UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

IMPLEMENTATION OF AN OPEN HARDWARE AND FIRMWARE MODULAR RADIOSONDE

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

LUIS ALBERTO HERRERA GONZÁLEZ

PROFESOR GUÍA:
MARCOS DÍAZ QUEZADA

MIEMBROS DE LA COMISIÓN:
MARCOS DÍAZ QUEZADA
ROBERTO RONDANELLI
RICHARD QUEREL

SANTIAGO DE CHILE
SEPTIEMBRE 2013

RESUMEN DE LA MEMORIA PARA OPTAR AL TÍTULO DE: Ingeniero Civil Eléctrico
POR: Luis Alberto Herrera González
FECHA: 10/09/2013
PROFESOR GUÍA: Marcos Díaz Quezada

Implementation of an Open Hardware and Firmware Modular Radiosonde

Las radiosondas son una importante herramienta utilizada por los servicios meteorológicos a través del mundo para ayudar, entre otros usos, a predecir el tiempo. Estos instrumentos son también utilizados en la investigación científica. Los datos recogidos ayudan a advertir al público y pilotos acerca de condiciones de mal tiempo. Desafortunadamente, solo un puñado de modelos de radio-sondas están disponibles comercialmente. Peor aún, los detalles del funcionamiento interno de estos son desconocidos para los usuarios. La personalización de estos modelos es poco factible.

Este informe describe el trabajo desarrollado en el contexto del proyecto Ckelinar. El objetivo de este proyecto es entregar a la comunidad una radio-sonda de diseño abierto. En este contexto abierto significa que el usuario tiene acceso completo al diseño, puede construir, modificar y extender el instrumento. En particular la contribución de este trabajo es definir un *bus* que permita acelerar el desarrollo al dividir el sistema en varios módulos, los que pueden ser desarrollados independientemente y que solo interactúan a través del *bus*. Esto también permite que el sistema pueda ser fácilmente extendido para agregar nuevos sensores según las necesidades del usuario final. Este trabajo también define una metodología de trabajo que permite a los diseñadores a trabajar colaborativamente y mantener registro de las diferente versiones del proyecto. Finalmente se realiza un análisis acerca de las diferentes licencias libres que permiten dar acceso a la comunidad al diseño, considerando que el proyecto esta compuesto tanto de *software* como *hardware*.

Summary

Radiosondes are an important tool used for weather services across the globe to help with weather forecasting. These instruments are also used in scientific research. The data collected helps warn the public and pilots about severe weather conditions. Unfortunately, only a handful of models of radiosondes are commercially available. Even worse, most the details of the internal working of them are unknown to the users. Customization of these models is not likely.

This report describes the work done in the context of the Ckelinar project. Ckelinar means air or atmosphere in Kunza, a language spoken by Atacama region natives. The aim of this project is to provide the community with an open radiosonde. In this context open means that the users have access to the full specifications of the design, can build, modify and extend the instrument. In particular this work's main contribution is to define a bus that allows rapid development by dividing the system into several modules that can be developed independently and that communicate through the bus. This also allows one to easily extend the system and add new sensors depending on the needs of the final user. In addition, this work defines a working methodology of design that allows designers to work collaboratively and keep track of the different versions of the project. Finally an analysis is made about the different free licenses that allow the maker to give the community access to the design considering that this project is comprised of both software and hardware.

"I still don't have all the answers,
but I'm beginning to ask the right questions."
—Lee Lorenz

Acknowledgements

Quisiera comenzar por agradecer en general a mi familia y amigos. Gracias al apoyo de ellos estos años se han sentido muy cortos, quizás demasiado cortos. Agradezco a mi padre, Luis Herrera, por su incondicional apoyo en todas las locuras en las que me he embarcado. Desde construir robotitos hasta internarme en los Andes con tan solo una bici y un poco de agua, él siempre ha estado detrás mío, lo que me ha permitido enfocar mis esfuerzos en las cosas que me apasionan. Agradezco a mi hermano Nicolás, por la alegría que nos entrega día a día. De a poco te he visto madurar y espero verte en el futuro siguiendo tus propios sueños. A Karina Barraza muchas gracias por estar siempre apoyándome ejecutivamente en todos los detalles de mis aventuras.

No puedo dejar de mencionar y agradecer a mis grandes maestros. Profesora Patricia Acosta, por ser la que me guió en mis primeros pasos en el mundillo de la programación. Francisco *fraespin* Espinoza y Leonardo Moreno por enseñarme el arte de Newton. A Francisco también le debo en gran parte mi amor por las bicicletas. Al gurú John Mauro Pereira por su apoyo incondicional al Taller de Robótica del Instituto Nacional, y por ayudarnos con el don de la palabra a conseguir fondos para viajar al extranjero a representar al país en más de una ocasión. A Pablo Guerrero por ser mi mentor cuando aún era un tierno mechón en la Escuela de Ingeniería. Y finalmente Marcos Diaz, quien con su particular estilo pedagógico nos ha forzado a muchos a hacernos y resolver preguntas en vez de solo venir a la Universidad a meternos libros en la cabeza.

También quisiera agradecer a todos mis amigos, que no son muchos, pero son los mejores. José David Marroquín, mi eterno compañero en la bicicleta y otros deportes (especialmente los que implican un alto nivel de sufrimiento). No tengo dudas de que te convertirás en un gran Ingeniero. Pablo Escobar con quien he compartido infinitas conversaciones del más alto nivel filosófico, algún día lograremos encontrar la pregunta fundamental de la vida, el universo y todo lo demás. Francisco Espinoza, un ejemplo a seguir como persona, nunca olvidaré la experiencia que tuvimos en la Ruta del Cóndor ni todas las enseñanzas de mecánica ciclística. Karel Mundnich, uno de mis "nuevos" amigos (solo unos cuantos años) una de las personas más inteligentes que tuve como compañero, cualidad que no solo se aprecia en lo abstracto de su pensamiento, sino que también en lo agradable de su compañía. José David, Francisco y Karel, se que nos volveremos a ver muy pronto ruteando, Le Tour de France nos espera! No puedo dejar afuera a Elizabeth Inostroza quien, a pesar de todo, ha sido una gran amiga y hermosa compañía. Mis solitarias tardes en USA se hubiesen hecho eternas si tú no hubieses estado siempre ahí, acompañandome a la distancia. Nunca olvidaré los hermosos momentos que pasamos juntos. Espero que en lo que viene las cosas cambien para mejor y que seas feliz independientemente de lo que traiga el futuro para ti y para mi.

Finalmente, quisiera agradecer a todas las personas que he no mencionado aquí, pero que han sido claves en mi vida. Ellos saben quienes son y el cariño que tengo por ellos. Mención especial a Stephanie Chan, quien descubrió mi talento y me abrió las puertas en el Silicon Valley.

Contents

Chapter 1

The Radiosonde Project

The work presented in this report is in the context of the Ckelinar project. This is a joint effort of the Department of Electrical Engineering and the Department of Geophysics with the goal of probing the lower atmosphere by designing and building a radiosonde. The targets for this device are to be as good as commercial grade radiosondes, and at the same time, being an open design that can be inspected, used, and improved by the community. This project began in the Engineering Project Workshop course. Groups of second year engineering students from different areas were challenged to design and build a radiosonde to measure several meteorological variables of the atmosphere. The project has already had a couple of successful iterations. Two campaigns have been carried out. The first was in Santiago, reaching an altitude of 3.5 km and a posterior recovery of the radiosonde. The second one launched from Santo Domingo reaching an altitude of 12 km. Unfortunately the radiosonde could not be recovered in the second campaign, however, both can be considered successful due to the data gathered during both flights. A complete description of the project and a deep analysis of the data collected on the campaigns can be found in the paper "The Life Cycle of a Radiosonde" [1].

Despite reaching the desired objectives for the project there have been issues concerning the speed of development of new versions of the radiosonde. So far, the implementation of the hardware has been monolithic and even small changes require a redesign looking at the whole structure to modify or add a single element. The purpose of this work is to define a design to overcome this issue, and allow independent parts of the system to be designed and evolved separately. For this, it is necessary to define a common interface that allows information to flow from the different subsystems and interact without conflicts.

1.1 Objectives

The main objective of this work is to design and implement an open platform that can measure climate variables through the atmosphere, i.e. a radiosonde. It must be able to work autonomously and be small and lightweight. The design must allow people with little knowledge of electronics to build the radiosonde, as well as add and change payloads of the system.

As secondary objectives this work should establish a workflow that allows seamless contribution by different designers and developers to the project, and the definition of a scheme that allows the free release of both hardware and software to be useful for the community.

1.2 Structure of the Report

The second chapter of this report supports the decision of the release of the designs of this project by showing the example of the historical impact of this decision in the context of the free software. It also describes commercial and amateur radiosondes and states the intended contribution of this project.

Chapter three describes the main software used in the project. It briefly explains how hardware design is made and how to keep track of these designs. The core of this chapter is dedicated to the design of the communication bus itself and the design and construction of several modules that demonstrate the usefulness of the bus. In the last section, several free licenses are described and analyzed to finally decide which one best fits this project and describes how the license must be included in the sources of the project.

Chapter four shows several tests performed with the system to verify that the system can operate correctly. Finally, chapter five states the conclusions of the author about the work done as well as some of the work that can be performed in the future.

Chapter 2

Contextualization

2.1 Open Hardware

2.1.1 Free Software Movement, a Retrospective

The open hardware initiatives have a philosophical foundation in the open source movement. This movement started in the 80s when computer manufactures started shipping their computers with their own operating systems. They made users sign nondisclosure agreements in order to monopolize the right to modify and improve the system. Before that, cooperation and code sharing between hackers was usual but from then on this was labeled as piracy. Richard Stallman describes this situation and how he initiated the Free Software Foundation (FSF) in [2]. By definition [3] users of free software should have four essential freedoms: to run the program for any purpose; to study how the program works and change it so it does your computing as you wish; to redistribute copies so you can help your neighbor; and to distribute copies of your modified versions to others. A precondition to fulfill those freedoms is that the source code should be accessible to the users.

From a technical perspective the freedoms mentioned previously can only be put into practice with the correct set of tools, which must also be free. Thus, the GNU (a recursive acronym for "GNU's Not Unix!") Project was initiated by the FSF with the aim of creating a whole computer system using only free software. Even thought this system has not been completed yet, the project has been extremely productive. Some crucial pieces of software used today has been created by GNU. To name a few:

GNU Compiler Collection (GCC): The standard compiler for most UNIX-like systems. It supports many different languages and has been ported to most computer architectures, in fact it is commonly used as a cross compiler.

GNU C Library (glibc): This is the GNU implementation of the C Standard Library, a core library in almost any computer system. It provides basic operations such as input/output, mathematical operations and memory allocation among others.

GNU Debugger (GDB): A fully featured debugger for UNIX-like systems. It supports remote debugging and is usually employed as a debugger for embedded systems.

GNU Make: A tool for building executable programs from multiple sources files.

Another important free software project is Linux. Linux is an operating system kernel. It started as a hobby project in 1991 by Linus Torvalds but due to the great contribution of the community to its development it quickly reached version 1.0.0 in 1994. Linux is widely used on servers, mainframes and supercomputers. Its penetration in the desktop market has also increased in recent years. But definitely it has had a breakthrough in popularity among embedded devices. Commonly routers, media players and other consumer electronics have used Linux as operative system, but with the increased capabilities of cellular telephones nowadays, they also started using Linux [4], dramatically increasing the number of devices now running Linux.

As the reader may realize, many key pieces of software in today's world are in fact free software and thus anyone can make changes, improve and redistribute this software. It is important to notice how the free software development model has shaped the way in which those fundamental tools have been designed and built, and how it has contributed to their increased quality.

2.1.2 Business Model

The open source initiatives have been able to have commercial success despite being in themselves free. This is because while the price of a certain product can be zero, it usage can lead to other necessities. Companies using open source products usually follow another, more elaborated, goal when using them. This way those products allow them to be relieved of the costs of implementation and invest more in their differentiating qualities, obtaining a positive reward, even though using free software does have some side effects. While true that they will not need developers for writing software, they will need to invest in training and support.

Other companies have especially exploited this last case. The so-called "software distributions" have created an entire industry around support services. As free software is distributed independently by its authors it can be difficult for end users to put together a fully working system and keep track of its updates. Software distribution companies, for example Red Hat [5], put together several pieces of software and sell them along with an installer. That is the first source of profit: distribution. The second support. They keep track of updates and security patches and release them to the users. They provide support when users have difficulties using the software as well as training. Thus, it is possible to see that profitable industries can indeed grow around open source technologies.

Open source projects can benefit from the industry as well. In order to provide patches quickly to the users, the companies hire developers from those projects and allow them to keep contributing [6]. They can then prioritize the work of the developers to fix client bugs. Other companies build new features internally, in order to run their business, that are then released publicly.

In the open hardware arena, the projects that release their sources can have commercial success. Here users could possibly build their own hardware and do not need to buy it from the provider, but this does not happen. Users want to integrate and build on top of these products and they are interested in building new designs. The end user is not interested in manufacture and the provider can use economies of scale to provide the product at a lower price. As the design is open those products allow the user to improve them. This is one way to attract users. The other way is that other providers can fabricate devices that communicate and interact with the design. In his way, the user has a wider range of products to use together. A notable example of this is Lego Mindstorms NXT which had released its hardware specification [7]. The sensor communication system is open and other manufactures, like HiTechnic, have created a new set of sensors [8], like force, infrared and barometric pressure. This is a win-win for Lego and HiTechnic. Lego can have more customers and HiTechnic can operate in a specific market.

2.1.3 Open Hardware Projects

Without doubt Arduino has been one of the open hardware projects that has grown most quickly in the last few years. Arduino consists of a hardware micro-controller platform (e.g. the one shown in Figure 2.1), libraries, bootloader and IDE, aimed to be easily used in multidisciplinary projects. It has had a very favorable reception in the hobbyist community worldwide. For example, APM Multiplatform Autopilot [9] is a UAV (unmanned aerial vehicle) autopilot system that is based on the Arduino board and uses the same software libraries. There are plenty of extension boards, called shields, for Arduino in the market. Examples of shields are motor controllers, Ethernet [10], WiFi [11], CAN (controller area network), LCD (liquid-crystal display) and MP3 player. This shows the commercial impact that this platform have, mainly boosted by the engagement of the community with the project.

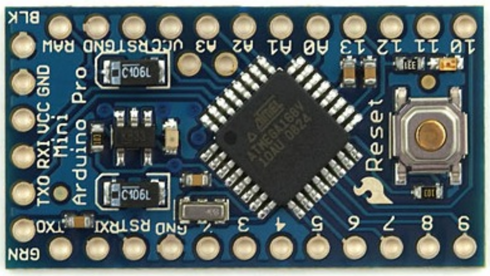


Figure 2.1: Arduino Pro Mini Board.

OpenCores is another popular open hardware initiative. In this case the idea is to provide implementation of hardware IP-cores that can be instantiated using field-programmable gate arrays (FPGA). Many designs can be downloaded from the OpenCore, for example: processors, UART, I^{2}C, Ethernet, FFT and cryptographic cores. This community relies on an open bus interface called Whisbone Bus.

2.2 Meteorological Sounding Challenges

2.2.1 Commercial Radiosondes

Two commercial radiosonde companies and products are the main actors in the sounding market. On the one side, Vaisala is a Finnish company that provides solutions for environmental and industrial measurements. Vaisala radiosonde RS92, shown in Figure 2.2, is advertised as the sonde providing the highest level of pressure, temperature and humidity measurement performance [12]. On the other side, the Intermet iMet-1 sonde is a high accuracy upper-air sonde that features a small and light ground station [13]. Besides sensing there are other components that are interesting as well. For example in order to use either system you need to use the ground station of the same brand. While the sonde itself may be inexpensive the ground station is usually a larger investment. In such a closed system, algorithms for processing the data are unknown and cannot be changed. Another constraint on the payload side is that if someone develops a meteorological sensor and wants to incorporate it into a commercial radiosonde it may not be possible or it will not be well documented as to how to do it. Developing a sensing and transmission system could be impossible for someone with no prior experience in electronics.

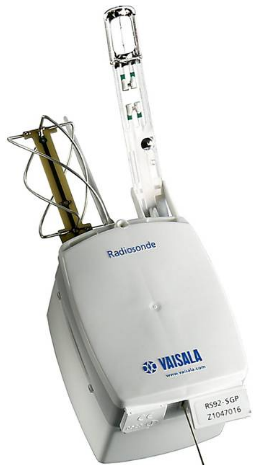


Figure 2.2: Vaisala RS-92.

