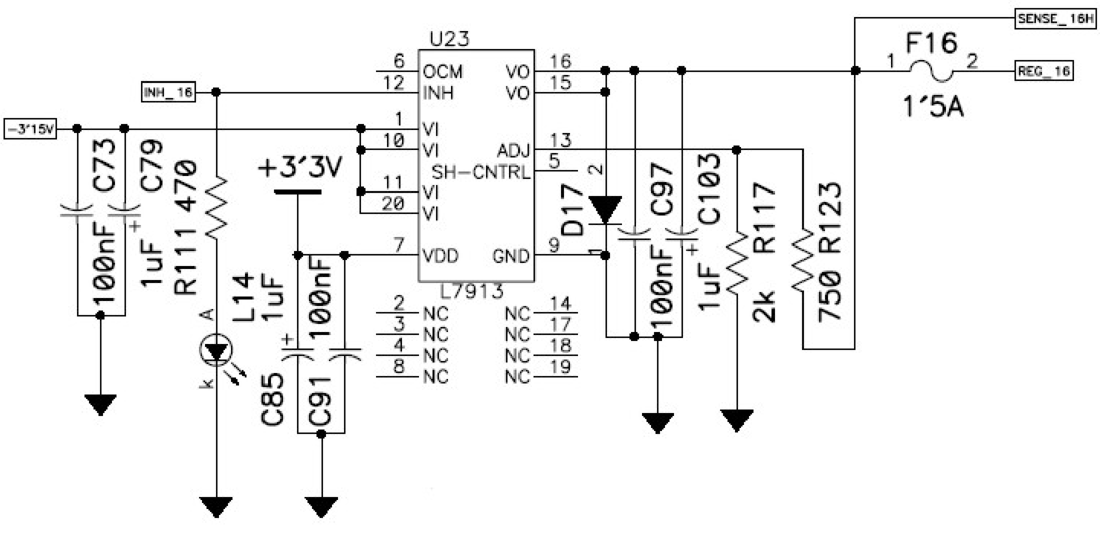


Figure 20. Negative regulators

3.3.3 Current sensing

We have two diferent cases in current sensing, one for the positive regulators and one for the negative regulators. The basis of the circuitry is the same; differential amplifiers with a power supply high enough in order not to have common mode voltage problems. The difference is where are connected the amplifier inputs and the power supply of the amplifier as can be seen in figures 21 and 22. In this schematic the gain is set to 20 because the estimated resistance of the fuses is set to 100 mOhms. As said before the fuses are located after or before the regulator in order to have the adequate common mode for the amplifiers.

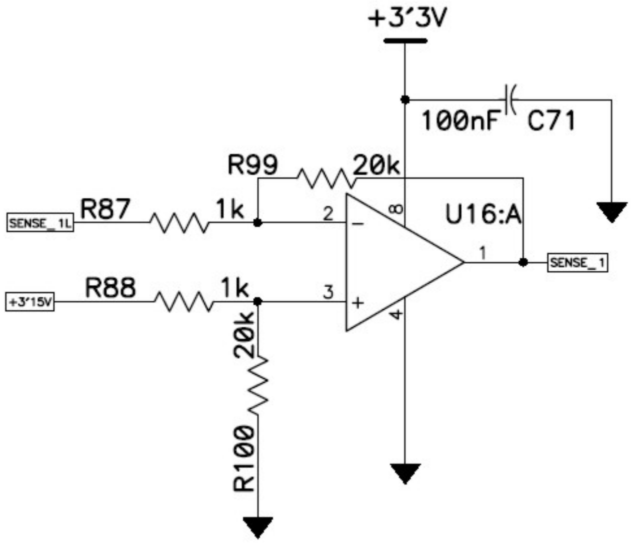


Figure 21. Positive regulators current sensing

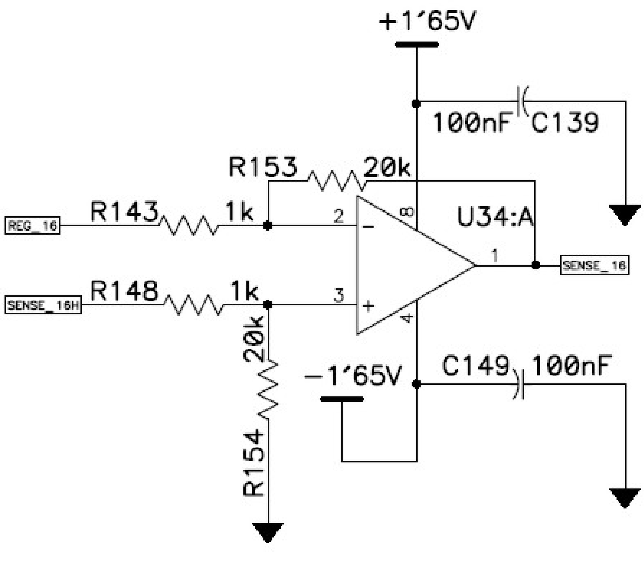


Figure 22. Negative regulators current sensing

3.3.4 FPGA

The FPGA has it's power supply in +2'5V at the FPGA Core and +3'3V at the pin I/O. Basically it has an oscilator input wich provides a 40MHz clock independent from the experimental clock, so it can run without the need of the rest of the experiment. It has an input of the I2C address to be used wich will be selected with an onboard mini switch. Finally has two groups of Outputs one big group wich controls all the inhibit pins of the regulators, and the other one wich is used to control the ADC and the analog switches wich select an analog channel to perform the readout. The FPGA delivers the clock to the ADC so we can control the sampling rate. There's also a control for the delatcher circuitry of the ADC. For debuging purposes there is a reset pulser with a typical RC circuit and a led output.

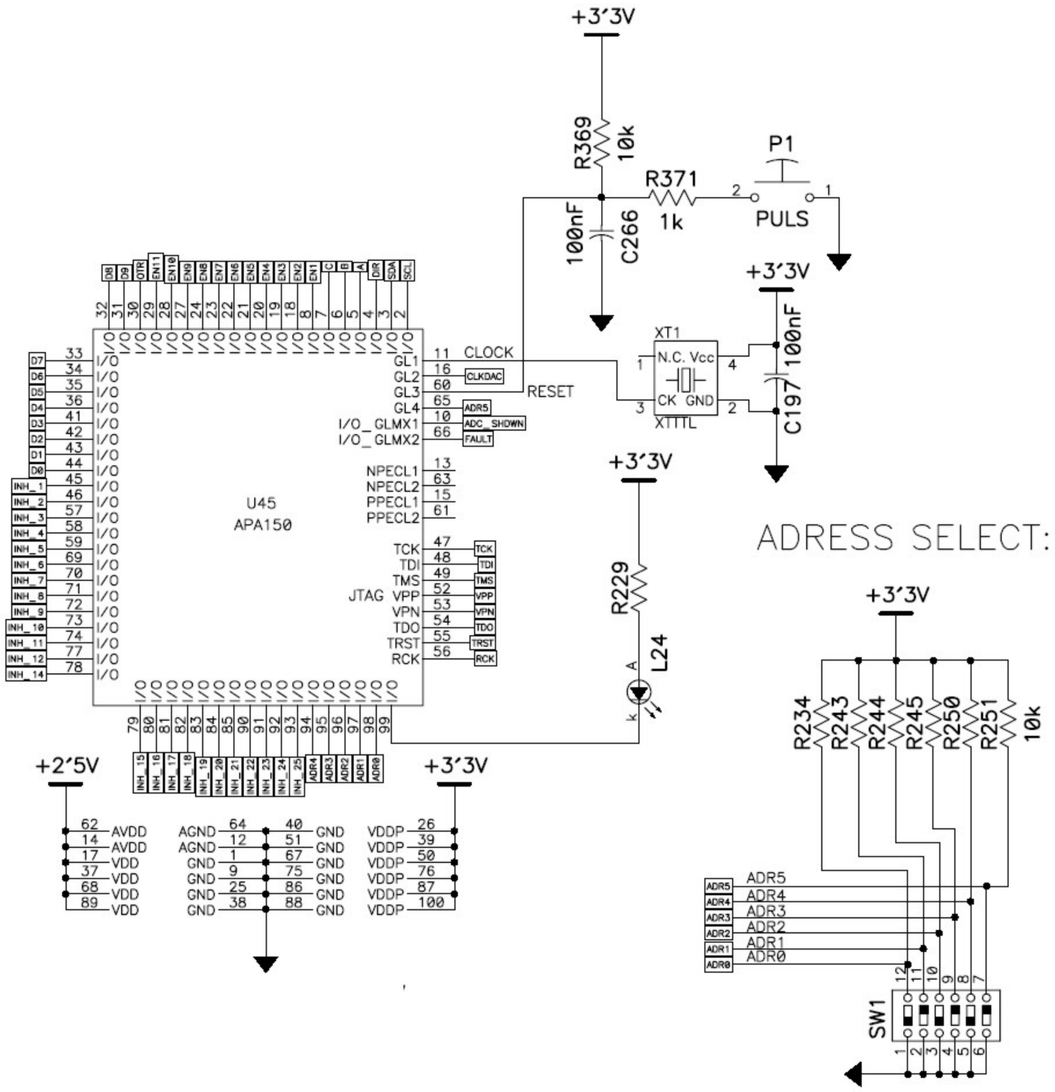


Figure 23. FPGA

The other important connection of the FPGA is the JTAG interface wich permits In Circuit Programming (for more information see [13]). The idea to minimize the pins used in the FPGA for the multiplexors control is to share the tree inputs selection and act over enable pins. The last part where connects the FPGA is the I2C link detailed in next section.

3.3.5 I2C link

The hardware link of the I2C bus is already specified by the Control Board. The use of the DS92LV010 means a differential pair of communication with a fixed direction SCL signal and a bidirectional SDA signal controlled from the FPGA. The basic scheme of these two lines is described in figure 24, 25 and 26. In figure 24 can bee seen the blocs diagram of the transceiver wich has one emmiter and one transmitter connected to the same transmission line, one of the two active.

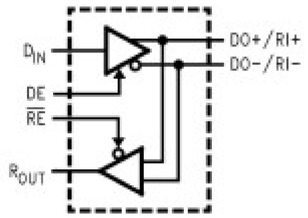


Figure 24. DS92LV010 bloc diagram

In the bidierctional SDA line the FPGA has control of DE and !RE wich are tyed togheter, while in the unidirectional configuration of figure 26, the direction is fixed by hardware.

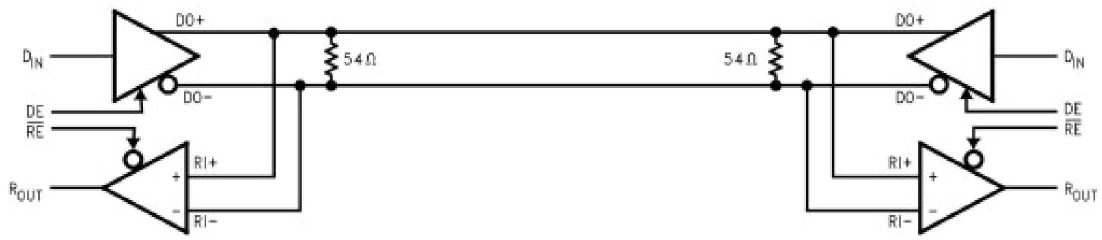


Figure 25. SDA bidirectional link

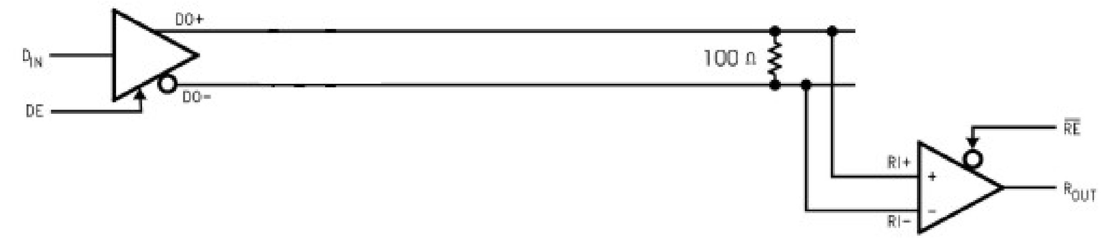


Figure 26. SCL unidirectional link

3.3.6 ADC

The ADC is connected in order to have an internal reference of 2V used for the conversion, so we can work with 0-2V single ended inputs. The input is connected directly to the analog multiplexors output. The only particuliarity of this circuit is the use of the delatcher to limit the current drawn by the ADC and report to the FPGA if it exceeds a maximum, then the FPGA can switch on again the delatcher with a certain delay.

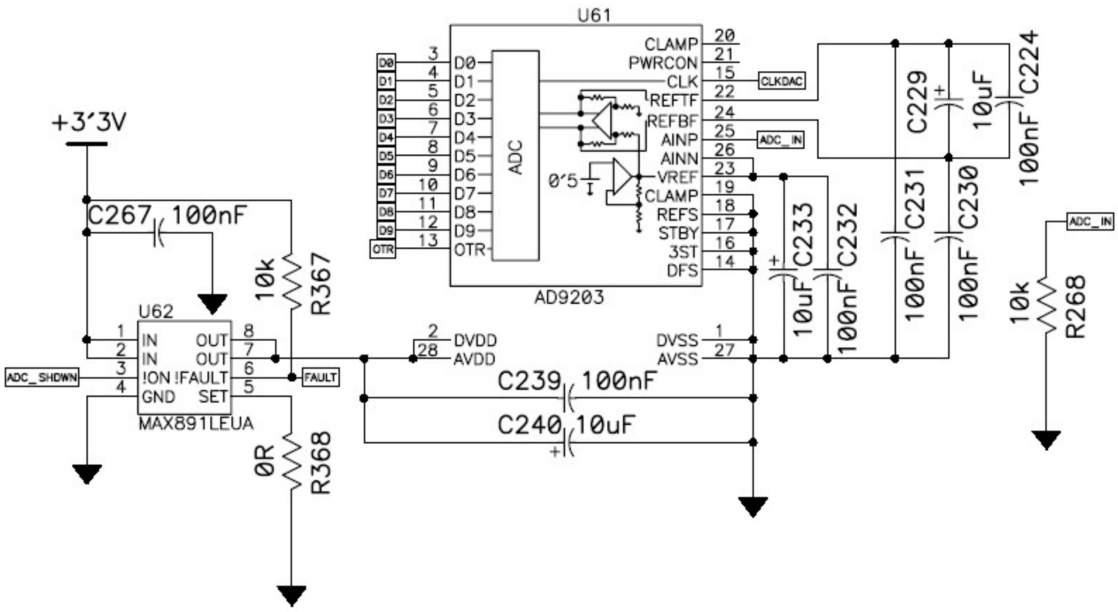


Figure 27. ADC and delatcher

3.3.7 Analog switches

The analog multiplexors/switches permit the use of only one ADC in the system using less FPGA pins. To continue with this low I/O usage the multiplexors share the three selection inputs and are controled by the enable pin to activate just one of them. Initially the outputs of the switches where connected togeter and to the ADC input, but due to diferent caracteristics in the switches we had current flowing from one to another and that was solved using another switch, and finally to improve the analog signal a voltage follower was introduced between switches output and ADC so that switches don't deliver current. In figure 28 there is the first aproach to the conection of those analog switches (just 2 in the figure).

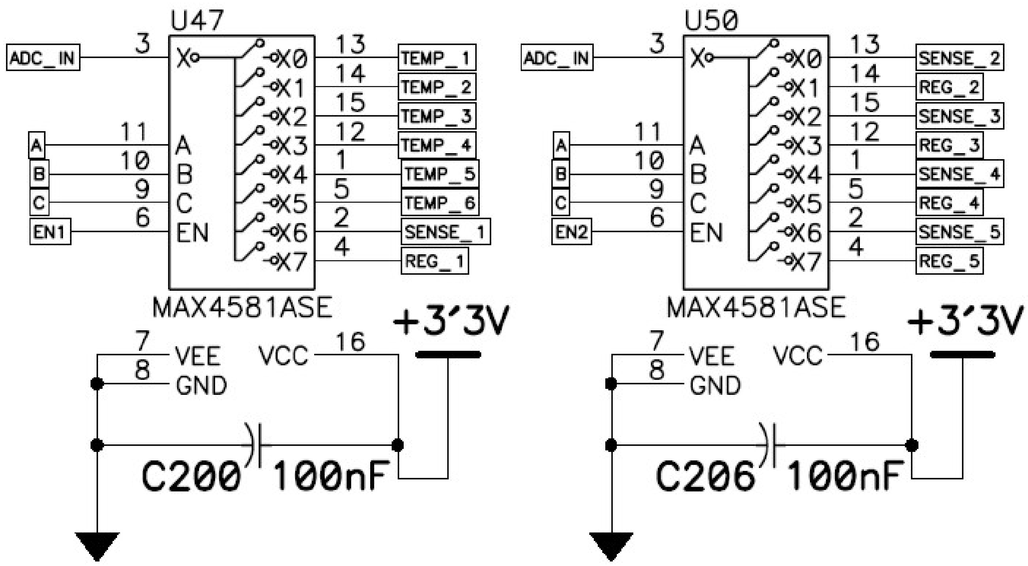


Figure 28. Analog multiplexors/switches

3.3.8 Temperature sensors

The first aproach of the temperature sensors was to use NTC resistors in a tipicall Weatstone Bridge configuration with a diferential amplifier at the output. This has been reported to work perfectly in our aplication, but since PS will use a current sensor (TMP17) wich is radiation tolerant and has more stability and better linearity this circuitry would be replaced in the final version.

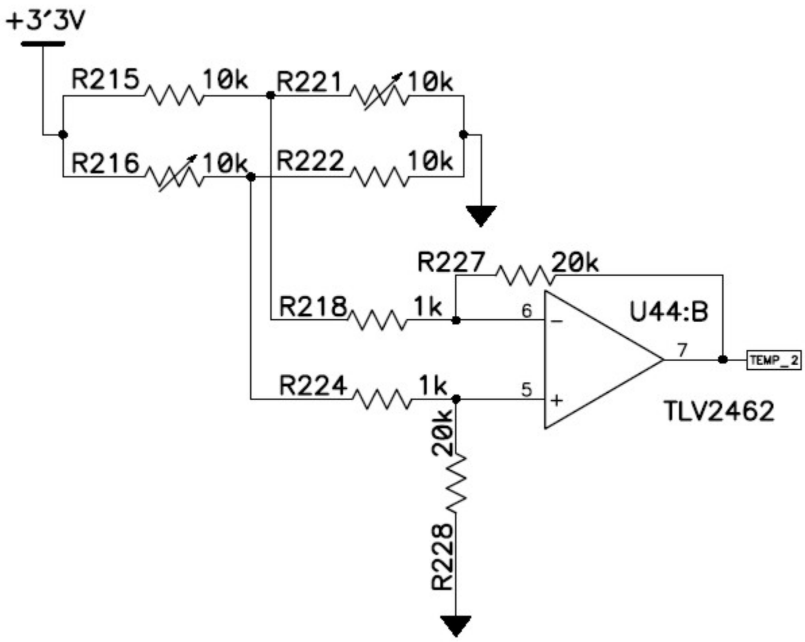


Figure 29. NTC temperature sensors circuit

4. FIRMWARE

4.1.1 General description and design tools

The FPGA firmware has been developed using the Actel tools (Sinplify, Designer,…) that form the Libero IDE. They use a free licence for devices upto 300k gates. The programming language choosen was VHDL. Most of the parts of the code are FSM's but there are also a few combinational blocs (see figure 30). For a complete listing of the vhdl code see **APPENDIX 2: FIRMWARE**. Currently the firmware version is 0.02, the first version (0.01) was a minimal one just to check I2C bloc and a few other things, nearly all from a combinational point of view.

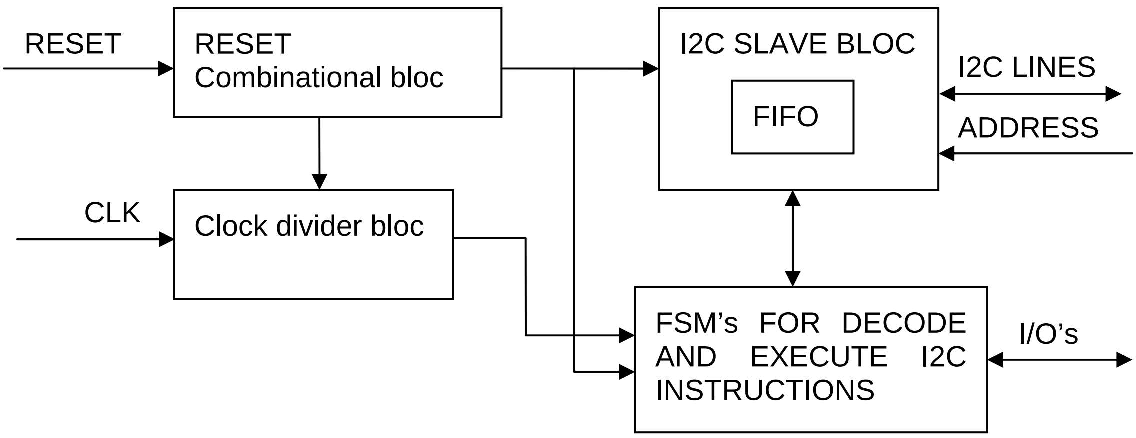


Figure 30. Firmware blocs

The two blocs in the left are simply combinational logic. The RESET bloc has a flip-flop in order to assure a sincronous clock to the rest of the system. The Clock divider bloc is a counter with one of the bits wich goes to the output, resulting in a clock division. Finali the I2C and FSM's blocs has all the control logic for the device functionality. The I2C bloc is an independent bloc with it's FIFO and control signals wich are controlled by the FSM's bloc wich is the one that does the decode and execution of the I2C commands received.

4.1.2 I2C bloc interface

The I2C bloc is the same as the used in the VFE FPGA, it was developed by Sebastià Tortella at La Salle. For this reason it's viewed as a closed box with just the in/out signals;

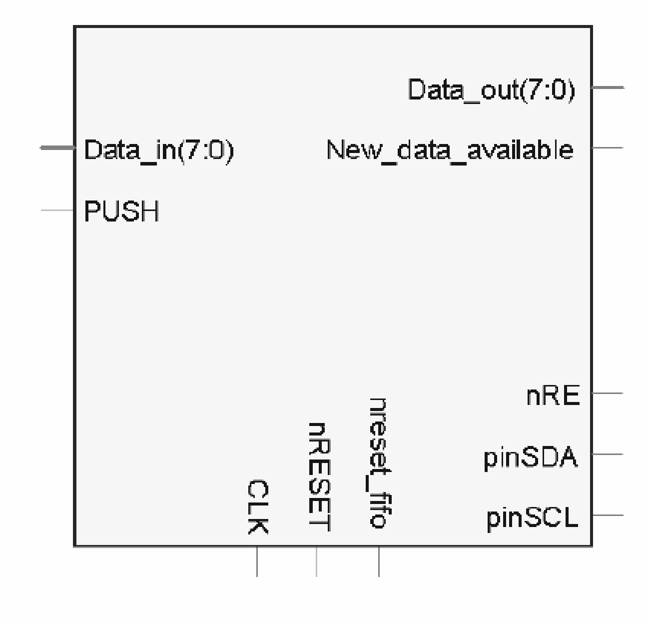


Figure 31. I2C entity

The control signals functionalities are;

- DATA_IN(7:0): Input of data byte to be writen at I2C bus when asked.
- PUSH: When active, the data of DATA_IN is stored in the output FIFO.
- DATA_OUT(7:0): Output of last data byte received by the bus.
- NEW_DATA_AVAILABLE: Active when a new byte is read from the bus.
- nRE, SDA, SCL: Connections direct to the transceivers.
- CLK: Clock input (40MHz).
- nRESET: I2C reset active by "0".
- nReset_fifo: I2C fifo reset, active by "0".

This bloc is always active and the rest of FSM's has the function to process the data input and supply the data output to this module.

4.1.3 Main FSM

When any data is received from the bus this State Machine becomes active. It has three states defined, as seen in figure 32; IDLE, DECODE and EXECUTE. When a command is received from the bus this state machine switches to DECODE state, decodes the instruction and jumps to EXECUTE waiting for the end command flag to be set by other FSM's. Any process that must be done during the EXECUTE state is performed by other FSM's. In the IDLE state there is a reset of all the signals involved in the state transition to assure a correct state flow.

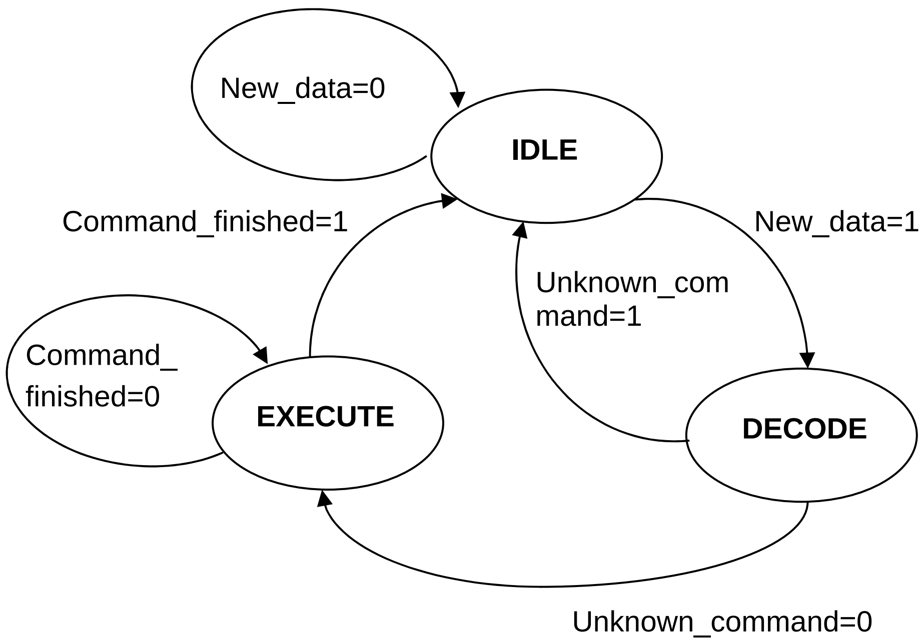


Figure 32. Main FSM

4.1.4 Send data to I2C state machine

When any data must be sent to the I2C state machine, there is a special FSM that handles all the signals necessary to input data in the FIFO. This has been done in order to be impossible to have to FSM's that want to send data at the same time and to avoid the repetition of the same code in every state machine that must send data. In this case we have the states IDLE, LOAD and SEND (see figure 33). In the first one the FSM is waiting for a control signal to become active, once active it will proceed to the LOAD state where the fsm loads the data that must be send to the comunications bus, and finally the SEND state asserts PUSH signal to introduce the data inside the FIFO.

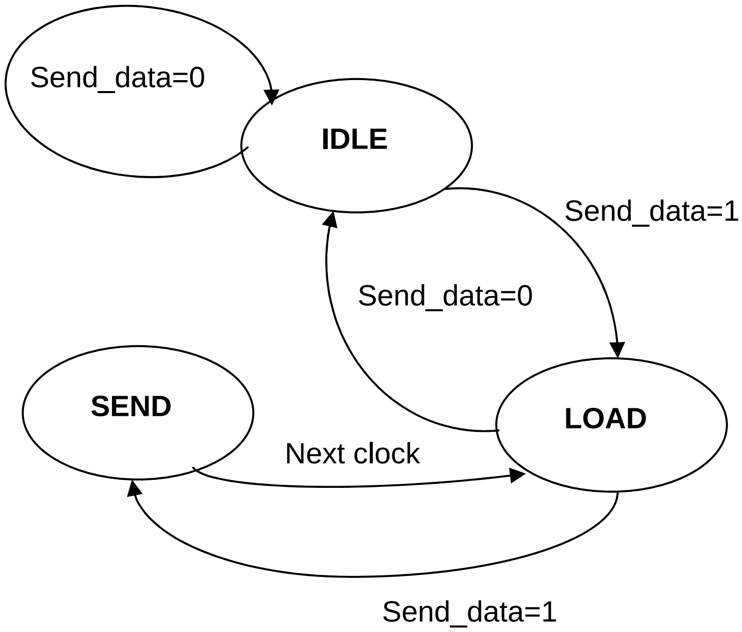


Figure 33. Send data to I2C bloc FSM

4.1.5 ID command state machine

When the DECODE execution of the main FSM decodes the command corresponding to asking for ID string this state machine becomes active. Similar to the other ones this FSM has the states of IDLE, and SEND. In the first the FSM waits for a signal to become active and in the second state the FSM brings the data to the output FIFO.

