This Master Thesis is devoted to the design of the first protoyype of the Telemetry, Tracking and Command board that would be used in a cubeSat. It is intended to send data from other modules in the satellite and pictures taken on command trough 2 RF channels: VHF (145.9 MHz) and UHF (433 MHz). In this project we aimed to use trendy technologies such as GNU Radio, which allowed us to verify that the design was functional.

Catherine Dehollain is responsible for research projects at Swiss level (SNF projects, CTI projects) as well as at European level (FP7 projects, Marie-Curie projects). Her activities are focused on wireless data communications of sensor nodes for telecoms applications and biomedical applications. She is also interested by remote powering of sensor nodes in order to avoid the use of a micro-battery.

Juan Antonio Martín Galicia was born in Granada (Ciud (Andalucia, Spain) He finished his bachelor in Telecommunications Engineering in 2017, specialized in electronic circuits design. Then he started his master studies in Electrical Engineering at EPFL

Andrés María Roldán Aranda is the academic head of the present project, and the student's tutor. He is professor in the Departament of Electronics and Computers Technologies

Copy for the student / Copia para el alumno

MASTER THESIS

Telemetry, Tracking & Command Board Design of a CubeSat

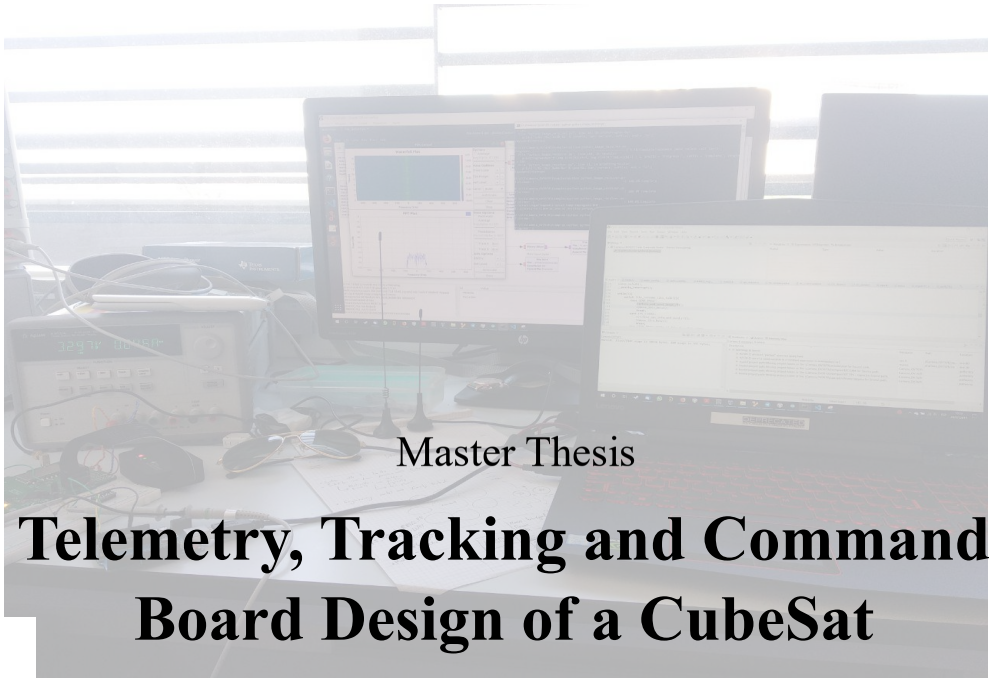
Juan Antonio Martín Galicia

ELECTRICAL ENGINEERING

18/19

UNIVERSITY OF GRANADA

Master in Electrical Engineering



Master Thesis

Telemetry, Tracking and Command Board Design of a CubeSat

Juan Antonio Martín Galicia

2018/2019

Tutora: Catherine Dehollain
Tutor: Andrés María Roldán Aranda

Telemetry, Tracking & Command board for CubeSAT

Designing the COMM subsystem of a cubeSat

Master Thesis 2019
presented the 8 août 2019

École polytechnique fédérale de Lausanne

EPFL

Juan Antonio Martín Galicia

Dr. Catherine Deholain, professor at EPFL
Dr. Andrés María Roldán Aranda, professor at GranaSAT

Lausanne, EPFL, 2019

No nos pidas la luna,
Nosotros tenemos las estrellas.
– La extraña pasajera (1942)

To my parents. . .

Contents

List of Figures

List of Tables

1 Introduction

A cubeSat is a pico satellite standard that was created in 1999 by California Polytechnic State University and Stanford University as an attempt to facilitate access to the space for the students. Because of the extreme conditions that these devices face in the space while ensuring a certain quality of service they tend to be quite complex and expansive. The main idea behind a cubeSat is to limit its size and mass in order to come with a simpler and more affordable design to launch.

These limitations indeed make the design process shorter and simpler; however, they also come at a prize, and it is that due to this simplification the design is also more likely to fail in mission and has a shorter life on orbit, and this is something known and accepted by the community.

Even when these problems exist, the idea of a type of satellite that can be rapidly designed, built, tested and launched still really attractive and it has caught the attention of hundreds of organizations, creating an open community sharing their approaches and perspectives.

One of these organizations is GranaSAT group, who wants to develop their first cubeSat. There are currently 3 people working on that project, each of them designing a different board with their own functionality: Energy Powering Subsystem or EPS by Luis Sánchez Velasco, On-Board Computer or OBC, by José Carlos Martínez Durillo and the Telemetry, Command and Tracking board or TTC, that I will be doing for my master thesis. The objective of this project is design, build and test a system able to communicate with the other boards via a bus, capture images and send data to a ground station through 2 different channels: one at 145.9 MHz in the VHF band and a secondary one at 433 MHz, the UHF band.

Figure 1.1 – GranaSAT Logo

The structure of this report is the following:

- **Chapter 2, Requisites:** In this section the functional requisites given by the company are specified.
- **Chapter 3, Analysis:** From the functional requisites the technical requirements are inferred, and then the block diagram of the system can be designed from them. The analysis continues with a comparison between devices that could be used for the system and choosing the most appropriate one for the specific application. Finally, a power and mass budget are provided.
- **Chapter 4, Design:** At this point all the components in the project have been chosen and the block diagram is known. The next step would be to design the PCB, each of the subsystems and also an environment to verify that they are functional.
- **Chapter 5, Implementation & test:** We will also show the tests that were done to the modules, show the issues that appeared while performing them and their solutions.
- **Chapter 6, Conclusions & future work:** Finally, in this section we will summarise what has been achieved in the project and what could be the next steps or improvements to make in the future.

2 Requirements

In this chapter we show which are the requirements for the project proposed by GranaSAT. From them we will extract the functional requirements later on in Chapter 3. We can divide them in primary or secondary depending on their importance.

2.1 Primary requirements

1. Communicate with the Energy Power Subsystem (EPS).
2. Communicate with the On-Board Computer (OBC).
3. Take pictures with a camera module on command.
4. Send Telemetry data over a 433 MHz channel.
5. Send pictures over a RF channel at 145.9 MHz.
6. Design of a PCB with all the components to perform the aforementioned tasks.
7. Dimensions limited to 10 cm x 10 cm
8. Mass limited to 1.33 kg total
9. Must include a bus compatible with the signals shown in Figure 3.5

2.2 Secondary requirements

1. Make the RF modules work with Arduino.
2. Being able to reduce the power consumption dynamically
3. Receive and demodulate messages in a ground station
4. Program RF modules as receivers in at least one of the channels.

5. The output power from the RF system should be 30 dBs.

3 Analysis

In this chapter we infer the technical requirements from the requisits given in Chapter 2, and from them we will create the system diagram as a set of interconnected blocks that perform different tasks. Then, from this diagram and the technical requirements we will compare components that could be used and choose the most appropriate ones.

From the specifications it is clear that one of the main objectives of the system is to be able to transmit packets at 145.9 and 433 MHz, and the data that needs to be sent will come from other subsystems connected via a bus and a camera.

The fact that is is going to be used in a satellite also has several implications. First of all, satellites have a powering system that is based on batteries storing energy and photovoltaic panels refilling them. However, these panels will not always produce enough power to fill these batteries, either because of the positions of the panels with respect the sources of radiation or because the board is performing heavy operations that consumes more power. So it is important for the system to be **low power** and a desirable feature would be **to be able to reduce functionality as it is running out of power**, to make it easier for the EPS to recover energy. Also, because of the application the system needs to be able to survive conditions of **low temperatures**.

Finally, it is an engineering model, so we want it to be easily tested. Hence, in the design phase we will add test points so that signals can be measured with an oscilloscope. Enabling an UART connection between the microcontroller and the computer programming it would allow us to debug digital devices such as the camera, the RF modules and the microcontroller itself, as it would make it possible to send state variables and register values back to the programming computer.

From all this considerations and taking the FloripaSat Block Diagram (Figure 3.1) as an inspiration we designed our own diagram, which is presented in Figure 3.2.

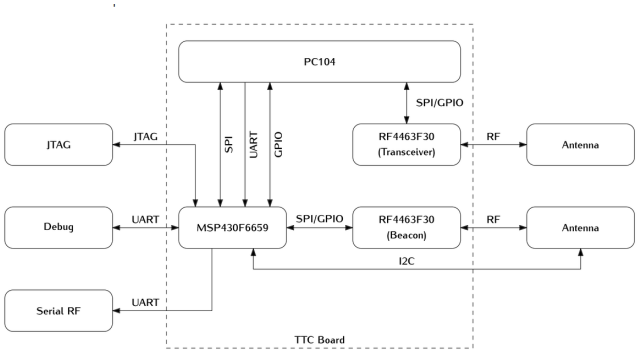


Figure 3.1 – FloripaSat TTC Block Diagram, Image source

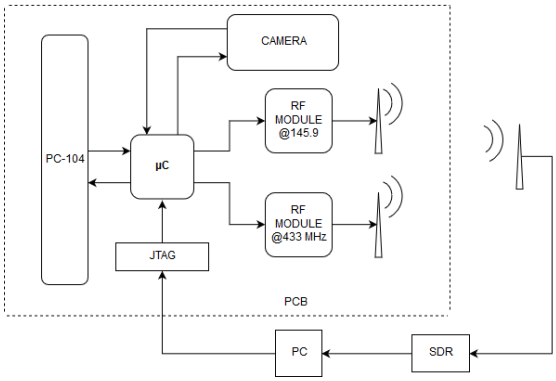


Figure 3.2 – Block diagram

3.1 Hardware components

In this section we will analyse all the relevant blocks in the diagram of Figure 3.2 (the PC would be my personal computer and the JTAG is just for programming the microcontroller, so there is no interest on that). Then, we will compare components that could be used to implement the functions for each of them and select the most appropriate one.

3.1.1 Microcontroller

As mentioned before, one of the desired properties for this block is to be low power, but it also needs to be able to connect with other components using serial bus protocols (so it should have Universal Serial Communication Interface or USCI).

The proposed microcontroller in the FloripaSAT project was MSP430F6659, which is a low power model with 6 USCI ports in total, multiple GPIO pins, Universal Serial Bus (USB). However, after further search in the market we found another model that matched with the specifications: MSP430FR6989.

The main advantage over the previous one that lead us to use it is that its power consumption is almost 3 times lower in active mode (in every other mode it is 2 times lower). It also has some minor improvements that can be taken in consideration such as the use of eUSCI (enhanced USCI) instead of USCI (differences are shown in Figure 3.3), some ports can digitally activate pull-ups and/or capacitors and it has a Ferrite RAM, which is non-volatile, low power but also more expansive. The biggest issue with this microcontroller is that it only has 4 USCI ports while the other has 6, but it is a problem we can deal with, as will be explained in Subsection 3.4. All these features are summarized in Table 3.1

Table 3.1 – Microcontroller comparison

Feature	MSP430F6659	MSP430FR6989
Power consumption in active mode (uA/MHz):	295	100
Power consumption in standby mode (uA/MHz):	2.2	0.4
Power consumption in shut-down mode (uA/MHz):	0.45	0.02
RAM	SRAM	Ferrite RAM (FRAM)
Universal Series Communication Interface (USCI)	USCI	enhanced USCI (eUSCI)
Number of USCI/eUSCI ports	6	4
Full speed USB	Yes	No
Internal and digitally enabled capacitors/pull-ups	No	Yes
Prize ($)	6.63	7.12

Parameter	MSP430F2xx	MSP430FR57xx
UART		
Enhanced baud rate generation	No	Yes
TXEPT interrupt (from USART)	No	Yes
Start edge interrupt	No	Yes
Selectable glitch filter	No	Yes
Interrupt vector generator	No	Yes
SPI		
Enhanced baud rate generation	No	Yes
Enhanced bit rate specs	4 to 6 MHz	10 MHz
Interrupt vector generator	No	Yes
I2C		
Preload of transmit buffer	No	Yes
Clock low timeout	No	Yes
Byte counter	No	Yes
Multiple slave addresses	No	Yes
Address bit mask	No	Yes
Hardware clear of interrupt flags	Yes	No
Interrupt vector generator	No	Yes

Figure 3.3 – eUSCI vs USCI

3.1.2 Camera module

For the camera, it would also be desirable to have low power consumption and being able to reprogram it so that we can control the resolution of the image. This way we can respect the secondary requirement 2 by lowering the image quality when the battery is depleting. Other interesting features of the camera are the pixel size, pixel format, SNR, dynamic range and prize.

There are several devices in the market that fulfil these requisites. The ones that interested us

the most are presented in Table 3.2.

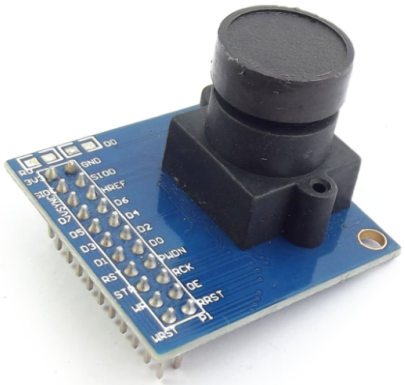
Table 3.2 – Camera comparison

Feature	OV7670	OV7670 w FIFO	Adafruit TTL 397
Voltage Supply (V)	3.3	3.3	5
Current consumption(mA)	40	50	75
Power consumption (mW)	132	165	375
Pixel format	Raw RGB, GRB422, RGB565/555/444, YUV422 & YCbCr422	Raw RGB, GRB422, RGB565/555/444, YUV422 & YCbCr422	JPEG & M-JPEG
SNR (dBs)	46	46	45
Dynamic Range (dBs)	52	52	60
Programming protocol:	SCCB	SCCB	UART TTL
Need for an external clock	Yes	No	No
Prize ($)	2.73	17.95	34,89

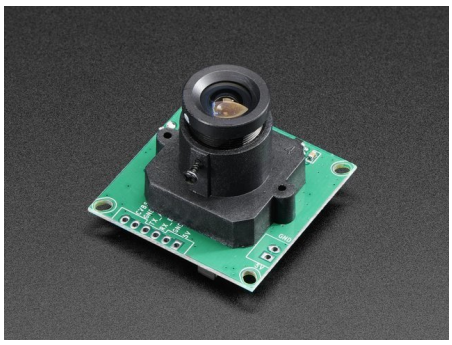
These 3 models are commonly used by arduino users, which is already an advantage because it is an open community that shares their experiences and knowledge in forums.

The first camera that we discarded was model OV7670 without FIFO because it makes the PCB design and firmware development harder. This model requires an input clock running at 8 MHz minimum, so we would need to provide it externally and then be perfectly synchronized when retrieving images, meaning that the microcontroller should run at least at the same frequency. Meanwhile, there is another available model that already has an integrated input clock running at 24 MHz and a FIFO buffer (AL422B) that allow users to retrieve data at any rate. The prize we have to pay for it is an slightly higher power consumption and a more expansive camera.

So the final decision was between adafruit 397 (Figure 3.4b) and OV7670 with FIFO (Figure 3.4a). The main advantages of the first model is that it has a less complex pinout, as it has the supply pins (5V and GND) plus TX and RX UART pins that are used for setting the image parameters and retrieving it respectively; however, it consumes and costs twice as much and the lab already had the other model because it was used in previous projects. By using camera OV7670, that has a bus of 20 signals (this will be covered in Section 4), we would save power and money at expanses of a possibly longer and harder design/debug phase.



(a) OV7670 & FIFO. Image source



(b) Adafruit 397, Image source

Figure 3.4 – Camera modules

3.1.3 RF module

It is possible to find in the market a huge variety of transceivers. Most of them allow the user to choose the type of modulation, the data rate and even higher level features such as the preamble, synchronization word, CRC, etc. But for any case, due to the limitation of size and weight we will use a single antenna for transmit and receive for each frequency band, so there will be 2 antennas. This means that if the transceiver has a different port for transmitting and receiving it is necessary to add a switch so that when the module is transmitting the signal follows a path in which it is amplified with a PA and matched to the antenna and when it is receiving it follows another in which it is also amplified, but with with a LNA.

In the FloripaSat-I project a module called RF4463F30 is used. It has an integrated low power transceiver (Si4463), PA, RF matching circuits, built-in antenna switching for RX/TX modes and a LDO to have a stable source of 3.3 V for the transceiver while feeding the PA with the input voltage.

The rest of the transceivers that we found did not include these integrated components, making it more useful, power hungry and expansive, so in the comparison made in 3.3 these facts should be taken it into account.