Regarding the measuring capabilities of the devices the World Meteorological Organization (WMO) conducts regular comparisons of radiosonde systems [14]. From the results of this work it is possible to see that not all commercial radiosondes provide adequate sensing characteristics. The WMO defines a grading scale that standardizes the different levels of performance. This gives some warranties but may not be enough for certain users. Consider for example a geophysicist who wants to measure humidity. This is one of the most difficult atmospheric variables to measure. As he may need maximum precision he would likely use a Chilled Mirror Hygrometer as it is the most precise method available. In the current state of the radiosonde market it would be impossible to buy one fulfilling those characteristics.

The WMO establish several levels of performance for instruments and methods of observations. The Commission for Instruments and Methods of Observations (CIMO) provides several guidelines for measurement of several meteorological variables such as pressure, temperature, humidity, wind and others. These guides also expand to topics like quality management, testing, calibration and intercomparison. The Global Climate Observing System (GCOS) also provides guidelines for instruments integrating the GCOS Reference Upper-Air Network (GRUAN). In the WMO Intercomparison of High Quality Radiosonde Systems [14] the quality of measurements has been graded with a score from 1 to 6 as describe next:

Score 1 Minimum acceptable performance, given current available technology.

Score 2 Close to the accuracy requirement for operations in the CIMO Guide.

Score 3 Just meets the operational requirements of the CIMO Guide.

Score 4 Better than operational requirements of CIMO Guide, but still needs some improvement to become ideal for GRUAN.

Score 5 As good as can be expected fro GRUAN at the moment.

Score 6 Documentation of error corrections and processing supplied to GRUAN Lead Centre.

As an example, in this scale a temperature accuracy (from surface level up to 100 hPa) of 1 K receives a score of 1, while a accuracy of 0.5 K and 0.3 K scores of 3 and 5 respectively. For humidity (above -40^0 C) a minimum score of 1 is achieved with a 20% RH accuracy and a score of 5 with 3% RH accuracy. This guidelines must be taken into consideration when assessing the performance of radiosondes.

This said, it is likely that the radiosonde market lacks several characteristics that may be relevant for certain users. They may need more flexibility in the sensing methods. And the business model and pricing of the current systems leaves space to other actors to participate in this market.

2.2.2 Amateur Radiosondes

Several groups of hobbyists have assembled sounding systems for various purposes, usually to measure atmospheric variables or capture near space images. The description of some of them, taken from their own websites, are:

Project Horus [15] is an amateur high altitude balloon project based in Adelaide, Australia. The aim of the project is to build and fly low cost weather balloon payloads into the stratosphere, capturing photographs, recording sensor data and providing a reliable launch platform for high altitude experiments.

The Icarus Project [16] is a "home brew" project to send a camera high into the stratosphere to take pictures of the Earth from near space (see Figure 2.3). The camera is lifted to an altitude of approximately 35 km above sea level.

The Pegasus High Altitude Balloon project [17] is a UK based amateur student run project that involves launching payloads to near space (flying between an altitude of 60,000 ft (20 km) and 115,000 ft (35 km)).

ALIEN [18] is a project by three students from Reading, UK to launch a high altitude weather balloon to 30 km altitude and take photos along the way.

Figure 2.3: Photo taken at 28.3 km by the Icarus project. The darkness of outer space can be appreciated. Long Island can be seen on the surface of the Earth.

All those projects share the same basic characteristics. They are low budget projects carried out by small groups of students or hobbyists. The payloads consist of GPS, camera, and sensors. As those devices are prototypes recovery is a must.

From the point-of-view of hardware itself, most of those projects make the design of the systems publicly available. This implies specification of sensors, schematic diagrams and software. In this way, it is possible for users wanting to implement those sondes with small changes or improvements to do so. However, due to the nature of the systems, this task may be hard to complete for people with little knowledge of electronics and printed circuit board construction. For example, in most of those projects adding a sensor implies building the whole system. For a user that is not interested in the electronic details it would be desirable to buy the core components and then attach new sensors or capabilities to that core.

The Open Hardware Radiosonde from the Universidad de Chile is similar to those mentioned above. It was described in the introduction of this work and here the implementation issues are discussed. The different versions of this project have been based on Arduino and have been implemented as a single shield. This approach was simple in the first iterations of the project but as changes and flexibility were necessary it proved to be too rigid to allow for fast iterations.

2.3 Project Contribution

The overall contribution of the Ckelinar project is to show that it is possible to build commercial grade radiosondes using off-the-shelf components. By being an open source project it actually provides the design files and software to a worldwide community of developers.

Within the framework of this project, this thesis work contributes to the standardization of the system, providing a platform that can be used internally and externally to build different modules that can connect to each other to build a complete system. Locally this will impact on the iteration speed of the project and could also reach other projects that run in the department that decide to adhere to the standards defined here. That way sharing and collaboration can be easily performed by groups working in similar projects and facing common issues. Globally this work can create the framework for hobbyist to create their own modules to connect to the system, leading possibly to the creation of new markets and commercial opportunities.

Chapter 3

Design and Implementation

3.1 Development Tools

This section describes some of the main software tools used to develop this project. Mainly it will discuss how the PCB design is performed and how changes from different contributors can be tracked together.

3.1.1 Hardware Design

The process of creating the hardware of a module consists of several steps. While it could be possible to directly draw how the copper shape should be on the PCB, this approach is difficult and does not allow separation of the different stages of the design. Instead the design process starts by describing the schematic of the module at the level of logical connections. Then the software will translate the components and connections on the schematic into footprints and routing guidance. Using that the designer can draw the layers of copper on the PCB, while the software takes care of ensuring that the copper connections correspond to those in the original schematic.

The general category of tools to help in this process are denominated "electronic design automation" (EDA) tools. There is a wide variety of EDA tools depending on the purpose and level of integration. The focus of this work is on the design of printed circuit boards. In this case each one of the components is bought from a electronic store, and a board with copper connections printed on its surface is manufactured. Then the components are attached to this surface. For this purpose, it is useful that the tools have libraries with different components as well as their footprints. Otherwise it is the job of the designer to create them.

There are several tools that can perform the tasks specified previously. One of the most popular is Eagle. Its popularity and widespread use not withstanding, Eagle has important drawbacks that make it unsuitable for this project. The first is that Eagle is not free. There is a freeware version [19] that restricts the size of the boards and the number of layers that can be routed. These restrictions are not critical but there is one that it is. This version can only be used for "non-profit applications or evaluation purposes". That implies that everyone that would like to modify the design of the project and then build a for-profit application would have to buy Eagle. Associating a free hardware project to non-free software restricts the freedom of the user regarding the design, hence it is clear that both are not compatible. Another drawback of Eagle is that it stores the design in a binary format which makes it difficult to keep track and explore changes when using a version control system. For example tools like `diff` simply do not work on binary files. A fully featured free software alternative to Eagle is KiCad which is described next.

KiCad

The software tool that was selected for this project is KiCAD [20]. As described in the website of the project, KiCad is an EDA software suite for the creation of professional schematics and printed circuit boards. KiCad is free and released under GNU GPL v2. Besides being a complete tool offering all the operations needed for PCB design, KiCad has the characteristic that all the file formats are ASCII text. This is convenient when storing multiple versions of a design since it is easy to generate and store only the diffs from version to version. Internally the software for tracking changes can work more efficiently when working in this format, and developers can see more easily what are the changes being made from version to version.

The first tool of KiCad is `eeschema`. This allows the designer to create the schematic of the module. With this tool the developer can also view and edit libraries of components. Assuming that all of the components needed are in the libraries, the process of creating a schematics starts by placing all the components into the design. Most commonly used components are already shipped with the program, otherwise a component must be created and added to a library. Once all the components are present in the design a wire tool allows one to electrically connect them by touching any two parts that need to be connected. Finally, an electric rule check allows one to verify that the design meets several rules that can avoid common pitfalls, e.g. unconnected wires.

When the schematic is complete, a sketch of the PCB is needed. To do this the designer must perform the following steps: 1) The design is converted to a netlist, 2) Using the tool `cvpcb` a footprint is selected for every component on the netlist, 3) With these two files, the designer is ready to start working on the PCB.

The last step is to dump all of the footprints into an empty PCB. The footprints will have lines indicating the pads that must be connected. First the designer must place the components on the board. Usually related components are placed together since there may be more connections among them. After placing, the designer must draw copper lines to connect the components on the PCB. Once this is done, a design rule check is performed to ensure that all components are connected properly, as described in the schematic, and that the manufacturing rules are met. Finally the program can output Gerber files that can be used to build the PCB using CNC machines.

KiCad has complete documentation available online [21]. One of the most valuable resources is the "KiCad Getting Started Guide" that is a tutorial that allows the reader to easily learn all of the basic aspects of PCB design using this software.

3.1.2 Version Control

One of the most basic aspects of working on teams corresponds to tracking the source code of the project. The tools that solve this problem are called version control systems. There are many alternative tools to use, and each one has particular characteristics: Perforce [22], Subversion [23] and Git [24] rank as the most popular ones. For this project the selected tool was Git. This is based mainly on the current tendency of the open software movement to use this tool and that technically tracking hardware designs is the same as tracking software. Despite having a good performance and being able to track millions of lines of code Perforce is ruled out because it is not free software, thus while being popular in the corporate environment it is not among students and hobbyists. Subversion is based on a centralized server-client architecture. While being widely used some years ago, Subversion has two main drawbacks. Its architecture does not allow clients to operate on the repository if they are disconnected from the server. This also implies that every operation has to pay the latency delay to the server, so most operations become very slow. The second drawback is that it does not keep track of the starting point of each change, so when performing a branch (splitting the workflow history) and repeated merges (joining two workflow histories) the developer will be forced to resolve spurious conflicts.

Technically, Git is a content-indexed database of snapshots of versions of the project. Each one of these snapshots stores the content of the files as well as information of the preceding versions of those changes. Usually those snapshots are called commits. The information of the origin of each commit form an acyclic directed graph. When working on a project the developer will have on its computer a Git repository, usually under the `.git` folder, with all of the history of the project, a checkout of a version of the project called *working directory*, and a *staging* area, where the next commit is being created. Those elements can be visualized in the Figure 3.1. To create these structures the developer must clone the repository from another developer or from a centralized copy. In the case of this project the central repository has been stored on GitHub[1]. To clone the repository the developer can execute the command:

```
$ git clone https://github.com/herrera-luis-alberto/OpenSonde.git
```

This will ask for a username and password. On the current directory a folder called `OpenSonde` will be created containing the `.git` directory and a working directory of the latest version of the branch master. Once this is done the user can examine all the files in the directory and for example generate the output for fabricating the modules. If the user is a developer and wants to contribute to the project he must proceed as described next.

To contribute to the project the developer must create one or more commits into his local repository. A good rule for adding changes to a repository is that every commit must have

[1]https://github.com/

one and only one purpose, which must be clearly stated in the commit description. The purpose of this is it allows one to quickly examine the repository in case something goes wrong and a specific change must be analyzed. Thus, first to create a commit the developer needs to modify or create files in the working directory. Then, the modified changes and new files are added to the staging area using the command `git add`. In case a file must be deleted it is necessary to use the command `git rm` instead of a plain `rm`. Once the stage contains all the modifications that will be added or deleted from the last commit it can be inserted into the repository using the `git commit` command. At this point those changes are still locally stored on the developer's machine. This process can be repeated as many times as necessary to implement a feature. The creation of a commit can be visualized in Figure 3.2.

After that point the developer may want to publish the changes so they can be visible to other developers. Here there are two options depending on the writing permissions of the main repository. If the developer can write to the repository, publishing the changes area as simple as synchronizing with the main repository (e.g. `git pull`) and pushing the new changes with the command `git push`. If the developer cannot write to the main repository one option is to publish the changes under another repository on GitHub (using `git push` as well) and request to any developer with writing permissions to pull the changes and then push them to the main repository. This model is common in open source projects where there is a large community of developers but only a few of them are gatekeepers of the main repository.

The work-flow described here is quite simple, but Git supports many other ways to collaborate and share changes among developers. An exhaustive account of all of them, and other interesting technical issues, can be found on the Git Book available on-line at `http://git-scm.com/book`. Another good guide is "A Visual Git Reference"[25] that contains visualizations of the different operations that can be performed on a Git repository (some of them shown in this document).

3.2 System Design

3.2.1 Communication Interface - the Open Sonde Bus

The core of this work correspond to the communication interface, which was named as Open Sonde Bus (OSB). This bus aims to solve several issues seen in previous iterations of the project. It provides a standard and robust way to communicate different modules. This has several advantages:

Reduce coupling Defining a frontier that separate one problem into several sub-problems the bus architecture allows those sub-problems to be solved by different teams, using different approaches and strategies, as long as they respect the external interface of the modules. More over, each one of the different pieces of the system can be improved and updated at different paces.

Improve focus As every developer now works in only one sub-system they can concentrate

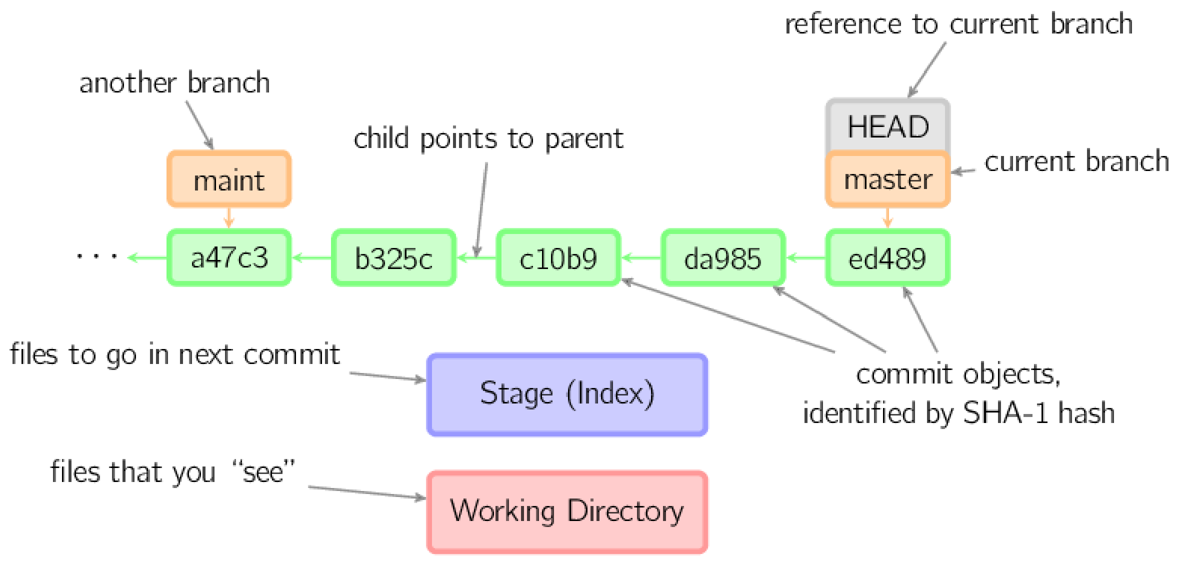


Figure 3.1: Graphical description of the elements of a Git repository. Image taken from "A Visual Git Reference". ©Mark Lodato. CC BY-NC-SA.