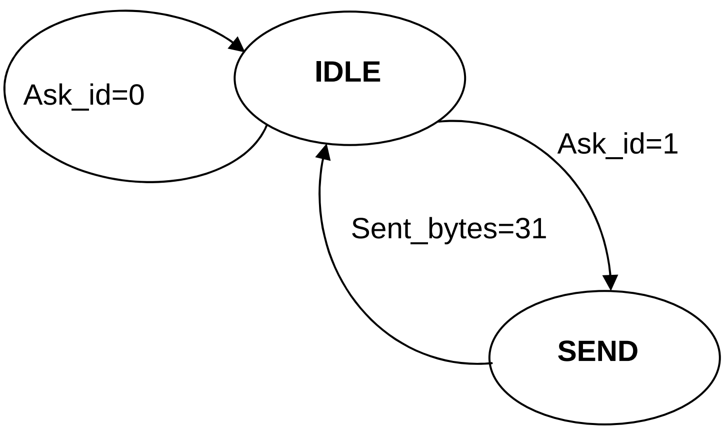


Figure 34. ID command FSM

4.1.6 Read analog channel state machine

This is perhaps the most complex FSM in the design because when the corresponding command has been received the system must wait for the channel to be read wich is an also byte from I2C bus. Once received all the information the FSM must select the analog channel, wait for the ADC to finish conversion, and insert the resulting 10 bits data to the I2C FIFO, divided in two bytes. The diagram of the FSM can be seen in figure 35.

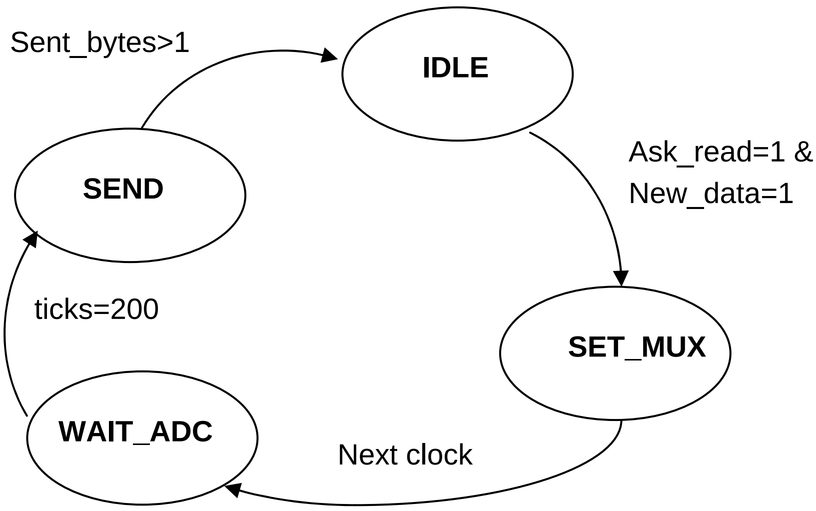


Figure 35. Read analog channel FSM

5. TESTS SETUP

In this section we describe the test setup used in order to test all the functionality of the regulators board by the moment. Auxiliary boars have been designed for two main purposes, the first one, to interface a PC with the Regulators Board, simulating the I2C protocol that would give the Control Board, and the second one to simulate the VFE load to the regulators.

5.1.1 Comunications board

The comunications board is based in the DS92LV010 transceiver. It uses the standard printer port of a PC connected directly to the transceiver. Data lines of the paralel port are used for powering the transceivers, signal pin SDA, signal pin SCL and control of direction of SDA, in the same way as used in the FPGA side. All the protocol is build by "bit banging" the data lines of the paralel port. To avoid common mode problems the ground is connected between boards through the cat-5e shielded cable used for the comunication. For the complete schematic see **APPENDIX 4: AUXILIARY BOARDS SCHEMATICS AND PCB's**. The PCB is a small two sided board in order to minimize costs.

5.1.2 Load board

The Load board is a simply resistive board with the values calculated to have a power consumption similar to the VFE. For a detailed schematic and PCB see **APPENDIX 4: AUXILIARY BOARDS SCHEMATICS AND PCB's.** In table 7 there is a description of the current drawn by each load board.

Voltage sides	Resistor load	Current expected
±1'65V A	2 Ω 3 W	1'65 A
±1'65V D	15 Ω 1 W	0'22 A
+0'85V D to -1'65V D	30 Ω 1/4W	0'083 A
+3'3V D to GND	15 Ω 1 W	0'22 A
+3'3V A to -1'65V A	60 Ω 1/4W	0'0825 A

Table 7: Load board consumption

5.1.3 Software

As said in the previous point 5.1.1 all the emulation of the I2C comunication protocol is done by software. The software of this project has been done using a free development IDE (see figure 36), DevC++ wich uses a windows port of the well known linux C compiler gcc. All the code is C++ and uses the graphical interface libraries called wxWindows. For a detailed code listing see **APPENDIX 3: SOFTWARE**.

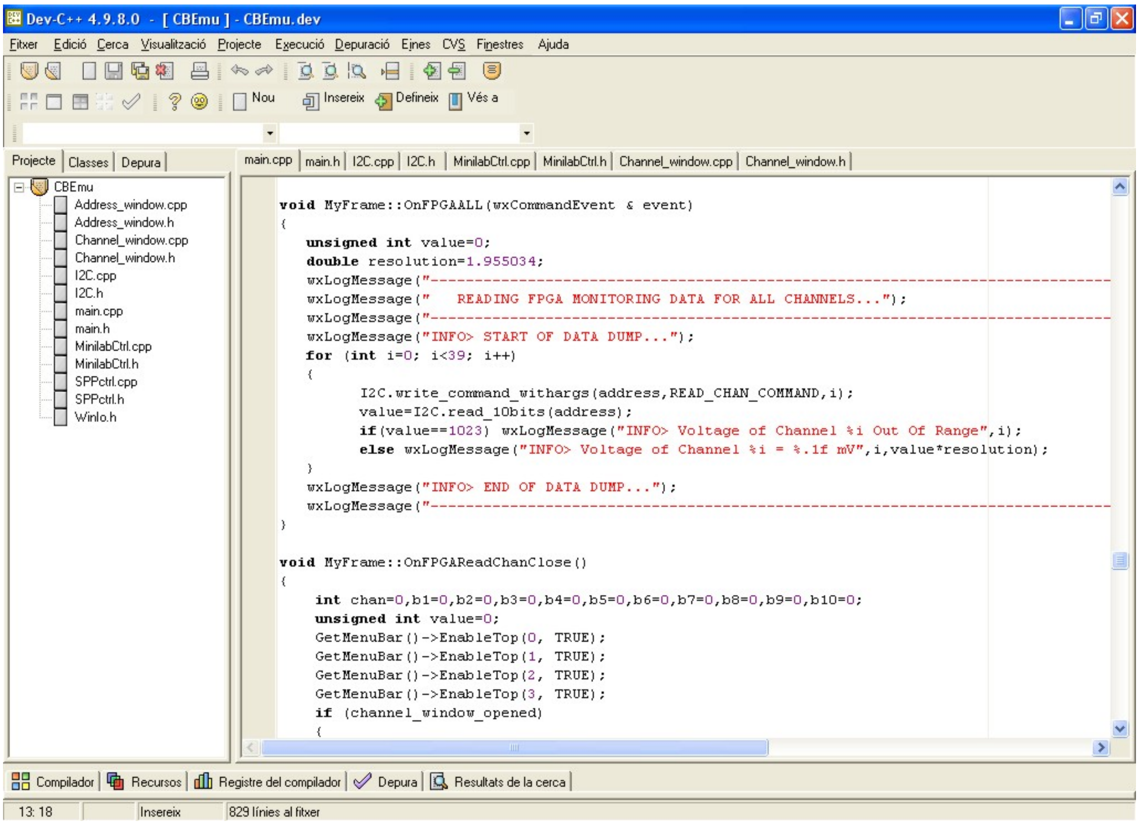


Figure 36. Dev C++ IDE

The current version and probably the only one is 0.1 since it's not foreseen to be used in the final detector and it's only for testing purposes. The aplication has been called CBEmu in honor of the Control Board Emulation functions that perform. In order to access the paralel port under Windows XP the program uses the libraries of WinIO wich permit an old stile access to any port.

The graphical interface has been designed to be as simple as possible (see figure 37), when you start the aplication you see a welcome and licence message and the software opens the port and waits for a user input. It also debugs the paralel port address range being used wich is fixed by software by the moment (most of the PC's use the same range so there has not been any need of changing it).

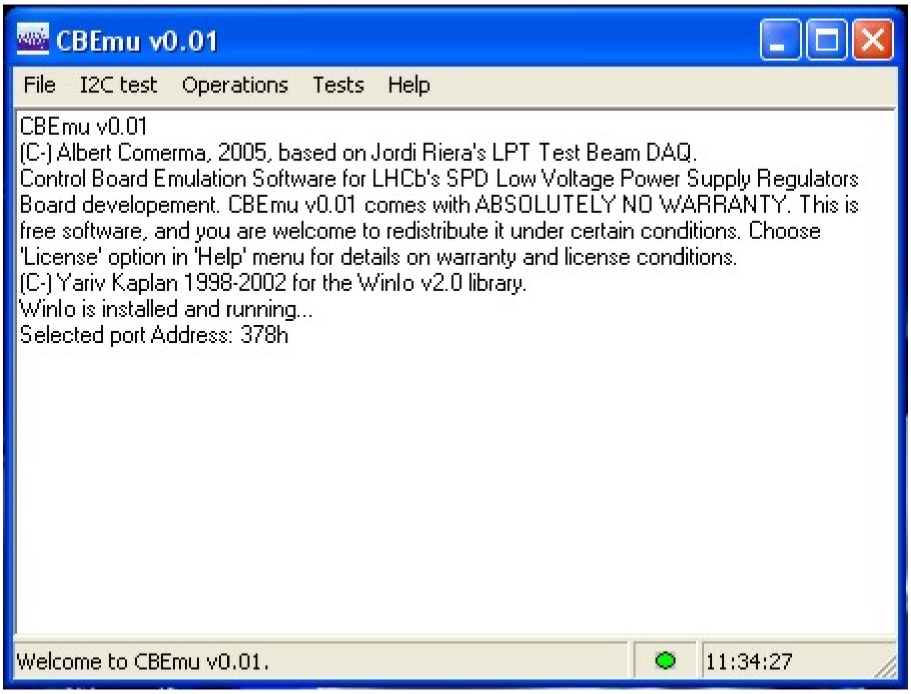


Figure 37. CBEmu at startup

Once something is pressed in the menu bar you get a sliding menu with all the possibilities. In the File menu we have the usual comands, Save Log (wich permit to save all the text in the program window to a text file) and Exit (wich quits the program). In the I2C test we have very basic functions wich where used to test the hardware interface, they are; Power up (wich bring power to the data lines that feed the transceivers), SCL and SDA (both of them generates a clock signal in the corresponding data pin). In the Operations menu we have more elaborated commands wich start to use the I2C protocol. The first operation, search for devices just sends all the possible addresses (from 1 to 127) to the bus, in write mode, and waits if any of the addresses responds with an acknowledge condition, if so it saves the address and reports it to the user (see figures 38 and 39). The next operation brings a pop-up window with a dialog in order to enter one of the valid addresses reported. This addres will be the one used in the rest of the operations. Get board ID operation asks to the device with a concrete command and waits for a response, the response is a string null terminated. This string is a text identifier to know wich kind of device is connected to the I2C (see figure 40). Next commands, send Power UP, send Power DOWN and send Power UP FIRST just sends some commands to the FPGA in write mode without waiting for any response wich will permit to get all the regulators ON, OFF, or all the regulators of the first VFE ON respectively.

The Tests menu hides even more complex operations, the first one, Read Minilab is used in order to read an external acquisition hardware wich connects to the PC through a USB connection and wich will perform analogue readout of temperature probes. The second one is Start temperature test wich will perform an automatic test of the temperature probes read by the minilab and current and voltage reads through FPGA at a fixed rate, these two functions are already under development. The next one is FPGA read channel, this functions shows a dialog window similar to the address selection in order to enter the number of the analog channel wich can be read by the FPGA, it returns the result in plain binary (see figures 41 and 42). Finally the last two Tests that can be done are FPGA data Dump and FPGA Analog channels Dump, they are quite similar, performing a read of all the analog channels connected to the FPGA, buit the first comand sends back the results translated to volage and current depending to wich channel is reading and appling the gain coeficients introduced by the Op Amps, while the last command just returns the voltage measurement (see figure 43).

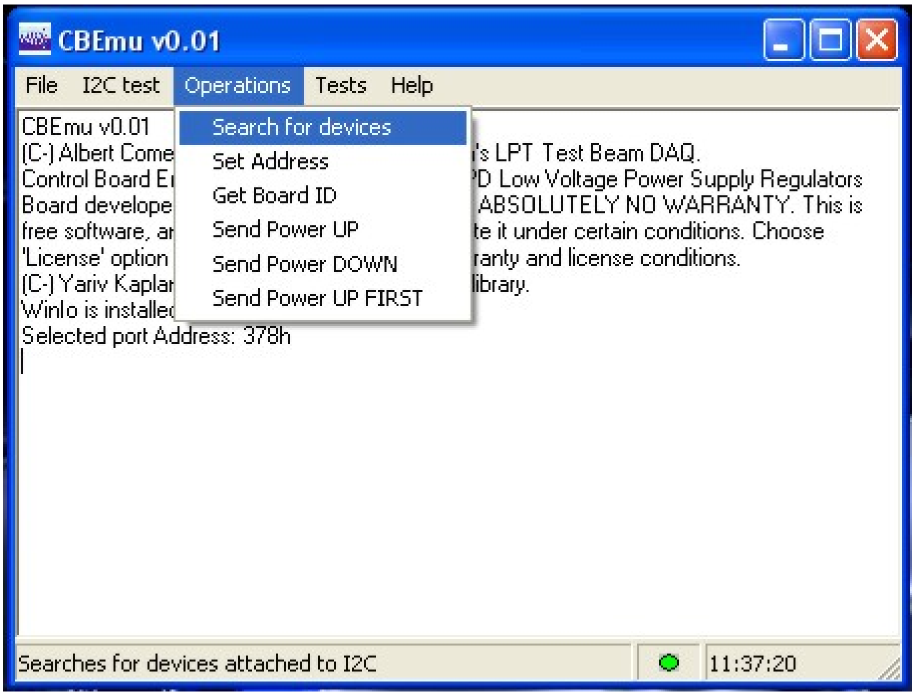


Figure 38. Operations menu

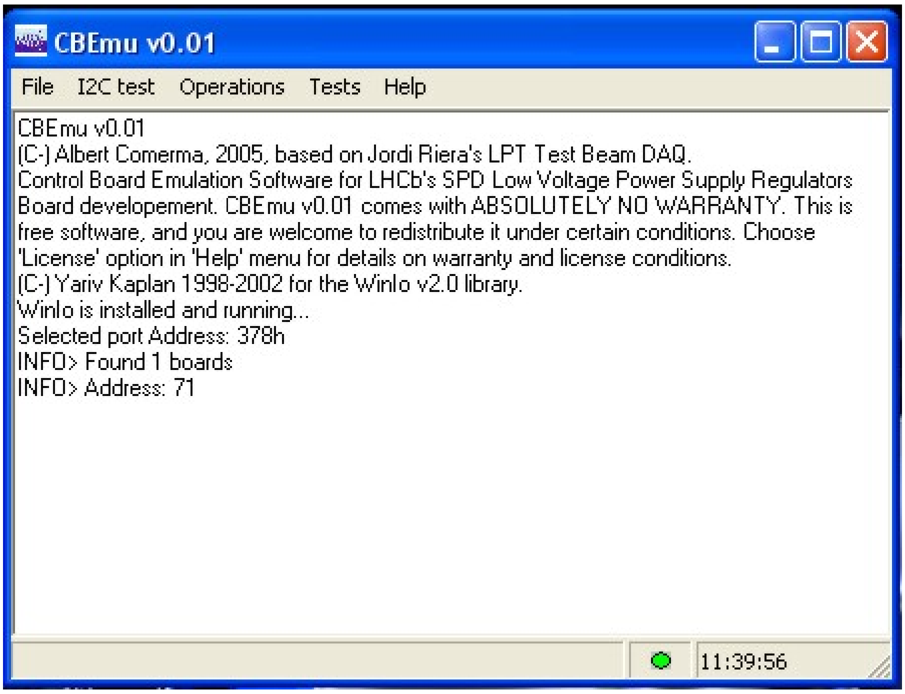


Figure 39. Search for devices result with one regulators board connected

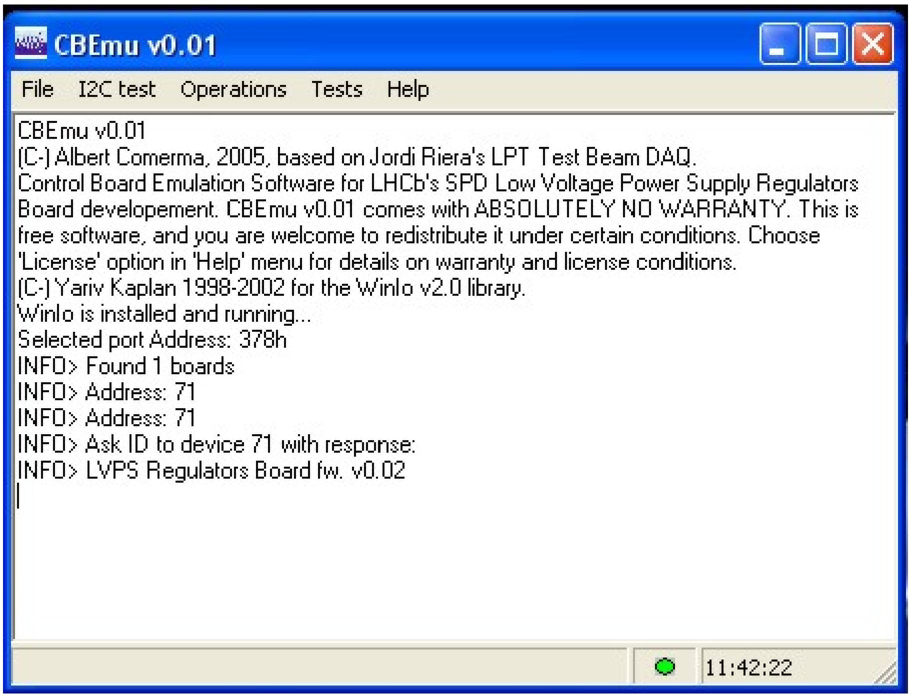


Figure 40. Get board ID result with a LVPS connected

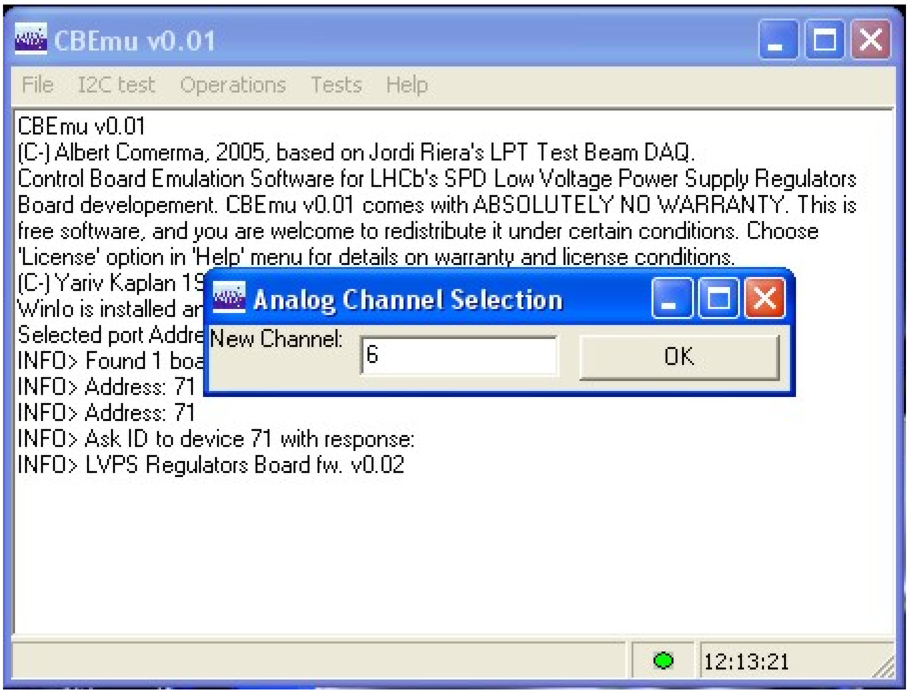


Figure 41. Analog Channel Selection

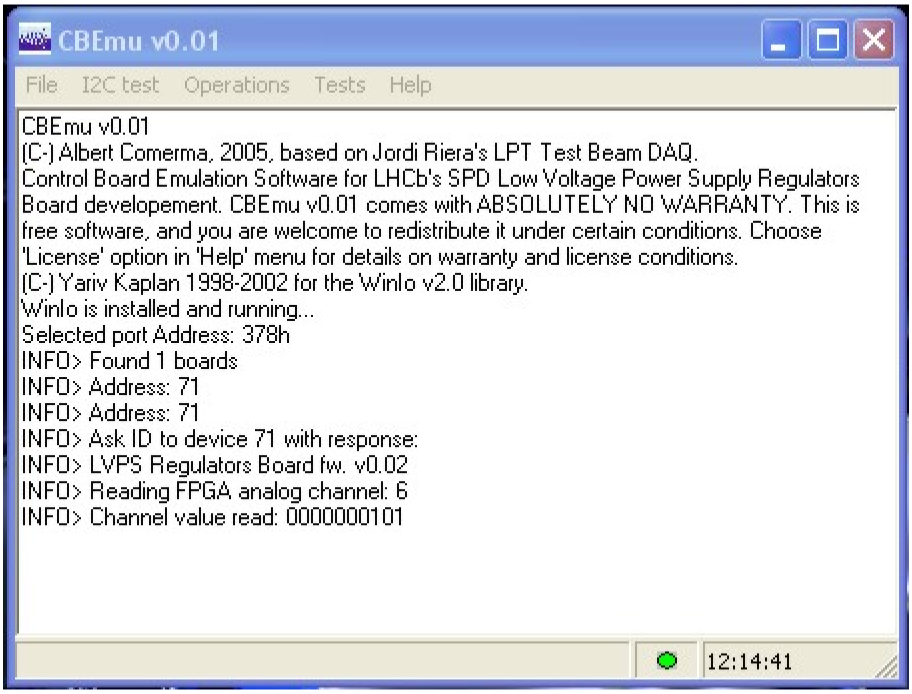


Figure 42. Analog Channel Readout

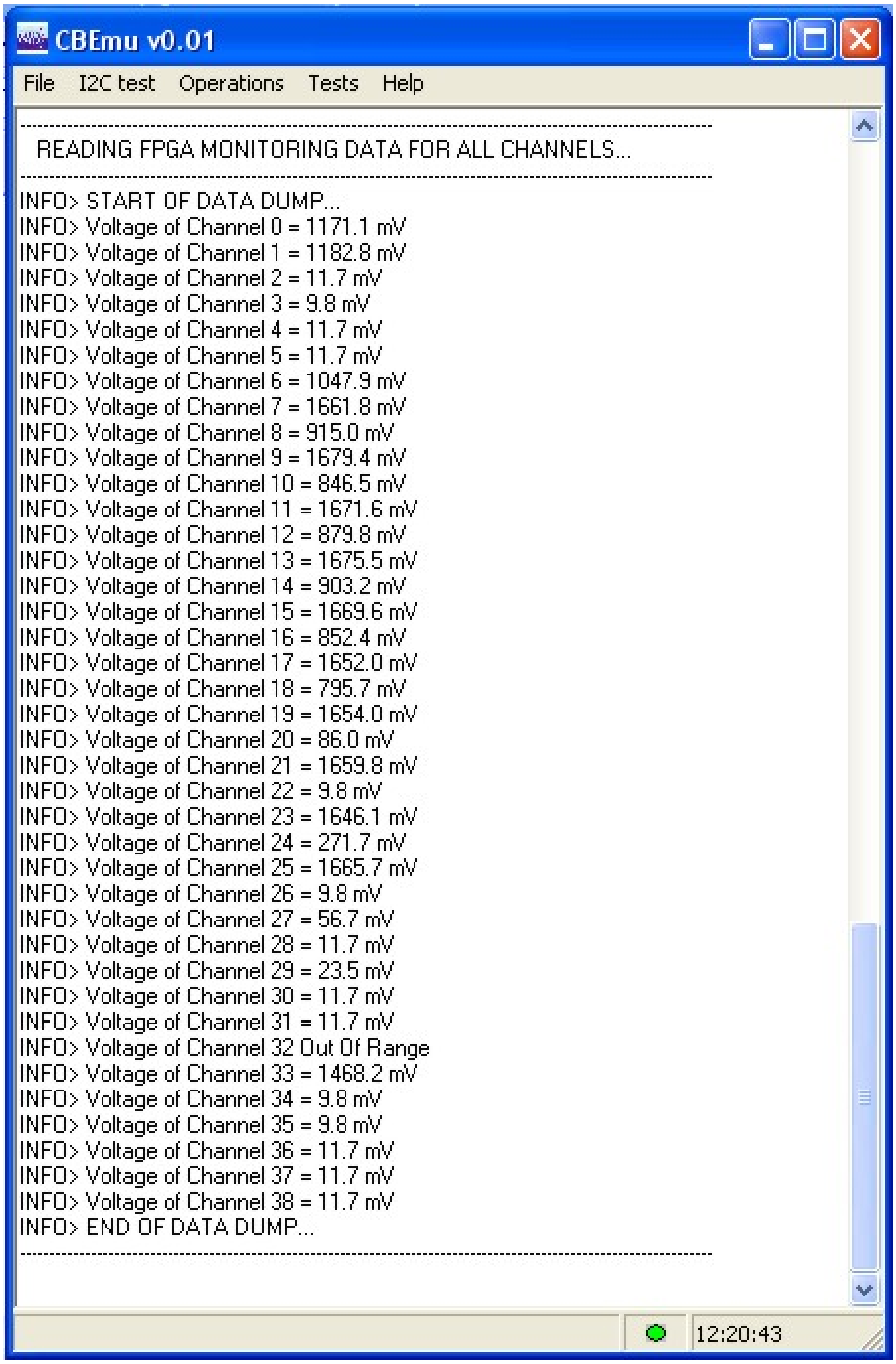


Figure 43. FPGA Analog Channels Dump

6. PERFORMANCE TESTS

In this chapter there is a small description of the functionality wich has been tested by the moment.

6.1.1 I2C performance

One of the first worries in the start-up of the prototype was the I2C bus, because it involved a pair of boards wich may be tested at the same time and the FPGA had to be programmed to check the comunication.

The JTAG programming of the FPGA was succesfully tested at the first try (second if you take into account that there where a pair of pins of the FPGA not soldered in the first try), so the next step was to look for the I2C communication.

Sowly but with the flexibility that gave us the programming of the paralel port commands and the flash FPGA we could add more and more instructions to the VHDL code, in the next figures are some examples of the comunication achieved.