Table 3.3 – RF modules comparison

Feature	RF4463F30	ADF7021-N	AX5043
Manufacturer	NiceRF	Analog Devices	ON Semiconductor
Voltage Supply (V)	3.3 - 6.5	2.3 - 3.6	1.8 - 3.6
Current in TX mode (mA)	550	23	51.6
Current in RX mode (mA)	10	18.3	9.5
Current in sleep mode (uA)	3	1	0.05
Operating frequencies(MHz)	150/433	80-960	27-1050
Modulations	(G)FSK,4(G)FSK, (G)MSK,OOK & ASK	2FSK, 3FSK, 4FSK, MSK	(G)FSK,(G)MSK,4FSK, ASK,AFSK,FM,PSK
Data rate (kbps)	0.1 - 1000	0.05-24	0.1-125
Receiver Sensitivity (dBm)	-126	-125	-126
Output power (dB):	30	13	16
Prize ($)	8.50	2.05	2,38

The other 2 modules have clear advantages over module RF4463F30, such as a much lower power consumption in TX mode and lower prize. However, the huge gap when comparing the first feature is due to the PA raising the output power and the consumption of the other built-in components it has.

The module that we will use in the end was indeed RF4463F30 (Figure 4.15). Firstly because it combines a good output power (30 dB max) with high sensitivity (-126 dBm), it fulfils Requirement 5 without adding an external PA, we do not need to create circuit to switch TX/RX paths to the antenna, it offers high data rates and some other interesting functions such as preamble detection in RX mode and configurable packet structure.

3.2 Bus PC-104

As explained in Chapter 1, there are several boards in a single cubeSat, and for them to exchange information they make use of a bus of 104 signals. Because it can be accessed by all the boards, designers should make sure that they do not use any line that is already in use by another designer; in fact, this was already stated in the requirements list (more specifically Requirement 9). The bus structure that was agreed to be used is presented in Figure 3.5.

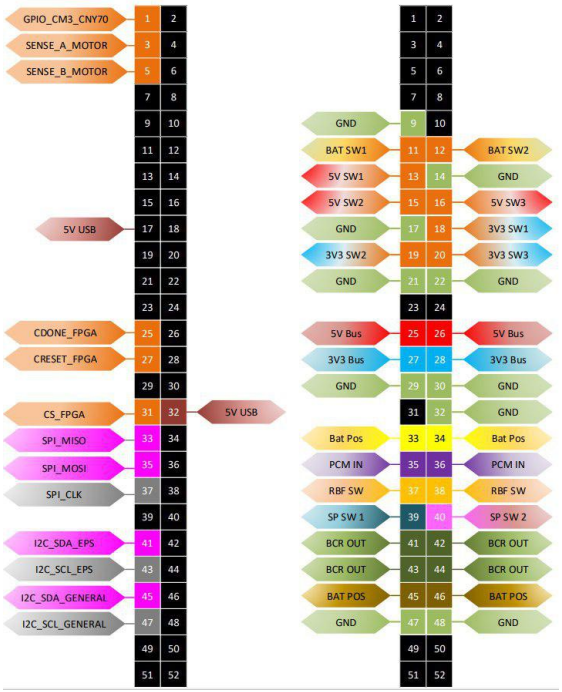


Figure 3.5 – Bus PC-104

Most of these signals come from the EPS; and for this project the only ones that will be used are:

- 5 V supply, pins 25 and 26
- 3.3 V supply, pins 27 and 28
- GND, pins 9, 14, 17, 21, 22, 29, 30, 32, 47 and 48
- EPS I2C, pins 41 and 43
- General I2C (connected to the OBC), pins 45 and 47

3.3 Software-Defined Radio & GNU Radio

A secondary requisite that results really interesting to achieve is Requisite 3, because if we are able to simulate a ground station receiving and demodulating packets we can also verify that the design of the RF modules works and also to perform some measurements. One of the first projects made in GranaSAT was the design of a ground station, which was gradually enhanced (link to all the projects related to it). However, we aim for something easier to handle and provide the company with the *know-how* of a tool that is becoming increasingly popular: Software Defined Radio (SDR).

SDR is a new trend of communication systems in which building blocks are software defined; so instead of some specific hardware to implement the system the only elements needed are an antenna to transmit/receive, the SDR device to perform the operation depicted in Figure 4.24 when receiving and 4.25 when transmitting and a computer or an embedded system connected to the SDR with a framework to implement the signal processing blocks. The model that will be used is provided by the lab (Figure 3.6), which can accept as input frequency bands 174 MHz-230 MHz and 470 MHz-862 MHz. 433 MHz is not within any of these frequency bands, but it is close enough to detect it, the performance however would be lower.

Figure 3.6 – Mini RTL-SDR + DAB FM USB DVB-T

This technology has an incredible potential, as it allows engineers to run different radio systems with the same pieces of hardware (which are not even expansive) and it could be reconfigured at any time to be used for another radio architecture and band.

Concerning the SDR framework, they provide the user with some pre-defined software blocks used to implement common signal processing functions and also allows them to define their own. There are plenty of possibilities such as GNU Radio, Pothos SDR and Redhawk SDR. But the one that we will use is the first one: GNU Radio (Figure 3.7), which is an open source development kit made in Python. This decision was made because the large community that it has created, the amount of documentation that can be found for this framework and because these building block are programmed in Python, which is a familiar programming language for the Lab.

Figure 3.7 – GNURadio Logo

Using this technology we can implement a receiver and demodulator (and even a transmitter). In Section 4.5 we will explain more deeply how the receiver was designed.

3.4 Final low level schematic

Now that all the components have been chosen, the protocols to communicate with each of them are defined too and hence, the number of USCI ports that will be used. In Subsection 3.1.1 it was mentioned that the MSP430FR6989 only has 4 USCI ports total; meanwhile we would need several ports:

1. I2C slave to OBC
2. I2C master of EPS
3. SPI connection with UHF RF module
4. SPI connection with VHF RF module
5. UART connection to PC
6. SCCB (I2C) connection to camera module

This is obviously bigger than 4. We could save one port by controlling the RF modules with the same SPI protocol by using 2 Slave Select signals, but we would need to multiplex another I2C connection so that we only use 2 of them at a time. This ended up not being necessary because it turned out that SCBB is not compatible with I2C as will be explained in Section 5.2.1. As a result, we had to code that protocol with GPIO pins and used the 4 available USCI ports. The function associated to each of them is shown in Table 3.4 and the final block diagram with these modifications in Figure 3.8.

Table 3.4 – USCI ports

Port	Function
USCI A0	UART
USCI A1	SPI
USCI B0	I2C SLAVE
USCI B1	I2C MASTER

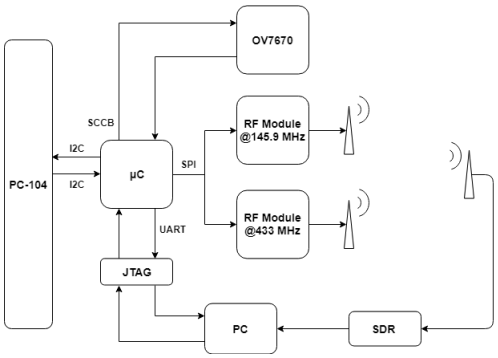


Figure 3.8 – Final block diagram

3.5 Power & mass budget

In the introduction we presented the characteristics of a cubeSat, and it was mentioned that they have 2 intrinsic limitations: size and mass. Size limitation is covered by the PCB design, but before proceeding we need to ensure that the components are not too heavy. The weight of each element used in the design is presented in Table 3.5, taking the PCB measures from the EPS PCB as an estimate. The total weight would be around 51 g, which is less than 4% of the maximum mass it is allowed (1.33 Kg).

Table 3.5 – Mass budget

Component	Mass (g)	Amount
RF4463F30	4	2
OV7670	12.9	1
Camera connector	2	1
MSP430FR6989	0.6	1
PC-104	2.7	1
SMA connector	1.5	2
PCB	22	1
Total	51.2	

At the beginning of this Chapter we explained that because of the application it is important to have a low power design, so keeping track of the power that the module would consume before designing it is important as well. In Table 3.6 the power budget is presented, showing the consumption of each module in sleep and active mode. For the microcrontoller consumption we assumed that the clock frequency is 8 MHz. From the table we can observe that the dominant term in active mode comes from the RF modules, which makes sense taking into account that it drains 0.5 mA when transmitting.

Table 3.6 – Power budget

Component	Sleep mode (uW)	Active mode (mW)
MSP430FR6989	10.56	2.64
OV7670	66	165
RF modules	15	2750
Total	91.56	2917.64

4 Design

By the end of the previous section the final block diagram was drawn and the components to be used were chosen. In this section we will first design the PCB to hold all the system and then each of the blocks of the diagram.

4.1 PCB Board design

The design is entirely shown in Appendix 7, which respected Requirement 7 by limiting its dimensions. Also, because it is the first prototype, debug LEDs, connectors and test points were included to ease the measurements.

Apart from this, there is a detail that is worth to explain more deeply: antenna adaptation.

The output of the RF modules are connected directly to the antennas, which are 50 Ω, so we need to control the microstrip line parameters so that its impedance is matched to the load. The manufacturer is JLCPCB, and they offer impedance control for designs with more than 2 layers. This is not our case, so we will need to use Equation 4.1 to make it as close to 50 Ω as possible.

$$Z_0 = \frac{\eta_0}{2\pi\sqrt{2}\sqrt{E_r+1}} ln\left(1+4\left(\frac{H}{W_{eff}}\right)(X_1+X_2)\right)$$

Where:

$$W_{eff} = W + \left(\frac{t}{\pi}\right) ln\left\{\frac{4e}{\sqrt{(\frac{T}{H})^2+(\frac{T}{W\pi+1.1T\pi})^2}}\right\}\frac{E_r+1}{2E_r}$$

$$X_1 = 4(\frac{14E_r+8}{11E_r})(\frac{H}{W_{eff}})$$

$$X_2 = \sqrt{16(\frac{H}{W_{eff}})^2(\frac{14E_r+8}{11E_r})^2+(\frac{E_r+1}{2E_r})\pi^2}$$

Z_0 = characteristic impedance of the microstrip in ohms (Ω).

H = subtrate height

W = trace width

T = trace thickness

ϵ_r = substrate dielectric

Figure 4.1 – Microstrip line impedance, Image source

We could modify the substrate height and the trace thickness, and by playing with them we found a set of parameters that achieved 52 Ω (Figure 4.2).

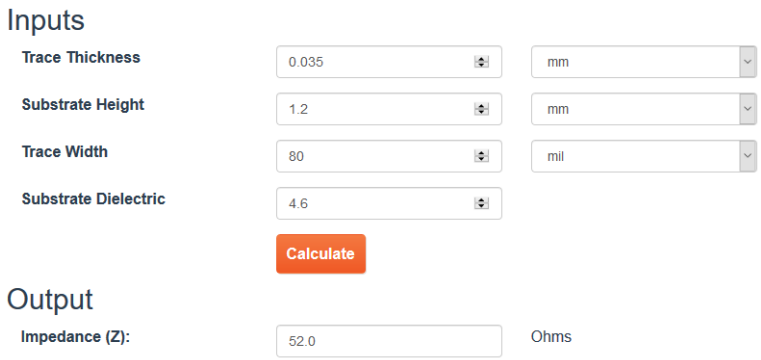


Figure 4.2 – Impedance calculation

4.2 I2C Master/Slave

As mentioned before, the TTC board need to communicate with the other subsystems present in the cubeSat so that it can send all the relevant information via RF. In the lab we agreed to use I2C protocol to communicate between boards because a high data rate protocol is not necessary to exchange this information and it would only use 2 pins from the bus.

For now the TTC will communicate with 2 other subsystems: EPS and OBC. The OBC will send the commands and data to the COM subsystem, so it will act as a master. On the other side, it would be the COM subsystem the one in charge of requesting data from the EPS, so our system would act as a master. In the following subsections these 2 connections will be explained.

4.2.1 Slave mode - OBC

The system may receive from the OBC data and commands, which are distinguishable if we define registers to save data and registers to trigger actions. The data we could receive has not been defined yet, so for now the decision was to leave 50 registers for saving it plus 2 to record the virtual length of the buffer (how many of the available registers has been filled by the OBC) and the ID of the OBC.

On the other side, to trigger a command the OBC needs to write one of the 2 other special register. One of them (0x36) calls a debug function that will make the LEDs start blinking; and the other one , the Configuration Register (0x35), will call a function depending on its content. These functions are:

- RF SEND - Sends the data saved in the buffer via RF
- RF SEND PICTURE - Sends a picture via RF
- UART SEND PICTURE - Sends a picture through the UART port
- RF SEND EPS STATUS - Sends EPS information via RF
- UART SEND EPS STATUS - Sends EPS information through the UART port
- RF SEND DUMMY BEACON - Sends dummy packet via RF

The addresses of all these registers and their functions are summed up in Table 4.1.

Table 4.1 – OBC Registers

Register Address	Function
0x00	Start Data Register
0x32	End Data Register
0x33	Buffer Length
0x34	Configuration Register
0x35	ID Register
0x36	LEDs Register

4.2.2 Master mode - EPS

For this case the design was simpler because the TTC only need to request data from the EPS, and we had a more clear idea of the information that would be requested. The addresses and content of these registers are shown in Table 4.2.

Table 4.2 – EPS Registers

Register Address	Function
0x01	X Panel Voltage
0x02	X Panel Current
0x03	Y Panel Voltage
0x04	Y Panel Current
0x05	Z Panel Voltage
0x06	Z Panel Current
0x11	Battery Voltage
0x12	Battery Current
0x17	5V Switch Voltage
0x18	5V Switch Current
0x1D	3V3 Switch Voltage
0x1E	3V3 Switch Current
0x23	Temperature

4.3 Camera Module

The camera that we chose can be directly connected to the microcontroller and then controlled by the firmware, as they operate at the same voltage and none of the signals need a pull-up, pull-down or capacitor. However, to check if the camera works as intended and that it has been properly configured we need a simple way to send the image to other device able to plot it. To do so we will get advantage of the Flash Emulation Tool (Figure 4.3), which has an UART connection to the programming computer. So, the bytes read from the camera will be forwarded to the PC and a listening python script can take them and reconstruct the image.

Figure 4.3 – TI Flash Emulation Tool

All the steps of this procedure, that will be explained in more detail in this section, are depicted in Figure 4.4.

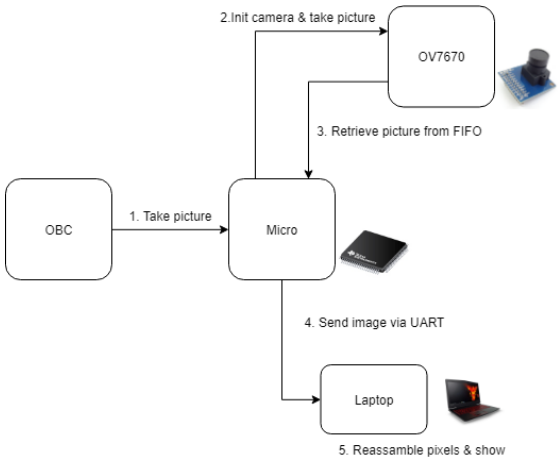


Figure 4.4 – Procedure to take a picture

The first point was already discussed in Section 4.2. Then, points 2 to 3 require communication between the microcontroller and the camera to initialize, reset, take the picture and retrieve it from the FIFO. This is explained in detail in Subsection 4.3.1. Finally, all that is left is to send the information to a PC via UART and reassembles it (4 and 5).

4.3.1 Handling the camera with the microcontroller

In order to control the camera we first need to understand its internal schematic (Figure 4.5) and the function of the signals in the pinout.

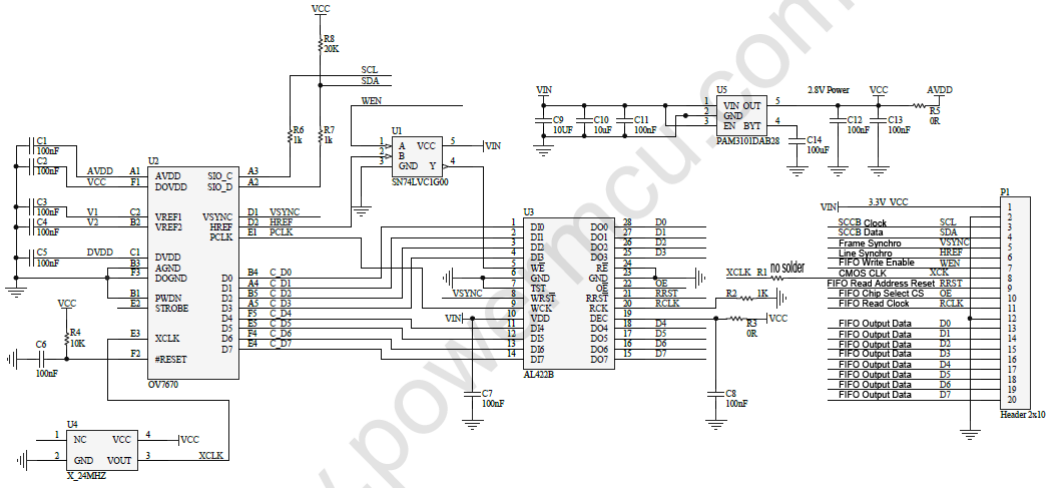


Figure 4.5 – OV7670 & AL422B schematic

The important details about the schematic is that the clock of the camera is already given and it operates at 24 MHz, the write clock and read enable of the FIFO are also given for the

camera, so we do not need to handle them; and that it has 2 enable signals (one for the camera and the other for the FIFO) and 3 resets (one for the camera and the other two for the FIFO). The pinout of the module and hence, the accessible signals are shown in Table 4.3.

Table 4.3 – OV7670 & AL422B pinout

Signal	In-Out	Description
3V3 BUS	IN	Power supply, as its names states it should be 3.3 V
GND	IN	Ground
SCL SCCB	IN	Equivalent to Signal Clock in I2C
SDA SCCB	IN	Equivalent to Signal Data in I2C
VSYNC	OUT	Vertical synchronization
HREF	OUT	Horizontal timing
D0-D7	OUT	Data bits
RST	IN	Camera reset
PWDN	IN	Power down of the camera (active high)
STR	IN	Strobe (not used)
RCK	IN	Read clock
WR	IN	Write enable
OE	IN	Chip select for the FIFO (active high)
WRST	IN	Resets the FIFO writing pointer (active low)
RRST	IN	Resets the FIFO reading pointer (active low)

Knowing their functions and with help from several sources such as [2], [3], [4] and [5]; it was possible to understand the steps to follow in order to use the camera. This procedure is depicted in Figure 4.6.

