AQUI FALTA INTRO

Daniel Adyl Fidalgo Jammoue is an engineering student from Granada, Spain. He finished his Bachelor's Degree in 2020, within the speciality of Communication Systems.

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

BACHELOR'S THESIS

Low Speed Telemetry Link System for CubeSat

Daniel Adyl Fidalgo Jammoue

TELECOMMUNICATION ENGINEERING

UNIVERSITY OF GRANADA

Bachelor's Degree in Telecommunication Technology and Engineering

Low Speed Telemetry Link System for CubeSat

Daniel Adyl Fidalgo Jammoue

2019/2020

Tutor: Andrés María Roldán Aranda

Cover credits **NASA**.

“Low Speed Telemetry Link System for CubeSat”

Bachelor's Degree in Telecommunication Technology and Engineering

Bachelor's Thesis

"Low Speed Telemetry Link System for CubeSat"

ACADEMIC COURSE: 2019/2020

Daniel Adyl Fidalgo Jammoue

Bachelor's Degree in Telecommunication Technology and Engineering

"Low Speed Telemetry Link System for CubeSat"

AUTHOR:

Daniel Adyl Fidalgo Jammoue

SUPERVISED BY:

Andrés María Roldán Aranda

DEPARTMENT:

Electronics and Computer Science

D. Andrés María Roldán Aranda, Profesor del departamento de Electrónica y Tecnología de los Computadores de la Universidad de Granada, como director del Trabajo Fin de Grado de D. Daniel Adyl Fidalgo Jammoue,

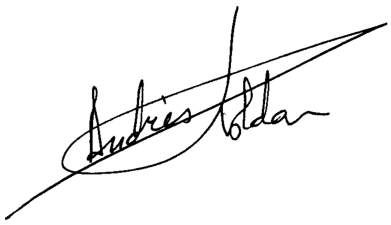
Informa:

Que el presente trabajo, titulado:

"Low Speed Telemetry Link System for CubeSat"

ha sido realizado y redactado por el mencionado alumno bajo mi dirección, y con esta fecha autorizo a su presentación.

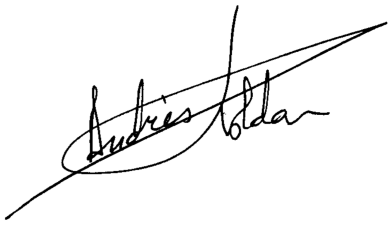
Granada, FECHA



Fdo. Andrés María Roldán Aranda

Los abajo firmantes autorizan a que la presente copia de Trabajo Fin de Grado se ubique en la Biblioteca del Centro y/o departamento para ser libremente consultada por las personas que lo deseen.

Granada, FECHA



Fdo. Daniel Adyl Fidalgo Jammoue

Fdo. Andrés María Roldán Aranda

Low Speed Telemetry Link System for CubeSat

Daniel Adyl Fidalgo Jammoue

KEYWORDS:

CubeSat, Altium Designer® 19, Aerospace design, , , , Electronic, , , EPS, , PCB Design, .

ABSTRACT:

The main purpose of this project is developing a multidisciplinary Simulation Platform for **CubeSats**. It will be composed of three differentiated blocks, around which the project is structured: a mechanical simulation platform, a management software and a prototype that will be the base for the future **GranaSAT-I**.

This Master's Thesis is addressed from a double perspective: on the one hand, the development of a Simulation Platform of great usefulness in an academic environment, as a way to get students from multiple degrees closer to the aerospace world, and particularly to CubeSats, given its current context of peak, being fostered by institutions such as **European Space Agency** (); on the other hand, in a research environment, providing with a mean to implement new communication algorithms, orbit controllers, and generally speaking, for the development and test of new technologies and techniques, before launching.

The development and implementation of this project is performed following methodologies of System Engineering contrasted in the aerospace industry, giving realism and getting the student closer to professional techniques, widely recognized in the job market. Furthermore, the complexity and multidisciplinary scope of this Master's Thesis allows covering not only the different specialties of the Master in **Telecommunication** Engineering but also acquiring knowledge and transversal abilities from other fields of the Engineering, such as **Mechanical** or **Aerospace**. Besides specific software of each of the mentioned areas, advanced techniques of **machining** (aluminum milling), **manufacturing** (solder reflow) or **characterization** of different devices (lithium batteries, silicon solar cells...) among others, have been analyzed and applied.

The result of the exposed culminates with the obtention of a complete and functional simulation environment, which complies with the requirements defined in the preliminary stages, and supposes the finalization of the Master.

Low Speed Telemetry Link System for CubeSat

Daniel Adyl Fidalgo Jammoue

PALABRAS CLAVE:

, Altium Designer® 19, Diseño aeroespacial, , , , Electrónica, , , EPS, , Diseño de PCB, .

RESUMEN:

El objetivo principal del presente proyecto es desarrollar una Plataforma de Simulación multidisciplinar de **CubeSats**. Estará compuesta de tres bloques diferenciados, en torno a los cuales pivotará el proyecto: una plataforma de simulación mecánica, un software de gestión de y un prototipo de , que constituirá la base del futuro **GranaSAT-I**.

Este Trabajo Fin de Máster se aborda desde una ambiciosa doble perspectiva: por un lado, el desarrollo de una Plataforma de Simulación de amplia utilidad en el ámbito académico, como medio para el acercamiento del alumnado de múltiples titulaciones al mundo aeroespacial y en concreto a los CubeSats, en el contexto de auge actual, fomentado por instituciones como la **Agencia Espacial Europea** (); en segundo lugar, en el ámbito de investigación, proveyendo de un medio para la implementación de nuevos algoritmos de comunicación, de control orbital y, en general, para el desarrollo y testeo de tecnologías y técnicas novedosas, de manera previa a su lanzamiento.

El desarrollo e implementación de este proyecto se lleva a cabo siguiendo metodologías de **Ingeniería de Sistemas** contrastadas y asentadas en la industria espacial, dotándolo de realismo y acercando al alumno a técnicas profesionales de amplio reconocimiento en el mercado de trabajo. Asimismo, la complejidad y ámbito multidisciplinar de este Trabajo Fin de Máster le permite cubrir, no sólo las diferentes especialidades del Máster de Ingeniería de **Telecomunicación**, sino también adquirir conocimientos y habilidades transversales o específicos de otros campos de la Ingeniería, como la **Mecánica** o la **Aeroespacial**. Así, además de software especialista de cada uno de los campos mencionados, se han analizado y aplicado técnicas avanzadas de **mecanizado** (fresado de aluminio mediante control numérico), **fabricación** (soldadura utilizando técnicas de *reflow*) o **caracterización** de diferentes dispositivos (baterías de litio, células solares de silicio...), entre otros.

El resultado de todo lo expuesto culmina con la obtención de un entorno de simulación completo y funcional, que cumple con los requisitos definidos en etapas iniciales, y con el cual se cierra la etapa universitaria de Máster.