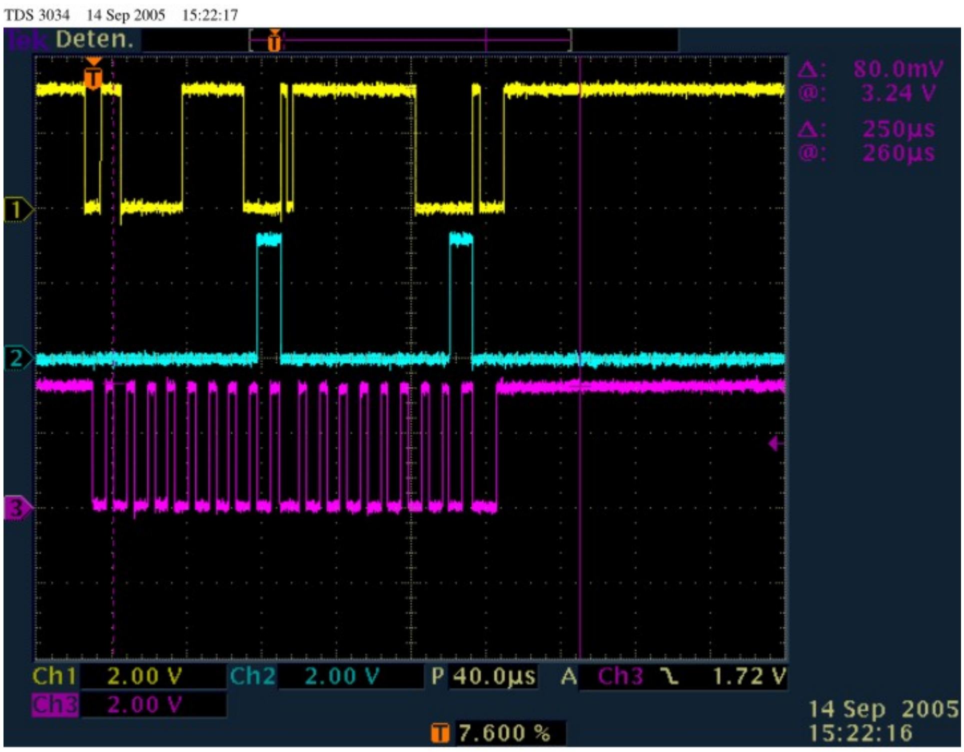


Figure 44. I2C power up command (SDA=yellow, nRE=blue, SCL=red)

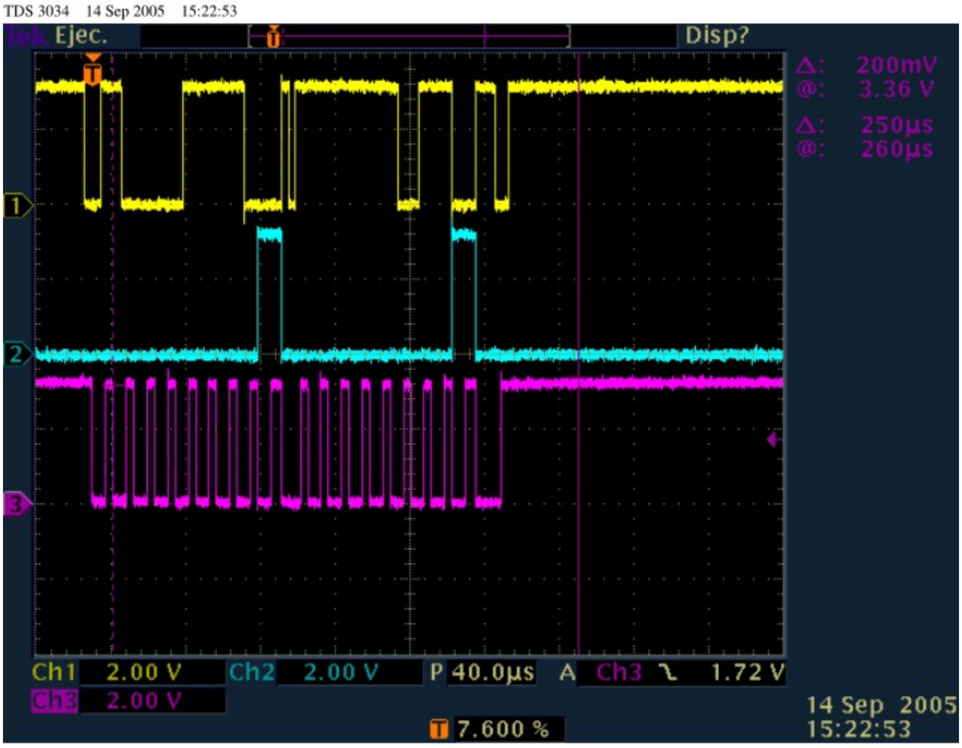


Figure 45. I2C power down command (SDA=yellow, nRE=blue, SCL=red)

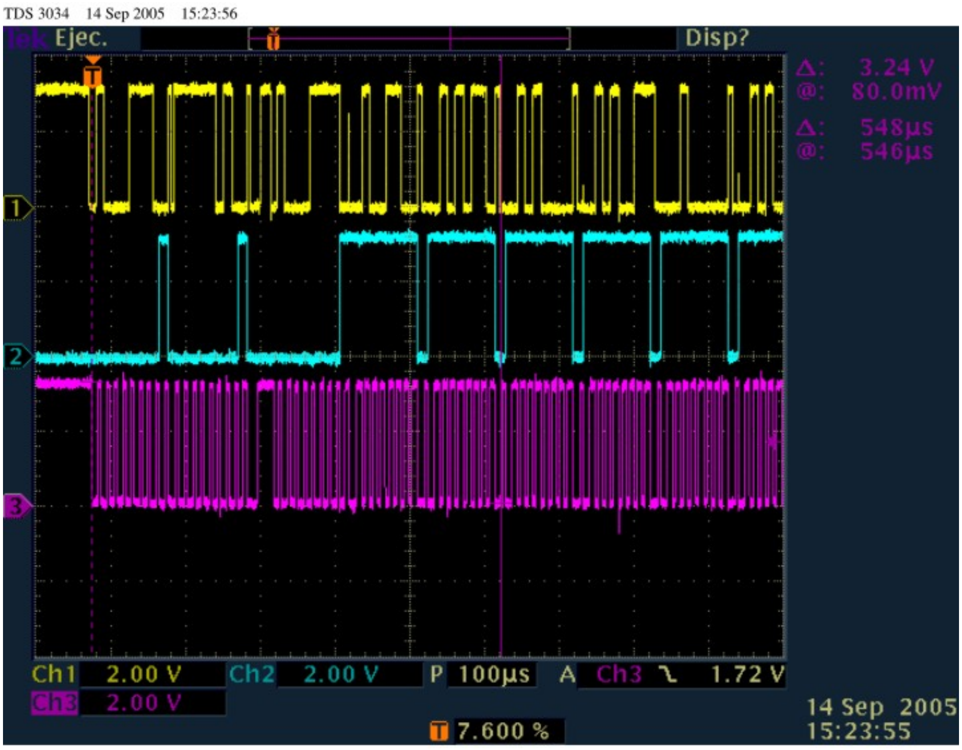


Figure 46. I2C ask ID command and partial response (SDA=yellow, nRE=blue, SCL=red)

6.1.2 Regulation performance

By the moment all the tests of voltage stability and ripple have been quite good (ripple lower than 100mV), but a longer with power consumption variations test is still under development.

6.1.3 Data acquisition with FPGA

The data acquisition with the FPGA using the ADC and the VHDL code has been probed to ve very reliable and stable, the only change that was done was the introduction of more delay between the multiplexors setup and the data readout, because it seemed not to have enough time to stablilize.

6.1.4 Cooling tests

Cooling tests where perfomed during July using the regulators board prototype and a VFE prototype in Clermont Ferrand. The tests setup was the same used for the initial debugging but we introduced the metalic heatsinks designed at Clermont with the water pipes. The heatsink structure of the VFE can bee seen in a PS VFE in figure 47. In figure 48 and 49 the LVPS heatsink and setup can be observed.

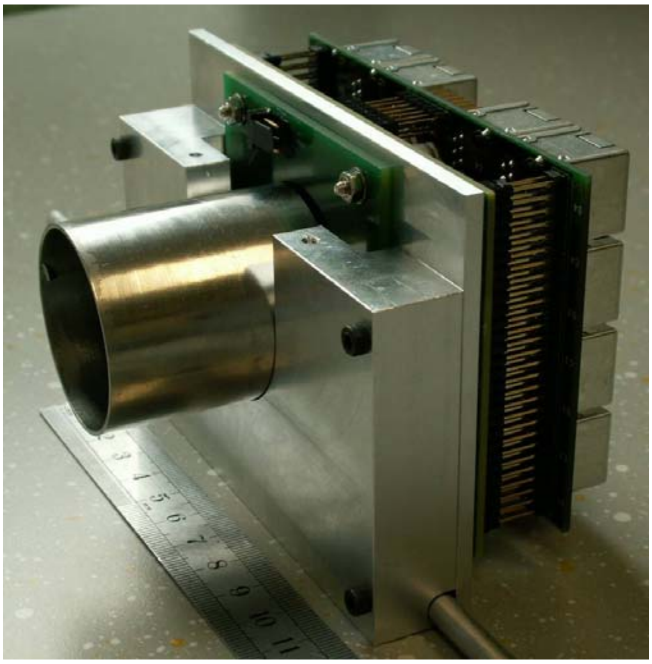


Figure 47. PS VFE cooling

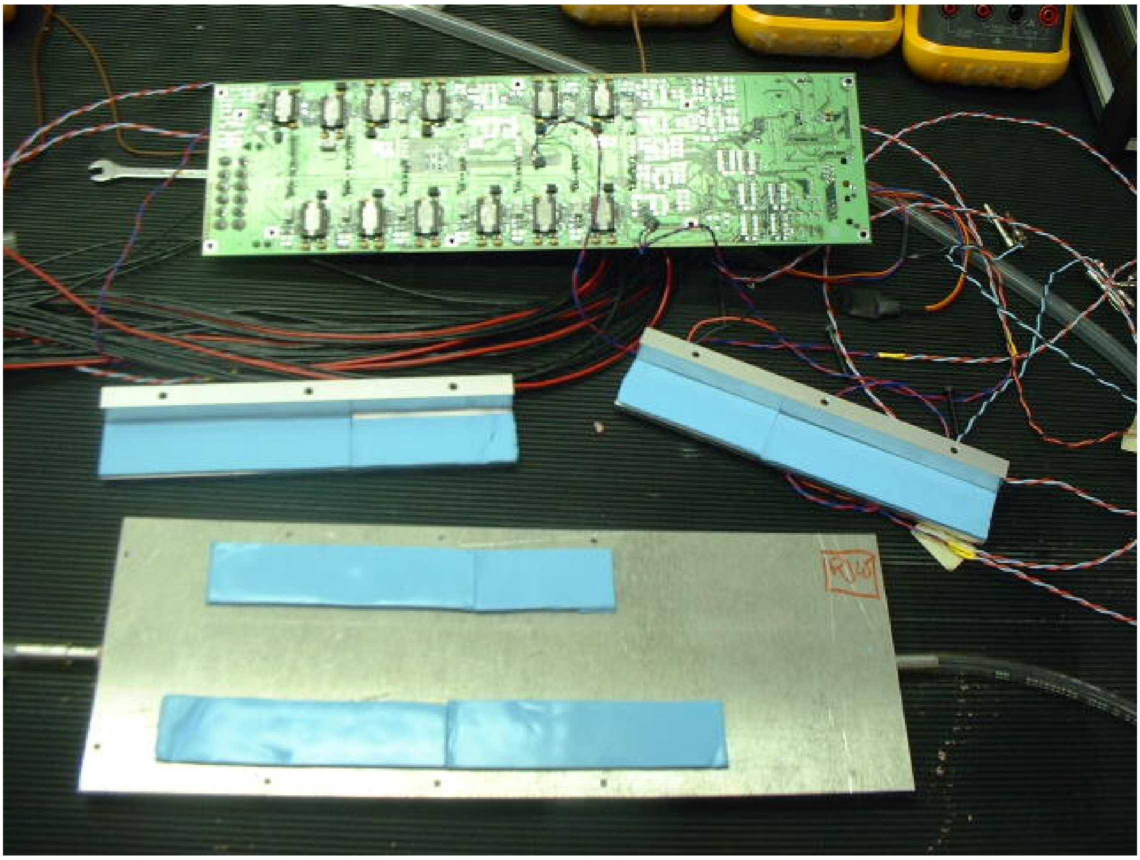


Figure 48. LVPS heatsinks with prototype board

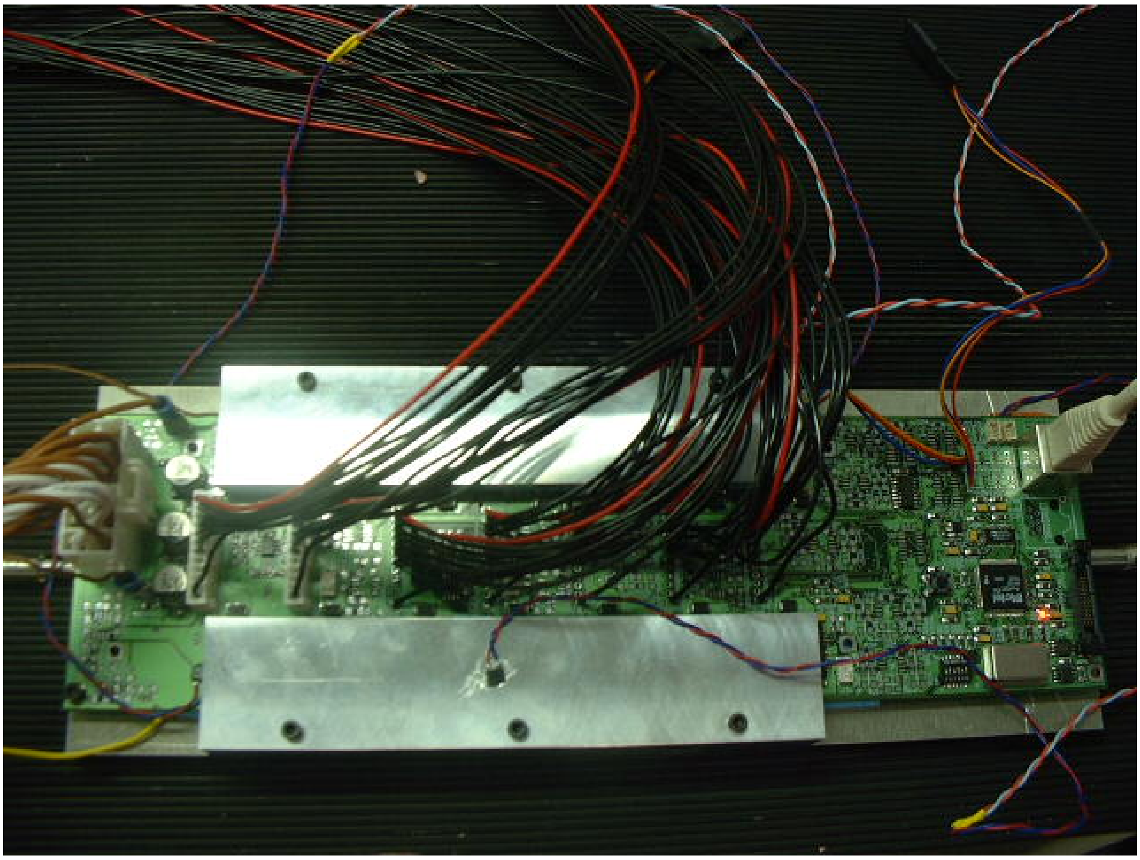


Figure 49. LVPS with cooling system

The results of the cooling tests were better than what we expected, this regulators can keep working up to 125ºC (external temperature), and with no heatsink this temperature can be reached in about a pair of minutes, but that was not seen with the heatsinks, see figure 50.

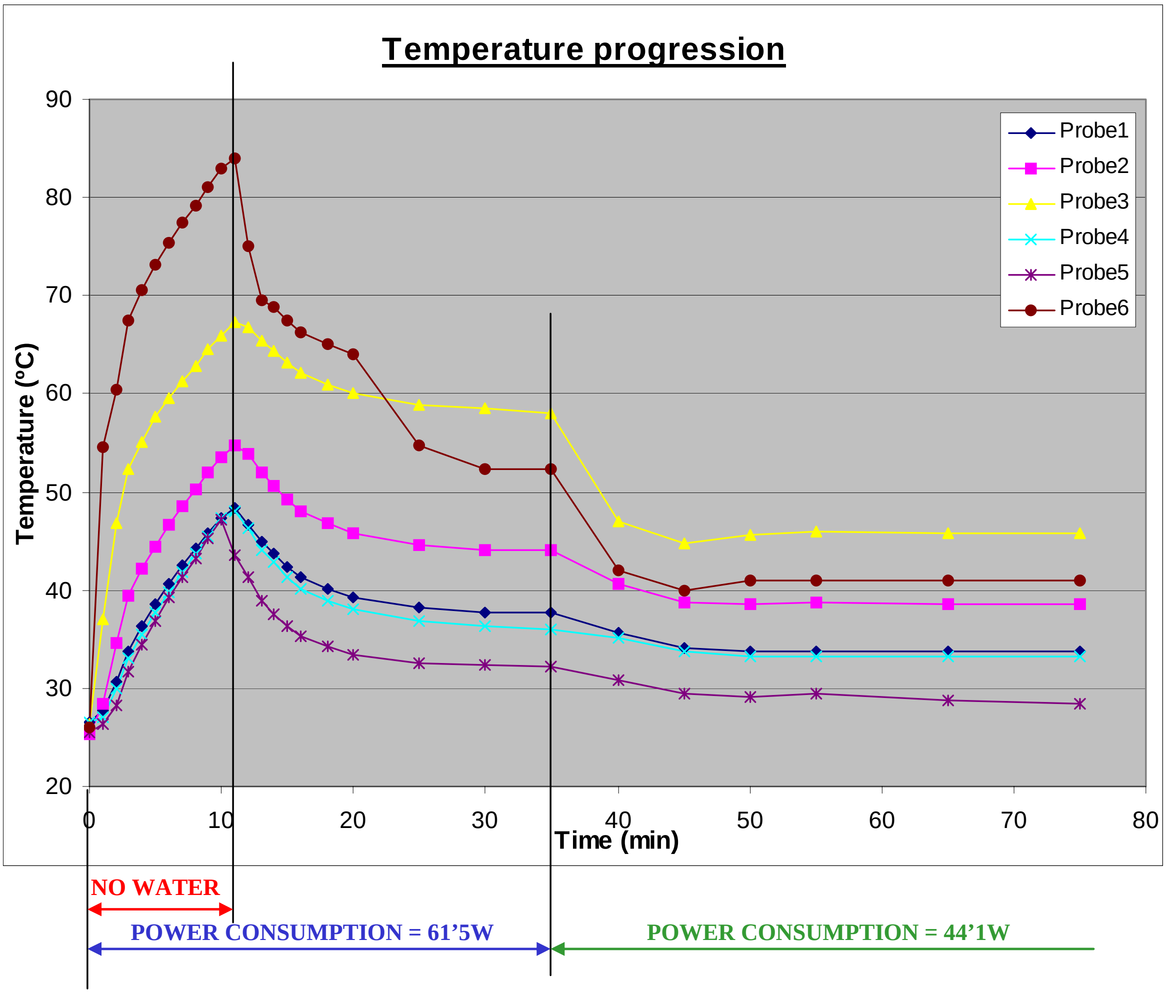


Figure 50. LVPS cooling tests

As it can be observed in figure 50 with a normal power consumption of arround 44W and with the water cooling system on the temperature is below 50ºC in the regulators surface and even cooler in other points measured.

7. CHRONOGRAM PREVISION

7.1 Scheduled Tasks

	Tasks	Time Length Expected
STAGE 1: First prototype design	Understand the requirements of the power supply: - Current limitation - Voltage regulation - Temperature monitoring Preliminary tests with regulators. Selection of analog mux. Selection of ADC. Design of amplifier circuits for sensing with ADC.	1 month
STAGE 2: Defining the hardware	Propose a prototype schematic. - Positive regulator blocs. - Negative regulator blocs. - Adaptation blocs. - Analog multiplexor and ADC blocs. - FPGA and communications bloc.	1 month
STAGE 3: Implementation	Design of the prototype pcb. Production of the pcb. Start-up process of every block except FPGA	2 month
STAGE 4: Firmware	Study of the firmware requirements. VHDL coding of blocs for sensing control and communications protocol. Test with prototype hardware.	3 month
STAGE 5: Final design	Hardware design with problems detected in previous stages. Test the final hardware ready for production.	2 month
Total time		9 months

Table 4. Scheduled tasks

7.2 Current status

At the moment of the writing of this document we are at STAGE 5, ready to do the design of the final PCB, with most of the FPGA functions tested on the actual code. And nearly all the hardware tested with some changes in mind to improve the system.

7.3 To do list

This is a list of the things left to finish this project;

- Extensive tests of voltage regulation performance.
- Extensive test of cooling.
- Frecuency rejection tests.
- Final PCB design.
- Final prototype test.
- Final VHDL code with triple voting.

8. Bibliography and references

[1] *The LHCb Technical Proposal*. CERN/LHCC 98-4.

[2] *LHCb Calorimeters Technical Design Report*. CERN/LHCC/2000-0036.

[3] Ernest Aguiló et al. *Test of Multi-anode Photomultiplier Tubes for the LHCb Scintillator Pad Detector*.

[4] *SPECS: the Serial Protocol for the Experiment Control System of LHCb*. LHCb note 2003-004.

[5] *Scintillator Pad Detector Front End Electronics Design*. LHCb note 2000-027.

[6] *Discriminator ASIC for the SPD VFE of the LHCb calorimeter*. LHCb note 2004-042.

[7] D. Gascón et al. *A BiCMOS Synchronous Pulse Discriminator for the LHCb calorimeter system*. Proc. 8th Workshop on Electronics for LHC Experiments, Colmar (France), 9-13 September 2002

[8] X. Cano et al. *Radiation Hardness on Very Front-End of SPD*. Draft.

[9] N.J. Buchanan and D.M. Gincrich, *Proton Radiation Effects in XC4036XLA Field Programmable Gate Arrays* IEEE Trans. Nucl.Sci., Vol 50, No 2, pp.263-271, 2003.

[10] L4913 datasheet from STMicroelectronics.

[11] L7913 datasheet from STMicroelectronics.

[12] ACTEL APA750 and A54SX32A LANSCE Neutron Test Report. White paper.

[13] ACTEL In Circuit Programming Application Note.

[14] *Recent Radiation Damage and Single Event Effect Results for Candidate Spacecraft Electronics* (nsrec01_W15.pdf).