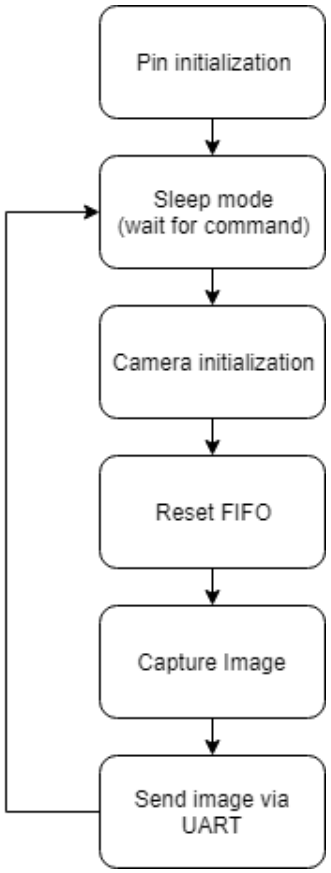


Figure 4.6 – Procedure to take a picture

Pin initialization Firstly, we must specify which pins from the microcontroller are inputs or outputs, and then the camera would remain in sleep mode until a command for taking a picture is received from the OBC.

Camera initialization By activating PWDN and OE signals the camera exits sleep mode. Then it is reseted by applying a short low pulse on the RST signal (Figure 4.7) and finally, we initialize camera registers with SCCB (Figure 4.8).

In the initialization we can change the pixel format (which may be RGB565, YCbCr or Bayer RGB), the resolution (qqVGA, qVGA and VGA), gamma matrix and enable some other features such as the test pattern, that makes the camera send a pre-saved image (Figure 5.5) to the FIFO.

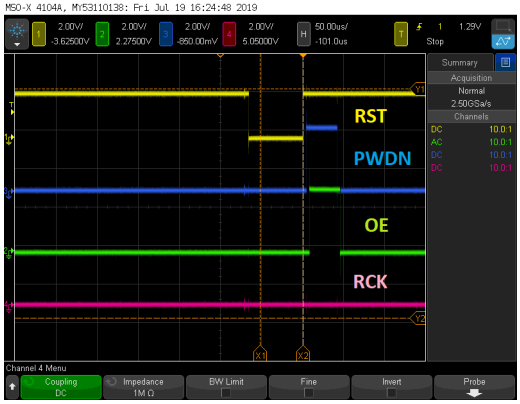


Figure 4.7 – Camera RST

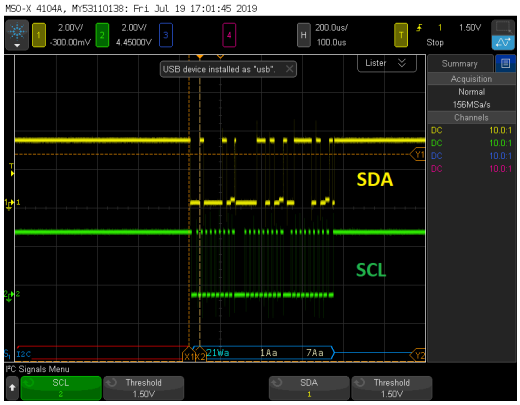


Figure 4.8 – Camera SCCB

Reset FIFO Then we need to prepare the FIFO for receive and then retrieve data. To do so we apply RRST (Read Reset) and WRST (Write Reset) by setting them to low during a rising edge of the RCK (4.9). Read reset will return the reading pointer to the beginning and Write Reset will do the same with the writing pointer.

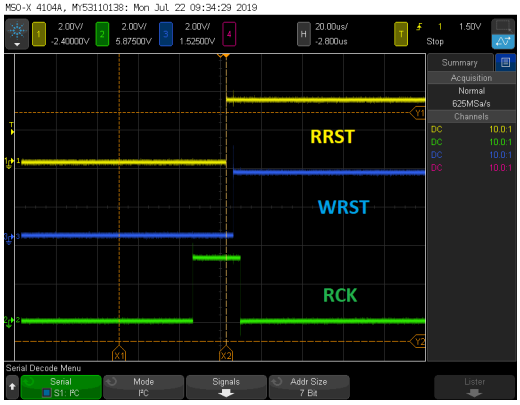


Figure 4.9 – FIFO RST

Capturing an image Once we reached this point the camera is continuously taking pictures, but write enable (WR) is not activated yet, so no image is being saved in the FIFO. To do so we need to wait for a falling edge in VSYNC, because this means that a new image has been taken. Then we allow the memory to be written by setting WR to high until the next rising edge in VSYNC (Figure 4.10).

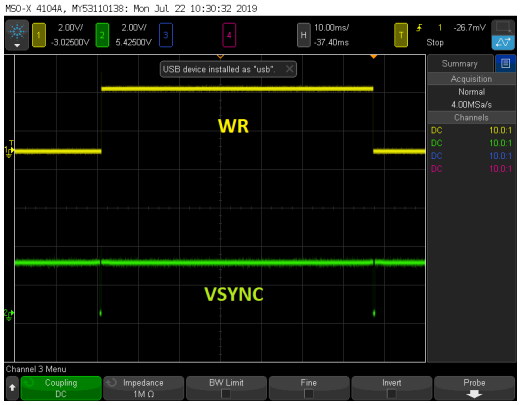


Figure 4.10 – VSYNC

Retrieving an image The image is now saved in the FIFO, reading it is as simple as applying pulses in RCK and then read the data through pins D0 – D7 (Figure 4.11).

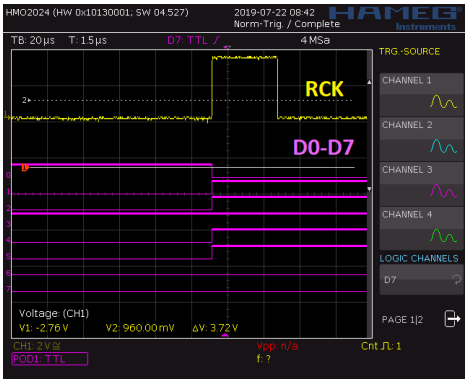


Figure 4.11 – Capture Image

4.3.2 Reassemble the image

For this last step a python script was used. It would listen to an UART port from which it receives the data from the Emulation Tool (Figure 4.12) and saves the pixels in a matrix. Then it converts their format to the ones available in the PIL library (RGB888 and YcbCr) with a simple operation and finally plots the image. The code is located here.

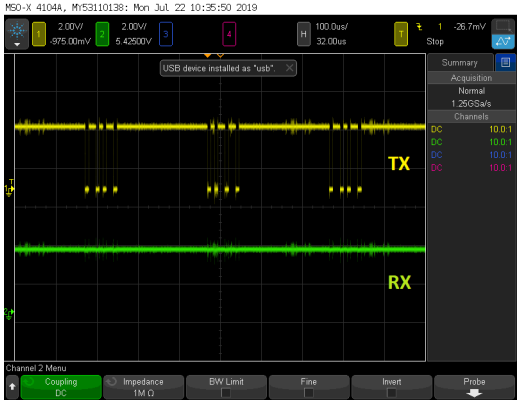


Figure 4.12 – UART

All the available pixel formats are: Raw RGB (RGB888), RGB (GRB 4:2:2, RGB565), YUV (4:2:2) and YCbCr (4:2:2). The first one is the easiest one to understand: a pixel is coded in 3 bytes and each of them correspond to colors Red, Green and Blue respectively. For the rest of the formats 2 bytes are sent per pixel; hence, for qVGA this would mean that 320*240*2 bytes are sent and 160*120*2 for qqVGA.

Some of these formats require no conversion, as RGB888 and YCbCr. However, the rest of them must be converted to one of these formats so that that python can plots them:

YUV 422 This encoding method is a way of encapsulating 12 pixels in 6 by reusing the information contained in Cb and Cr channels. The camera sends the following information word by word:

	Byte 0	Byte 1	Byte 2	Byte 3
Word 0	Cb0	Y0	Cr0	Y1
Word 1	Cb2	Y2	Cr2	Y3
Word 2	Cb4	Y4	Cr4	Y5

Figure 4.13 – How YUV422 pixels are sent

And then, to obtain The YCbCr equivalent we just need to reorder the information and reuse Cb and Cr channels for contiguous pixels:

Pixel 0	Y0 Cb0 Cr0
Pixel 1	Y1 Cb0 Cr0
Pixel 2	Y2 Cb2 Cr2
Pixel 3	Y3 Cb2 Cr2
Pixel 4	Y4 Cb4 Cr4
Pixel 5	Y5 Cb4 Cr4

Figure 4.14 – How YUV422 pixels are converted to YcbCr

RGB 565 This conversion from RGB565 to RGB888 is straightforward. An effective and simple way to perform it is by recalculating each component this way:

R8 = (R5 * 527 + 23) » 6
G8 = (G6 * 259 + 33) » 6
B8 = (B5 * 527 + 23) » 6

This conversion was found [1].

4.4 RF modules

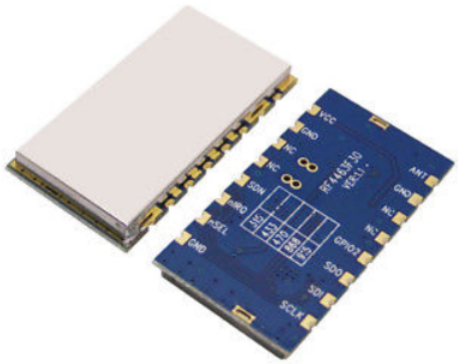


Figure 4.15 – Module RF4463F30

This module relies on the transceiver Si4463, but it also includes a power amplifier to raise the output power, which would be 20 dB at maximum without it, and an antenna switch to change from transmitter to receiver mode plus the corresponding matchings. The schematic with all these components is presented in Figure 4.16 and its pinout in Figure 4.17.

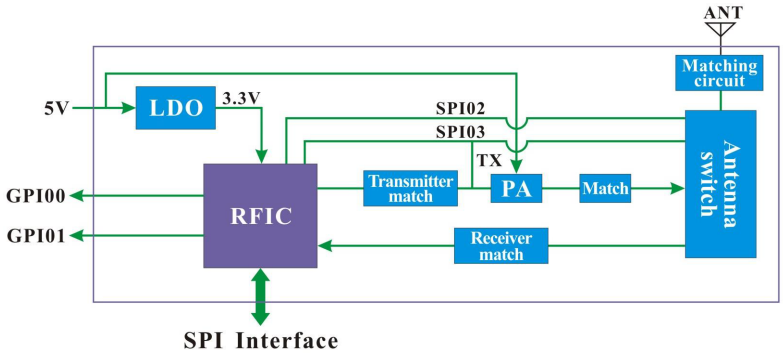


Figure 4.16 – Schematic of the module RF4463

Pin Number	Pin Definitions	Description
1	VCC	Positive power supply
2	GND	Connected to power ground
3,4,14	NC	Vacant, not connected
5	SDN	Power down control. SDN = 1, power down SDN = 0, normal working
6	nIRQ	Interrupt output
7	nSEL	Serial data selection for SPI interfaces.
8	GND	Connected to power ground
9	SCLK	Serial data clock for SPI interface.
10	SDI	Serial data in for SPI interface.
11	SDO	Serial data out for SPI interface.
12	GPIO1	GPIO1 of Si4463
13	GPIO0	GPIO0 of Si4463 for I/O interface
15	GND	Connected to power ground
16	ANT	From 50 ohm coaxial antenna

Figure 4.17 – Pin map of module RF4463

The following task, as for the camera, is to understand how the module works. From that point the state machine diagram presented in Figure 4.18 was drawn, with all the important tasks it should perform in order to transmit a packet.

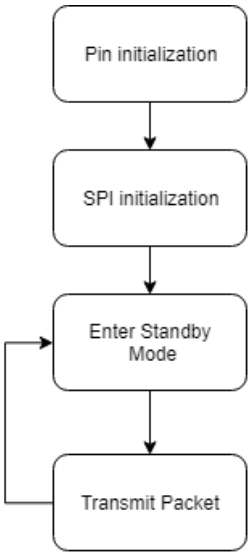


Figure 4.18 – Procedure to send a RF packet

Pin initialization First, we need to specify in the microcontroller the function of each pin. For this case all of them are GPIOs except for the serial clock, data in and data out; which are used in SPI mode.

No that Slave Select signals are also in GPIO, this is because this model of MSP may use only one SS in SPI mode, so if we want to control more slaves the most appropriate solution is to handle these signals as digital outputs

SPI transfers & register initialization

This device needs a previous configuration to select the frequency, modulation type, specifying the preamble, synchronization word, etc. For our case we will have 2 transceivers, one of them operating at 145.9 MHz and the other one at 433 MHz, so we will have 5 SPI signals. Fortunately, we will not need to implement SPI as we did for SCCB, the microcontroller has a set of higher-level functions that allow us to send and receive SPI messages easily.

When programming the camera we had to write the configuration file by searching in the application notes which registers to initialize to which value; but there is an interesting application made by Silicon Labs called Wireless Development Suite or WDS (Figure 4.19) in which you may specify the desired configuration in a more intuitive way and it automatically generates a header file with the messages that should be sent to the module via SPI to get that configuration.

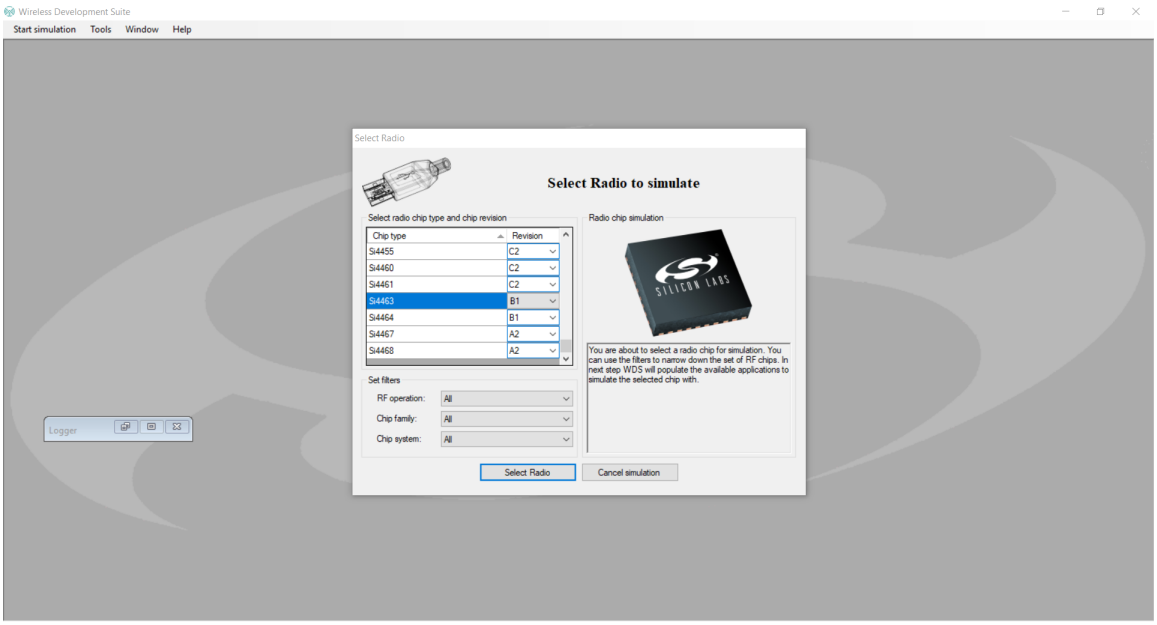


Figure 4.19 – Wireless Development Suite (WDS) view

The only remaining task is to write a set of functions to make these transactions. Fortunately, developers of the FloripaSat-I project uploaded in Github their code, so we could take this library as an starting point and adapt it to our project. This adaptation consisted in changing the USCI ports that were being used, SPI configurations and handling the SS signals, as for their case one of the RF modules is controlled directly by a different board, so they have a single device controlled with SPI.

These are the options that we will use for the transmissions:

- Carrier frequency: 145.9 MHz
- Modulation type: 2-FSK
- Frequency deviation: 0 Hz
- Preamble: 0xAA, 0xAA, 0xAA, 0xAA
- Synchronization word: 0x5D, 0xE6, 0x2A, 0x7E
- CRC: Not used

Standby Mode

This is done by sending a *CHANGE STATE* command, which has the structure depicted in Figure 4.20 . Where "Wait for CTS" is the time it takes for the RF module to set the serial data out to 0xFF, notifying that it is ready to receive more data/commands.

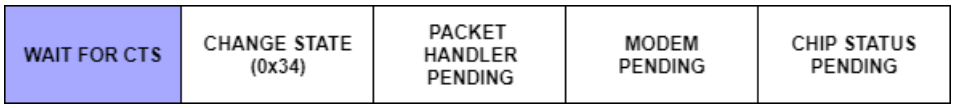


Figure 4.20 – Enter Standby Mode Packet

Transmit packet

Transmitting a packet requires 4 commands:

1. Reset TX FIFO - Empties the FIFO for not mixing the previous stored data with the fresh one. The packet is identical to the one in Figure 4.20, but the command is 0x15 instead.
2. Write TX FIFO with data - Fills the TX FIFO with new data. The structure is shown in Figure 4.21.

Figure 4.21 – Write TX FIFO Packet

3. Clear Interrupts - Reads the interrupt from the module, which may came from 3 sources: Packet Handler, Modem or Chip Status. To clear it we need to read the interrupt status of the source that generated it or read all of them with a single command, which is what we do in Figure 4.22.