'Tough and competent'

Acknowledgments:

HERE ACKNOWLEDGMENTS IN ENGLISH

Agradecimientos:

AQUÍ AGRADECIMIENTOS EN ESPAÑOL

Index

Defense authorization vii

Library deposit authorization ix

Abstract xi

Dedication xv

Acknowledgments xvii

Index xix

List of Figures xxi

Code Index xxiii

List of Videos xxv

List of Tables xxvii

Glossary xxix

Acronyms xxxi

1 Introduction 1

2 Requirements **3**

3 Analysis **5**

3.1 Hardware Components in the Transmitter 6

3.1.1 Microcontroller 6

3.1.2 RF Module 8

3.1.3 Camera Module 11

3.1.4 BUS PC-104 13

3.2 Software Defined Radio and GNU Radio in the Receiver 14

3.3 Power and mass characteristics 15

4 System Design **17**

4.1 Transmitter design 17

4.2 Receiver design 17

5 Tests and results **21**

References **25**

A Microcontroller's migration **27**

B Blinking Test **29**

B.1 Code and explanation 29

List of Figures

3.1 Project's Block Diagram 6
3.2 MSP430FR6989 Functional Block Diagram [6] 7
3.3 RF4463F30 Schematic [10] 8
3.4 Si4463 Functional Block Diagram [11] 9
3.5 Camera Block Diagram [9] 12
3.6 FIFO Block Diagram [2] 12
3.7 Camera Module 13
3.8 FIFO 13
3.9 BUS PC-104 Schematic [13] 13
3.10 RTL-SDR & DAB FM DVB-T 14

4.1 Software Defined Radio in reception 17
4.2 FSK Demodulator in GNU Radio 18

5.1 FM broadcasting wrongly received 22
5.2 RTL test 23
5.3 FM broadcasting well received 24

A.1 MSP430F6659 Functional Block Diagram [5] 27

B.1 LEDs system 30
B.2 Blinking Test Flow Diagram 30

Code Index

B.1 Blinking test main code 30
B.2 LEDs initialization 31
B.3 Clock configuration 31
B.4 Turn off LEDs 32
B.5 Turn on LEDs 32
B.6 LEDs definition in PINMAP 32

List of Videos

List of Tables

3.1 Comparisson among transceivers 10
3.2 Comparisson among camera modules 11
3.3 Weight of the components 15
3.4 Power budget 15

A.1 Differences between microcontrollers 28

Glossary

CubeSat Miniaturized satellite made up of multiples of 10 cm × 10 cm × 10 cm cubic units.

FloripaSat-1 Brazilian CubeSat mission.

Linux Open-source and free OS.

microcontroller Compact integrated circuit used to take control over an operation in an embedded system [14].

MSP430FR6989 MCU used in this project.

RF4463F30 RF module used in this project.

Texas Instruments Technology company the microcontroller has been bought from.

transceiver Device able to transmit and receive.

Acronyms

(G)FSK Gaussian Frequency Shift Keying.

(G)MSK Gaussian Minimum Shift Keying.

ADC Analog-to-Digital Converter.

AES Advanced Encryption Standard.

AM Amplitude Modulation.

ASK Amplitude-Shift Keying.

CCS Code Composer Studio.

Comp Comparator.

CRC Cyclic Redundancy Check.

DAC Digital-to-Analog Converter.

DMA Direct Memory Access.

EPS Energy Powering Subsystem.

eUSCI Enhanced Universal Serial Communication Interface.

FIFO First In First Out.

FM Frequency Modulation.

FRAM Ferrite Random Access Memory.

FSK Frequency Shift Keying.

GND Ground.

I/O Input/Output.

I2C Inter-Integrated Circuit.

IF Intermediate Frequency.

IrDA Infrared Data Association.

LCD Liquid Crystal Display.

LDO Low-Dropout Regulator.

LED Light-Emitting Diode.

LNA Low Noise Amplifier.

LO Local Oscillator.

MCU Microcontroller Unit.

MPY Hardware Multiplier.

NASA National Aeronautics and Space Administration.

OOK On-Off Keying.

OS Operating System.

PA Power Amplifier.

PCB Printed Circuit Board.

RAM Random Access Memory.

RDS Radio Data System.

RF Radio Frequency.

RISC Reduced Instruction Set Computing.

RTC Real Time Clock.

RX Receive.

SDR Software Defined Radio.

SMA SubMiniature version A.

SPI Serial Peripheral Interface.

SRAM Static Random Access Memory.

TX Transmit.

UART Universal Asynchronous Receiver-Transmitter.

USB Universal Serial Bus.

USCI Universal Serial Communication Interface.

Chapter 1

Introduction

AQUÍ VA LA INTRO

1

Chapter 2

Requirements

requerimientos

2

Chapter 3

Analysis

In this chapter, every part of the current project will be presented and explained. Each single element is chosen in order to achieve the requirements exposed in Chapter 2. As well, a very relevant topic that cannot be forgotten is the fact that those eventually selected elements have to be within a CubeSat, so its features in terms of energy consumption, supported temperatures, size and weight are taken into account meticulously in the election process.

As this project is an adaptation of the FloripaSat-1's, some of the elements are the same as the used in the Brazilian project, whereas others are replaced by options that have been considered better choices for the applications of the current work.

To visualize the the whole system in an easy way, in the next figure the of the project is shown:

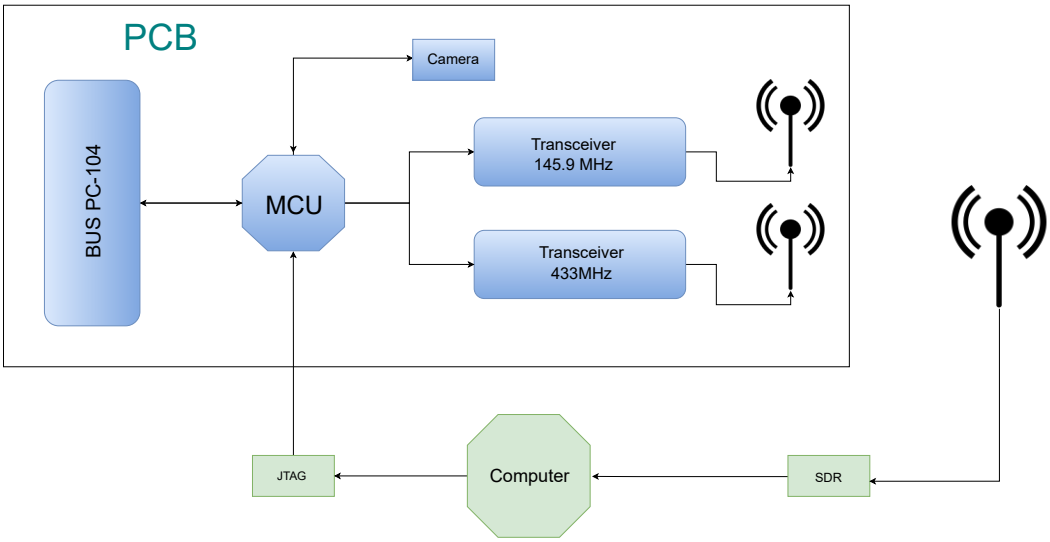


Figure 3.1 – *Project's Block Diagram*

3

3.1 Hardware Components in the Transmitter

3.1.1 Microcontroller

The microcontroller used in this project is Texas Instruments MSP430FR6989 [6], of which is shown in Figure 3.2.