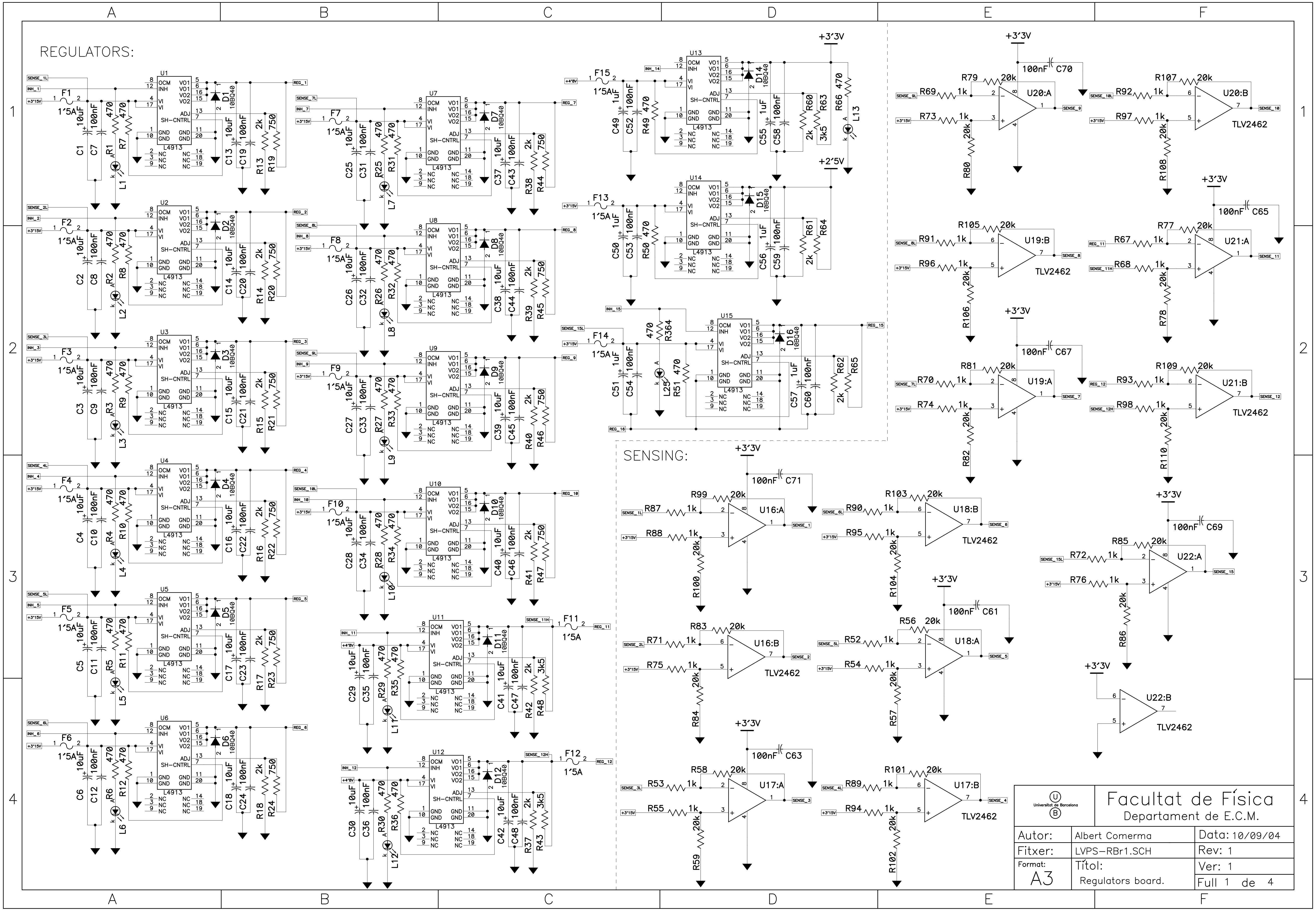
http://www.cern.ch

http://www.actel.com

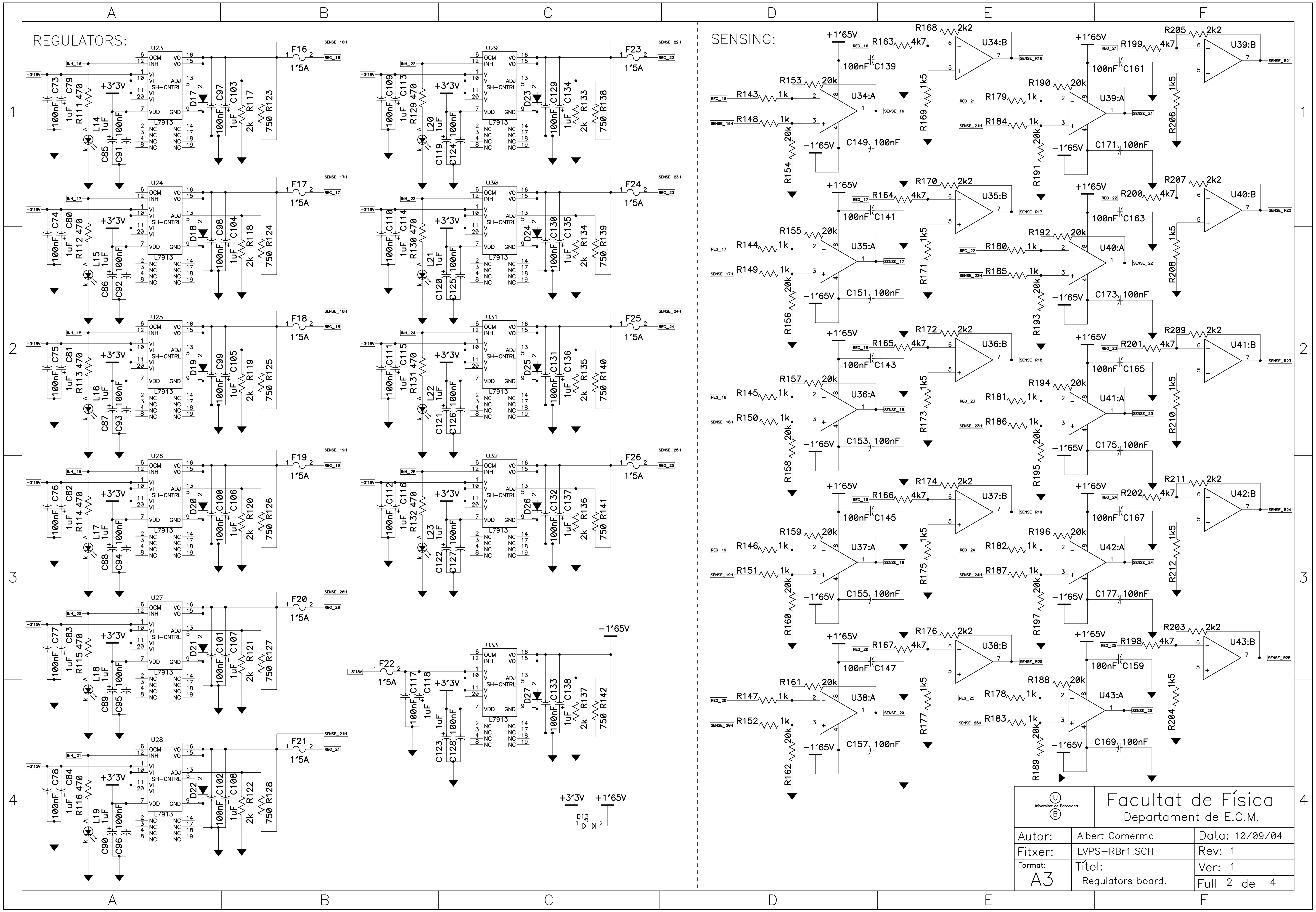
http://lhcb.cern.ch

http://www2.slac.stanford.edu/vvc/theory/fundamental.html

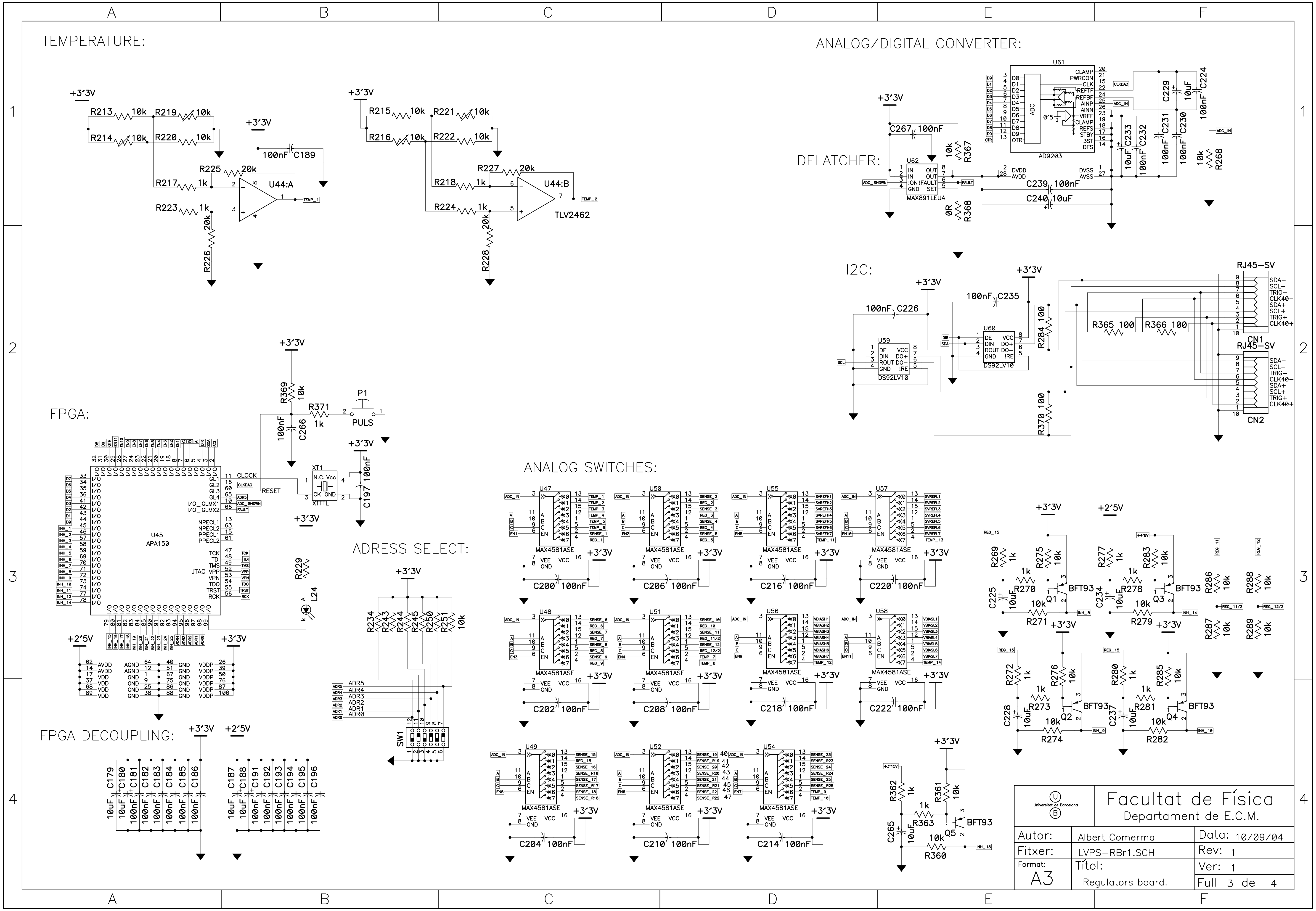
APPENDIX 1: LVPS SCHEMATICS AND PCB

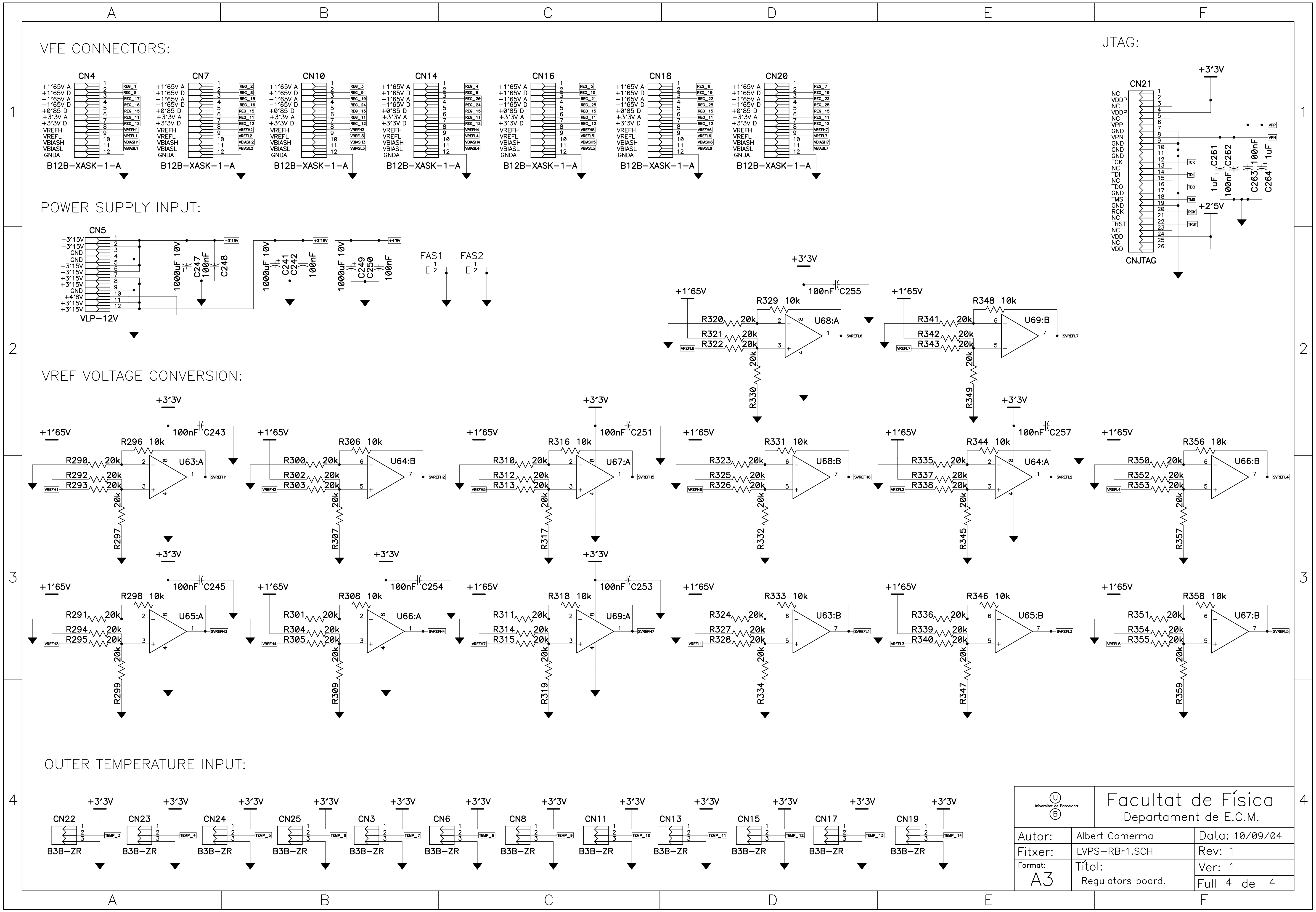


Facultat de Física	Departament de E.C.M.
Autor: Albert Comerma	Data: 10/09/04
Fitxer: LVPS-RBr1.SCH	Rev: 1
Format: A3 Títol: Regulators board.	Ver: 1
	Full 1 de 4

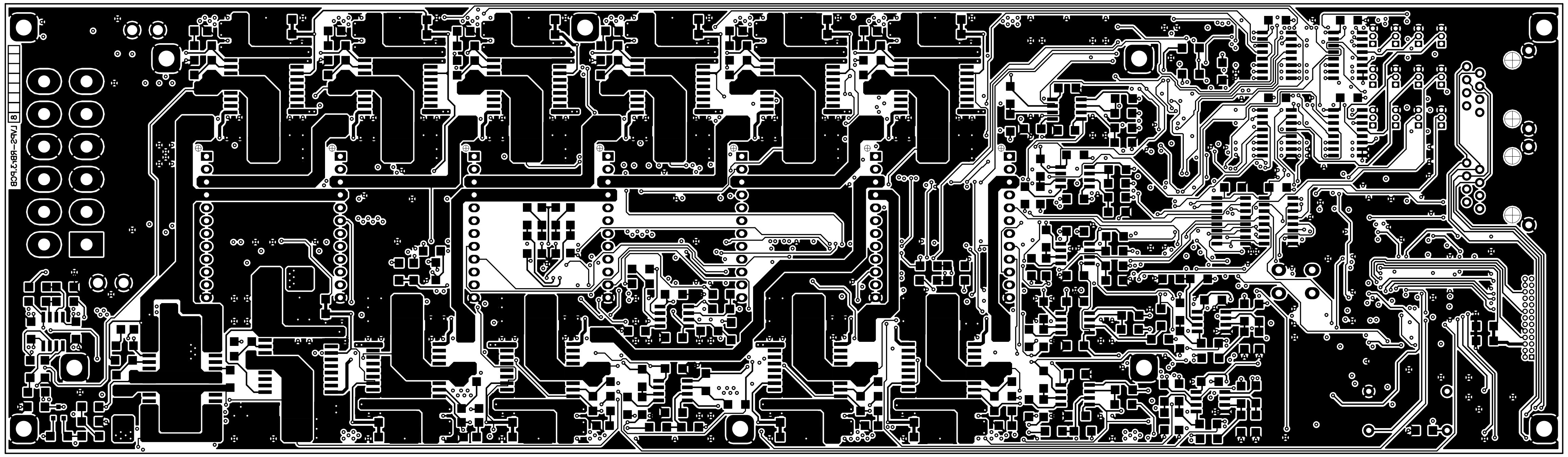


REGULATORS:
SENSING:
U23 U24 U25 U26 U27 U28 U29 U30 U31 U32 U33
L7913
U34:A U34:B U35:A U35:B U36:A U36:B U37:A U37:B U38:A U38:B
U39:A U39:B U40:A U40:B U41:A U41:B U42:A U42:B U43:A U43:B
+3'3V
+1'65V
-1'65V
1'5A
Universitat de Barcelona
Facultat de Física
Departament de E.C.M.
Autor: Albert Comerma
Data: 10/09/04
Fitxer: LVPS-RBr1.SCH
Rev: 1
Format: A3
Títol: Regulators board.
Ver: 1
Full 2 de 4

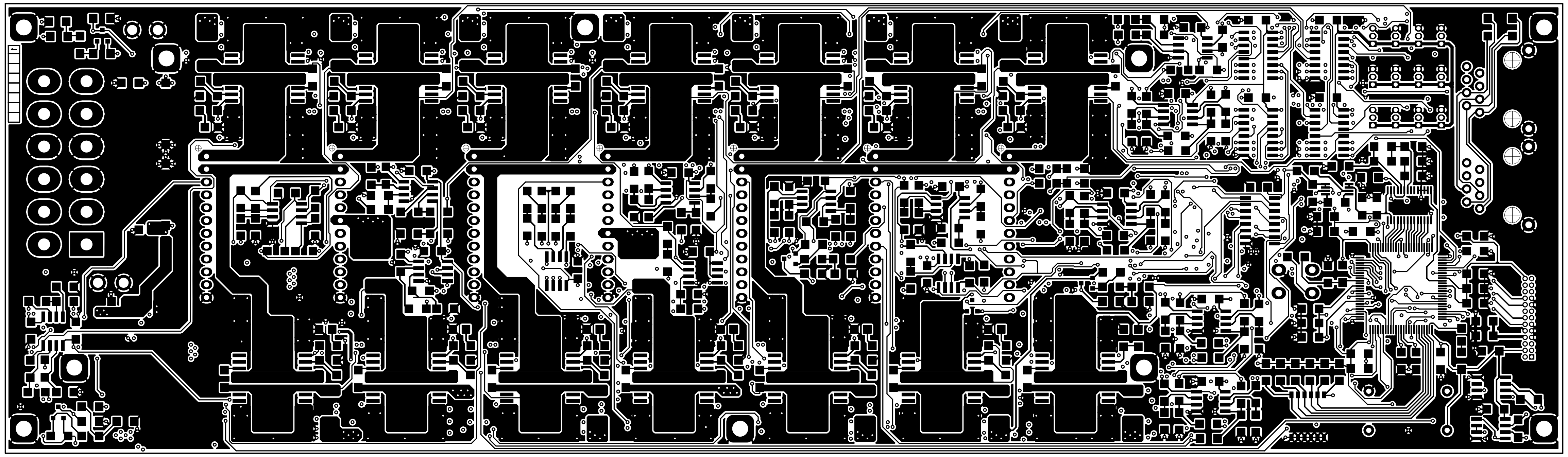

TEMPERATURE:
ANALOG/DIGITAL CONVERTER:
DELATCHER:
I2C:
FPGA:
ANALOG SWITCHES:
ADRESS SELECT:
FPGA DECOUPLING:
Facultat de Física
Departament de E.C.M.
Autor: Albert Comerma
Data: 10/09/04
Fitxer: LVPS-RBr1.SCH
Rev: 1
Format: A3
Títol: Regulators board.
Ver: 1
Full 3 de 4


VFE CONNECTORS:
JTAG:
POWER SUPPLY INPUT:
VREF VOLTAGE CONVERSION:
OUTER TEMPERATURE INPUT:
Facultat de Física
Departament de E.C.M.
Universitat de Barcelona
Autor: Albert Comerma
Fitxer: LVPS-RBr1.SCH
Data: 10/09/04
Rev: 1
Ver: 1
Format: A3
Títol: Regulators board.
Full 4 de 4