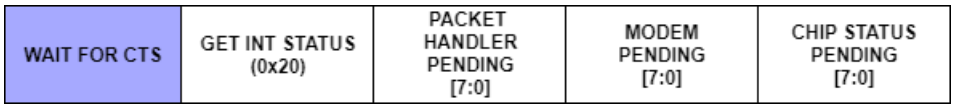


Figure 4.22 – Clear Interrupt Packet

4. Enter TX Mode - Finally, the module enters TX mode and sends the data saved in the FIFO (Figure 4.23).

Figure 4.23 – Start TX Packet

4.5 GNU Radio demodulator

The combination of the antenna, the SDR and GNU Radio will build the the simulation enviroment for the ground station. First, the antenna will transform the electromagnetic

wave into a voltage and then the resulting signal is passed to the SDR device. When used as a receiver, it performs the functions shown in Figure 4.24 and the contrary when transmitting (Figure 4.25).

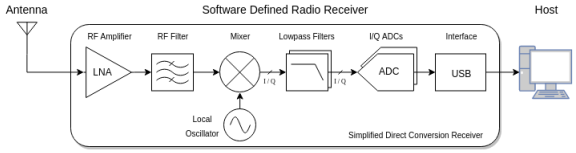


Figure 4.24 – SDR conversion from antenna to host PC

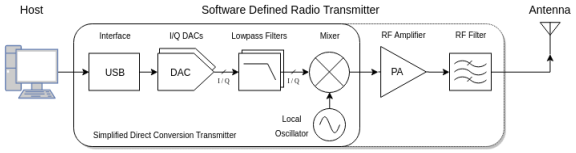


Figure 4.25 – SDR conversion from host PC to the antenna

When receiving, the signal is amplified with a Low Noise Amplifier (LNA) and then filtered. The tuning frequency is a parameter defined by the user and it is used in the SDR for doing the down conversion with a mixer and a low pass filter to suppress the harmonics. The I/Q components are then digitalized (the number of bits and hence the resolution depends on the SDR that is used) and sent to the host at a sample rate that should be lower than 2.8 MS/s to avoid problems, as mentioned in [7].

Hence, the input signal that we have in the framework is a digital complex base band signal with the specified sample rate. From this point we can design a normal FSK demodulator, like the one depicted in Figure 4.26 and taking elements from [8] and [9].

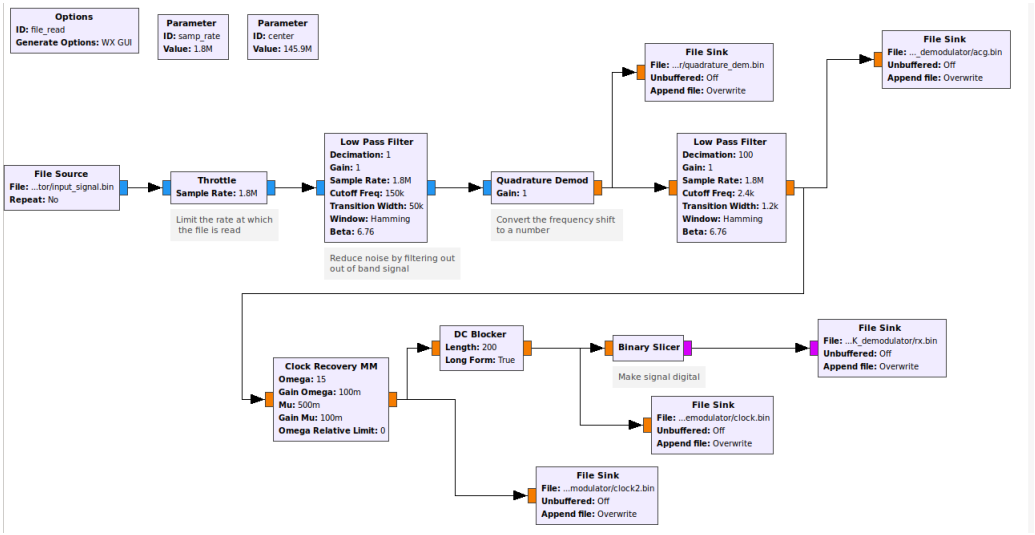


Figure 4.26 – FSK demodulator

The first low pass filter suppress out-of-band signals to enhance the performance of the receiver, and the result enters the quadrature demodulation block. Mathematically speaking, what this block does is to take a signal as an input and output its frequency. The digital output formula is:

$$y[n] = arg(x[n]\overline{x}[n-1])$$

When the input signal is a sinusoidal it can be represented as $x[n] = Ae^{j2\pi f/f_s n}$. Combining both expressions:

$$y[n] = arg(A^2 e^{j2\pi f/f_s n} e^{-j2\pi f/f_s (n-1)}) = arg(A^2 e^{j2\pi f/f_s n - j2\pi f/f_s (n-1)}) = arg(A^2 e^{j2\pi f/f_s}) = f/f_s$$

The next block has been problematic for the GNU Radio community; but it is necessary for asynchronous systems, as it recovers the phase and frequency in the transmitter side do recover the samples: M&M (Mueller and Muller) Clock synchronization. The problem about it is that from all its parameters, Omega is the only one that is easily calculated, as it is the symbol period expressed in samples per symbols (for our case the sample rate is 1.8e6 and the baud rate is 1.2 k symbols and we use a decimation of 100; meaning that Omega is $\frac{1.8e^6}{1200*100} = 15$. The rest of the parameters are set with the recommendations that we found and by trial and error.

Then, a DC blocker was added because a DC offset made the system to perform incorrect binary slicing. The problem about this block is that the length of the vector to extract the offset is delicate, as it will be shown in 5.3.2. Finally, a binary slicer recovers the bytes from the previous block and saves them in a binary file to be analysed afterwards.

A python script was written to read this binary file, detect the preamble, identify the synchronization word and show the message in the terminal. It was also uploaded to the Lab

repository, (link to the script).

5 Implementation & test

In this section we will explain the set-ups that were used to test the functionality of the subsystems, the problems encountered in each of them, their solutions and results.

5.1 I2C Tests

For these tests, an Arduino board was used to simulate the EPS/OBC. It is worth to mention that Atmel microcontrollers have an interesting feature. As we are using I2C and we already have a pull-up resistor soldered to the PCB for each line (SDA and SCL). It is possible to set pins in Arduino as inputs even if the board is the master and hence the one driving them because doing this disables the internal pull-up to 5 V (Figure 5.1).

Then, when the microcontroller integrated in the Arduino board needs to output a '1' it sets that port to high Z mode and thanks to the external pull-up (tied to 3.3 V instead), the line is set to 3.3 V. On the other side, when it wants to output a '0' it just pulls the line down to GND and sets it to 0 V. As a consequence, we do not need to place any level shifter in between.

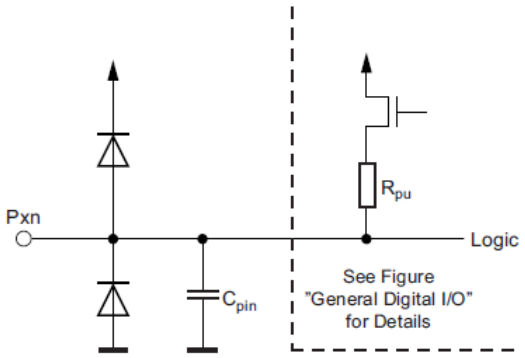


Figure 5.1 – Atmel-7810 I/O schematic

The codes that were used for the boards are also in the GitLab repository.

5.1.1 OBC - TTC

The Arduino code that simulated the OBC behaviour tests all the registers reserved for this module.

First, the board writes the first 9 data registers and then reads them to check that the operation was done correctly. Then, there are different actions that the OBC may trigger by writing the Configuration Register, and it is possible for the user to choose which one to activate by uncomment it in the code. All these actions are the ones shown in Table 4.1. We could verify that all these operations were triggered correctly.

The main issue that we encountered was that I2C did not seem to work when the clock was set to 8 MHz as for the rest of the tasks, and it took some time for us to realize that because we could not imagine that this would be important. In fact, still we do not know why it does not work for 8 MHz.

5.1.2 EPS - TTC

This test is much simpler than the previous one. The main difference is that Arduino will act as the slave now, and it will wait for read requests coming from the master to answer them.

The I2C library used for the master is generic and allows the user to write registers; however this feature would not be activated in the final design and the EPS would also discard these type of requests.

5.2 Camera tests

5.2.1 First Camera Set-up

Our first camera setup is the one shown in Figure 5.2, which consist of the camera plugged in the PCB board and the SCCB pins connected to an Arduino board due to some initial problems that we had with this protocol.

At first I2C implementation from *driverlib* library was used to initialize the camera; however, after debugging we realized that the problem was that I2C and SCCB are not compatible protocols.

They are extremely similar, but the only difference is the ACK bit, which for SCCB is a don't care bit. As a result, some of the transmissions were ok but when this bit was '0' (a NACK), the master interpreted it as if the slave was waiting for more data so the microcontroller automatically sent the same data again, as seen in Figure 5.3.

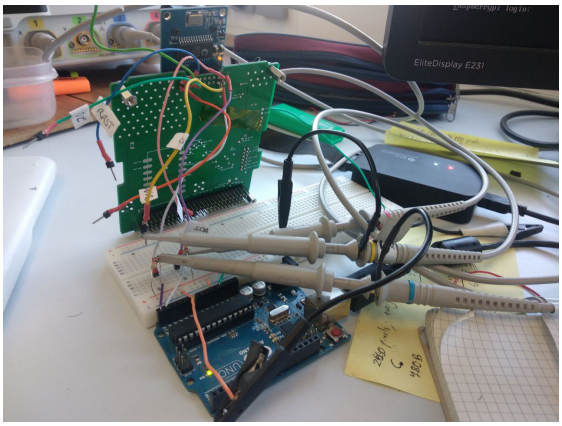


Figure 5.2 – First camera setup

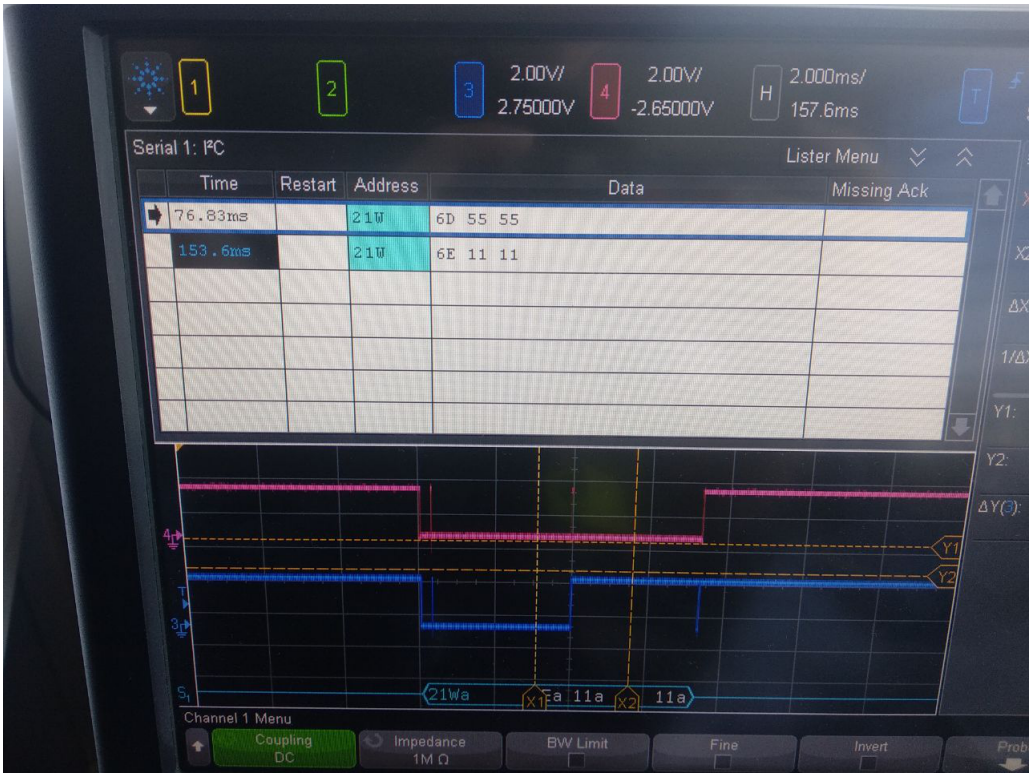


Figure 5.3 – I2C writing 2 consecutive registers due to protocol incompatibility

This is problematic, because when this happens, the camera writes in 2 consecutive registers. Taking the previous picture as an example, register 0x6D would be written 0x55 and the final byte sent would write 0x55 in register 0x6E.

Both protocols send 9 clock pulses and data bits in each phase. During the first transaction the first 7 bits the slave address is sent, then the 8th one encodes the operation (read = '0' and write = '1'); for the second and third transaction there are 8 bits of data, but the 9th bit is the

problematic one. In I2C the master waits for the slave to ACK the transmission by pulling down the data line, and when received the master knows that the data has been received correctly and may proceed; however, SSCB does not always do this because this last bit is a don't care no matter if the data was received by the camera.

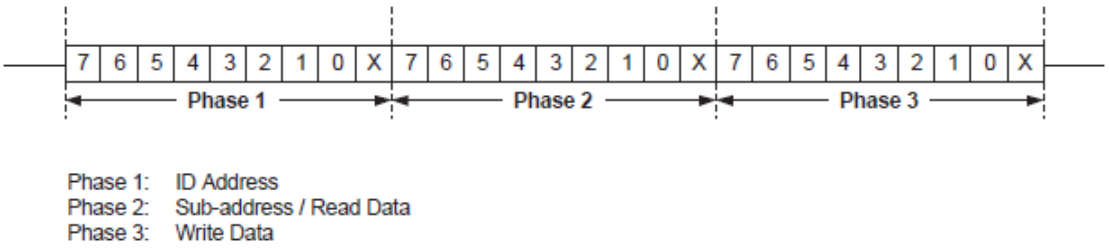


Figure 5.4 – Writing with SCCB protocol

The problem with the microcontroller implementation of I2C is that it was created in such a way that these ACKs are necessary to continue the protocol, as it would go on sending data until it is ACKed.

At this point there were 2 possible solutions, the first one consisting in implementing I2C protocol using pins as GPIOs in such a way that these 9th bits are ignored (SCCB protocol) or initialize the camera in a different way. The first solution was picked as an starting point because it would allow us to fix it quickly and make sure that everything else was working properly and it was always possible to come back to this and implement the other solution.

Arduino implements an "I2C" protocol (for the device it is called Wire) that is much more permissive, and does not waits for ACKs to arrive, it just notifies if they are received. So, what we did in this first test was to connect the Arduino board to the SCCB pins of the camera to configure its internal registers, and this proved to be working.

The Arduino code that was used is located here.

While doing the test we encountered 2 other issues that we will comment in the following subsections:

VGA resolution issue

The procedure shown in 4.6 proved to be working; but it presents a problem because the picture is first saved in memory and then it is read. The FIFO may store 384 KB (393216 bytes), and taking into account that in RGB565 and YUV422 each pixel is 2 bytes long we can calculate the image size:

Table 5.1 – Images size

Resolution	Dimensions	Bytes per pixel	Total size (bytes)
qqVGA	160*120	2	38400
qVGA	320x240	2	153600
VGA	640x480	2	614400

As shown in Table 5.1, the only 2 resolutions that allow to store a whole image in the FIFO are qqVGA and qVGA, and in order to take VGA pictures we should read the FIFO while writing it.

UART configuration and buffer size considerations

At the beginning of the python script the baudrate must be indicated so that transmitter and receiver are well-synchronized; however, there is actually a problem if the baudrate is too high because the receiver UART buffer located in the PC may read data at a slower rate and after some time it could overflow and lose some pixels. This is supposed to be easy to control in Linux distributions as this buffer size can be increased as it is desired; but for Windows it is not that simple and I had to solve it by introducing delays between UART transaction, which caused a lot of trouble in an experiment that was done later when trying to send an image via RF.

5.2.2 Second Camera Set-up

Once we managed to take proper pictures with the camera we came back to the register initialization protocol and implemented SCCB, which for now is only able to write. So now we do not need any external device to initialize the camera and everything can be handled by the MSP. All the code for the camera can be found on this folder. The only thing that remained unsolved was the VGA images problem.

5.2.3 Results

First, we tried to plot the test pattern saved in the camera to make sure that the decoding in RGB565 was done correctly and the result was the following one:

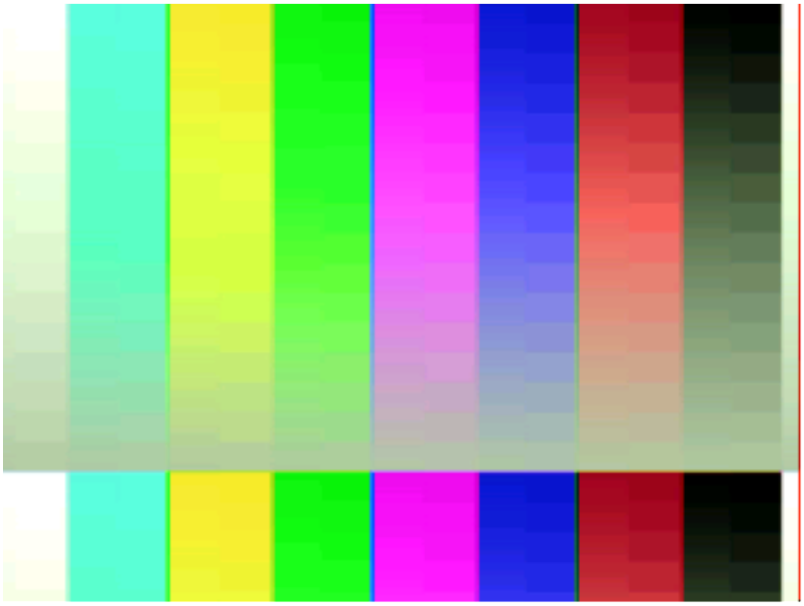


Figure 5.5 – OV7670 test pattern

This seemed to have a problem due to the weird faded green line that appears in the middle of it; however, after deactivating the test pattern and taking a picture we observed that it was working fine. That green line looked like something intentional, as we tried this with other 2 identical cameras and we observed the exact same pattern.