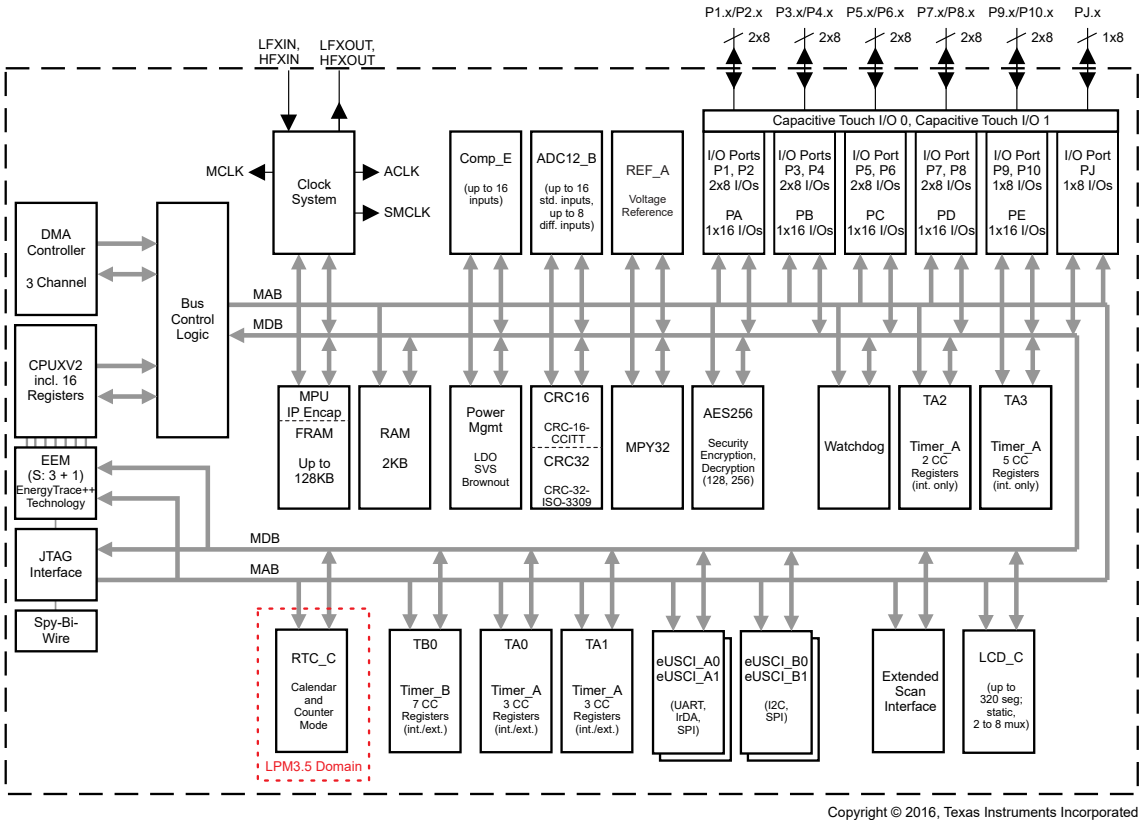


Figure 3.2 – *MSP430FR6989 Functional Block Diagram [6]*

This MCU has an embedded FRAM non-volatile memory up to 128 kB with fast write of 64 kB in 4 ms, 1015 write cycle endurance and an ultra-low-power system architecture, either in active mode or standby and shut-down mode, which make it useful to increase performance at lowered energy budgets.

The MSP430FR6989 has internal and digitally enabled capacitors/pull-ups, 4 eUSCI ports, 3-channel Internal DMA, a voltage range from 1.8 V up to 3.6 V, being 3.3 V the value used in this project, 32-bit MPY, AES Security Encryption and Decryption Coprocessor and 16-Bit RISC architecture up to 16-MHz clock, among other outstanding features that make it fit right in this project.

Those characteristics that have not been named but can be seen in the above are explained in the Appendix A, where this MCU is compared with the microcontroller used in FloripaSat-1. The migration from this MCU to ours is also explained within the appendix.

3.1.2 RF Module

The RF module used in this project is the RF4463F30 [10], of which schematic is shown in Figure 3.3. In this case, it is the same as the one used in FloripaSat-1 because its features are better than those in other options found on the market. Therefore, it is not necessary to migrate from RF modules like in the case of the microcontroller.

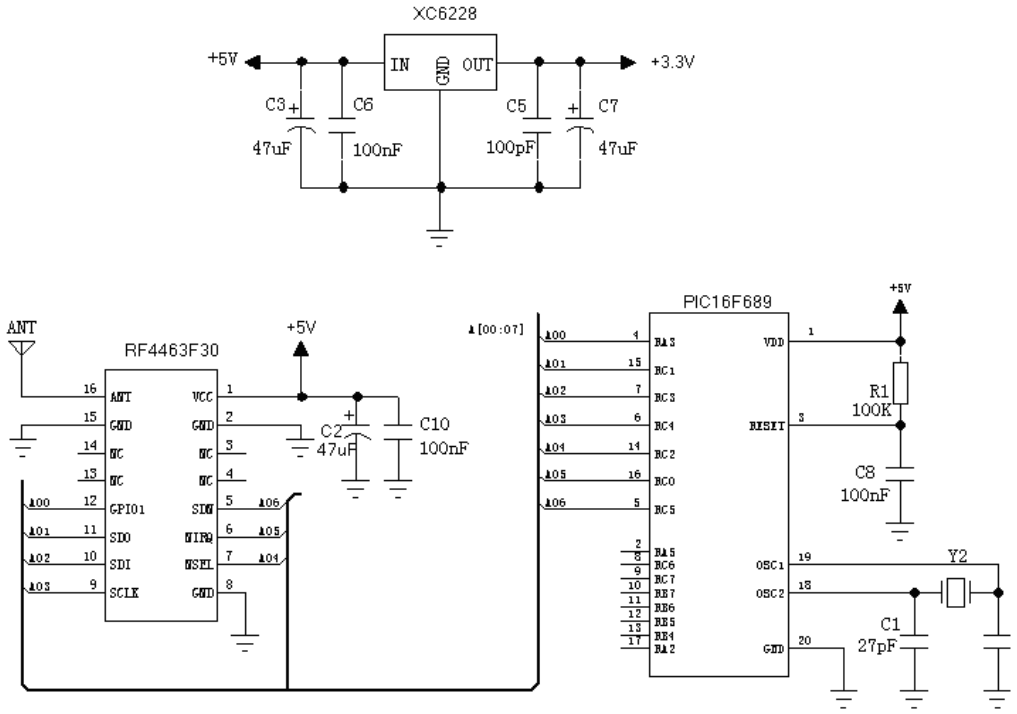


Figure 3.3 – *RF4463F30 Schematic [10]*

The RF4463F30 is a high power wireless transceiver module, with a [11] chip (Figure 3.4), which is an integrated low power transceiver. Furthermore, this RF module also has a PA, so there is no need to use an external PA.