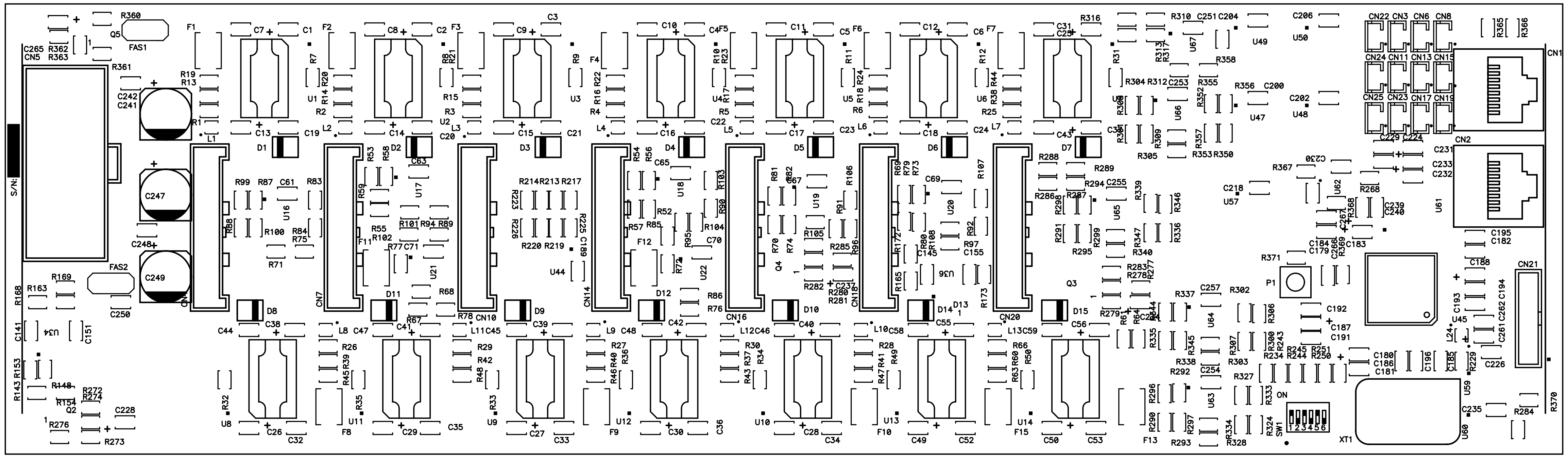
Bottom layer



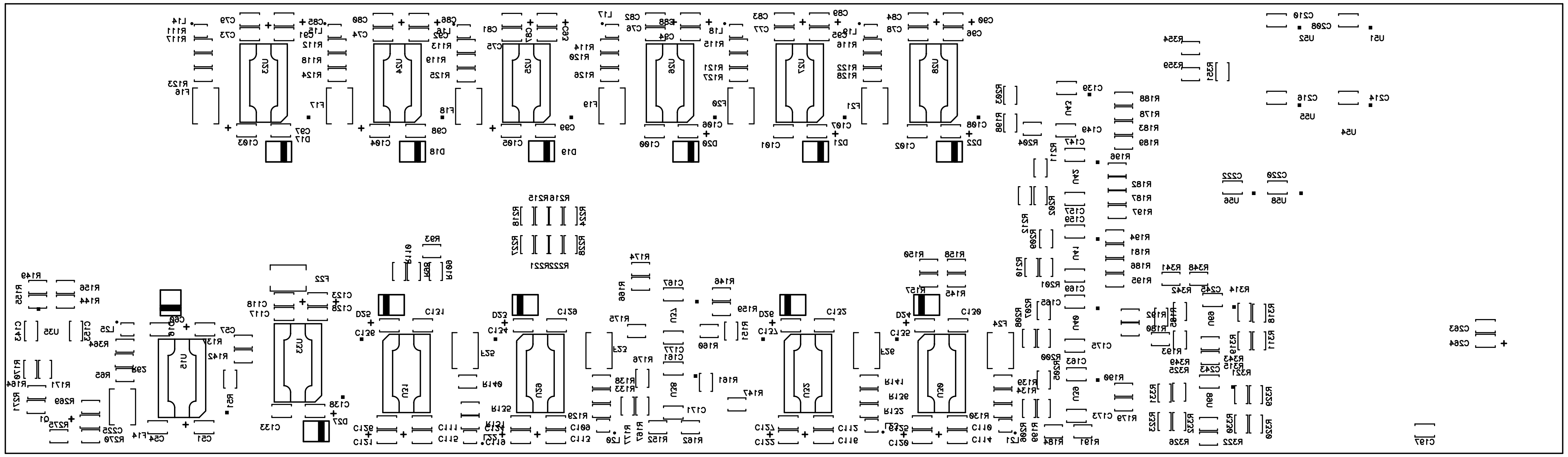
Top layer



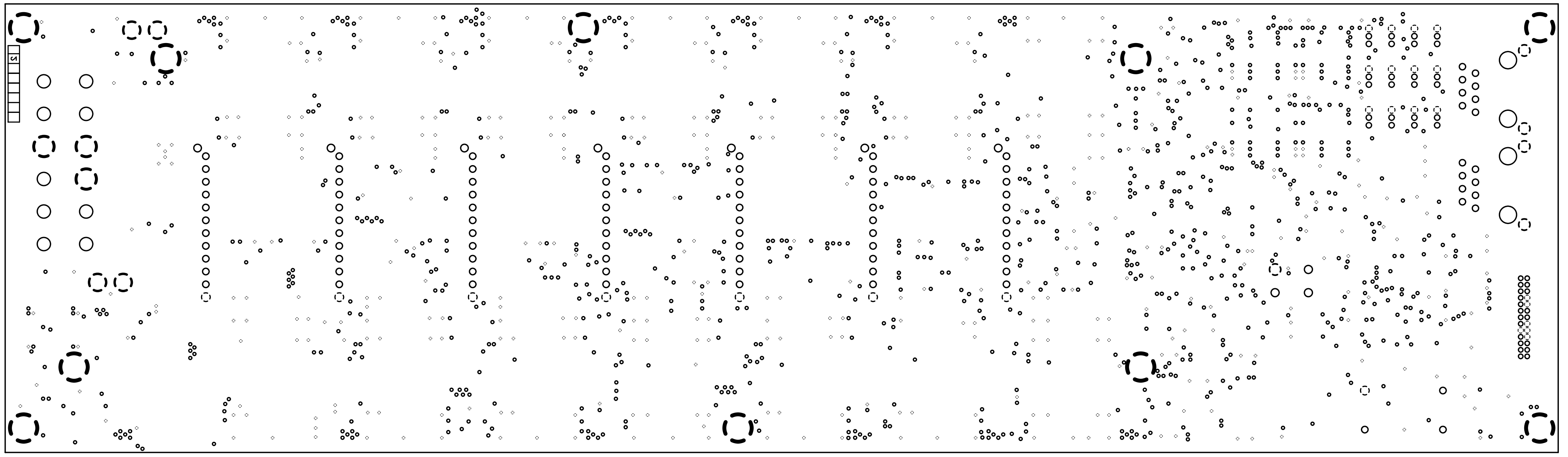
Top Silk layer



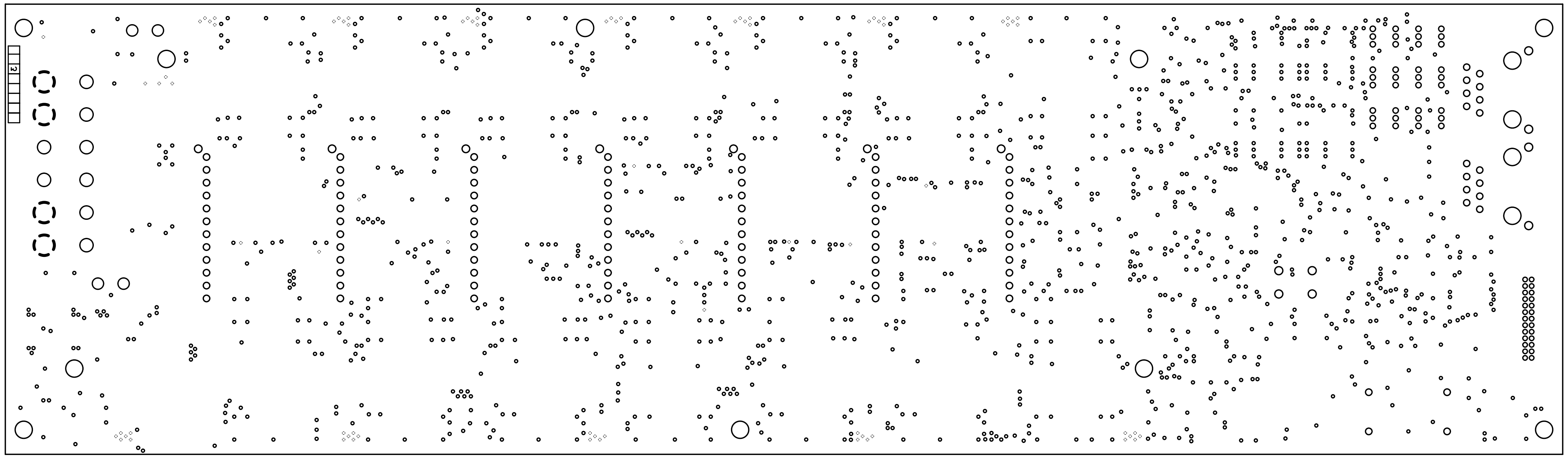
Bottom silk layer



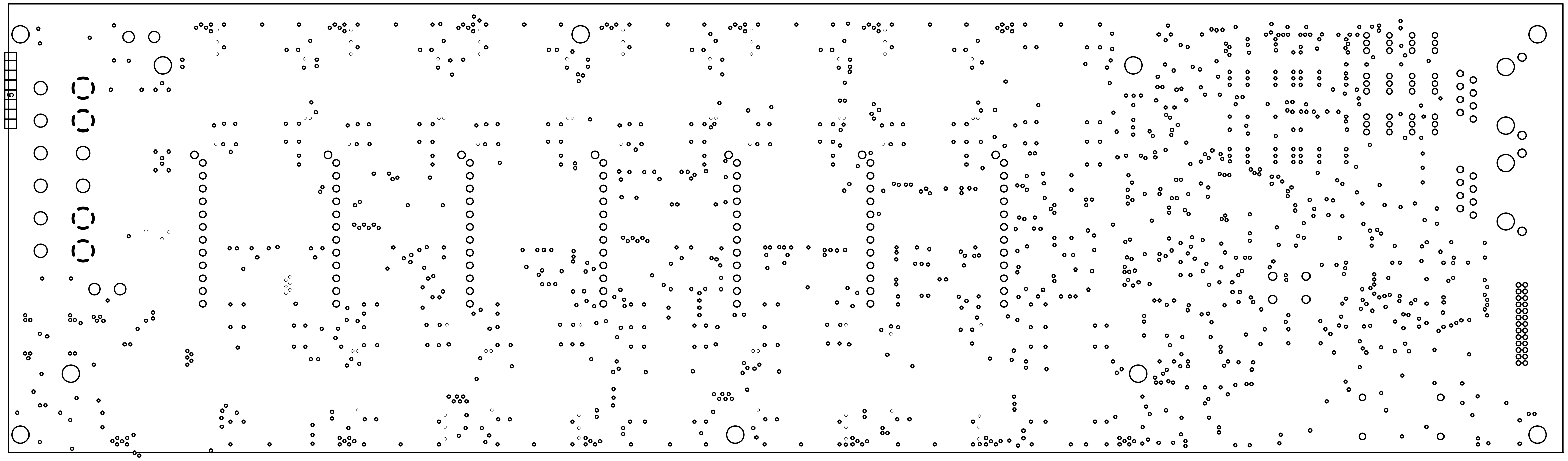
Internal ground plane



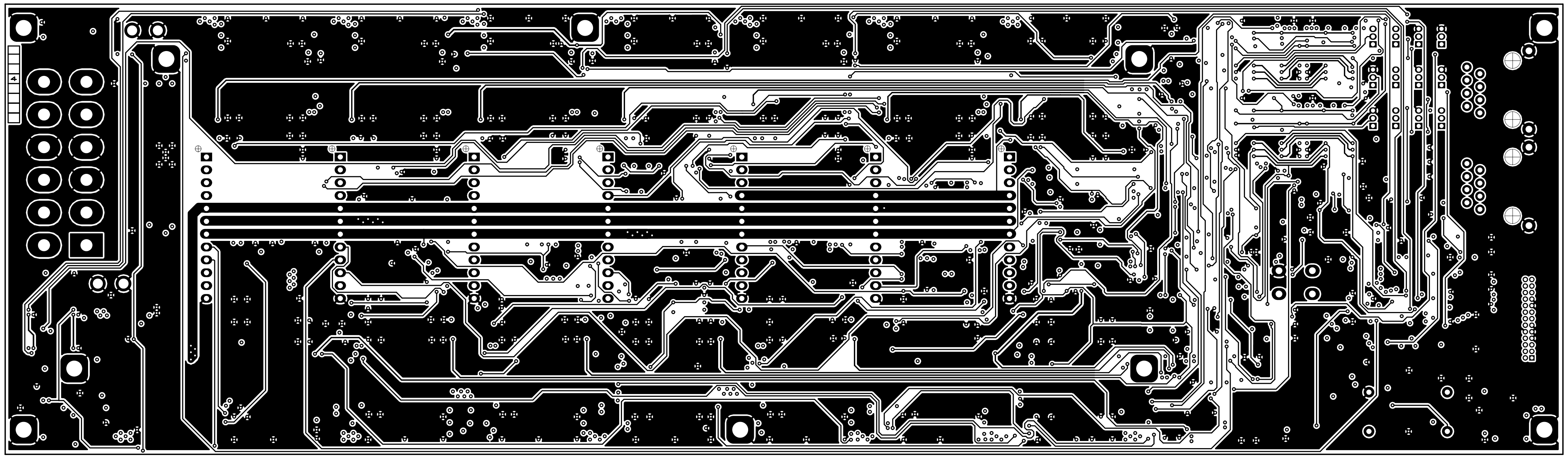
Internal power plane



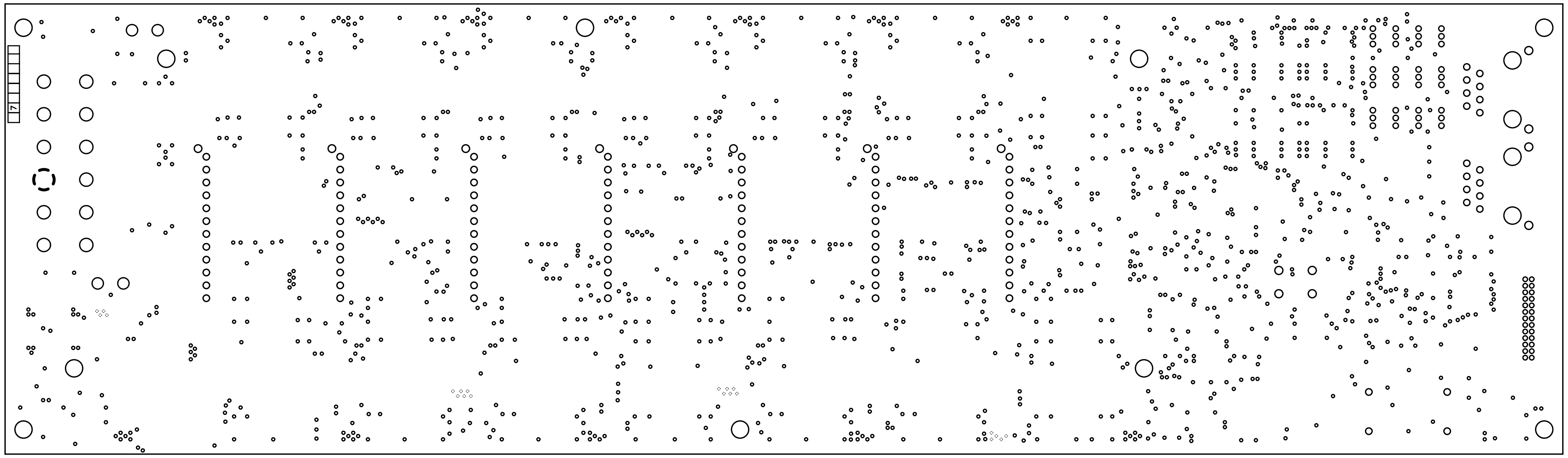
Internal power plane



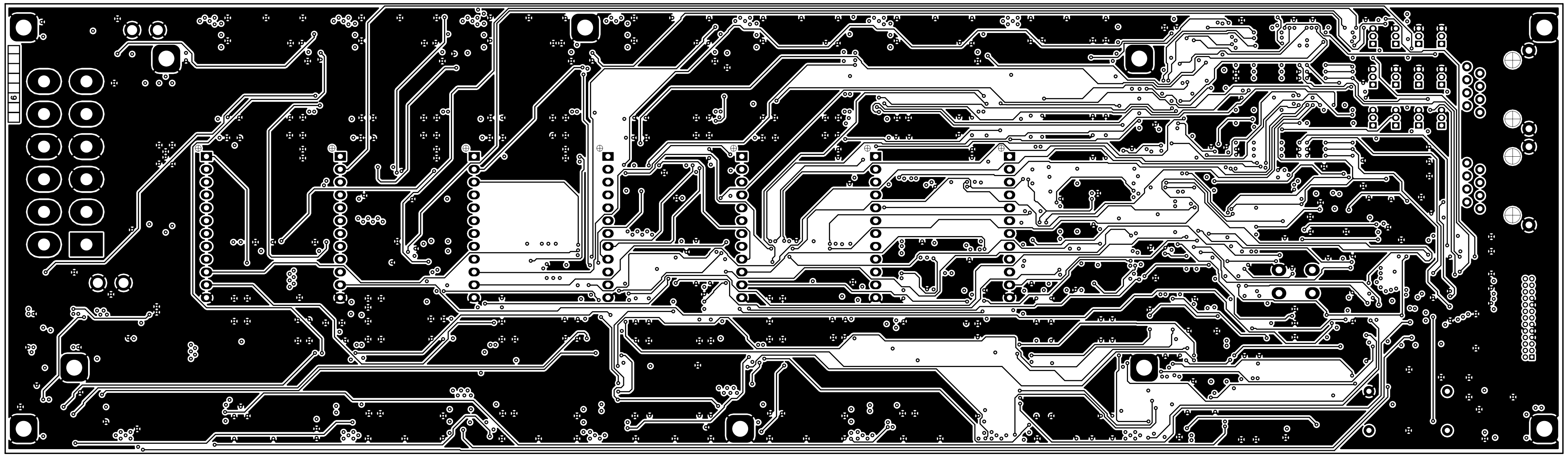
Internal signal layer 1



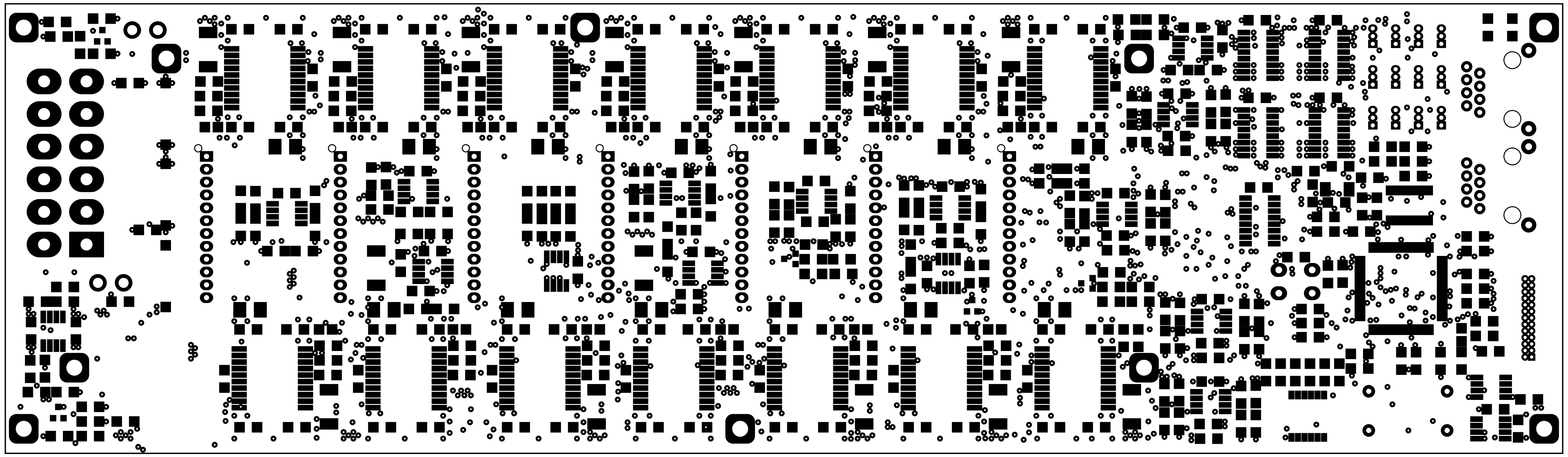
Internal power plane



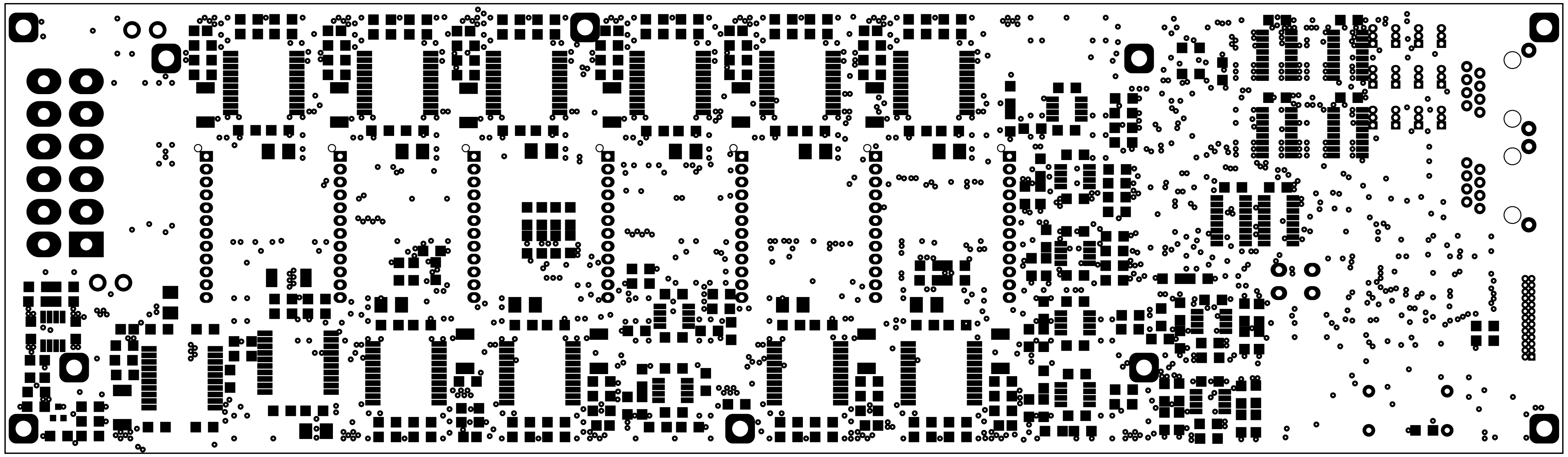
Internal signal layer 2



Top mask



Bottom mask



APPENDIX 2: FIRMWARE

File: c:\reguladors\v002\main\main.vhd

```vhdl
--Low Voltage Regulator Board controller
--Each Channel sampling and comunication via differential I2C
--Albert & Sebas Jan 2005
--Main file

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity main is
   port(
           CLK: in STD_LOGIC;                                     -- Main system clock input
           CLK_ADC: out STD_LOGIC;                                -- ADC clock output
           OTR_ADC: in STD_LOGIC;                                 -- Out Of Range ADC input
           DATA_ADC: in STD_LOGIC_VECTOR (9 downto 0);            -- 10 bits of ADC sample
           Enable: out STD_LOGIC_VECTOR (10 downto 0);            -- Multiplexors enable pins
           ABC: out STD_LOGIC_VECTOR (2 downto 0);                -- Multiplexors selection pins
           INHIBIT: out STD_LOGIC_VECTOR (23 downto 0);           -- Regulators Inhibit control
           PIN_RESET: in STD_LOGIC;                               -- System Reset input
           SDA: inout STD_LOGIC;                                  -- I2C SDA
           nRE: out STD_LOGIC;                                    -- SDA direction control
           SCL: in STD_LOGIC;                                     -- I2C SCL
           I2C_ADDRESS: in STD_LOGIC_VECTOR (5 downto 0);         -- I2C Adress switches
           FAULT: in STD_LOGIC;                                   -- Adc delatcher input
           ADC_SHDWN: out STD_LOGIC;                              -- Adc delatcher control
           LED: out STD_LOGIC                                     -- Led output
       );
end main;

architecture Structure of main is

   Component comptador is                                         -- A Big counter for driving
       port(                                                      -- led blinking and division
           CLK : in STD_LOGIC;                                    -- of main clock to a slower one
           nRESET : in STD_LOGIC;
           Q_MSB : out STD_LOGIC;
           Q_4DIV : out STD_LOGIC
        );
   end component;


   component slave_i2c is                                         -- I2C slave fsm.
   port(
       DEVICE: in STD_LOGIC_VECTOR(6 DOWNTO 0);                   --SLAVE ADDRESS TO IMPLEMENT
       CLK,nRESET: in std_logic;
       pinSCL:IN STD_LOGIC;
       pinSDA:INOUT STD_LOGIC ;
```

```
        nRE: out std_logic;                                   --1 if SDA is acting as an output or 0 otherwise.
        nreset_fifo: in std_logic;
        New_data_available: OUT STD_LOGIC;                    --positive pulse when new data is written to this device
        Data_out: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);           --last data written to this device
        Data_in: in STD_LOGIC_VECTOR(7 DOWNTO 0);             --data to send in the following read accesses
        PUSH: IN STD_LOGIC
        );
    end component;

    component fsm is                                          -- fsm for data input processing
     port(
         fifo_nRESET: out STD_LOGIC;
         CLK : in STD_LOGIC;
         NEW_DATA: in STD_LOGIC;
         nRESET : in STD_LOGIC;
         DATA_IN : in STD_LOGIC_VECTOR (7 downto 0);
         DATA_OUT : out STD_LOGIC_VECTOR (7 downto 0);
         PUSH : out  STD_LOGIC;
         CLK_ADC: in STD_LOGIC;                               -- ADC clock input
         OTR_ADC: in STD_LOGIC;                               -- Out Of Range ADC input
         DATA_ADC: in STD_LOGIC_VECTOR (9 downto 0);          -- 10 bits of ADC sample
         Enable: out STD_LOGIC_VECTOR (10 downto 0);          -- Multiplexors enable pins
         ABC: out STD_LOGIC_VECTOR (2 downto 0);              -- Multiplexors selection pins
         INHIBIT: out STD_LOGIC_VECTOR (23 downto 0)
         );
    end component;

    component reset is
     port(
        CLK: in STD_LOGIC;
        IN_RESET: in STD_LOGIC;
        nRESET: out STD_LOGIC
        );
    end component;

Signal Q: STD_LOGIC;
Signal NEW_DATA: STD_LOGIC;
Signal Data_in: STD_LOGIC_VECTOR(7 downto 0);
Signal Data_out: STD_LOGIC_VECTOR(7 downto 0);
Signal PUSH: STD_LOGIC;
Signal nRESET: STD_LOGIC;
Signal DEVICE: STD_LOGIC_VECTOR (6 downto 0) := "1000000";
Signal Clock_conv: STD_LOGIC;
Signal fifo_nRESET: STD_LOGIC;
Signal COUNT: integer := 0;

begin

    RES1: reset
    port map (CLK, PIN_RESET, nRESET);

    CNT1: comptador
    port map (CLK, nRESET, Q, Clock_conv);

    I2C1: slave_i2c
```

```
   port map (DEVICE, CLK, nRESET, SCL, SDA, nRE, fifo_nRESET, NEW_DATA, Data_In, Data_Out, PUSH);

   FSM1: fsm
   port map (fifo_nRESET, CLK, NEW_DATA, nRESET, Data_In, Data_Out, PUSH, Clock_conv, OTR_ADC, DATA_ADC, Enable, ABC, INHIBIT);

   LED <= Q;
   DEVICE(6) <= '1';
   DEVICE(5 downto 0) <= I2C_ADDRESS;
   CLK_ADC <= Clock_conv;


   adc_fault: process (CLK, nRESET)
   begin
       if nRESET ='0' then
           ADC_SHDWN <= '1';
--     elsif CLK'event and CLK='1' then
--         if (FAULT = '0') then
--             if (COUNT < 50) then
--                 ADC_SHDWN <='1';
--                 COUNT <= COUNT + 1;
               else
                   ADC_SHDWN <='0';
--             end if;
--         else
--             ADC_SHDWN <= '0';
--             COUNT <= 0;
--         end if;
       end if;
   end process adc_fault;

end Structure;
```

```
---------------------------------------------------------------------------------------------------
--
-- Title       : reset
-- Author      : albert
-- Company     : UB
--
---------------------------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity reset is
     port(
            CLK : in STD_LOGIC;
            IN_RESET : in STD_LOGIC;
            nRESET : out STD_LOGIC
         );
end reset;

architecture structure of reset is

component tmr_flipflop is
    port(
        clk : in  std_logic;
        enable: in std_logic;
        nReset: in std_logic;
        Set: in std_logic;
        D: in std_logic;
        Q: out std_logic
    );
end component;

--signals here

Signal reset_intermig: std_logic;
Signal reset_intermig2: std_logic;
Signal no_reset_int: std_logic;

begin

    FLIP1: tmr_flipflop
    port map (CLK, '1', '1', '1', no_reset_int, reset_intermig);

    FLIP2: tmr_flipflop
    port map (CLK, '1', '1', '1', reset_intermig, reset_intermig2);

    nRESET <= no_reset_int and reset_intermig2;
--  no_reset_int <= not IN_RESET;       --if input reset active for "1"

    no_reset_int <= IN_RESET;           --if input reset active for "0"

end structure;
```

```vhdl
--------------------------------------------------------------------------------------------------------
--
-- Title       : comptador
-- Design      : v2.0
-- Author      : albert
-- Company     : UB
--
--------------------------------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity comptador is
     generic (LONGITUD: integer := 23);
     port(
         CLK : in STD_LOGIC;
         nRESET : in STD_LOGIC;
         Q_MSB : out STD_LOGIC;
         Q_4DIV : out STD_LOGIC
         );
end comptador;

architecture comptador of comptador is
begin
    process (CLK, nRESET)
        variable Qint : STD_LOGIC_VECTOR (LONGITUD downto 0);
    begin
        if nRESET ='0' then
            Qint := (others => '0');
        else
            if CLK'event and CLK='1' then Qint:=Qint+1;
            end if;
        end if;
    Q_MSB <= Qint(LONGITUD);
    Q_4DIV <= Qint(1);
end process;
end comptador;
```

```vhdl
-------------------------------------------------------------------------------------------------
--
-- Title       : fsm
-- Design      : Low Voltage Power Supply, Regulators Board Control
-- Author      : Albert Comerma
-- Company     : UB
--
-------------------------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.STD_LOGIC_ARITH.all;

entity fsm is
   port(
        fifo_nRESET : out STD_LOGIC;                          -- Control of i2c fifo reset pin
        CLK : in STD_LOGIC;                                   -- Clock input
        NEW_DATA: in STD_LOGIC;                               -- New data at i2c register present
        nRESET : in STD_LOGIC;                                -- Reset
        DATA_IN : in STD_LOGIC_VECTOR (7 downto 0);           -- Data in from i2c
        DATA_OUT : out STD_LOGIC_VECTOR (7 downto 0);         -- Data out to i2c fifo
        PUSH : out  STD_LOGIC;                                -- Control signal for i2c fifo write
        CLK_ADC: in STD_LOGIC;                                -- ADC clock input
        OTR_ADC: in STD_LOGIC;                                -- Out Of Range ADC input
        DATA_ADC: in STD_LOGIC_VECTOR (9 downto 0);           -- 10 bits of ADC sample
        Enable: out STD_LOGIC_VECTOR (10 downto 0);           -- Multiplexors enable pins
        ABC: out STD_LOGIC_VECTOR (2 downto 0);               -- Multiplexors selection pins
        INHIBIT: out STD_LOGIC_VECTOR (23 downto 0)
        );
end fsm;

architecture behave of fsm is

-- main fsm state definitions and control signals
type STATE is (IDLE, DECODE, EXECUTE);
signal CURRENT_STATE, NEXT_STATE : STATE;
signal UNKNOWN_COMMAND, COMMAND_FINISHED : STD_LOGIC;
signal ask_id : STD_LOGIC;
signal ask_read : STD_LOGIC;
signal send_data : STD_LOGIC;
signal SENT_BYTES : integer;
signal ticks : integer;
signal DATA : STD_LOGIC_VECTOR(9 downto 0);
signal send_data_req_id : STD_LOGIC;
signal send_data_req_read : STD_LOGIC;
-- id fsm state definitions and control signals
type ID_STATE is (IDLE, SEND);
signal ID_CURRENT_STATE, ID_NEXT_STATE : ID_STATE;
-- send to i2c fsm state definitions and control signals
type SEND_STATE is (IDLE, LOAD, SEND);
signal SEND_CURRENT_STATE, SEND_NEXT_STATE : SEND_STATE;
```

```
-- read fsm state definitions and control signals
type READ_STATE is (IDLE, SETMUX, WAIT_ADC, SEND);
signal READ_CURRENT_STATE, READ_NEXT_STATE : READ_STATE;
-- table of constants to use in i2c routines (commands and static data)
constant ID : String(1 to 31) := "LVPS Regulators Board fw. v0.02";
constant ID_COMMAND : std_logic_vector(7 downto 0) := X"FA";
constant POWU_COMMAND : std_logic_vector(7 downto 0) := X"FB";
constant POWD_COMMAND : std_logic_vector(7 downto 0) := X"FC";
constant POWU_ONE_COMMAND : std_logic_vector(7 downto 0) := X"FD";
constant READ_COMMAND : std_logic_vector(7 downto 0) := X"FE";
-- function to convert character to standard logic vector
function Char2SLV8 (c : character) return std_logic_vector is
begin
return std_logic_vector(conv_unsigned(character'pos(c),8));
end function;

begin
-- main fsm implementation
   main_fsm_reg: process (CLK, nRESET)
   begin
        if nRESET ='0' then
            fifo_nRESET <= '0';
            CURRENT_STATE <= IDLE;
            UNKNOWN_COMMAND <= '0';
            INHIBIT <= (others => '1');
            send_data <= '0';
            ask_id <= '0';
            ask_read <= '0';

        elsif CLK'event and CLK='1'
            then CURRENT_STATE <= NEXT_STATE;
            case CURRENT_STATE is
            when IDLE => fifo_nRESET <= '1';
                         UNKNOWN_COMMAND <= '0';
                         COMMAND_FINISHED <= '1';
                         ask_id <= '0';
                         ask_read <= '0';
                         send_data <= '0';
            when DECODE =>  case DATA_IN is
                            when ID_COMMAND =>  ask_id <= '1';
                                                COMMAND_FINISHED <= '0';
                            when POWU_COMMAND => INHIBIT <= (others => '0');
                            when POWD_COMMAND => INHIBIT <= (others => '1');
                            when POWU_ONE_COMMAND => INHIBIT <= "111111110001001101111110";
                            when READ_COMMAND => ask_read <= '1';
                                                 COMMAND_FINISHED <= '0';
                            when others =>  UNKNOWN_COMMAND <= '1';
                            end case;

            when EXECUTE => if (ask_id = '1') then
                                if (SENT_BYTES > 32) then
                                COMMAND_FINISHED <= '1';
                                ask_id <= '0';
                                end if;
                            elsif (ask_read = '1') then
                                if (SENT_BYTES > 2) then
                                COMMAND_FINISHED <= '1';
                                ask_read <= '0';
                                end if;
                            end if;
                            send_data <= send_data_req_id or send_data_req_read;
```

```
            when others => COMMAND_FINISHED <= '1';
                           UNKNOWN_COMMAND <= '1';
                           fifo_nRESET <= '0';
        end case;
        end if;
    end process main_fsm_reg;

    main_fsm_proc: process (CURRENT_STATE, NEW_DATA, UNKNOWN_COMMAND, COMMAND_FINISHED)
    begin
        NEXT_STATE<=CURRENT_STATE;
        case CURRENT_STATE is
            when IDLE => if (NEW_DATA = '1') then
                            NEXT_STATE <= DECODE;
                         end if;
            when DECODE => if (UNKNOWN_COMMAND = '1') then
                                NEXT_STATE <= IDLE;
                           else
                                NEXT_STATE <= EXECUTE;
                           end if;
            when EXECUTE => if (COMMAND_FINISHED = '1') then
                                NEXT_STATE <= IDLE;
                            else
                                NEXT_STATE <= CURRENT_STATE;
                            end if;
        end case;
    end process main_fsm_proc;

-- id comand state machine

    id_fsm_reg: process (CLK, nRESET)
    begin
        if nRESET ='0' then
            ID_CURRENT_STATE <= IDLE;
        elsif CLK'event and CLK='1'
        then ID_CURRENT_STATE <= ID_NEXT_STATE;
        case ID_CURRENT_STATE is
            when IDLE => send_data_req_id <= '0';
            when SEND => send_data_req_id <= '1';
        end case;
        end if;
    end process id_fsm_reg;

    id_fsm_proc: process (ID_CURRENT_STATE, ASK_ID, SENT_BYTES)
    begin
        ID_NEXT_STATE<=ID_CURRENT_STATE;
        case ID_CURRENT_STATE is
            when IDLE => if (ask_id = '1') then
                            ID_NEXT_STATE <=SEND;
                         end if;
            when SEND => if (sent_bytes > 31) then
                            ID_NEXT_STATE <= IDLE;
                         end if;
        end case;
    end process id_fsm_proc;

-- send data to i2c state machine

    send_fsm_reg: process (CLK, nRESET)
    begin
        if nRESET ='0' then
            SEND_CURRENT_STATE <= IDLE;
            SENT_BYTES <= 1;
            PUSH <= '0';
        elsif CLK'event and CLK='1'
```

```vhdl
        then SEND_CURRENT_STATE <= SEND_NEXT_STATE;
        case SEND_CURRENT_STATE is
            when IDLE => SENT_BYTES <= 1;
                         PUSH <= '0';
            when LOAD => if (ask_id = '1') then
                            if (SENT_BYTES > 31) then
                                DATA_OUT <= (others => '0');
                            else DATA_OUT <= Char2SLV8(ID(SENT_BYTES));
                            end if;
                         elsif (ask_read = '1') then
                             if (SENT_BYTES = 1) then
                                 DATA_OUT <= (others => '0');
                                 DATA_OUT(7) <= DATA(9);
                                 DATA_OUT(6) <= DATA(8);
                             else
                                 DATA_OUT <= DATA(7 downto 0);
                             end if;
                         end if;
                         SENT_BYTES <= SENT_BYTES + 1;
                         PUSH <= '0';
            when SEND => PUSH <= '1';
        end case;
        end if;
    end process send_fsm_reg;

    send_fsm_proc: process (SEND_CURRENT_STATE, SEND_DATA)
    begin
        SEND_NEXT_STATE<=SEND_CURRENT_STATE;
        case SEND_CURRENT_STATE is
            when IDLE => if (SEND_DATA = '1') then
                            SEND_NEXT_STATE <= LOAD;
                         end if;
            when LOAD => if  (SEND_DATA = '0') then
                            SEND_NEXT_STATE <= IDLE;
                         else
                            SEND_NEXT_STATE <= SEND;
                         end if;
            when SEND => SEND_NEXT_STATE <= LOAD;
        end case;
    end process send_fsm_proc;

 -- read channel state machine

    read_fsm_reg: process (CLK, nRESET)
    begin
        if nRESET ='0' then
            READ_CURRENT_STATE <= IDLE;
            ticks <= 0;
            send_data_req_read <= '0';
            Enable <= (others => '1');
        elsif CLK'event and CLK='0'
        then READ_CURRENT_STATE <= READ_NEXT_STATE;
        case READ_CURRENT_STATE is
            when IDLE => send_data_req_read <= '0';
                         ticks <= 0;
            when SETMUX => ABC(0) <= DATA_IN(0);
                           ABC(1) <= DATA_IN(1);
                           ABC(2) <= DATA_IN(2);
                           case DATA_IN(6 downto 3) is
                               when "0000" => Enable <= "00001111110";
                               when "0001" => Enable <= "10001111101";
                               when "0010" => Enable <= "00101111011";
                               when "0011" => Enable <= "10101110111";
                               when "0100" => Enable <= "00011101111";
                               when "0101" => Enable <= "10011011111";
```

```
                              when "0110" => Enable <= "00110111111";
--                            when "0111" => Enable <= "11101111111";
--                            when "1000" => Enable <= "11011111111";
--                            when "1001" => Enable <= "10111111111";
--                            when "1010" => Enable <= "01111111111";
                              when others => Enable <= "00001111111";
                         end case;
          when WAIT_ADC =>  if (ticks < 200) then
                              ticks <= ticks + 1;
                            end if;
                            if (ticks = 200) then
                              if (OTR_ADC = '1') then
                              DATA <= (others => '1');
                              else
                              DATA <= DATA_ADC;
                              end if;
                            end if;
          when SEND => send_data_req_read <= '1';
      end case;
      end if;
  end process read_fsm_reg;

  read_fsm_proc: process (READ_CURRENT_STATE, ASK_READ, NEW_DATA, ticks, SENT_BYTES)
  begin
      READ_NEXT_STATE<=READ_CURRENT_STATE;
      case READ_CURRENT_STATE is
          when IDLE => if (ask_read = '1') then
                          if (new_data = '1') then
                          READ_NEXT_STATE <= SETMUX;
                          end if;
                       end if;
          when SETMUX => READ_NEXT_STATE <= WAIT_ADC;
          when WAIT_ADC => if (ticks = 200) then
                              READ_NEXT_STATE <= SEND;
                           end if;
          when SEND => if (SENT_BYTES > 1) then
                       READ_NEXT_STATE <= IDLE;
                       end if;
      end case;
  end process read_fsm_proc;

end behave;
```

```
--I2C slave controller
--When a byte of data is written, an active-high pulse is asserted in signal new_data_available for a clock cycle,
--while data can be read in signal data_out
--Data to send in read acccesses is taken from a fifo. To push data in this fifo, use signal PUSH and the bus 'data_in'

library IEEE;
use IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.ALL;
use ieee.std_logic_unsigned.all;

entity slave_i2c is
   PORT(
        DEVICE : in STD_LOGIC_VECTOR (6 downto 0); -- Slave address
        CLK,nRESET : in std_logic;
        --prova : out std_logic;
        pinSCL :IN STD_LOGIC;
        pinSDA :INOUT STD_LOGIC;
        nRE : out std_logic;      --1 if SDA is acting as an output or 0 otherwise.
        nreset_fifo : in std_logic; --to erase all data stored in the fifo
        New_data_available : OUT STD_LOGIC; --positive pulse when new data is written to this device
        Data_out : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);    --last data written to this device
        Data_in : in STD_LOGIC_VECTOR(7 DOWNTO 0);  --data to send in the following read accesses
        PUSH : in STD_LOGIC
        );
END SLAVE_I2C;

architecture behav of SLAVE_I2C is
   --tmr register declaration to store state signals
   component tmr_state is
        generic (N: integer:=2); -- (N>=2)
        port(
            clk : in  std_logic;
            enable: in std_logic;
            nReset: in std_logic; -- a reset makes all outputs equal to 0, except LSB (for one hot encoding, 000..001 should be the idle state)
            D: in std_logic_vector(N-1 downto 0);
            Q: out std_logic_vector(N-1 downto 0)
            );
   end component;
   component tmr_reg is
        generic (N: integer:=2); -- (N>=2)
        port(
            clk : in  std_logic;
            enable: in std_logic;
            nReset: in std_logic;
            D: in std_logic_vector(N-1 downto 0);
            Q: out std_logic_vector(N-1 downto 0)
            );
   end component;
   component TMR_flipflop is
        port(
            clk : in  std_logic;
            enable: in std_logic;
            nReset: in std_logic;
            set: in std_logic;
            D: in std_logic;
            Q: out std_logic
            );
```

```
    end component;
    constant Vcc: std_logic:='1';
    constant GND: std_logic:='0';
    signal nRE_a: std_logic;
    --Signals for syncronization purposes
    signal pinSDA_ntrst,pinSCL_ntrst,pinSDA0,pinSCL0,sync_pinSCL, sync_pinSDA : std_logic;
    --start and stop condition related signals
    constant SC_SDA_high:std_logic_vector(1 downto 0):="01";
    constant SC_SDA_low: std_logic_vector(1 downto 0):="10";
    --type state_StartStop_conditions is (SC_SDA_high,SC_SDA_low);
    signal state_startstop, next_state_startstop:std_logic_vector(1 downto 0);--: state_StartStop_conditions;
    signal start_condition, stop_condition, next_start_condition, next_stop_condition: std_logic; --a pulse when a start or stop condition is detected
    --main state machine
    constant CO_idle: std_logic_vector( 4 downto 0):="00001";
    constant CO_addressing:std_logic_vector(4 downto 0):="00010";
    constant CO_write:std_logic_vector(4 downto 0):="00100";
    constant CO_not_addressed:std_logic_vector(4 downto 0):="01000";
    constant CO_read:std_logic_vector(4 downto 0):="10000";
    --type state_slave_control  is (CO_idle,CO_addressing,CO_read,CO_write,CO_not_addressed);
    signal state_control, next_state_control:std_logic_vector(4 downto 0 );--: state_slave_control;
    signal enable_read, enable_write: std_logic;
    --signals for the address cycle
    constant AD_idle: std_logic_vector(5 downto 0):="000001";
    constant AD_wait_SCL_low: std_logic_vector(5 downto 0):="000010";
    constant AD_wait_SCL_high: std_logic_vector(5 downto 0):="000100";
    constant AD_wait_beginACK: std_logic_vector(5 downto 0):="001000";
    constant AD_ACK: std_logic_vector(5 downto 0):="010000";