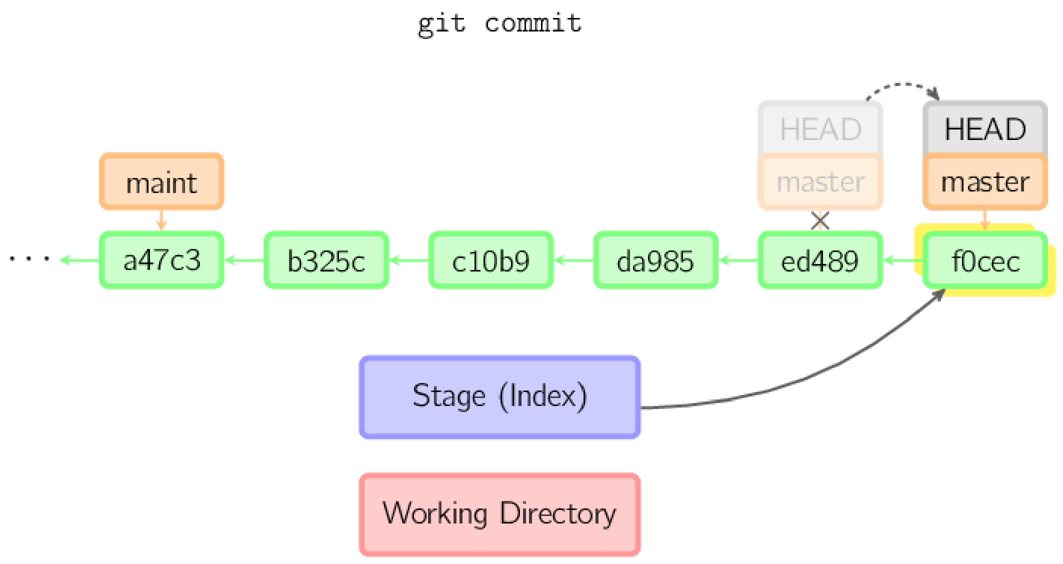


Figure 3.2: Graphical description of a commit operation. Image taken from "A Visual Git Reference". ©Mark Lodato. CC BY-NC-SA.

their efforts into one specific task.

The bus is designed to deal with a variety of common devices. The two main purposes are to provide power and communication to each module. And thus the bus is divided into two sections one for each function. This is inspired from the SATA connectors[26].

The power lines provide +3.3V, +5V and a direct connection to the power source. The first two are regulated, while the last one is not. Modules using custom voltages may use the unregulated power but must provide their own regulation.

The data lines are a standard I^2C bus. These lines are duplicated in 3.3 and 5 volts. Each one of these buses have an independent space address, but some controllers that lack of multiple I^2C controllers may join the two buses. The addressing capabilities of this bus allows several modules to share the same communication lines without interfering with each other. Other protocols such as RS232 are simpler to use but only one pair of devices can be connected simultaneously.

In a nutshell, the I^2C protocol has two lines: a bidirectional pull-up data line, and a master driven pull-up clock line. Each bit is transmitted in the high level of the clock. A special condition signals the start and stop of each frame: pulling the data line low while the clock is high is called a start condition, releasing the data to high while the clock is high signals a stop condition. The bus is idle when both lines are high. On each frame the slave address is transmitted as the first byte. If a slave recognizes its address it acknowledges by pulling the data line low. After that the devices have established a one to one communication and can start transferring the actual data. A complete specification of this bus can be found on the "I2C-bus specification and user manual" [27].

In the Figure 3.3 it is possible to see the distribution of each pin on the bus and its function. On the right side are located the power pins. Each one of the voltages in the bus is provided by two pins. They are separated by groups of three ground connections. From top to bottom the twelve pins of the right side are: two 3.3V pins, three ground pins, two 5V pins, three ground pins, and finally two unregulated power supply pins. On the left side the data pins are located. This side is divided in two, each half operates at a different voltage. On the upper side the bus works at 3.3V level. On the lower side it operates at 5V. Within each section the distribution of eight pins is as follow: one ground pin, two I^2C data pins, two ground pins, two I^2C clock pins and one ground pin.

3.2.2 Power Subsystem

The power subsystem provides energy to the bus. The actual characteristics of this module depends on the final purpose of the system. Or conversely the amount of modules that can be connected to the bus depends on how much energy the power module can provide. In the case of this project, and as a first iteration, the power module is comprised of two fixed voltage linear regulators. This module can provide each up to one ampere of current. That capacity is more that enough to power one hundred microcontrollers. For example an ATmega168 consumes 4mA at 5V running at 8Mhz [28]. One ampere can also power 35,000

Sensirion (typical consume: 28μA), 200,000 BMP085 (typical consume: 5μA) or 26 GPS receivers (typical consume: 38mA).

The schematic of this module can be seen in Figure 3.4. The circuit for the power regulators was taken from the Typical Applications section on the LM7805 datasheet [29]. When being used as Fixed Output Regulator the regulator should have a 0.33μF capacitor between ground and input. That capacitor works by decoupling the regulator from the source of power and is required when the regulator is located at a considerable distance from the power source. The capacitor between ground and output is 0.1μF and helps improve stability and transient response. The module layout is shown in Figure 3.5.

3.2.3 Control Subsystem

The Control Subsystem is responsible for interacting with the other modules to make the system work. For example if the purpose of the system is to pilot an UAV then this module should read the gyroscope and accelerometers attached to it, then estimate the attitude, and finally apply the control signal to a module driving the servo motors. In the case of this project, the task of the control module is to poll the different sensors attached to it, encode that information and transmit it through the communication module.

This module consists mainly of an Arduino micro-controller. As most of the operations mentioned before can be directly translated to operations on the I^2C bus, the only external data connections of the micro-controller are to the I^2C pins of the OSB. As the OSB has two different sets of I^2C lines, one operating at 3.3V and another at 5V, it is necessary to connect them both using a voltage level shifter. Usually this module will be the only master operating on the bus, and so, it has the responsibility to initiate all communications.

As all the operations of the modules are very similar an interface was created to make access to the modules more simple. This interface implies completing three functions. `DeviceAddress` defines the 7-bit address of the slave. `SetUp` is called once at the beginning of the program and must initialize the slave. `Poll` requests information from the slave and sends it though a `Print` interface that is passed as a parameter. This interface is implemented in the class `Driver`. This class implements some methods that help access the I^2C devices such as `WriteChar`, `ReadChar`, `ReadWord` and `ReadBuffer`. The `Poll` function is expected to output the information enclosing every frame in angle brackets(`<>`). Inside a list of comma separated values where the first one corresponds to the slave device address, and the other depend on the device. For example if the device `0x29` wants to output the values 1 and 2, it should print `<29,1,2>`.

As an example of this functionality let's show how to read the accelerations measured by an ITG3200 gyroscope [30]. First the driver must inherit from the Driver class (line 3 of the next listing) and define the device slave address as shown on line 5.

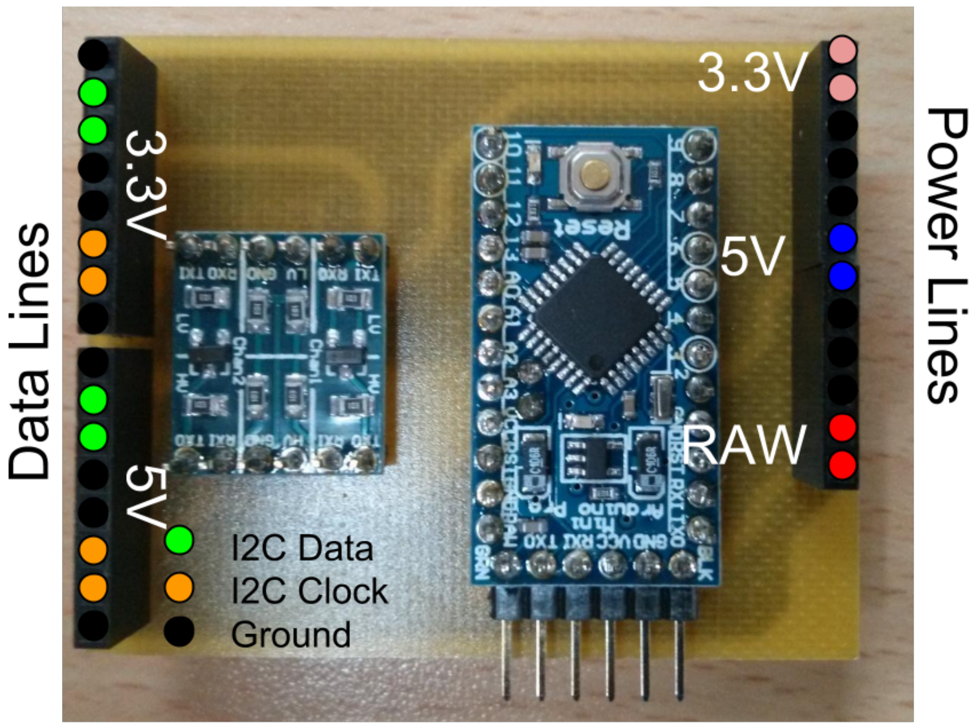


Figure 3.3: Description of the pins on the Open Sonde Bus.

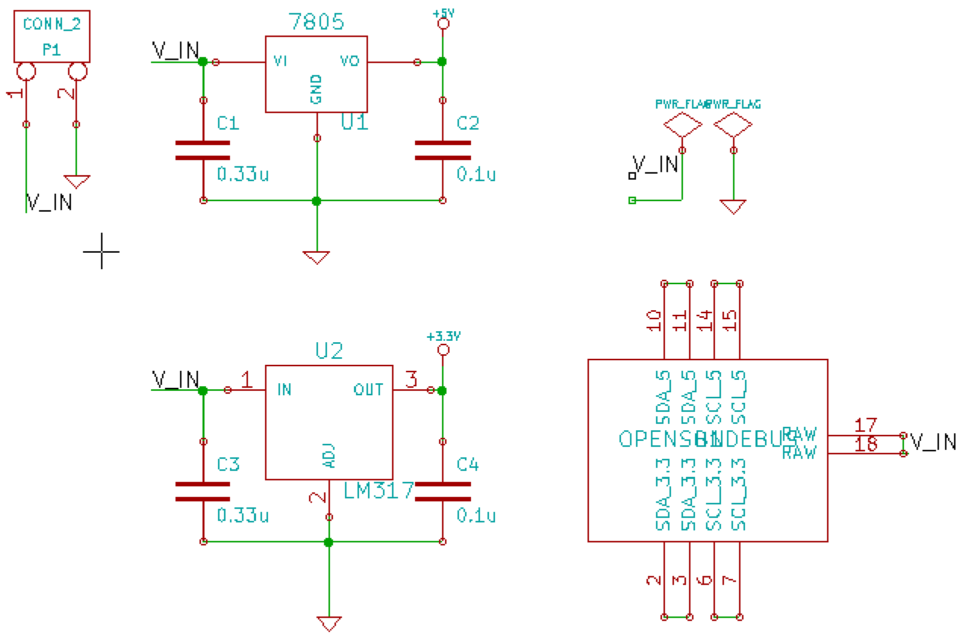


Figure 3.4: Schematic of the power module.

Listing 3.1: Header for device driver.

```
#include "Driver.h"

class ITG3200Driver : public Driver {
public:
  virtual uint8_t DeviceAddress() { return 0x69; };
  virtual void SetUp();
  virtual void Poll(Print &data_transmitter);
private:
  // Configurations
  const uint8_t FS_SEL = 3;
  const uint8_t DLPF_CFG = 3;
  // Register addresses
  const uint8_t DLPF =  0x16;
  const uint8_t GYRO_XOUT_H = 0x1D;
  const uint8_t GYRO_YOUT_H = 0x1F;
  const uint8_t GYRO_ZOUT_H = 0x21;
}
```

It is possible to see that that the constants defined in the device datasheet are stored here as constant variables. The same is done with the configuration variables, so that all parameters of the module can be modified from a single place in the driver. Then the driver must define the `SetUp` and `Poll` methods. The first one writes registers that configure the gyroscope and make it start sensing. The second one simply reads the registers with the values of the acceleration and outputs that information with the format described before. The implementation of both methods is shown in the following listing:

Listing 3.2: Implementation for device driver.

```
#include "ITG3200.h"

void ITG3200Driver::SetUp() {
  WriteChar(DLPF, FS_SEL<<3 | DLPF_CFG);
}

void ITG3200Driver::Poll(Print &data_transmitter) {
  uint16_t output_X, output_Y, output_Z;

  output_X = ReadWord(GYRO_XOUT_H);
  output_Y = ReadWord(GYRO_YOUT_H);
  output_Z = ReadWord(GYRO_ZOUT_H);

  data_transmitter.print("<");
  data_transmitter.print(output_X, DEC);
  data_transmitter.print(",");
  data_transmitter.print(output_Y, DEC);
  data_transmitter.print(",");
  data_transmitter.print(output_Z, DEC);
  data_transmitter.println(">");
}
```

3.2.4 Sensing Subsystem

In a typical system, as the one being built in this project, the payload will be composed of different kinds of sensors. In this particular work a sensing module that captures pressure, temperature and humidity has been built as the principal module of the sonde. These sensors correspond to two integrated circuits. One measures atmospheric pressure and the other, called Sensirion, measures temperature and humidity. As can be seen in Figure 3.6 the humidity and temperature sensor is located off board so it can be in direct contact with the atmosphere. The architecture of this module in particular is an example of different ways to interface different sensors with the bus. While the pressure sensors can communicate directly with the I^2C bus, the Sensirion has its own, and very particular, communication protocol and thus requires an intelligent bridge to connect it to the bus. This intercommunication can be achieved using a micro-controller. Moreover this can be implemented in a way that the logic of the module behaves just like a standard Arduino, allowing the developers to quickly port their software onto the module. Simpler controllers can be used as well, but providing a well known software development framework is more attractive for developers already using Arduino.

As can be seen on the bottom left of Figure 3.7, the connection of the pressure sensor is quite trivial. It is only needed to wire the I^2C data and clock pins to the corresponding pins on the bus. The reason behind this is not chance, as most of the sensors in the market and several other circuits communicate directly using this protocol, it seemed an obvious election as the communication protocol for the Open Sonde Bus.

On the right side of Figure 3.7 it is possible to see an ATmega328 connected as an Arduino. The only external circuitry it needs to work is a crystal resonator, with its corresponding capacitors, and a pull-up connection on the reset pin. The rest of the 1-pin connections to the ATmega allow programming the device. Commonly, Arduino uses a self programming method that works over the serial port. To do that Arduinos are shipped with a custom bootloader. As ATmega ICs are not shipped with this bootloader, the module needs to provide access to the In-System Programming (ISP) pins as well. Thus, after assembling the module, the first step is to load the bootloader through the ISP interface. After that the module can be treated like any other Arduino. In our particular example this micro-controller reads the Sensirion information and exposes those values to the I^2C bus.

One of the features of the modules based on a micro-controller is the need to load the firmware that runs on them. The ATmega micro-controllers provide an In-System Programming interface. Using this interface requires the use of external hardware to program the device. The ATmega328 allows one to burn a bootloader onto the device making the external hardware unnecessary after the first load. In the case of this project the ISP interface will be used. A simple Arduino will be used as loader. Under the folder `tools/ArduinoISP` there

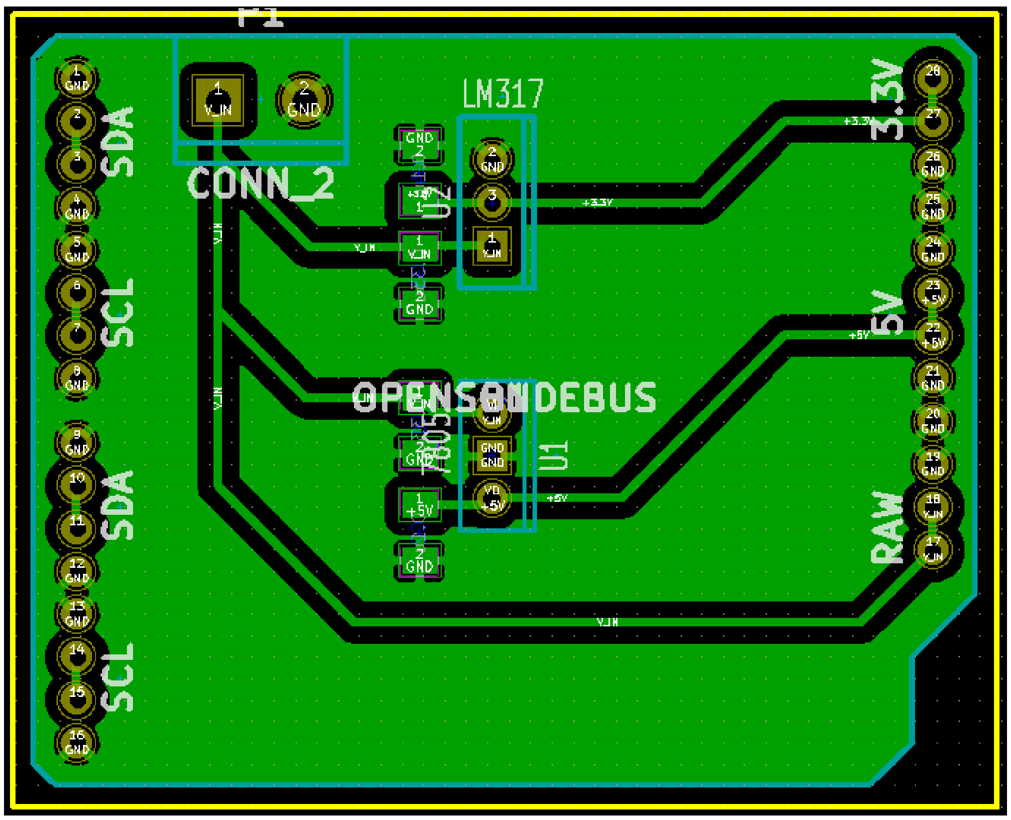
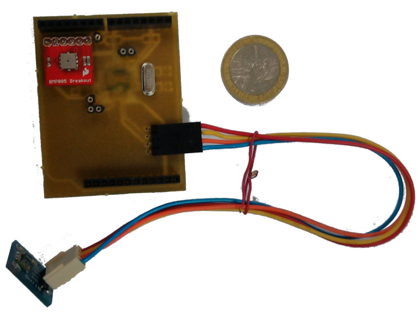


Figure 3.5: Board layout of power module.