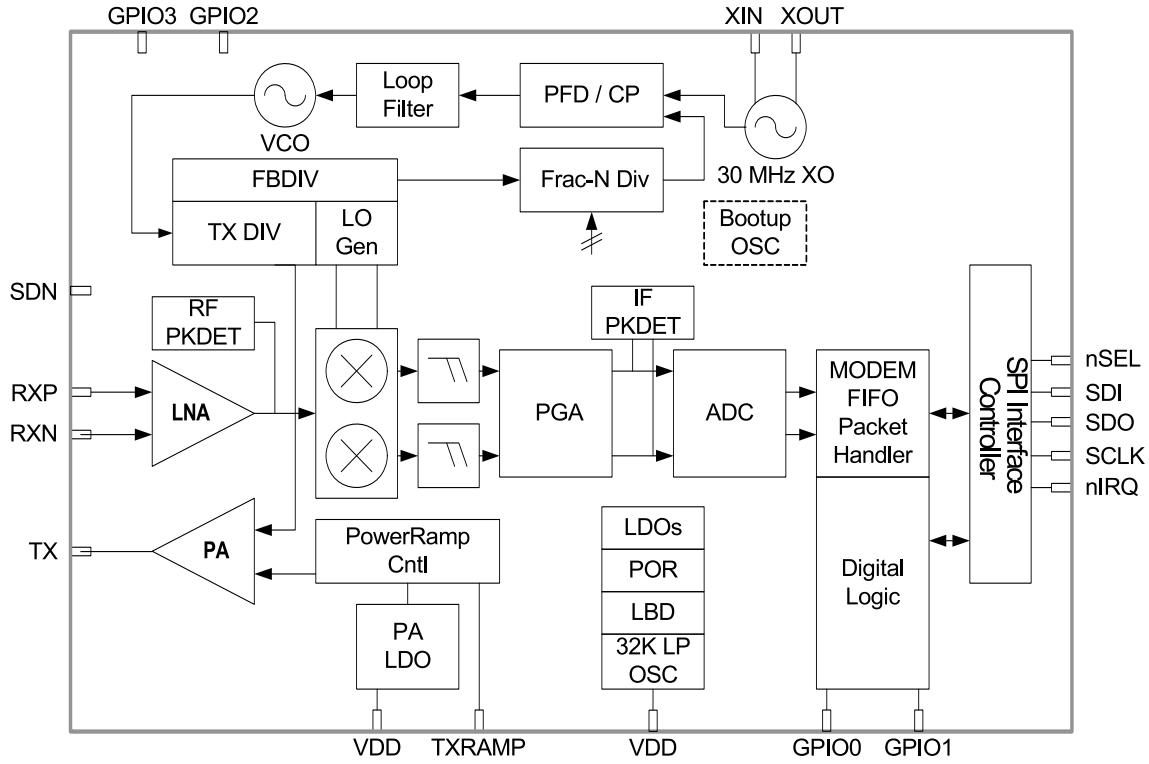


Figure 3.4 – *Si4463 Functional Block Diagram [11]*

Another component that can be done without is a circuit to switch the RX and TX paths, since this module has a built-in antenna switching for both modes. This feature has a huge relevance because it allows the use of a single antenna for transmitting and receiving in each frequency band, instead of using one for the and another for the .

Among its characteristics, some that must be highlighted are:

- Output power of 30 dBm maximum.
- Very high receiving sensitivity (-126 dBm).
- Power supply range from 1.8 V to 3.6 V, even though it has a LDO to have a constant source of 3.3 V for the transceiver.
- High data transfer rate from 0.1 kbps up to 1 Mbps.
- Preamble detection in RX mode.
- Customizable frequency range from 142 MHz up to 1050 MHz.
- Ultra low consumption shut-down mode.

- 64/128-byte RX and TX FIFO data register.
- Different possible modulations such as (G)FSK, 4(G)FSK, (G)MSK, ASK and OOK.

All these features explained make this transceiver fit better in the current project than other candidates, which were the RF module ADF7021-N [1] and the AX5043[3].

Followingly, some of the main characteristics of these three transceivers are compared:

Feature	RF430F30	AX5043	ADF7021-N
Output power (dBm)	30	16	13
Receiving sensitivity (dBm)	-126	-126	-122
Power supply range (V)	1.8 - 3.6	1.8 - 3.6	2.3 - 3.6
Data transfer rate (kbps)	0.1 - 1000	0.1 - 125	0.5 - 24
Modulations	(G)FSK/ 4(G)FSK/ (G)MSK OOK ASK	FSK/MSK/ 4FSK/ (G)FSK/ (G)MSK/ ASK/ AFSK/ FM/PSK	2FSK/ 3FSK/ 4FSK/MSK
TX mode current (mA)	550	51.6	23
RX mode current (mA)	10	9.5	18.3
Price (€)	7.74	2.17	1.87

Table 3.1 – *Comparisson among transceivers*

It is easy to see in this table that the other transceivers are much cheaper and have some advantages over the one used here. However, the chosen one is the RF4463F30 due to its outstanding parity between a high output power and a very good receiving sensitivity.

This, combined with everything explained previously, make this RF module a very good choice.

3.1.3 Camera Module

The camera used in this project is the with AL422B FIFO [8]. During the choosing process, other modules were taken into account. Beyond size specifications, image quality and power consumption related features have been highly relevant. The other possible modules were the OV7670 without FIFO [9] and the Adafruit TTL 397 [12].

Feature	**OV7670 with AL422B FIFO**	**OV7670**	**Adafruit TTL 397**
Maximum Power Supply (V)	3.3	3.3	5
Temperature range (ºC)	-30 to 70	-30 to 70	No information
Output formats	YUV/YCbCr 4:2:2 RGB565/555/444 GRB 4:2:2 Raw RGB Data	YUV/YCbCr 4:2:2 RGB565/555/444 GRB 4:2:2 Raw RGB Data	Standard JPEG / M-JPEG
Maximum Transfer Rate (fps)	30	30	30
Sensitivity (V/Lux ·*sec*)	1.3	1.3	No information
S/N Ratio (dB)	46	46	45
Pixel size (μm)	3.6 x 3.6	3.6 x 3.6	5.6 x 5.6
Power Requirements	Active = 60 mW Standby < 20 μA	Active = 60 mW Standby < 20 μA	75 mA
Dynamic Range (dB)	52	52	60
Price (€)	11.01	5.99	35.92

Table 3.2 – *Comparisson among camera modules*

As seen in Table 3.2, first two modules are very similar. In fact, these camera modules themselves are exactly the same. However, one of them has a FIFO queue. This characteristic is a huge difference in terms of design because, thanks to that, an external clock is not needed to sync up the camera with the MCU. If the module without FIFO was used, a slight desynchronization between the microcontroller and the camera would mean the image loss.

Therefore, once the without FIFO has been discarded, the next step is comparing the module with FIFO with the Adafruit one. Here, the election seems quite easy just by seeing the difference between prices and between power requirements.

Eventually, and as said at the beginning of this subsection, the camera module chosen is the with FIFO.