    constant AD_ACK_end: std_logic_vector(5 downto 0):="100000";
    --type state_addressing is(AD_idle,AD_wait_SCL_low,AD_wait_SCL_high,AD_wait_begin
ACK_end);
    signal state_addr, next_state_addr:std_logic_vector(5 downto 0);--state_addressing;
    signal end_addressing,next_end_addressing:std_logic;
    alias begin_addressing is start_condition;--addressing starts just after an start condition
    signal shift_register_addr,next_shift_register_addr: std_logic_vector(7 downto 0);
    alias RnW is shift_register_addr(0); alias addressed_device is shift_register_addr(7 downto 1);
    signal SDA_addr,nRE_addr: std_logic;
    signal nRE_preread0,nRE_preread1,nRE_preread2: std_logic;
    signal next_reset_addressed_bits, next_enable_increment_addressed_bits: std_logic;
    --write cycle related signals
    constant W_idle:std_logic_vector(4 downto 0):="00001";
    constant W_wait_SCL_high:std_logic_vector(4 downto 0):="00010";
    constant W_wait_SCL_low:std_logic_vector(4 downto 0):="00100";
    constant W_wait_ACK_begin:std_logic_vector(4 downto 0):="01000";
    constant W_wait_ACK_end:std_logic_vector(4 downto 0):="10000";
    --type state_write_cycle is(W_idle,W_wait_SCL_high,W_wait_SCL_low,W_wait_ACK_begi
d);
    signal state_write, next_state_write:std_logic_vector(4 downto 0); --state_write_cycle;
    signal shift_register_Write, next_shift_register_write: std_logic_vector(7 downto 0);
    signal next_reset_writen_bits, next_enable_increment_writen_bits: std_logic;
    signal next_new_data_Available:std_logic;
    signal SDA_write,nRE_write:std_logic;
```

```
   --read cycle signals
   constant R_idle: std_logic_vector(7 downto 0)              := "00000001";
   constant R_read_fifo_request: std_logic_vector(7 downto 0)  := "00000010";
   constant R_read_fifo: std_logic_vector(7 downto 0)          := "00000100";
   constant R_put_bit: std_logic_vector(7 downto 0)            := "00001000";
   constant R_wait_SCL_low: std_logic_vector(7 downto 0)       := "00010000";
   constant R_begin_ACK: std_logic_vector(7 downto 0)          := "00100000";
   constant R_end_ACK: std_logic_vector(7 downto 0)            := "01000000";
   constant wait_stop_condition: std_logic_vector(7 downto 0)  := "10000000";
   --type state_read_cycle is (R_idle,R_read_fifo_request,R_read_fifo, R_put_bit, R_wait_SCL_low,R_begin_ACK,R_end_ACK);
   signal state_read, next_state_read:std_logic_vector(7 downto 0);--: state_read_cycle;
   signal next_reset_read_bits, next_enable_increment_read_bits: std_logic;
   signal next_nrd_fifo: std_logic;
   signal shift_register_read, next_shift_register_read: std_logic_Vector(7 downto 0);
   signal SDA_read,nRE_read: std_logic;
   --counter for shifted bits
   --signal reset_addressed_bits, enable_increment_addressed_bits: std_logic;
   --signal reset_writen_bits, enable_increment_writen_bits:std_logic;
   --signal reset_read_bits, enable_increment_read_bits: std_logic;
   signal reset_shifted_bits, next_reset_shifted_bits: std_logic;
   signal next_enable_increment_shifted_bits, enable_increment_shifted_bits: std_logic;
   signal next_shifted_bits_counter,shifted_bits_counter: std_logic_vector(2 downto 0);
   --component definition  and fifo related signals
   COMPONENT TMR_slave_fifo_out IS     port( data : in std_logic_vector (7 downto 0); q : out
           std_logic_vector(7 downto 0);wrreq, rdreq, WClock, RClock :
           in std_logic;  FULL, EMPTY : out std_logic;  --ACLR : in std_logic,
           reset:in std_logic
           );
   end component;
   signal npush, nRd_fifo,fifo_empty: std_logic;
   signal data_to_send: std_logic_vector(7 downto 0);
begin

   --syncronization of I/O signals
   pinSDA_ntrst<='0' when pinSDA='0' else '1';
   pinSCL_ntrst<='0' when pinSCL='0' else '1';
   U7: tmr_flipflop port map(
       clk=>clk, nreset=>Vcc, set=>nreset, enable=>Vcc,
       D=>pinSDA_ntrst,Q=>pinSDA0
       );
   U8: tmr_flipflop port map(
       clk=>clk, nreset=>Vcc, set=>nreset, enable=>Vcc,
       D=>pinSDA0,Q=>sync_pinSDA
       );
   U9: tmr_flipflop port map(
       clk=>clk, nreset=>Vcc, set=>nreset, enable=>Vcc,
       D=>pinSCL_ntrst,Q=>pinSCL0
       );
```

```
    U10: tmr_flipflop port map(
        clk=>clk, nreset=>Vcc, set=>nreset, enable=>Vcc,
        D=>pinSCL0,Q=>sync_pinSCL
        );



    --start and stop conditions
    process(state_startstop,sync_pinSDA,sync_pinSCL)
    begin
        next_state_startstop<=state_startstop;
        next_start_condition<='0';next_stop_condition<='0';
        case state_startstop is
            when SC_SDA_low=>
            if sync_pinSDA/='0' then
                next_state_startstop<=SC_SDA_high;
                if sync_pinSCL/='0' then
                    next_stop_condition<='1';
                end if;
            end if;
            when SC_SDA_high=>
            if sync_pinSDA='0' then
                next_state_startstop<=SC_SDA_low;
                if sync_pinSCL/='0' then
                    next_start_condition<='1';
                end if;
            end if;
            when others=> --
        end case;
    end process;
    U2: tmr_state generic map(2)
    port map(
        clk =>clk,  enable=>Vcc,nReset=>nreset,
        D=>next_state_startstop,
        Q=>state_startstop
        );

    U99:tmr_flipflop port map(
        clk=>clk,enable=>Vcc,nReset=>nreset,set=>Vcc,
        D=>next_start_condition,Q=>start_condition);
    U90:tmr_flipflop port map(
        clk=>clk,enable=>Vcc,nReset=>nreset,set=>Vcc,
        D=>next_stop_condition,Q=>stop_condition);

    --main fsm
    process(state_control,start_condition,stop_condition,end_addressing,addressed_device,Rnw)
    begin
        next_state_control<=state_control;
        if state_control=CO_addressing and end_addressing='1' then
            if addressed_device/=device then next_state_control<=CO_not_addressed;
            elsif Rnw='0' then next_state_control<=CO_write;
            else next_state_control<=CO_read;
            end if;
        end if;
        if start_condition='1' then
            next_state_control<=CO_addressing;
        elsif stop_condition='1' then
            next_State_control<=CO_idle;
        end if;
    end process;

    U3: tmr_state generic map(5)
    port map(
```

```
        clk =>clk,  enable=>Vcc,nReset=>nreset,
        D=>next_state_control,
        Q=>state_control
        );


    process(clk,nreset)
    begin
        if nreset='0' then
            --state_control<=CO_idle;
            enable_read<='0';   enable_write<='0';
        elsif clk'event and clk='1' then
            --state_control<=next_state_control;
            enable_read<='0';   enable_write<='0';
            case state_control is
                when CO_idle=>
                when CO_addressing=>
                when CO_read=>
                enable_read<='1';
                when CO_write=>
                enable_write<='1';
                when CO_not_addressed=>
                when others=>
            end case;
        end if;
    end process;

    --address cycle
    process(state_addr,begin_addressing,sync_pinSCL,sync_pinSDA,shifted_bits_counter,shift_register_addr)
    begin
        next_state_addr<=state_addr;
        next_shift_register_addr<=shift_register_addr;
        next_end_addressing<='0';   next_reset_addressed_bits<='0';next_enable_increment_addressed_bits<='0';
        case state_addr is
            when AD_idle=>
            if begin_addressing='1' then
                next_state_addr<=AD_wait_SCL_low;
                next_reset_addressed_bits<='1';
            end if;
            when AD_wait_SCL_low=>
            if sync_pinSCL='0' then
                next_state_addr<=AD_wait_SCL_high ;
            end if;
            when AD_wait_SCL_high=>
            if sync_pinSCL/='0' then
                next_enable_increment_addressed_bits<='1';
                next_shift_register_addr<=shift_register_addr(6 downto 0)&sync_pinSDA;
                if shifted_bits_counter/=7 then
                    next_state_addr<=AD_wait_SCL_low ;
                else
                    next_state_addr<=AD_wait_beginACK;
                end if;
            end if;
            when AD_wait_beginACK=>
            if sync_pinSCL='0' then
                next_state_addr<=AD_ACK;
            end if;
            when AD_ACK=>
            if sync_pinSCL/='0' then
                next_state_addr<=AD_ACK_end;
            end if;
            when AD_ACK_end=>
```

```
                if sync_pinSCL='0' then
                    next_state_addr<=AD_idle;
                    next_end_addressing<='1';
                end if;
                when others=>
            end case;
        end process;
        --prova
        process(clk)
        begin
            if clk'event and clk='1' then
                nRE_preread0<=end_addressing;
                nRE_preread1<=nRE_preread0;
                nRE_preread2<=nRE_preread1;
            end if;
        end process;

        U4: tmr_state generic map(6)
        port map(
            clk =>clk,  enable=>Vcc,nReset=>nreset,
            D=>next_state_addr,
            Q=>state_addr
            );

        process(clk,nreset)
        begin
            if nreset='0' then
                --state_addr<=AD_idle;
                end_addressing<='0';
                --reset_addressed_bits<='0'; enable_increment_addressed_bits<='0';
                --  shift_register_addr<=(others=>'0');
                SDA_addr<='1';
                nRE_addr<='0';
            elsif clk'event and clk='1' then
                --state_addr<=next_state_addr;
                end_addressing<=next_end_addressing;
                --  shift_register_addr<=next_shift_register_addr;
                --  reset_addressed_bits<=next_reset_addressed_bits;
                -- enable_increment_addressed_bits<=next_enable_increment_addressed_bits;
                if (state_addr=AD_ACK or state_addr=AD_ACK_end) and addressed_device=device then
                    SDA_addr<='0';
                    nRE_addr<='1';
                else SDA_addr<='1'; nRE_addr<='0';
                end if;
            end if;
        end process;
        --counter for shifted bits
        next_reset_shifted_bits<=not(next_reset_read_bits or next_reset_addressed_bits or next_reset_writen_bits);
        next_enable_increment_shifted_bits<=next_enable_increment_read_bits or next_enable_increment_writen_bits or next_enable_increment_addressed_bits;
        next_shifted_bits_counter<=shifted_bits_counter+1;
        U12: TMR_flipflop port map(clk=>clk, nreset=>nreset, enable=>Vcc,set=>Vcc,D=>next_reset_shifted_bits,Q=>reset_shifted_bits);
        U15: TMR_flipflop port map(clk=>clk, nreset=>nreset, enable=>Vcc,set=>Vcc,D=>next_enable_increment_shifted_bits,Q=> enable_increment_shifted_bits);
        U20: TMR_reg generic map(3) port map(
            clk=>clk, nreset=>reset_shifted_bits, enable=> enable_increment_shifted_bits,
            D=>next_shifted_bits_counter, Q=>shifted_bits_counter);
```

```


  --write cycle
  process(state_write,sync_pinSDA,sync_pinSCL,shifted_bits_counter,enable_write,shift_register_write)
  begin
      next_state_write<=state_Write;
      next_new_data_available<='0';
      next_reset_writen_bits<='0';next_enable_increment_writen_bits<='0';
      next_shift_Register_write<=shift_register_write;
      if enable_write='0' then next_state_write<=W_idle;
      else
          case state_write is
              when W_idle=>
              if enable_write='1' then
                  next_state_write<=W_wait_SCL_high;
                  next_reset_writen_bits<='1';
              end if;
              when W_wait_SCL_high =>
              if sync_pinSCL/='0' then
                  next_state_write<=W_wait_SCL_low;
                  next_shift_register_write<=shift_register_write(6 downto 0)&sync_pinSDA;
              end if;
              when W_wait_SCL_low=>
              if sync_pinSCL='0' and shifted_bits_counter=7 then
                  next_state_write<=W_wait_ACK_begin;

              elsif sync_pinSCL='0' then
                  next_state_write<=W_wait_SCL_high;
                  next_enable_increment_writen_bits<='1';
              end if;
              when W_wait_ACK_begin=>
              if sync_pinSCL/='0' then
                  next_state_write<=W_wait_ACK_end;
              end if;
              when W_wait_ACK_end=>
              if sync_pinSCL='0' then
                  next_state_write<=W_wait_SCL_high;
                  next_reset_writen_bits<='1';
                  next_new_data_available<='1';
              end if;
              when others=>
          end case;
      end if;
  end process;
  U5: tmr_state generic map(5)
  port map(
      clk =>clk,  enable=>Vcc,nReset=>nreset,
      D=>next_state_write,
      Q=>state_write
      );

  U22: tmr_flipflop port map(
      clk=>clk, nreset=>nreset, enable=>Vcc,set=>Vcc,
      D=>next_new_data_available, Q=>new_data_available);

  process(clk,nreset)
  begin
      if nreset='0' then
          --state_write<=W_idle;
          --  new_data_available<='0';
          --  enable_increment_writen_bits<='0';
          --  reset_writen_Bits<='0';
```

```
            data_out<=(OTHERS=>'0');
            --shift_register_write<=(others=>'0');
            sda_write<='1';
            nRE_write<='0';
        elsif clk'event and clk='1' then
            --state_write<=next_state_Write;
            --  new_data_Available<=next_new_data_available;
            --shift_Register_write<=next_shift_register_write;
            --  reset_writen_bits<=next_reset_writen_bits;
            --enable_increment_writen_bits<=next_enable_increment_writen_bits;
            if state_write=W_wait_ACK_end then
                data_out<=shift_register_write;
            end if;
            if state_write=W_wait_ACK_end or state_write=W_wait_ACK_begin then
                SDA_write<='0';
                nRE_write<='1';
            else SDA_write<='1';nRE_write<='0';
            end if;
        end if;
    end process;
    --fifo where data to be send is stored
    U1: TMR_slave_fifo_out PORT MAP (
        data     => data_in,
        wrreq    => nPUSH,
        rdreq    => nRd_fifo,
        Wclock   => clk,
        Rclock=>clk,
        q    => data_to_send,
        full     => open,
        empty    => fifo_empty,
        reset=>nreset_fifo
        );
    nPUSH<=not push;
    --read cycle

    process(state_read,sync_pinSCL,sync_pinSDA,shifted_bits_counter,enable_read,shift_register_read,data_to_send)
    begin
        next_state_read<=state_read;
        next_nRd_Fifo<='1'; next_reset_read_bits<='0'; next_enable_increment_read_bits<='0';
        next_shift_register_read<=shift_register_read;
        if enable_read='0' then
            next_state_read<=R_idle;
        else
            case state_read is
                when R_idle=>
                if enable_read='1' then
                    next_state_read<=R_read_fifo_request;
                    next_nrd_fifo<='0';
                end if;
                when R_read_fifo_request=>
                next_state_read<=R_read_fifo;
                next_reset_read_bits<='1';
                when R_read_fifo=>
                next_state_read<=R_put_bit;
                next_shift_register_read<=data_to_send;
                when R_put_bit=>
                if sync_pinSCL/='0' then
                    next_shift_register_read<=shift_register_read(6 downto 0)&'0';
                    next_state_read<=R_wait_SCL_low;
                end if;
```

```
                when R_wait_SCL_low=>
                if sync_pinSCL='0' and shifted_bits_counter/=7 then
                    next_enable_increment_read_bits<='1';
                    next_state_read<=R_put_bit;
                elsif sync_pinSCL='0' then
                    next_state_read<=R_begin_ACK;
                end if;
                when R_begin_ACK=>
                if sync_pinSCL/='0' then
                    next_state_read<=R_end_ACK;
                end if;
                when R_end_ACK=>
                if sync_pinSCL='0' then
                    next_State_read<=R_read_fifo_request;
                    next_nrd_fifo<='0';
                    if sync_pinSDA/='0' then
                        next_State_read<=wait_stop_condition;
                        next_nrd_fifo<='1';
                    end if;
                end if;
                when others=>
            end case;
        end if;
    end process;
    U6: tmr_state generic map(8)
    port map(
        clk =>clk,  enable=>Vcc,nReset=>nreset,
        D=>next_state_read,
        Q=>state_read
        );

    process(clk, nreset)
    begin
        if nreset='0' then
            --state_read<=R_idle;
            nRd_Fifo<='1';--reset_read_bits<='0';enable_increment_read_bits<='0';
            --shift_register_read<=(others=>'0');
            sda_read<='1';
            nRE_read<='0';
        elsif clk'event and clk='1' then
            if fifo_empty='1' then
                nRd_fifo<='1';
            else
                nRd_fifo<=next_nRd_Fifo;
            end if;
            --  reset_read_bits<=next_reset_read_bits;enable_increment_read_bits<=next_enable_increment_read_bits;
            --state_read<=next_state_read;
            --  shift_register_read<=next_shift_register_read;
            nRE_read<='1';
            case state_read is
                when R_idle=> SDA_read<='1';nRE_read<='0';
                when R_read_fifo_request=>
                when R_read_fifo=>
                when R_put_bit=>SDA_read<=shift_register_read(7);
                when R_wait_SCL_low=>--
                when R_begin_ACK=>SDA_read<='1';nRE_read<='0';
                when R_end_ACK=>nRE_read<='0';
                when wait_stop_condition=>nRE_read<='0';
                when others=>
            end case;
        end if;
    end process;
    --shift registers
```

```
   U11: TMR_reg generic map (8) port map(
       clk=>clk, nreset=>nreset, enable=>Vcc,
       D=>next_shift_register_read,Q=>shift_register_read
       );
   U18: TMR_reg generic map (8) port map(
       clk=>clk, nreset=>nreset, enable=>Vcc,
       D=>next_shift_register_write,Q=>shift_register_write
       );
   U19: TMR_reg generic map (8) port map(
       clk=>clk, nreset=>nreset, enable=>Vcc,
       D=>next_shift_register_addr,Q=>shift_register_addr
       );


   --SDA value
   pinSDA<='0' when SDA_addr='0' or SDA_read='0' or SDA_write='0' else 'Z';
   --pinSDA<='0' when SDA_addr='0' or SDA_read='0' or SDA_write='0' else '1' when  nRE_a='1' else'Z';
   nRE<=nRE_a;
   nRE_a<=nRE_addr or nRE_read or nRE_write or nRE_preread0 or nRE_preread1 or nRE_preread2;


   --prova<='0' when SDA_addr='0' or SDA_read='0' or SDA_write='0' else '1';
END behav;
```

APPENDIX 3: SOFTWARE

main.cpp

```
/////////////////////////////////////////////////////////////////////////////
// Name:        main.cpp
// Purpose:     CBEmu main window
// Author:      Albert Comerma
// Licence:     wxWindows licence
/////////////////////////////////////////////////////////////////////////////

// ============================================================================
// declarations
// ============================================================================

// ----------------------------------------------------------------------------
// headers
// ----------------------------------------------------------------------------

// For compilers that support precompilation, includes "wx/wx.h".
#include "wx/wxprec.h"

#ifdef __BORLANDC__
    #pragma hdrstop
#endif

// for all others, include the necessary headers (this file is usually all you
// need because it includes almost all "standard" wxWindows headers)
#ifndef WX_PRECOMP
    #include "wx/wx.h"
#endif

#include <wx/image.h>
#include <wx/filename.h>
#include <wx/textctrl.h>
#include <wx/datetime.h>
#include <wx/progdlg.h>
#include <wx/stream.h>
#include <time.h>
#include <windows.h>
#include "MinilabCtrl.h"
#include "I2C.h"
#include "Address_window.h"
#include "Channel_window.h"
#include "main.h"


// ----------------------------------------------------------------------------
// resources
// ----------------------------------------------------------------------------

// the application icon (under Windows and OS/2 it is in resources)
#if defined(__WXGTK__) || defined(__WXMOTIF__) || defined(__WXMAC__) || defined(__WXMGL__) || defined(__WXX11__)
    #include "mondrian.xpm"
#endif

#ifdef USE_STATIC_BITMAP
    #include "green.xpm"
    #include "red.xpm"
```

```
#endif // USE_STATIC_BITMAP

static const int BITMAP_SIZE_X = 32;
static const int BITMAP_SIZE_Y = 15;

// ----------------------------------------------------------------------------
// private classes
// ----------------------------------------------------------------------------

// Define a new application type, each program should derive a class from wxApp
class MyApp : public wxApp
{
public:
    // override base class virtuals
    // ----------------------------

    // this one is called on application startup and is a good place for the app
    // initialization (doing it here and not in the ctor allows to have an error
    // return: if OnInit() returns false, the application terminates)
    virtual bool OnInit();
};

// Forward declaration
class MyFrame;

// A custom status bar which contains controls, icons &c
class MyStatusBar : public wxStatusBar
{
public:
    MyStatusBar(MyFrame *parent);
    virtual ~MyStatusBar();

    void UpdateClock();

    // event handlers
    void OnTimer(wxTimerEvent& event) { UpdateClock(); }
    void OnSize(wxSizeEvent& event);
    void UpdateTimeoutStatus(const bool isOK);
    MyFrame* GetParent() {return m_Parent;}

private:
    wxBitmap CreateBitmapForButton(bool on = FALSE);

    enum
    {
        Field_Text,
        Field_TIMEOUT,
        Field_Clock,
        Field_Max
    };

    MyFrame *m_Parent;
    wxTimer m_timer;

#ifdef USE_STATIC_BITMAP
    wxStaticBitmap *m_statbmp;
#else
    wxBitmapButton *m_statbmp;
#endif

```

```
    DECLARE_EVENT_TABLE()
};

// Define a new frame type: this is going to be our main frame
class MyFrame : public wxFrame
{
public:
    // ctor(s)
    MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size,
            long style = wxDEFAULT_FRAME_STYLE);

    wxString myAppName;     // This is my application name
    wxString myFilename;    // This is my filename
    wxTextCtrl *textW;      // Stores the log of commands/responses
    wxLogTextCtrl *log;
    MyStatusBar *m_statbar;

    void OnFileSave(wxCommandEvent& event);

    // event handlers (these functions should _not_ be virtual)
    void OnQuit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);
    void OnLicense(wxCommandEvent& event);

    //I2C

    I2Cctrl I2C;
    void OnI2C_POW(wxCommandEvent& event);
    void OnI2C_SCL_test(wxCommandEvent& event);
    void OnI2C_SDA_test(wxCommandEvent& event);

    //Operations
    void On_Search(wxCommandEvent& event);
    void OnSetAddr(wxCommandEvent& event);
    void OnAddrClose();
    void OnAskId(wxCommandEvent& event);
    void OnSendPU(wxCommandEvent& event);
    void OnSendPD(wxCommandEvent& event);
    void OnSendPUFIRST(wxCommandEvent& event);

    Address_window &GetAddress_window() {return m_Address_window;};
    Channel_window &GetChannel_window() {return m_Channel_window;};

    //Minilab

    MinilabCtrl Minilab;
    void OnReadChannels(wxCommandEvent& event);
    void OnStartTempTest(wxCommandEvent& event);
    void OnFPGAReadChan(wxCommandEvent& event);
    void OnFPGAMon(wxCommandEvent& event);
    void OnFPGAALL(wxCommandEvent& event);
    void OnFPGAReadChanClose();

private:

    Address_window m_Address_window;    //Child
    Channel_window m_Channel_window;

    void ShowTimeoutStatus();
    wxString GetAboutMessage();

    // any class wishing to process wxWindows events must use this macro
    DECLARE_EVENT_TABLE()
};