(a) Top

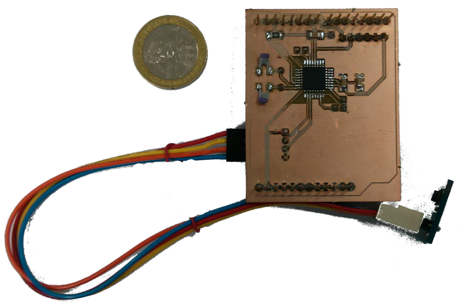

(b) Bottom

Figure 3.6: Pressure, Temperature and Humidity module.

is the firmware that allows the Arduino to program the ATmega using the ISP interface. After that is done the ATmega is connected as shown in Figure 3.8. The detail of the connections is: pin 10 is connected to slave reset, pin 11 to MOSI, pin 12 to MISO and pin 13 to SCK. Then the firmware can be loaded into the micro-controller in the usual way, taking care that in the `Makefile` the parameter `AVRDUDE_ARD_PROGRAMMER` is set to `avrisp` and `AVRDUDE_ARD_BAUDRATE` is set to 19200.

3.2.5 Global Positioning Subsystem

This module provides Global Positioning System (GPS) data to the system. Most GPS systems use a serial interface for communication, so it is necessary to transform that data in a way that can be transmitted using the OSB. The two approaches could be to use a serial to I^{2}C bridge or add a micro-controller to parse the GPS data. The first approach is used in the telecommunication and the second one in the sensing subsystem. For the sake of lightening the load on the main controller the second approach would be used. Besides, the bridges have buffer sizes that are close to the size of a GPS frame, making this approach less suitable. The schematic and PCB of this module are shown in Figure 3.9 and Figure 3.10 respectively.

This modules exposes 4 registers of 4 bytes each. These registers contain the time, latitude, longitude and altitude. Latitude and longitude are signed and the last four digits must be considered as the decimal part of the data. The last three digits of the time correspond to the milliseconds. During each read the master is supposed to read all 16 bytes of information available in the device.

To parse the GPS data into the registers mentioned above a finite state machine (FSM) was implemented. The frame of interest is `GPGGA`. The FSM must: detect the beginning of a data frame, check that the header correspond to the one that must be parsed, and then, using the fields separator, split the fields into individual strings. Once the FSM has finished, the data can be transformed to a signed 4 bytes integer using the C function `atol`. The states of the FSM are: WaitForStart, ReadHeader, ReadDataField and DataOK. The auxiliary variables field_number and field_position are used to split the data fields. The FSM can be visualized in Figure 3.11.

3.2.6 Telecommunication Subsystem

This module provides radio communications with a ground station providing real time tracking and monitoring of the system. This module is based on a custom HAC-LM [31] transmitter. The HAC-LM is a bidirectional transceiver. It transform a serial stream of data into an electromagnetic signal that can be propagated through space. These custom transmitters are designed to work with a 9V power supply, thus the board must provide its own voltage regulation. The transmitted power corresponds to 30dBm or 1 Watt. As the HAC-LM interface is serial it is necessary to add a I^{2}C-UART bridge to connect the device to the bus. The final assembly of this module can be seen in Figure 3.12.

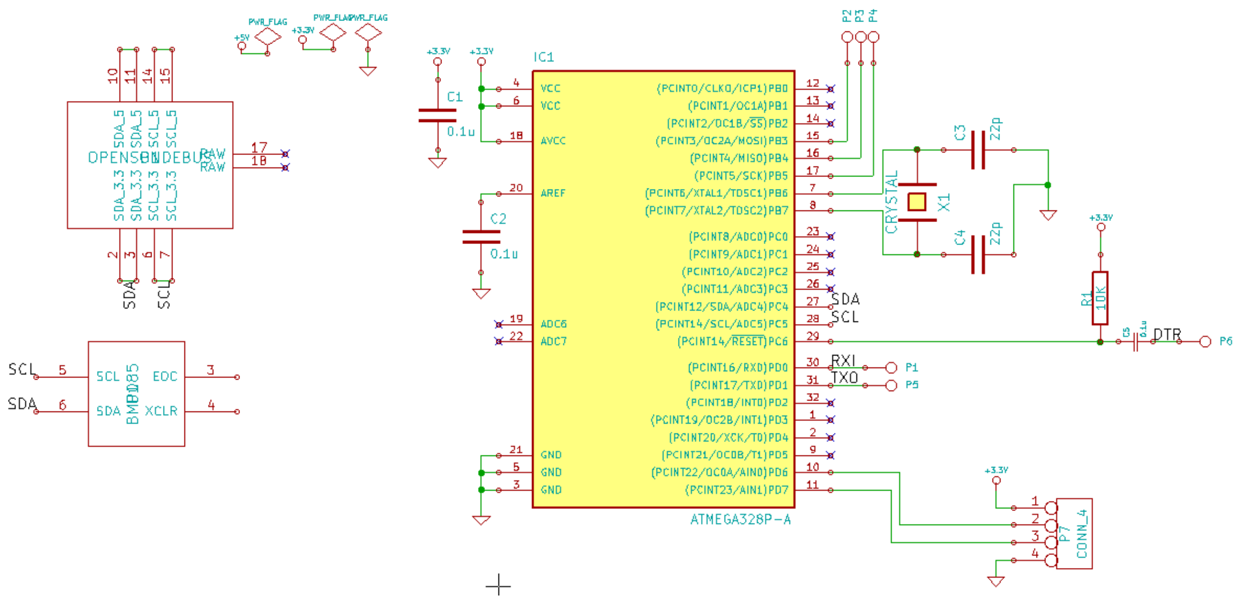


Figure 3.7: Schematic of sensing module.

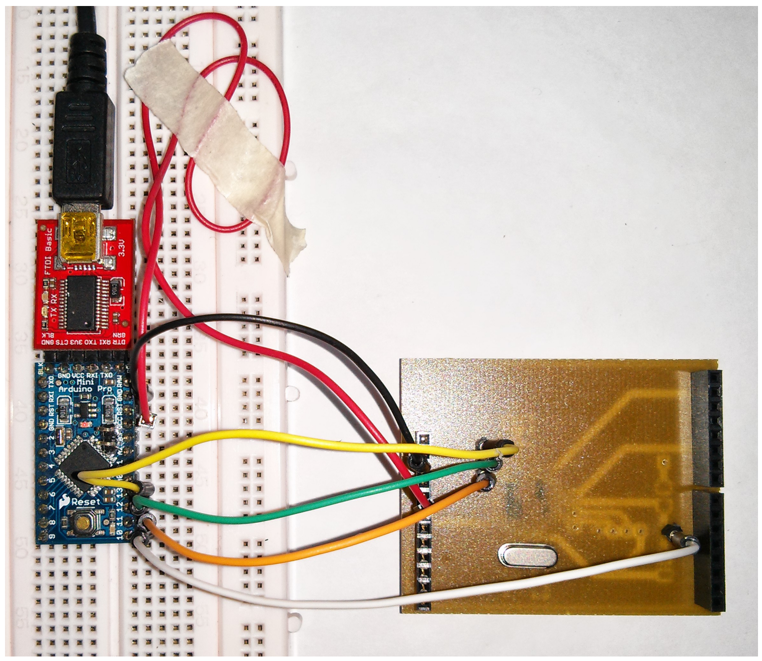


Figure 3.8: Connections to upload the firmware to the micro-controller.

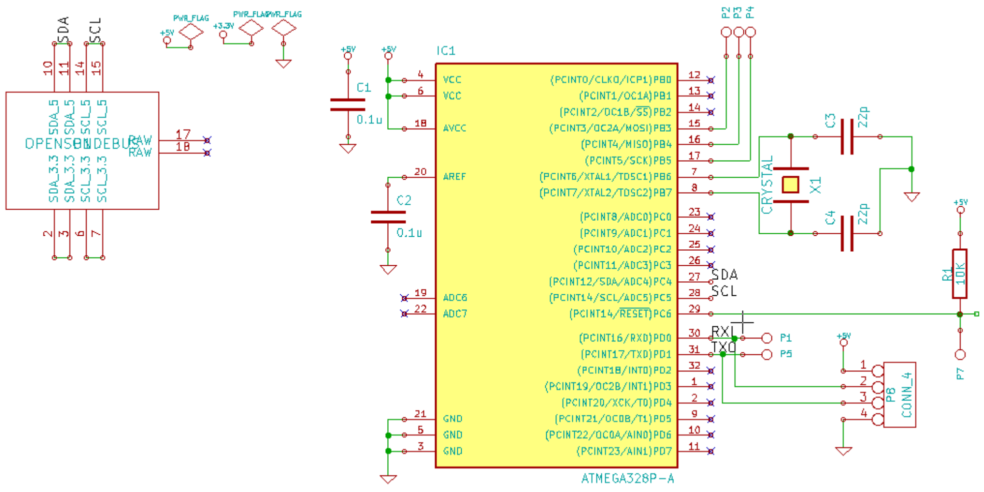


Figure 3.9: Schematic of GPS module.

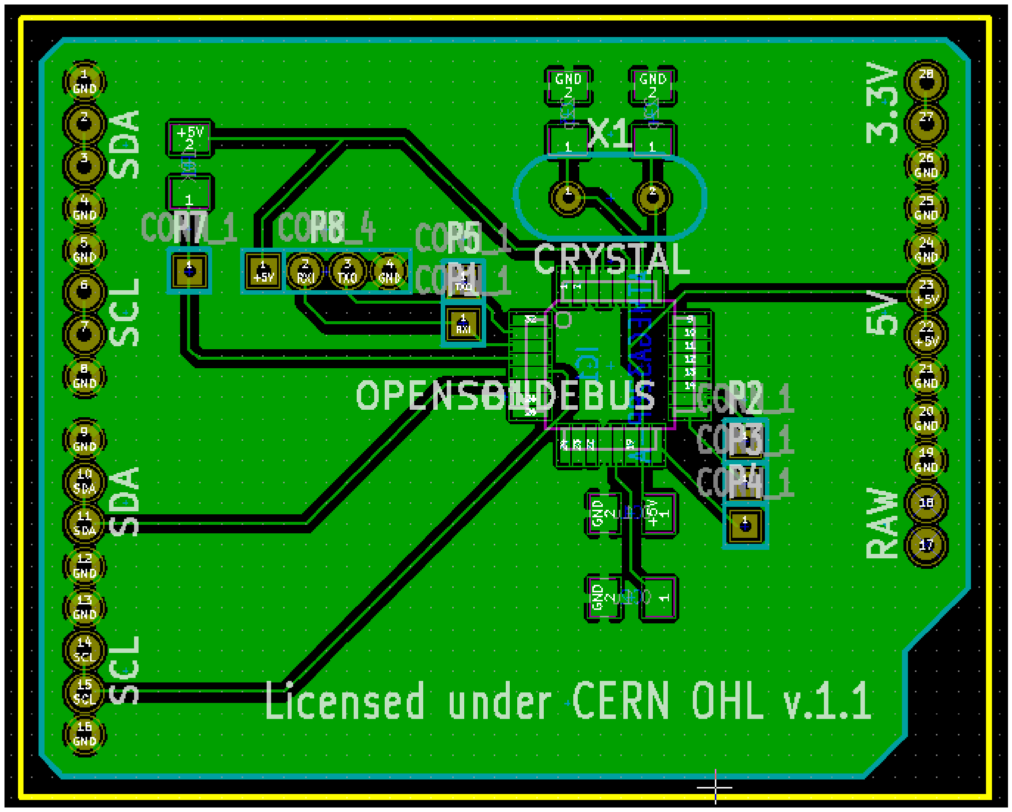


Figure 3.10: Board layout of GPS module.

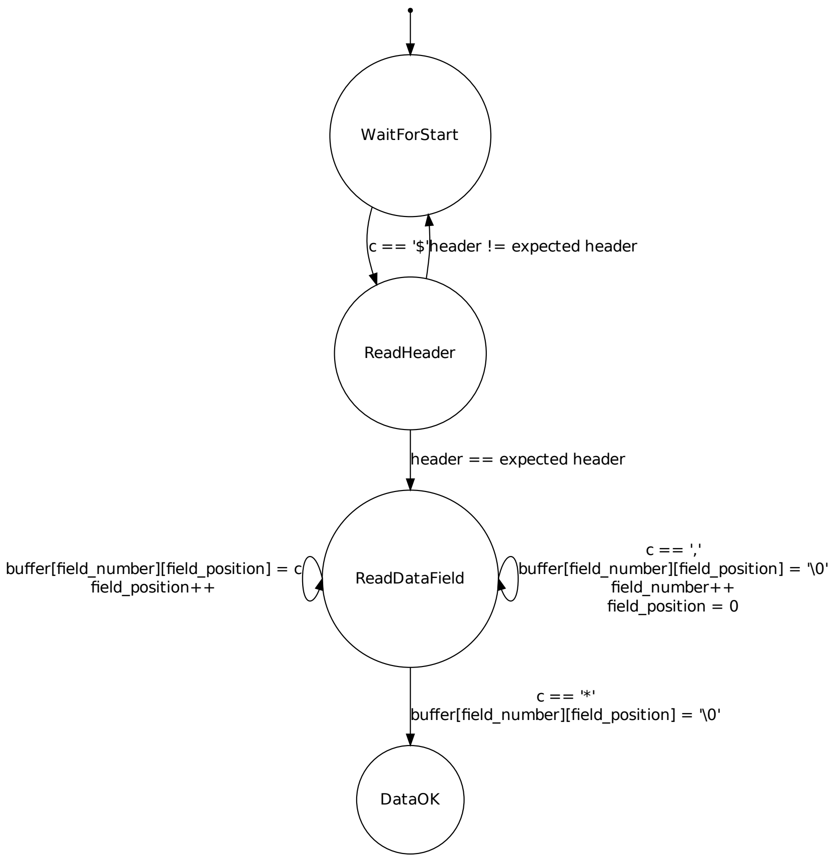

WaitForStart
c == '$'
header != expected header
ReadHeader
header == expected header
ReadDataField
buffer[field_number][field_position] = c
field_position++
c == ','
buffer[field_number][field_position] = '\0'
field_number++
field_position = 0
c == '*'
buffer[field_number][field_position] = '\0'
DataOK

Figure 3.11: FSM for parsing GPS data.

The module as shown in Figure 3.13 is divided in 3 blocks. First, on the upper left, there is a power regulator. This is an adjustable power regulator, and the output voltage is defined by the voltage divider located to its right. The equation relating the input and output voltages with the resistors is:

$$V_{out} = V_{ref}\left(1 + \frac{R1}{R2}\right)$$

Considering V_{ref} as 1.240V and a target output voltage of 9V the ideal ratio of R1 to R2 is 6.2581. As not all values of resistors are available in the market, this ratio is approximated by the sum of 56k and 7.5k resistors as R1, and a 10k resistor as R2. The ratio for this network is 6.350, 1.5% off from the ideal value. Anyway this value is not critical as V_{ref} also change from device to device in the range of ±0.5V.

The second block of the module corresponds to the I^{2}C-UART bridge. The circuit SC16IS752 is used as a bridge. It only needs a few external components to operate. Most notably it must have an oscillator circuit to generate the timing for the serial output. This circuit is composed of a crystal oscillator and two capacitors. The value of the crystal is not critical as it is prescaled inside the device, but larger values can provide more accuracy at the baud rates the module is working on. The HAC-LM communicates at 1200 bps. To achieve that baud rate with a 16Mhz crystal as the design uses, the internal prescaler of the bridge needs to be set to 833. This will generate a baud rate of 1200.5 (less than 0.05% of error).