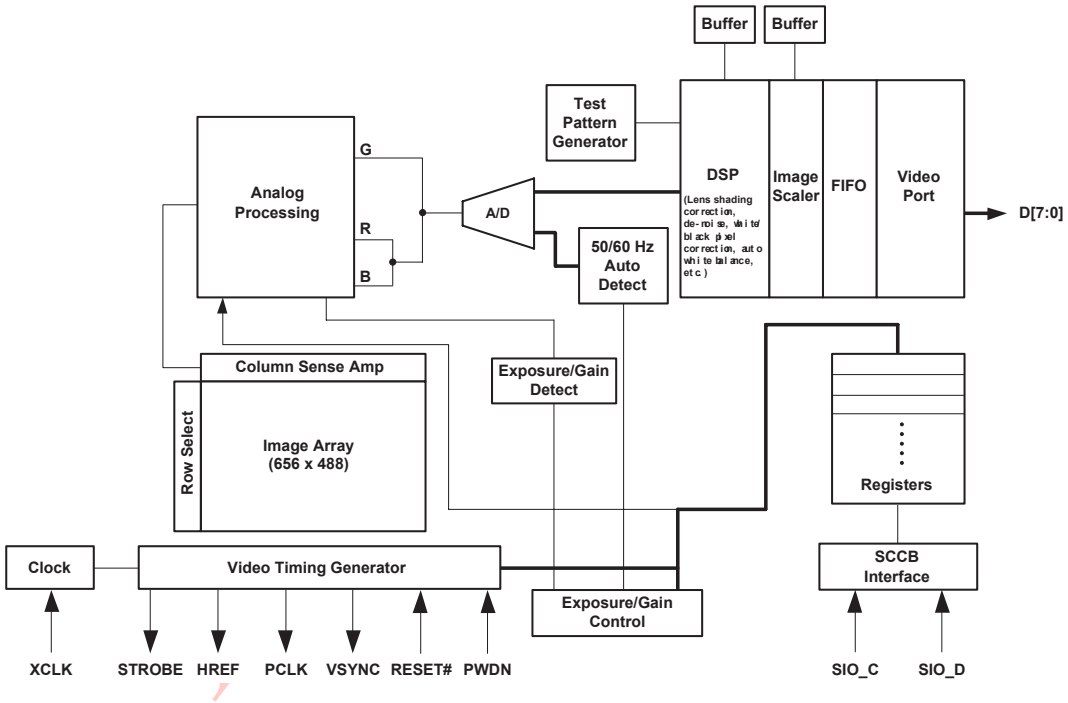


Figure 3.5 – *Camera Block Diagram [9]*

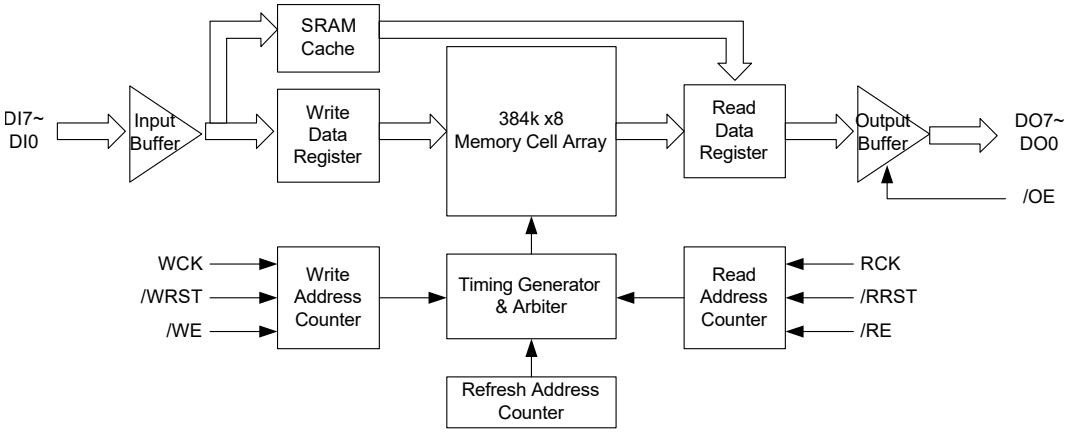


Figure 3.6 – *FIFO Block Diagram [2]*

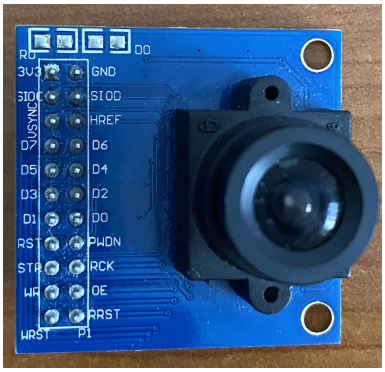


Figure 3.7 – *Camera Module*

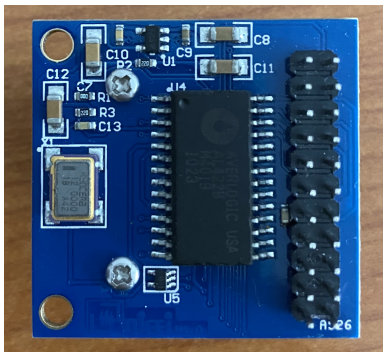


Figure 3.8 – *FIFO*

3.1.4 BUS PC-104

As the CubeSat that will hold the PCB of this project has multiple boards, it is needed to add a bus of 104 signals to exchange information among them.

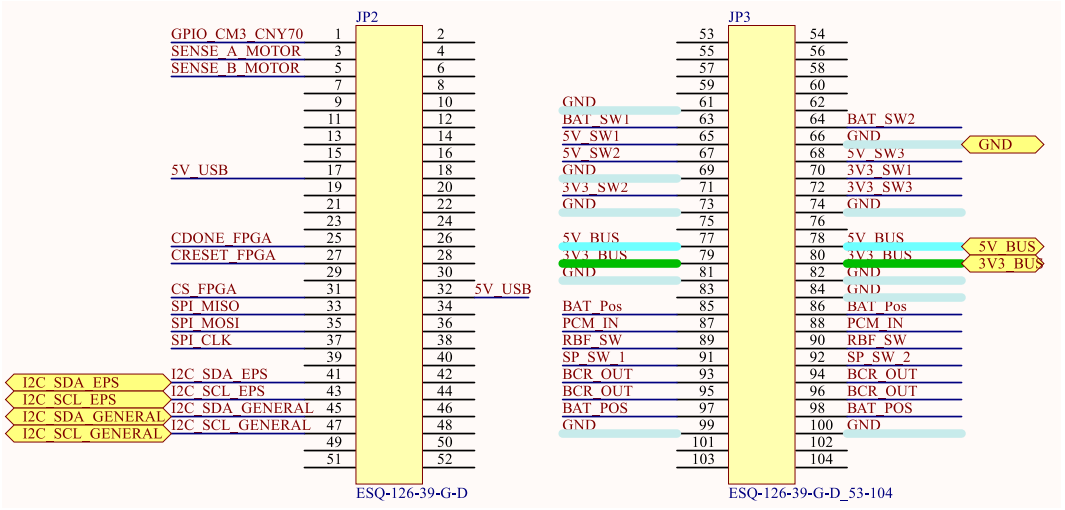


Figure 3.9 – *BUS PC-104 Schematic [13]*

Most of the signals seen in Figure 3.9 are from the EPS, but only some of them are actually used in the current project, which are those highlighted [13]:

- 5V bus, in s 77 and 78
- 3V bus, in s 79 and 80
- GND, in s 61, 66, 69, 73, 74, 81, 82, 84, 99 and 100
- I2C EPS signals, in s 41 and 43

- General I2C signals, in s 45 and 47

3.2 Software Defined Radio and GNU Radio in the Receiver

To simulate a , a very good idea is using an SDR connected to an antenna (Figure 3.10) and both connected to a computer with a built schematic.

Figure 3.10 – *RTL-SDR & DAB FM DVB-T*

The antenna is used for the RX function and the SDR device for the operation seen in Figure 4.1, where the analog received signals are transformed into digital using an ADC. These elements bring this project enough power sensitivity and receiving frequency range to perform as a good in the distance that the tests will be done.

Obviously, these two elements themselves are not enough to demodulate the received signals, which is the main part of the receiving task. If the information is not demodulated, the transmission would be senseless. Hence, as said before, the SDR + antenna must be connected to a computer with a signal processing capable software. In this case, the chosen one is due to some interesting reasons, being the main one that it is programmed in . The clear advantage of this feature is that it is a widely spread language, which is translated in a very large community with a good amount of documentation to solve different problems that will be dealt with along the project process.

Another ease given by is that it can be linked without any problem to a coded project, which makes it very useful in the task of presenting the received packets to the user.

3.3 Power and mass characteristics

As said in Chapter 2, this PCB will be part of a CubeSat. This means that there are some power, size and mass limitations that must be respected to make this project possible.

The first matter to care about is the total allowed weight. An important reason to choose a component over another is its weight because if it is too heavy, the previous requirements would not be achieved. The weight of each component is shown in the following table:

Component	Weight (g)
PCB	22
Microcontroller	0.6
RF Module	4
PC-104	2.7
Camera	12.9
Camera Connector	2
SMA Connector	1.5

Table 3.3 – *Weight of the components*

Taking into account that there are two RF modules and two SMA connectors, the total weight is 51.2 g, which meets with the maximum allowed mass requirement of 1.33 kg.

In terms of size, only with a right PCB the limitations are covered. Therefore, this is also good for the project.

The last task is related to power consumption. Here, the most relevant components are the camera module, the MSP430FR6989 and the RF modules. Saving energy is one of the most important matters. Hence, a very low power consumption in sleep mode and a not so high power consumption in active mode are searched while choosing the components. In Table 3.4 these consumptions are shown:

Component	Sleep Mode (μW)	Active Mode (mW)
Camera	66	165
Microcontroller	10.56	2.64
RF Module	15	2750

Table 3.4 – *Power budget*

As seen, the components chosen have a good energy consumption either in sleep

mode and active mode. Therefore, this requirement is correctly respected.

Chapter 4

System Design

After analyzing the different parts of both, the transmitter and the receiver, it is important to explain how they are designed. To begin with, the transmiiter has some systems and procedures that must be understood in order to design a well simulated ground station that does the inverse processes. Obviously, if both parts are not equivalent, the whole system will not performed in the desired way.