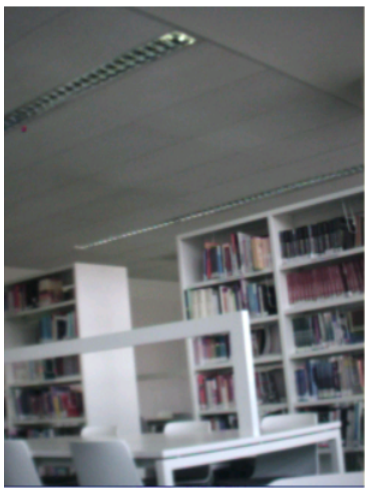


Figure 5.6 – First OV7670 picture

5.3 RF Module

5.3.1 First Set-up

A good starting point to check that the modules were correctly configured would be to use some example Arduino codes that the manufacturer provided (which were not correct and had to fix it later), fulfilling Requirement 1 as well, which is important for the lab because it eases them to use the module with a generic board to run other tests.

The first set-up consisted of 2 PCBs with a soldered RF module, 2 Arduinos (one for each module) and two level shifters to place between them because they were operating at different voltages and the signals do not have pull-ups as in 5.1. Then, one of the modules will start transmitting packets when a character is written in the first Arduino serial port and the other one will receive, demodulate and show them in the serial port of the receiving module. This set-up is depicted in Figure 5.7.

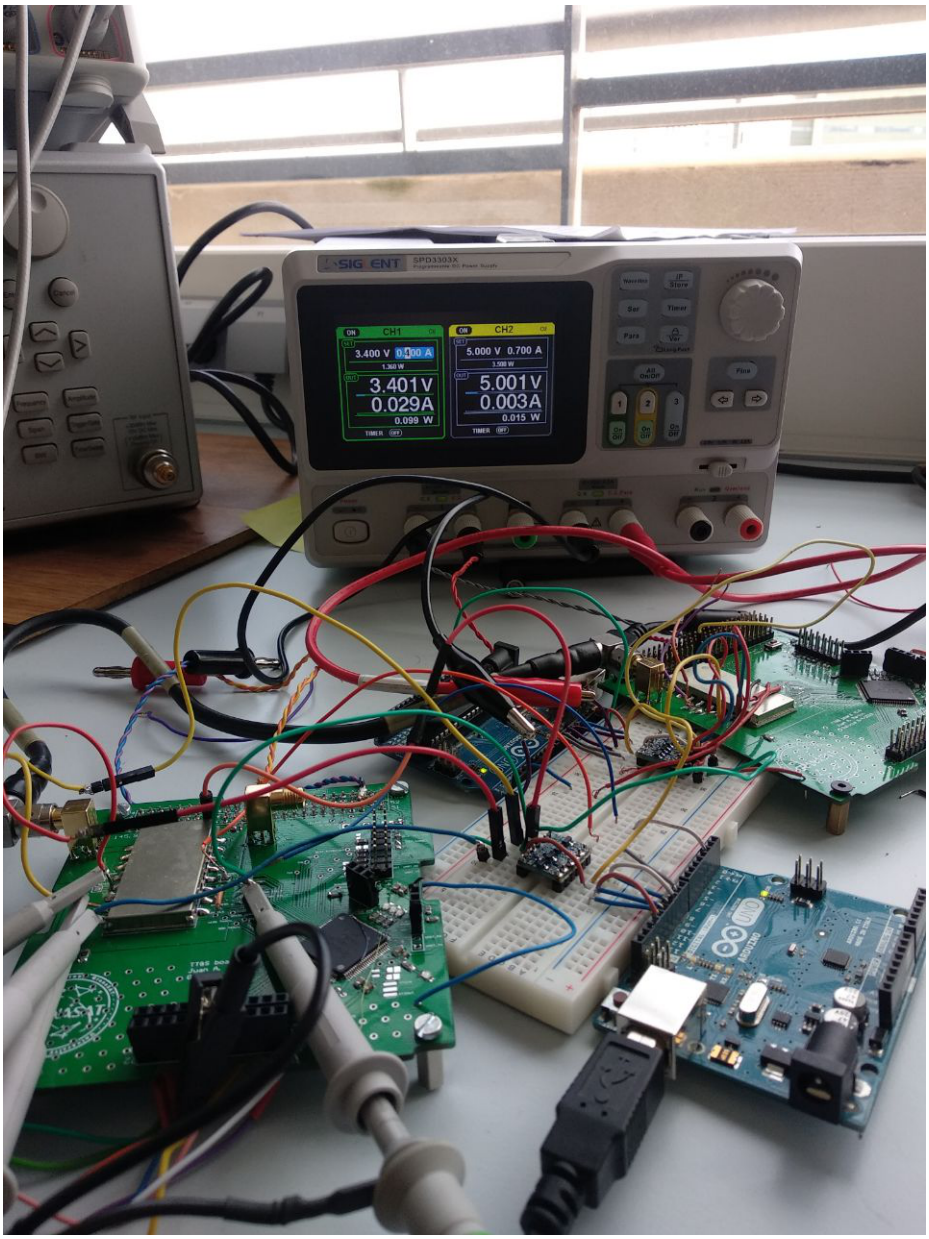


Figure 5.7 – Transmitter and receiver controlled by Arduino boards

The transmitter was running *rf6643_master* and the receiver *rf4463_rx*, which can be found in this link. We could first verify that it was receiving and processing SPI messages thanks to a command: *PART INFO* (Figure 5.8), which is answered by the module. If bytes *PART[15:0]* from this answer were 0x4463 we could state that the module is responsive.

- Command Stream

PART_INFO Command	7	6	5	4	3	2	1	0
CMD	0x01							

- Reply Stream

PART_INFO Reply	7	6	5	4	3	2	1	0
CTS	CTS[7:0]							
CHIPREV	CHIPREV[7:0]							
PART	PART[15:8]							
PART	PART[7:0]							
PBUILD	PBUILD[7:0]							
ID	ID[15:8]							
ID	ID[7:0]							
CUSTOMER	CUSTOMER[7:0]							
ROMID	ROMID[7:0]							

Figure 5.8 – *PART INFO* command

After verifying that the modules were responsive, the RF test was carried and it proved to be working, so the modules were correctly configured. The carrier frequency was the expected one for the 145.9 MHz module and messages sent in the transmitter were correctly demodulated in the receiver board (Figure 5.9). It was responding correctly to *PART INFO* command; however, we were not able to finish the same RF test with 433 MHz modules successfully. The only things that changed were the modules and the configuration file, so it is very likely that this file is the problem, but even that file was automatically generated by WDS, and this was done several times; and in the end we were not able to fix this.

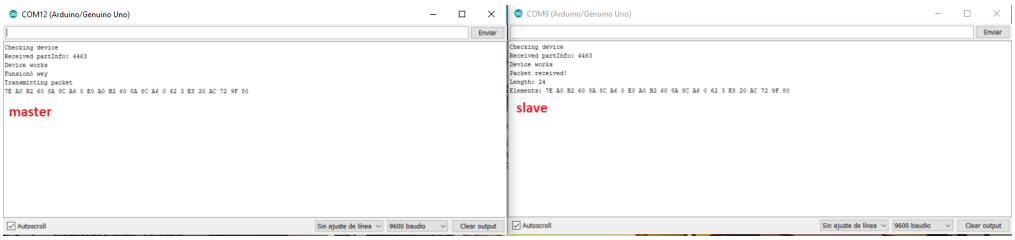


Figure 5.9 – Serial ports in Tx and Rx Arduino boards

The next step will be to implement the same with the microcontroller and now that we know that the transmitter is functional, we can program the ground station in GNU Radio to receive and decode packets.

5.3.2 Second Set-up

Transferring the code to the microcontroller was not a difficult task, as we had a library to configure radio modules uploaded in FloripaSat-I Github, which we modified to fit our architecture.

When transmitting we realized that power consumption was 0.5 mA as stated in the datasheet,

but in every mode and it was not responsive. After debugging it we realized that the source of the problem was the supply voltage. When we lowered it to 3.3 from 5 V we could observe that after changing the module to standby mode the module responded *PART INFO* messages and we could see the spikes centered at 145.9 MHz in the spectrum analyser.

At this point we are able to try the demodulator shown previously in Figure 4.26. In this experiment 6 packets in total will be sent. The first one is different from the others and after an small delay the next 5 are sent.

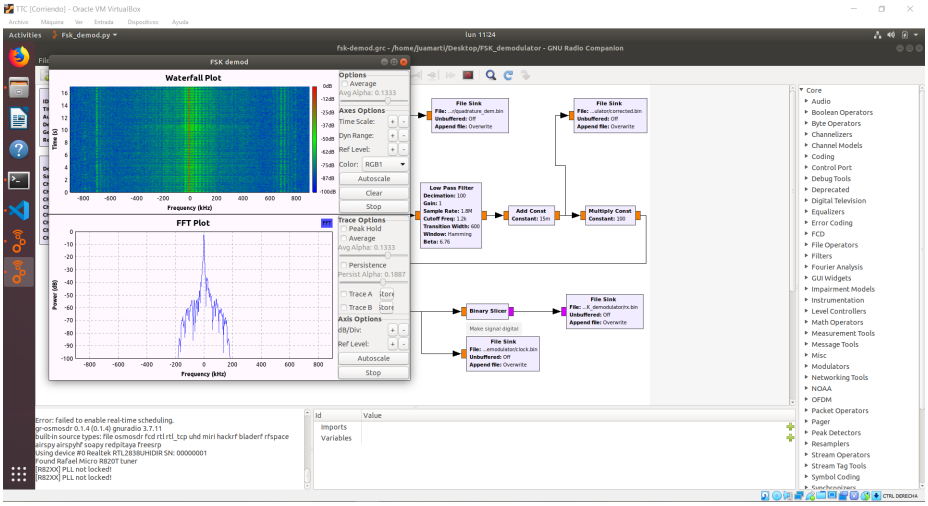
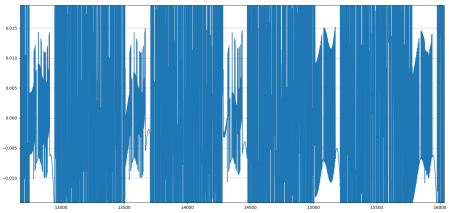
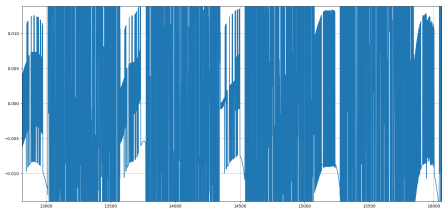


Figure 5.10 – Waterfall capture from GNU Radio

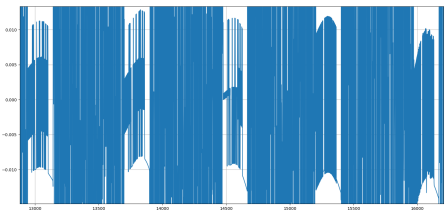
In Section 4.5 we described this demodulator, and one of the problems we solved was the DC offset problem with an add constant block, which worked indeed but was not the optimal solution because we should know what this offset is and it could also change. So we improved it by substituting it with an alternative block called *DC Blocker*. It receives the signal as an argument and the length of the delay line used for determining the DC level and outputs the corrected signal. This last parameter is critical for the system to work. In Figures 5.11 we show the signal after the clock synchronization and offset correction. For 5.11a and 5.11b the signal is distorted , and the python scrip missed 3 of the packets because it did not find the preamble. When it is high enough, which was the case for the other 2 length that we tried (5.11c and 5.11d), packets are correctly demodulated.



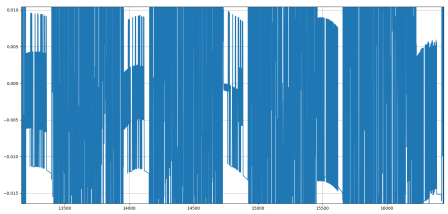
(a) DC Blocker with Length of 32 bits



(b) DC Blocker with Length of 64 bits



(c) DC Blocker with Length of 128 bits



(d) DC Blocker with Length of 256 bits

Figure 5.11 – Packets with different DC Blocker Lengths

The python script plays a fundamental role in this test, as it determines if a packet was correctly demodulated by checking the synchronization word. In fact, at first this was a source of problems because we did not considered that before the preamble the bits extracted from the noise could get confused by the beginning of the preamble, so the program would consider that the packet starts earlier and it would fail to identify the synchronization word. This made us believe that the demodulator was wrong for a long time, when the problematic element was this script.

5.4 Final Set-up

The last experiment done was to combine all these previous tests: first, an Arduino would simulate an OBC and send a command to the TTC board to take a picture with qVGA resolution that would then be sent via RF; then received and demodulated with GNU Radio. This set-up is shown in Figure 5.12.

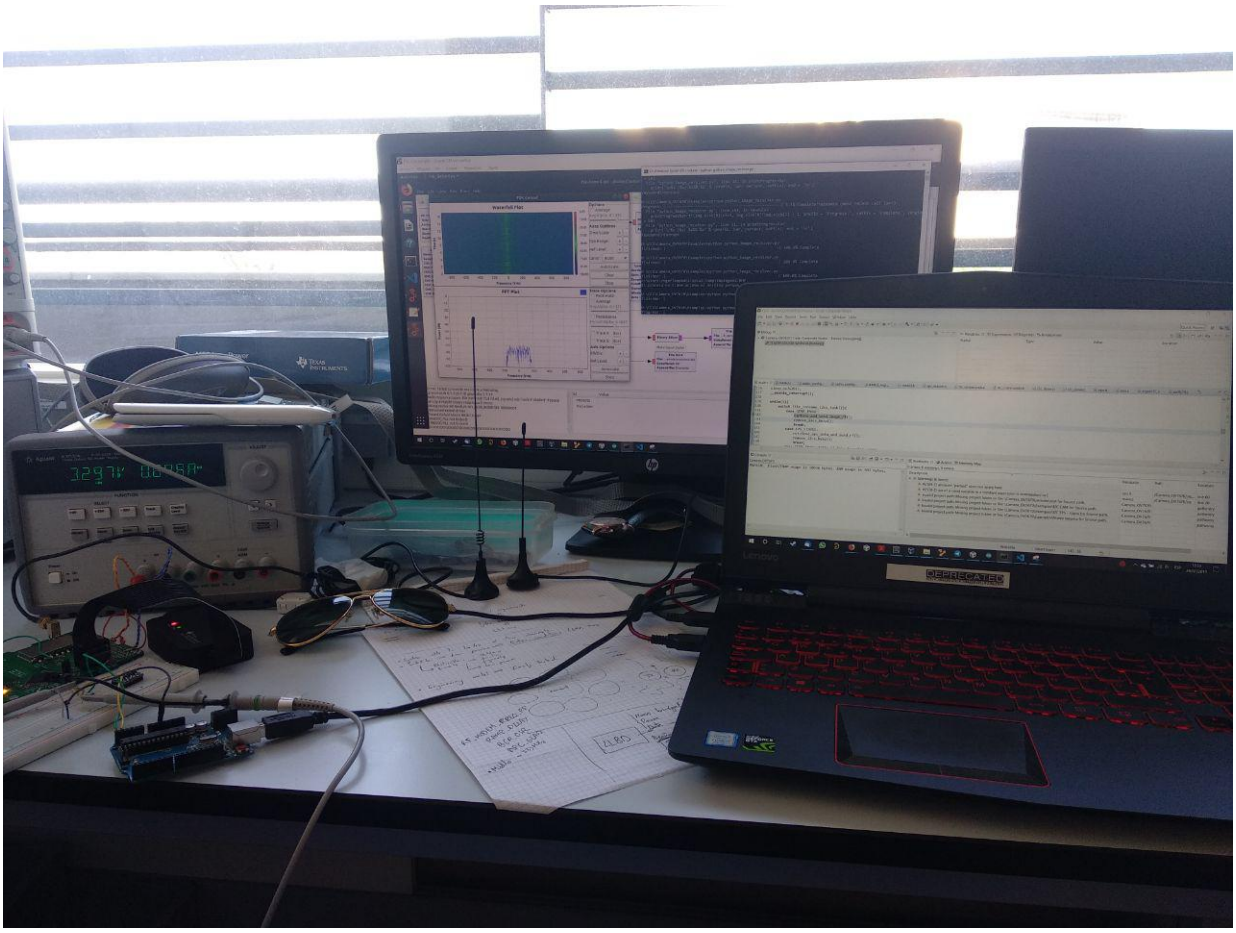


Figure 5.12 – Final Set-up

Transmission FIFO is limited to 20 bytes, which means that the number of packets to transmit a whole picture in qVGA is: $\frac{320*240*2}{20} = 7680$.

By using a DC blocker with a length of 512 bits we could demodulate only 5960 out of those packets, so we started incrementing the length to see if this result improved, and it did as proved in Figure 5.13.