Finally, the last block is the HAC-LM transmitter itself. This block is connected directly to the 9V voltage and the output of the I^{2}C-UART bridge. Even though the control port of the HAC-LM uses TTL and the I^{2}C-UART bridge works on 3.3V, there is no need to add a voltage level shifter, and the two blocks can be connected directly. When designing the layout of the module it is necessary to place decoupling capacitors very close to the power pins of the HAC-LM. As it consumes large bursts of current when transmitting, an incorrect decoupling can induce noise to the power of other modules generating spurious resets on other ICs in the system.

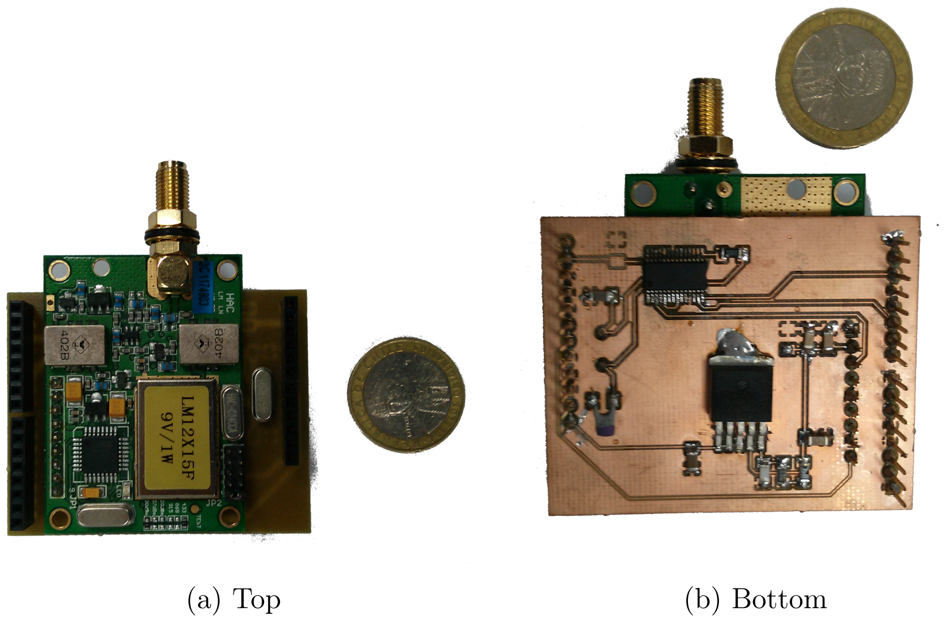


Figure 3.12: Telecommunication module.

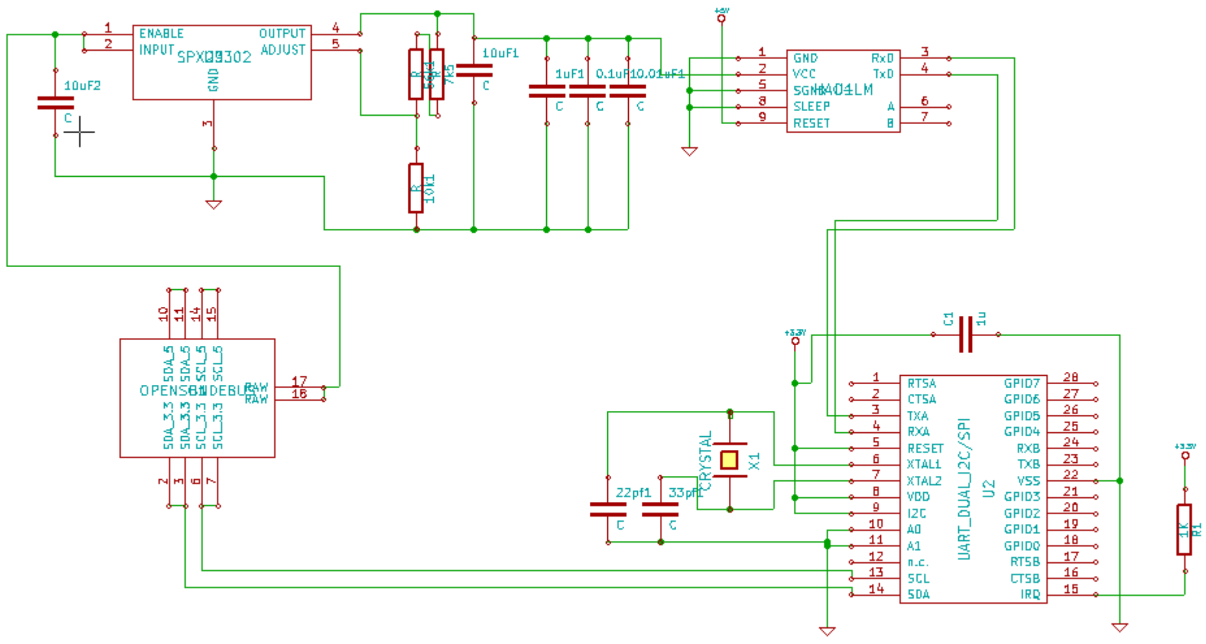


Figure 3.13: Schematic of telecommunication module.

3.2.7 Summary of Parts

As a summary of the parts and sensors used in the project the following table state their main characteristics.

Name	Description	Part number	Vendor/ manufacturer	Variable/ function	Measuring/ operating range	Error	Response time/ sampling frequency	Value ($)
Humidity and temperature digital sensor	Capacitive sensor	SHT10	Sensirion	Humidity	0%-100 % RH	±4.5%	8 s	15
Humidity and temperature digital sensor	Band gap sensor	SHT10	Sensirion	Temperature	-40^0 to 124^0C	$\pm 0.5^0C$	5 to 30 s	15
Digital pressure sensor	Piezo-resistive pressure sensor	BMP085	BOSCH	Pressure	300 - 1100 hPa	0.03 hPa	128 Hz	20
GPS module	GPS Sirf Star III module with internal patch antenna	MOD-GPS	Olimex	Wind, position	-40^0 to 85^0C (<18-km height)	1-3 m/s (at 10 m/s)	<1 s	50
Communication transceiver	Data radio module	HAC-LM12	Shenzen HAC Technology	Radio transmission	402-405 MHz			50

3.3 Licensing

One of the fundamental characteristics of this project is that it must be released as open hardware. As described in the previous chapter this may have several advantages. It fosters collaboration from the community around the globe, creates new business opportunities and increases clients' trust in the product. Although from the management of the project this is a straightforward decision, the implementation is not so easy. As it happens in the software world there are many different free licenses. Each one of these has its particular details. In this section some of the most prominent licenses are described and finally a decision is made about which one adapts bests to the needs of this project.

3.3.1 Open Licenses Description

GNU General Public License

The GNU General Public License [32] (GPL) was created by the Free Software Foundation (FSF). It aims to protect the four freedoms of the user of *free* software. Those four freedoms are freedom to: run the software, study how it works and change it, redistribute it and distribute copies of modified versions. To fulfill those freedoms it is necessary that the user has access to the source code and documentation of the program. The strongest restriction for other developers using GPL software is that this license requires software that links to GPL libraries to be licensed under GPL itself. This may be a problem in a business model where the software being produced is not free, since it cannot be linked to any GPL software. An extensive list of software is licensed under GPL. Some notable examples are the Linux Kernel, GCC and KDE.

GNU Lesser General Public License

The GNU Lesser General Public License [33] (LGPL) is quite similar to GPL, but it is weaker in term of restrictions to use LGPL software. While GPL does not allow libraries to be linked by non-GPL software, LGPL release this restriction allowing non free software to link free libraries. This allow developer of proprietary software to use free software. LGPL still requires that changes made to the free code must be made public. Examples of software under LGPL includes mainly libraries like Qt and libssh.

Creative Commons

Creative Common (CC) is a license that allow users to give some of the rights of copyrighted work. Creative Commons is supported by a non-profit organization of the same name [34]. They provide guidance, documentation and tools to use this license. CC is a meta-license and the actual license depends on which rights the licensor wants to grant. Those are:

Attribution Whether redistribution of the work must attribute the work to the author.

Share Alike Whether derivatives of the work must be done under the same license or not.

Noncommercial Whether the redistribution of the work is allow under commercial use.

No Derivative Works Whether it is allowed to create derived work from the one that was licensed.

Licensors can select which rights to grant or not. But there are six common types of licenses: Attribution , Attribution-ShareAlike, Attribution-NoDerivs, Attribution-NonCommercial, Attribution-NonCommercial-ShareAlike and Attribution-NonCommercial-NoDerivs.

While GPL and LGPL are applicable to software work only, CC can apply to any work that can be copyrighted. It is also more flexible and simple to use.

Wikipedia and OpenStreetMap stand as two examples of collaborative work that is licensed under Creative Commons. There are several content sites where people upload their work where the CC can be easily applied like Flickr, Picasa and YouTube. In the area of open hardware, the design files of the Arduino project are also under CC.

Public Domain

Public Domain corresponds to works that are of public knowledge and widely available. This can happen when copyrights are expired, for example like the work of old composers and writers. After a work is in the public domain the authors do not have any ownership of it.

Copyright laws extend automatically to the author of a work whether he wants those rights or not. To allow individuals to easily give away those rights and put their work in the public domain there is a license called CC0 [35], also from the Creative Commons foundation. Licensing under CC0 will effectively give away all the rights the author has over their work, and cannot be considered the owner of the work anymore. This license also extends beyond software.

CERN Open Hardware Licence (OHL)

The CERN Open Hardware Licence (OHL) [36] was created by the European Organization for Nuclear Research on 2011 and is quite different from the previously discussed since it has been specifically designed for hardware. The text of the license defines the documentation as the schematics and diagrams that describe the design. This is the equivalent to the source code in the case of software. But it also defines the product as the materialization of the documentation and grants permission to the licensee to manufacture and distribute them, provided that the original documentation and the license is made available with the product. The modification of the documentation, i.e. creating a derived work, can be done only if that work is licensed under the same terms. This license does not apply to any firmware that is included into the hardware.

Summary

The next table is a summary of the characteristics of the licenses discussed so far. An asterisk (*) indicates that the license has the feature. A space indicates that it does not. A - indicates that the characteristic does not apply, e.g. how to license derived work when no derived work is allowed.

License	Appies to Software	Applies to Hardware	Allows derived work	Derived work must be open	Allows commercial use
GPL	*		*	*	*
LGPL	*		*	*	*
CC Attribution	*	*	*		*
CC Attribution-ShareAlike	*	*	*	*	*
CC Attribution - NoDerivs	*	*		-	*
CC Attribution - NonCommercial	*	*	*		
CC Attribution - NonCommercial - ShareAlike	*	*	*	*	
CC Attribution - NonCommercial - NoDerivs	*	*		-	
Public Domain	*	*	*		*
CERN Open Hardware		*	*	*	*

3.3.2 License Selection and Licensing

This project involves both software and hardware. So, considering the coverage of the licenses discussed before there are two main approach: select one license that cover hardware and software altogether or select one for each one. In the first case the election should be Creative Commons. Nevertheless, the CERN Open Hardware license is quite more specific to hardware and completely matches the desires of the developers of the project. Using it would require another license for the software. In that respect GPL is the one that closest matches the intention of the project, providing that any public modification to both hardware and software will be freely made available to the community. For those reasons the licensing approach selected for this project is a combination of GPL and CERN Open Hardware licenses.

The process for licensing a design under CERN OHL is described in [37]. In summary this documents states that the following files must be included in the design:

LICENSE.PDF The text of the CERN OHL itself.

CONTRIB.TXT Point of contact of the licensors who want to receive modifications to the design.

PRODUCT.TXT Point of contact of people who wish to receive information about manufactured products.

CHANGES.TXT A list of changes made by the licensee.

Additionally it is necessary to include design files and a notice indicating that the design is licensed under CERN OHL. This text must be included into the schematic and board files. In this project it was added as a text box. On the board file it was placed in the `comments` layer, for it not to interfere in the manufacturing. The text box is visible when editing the file on KiCAD, but as it saves the files in ASCII format, the notice is also readable when opening the file as plain text. The actual notice included is the following:

```
Copyright UNIVERSIDAD DE CHILE 2013.

This documentation describes Open Hardware and is licensed under the
CERN OHL v. 1.1 or later.

You may redistribute and modify this documentation under the terms of the
CERN OHL v.1.1. (http://ohwr.org/cernohl). This documentation is distributed
WITHOUT ANY EXPRESS OR IMPLIED WARRANTY, INCLUDING OF
MERCHANTABILITY, SATISFACTORY QUALITY AND FITNESS FOR A
PARTICULAR PURPOSE. Please see the CERN OHL v.1.1 for applicable
conditions
```

The last step was to include in the silkscreen of the PCB the notice `Licensed under CERN OHL v.1.1`

To license the source code under GPL the best way, as describe by the GPL license, is to add a notice to every source file including: a one line description of the program, a line describing who has the copyright, a copying permission statement and a disclaimer of liability. The full notice is shown bellow:

```
/*
 * This file is part of the Ckelinar Project.
 *
 * Copyright (C) 2013  UNIVERSIDAD DE CHILE.
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
```

A copy of the license text is required to be distributed with the program. In this case it was located under `license/gpl/gpl.txt`.

Chapter 4

Results

4.1 Power Module

One of the main requisites for the power module is to maintain a fixed voltage independent of the load on the system. A standard way to visualize how the power supply behaves with different loads is to draw the relationship of the current that the source is supplying and the voltage on the load. The result of this test is shown in Figure 4.1. The regulators can handle up to 1 ampere but due to the lack of heat sink in design it is not recommended to reach that value, since when running on 11.1V batteries the dissipated power would be 6.1 Watts, much more than can be handled by the device in such conditions. During the test the current limit was set at 0.5 ampere. This point of operation is not intended to be used continuously but in burst only. The maximum voltage deviation from the nominal value is less than 1% for the 5V output and around 3.5% for the 3.3V output.

4.2 Controller Module

To test the functionality of the micro-controller the main requirement is the ability to communicate over the I^2C lines on the bus. The experiment to test this consists of communicating with one of the most simple devices that communicates over I^2C, an EEPROM memory. The IC used for testing is 24LC04B, a 4 Kbit EEPROM memory. The write operation is completed on a single I^2C frame. The master sends the device address with the write bit set to zero and then the memory address and the content to be written. The slave responds acknowledging every byte. When writing the master sends two commands. The first one correspond to the two first parts of the writing command i.e. the device address and the memory address. And the second command correspond to the device address with the write bit set to one and then the memory answers with the content of that memory. The setup for this experiment is shown in Figure 4.2. The memory is powered from the bus, and the data and clock lines are connected to the corresponding pins on the memory.

In Figure 4.3 it is possible to see the data and clock signals when the master is writing into the slave. It is possible to see the start condition at the very beginning (the data going down while the clock is up) and the stop condition at the end (the data going up while the clock is up) as well as the acknowledgments from the slave on the 9th, 18th and 27th pulses of clock. In Figure 4.4 it is possible to see the two commands needed to read the memory. On the left the master sends to the slave the address it wishes to read. On the right the slave sends the information stored at that location. From this signal it is possible to see that the distance from edge to edge in the clock signal is 10μ seconds. That corresponds to the 100 kHz frequency of the bus. It is also possible to see that the acknowledges introduce a certain delay that reduces the expected speed of the bus.

Listing 4.1: Code for I^{2}C communication test.

```
#define MAGIC_NUMBER 0xA5

MemoryTest::MemoryTest(uint8_t addr)
  : addr_ (addr)
{
}

uint8_t MemoryTest::DeviceAddress() {
  return 0x50;
}

void MemoryTest::SetUp() {
  WriteChar(addr_, MAGIC_NUMBER);
}

void MemoryTest::Poll(Print &data_transmitter) {
  ReadChar(addr_);
}
```

4.3 GPS Module

Once the power and controller modules have been tested the GPS module can be tested at a higher level. Thus, the setup for this test was to connect all three mentioned modules and set the controller to output the data through the serial port instead of the transmitter. On the computer the program BaseCamp was used to parse the data and display it. That way it is easy to visualize whether the GPS is working properly. This approach has been useful before and allowed to spot a difference between the format of the GPS output and the one used by Google Maps and other tools. The resulting map of this test can be seen in Figure 4.5.