4.1 Transmitter design

4.2 Receiver design

As explained in Section 3.2, the receiver will be simulated using a SDR connected to . After receiving the signals, the antenna's task is to transform the electromagnetic wave into a signal that the SDR device is able to understand. In this project, the Software Defined Radio will only be used as a receiver. Therefore, its schematic is shown in the next figure:

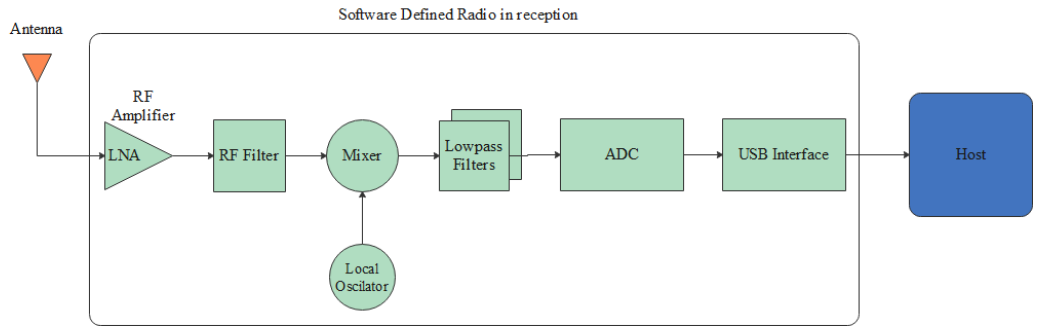


Figure 4.1 – *Software Defined Radio in reception*

Before explaining all these parts, the system must be calibrated. To do so, a software called *rtl_test* has been used. This is a very good and useful program because with this, the frequency correction needed by the SDR is easily found. This process will be explained in the next chapter.

Just after the signal is received by the antenna, it is amplified using a LNA and followingly it is filtered. The next step during the signal processing is moving the input signal to a desired frequency and, to do so, a LO with a mixer is used. After it, it is needed to send the signal to a Low Pass Filter in order to supress the harmonics. Finally, this signal is converted to digital with an ADC and then sent to the host where the schematic is implemented through a USB interface. It is important to mention that the sample rate during the conversion must not be too high (lower than 2.8 MS/s) to avoid problems [13].

Once the digital signal has reached the host, it is time for to demodulate it. After this, the signal is sent to a program in able to find the preambles and synchronization word of each packet and to show the user all the values sent by the transmitter. Firstly, in Figure 4.2, all the parts of the demodulator can be seen:

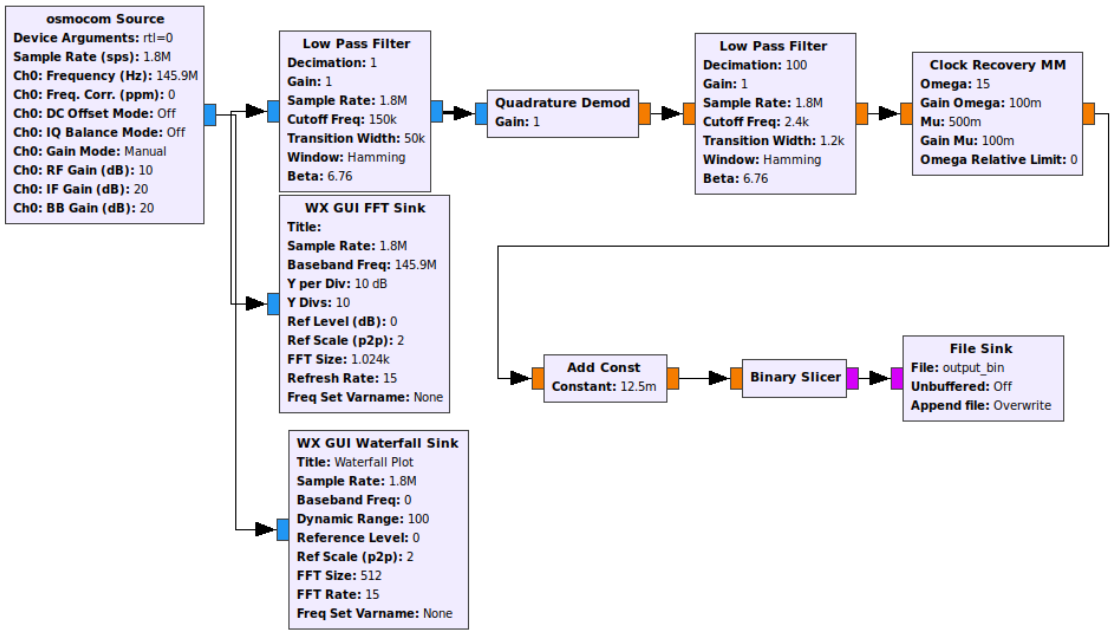


Figure 4.2 – *FSK Demodulator in GNU Radio*

The first block of this demodulator *osmocom source*, which used to locate different devices on the host's system. In this case, it is used to link all the demodulation station to the antenna + SDR system. Among the parameters that this block uses are the sample rate, the frequency of the channel and several gains. The output type of this block is a

complex float32 signal [4] that goes to a Low Pass Filter, an FFT Sink and a sink.

These two sinks are only relevant to visualize the reception of packets during the transmission. However, the Low Pass Filter is important to suppress the out-of-band signals in order to improve the performance of this demodulator. After being filtered, the signal goes through a quadrature demodulation block with gain 1 that can be used to demodulate signals in FM, FSK, (G)MSK, etc. Mathematically, this block the one-sample delayed input and the conjugate undelayed signal. Followingly, it calculates the argument of the resulting complex number. In mathematical terms:

$$y[n] = arg(x[n]\bar{x}[n-1]) \tag{4.2.1}$$

It is relevant to understand that, basically, when the input signal is a sinusoidal, what this block does is:

$$y[n] = \frac{f}{f_s} \tag{4.2.2}$$

being f the signal's frequency and f_s the sampling frequency.

The next block is another Low Pass Filter used with the same purpose as the previous. Its output goes to a *Mueller & Muller clock recovery* block. As its name says, this block is used to recover the clock of the input signal. However, this block brings many problems due to its input parameters. As can be seen in Figure 4.2, this block's parameters are: *omega* (symbol period in samples per symbol), *gain omega*, *mu* (detected phase shift in samples between the receiver and transmitter), *gain mu* and *omega relative limit* (limits within which the omega can change during run time, if for example omega is 10, omega relative limit can be 0.1, but this value depends on the signal) [7]. The main issue with this block is that the only clear parameter is omega, which can be calculated as sample rate devided by baud rate. The rest of the parameters are found empirically until the block has a correct performance.

Due to all the problems that the clock recovery block meant to the project, it was needed to insert an *Add Constant* block. What this block does is to delay the biginning of the clock's high pulse in order to record correctly the value of the signal. Finally, the signal goes through a binary slicer to recover the bytes sent within the signal, which are saved in a binary file using the *File Sink*.

4

Chapter 5

Tests and results

Before starting with the tests, it has a huge relevance to make sure that the antenna + SDR system is well calibrated. For this task, the software called SDR Sharp (SDR#) is very useful because it allows to receive different types of signals such as AM or FM broadcasting and record them.

To calibrate an antenna means to find out the gain and frequency correction needed to receive signals without errors. The gain value is important because if it is too low, no signal would be received. However, if its value is too high, it would be possible to receive signals but the noise would be increased as well. On the other hand, the frequency correction helps to receive the signals in the middle of the channel. Even though an antenna is set to receive at a specific frequency, if the frequency correction is not well chosen, the received signals would be shifted, which means the loss of information.