```

```
// ----------------------------------------------------------------------------
// constants
// ----------------------------------------------------------------------------

// IDs for the controls and the menu commands
enum
{
    // menu items
    Minimal_Quit = 1,
    wxM_FILESAVE,

    //I2C

    wxM_I2C_POW,
    wxM_I2C_SCLt,
    wxM_I2C_SDAt,

    //Operations

    wxM_SEARCH,
    wxM_SET_ADDR,
    wxM_ASK_ID,
    wxM_SEND_PU,
    wxM_SEND_PD,
    wxM_SEND_PUF,

    //Minilab

    wxM_Read_Channels,
    wxM_Temp_Test,
    wxM_FPGA_Read_Chan,
    wxM_FPGA_DUMP,
    wxM_FPGA_ALL,
    // it is important for the id corresponding to the "About" command to have
    // this standard value as otherwise it won't be handled properly under Mac
    // (where it is special and put into the "Apple" menu)
    Minimal_About = wxID_ABOUT,
    wxM_License
};

// ----------------------------------------------------------------------------
// event tables and other macros for wxWindows
// ----------------------------------------------------------------------------

// the event tables connect the wxWindows events with the functions (event
// handlers) which process them. It can be also done at run-time, but for the
// simple menu events like this the static method is much simpler.
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(Minimal_Quit,  MyFrame::OnQuit)
    EVT_MENU(Minimal_About, MyFrame::OnAbout)
    EVT_MENU(wxM_License, MyFrame::OnLicense)
    EVT_MENU(wxM_FILESAVE, MyFrame::OnFileSave)

    //I2C

    EVT_MENU( wxM_I2C_POW, MyFrame::OnI2C_POW)
    EVT_MENU( wxM_I2C_SCLt, MyFrame::OnI2C_SCL_test)
    EVT_MENU( wxM_I2C_SDAt, MyFrame::OnI2C_SDA_test)
```

```

    //Operations

    EVT_MENU( wxM_SEARCH, MyFrame::On_Search)
    EVT_MENU( wxM_SET_ADDR, MyFrame::OnSetAddr)
    EVT_MENU( wxM_ASK_ID, MyFrame::OnAskId)
    EVT_MENU( wxM_SEND_PU, MyFrame::OnSendPU)
    EVT_MENU( wxM_SEND_PD, MyFrame::OnSendPD)
    EVT_MENU( wxM_SEND_PUF, MyFrame::OnSendPUFIRST)

    //Minilab

    EVT_MENU( wxM_Read_Channels, MyFrame::OnReadChannels)
    EVT_MENU( wxM_Temp_Test, MyFrame::OnStartTempTest)
    EVT_MENU( wxM_FPGA_Read_Chan, MyFrame::OnFPGAReadChan)
    EVT_MENU( wxM_FPGA_DUMP, MyFrame::OnFPGAMon)
    EVT_MENU( wxM_FPGA_ALL, MyFrame::OnFPGAALL)

END_EVENT_TABLE()

BEGIN_EVENT_TABLE(MyStatusBar, wxStatusBar)
    EVT_SIZE(MyStatusBar::OnSize)
    EVT_TIMER(-1, MyStatusBar::OnTimer)
END_EVENT_TABLE()

// Create a new application object: this macro will allow wxWindows to create
// the application object during program execution (it's better than using a
// static object for many reasons) and also declares the accessor function
// wxGetApp() which will return the reference of the right type (i.e. MyApp and
// not wxApp)
IMPLEMENT_APP(MyApp)

// ============================================================================
// implementation
// ============================================================================

    int address = 85;
    bool address_window_opened = false;
    bool channel_window_opened = false;

// ----------------------------------------------------------------------------
// the application class
// ----------------------------------------------------------------------------

// 'Main program' equivalent: the program execution "starts" here
bool MyApp::OnInit()
{
    // create the main application window
    MyFrame *frame = new MyFrame(_T("CBEmu v0.01"),
                                 wxPoint(50, 50), wxSize(450, 340));

    // and show it (the frames, unlike simple controls, are not shown when
    // created initially)
    frame->Show(TRUE);

    // Set LHCb icon
    ::wxInitAllImageHandlers();
    wxIcon myIcon ;
```

```
    wxFileName fn ( argv[0] );
    myIcon.CopyFromBitmap(wxBitmap(wxImage(fn.GetPathWithSep() + wxT("CBEmu.ico"))));
    frame->SetIcon(myIcon);
    frame->GetAddress_window().SetIcon(myIcon);
    frame->GetChannel_window().SetIcon(myIcon);
    SetTopWindow (frame);

    // success: wxApp::OnRun() will be called which will enter the main message
    // loop and the application will run. If we returned FALSE here, the
    // application would exit immediately.
    return TRUE;
}

// ----------------------------------------------------------------------------
// main frame
// ----------------------------------------------------------------------------

// frame constructor
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size, long style)
       : wxFrame(NULL, -1, title, pos, size, style)
{

    wxString msg;

    myAppName = title;

    // set the frame icon
    SetIcon(wxICON(mondrian));

    // create a child control here
    textW = new wxTextCtrl (this, -1, wxEmptyString, wxDefaultPosition, wxDefaultSize,
                        wxTE_MULTILINE|wxTE_READONLY|wxTE_RICH);
    log = new wxLogTextCtrl(textW);
    log->SetActiveTarget(log);       // Logs to the frame...
    log->SetTimestamp(NULL);

    wxLogMessage(myAppName.c_str());
    wxLogMessage(GetAboutMessage());
    I2C.init();

#if wxUSE_MENUS
    // create a menu bar
    wxMenu *menuFile = new wxMenu;

    // the "About" item should be in the help menu
    wxMenu *helpMenu = new wxMenu;
    helpMenu->Append(wxM_License, _T("&Licence\tAlt-L"), _T("Show license file"));
    helpMenu->Append(Minimal_About, _T("&About...\tF1"), _T("Show about dialog"));

    // create a File menu bar

    menuFile->Append(wxM_FILESAVE, wxT("&Save log\tAlt-S"), wxT("Saves log text") );
    menuFile->Append(Minimal_Quit, _T("E&xit\tAlt-X"), _T("Quit this program"));

    // create an I2C menu bar
```

```
    wxMenu *menuI2C = new wxMenu;
    menuI2C->Append(wxM_I2C_POW, wxT("Power Up"), wxT("Powers Up transceivers throug LPT") );
    menuI2C->Append(wxM_I2C_SCLt, wxT("SCL"), wxT("Test I2C Bus SCL signal") );
    menuI2C->Append(wxM_I2C_SDAt, wxT("SDA"), wxT("Test I2C Bus SDA signal") );

    // create an Operations menu bar
    wxMenu *menuOperate = new wxMenu;
    menuOperate->Append(wxM_SEARCH, wxT("Search for devices"), wxT("Searches for devices attached to I2C") );
    menuOperate->Append(wxM_SET_ADDR, wxT("Set Address"), wxT("Configures the I2C address to use") );
    menuOperate->Append(wxM_ASK_ID, wxT("Get Board ID"), wxT("Returns the result of asking board's ID string") );
    menuOperate->Append(wxM_SEND_PU, wxT("Send Power UP"), wxT("Sends power up command to device"));
    menuOperate->Append(wxM_SEND_PD, wxT("Send Power DOWN"), wxT("Sends power down command to device"));
    menuOperate->Append(wxM_SEND_PUF, wxT("Send Power UP FIRST"), wxT("Sends power up first output command to device"));

    // create a Tests menu bar

    wxMenu *menuTests = new wxMenu;
    menuTests->Append(wxM_Read_Channels, wxT("Read Minilab"), wxT("Reads all eight minilab analog channels") );
    menuTests->Append(wxM_Temp_Test, wxT("Start Temperature test"), wxT("Starts a cycle of reading temperature probes every 1 sec") );
    menuTests->Append(wxM_FPGA_Read_Chan, wxT("FPGA read Channel"), wxT("Reads an analog channel through FPGA") );
    menuTests->Append(wxM_FPGA_DUMP, wxT("FPGA data Dump"), wxT("Reads some monitoring data") );
    menuTests->Append(wxM_FPGA_ALL, wxT("FPGA Analog channels Dump"), wxT("Reads all the analog channels voltage") );

    // now append the freshly created menu to the menu bar...
    wxMenuBar *menuBar = new wxMenuBar();
    menuBar->Append(menuFile, _T("&File"));
    menuBar->Append(menuI2C, _T("&I2C test"));
    menuBar->Append(menuOperate, _T("&Operations"));
    menuBar->Append(menuTests, _T("&Tests"));
    menuBar->Append(helpMenu, _T("&Help"));

    // ... and attach this menu bar to the frame
    SetMenuBar(menuBar);

#endif // wxUSE_MENUS

#if wxUSE_STATUSBAR
    // create a status bar just for fun (by default with 1 pane only)
    m_statbar = new MyStatusBar(this);
    SetStatusBar(m_statbar);
    GetStatusBar()->Show();
    PositionStatusBar();
    msg.Printf(_T("Welcome to %s."), myAppName.c_str());
    SetStatusText(msg);
#endif // wxUSE_STATUSBAR
}


// event handlers

void MyFrame::OnQuit(wxCommandEvent& WXUNUSED(event))
```

```
{
    // TRUE is to force the frame to close
    Close(TRUE);
}

void MyFrame::OnAbout(wxCommandEvent& WXUNUSED(event))
{
    wxString msg;
    msg.Printf( _T("(C-) Albert Comerma, 2005, based on Jordi Riera's LPT Test Beam DAQ.\n"
                   "Using Yariv Kaplan's WINIO driver for port access"));

    wxMessageBox(msg, _T("About"), wxOK | wxICON_INFORMATION, this);
}

void MyFrame::OnLicense(wxCommandEvent& WXUNUSED(event))
{
// System exec to call for notepad file of license
   system("notepad GPL.txt");
}

wxString MyFrame::GetAboutMessage()
{
    wxString msg;
    msg.Printf( _T("(C-) Albert Comerma, 2005, based on Jordi Riera's LPT Test Beam DAQ.\n"
                    "Control Board Emulation Software for LHCb's SPD Low Voltage Power Supply "
                    "Regulators Board developement. "
                    "%s comes with ABSOLUTELY NO WARRANTY. "
                    "This is free software, and you are welcome to redistribute it under certain "
                    "conditions. Choose 'License' option in 'Help' menu for details on warranty "
                    "and license conditions."), myAppName.c_str());
    return msg;
}

void MyFrame::OnI2C_POW (wxCommandEvent & event)
{
   I2C.Power_transceivers();
   wxLogMessage("INFO> LVDS transceivers powered.");
}

void MyFrame::OnI2C_SCL_test (wxCommandEvent & event)
{
   wxLogMessage("INFO> SCL Cycling...");
   for(int i=0;i<50000;i++)   I2C.SCL_cycle();
}

void MyFrame::OnI2C_SDA_test (wxCommandEvent & event)
{
   wxLogMessage("INFO> SDA Cycling...");
   for(int i=0;i<50000;i++)   I2C.SDA_cycle();
}

void MyFrame::On_Search (wxCommandEvent & event)
{
   char* data;
   char data_byte;
   I2C.Power_transceivers();
   data=I2C.check_for_boards();
   data_byte=*data;
   if(!data_byte) wxLogMessage("ERROR> No boards present");
   else wxLogMessage("INFO> Found %i boards",data_byte);
```

```
   data++;
   data_byte=*data;
   while(data_byte!='\0')
      {
      wxLogMessage("INFO> Address: %i",data_byte);
      data++;
      data_byte=*data;
      }
}

void MyFrame::OnFPGAReadChan (wxCommandEvent & event)
{
   GetMenuBar()->EnableTop(0, FALSE);
   GetMenuBar()->EnableTop(1, FALSE);
   GetMenuBar()->EnableTop(2, FALSE);
   GetMenuBar()->EnableTop(3, FALSE);
   m_Channel_window.Show(TRUE);
   channel_window_opened = true;
}

void MyFrame::OnSetAddr (wxCommandEvent & event)
{
   GetMenuBar()->EnableTop(0, FALSE);
   GetMenuBar()->EnableTop(1, FALSE);
   GetMenuBar()->EnableTop(2, FALSE);
   GetMenuBar()->EnableTop(3, FALSE);
   m_Address_window.Show(TRUE);
   address_window_opened = true;
}

void MyFrame::OnFPGAMon(wxCommandEvent & event)
{
   unsigned int value=0;
   double resolution=1.955034, k=1.25;
   I2C.write_command_withargs(address,READ_CHAN_COMMAND,6);
   value=I2C.read_10bits(address);
   wxLogMessage("----------------------------------------------------------------------------------------------------------------------");
   wxLogMessage("   READING FPGA MONITORING DATA FOR ALL CHANNELS...");
   wxLogMessage("----------------------------------------------------------------------------------------------------------------------");
   wxLogMessage("INFO> START OF DATA DUMP...");
   if(value==1023) wxLogMessage("INFO> Current of +1'65A VFE1 Out Of Range");
   else wxLogMessage("INFO> Current of +1'65A VFE1 %.1f mA", value*resolution*k);
   I2C.write_command_withargs(address,READ_CHAN_COMMAND,7);
   value=I2C.read_10bits(address);
   if(value==1023) wxLogMessage("INFO> Voltage of +1'65A VFE1 Out Of Range");
   else wxLogMessage("INFO> Voltage of +1'65A VFE1 %.1f mV", value*resolution);
   I2C.write_command_withargs(address,READ_CHAN_COMMAND,8);
   value=I2C.read_10bits(address);
   if(value==1023) wxLogMessage("INFO> Current of +1'65A VFE2 Out Of Range");
   else wxLogMessage("INFO> Current of +1'65A VFE2 %.1f mA", value*resolution*k);
   I2C.write_command_withargs(address,READ_CHAN_COMMAND,9);
   value=I2C.read_10bits(address);
   if(value==1023) wxLogMessage("INFO> Voltage of +1'65A VFE2 Out Of Range");
   else wxLogMessage("INFO> Voltage of +1'65A VFE2 %.1f mV", value*resolution);
   I2C.write_command_withargs(address,READ_CHAN_COMMAND,10);
```

```
   value=I2C.read_10bits(address);
   if(value==1023) wxLogMessage("INFO> Current of +1'65A VFE3 Out Of Range");
   else wxLogMessage("INFO> Current of +1'65A VFE3 %.1f mA", value*resolution*k);
   I2C.write_command_withargs(address,READ_CHAN_COMMAND,11);
   value=I2C.read_10bits(address);
   if(value==1023) wxLogMessage("INFO> Voltage of +1'65A VFE3 Out Of Range");
   else wxLogMessage("INFO> Voltage of +1'65A VFE3 %.1f mV", value*resolution);
   I2C.write_command_withargs(address,READ_CHAN_COMMAND,12);
   value=I2C.read_10bits(address);
   if(value==1023) wxLogMessage("INFO> Current of +1'65A VFE4 Out Of Range");
   else wxLogMessage("INFO> Current of +1'65A VFE4 %.1f mA", value*resolution*k);
   I2C.write_command_withargs(address,READ_CHAN_COMMAND,13);
   value=I2C.read_10bits(address);
   if(value==1023) wxLogMessage("INFO> Voltage of +1'65A VFE4 Out Of Range");
   else wxLogMessage("INFO> Voltage of +1'65A VFE4 %.1f mV", value*resolution);
   I2C.write_command_withargs(address,READ_CHAN_COMMAND,14);
   value=I2C.read_10bits(address);
   if(value==1023) wxLogMessage("INFO> Current of +1'65A VFE5 Out Of Range");
   else wxLogMessage("INFO> Current of +1'65A VFE5 %.1f mA", value*resolution*k);
   I2C.write_command_withargs(address,READ_CHAN_COMMAND,15);
   value=I2C.read_10bits(address);
   if(value==1023) wxLogMessage("INFO> Voltage of +1'65A VFE5 Out Of Range");
   else wxLogMessage("INFO> Voltage of +1'65A VFE5 %.1f mV", value*resolution);
   I2C.write_command_withargs(address,READ_CHAN_COMMAND,16);
   value=I2C.read_10bits(address);
   if(value==1023) wxLogMessage("INFO> Current of +1'65A VFE6 Out Of Range");
   else wxLogMessage("INFO> Current of +1'65A VFE6 %.1f mA", value*resolution*k);
   I2C.write_command_withargs(address,READ_CHAN_COMMAND,17);
   value=I2C.read_10bits(address);
   if(value==1023) wxLogMessage("INFO> Voltage of +1'65A VFE6 Out Of Range");
   else wxLogMessage("INFO> Voltage of +1'65A VFE6 %.1f mV", value*resolution);
   I2C.write_command_withargs(address,READ_CHAN_COMMAND,18);
   value=I2C.read_10bits(address);
   if(value==1023) wxLogMessage("INFO> Current of +1'65A VFE7 Out Of Range");
   else wxLogMessage("INFO> Current of +1'65A VFE7 %.1f mA", value*resolution*k);
   I2C.write_command_withargs(address,READ_CHAN_COMMAND,19);
   value=I2C.read_10bits(address);
   if(value==1023) wxLogMessage("INFO> Voltage of +1'65A VFE7 Out Of Range");
   else wxLogMessage("INFO> Voltage of +1'65A VFE7 %.1f mV", value*resolution);
   wxLogMessage("INFO> END OF DATA DUMP...");
   wxLogMessage("---------------------------------------------------------------------------------------------------------------------");
}

void MyFrame::OnFPGAALL(wxCommandEvent & event)
```

```
{
   unsigned int value=0;
   double resolution=1.955034;
   wxLogMessage("---------------------------------------------------------------------------------------------------------------------");
   wxLogMessage("   READING FPGA MONITORING DATA FOR ALL CHANNELS...");
   wxLogMessage("---------------------------------------------------------------------------------------------------------------------");
   wxLogMessage("INFO> START OF DATA DUMP...");
   for (int i=0; i<48; i++)
   {
         I2C.write_command_withargs(address,READ_CHAN_COMMAND,i);
         value=I2C.read_10bits(address);
         if(value==1023) wxLogMessage("INFO> Voltage of Channel %i Out Of Range",i);
         else wxLogMessage("INFO> Voltage of Channel %i = %.1f mV",i, value*resolution);
   }
   wxLogMessage("INFO> END OF DATA DUMP...");
   wxLogMessage("---------------------------------------------------------------------------------------------------------------------");
}

void MyFrame::OnFPGAReadChanClose()
{
    int chan=0,b1=0,b2=0,b3=0,b4=0,b5=0,b6=0,b7=0,b8=0,b9=0,b10=0;
    unsigned int value=0;
    GetMenuBar()->EnableTop(0, TRUE);
    GetMenuBar()->EnableTop(1, TRUE);
    GetMenuBar()->EnableTop(2, TRUE);
    GetMenuBar()->EnableTop(3, TRUE);
    if (channel_window_opened)
    {
        channel_window_opened = false;
        chan = m_Channel_window.Get_Channel();
        wxLogMessage("INFO> Reading FPGA analog channel: %i",chan);
        m_Channel_window.Show(FALSE);
        I2C.write_command_withargs(address,READ_CHAN_COMMAND,chan);
        value=I2C.read_10bits(address);
        if(value&1) b1=1;
        else b1=0;
        if(value&2) b2=1;
        else b2=0;
        if(value&4) b3=1;
        else b3=0;
        if(value&8) b4=1;
        else b4=0;
        if(value&16) b5=1;
        else b5=0;
        if(value&32) b6=1;
        else b6=0;
        if(value&64) b7=1;
        else b7=0;
        if(value&128) b8=1;
        else b8=0;
        if(value&256) b9=1;
        else b9=0;
        if(value&512) b10=1;
        else b10=0;
        if(value>1022) wxLogMessage("INFO> Channel Out Of Range");
        wxLogMessage("INFO> Channel value read: %i%i%i%i%i%i%i%i%i%i",b10, b9,b8,b7,b6,b5,b4,b3,b2,b1);
    }
}
```

```
void MyFrame::OnAddrClose()
{
    GetMenuBar()->EnableTop(0, TRUE);
    GetMenuBar()->EnableTop(1, TRUE);
    GetMenuBar()->EnableTop(2, TRUE);
    GetMenuBar()->EnableTop(3, TRUE);
    if (address_window_opened)
    {
        address_window_opened = false;
        address = m_Address_window.Get_Address();
        wxLogMessage("INFO> Address: %i",address);
        m_Address_window.Show(FALSE);
    }
}

void MyFrame::OnAskId (wxCommandEvent & event)
{
   char* ID=new char[255];
   wxLogMessage("INFO> Ask ID to device %i with response:",address);
   ID=I2C.read_string(address,ID_COMMAND);
   wxLogMessage("INFO> %s",ID);
}

void MyFrame::OnSendPU (wxCommandEvent & event)
{
   bool check;
   wxLogMessage("INFO> Sending Power Up to device.");
   check=I2C.write_command(address,POWUP);
   if(check) wxLogMessage("ERROR> No ack received!");
}

void MyFrame::OnSendPD (wxCommandEvent & event)
{
   bool check;
   wxLogMessage("INFO> Sending Power Down to device.");
   check=I2C.write_command(address,POWDWN);
   if(check) wxLogMessage("ERROR> No ack received!");
}

void MyFrame::OnSendPUFIRST (wxCommandEvent & event)
{
   bool check;
   wxLogMessage("INFO> Sending Power Up First to device.");
   check=I2C.write_command(address,POWUPFIRST);
   if(check) wxLogMessage("ERROR> No ack received!");
}

void MyFrame::OnFileSave (wxCommandEvent & event)
{
    //Respond to menu here
    wxFileDialog dialog(this, wxT("Save as"), wxEmptyString, myFilename,
        wxT("CBEmu log files (*.cbl)|*.cbl|All files (*.*)|*.*"),
        wxSAVE|wxOVERWRITE_PROMPT);

    if (dialog.ShowModal() == wxID_OK)
    {
        myFilename = dialog.GetPath() ;
        // code might continue, but delete the WXUNUSED from the lines above
        textW->SaveFile(myFilename);
    }
}

void MyFrame::OnReadChannels (wxCommandEvent & event)
{
```

```
   float value_read, temperature;
   if(Minilab.MinilabInit())
   {
      wxLogMessage ("INFO> Minilab found...");
      for (int i=0; i<8; i++)
      {
        value_read=Minilab.GetChannel(i);
        temperature=(value_read/0.008)-273;
        wxLogMessage ("INFO> Read Temperature probe %i with value %.2f ºC", i,temperature);
        Sleep(100);
      }
   }
   else wxLogMessage("ERROR> Minilab not found...");
}

void MyFrame::OnStartTempTest (wxCommandEvent & event)
{
   if(Minilab.MinilabInit())
   {
      int counter=0;
      float chanarray[8];
      wxLogMessage ("INFO> Minilab found...");
      wxLogMessage ("INFO> Starting 1 sec read of temperature proves...");
      wxFileDialog dialog(this, wxT("Save as"), wxEmptyString, myFilename,
      wxT("CBEmu data files (*.dat)|All files (*.*)|*.*"),
      wxSAVE|wxOVERWRITE_PROMPT);
          if (dialog.ShowModal() == wxID_OK)
              {
                      myFilename = dialog.GetPath() ;

                      // code might continue, but delete the WXUNUSED from the lines above
              }
      wxProgressDialog status("Temperature aquisition in progress...", "Aquiring data...", 10, NULL, wxPD_AUTO_HIDE | wxPD_APP_MODAL | wxPD_CAN_ABORT);
      while(status.Update(counter))
         {
         Sleep(1000);
         for (int i=0; i<8; i++)
            {
               chanarray[i]= Minilab.GetChannel(i);
               //wxLogMessage ("INFO> Data read: ch%i %.3f V",i+1, chanarray[i]);
            }
         counter++;
         if (counter==10) counter=0;
         }
      wxLogMessage ("INFO> User Interrupt!");
   }
}

// ----------------------------------------------------------------------------
// MyStatusBar
// ----------------------------------------------------------------------------

#ifdef __VISUALC__
    // 'this' : used in base member initializer list -- so what??
    #pragma warning(disable: 4355)
#endif

MyStatusBar::MyStatusBar(MyFrame *parent)
```

```
           : wxStatusBar(parent, -1), m_timer(this) // , m_checkbox(NULL)
{
    m_Parent = parent;

    static const int widths[Field_Max] = { -1, BITMAP_SIZE_X, 100 };

    SetFieldsCount(Field_Max);
    SetStatusWidths(Field_Max, widths);

#ifdef USE_STATIC_BITMAP
    m_statbmp = new wxStaticBitmap(this, -1, wxIcon(green_xpm));
#else
    m_statbmp = new wxBitmapButton(this, -1, CreateBitmapForButton(TRUE),
                                   wxDefaultPosition, wxDefaultSize,
                                   wxBU_EXACTFIT);
#endif

    m_timer.Start(1000);

    SetMinHeight(BITMAP_SIZE_Y);

    UpdateClock();
}

#ifdef __VISUALC__
    #pragma warning(default: 4355)
#endif

MyStatusBar::~MyStatusBar()
{
    if ( m_timer.IsRunning() )
    {
        m_timer.Stop();
    }
}

wxBitmap MyStatusBar::CreateBitmapForButton(bool on)
{
    static const int BMP_BUTTON_SIZE_X = 10;
    static const int BMP_BUTTON_SIZE_Y = 9;

    wxBitmap bitmap(BMP_BUTTON_SIZE_X, BMP_BUTTON_SIZE_Y);
    wxMemoryDC dc;
    dc.SelectObject(bitmap);
    dc.SetBrush(on ? *wxGREEN_BRUSH : *wxRED_BRUSH);
    dc.SetBackground(*wxLIGHT_GREY_BRUSH);
    dc.Clear();
    dc.DrawEllipse(0, 0, BMP_BUTTON_SIZE_X, BMP_BUTTON_SIZE_Y);
    dc.SelectObject(wxNullBitmap);

    return bitmap;
}

void MyStatusBar::OnSize(wxSizeEvent& event)
{
    wxRect rect;
    GetFieldRect(Field_TIMEOUT, rect);

    wxSize size = m_statbmp->GetSize();

    m_statbmp->Move(rect.x + (rect.width - size.x) / 2,
                    rect.y + (rect.height - size.y) / 2);

    event.Skip();
}
```

```
void MyStatusBar::UpdateClock()
{
    // Check if menus should be Re-enabled in parent window
    if( !GetParent()->GetAddress_window().IsShown() ) GetParent()->OnAddrClose();
    if( !GetParent()-> GetChannel_window().IsShown() ) GetParent()->OnFPGAReadChanClose();
    SetStatusText(wxDateTime::Now().FormatTime(), Field_Clock);
}

void MyStatusBar::UpdateTimeoutStatus(const bool isOK)
{
#ifdef USE_STATIC_BITMAP
        m_statbmp->SetIcon(wxIcon((isOK? green_xpm: red_xpm));
#else
        m_statbmp->SetBitmapLabel(CreateBitmapForButton(isOK));
        m_statbmp->Refresh();
#endif
}
```

main.h

```
//
#define ID_COMMAND 0xFA
#define POWUP 0xFB
#define POWDWN 0xFC
#define POWUPFIRST 0xFD
#define READ_CHAN_COMMAND 0xFE
```