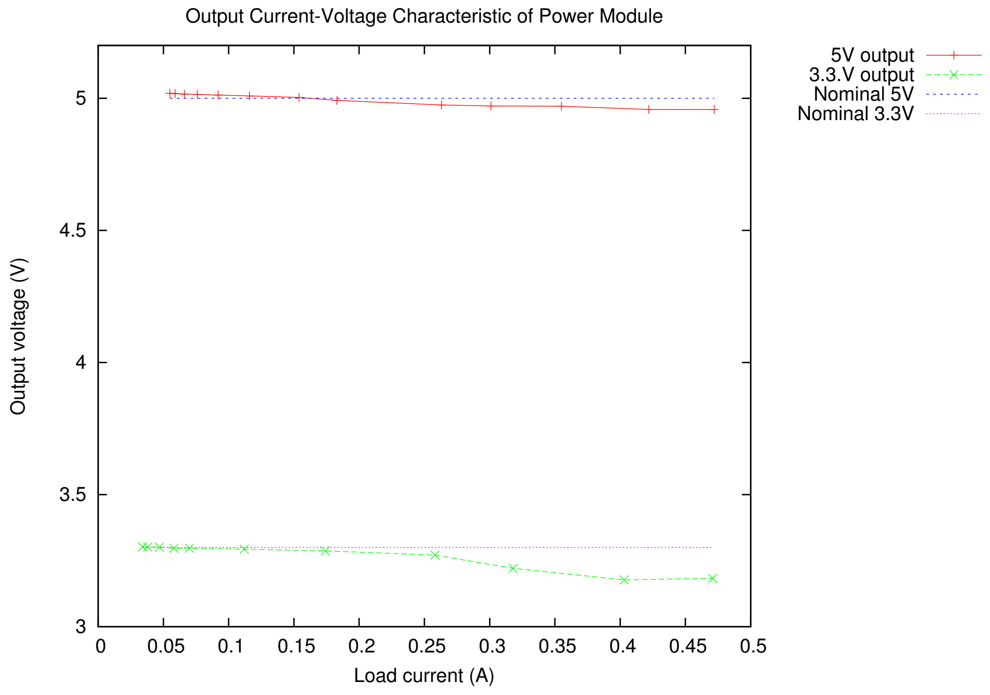


Figure 4.1: Output Current-Voltage Characteristic of Power Module.

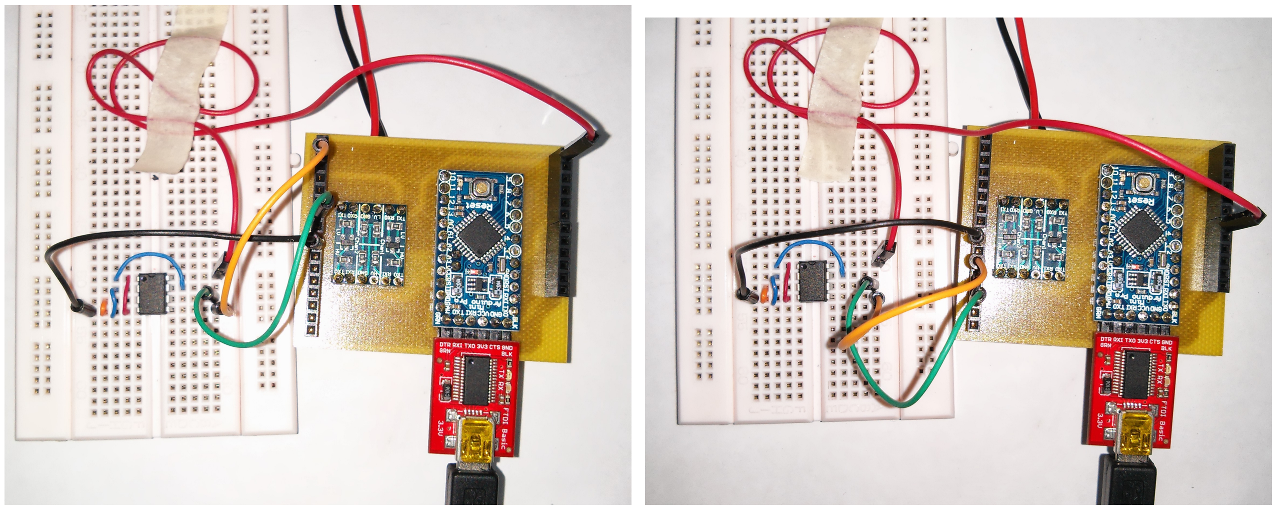


Figure 4.2: Connections for memory test. On the left the connections when operating at 3.3V. On the right when operating on 5V.

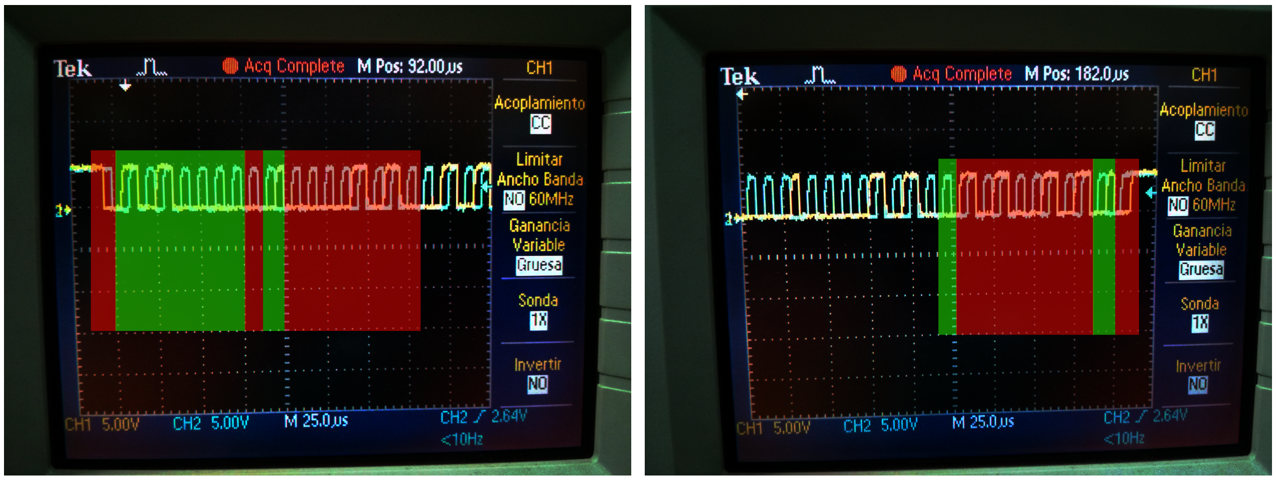


Figure 4.3: Data (yellow) and clock (blue) signals for writing into the memory. The different parts of the signal from left to right are: (i) Start condition, (ii) slave Address, (iii) write bit, (iv) slave acknowledge, (v) register address, (vi) slave acknowledge, (vii) register data (sent by master), (viii) slave acknowledge and (ix) stop condition.

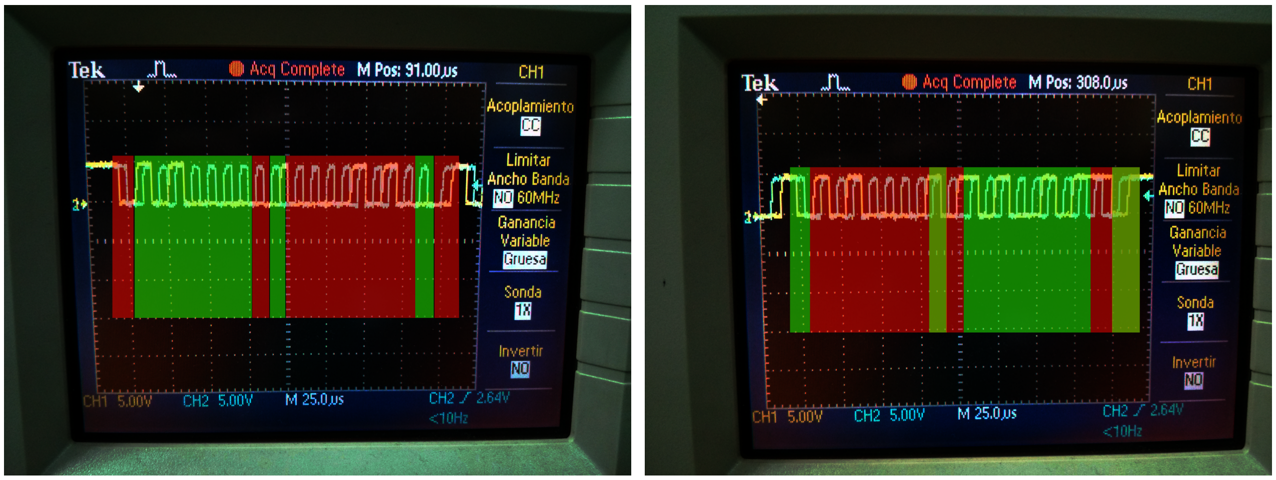


Figure 4.4: Data (yellow) and clock (blue) signals for reading from the memory. The different parts of the signal from left to right are: (i) Start condition, (ii) slave Address, (iii) write bit, (iv) slave acknowledge, (v) register address, (vi) slave acknowledge, (vii) stop condition, (viii) start condition, (ix) slave Address, (x) read bit, (xi) slave acknowledge, (xii) register data (sent by slave), (xiii) master not-acknowledge and (xiv) stop condition.

4.4 Sensing Subsystem

The main sensor of the module, the Sensirion, was tested in the temperature and humidity chamber at Dirección Meteorológica de Chile (DMC)[1] (see Figure 4.6). This chamber can set the temperature on a wide range, and the humidity at temperatures over 11^{o}C. The datasheet of this sensor [38] states that "each SHT1x is individually calibrated in a precision humidity chamber. The calibration coefficients are programmed into an OTP memory on the chip". Thus, this test has the main purpose of validating this claim. Even though the chamber is supposed to set the temperature and humidity to a certain point the actual value of those variables was registered using a Vaisala HMP155 Humidity and Temperature Probe. Several set points were programmed in the chamber and then the temperature and humidity measured by the devices was registered. Due to the dynamics of the sensors those set points should be maintained over a period of time, to allow sensors to respond to the change in the parameters. Unfortunately the slowness of the chamber makes this type of measurement impractical. Furthermore, under 11^{o}C the chamber cannot control the humidity and so this can not be assured to be steady. For this test the data was collected on the transient between the setpoints and the setpoints as well.

The data collected in this test is shown in Figure 4.7. This also include the temperature reading for the BMP085. Although this sensors will not be used to measure temperature in a real application since it is located inside the radiosonde and not exposed to the ambient. From that plot it is possible to see how well the sensors behave when the temperature is constant (from 0s to 1500s and around 5400s and 7000s). It is also possible to see that both BMP085 and Sensirion react faster to temperature changes than standard sensor. In the case of the humidity there are two measurements one is directly obtained from the sensor and the other (compensated) adds a component to the former depending on the temperature. From the plots it is also possible to see how the behavior of that signal depends on the stability of the temperature. Around 1000s it is possible to see how the humidity rises while the temperature remain constant. Between 2000s and 3000s the behavior is quite different when the humidity changing in the same direction and approximately the same temperature, and at the same time the temperature is changing.

The other experiment performed was to expose the sensor to an abrupt change in temperature and see how it responds to that change. The sonde was placed inside a conventional freezer at -8^{o}C until it reached equilibrium and then suddenly exposed to an ambient temperature of 9^{o}C. The data collected in that experiment can be seen in Figure 4.8. Here it was assumed that the response from the sensor would be exponential and then an exponential curve was fitted onto the data. The equation of the fitted data corresponds to $9.31 - 17.36\mathrm{e}^{-0.011259902t}$ which corresponds to a time constant of 88.81 seconds. This compares badly to the information provided by the datasheet. The manufacturer states that the time constant fluctuates between 5 to 30 seconds. The value found on the experiment for the response time is more than twice the nominal value.

[1]`http://www.meteochile.gob.cl/`

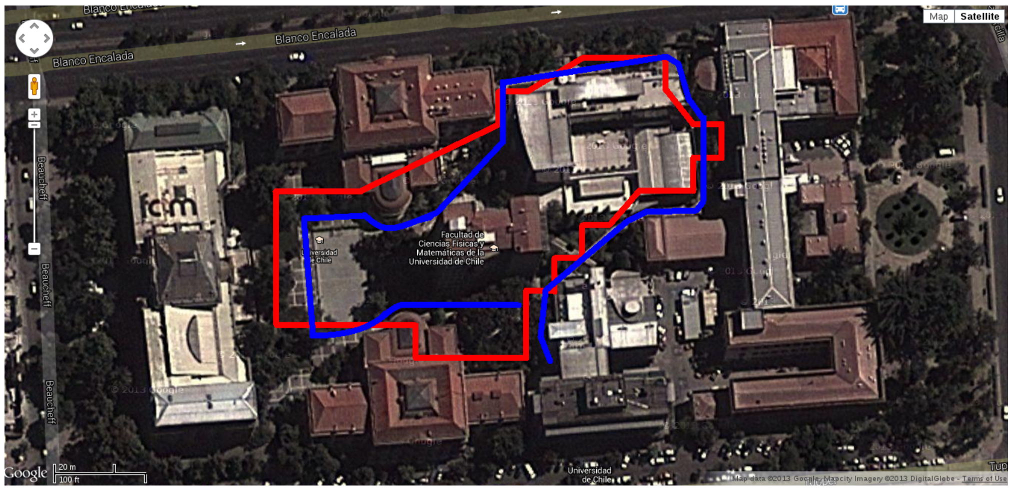

Blanco Encalada
Beauchef
Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile
Universidad de Chile
Map
Satellite
20 m
100 ft

Figure 4.5: Map of the GPS output during a walk on the university campus. Blue line indicates real path. Red line indicates the GPS position.

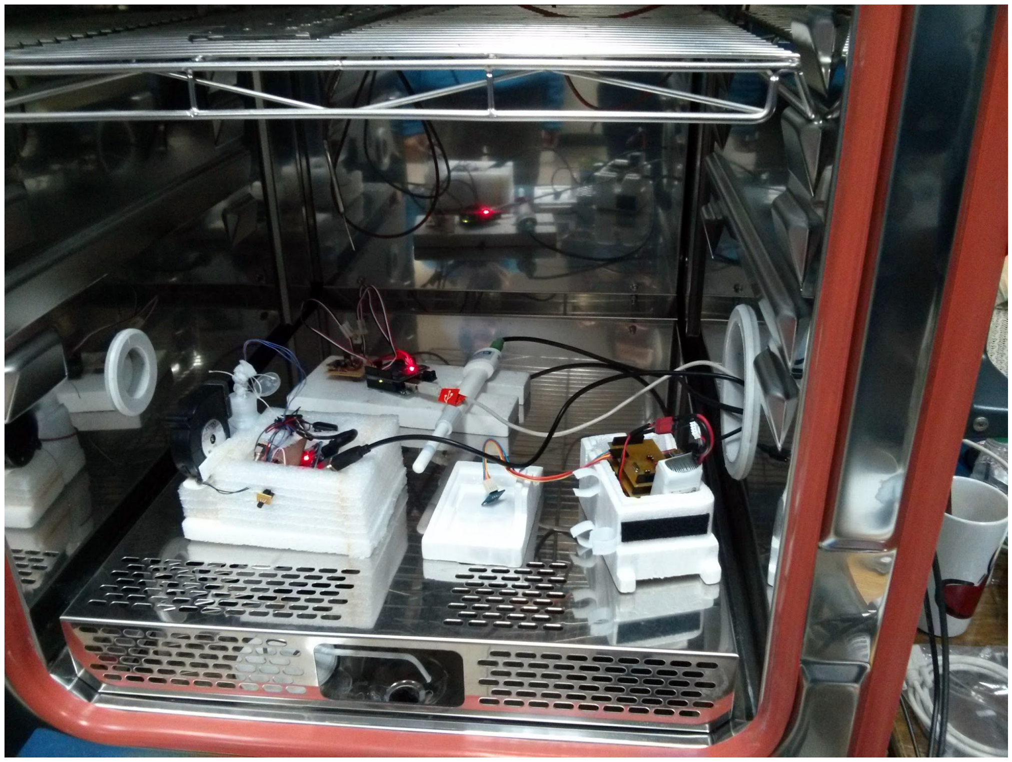


Figure 4.6: Photo of radiosonde inside the temperature and humidity chamber.