After understanding this, a good option is to receive FM broadcasting to calibrate the SDR in an easier way because these signals have known channels. When the software is run and a channel is selected, the received signal looks like in Figure 5.1. Here, the is too high and the channel is barely recognised. The reason is that neither the gain nor the frequency correction have been set.

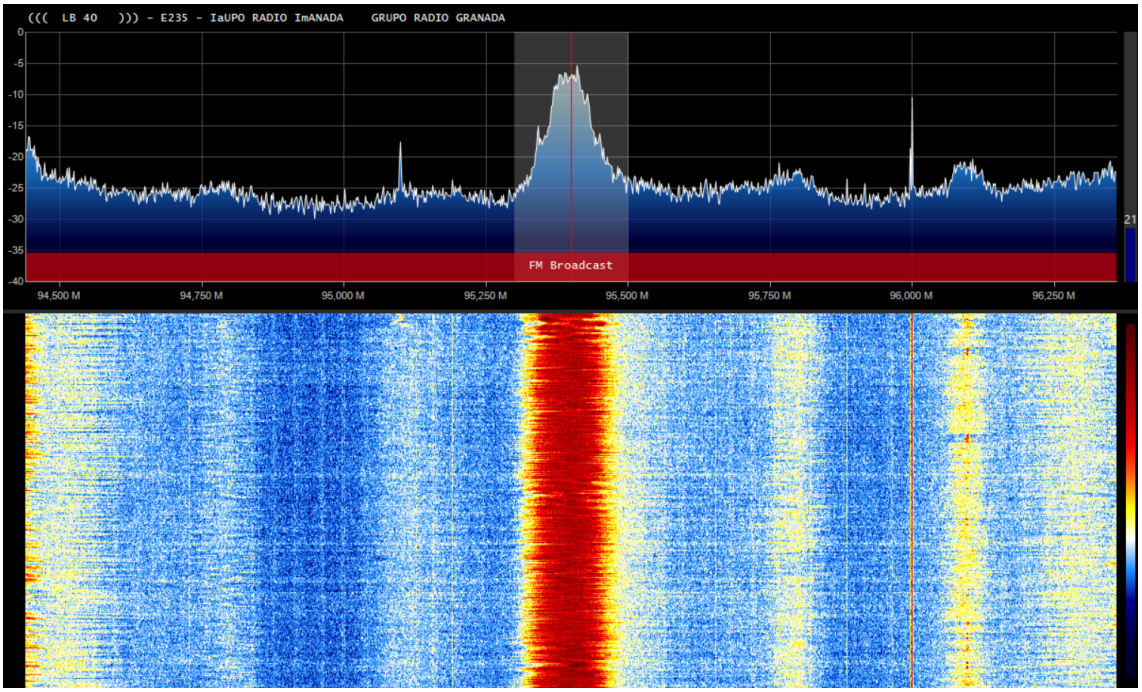


Figure 5.1 – *FM broadcasting wrongly received*

To find out the frequency correction, a software called *rtl_test* [15] has been used. This software is very easy because once it is downloaded in the host connected to the SDR, only with the command *rtl_test -p* in the console (for this task the OS used is Linux) it begins to receive information during a few minutes in order to know which is the frequency correction in parts-per-million of the antenna (see Figure 5.2).

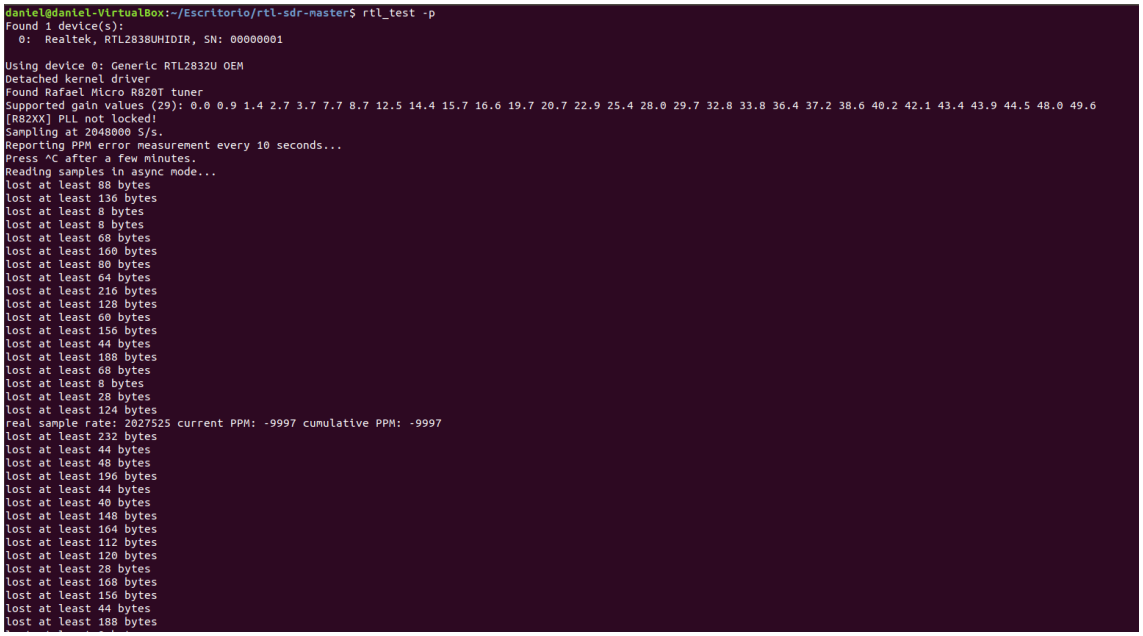


Figure 5.2 – *RTL test*

The frequency correction of the SDR + antenna system is 33. Now, with this value it is possible to receive the FM broadcasting signals in a proper way. In addition, it is needed to enable an IF noise reduction with a threshold of -120 dB, appart from establishing a 15.7 dB gain (value that allows a good reception of the signal with a low level of noise) and the RTL AGC mode (RF auto gain). With all these paremeters set, the received signal is as seen in Figure 5.3:

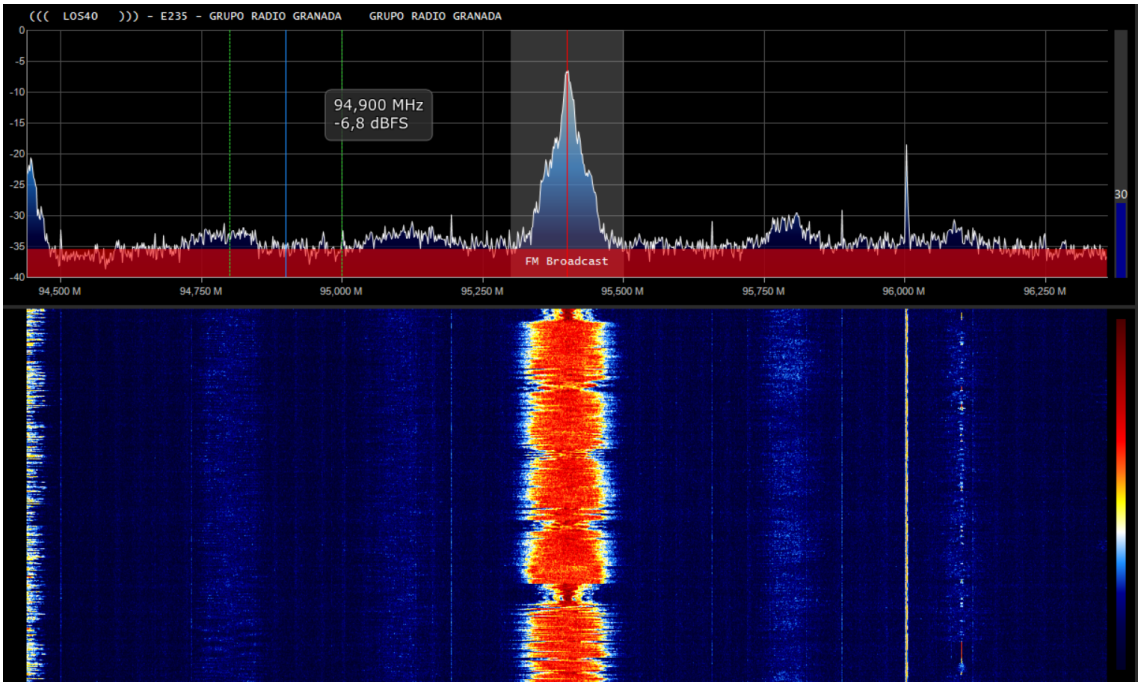


Figure 5.3 – *FM broadcasting well received*

Now that the antenna is calibrated, the received FM broadcasting signal looks better, with a lower . Another proof of its well calibration is that the SDR# is able to decode the RDS, which gives information about the channel like its name.