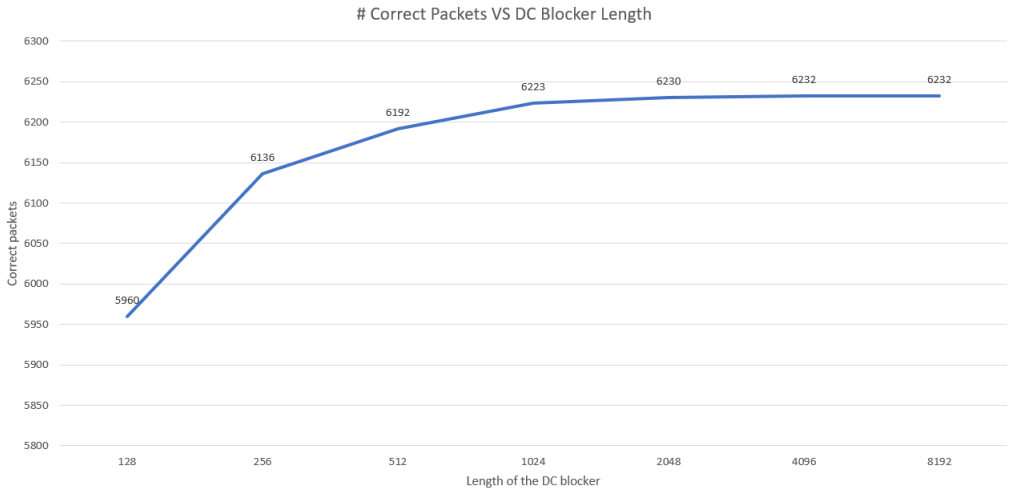


Figure 5.13 – Number of correct packets VS DC blocker length

In the end only 81 % of the packets were correct, which is not a good result.

6 Conclusions & Future Work

In the end we were able to fulfil all the primary requirements but 1: verify transmissions at 433 MHz. However, we are extremely close to achieve this, because the module itself is responsive, so probably it is due to a misconfiguration. The receiver would only need to be centred at 433 MHz and the rest would be the same once we fix this second RF module.

For the secondary requirements, we were also able to fulfil all of them but for Requirement 4. To reprogram the module as a receiver should not be difficult as we already have the libraries and the Arduino code, but it would be to create and debug a transmitter in GNU Radio to verify it.

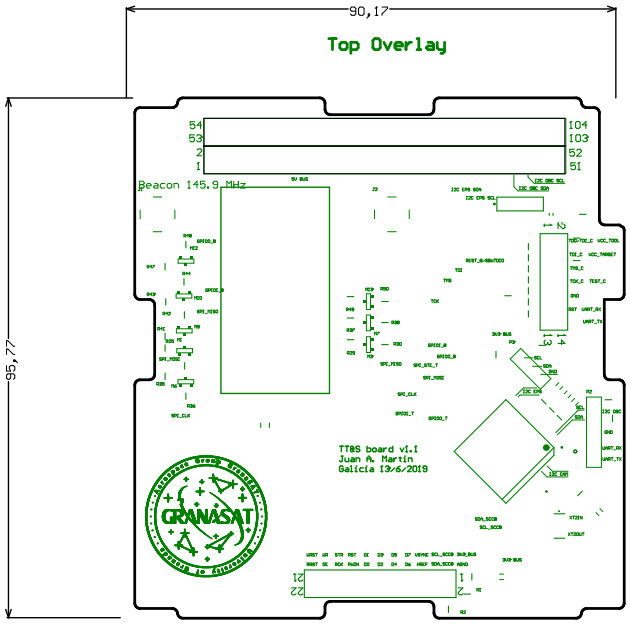
The design could be improved in the future by achieving these missing requirements, which would be the most important pending tasks, but there are also some other improvements that could be made.

When sending the image over RF we realized that it was taking too long (sending it in qVGA format, meaning that we were sending 153.5 kB over 7680 packets took more than half an hour). We could try to increase the data rate at the transmitter and receiver side to check if we can achieve faster transmissions, as an starting point.

In the previous chapter we showed that 81 % of the packets transmitted are correctly received and demodulated, which is not a good result because the experiment was done in optimal conditions. First we would need to demodulate all the packets in this scenario and then apply other effects in the channel that could make it more difficult such as longer distances, higher noise power and multipath. Then, we could try to add CRC to the messages to reconstruct single bit errors and a protocol to retransmit lost packets.

Finally, another improvement that could be made, as mentioned in Section 5.2.1, up to now we can only take pictures with low and medium resolution (qqVGA and qVGA) because it was first saved in the FIFO and then sent back to the microcontroller; however, there must be a different way to do this in order to capture VGA images.

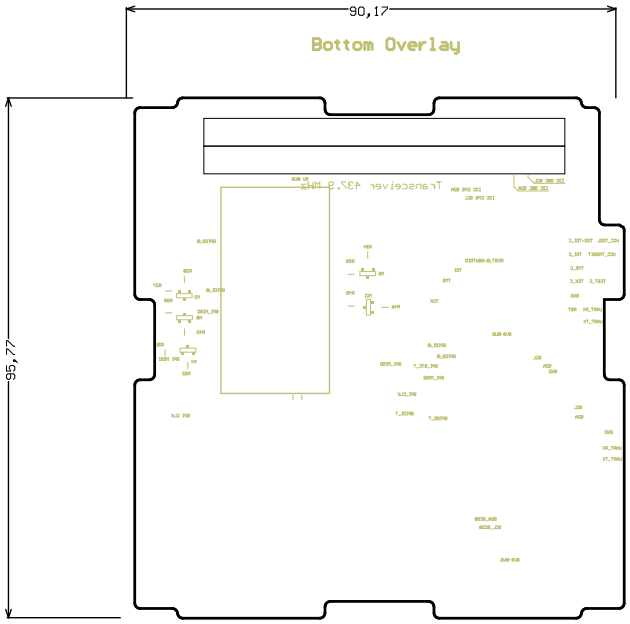
7 Appendix A - Final PCB



Colour Legend

Red: Top Layer
Blue: Bottom Layer
Dark Green: 3V3 BUS
Light blue: 5V BUS
Pink: Ground
Grey: USCI
Light green: I2C EPS
Orange: I2C OBC
Magenta: I2C camera
Brown: UHF ground
Yellow: UHF ground
Turquoise: SPI

Designer's signature:	Sheet title: TT&C - PCB	Dpto. Electronica y Tecnologia de Computadores University of Granada C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain Sr. Andres Roldan Aranda	
	Project title: GranaSAT-I		
Supervisor's signature:	Designer: Juan Antonio Martøn Galicia		
	Supervisor: Andres Roldan Aranda	Date: 27/03/2019 Revision: 2 Sheet X	

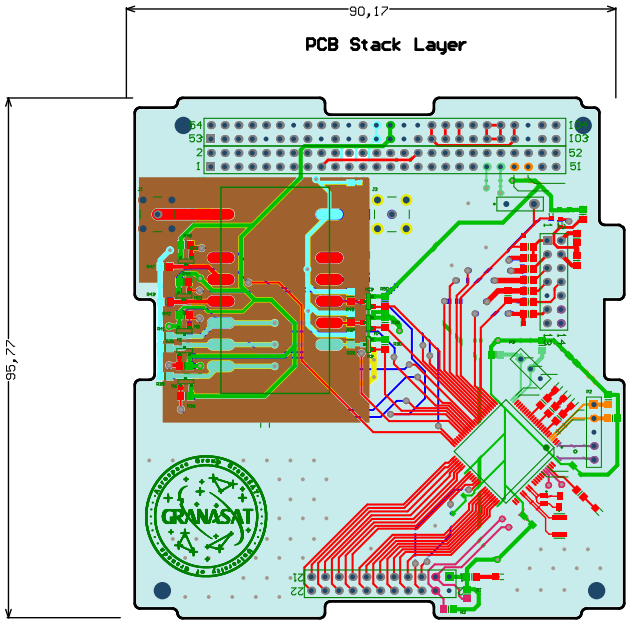


Colour Legend

Red: Top Layer
Blue: Bottom Layer
Dark Green: 3V3 BUS
Light blue: 5V BUS
Pink: Ground
Grey: USCI
Light green: I2C EPS
Orange: I2C OBC
Magenta: I2C camera
Brown: UHF ground
Yellow: UHF ground
Turquoise: SPI

Designer's signature:	Sheet title: TT&C - PCB	Dpto. Electronica y Tecnologia de Computadores University of Granada C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain Sr. Andres Roldan Aranda
	Project title: GranaSAT-I	
Supervisor's signature:	Designer: Juan Antonio Martøn Galicia	
	Supervisor: Andres Roldan Aranda	Date: 27/03/2019 Revision: 2 Sheet X

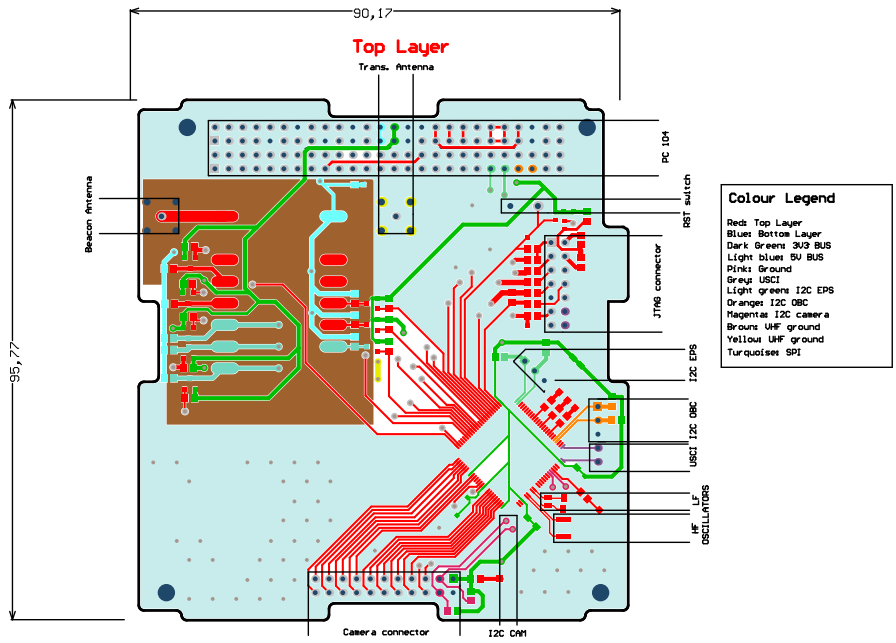
PCB Stack Layer



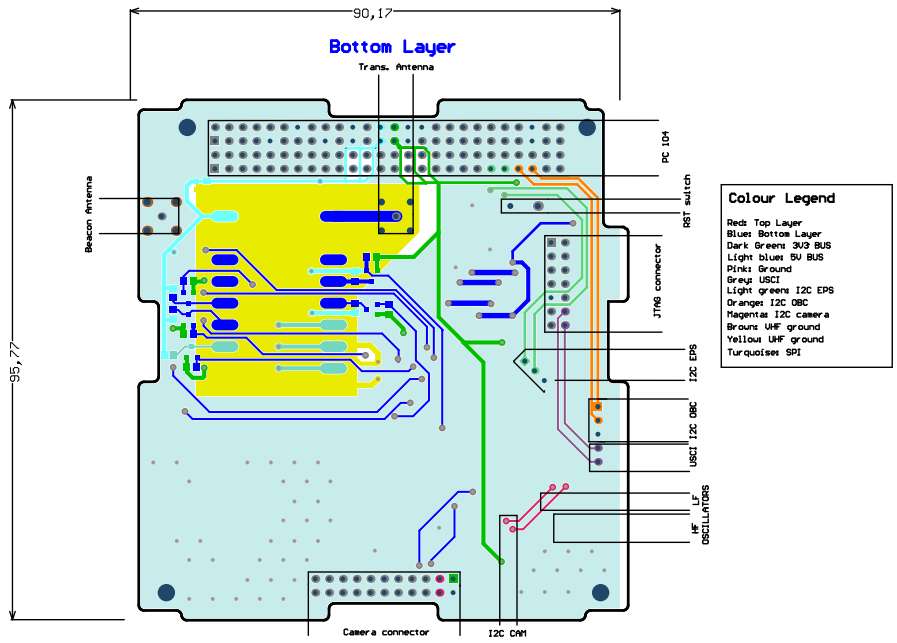
Colour Legend

Red: Top Layer
Blue: Bottom Layer
Dark Green: 3V3 BUS
Light blue: 5V BUS
Pink: Ground
Grey: USCI
Light green: I2C EPS
Orange: I2C OBC
Magenta: I2C camera
Brown: UHF ground
Yellow: UHF ground
Turquoise: SPI

Designer's signature:	Sheet title: TT&C - PCB	Dpto. Electronica y Tecnologia de Computadores University of Granada C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain Sr. Andres Roldan Aranda
	Project title: GranaSAT-I	
Supervisor's signature:	Designer: Juan Antonio Martøn Galicia	
	Supervisor: Andres Roldan Aranda	Date: 27/03/2019 Revision: 2 Sheet X

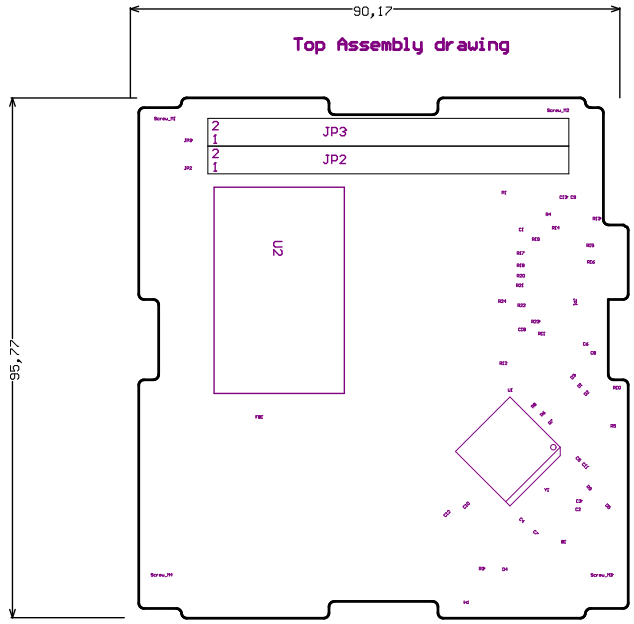


Designer's signature:	Sheet title: TT&C - PCB	Dpto. Electronica y Tecnologia de Computadores University of Granada C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain Sr. Andres Roldan Aranda
	Project title: GranaSAT-I	
Supervisor's signature:	Designer: Juan Antonio Martøn Galicia	
	Supervisor: Andres Roldan Aranda	Date: 27/03/2019 Revision: 2 Sheet X



Designer's signature:	Sheet title: TT&C - PCB	Dpto. Electronica y Tecnologia de Computadores University of Granada C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain Sr. Andres Roldan Aranda
	Project title: GranaSAT-I	
Supervisor's signature:	Designer: Juan Antonio Martøn Galicia	
	Supervisor: Andres Roldan Aranda	Date: 27/03/2019 Revision: 2 Sheet X

Top Assembly drawing

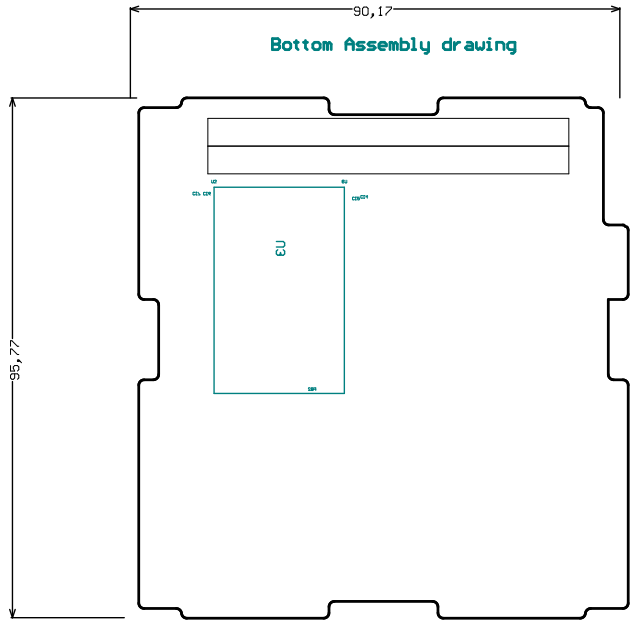


Colour Legend

Red: Top Layer
Blue: Bottom Layer
Dark Green: 3V3 BUS
Light blue: 5V BUS
Pink: Ground
Grey: USCI
Light green: I2C EPS
Orange: I2C OBC
Magenta: I2C camera
Brown: UHF ground
Yellow: UHF ground
Turquoise: SPI

Designer's signature:	Sheet title: TT&C - PCB	Dpto. Electronica y Tecnologia de Computadores University of Granada C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain Sr. Andres Roldan Aranda
	Project title: GranaSAT-I	
Supervisor's signature:	Designer: Juan Antonio Martøn Galicia	
	Supervisor: Andres Roldan Aranda	Date: 27/03/2019 Revision: 2 Sheet X