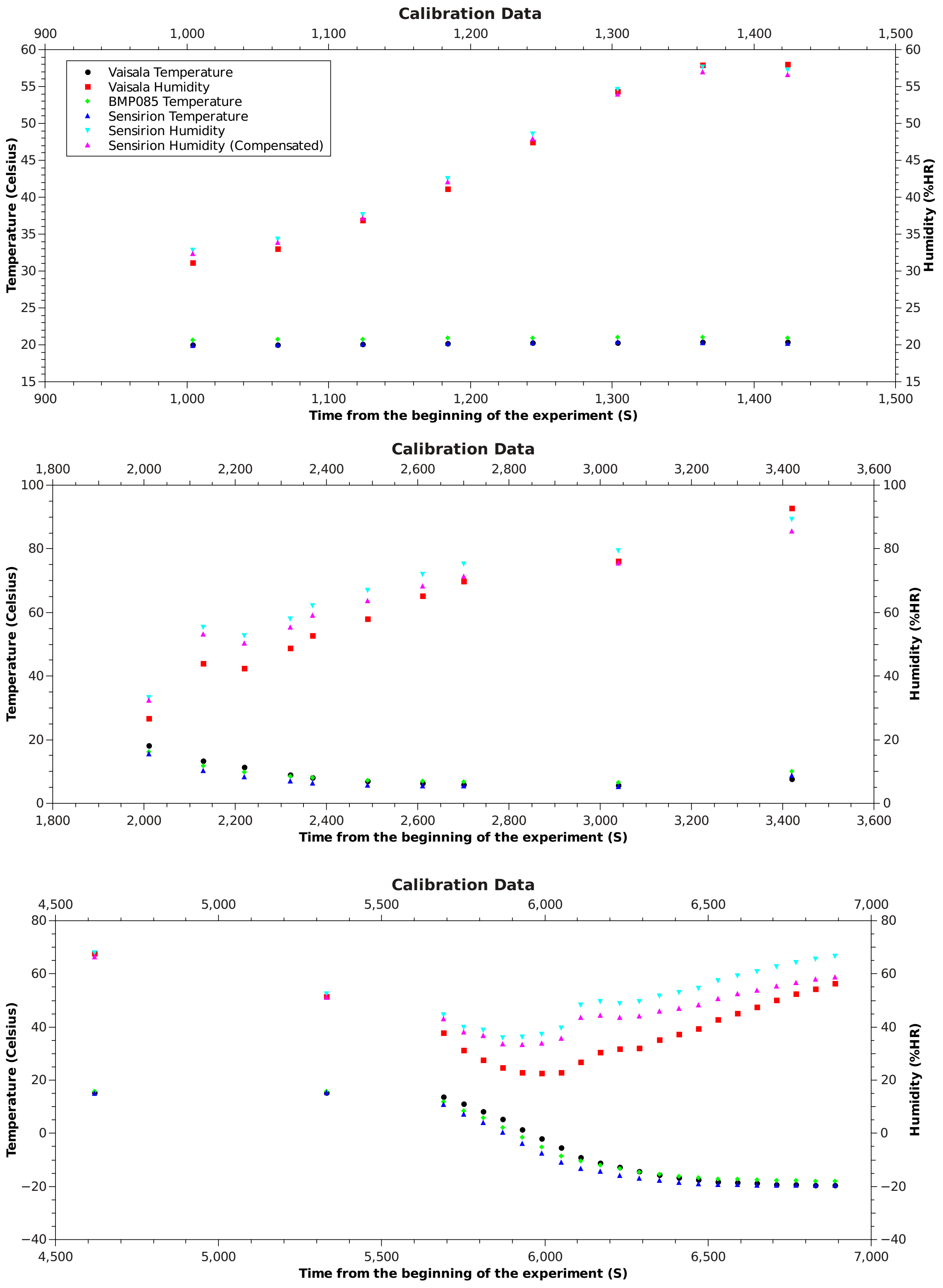


Figure 4.7: Data obtained from temperature-humidity test.

4.5 Telecommunication Module

To test this module independently from the others it was decided to transmit a fixed text every time the radiosonde was initiated. Using this fixed test the receiver can compare it to the actual data received using the Levenshtein distance. This seems the most straightforward way to measure the quality of the communication since it takes into account modified characters as well as missing ones. Missing characters do not allow to compare character by character. Consider for example a case when the first character is missing, comparing the expected text with the actual text will produce an error equal to length of the test data. A better measurement of the error should produce an error equals to one, since there is only one character that was not transmitted. The last result is the one provided by the Levenshtein distance.

This test was performed by locating the base station on the roof of the building of Electrical Engineering of the University of Chile (33°27′29.28″S, 70°39′42.31″W). The radiosonde was positioned at various spots in Parque O'Higgins where there was line of sight with the base station. The location of those points as well as the base station are shown in Figure 4.9.

Point	Distance [m]	Error rate [%]
1	153	0
2	197	0
3	346	0
4	446	0
5	458	0
6	655	4.35
7	722	0

4.6 Field Testing

To test the whole system performance under actual working conditions a campaign with a tethered balloon was carried out. Using a tethered balloon allows the radiosonde to ascend and descend several times without loosing the payload and with a minimal consumption of helium. The campaign was performed with the help of a team of researchers from the Department of Geophysics. The setup for the test included the device described in this document as well as their equipment. Immediately under the balloon there was located an aethalometer (for measuring black carbon), the device under test, and an Air radiosonde, used to provide ground truth data as shown on Figure 4.10. The launching site for this campaign was the DMC (33°26′42″S, 70°40′58.8″W) during August 27-28. Simultaneously to the several captive flights, DMC launched free soundings twice a day on both days.

As a result of the test the radiosonde completed three successful flights with transmission of data back to the base station. Unfortunately, due to a fault at the logging of the Air radiosonde, only one of them can be compared against ground truth data. However, from the standpoint of the operation of the system, it was already a success to obtain data. That proved that all basic modules as power, controller and transmission were functioning appropriately. The temperature vertical profiles can be seen in Figure 4.11. It is possible to see that the first sounding data is very noisy. A possible explanation for that can be due to the exposition of the sensor to direct sunlight. That was solved for the second day protecting the sensor into a tubular aluminum casing. For the flight with ground truth data there was also a flight from DMC at approximately the same time, so it is possible to compare the three records. The data for this comparison is shown in Figure 4.12. It is possible to see that the series stay within the same range of 3 degrees most of the time. But that glitches of the data of the device under test makes this difference large (up to 5 degrees) sometimes. It is necessary further analysis to detect the origin of those errors. A possibility is that the casing of the sensor restricted air flow into the sensor. On the Air radiosonde the temperature sensor casing aligns in the direction of the wind, so the air flows freely into the sensor. On the radiosonde under test the casing pointed vertically. This problem may have increased due to the slow rising speed and some stops of the sounding to add more probes along the cable holding the balloon. Those stops were made when the balloon was at 930 and 900 hPa, which correspond to the spikes in the data.

Regarding the working of the GPS module, there was no ground truth information for this variable. However the GPS locks was acquired successfully after every flight, condition that is indicated by a blinking led in the GPS antenna. From the plot of the data of the position of the radiosonde, Figure 4.13, it can be seen that the unit provided smooth positions, except from some moments when it lost its position and provided inaccurate data. The trajectories shown on Figure 4.13 agree with the trajectories observed during the test.

Other findings from the test is that the lifetime of the battery exceeded two hours. This time is much longer than the usual sounding time. This allows to lighten the radiosonde by using batteries of lesser capacity and less weight. The continuous working time of the radiosonde was of at least one hour. It was also possible to check that the radiosonde withstand a fall from over one meter and continue working without problems. One of the negative results of the campaigns was that in one of the soundings the radiosonde stopped transmitting shortly after the start of the fly without known reason. It was also discovered that disconnection the antenna can shutdown the transmitter by overloading, and thus overheating, the power regulator of that module.

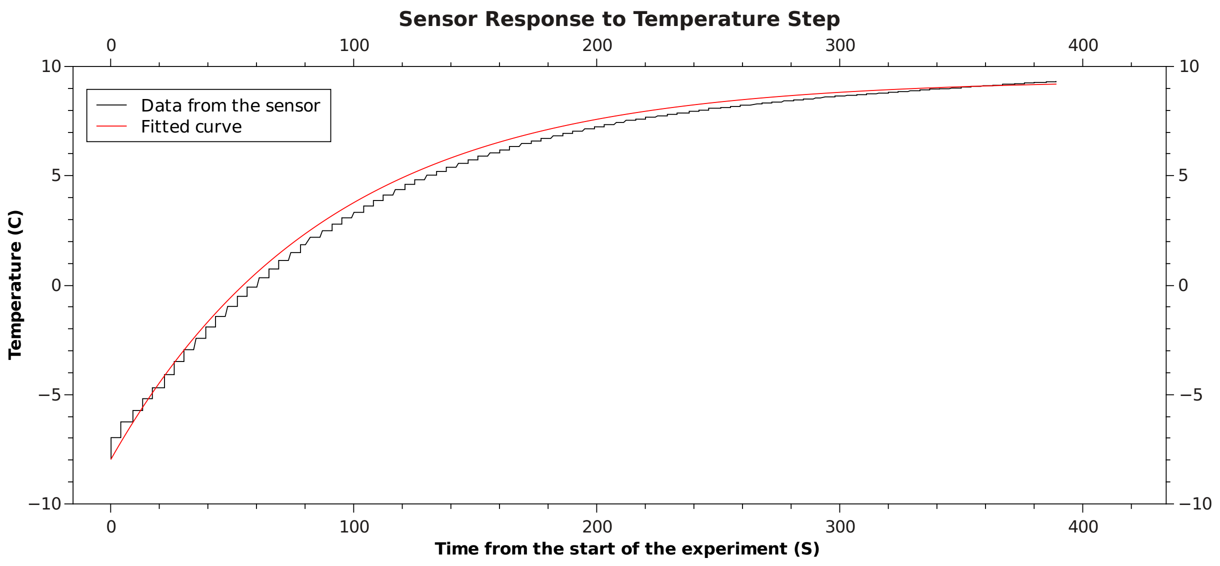


Figure 4.8: Data obtained from temperature step test.

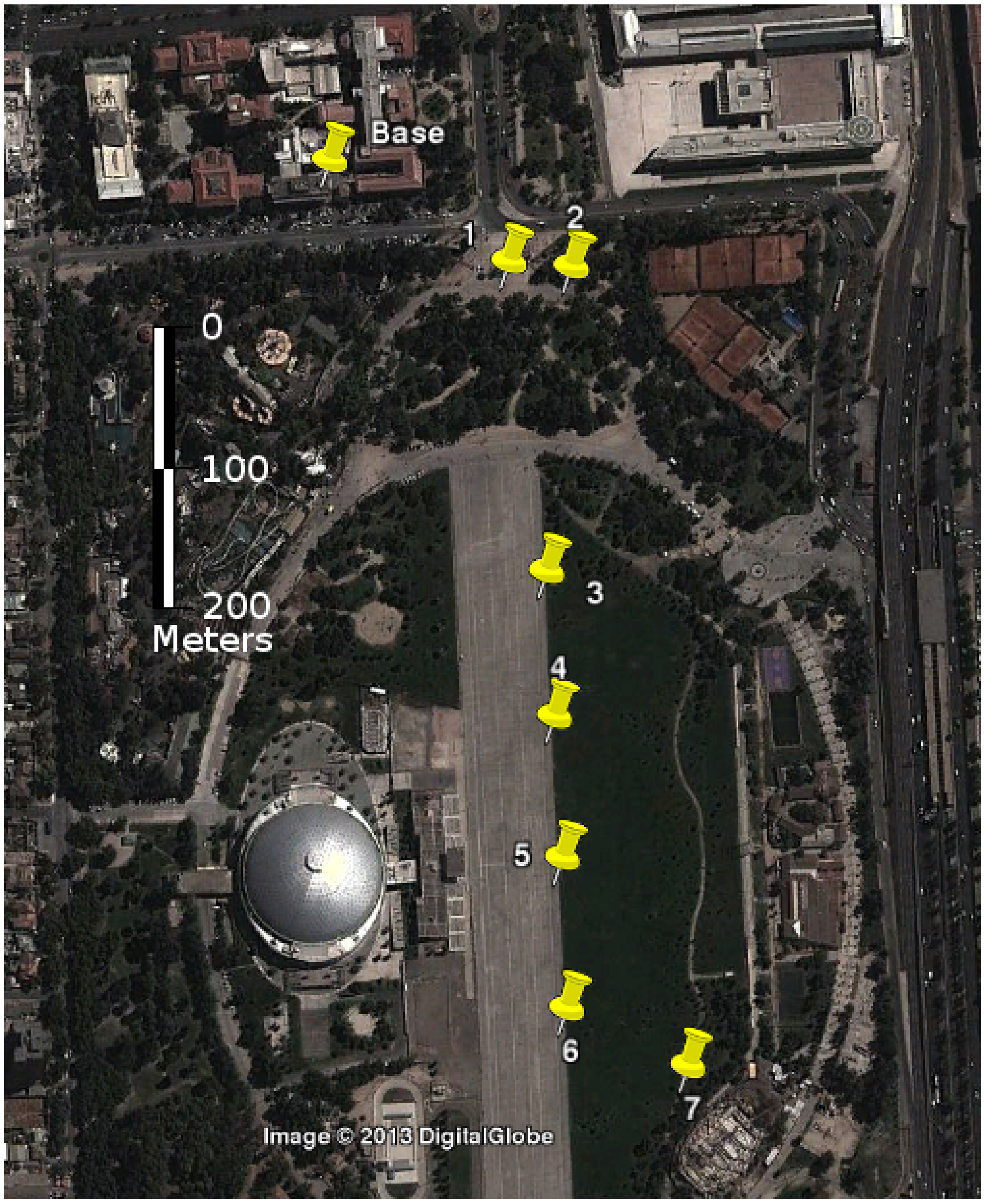


Figure 4.9: Map of the base station and locations of the radiosonde (indicated with numbers) for testing the transmitter module.

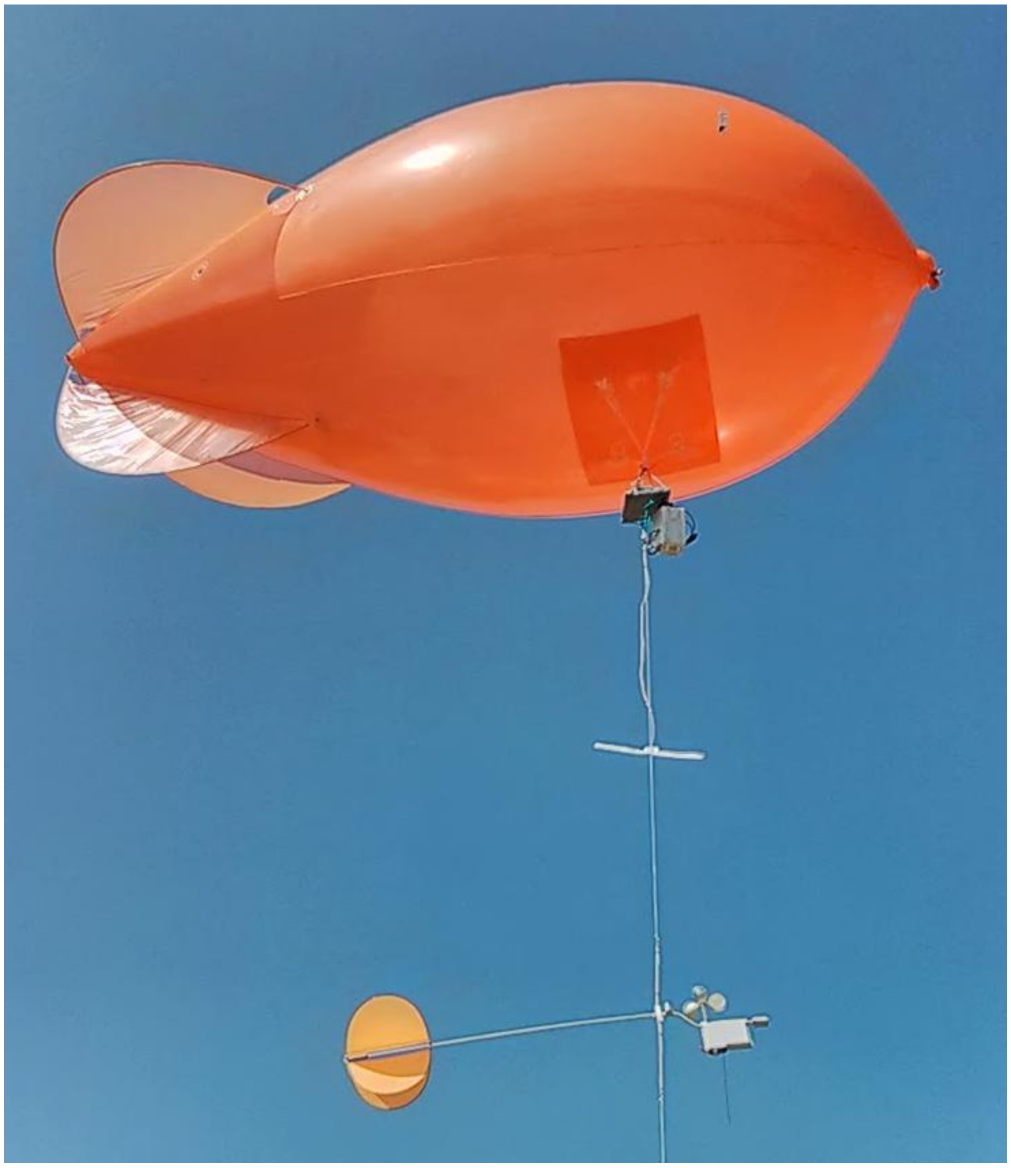


Figure 4.10: Setup of the captive balloon test. The elements in the photo from top to bottom are: balloon, aethalometer, radiosonde under test, radiosonde's antenna, Air radiosonde.