I2C.cpp

```
//Default I2C.cpp

// For compilers that support precompilation, includes "wx.h".
#include "wx/wxprec.h"

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP
#include "wx/wx.h"
#endif

#include <wx/defs.h>
#include <stdio.h>
#include "I2C.h"
#include "SPPctrl.h"

#define CY_SHIFT(a,b,n) a=((a<<b)|(a>>(n-b)))


// ============================================================================
// implementation
// ============================================================================

//Constructor
I2Cctrl::I2Cctrl()
{
}

void I2Cctrl::init()
{
    spp.init();
}

// ----------------------------------------------------------------------------
// Low level access to SPP port using WinIo driver
// ----------------------------------------------------------------------------

void I2Cctrl::SDA_out()
{
    spp.outd(0x10|(spp.ind()));
}

void I2Cctrl::SDA_in()
{
    spp.outd(0xEF&(spp.ind()));
}

void I2Cctrl::Power_transceivers()
{
    spp.outd(0x07|(spp.ind()));   //LED1 on, D1 and D2 on
}

void I2Cctrl::SCL_cycle()
{
    spp.outd(0x48|(spp.ind()));   //SCL = 1, LED2 ON
    Sleep(I2C_DELAY);
    spp.outd(0xB7&(spp.ind()));   //SCL = 0, LED2 Off
    Sleep(I2C_DELAY);
}
```

```
void I2Cctrl::SDA_cycle()
{
    SDA_out();
    spp.outd(0x60|(spp.ind()));   //SDA = 1, LED2 ON
    Sleep(I2C_DELAY);
    spp.outd(0x9F&(spp.ind()));   //SDA = 0, LED2 Off
    Sleep(I2C_DELAY);
}

void I2Cctrl::SDA_one()
{
    spp.outd(0x20|(spp.ind()));    //SDA = 1
}

void I2Cctrl::SDA_zero()
{
    spp.outd(0xDF&(spp.ind()));    //SDA = 0
}

void I2Cctrl::SCL_one()
{
    spp.outd(0x08|(spp.ind()));    //SCL = 1
}

void I2Cctrl::SCL_zero()
{
    spp.outd(0xF7&(spp.ind()));    //SCL = 0
}

void I2Cctrl::start()
{
   SDA_out();
   SDA_one();
   SCL_one();
   Sleep(I2C_DELAY);
   SDA_zero();
   Sleep(I2C_DELAY);
   SCL_zero();
   Sleep(I2C_DELAY);
}

void I2Cctrl::stop()
{
   SDA_out();
   SDA_zero();
   SCL_one();
   Sleep(I2C_DELAY);
   SDA_one();
   Sleep(I2C_DELAY);
}

void I2Cctrl::ack()
{
   SDA_out();
   SDA_zero();
   SCL_one();
   Sleep(I2C_DELAY);
   SCL_zero();
   SDA_one();
}

void I2Cctrl::nack()
{
   SDA_out();
```

```
   SDA_one();
   SCL_one();
   Sleep(I2C_DELAY);
   SCL_zero();
}

bool I2Cctrl::check_ack()
{
   bool error_bit;
   SDA_in();
   SCL_one();
   Sleep(I2C_DELAY);
   if(0x40&spp.ins()) error_bit=true;
   else error_bit=false;
   SCL_zero();
   Sleep(I2C_DELAY);
   SDA_out();
   return error_bit;
}

// ----------------------------------------------------------------------------
// No as low level access to SPP port using WinIo driver
// ----------------------------------------------------------------------------

bool I2Cctrl::send_byte(byte Value)
{
   bool error_bit;
   CY_SHIFT(Value,1,8);
   for(int i=0;i<8; i++)
   {
      if(Value&0x01) SDA_one();
      else SDA_zero();
      SCL_one();
      Sleep(I2C_DELAY);
      SCL_zero();
      Sleep(I2C_DELAY);
      CY_SHIFT(Value,1,8);
   }
   error_bit=check_ack();
   return error_bit;
}

char I2Cctrl::receive_byte(void)
{
   char dada=0;
   SDA_in();
   for(int i=0;i<8; i++)
   {
      CY_SHIFT(dada,1,8);
      SCL_one();
      Sleep(I2C_DELAY/2);
      if(0x40&spp.ins()) dada|=0x01;
      else dada&=0xFE;
      Sleep(I2C_DELAY/2);
      SCL_zero();
      Sleep(I2C_DELAY);
   }
   return dada;
}

char* I2Cctrl::check_for_boards()
{
   bool ack;
```

```
   char* valid_adresses=new char[127];
   int count_valid=0;

   spp.outd(0x40|(spp.ind()));                        //LED2 ON
   for(int i=0; i<127; i++)
      {
            start();
            ack=send_byte((i+1)<<1);
            if(!ack)                                  //Someone responds
               {
               count_valid++;
               valid_adresses[count_valid]=i+1;
               stop();
               }
      }
   valid_adresses[0]=count_valid;
   valid_adresses[count_valid+1]='\0';
   spp.outd(0xBF&(spp.ind()));                        //LED2 OFF
   return valid_adresses;
}

bool I2Cctrl::write_command(char address, char data)
{
   bool ack;
   spp.outd(0x40|(spp.ind()));                        //LED2 ON
   start();
   ack=send_byte((address<<1)&0xFE);                  //Send address, write
   if(!ack)   ack=send_byte(data);                    //Then send command byte
   stop();
   spp.outd(0xBF&(spp.ind()));                        //LED2 OFF
   return ack;
}

bool I2Cctrl::write_command_withargs(char address, char command, char arg)
{
   bool ok;
   spp.outd(0x40|(spp.ind()));                        //LED2 ON
   start();
   ok=send_byte((address<<1)&0xFE);                  //Send address, write
   if(!ok)   ok=send_byte(command);                 //Then send command byte
   if(!ok)   ok=send_byte(arg);                     //Then send argument byte
   stop();
   spp.outd(0xBF&(spp.ind()));                        //LED2 OFF
   return ok;
}

unsigned int I2Cctrl::read_10bits(char address)
{
   bool ok;
   unsigned int value;
   unsigned char temp1, temp2;
   spp.outd(0x40|(spp.ind()));                        //LED2 ON
   start();
   ok=send_byte((address<<1)|0x01);                  //Send address, read
   temp1 = receive_byte();
   ack();
   temp2 = receive_byte();
   value = (temp1 << 2) | temp2;
   nack();
   stop();
   spp.outd(0xBF&(spp.ind()));                        //LED2 OFF
   return value;

}

```

```
char* I2Cctrl::read_string(char address, char command)
{
    bool ok;
    char* data=new char[255];
    char dada=1;
    int i=0;
    spp.outd(0x40|(spp.ind()));                          //LED2 ON
    start();
    ok=send_byte((address<<1)&0xFE);                    //Send address, write
    if(!ok) ok=send_byte(command);                    //Then send command byte
    stop();
    start();
    send_byte((address<<1)|0x01);                     //Send address, read
    while((dada!=0)&(i<255))
       {
          dada=receive_byte();
          if(dada!=0) ack();
          else nack();
          data[i]=dada;
    //      wxLogMessage("INFO> Dada: %c.",dada);
          i++;
       }
    stop();
    spp.outd(0xBF&(spp.ind()));                          //LED2 OFF
    return data;
}
```

I2C.h

```
//Default I2Cctrl.h
#ifndef _I2Cctrl_h
#define _I2Cctrl_h

#include <windows.h>
#include <wx/textctrl.h>
#include "SPPctrl.h"

/* Constants */
const byte I2C_DELAY = 0;   // miliseconds

class I2Cctrl
{
public:
    // ctor(s)
    I2Cctrl();
    void SCL_cycle();
    void SDA_cycle();
    void Power_transceivers();
    void init();
    bool send_byte(byte Value);
    char receive_byte();
    char* check_for_boards();
    bool write_command(char address, char byte);
    bool write_command_withargs(char address, char command, char arg);
    char* read_string(char address, char command);
    unsigned int read_10bits(char address);

private:

    SPPctrl spp;            // SPP parallel port control
    void SDA_one();
    void SDA_zero();
    void SCL_one();
    void SCL_zero();
    void start();
    void stop();
    void ack();
    void nack();
    void SDA_out();
    void SDA_in();
    bool check_ack();


};

#endif
```

MinilabCtrl.cpp

```cpp
//Default CallibrationCtrl.cpp

// For compilers that support precompilation, includes "wx.h".
#include "wx/wxprec.h"

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP
#include "wx/wx.h"
#endif

#include <wx/defs.h>
#include <wx/progdlg.h>
#include <wx/file.h>
#include <wx/textfile.h>
#include <wx/datetime.h>
#include <wx/tokenzr.h>
#include "MinilabCtrl.h"
#include "cbw.h"


// ============================================================================
// implementation
// ============================================================================

#define EOL "\r\n"
#define TAB "\t"

#define MINILAB_BOARD 0

// constructor
MinilabCtrl::MinilabCtrl()
{

}

float MinilabCtrl::GetChannel(int channel)
{
   int    ML_Gain = BIP10VOLTS;
   int    ML_BoardNum = MINILAB_BOARD;
   WORD   DataValue = 0;
   float  EngUnits= 0.0;
   // Minilab access
   cbAIn(ML_BoardNum, channel, ML_Gain, &DataValue);
   cbToEngUnits(ML_BoardNum, ML_Gain, DataValue, &EngUnits);
   return (EngUnits);
}

bool MinilabCtrl::MinilabInit(void)
{
   // Minilab (ML) variables and settings
   float  ML_RevLevel = (float)CURRENTREVNUM;
   int    ML_BoardNum =MINILAB_BOARD;

   // Internal variables
   bool isOk;

   /* ML Declare UL Revision Level */  //Minilab init
   cbDeclareRevision(&ML_RevLevel);

   /* Initiate error handling
```

```
        Parameters:
            PRINTALL : all warnings and errors encountered will be printed
            STOPALL  : if any error is encountered, the program will stop
    */
    cbErrHandling (PRINTALL, STOPALL);
    if(NOERRORS == cbFlashLED(ML_BoardNum)) isOk=true;
    else isOk=false;
    return (isOk);
}
```

MinilabCtrl.h

```

#ifndef _MinilabCtrl_h
#define _MinilabCtrl_h

class MinilabCtrl
{
public:
    // ctor(s)
    MinilabCtrl(void);
    float GetChannel(int channel);
    bool MinilabInit(void);

private:

};

#endif

```

ChannelWindow.cpp

```
/////////////////////////////////////////////////////////////////////////////
// Name:        Channel_window.cpp
//
/////////////////////////////////////////////////////////////////////////////

// ============================================================================
// declarations
// ============================================================================

// ----------------------------------------------------------------------------
// headers
// ----------------------------------------------------------------------------

// For compilers that support precompilation, includes "wx/wx.h".

#ifdef __BORLANDC__
    #pragma hdrstop
#endif

// for all others, include the necessary headers (this file is usually all you
// need because it includes almost all "standard" wxWindows headers)
#ifndef WX_PRECOMP
    #include "wx/wx.h"
#endif

#include <wx/image.h>
#include <wx/filename.h>
#include <wx/defs.h>
#include <wx/file.h>
#include <wx/textfile.h>
#include <wx/datetime.h>

#include "Channel_window.h"

// ----------------------------------------------------------------------------
// constants
// ----------------------------------------------------------------------------

// IDs for the controls and the menu commands
enum
{
    //Address control
    wxM_CHAN,
    wxM_OK
};

// ----------------------------------------------------------------------------
// event tables and other macros for wxWindows
// ----------------------------------------------------------------------------

// the event tables connect the wxWindows events with the functions (event
// handlers) which process them. It can be also done at run-time, but for the
// simple menu events like this the static method is much simpler.
```

```
BEGIN_EVENT_TABLE(Channel_window, wxFrame)
    EVT_BUTTON(wxM_OK,  Channel_window::OnQuit)
    EVT_CLOSE(Channel_window::OnClose)
END_EVENT_TABLE()


// ============================================================================
// implementation
// ============================================================================
// constructor

    wxTextCtrl  *chan;

Channel_window::Channel_window(void):
    wxFrame(NULL, -1, _T( "Analog Channel Selection" ),wxDefaultPosition, wxDefaultSize,wxMINIMIZE_BOX | wxSYSTEM_MENU | wxCAPTION | wxMAXIMIZE_BOX )
{
    // Main window for channel window
    wxSizer *sizerTop = new wxBoxSizer(wxHORIZONTAL);
    panel=new wxPanel(this); //background pannel
    panel->SetAutoLayout(TRUE);
    panel->SetSizer(sizerTop);

    wxStaticText *Titol1=new wxStaticText(panel,-1,"New Channel: ");
    sizerTop->Add(Titol1, 0, wxALIGN_CENTRE_HORIZONTAL);
    chan = new wxTextCtrl(panel, -1, _T("0"),wxDefaultPosition, wxDefaultSize);
    sizerTop->Add(chan,1,wxALL,5);
    wxButton *btnOK = new wxButton(panel, wxM_OK, _T("OK"));
    sizerTop->Add(btnOK, 1, wxALIGN_CENTRE_HORIZONTAL | wxALL, 5);
    sizerTop->SetSizeHints( this );
}


 int Channel_window::Get_Channel()

 {
   wxString ChannelText;
   ChannelText = chan->GetValue();
   return (atoi((char*)(ChannelText.c_str())));
 }
```

ChannelWindow.h

```
//

#include <wx/spinctrl.h>
#include <wx/valtext.h>

class Channel_window : public wxFrame
{
public:
  //ctor
  Channel_window(void);
  void OnClose() {Show(FALSE);};
  void OnQuit() {Show(FALSE);};
  int Get_Channel();

private:

  wxPanel *panel;
  wxStaticBox *box1;
  wxButton *btnOK;
 //
 DECLARE_EVENT_TABLE()
};

```

AddressWindow.cpp

```
//////////////////////////////////////////////////////////////////////////////
// Name:        Address_window.cpp
//
//////////////////////////////////////////////////////////////////////////////

// ============================================================================
// declarations
// ============================================================================

// ----------------------------------------------------------------------------
// headers
// ----------------------------------------------------------------------------

// For compilers that support precompilation, includes "wx/wx.h".

#ifdef __BORLANDC__
    #pragma hdrstop
#endif

// for all others, include the necessary headers (this file is usually all you
// need because it includes almost all "standard" wxWindows headers)
#ifndef WX_PRECOMP
    #include "wx/wx.h"
#endif

#include <wx/image.h>
#include <wx/filename.h>
#include <wx/defs.h>
#include <wx/file.h>
#include <wx/textfile.h>
#include <wx/datetime.h>

#include "Address_window.h"

// ----------------------------------------------------------------------------
// constants
// ----------------------------------------------------------------------------

// IDs for the controls and the menu commands
enum
{
    //Address control
    wxM_ADDRESS,
    wxM_OK
};

// ----------------------------------------------------------------------------
// event tables and other macros for wxWindows
// ----------------------------------------------------------------------------

// the event tables connect the wxWindows events with the functions (event
// handlers) which process them. It can be also done at run-time, but for the
// simple menu events like this the static method is much simpler.
```

```
BEGIN_EVENT_TABLE(Address_window, wxFrame)
    EVT_BUTTON(wxM_OK,  Address_window::OnQuit)
    EVT_CLOSE(Address_window::OnClose)
END_EVENT_TABLE()


// ============================================================================
// implementation
// ============================================================================
// constructor

    wxTextCtrl  *addr;

Address_window::Address_window(void):
    wxFrame(NULL, -1, _T( "I2C Address Selection" ),wxDefaultPosition, wxDefaultSize,wxMINIMIZE_BOX | wxSYSTEM_MENU | wxCAPTION | wxMAXIMIZE_BOX )
{
    // Main window for address window
    wxSizer *sizerTop = new wxBoxSizer(wxHORIZONTAL);
    panel=new wxPanel(this); //background pannel
    panel->SetAutoLayout(TRUE);
    panel->SetSizer(sizerTop);

    wxStaticText *Titol1=new wxStaticText(panel,-1,"New I2C Address: ");
    sizerTop->Add(Titol1, 0, wxALIGN_CENTRE_HORIZONTAL);
    addr = new wxTextCtrl(panel, -1, _T("0"),wxDefaultPosition, wxDefaultSize);
    sizerTop->Add(addr,1,wxALL,5);
    wxButton *btnOK = new wxButton(panel, wxM_OK, _T("OK"));
    sizerTop->Add(btnOK, 1, wxALIGN_CENTRE_HORIZONTAL | wxALL, 5);
    sizerTop->SetSizeHints( this );
}


 int Address_window::Get_Address()

 {
   wxString AddressText;
   AddressText = addr->GetValue();
   return (atoi((char*)(AddressText.c_str())));
 }
```

AddressWindow.h

```
//

#include <wx/spinctrl.h>
#include <wx/valtext.h>

class Address_window : public wxFrame
{
public:
  //ctor
  Address_window(void);
  void OnClose() {Show(FALSE);};
  void OnQuit() {Show(FALSE);};
  int Get_Address();

private:

  wxPanel *panel;
  wxStaticBox *box1;
  wxButton *btnOK;
 //
 DECLARE_EVENT_TABLE()
};

```

SPPCtrl.cpp

```
//Default SPPctrl.cpp

// For compilers that support precompilation, includes "wx.h".
#include "wx/wxprec.h"

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP
#include "wx/wx.h"
#endif

#include <wx/defs.h>
#include <stdio.h>
#include "SPPctrl.h"


// ============================================================================
// implementation
// ============================================================================


// constructor
SPPctrl::SPPctrl()
{
}

void SPPctrl::init()
{
  m_myWinIoVersion = _T("WinIo v2.0");
  // Check for driver initialization
  if( !InitializeWinIo() ) {
    wxLogMessage(_T("\n> ERROR: is not installed."));
    wxLogMessage(_T("\n> INFO: You may install it from an administrative account."));
    wxLogMessage(_T("\n> INFO: Use 'WinIoDriver /i' command."));
    wxLogMessage(_T(".. (key) "));
    exit(1);
  } else {
    wxLogMessage(_T("(C-) Yariv Kaplan 1998-2002 for the %s library."), m_myWinIoVersion.c_str());
    wxLogMessage(_T("WinIo is installed and running..."));
    InitPortAddress();
    SetSPPmode();
  }
}

// ----------------------------------------------------------------------------
// Low level access to EPP port using WinIo driver
// ----------------------------------------------------------------------------

byte SPPctrl::inportb(int iPortId) {
  DWORD dwValue;
  byte bValue;

  if( GetPortVal(iPortId, &dwValue, 1) ) bValue = (byte) dwValue;
  else bValue = 255;

  return bValue;
}
```

```
void SPPctrl::outportb(int iPortId, byte bValue) {
  DWORD dwValue= bValue;

  SetPortVal(iPortId, dwValue, 1);
}

byte SPPctrl::ind(){
 DWORD dwValue;
 byte bValue;

 if( GetPortVal(m_lpt_data, &dwValue, 1) ) bValue = (byte) dwValue;
 else bValue = 255;

 return bValue;
}

byte SPPctrl::ins(){
 DWORD dwValue;
 byte bValue;

 if( GetPortVal(m_lpt_status, &dwValue, 1) ) bValue = (byte) dwValue;
 else bValue = 255;

 return bValue;
}

void SPPctrl::outd(byte bValue) {

  SetPortVal(m_lpt_data, bValue, 1);
}

void SPPctrl::outs(byte bValue) {

  SetPortVal(m_lpt_status, bValue, 1);
}


void SPPctrl::InitPortAddress() {

  unsigned int address;

  /* Set port "by hand" */
  address=0x0378;
  wxLogMessage("Selected port Address: %Xh", address);

  /* Updates port address variables */
  m_lpt_data    = address;      /* LPT1 Port */
  m_lpt_status  = address+1;
  m_lpt_control = address+2;
}

void SPPctrl::SetSPPmode(void){
/* Pre:  Port addresses are ok.
   Post: Port controlling registers are initialized.
     Port is set in EPP bidirectional mode.
     Returns nothing.
*/
  outportb(m_lpt_data,0x00);
  for (int i=0; i<5; i++)
     {
       outportb(m_lpt_data,0x01);
       Sleep(100);
       outportb(m_lpt_data,0x40);
```

```
        Sleep(100);
        outportb(m_lpt_data,0x80);
        Sleep(100);
    }
    outportb(m_lpt_data,0x00);
}

```

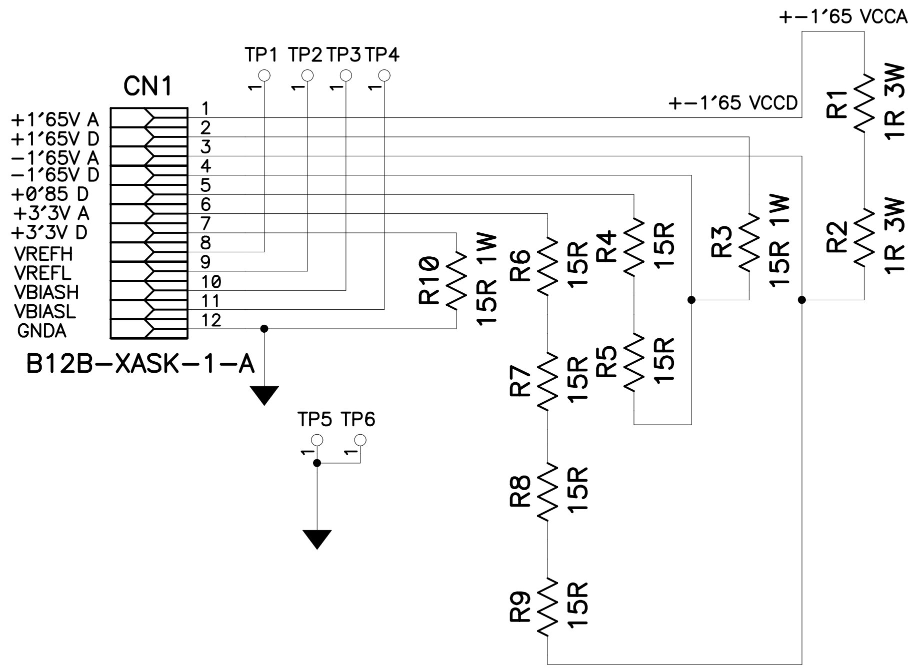
SPPCtrl.h

```
//Default SPPctrl.h
#ifndef _SPPctrl_h
#define _SPPctrl_h

#include <windows.h>
#include <wx/textctrl.h>
#include "winio.h"


// Bit position definition
const byte BIT0 = 0x01;
const byte BIT1 = 0x02;
const byte BIT2 = 0x04;
const byte BIT3 = 0x08;
const byte BIT4 = 0x10;
const byte BIT5 = 0x20;
const byte BIT6 = 0x40;
const byte BIT7 = 0x80;
const byte ALLONES = 0xFF;

/* Constants */
const byte M_SPP_DELAY = 100;   // 100 miliseconds

/* ATENCIO: Aquestes constants dependen del xip real EPP */
/*          Potser s'han de canviar en altres PC's */
#define INIT_m_lpt_data_port     0x00
#define INIT_m_lpt_status_port   0x00

#ifdef DELL
#define INIT_m_lpt_control_port  0x24 /* DELL 0x24, EPP 4008 0x04, Portatil 0x04 */
#else
#define INIT_m_lpt_control_port  0x04 /* DELL 0x24, EPP 4008 0x04, Portatil 0x04 */
#endif

#define INIT_m_lpt_ECR_port  0x80    /* Sets EPP mode (if ECR) - Does it work? */
#define INIT_m_lpt_ECR_clear 0x1F


class SPPctrl
{
public:
    // ctor(s)
    SPPctrl();
    void init();
    byte ind();
    byte ins();
    void outd(byte Value);
    void outs(byte Value);

private:
    wxString       m_myWinIoVersion;
    bool m_timeout;
    void InitPortAddress();
    void SetSPPmode(void);
    byte inportb(int iPortId);
    void outportb(int iPortId, byte bValue);
    int m_lpt_status;
    int m_lpt_control;
    int m_lpt_data;

};

```

```
#endif

```

APPENDIX 4: AUXILIARY BOARDS SCHEMATICS AND PCB’s

LOAD BOARD

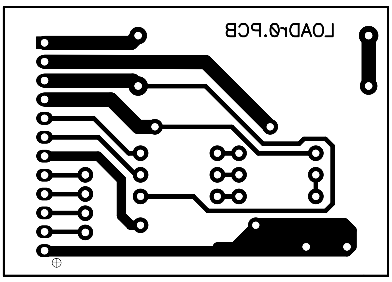


Universitat de Barcelona

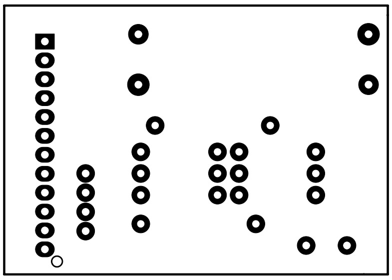
Facultat de Física
Departament de E.C.M.

Autor:	Albert Comerma	Data:	05/07/05
Fitxer:	LOADr0.SCH	Rev:	1
Format: A3	Títol: Regulators board.	Ver:	0
		Full	1 de 1

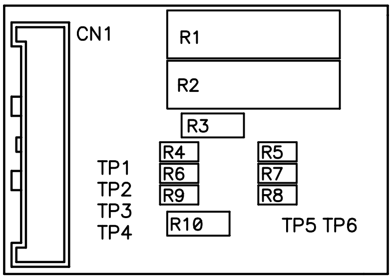
Bottom layer



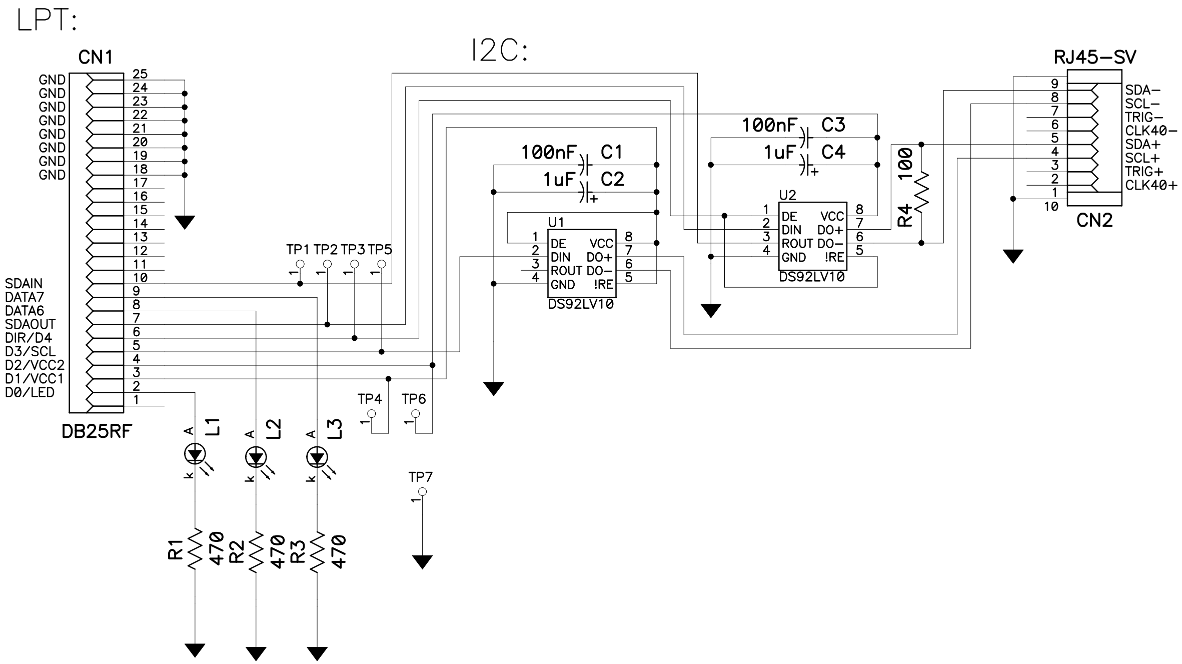
Bottom mask layer



Top silk layer

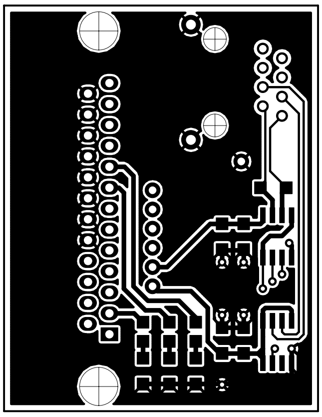


COMUNICATIONS BOARD

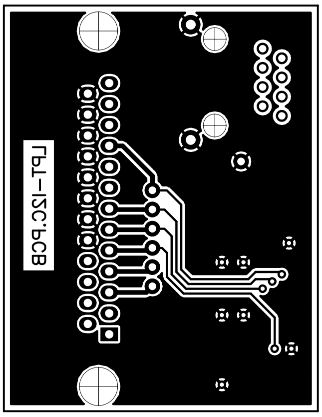


Universitat de Barcelona	Facultat de Física Departament de E.C.M.		
Autor:	Albert Comerma	Data: 10/01/05	
Fitxer:	LPT-I2C.SCH	Rev: 1	
Format: A3	Títol: Adaptador LPT-I2C	Ver: 1	
		Full 1 de 1	8

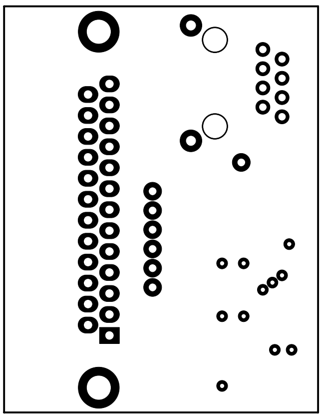
Top layer



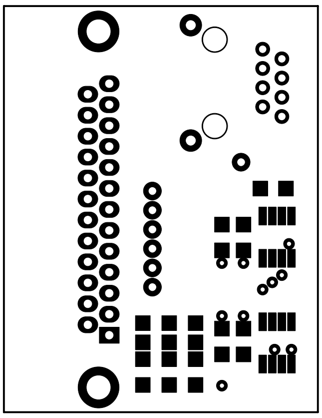
Bottom layer



Bottom mask layer



Top mask layer



Top silk layer