Bottom Assembly drawing

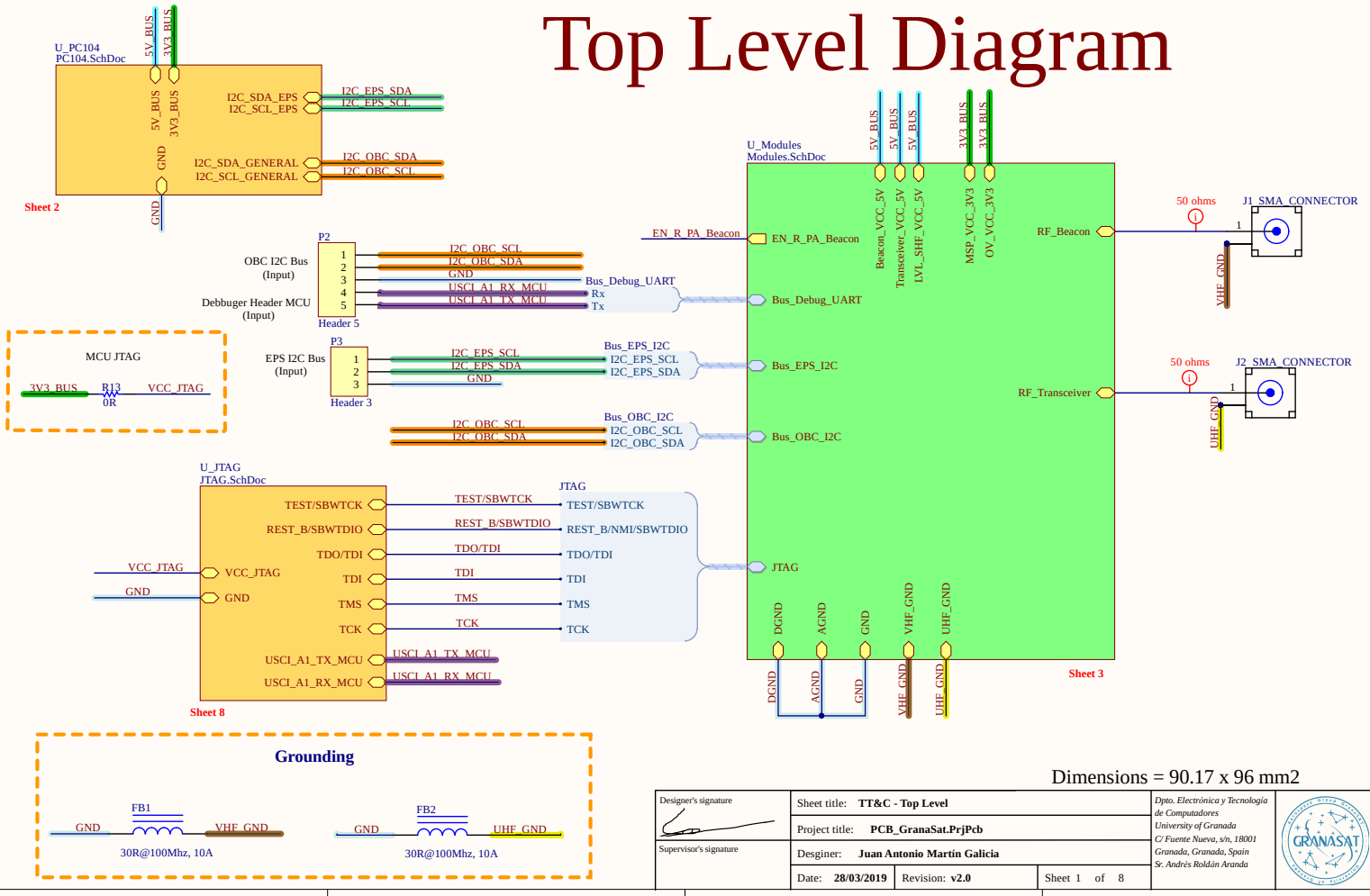


Colour Legend

Red: Top Layer
Blue: Bottom Layer
Dark Green: 3V3 BUS
Light blue: 5V BUS
Pink: Ground
Grey: USCI
Light green: I2C EPS
Orange: I2C OBC
Magenta: I2C camera
Brown: UHF ground
Yellow: UHF ground
Turquoise: SPI

Designer's signature:	Sheet title: TT&C - PCB	Dpto. Electronica y Tecnologia de Computadores University of Granada C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain Sr. Andres Roldan Aranda
	Project title: GranaSAT-I	
Supervisor's signature:	Designer: Juan Antonio Martøn Galicia	
	Supervisor: Andres Roldan Aranda	Date: 27/03/2019 Revision: 2 Sheet X

Top Level Diagram

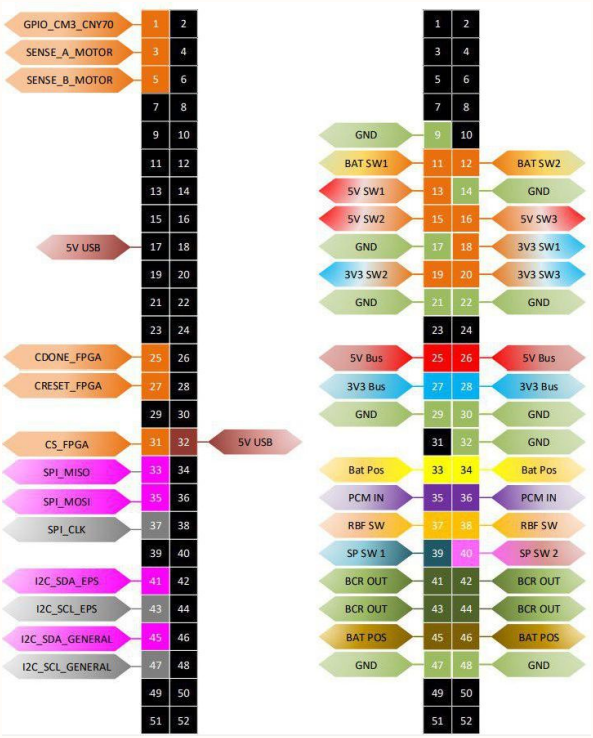


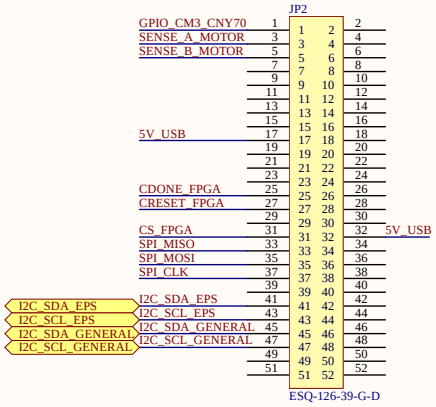
Dimensions = 90.17 x 96 mm2

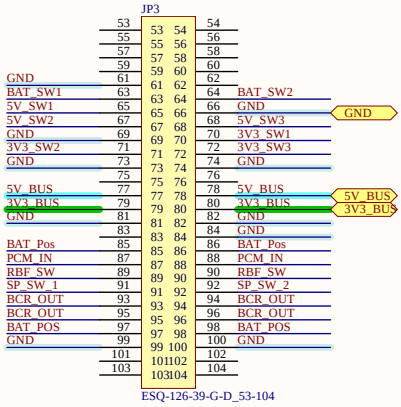
Designer's signature	Sheet title: **TT&C - Top Level**		*Dpto. Electrónica y Tecnología de Computadores*
Supervisor's signature	Project title: **PCB_GranaSat.PrjPcb**		*University of Granada*
	Desginer: **Juan Antonio Martín Galicia**		*C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain*
	Date: **28/03/2019**	Revision: **v2.0**	Sheet 1 of 8

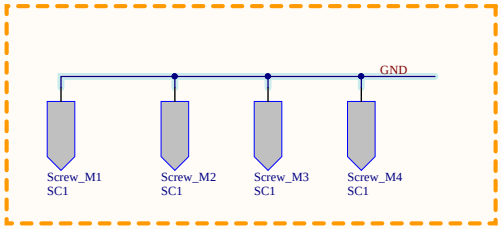
Sr. Andrés Roldán Aranda

Bus PC-104 Connections



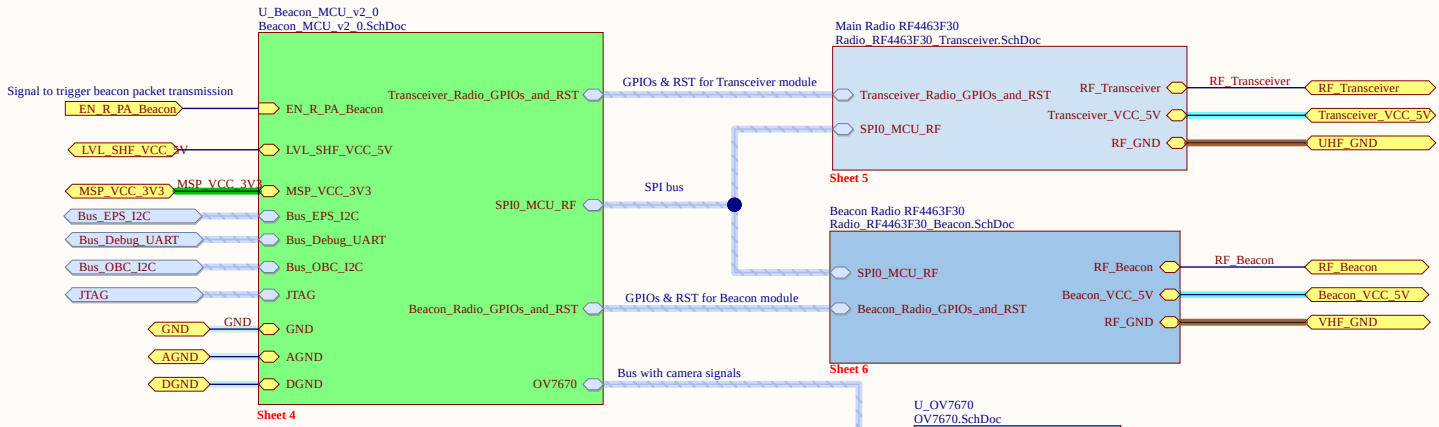




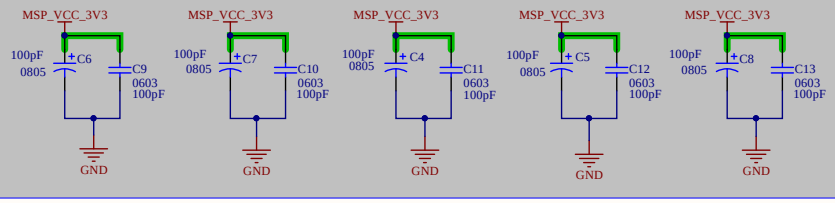


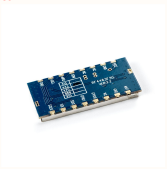
Designer's signature	Sheet title: **TT&C - PC104**	*Dpto. Electrónica y Tecnología de Computadores*
Supervisor's signature	Project title: **PCB_GranaSat.PrjPcb**	*University of Granada*
	Desginer: **Juan Antonio Martín Galicia**	*C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain*
	Date: **29/03/2019** Revision: **v2.0** Sheet 2 of 8	*Sr. Andrés Roldán Aranda*

Modules



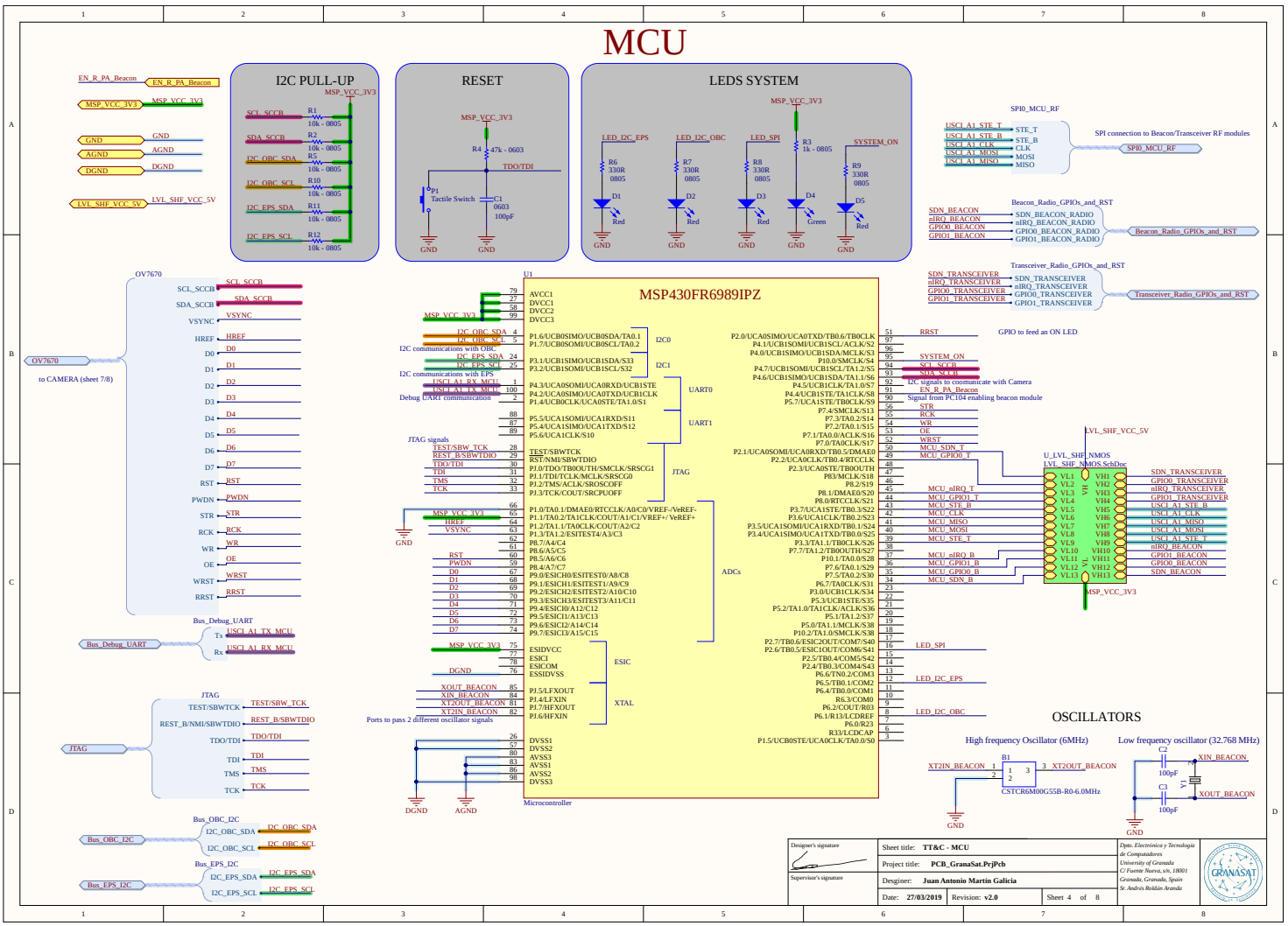
DECOUPLING CAPACITORS





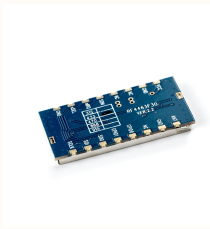
Designer's signature	Sheet title: **TT&C Modules**		*Dpto. Electrónica y Tecnología de Computadores*
Supervisor's signature	Project title: **PCB_GranaSat.PrjPcb**		*University of Granada*
	Desginer: **Juan Antonio Martín Galicia**		*C/ Fuente Nueva, s/n, 18001*
	Date: **28/03/2019**	Revision: **v2.0**	*Granada, Granada, Spain*
		Sheet 3 of 8	*Sr. Andrés Roldán Aranda*

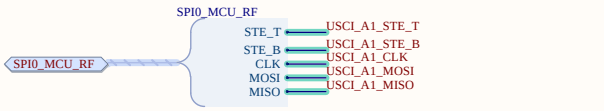
MCU



Designer's signature	Sheet title: **TT&C - MCU**	*Dpto. Electrónica y Tecnología de Computadores, University of Granada, C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain*
Supervisor's signature	Project title: **PCB_GranaSat.PrjPcb**	*Sr. Andrés Roldán Aranda*
	Desginer: **Juan Antonio Martín Galicia**	
	Date: **27/03/2019** Revision: **v2.0** Sheet 4 of 8	

Transceiver RF Module 437.9 MHz



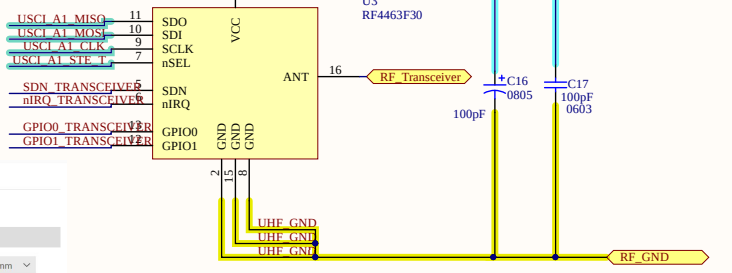


Pin Number	Pin Definitions	Description
1	VCC	Positive power supply
2	GND	Connected to power ground
3,4,14	NC	Vacant, not connected
5	SDN	Power down control. SDN = 1, power down SDN = 0, normal working
6	nIRQ	Interrupt output
7	nSEL	Serial data selection for SPI interfaces.
8	GND	Connected to power ground
9	SCLK	Serial data clock for SPI interface.
10	SDI	Serial data in for SPI interface.
11	SDO	Serial data out for SPI interface.
12	GPIO1	GPIO1 of Si4463
13	GPIO0	GPIO0 of Si4463 for I/O interface
15	GND	Connected to power ground
16	ANT	From 50 ohm coaxial antenna

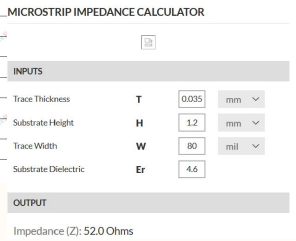
Current consumption					
Receiving current		13.5		mA	Low sensitivity mode
Receiving current		10		mA	High sensitivity mode
Transmitting current		500	550	mA	Vcc=5v,Tx=30dBm
Sleep current		<2	3	uA	
Output power range	29.5	30	31	dBm	433MHz，VCC = 5V
Receiving sensitivity	-125	-126		dBm	433MHz@data=600bps,Fdev=3kHZ

RF4463F30 Transceiver

- Operating frequency: 437.9 MHz
- Modulation: GFSK
- Transmission rate: 2.4 kbps
- Supply voltage: 3.3 - 6.5 V
- Dimensions : 38x20 mm2