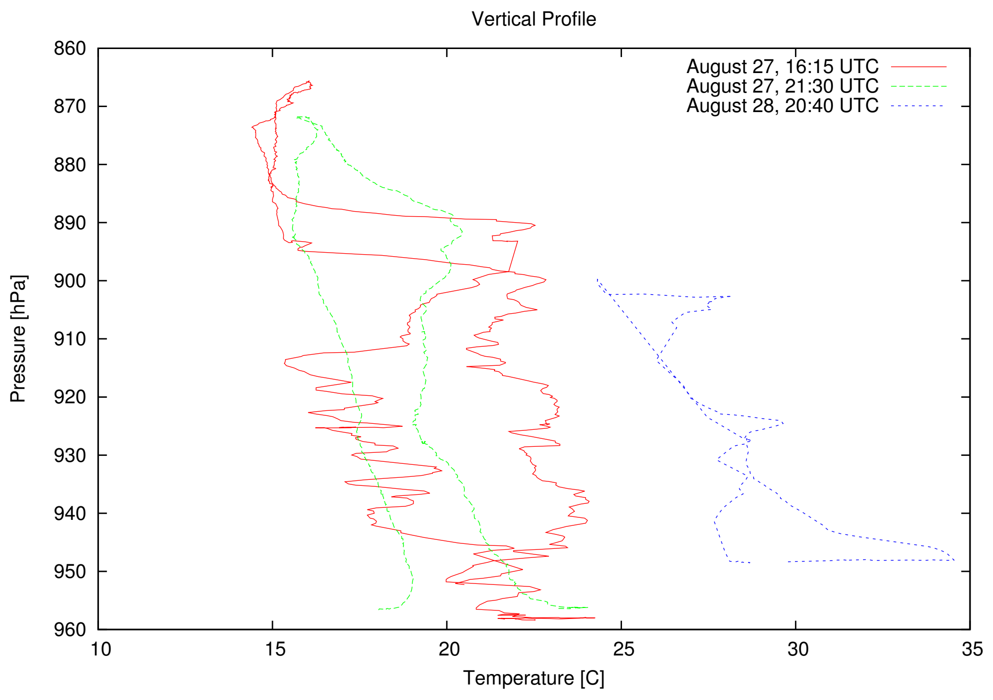


Figure 4.11: Temperature vertical profile of all soundings.

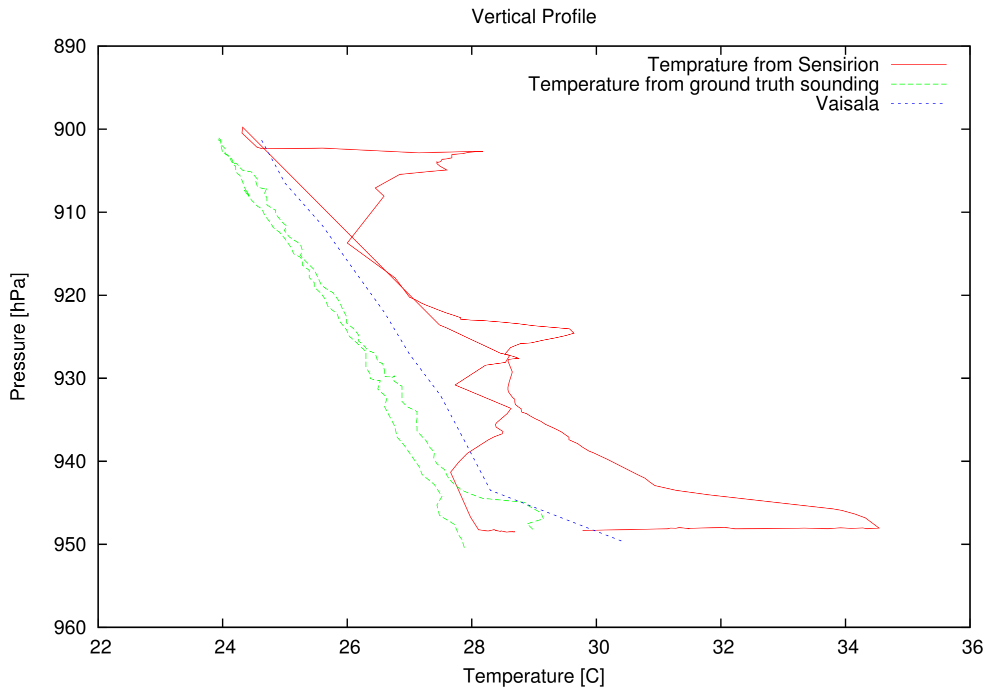


Figure 4.12: Temperature vertical profile of device under test, tethered Air radiosonde and free Vaisala radiosonde.

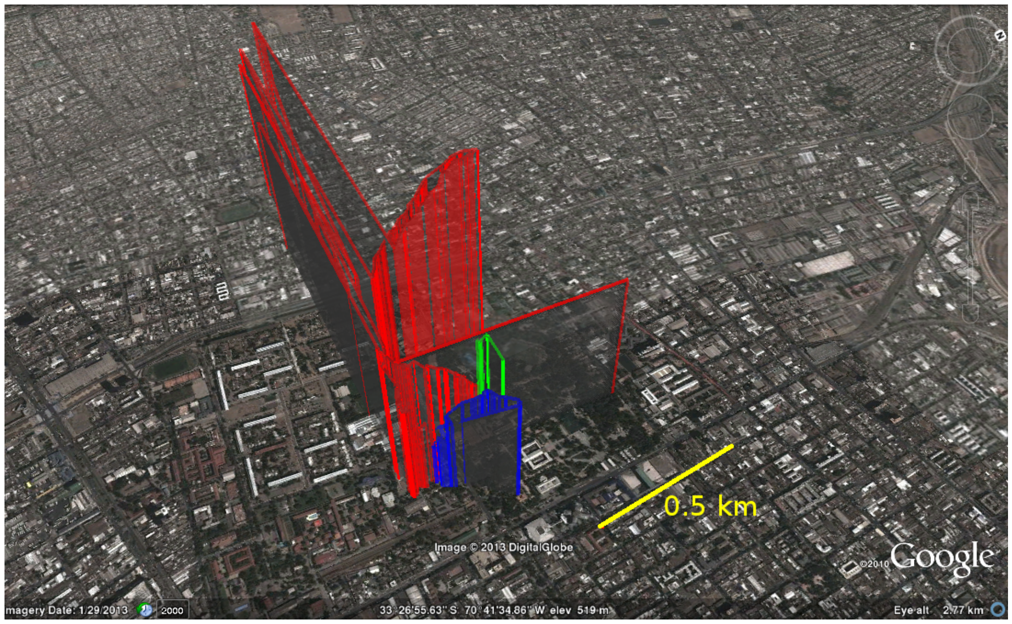

0.5 km
Image © 2013 DigitalGlobe
©2010 Google
magery Date: 1/29/2013
2000
33°26'55.63" S 70°41'34.86" W elev 519 m
Eye alt 2.77 km

Figure 4.13: Path of the radiosonde as given by the GPS.

Chapter 5

Conclusions

This work shows that it is possible to establish a design workflow for hardware as it is done for software. Using the correct tools it is possible to keep track of different versions, changes and enable collaboration with different developers. This concept also extends to include issues tracking and release cycles. This workflow also allows one to adopt good practices used in the context of free software, for example when a group of senior developers take the role of gatekeepers of the repository and review the changes submitted by less experienced developers.

A licensing scheme has been shown that allows one to release the project, including both software and hardware, under a dual licensing approach covering each of the aspects of the project. This way the software is released under a license that has been widely used in many other projects previously and, thus, the implications and cases of use are well understood. This leaves room to license the hardware design by the newly introduced CERN OHL which contains definitions that are specifically for hardware.

The communication bus allows one to partition the design into several sub-problems that can be faced independently. Using a communication protocol that provides slave addressing, such as I^2C, allows to connect many modules on the same bus without interfering with each other. On the contrary to other stackable shields systems there are no concerns about pin usage and physical conflicts between modules.

On the telecommunication module it is necessary to examine with more attention into the internals of the HAC-LM module. Originally the election of this module was made because it was a transparent layer of communication. Communication from the sonde to ground is as simple as transmitting the data serially. Although this module has shown not to behave as smoothly as desired. The transmission rate is not as good as described in the datasheet of the device. Moreover this is unexpected when contrasted to the actual amount of power that is transmitted by the device.

From the standpoint of the several modules that comprise the core of the radiosonde there is still a good amount of work that can be carried out.

The power module currently uses linear regulators to provide a fixed voltage. Those regulators are extremely easy to used, but also extremely inefficient. When powering the system with 11.1V batteries in fact more than half of the power is being dissipated as heat. Switching regulators could be used instead. Those regulators provide power intermittently at a high frequency. Controlling the duty cycle it is possible to deliver the desired output level. Using this technique there is no need to dissipate power as heat thereby reaching higher efficiency rates. This implies that lighter batteries could be used to achieve the same fly time.

Concerning the payload of the system, the sensing module, while the precision of their sensors is actually good the response time is an issue. At the speed the system is expected to move vertically through the atmosphere (roughly 5 m/s) it may be necessary to have faster sensors. A compromise solution could be to have both kinds of sensors and using techniques, such as Kalman Filter, assign more or less confidence to each sensor depending on the rate of change of the different signals, i.e. when the temperature is changing fast the less accurate sensors may be closer to the real value of the variable and conversely when the signal is steady. This approach will require a closer study of the static and dynamic characteristics of the sensors being used, modeling of the system, and calibration of the parameters of the model.

Bibliography

[1] Federico Flores, Roberto Rondanelli, Marcos Díaz, Richard Querrel, Karel Mundnich, Luis Alberto Herrera, Daniel Pola, , and Tomás Carricajo. The lifecycle of a radiosonde. *Bulletin of the American Meteorological Society*, 2012.

[2] Richard Stallman. The gnu project. `http://www.gnu.org/gnu/thegnuproject.en.html`, 1999.

[3] GNU. What is free software? `http://www.gnu.org/philosophy/free-sw.en.html`, 2012.

[4] Google. About android. `http://developer.android.com/about/index.htm`.

[5] Inc Red Hat. Red hat website. `http://www.redhat.com/`.

[6] Jonathon Corbet, Greg Kroah-Hartman, and Amanda McPherson. Linux kernel development. *How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It*, 2012.

[7] Lego. Lego nxt hardware developer kit. `http://mindstorms.lego.com/en-us/support/files/Advanced.aspx`.

[8] HiTech. Our products. `http://www.hitechnic.com/products`.

[9] 3D Robotics. Apm multiplatformn autopilot website. `http://ardupilot.com/`.

[10] Arduino. Arduino ethernet shield. `http://arduino.cc/en/Main/ArduinoEthernetShield`.

[11] Arduino. Arduino wifi shield. `http://arduino.cc/en/Main/ArduinoWiFiShield`.

[12] Vaisala. Vaisala radiosonde rs92-d. `http://www.vaisala.com/Vaisala%20Documents/Brochures%20and%20Datasheets/RS92-D-Datasheet-B210763EN-B-LoRes.pdf`, 2012.

[13] R.D. Wierenga and J. Parini. Intermet 403 mhz radiosonde system. In *Ninth Symposium on Integrated Observing and Assimilation Systems for the Atmosphere, Oceans, and Land Surface*, 2005.

[14] J. Nash, T. Oakley, H. Vömel, and LI Wei. WMO Intercomparison of High Quality Radiosonde Systems, Yangjiang, China, 12 July–3 August 2010. *WMO report*, 2011.

[15] Project Horus. Project horus website. `http://projecthorus.org/`.

[16] The Icarus Project. The icarus project website. `http://www.robertharrison.org/icarus/wordpress/`.

[17] James Coxon. Pegasus hab project website. `http://www.pegasushabproject.org.uk/`.

[18] Alien - altitude imaging entering near-space. `http://www.pegasushabproject.org.uk/`.

[19] CadSoft. Eagle freeware version. `http://www.cadsoftusa.com/download-eagle/freeware/?language=en`.

[20] KiCad Development Team. Kicad website. `http://www.kicad-pcb.org/`.

[21] KiCad Development Team. Kicad documentation. `http://www.kicad-pcb.org/display/KICAD/KiCad+Documentation`.

[22] Perforce Software. Perforce website. `http://www.perforce.com/`.

[23] The Apache Software Foundation. Subversion website. `http://subversion.apache.org/`.

[24] Git Development Team. Git website. `http://git-scm.com/`.

[25] Mark Lodato. A visual git reference. `http://marklodato.github.io/visual-git-guide/index-en.html`.

[26] ATA Serial. International organization: Serial ata revision 3.0. *Gold Revision, June*, 2, 2009.

[27] NXP Semiconductors. I2c-bus specification and user manual. `http://www.nxp.com/documents/user_manual/UM10204.pdf`.

[28] Atmel. Atmega328p data sheet. `http://www.atmel.com/Images/doc8161.pdf`.

[29] Fairchild Semiconductor. Lm78xx/lm78xxa 3-terminal 1a positive voltage regulator. `http://www.fairchildsemi.com/ds/LM/LM7805.pdf`.

[30] InvenSense Inc. Itg-3200 product specification revision 1.4. `https://www.sparkfun.com/datasheets/Sensors/Gyro/PS-ITG-3200-00-01.4.pdf`.

[31] HAC-LM Data RF Module. Shenzhen hac telecom technology co. ltda. `http://www.rf-module-china.com/products/RF_Modules/20.html`.

[32] Free Software Fundation. Gnu general public license. `http://www.gnu.org/licenses/gpl.html`, 2007.

[33] Free Software Fundation. Gnu lesser general public license. http://www.gnu.org/licenses/lgpl.html, 2007.

[34] Creative Commons. Creative commons website. http://creativecommons.org/, 2013.

[35] Creative Commons. Cc0 1.0 universal. http://creativecommons.org/publicdomain/zero/1.0/legalcode.

[36] European Organization for Nuclear Research. Cern open hardware license. http://www.ohwr.org/attachments/662/CERNOHLv1_1.pdf.

[37] European Organization for Nuclear Research. Guide to the cern ohl v.1.1. http://www.ohwr.org/attachments/663/CERNOHLv1_1_howto.pdf.

[38] Sensirion. Datasheet sht1x (sht10, sht11, sht15) humidity and temperature sensor ic. http://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity/Sensirion_Humidity_SHT1x_Datasheet_V5.pdf.

[39] Vaisala. Hmp155 humidity and temperature probe. http://www.vaisala.com/Vaisala%20Documents/Brochures%20and%20Datasheets/HMP155-Datasheet-B210752EN-E-LoRes.pdf.