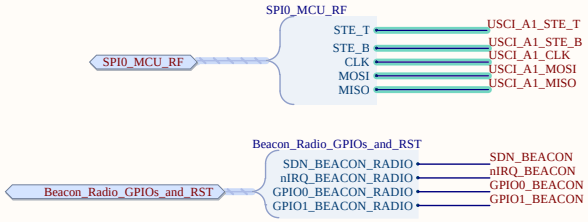


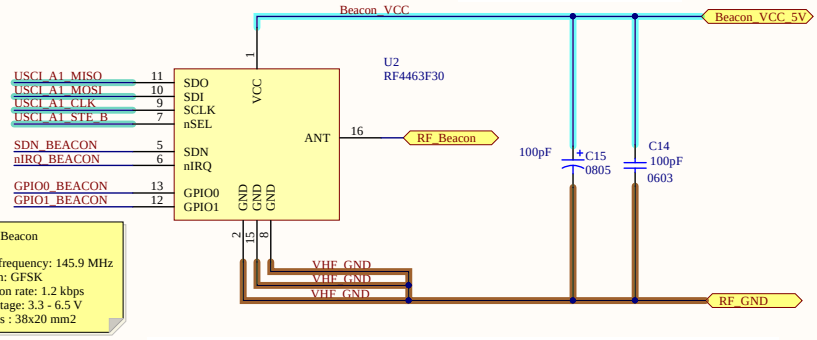
RF line impedance



Designer's signature	Sheet title: **TT&C Radio Transceiver 437.9 MHz**			*Dpto. Electrónica y Tecnología de Computadores*
Supervisor's signature	Project title: **PCB_GranaSat.PrjPcb**			*University of Granada*
	Desginer: **Juan Antonio Martín Galicia**			*C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain*
	Date: **28/03/2019**	Revision: **v2.0**	Sheet 5 of 8	*Sr. Andrés Roldán Aranda*

Beacon RF Module 145.9 MHz





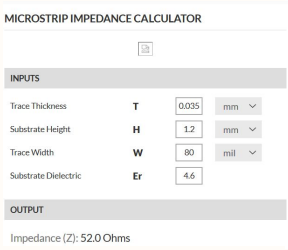
RF4463F30 Beacon

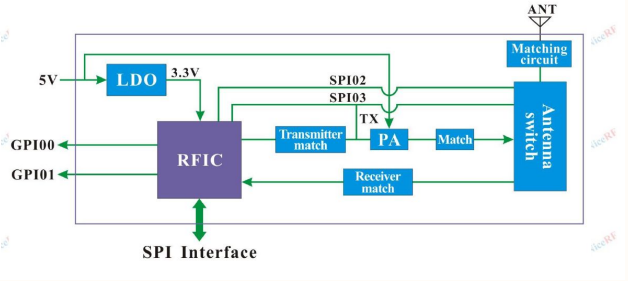
- Operating frequency: 145.9 MHz
- Modulation: GFSK
- Transmission rate: 1.2 kbps
- Supply voltage: 3.3 - 6.5 V
- Dimensions : 38x20 mm2

Pin Number	Pin Definitions	Description
1	VCC	Positive power supply
2	GND	Connected to power ground
3,4,14	NC	Vacant, not connected
5	SDN	Power down control. SDN = 1, power down SDN = 0, normal working
6	nIRQ	Interrupt output
7	nSEL	Serial data selection for SPI interfaces.
8	GND	Connected to power ground
9	SCLK	Serial data clock for SPI interface.
10	SDI	Serial data in for SPI interface.
11	SDO	Serial data out for SPI interface.
12	GPIO1	GPIO1 of Si4463
13	GPIO0	GPIO0 of Si4463 for I/O interface
15	GND	Connected to power ground
16	ANT	From 50 ohm coaxial antenna

Current consumption					
Receiving current		13.5		mA	Low sensitivity mode
Receiving current		10		mA	High sensitivity mode
Transmitting current		500	550	mA	Vcc=5v,Tx=30dBm
Sleep current		<2	3	uA	
Output power range	29.5	30	31	dBm	433MHz，VCC = 5V
Receiving sensitivity	-125	-126		dBm	433MHz@data=600bps,Fdev=3kHZ

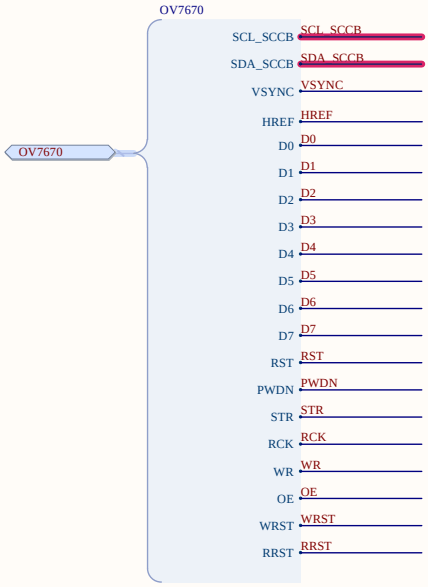
RF line impedance

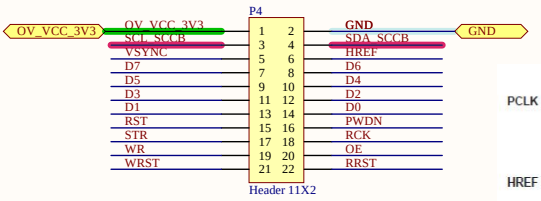




Designer's signature	Sheet title: **TT&C - Radio Beacon 145.9 MHz**		*Dpto. Electrónica y Tecnología de Computadores* *University of Granada* *C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain* *Sr. Andrés Roldán Aranda*
Supervisor's signature	Project title: **PCB_GranaSat.PrjPcb**		
	Desginer: **Juan Antonio Martín Galicia**		
	Date: **28/03/2019**	Revision: **v2.0**	Sheet 6 of 8

Camera Module - OV7670 & FIFO

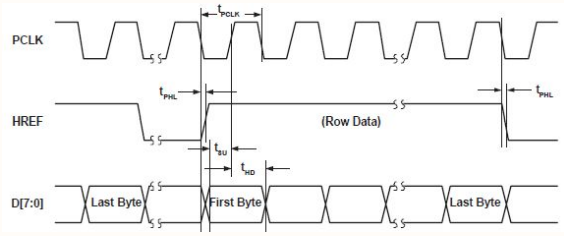




Specifications:

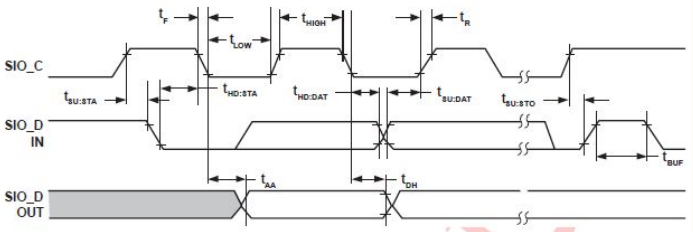
- Supply voltage: 3.3 V
- Energy consumption: 60mW/15 fps
- Buffer: AL422B
- Resolution: 640x480
- Current consumption in Sleep Mode: <20µA

Horizontal Timing



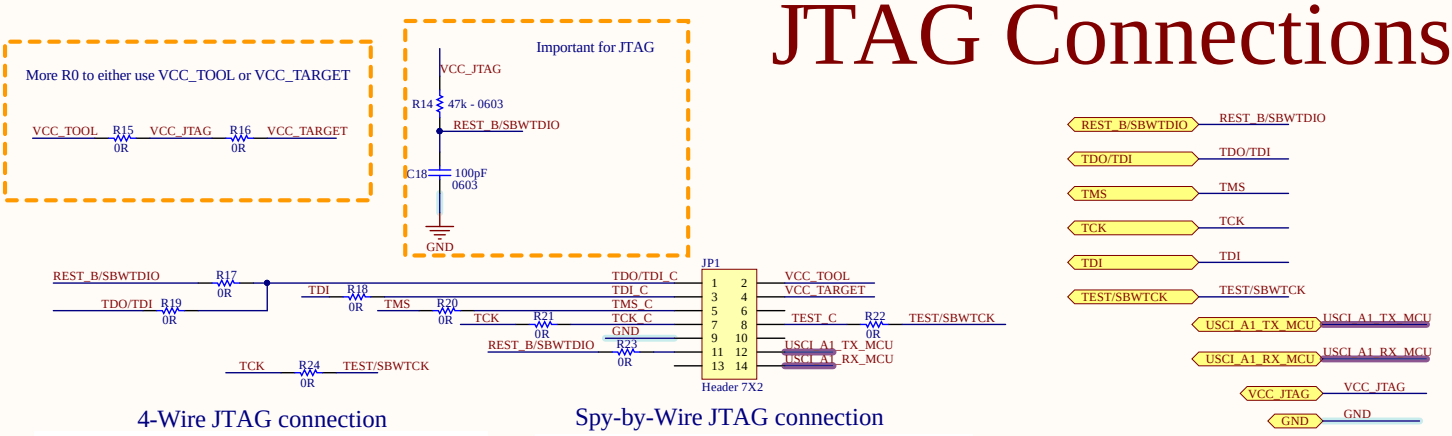
Inputs (PWDN, CLK, RESET)					
f_{CLK}	Input Clock Frequency	10	24	48	MHz
t_{CLK}	Input Clock Period	21	42	100	ns
$t_{CLK:DC}$	Clock Duty Cycle	45	50	55	%
$t_{S:RESET}$	Setting time after software/hardware reset			1	ms
$t_{S:REG}$	Settling time for register change (10 frames required)			300	ms
SCCB Timing (see Figure 4)					
f_{SIO_C}	Clock Frequency			400	KHz
t_{LOW}	Clock Low Period	1.3			µs
t_{HIGH}	Clock High Period	600			ns
t_{AA}	SIO_C low to Data Out valid	100		900	ns
t_{BUF}	Bus free time before new START	1.3			µs
$t_{HD:STA}$	START condition Hold time	600			ns
$t_{SU:STA}$	START condition Setup time	600			ns
$t_{HD:DAT}$	Data-in Hold time	0			µs
$t_{SU:DAT}$	Data-in Setup time	100			ns
$t_{SU:STO}$	STOP condition Setup time	600			ns
t_R, t_F	SCCB Rise/Fall times			300	ns
t_{DH}	Data-out Hold time	50			ns

I2C Timing

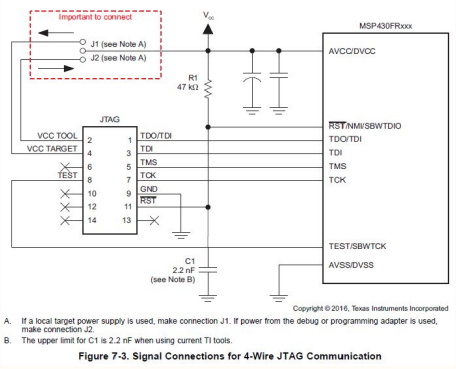


Designer's signature	Sheet title: **TT&C - Camera Module OV7670 & FIFO**		*Dpto. Electrónica y Tecnología de Computadores University of Granada C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain Sr. Andrés Roldán Aranda*
Supervisor's signature	Project title: **PCB_GranaSat.PrjPcb**		
	Desginer: **Juan Antonio Martín Galicia**		
	Date: **27/03/2019** Revision: **v2.0**	Sheet 7 of 8	

JTAG Connections



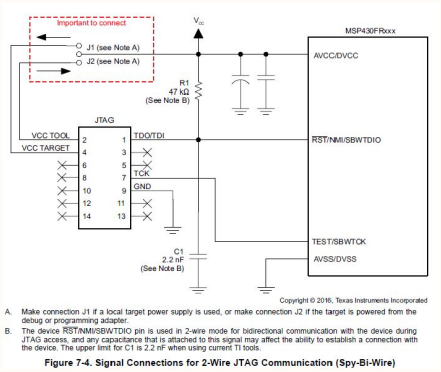
4-Wire JTAG connection



A. If a local target power supply is used, make connection J1. If power from the debug or programming adapter is used, make connection J2.
B. The upper limit for C1 is 2.2 nF when using current TI tools.

Figure 7-3. Signal Connections for 4-Wire JTAG Communication

Spy-by-Wire JTAG connection



A. Make connection J1 if a local target power supply is used, or make connection J2 if the target is powered from the debug or programming adapter.
B. The device RST/NMI/SBWTDIO pin is used in 2-wire mode for bidirectional communication with the device during JTAG access, and any capacitance that is attached to this signal may affect the ability to establish a connection with the device. The upper limit for C1 is 2.2 nF when using current TI tools.

Figure 7-4. Signal Connections for 2-Wire JTAG Communication (Spy-Bi-Wire)

There are 2 possible ways of connecting the JTAG to the MCU, one of them is known as Spy-By-Wire and the standard one, using 4 wires.

In order to try both of them, I placed dummy resistors making the connections, so to implement each JTAG mode an R0 resistor must be installed in:

SPY-BY-WIRE

Short resistors: R7, R12, R14

STANDARD JTAG

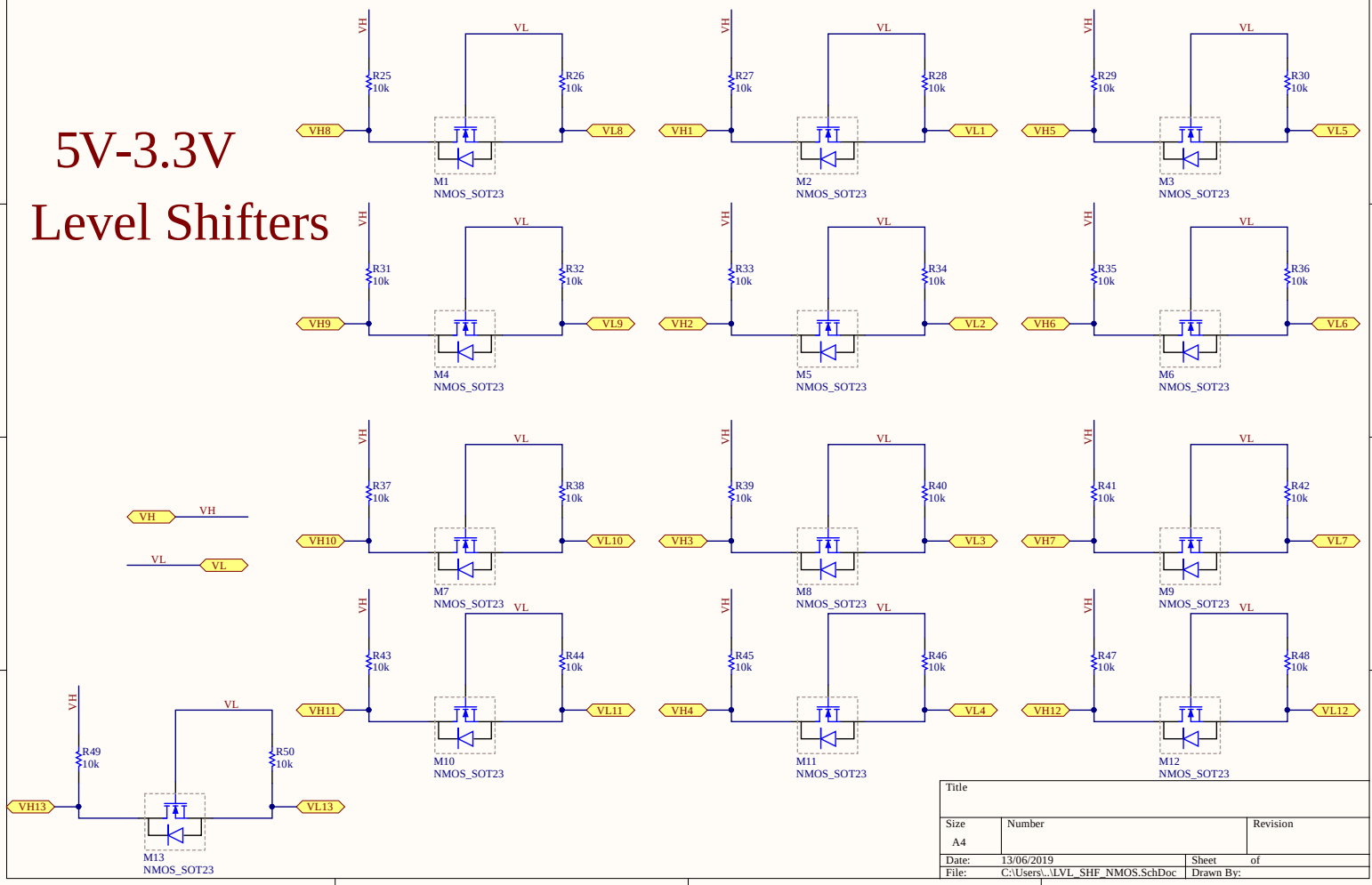
Short resistors: R8, R9, R10, R11, R13

RESISTOR NUMBERS MAY HAVE CHANGED IF FORCE ANNOTATE HAS BEEN USED

Designer's signature	Sheet title: **TT&C - JTAG**		*Dpto. Electrónica y Tecnología de Computadores*
	Project title: **PCB_GranaSat.PrjPcb**		*University of Granada*
Supervisor's signature	Desginer: **Juan Antonio Martín Galicia**		*C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain*
	Date: **28/03/2019**	Revision: **v2.0**	Sheet 8 of 8

Sr. Andrés Roldán Aranda

5V-3.3V Level Shifters



Title		
Size: A4	Number	Revision
Date: 13/06/2019		Sheet of
File: C:\Users\..\LVL_SHF_NMOS.SchDoc		Drawn By:

Bibliography

[1] Stack Overflow: How does one convert 16-bit RGB565 to 24-bit RGB888?, link to the page

[2] Visual Capturing with OV7670 on Arduino, link to the page

[3] Github: desaster/ov7670fifotest, link to the page

[4] Desaster's Blog: V7670-FIFO + MSP430 Launchpad, link to the page

[5] Embedded Programmer: Hacking the OV7670 camera module (SCCB cheat sheet inside) , link to the page

[6] LuaRadio: New to SDR?, link to the page

[7] RTL-SDR, link to the page

[8] Developing an FSK receiver step-by-step, link to the page

[9] Floripasat-I wiki page, link to the page

[10] Ferroelectric RAM, link to the page