5

References

[1] ADF7021-N: High Performance Narrow-Band Transceiver IC. https://www.analog.com/en/products/adf7021-n.html.

[2] AL422 Data Sheets (Revision V1.1). http://www.openimpulse.com/blog/wp-content/uploads/wpsc/downloadables/AL422-FIFO-Datasheet.pdf.

[3] AX5043: Ultra-Low Power Narrow-Band Sub GHz (27 - 1050 MHz) RF Transceiver. https://www.onsemi.com/pub/Collateral/AX5043-D.PDF.

[4] Description of some GNU Radio blocks. http://blog.sdr.hu/grblocks/types.html.

[5] MSP430F665x, MSP430F645x, MSP430F565x, MSP430F535x Mixed-Signal Microcontrollers. https://www.ti.com/lit/ds/symlink/msp430f6659.pdf?ts=1590693622195.

[6] MSP430FR698x(1), MSP430FR598x(1) Mixed-Signal Microcontrollers. https://www.ti.com/lit/ds/symlink/msp430fr6989.pdf?ts=1590598345240.

[7] Notes on M&M Clock Recovery. https://www.tablix.org/~avian/blog/archives/2015/03/notes_on_m_m_clock_recovery/.

[8] OV7670 Camera Module with AL422 FIFO. https://www.openimpulse.com/blog/products-page/product-category/ov7670-camera-module-with-al422-fifo/.

[9] OV7670/OV7171 CMOS VGA (640x480) Camera Chip Sensor with OmniPixel Technology. http://web.mit.edu/6.111/www/f2016/tools/OV7670_2006.pdf.

[10] RF4463f30 1W High Sensitivity Wireless Transceiver Module. https://www.nicerf.com/Upload/ueditor/files/2018-12-15/RF4463F30%201W%20High%20Power%20Wireless%20Transceiver%20Module%20V2.2-4a0e93b3-6bff-4d22-9efa-abbee5ac3de8.pdf.

[11] Si4464/63/61/60 High-Performance, Low-Current Transceiver. https://www.silabs.com/documents/public/data-sheets/Si4464-63-61-60.pdf.

[12] TTL Serial Camera. https://cdn-learn.adafruit.com/downloads/pdf/ttl-serial-camera.pdf.

[13] Galicia, J. A. M. Telemtry, Tracking & Command board for CubeSat. Master's thesis, École Polytechnique Fédérale de Lausanne, Laussane, August 2019.

[14] Rouse, M. Microcontroller. https://internetofthingsagenda.techtarget.com/definition/microcontroller.

[15] Salmond, R. Automatically calibrate PPM for RTL_SDR. https://rob.salmond.ca/automatically-calibrate-ppm-for-rtl-sdr/.

Appendix A

Microcontroller's migration

As explained in Subsection 3.1.1, the microcontroller used in this project is MSP430FR6989, whereas the one used in FloripaSat-1 project is . Hence, at the beginning it was necessary to migrate from one MCU to the other.

Firstly, it is important to understand which are the main differences between these two modules. Therefore, a good strategy is comparing the functional of both. The of our MCU is shown in Figure 3.2, while the FloripaSat-1's is here below:

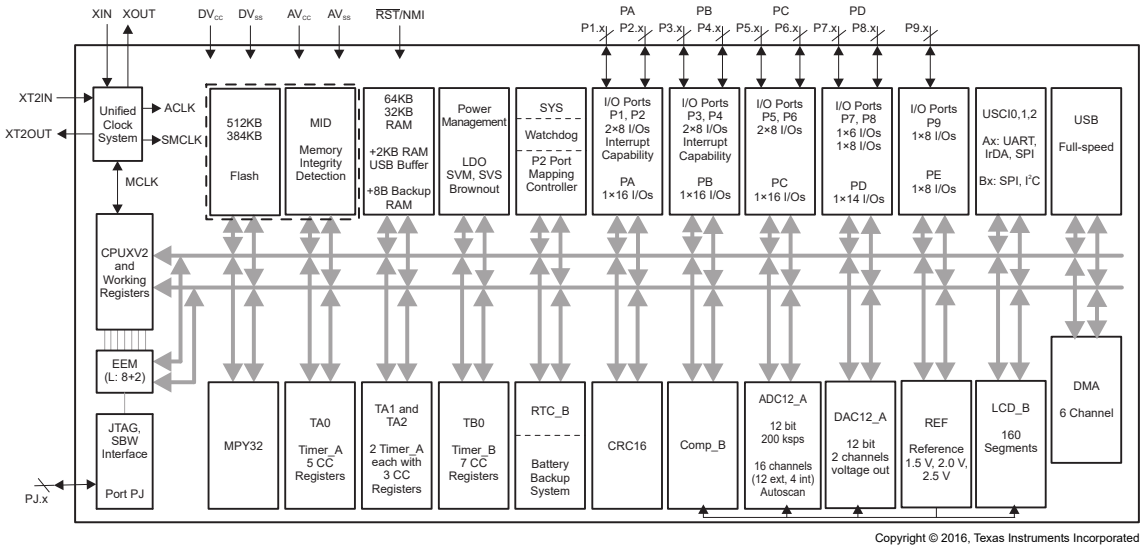


Figure A.1 – *MSP430F6659 Functional Block Diagram [5]*

To make this comparison easier, it is a good idea to show the main disparities in a table:

Feature	MSP430FR6989	MSP430F6659
Power consumption ($\mu A/MHz$)	Active Mode: 100	Active Mode: 295
	Standby Mode: 0.4	Standby Mode: 2.2
	Shut-Down Mode: 0.02	Shut-Down Mode: 0.45
CRC	16-Bit and 32-Bit	16-Bit
ADC	12-Bit	12-Bit
Full Speed USB	No	Yes
RAM	FRAM 128 kB	SRAM 64 kB / 32 kB
	No	Yes
USCI	4 eUSCI ports A0 and A1 for UART, IrDA and SPI B0 and B1 for SPI and I2C	6 USCI ports A0, A1 and A2 for UART, IrDA and SPI B0, B1 and B2 for SPI and I2C
I/O s	10	9
DAC	No	12-Bit / 2 Channels
DMA	3 Channels	6 Channels
16-Bit Timer	5	4
LCD	320 segments	160 segments
Clock frequency	16-Bit up to 16 MHz	16-Bit up to 20 MHz
AES	Yes	No
Comp	E	B
RTC	C	B
Internal and digitally enabled capacitors / pull-ups	Yes	No
Extended Scan Interface	Yes	No
Battery Backup System	No	Yes
Price (€)	5.89	9.85

Table A.1 – *Differences between microcontrollers*

Once all these differences have been seen and understood, it is needed to change all the code related to them. The microcontroller is part of the transmitter. Therefore, all its code is in implemented in using CCS.

Appendix B

Blinking Test

B.1 Code and explanation

One of the first tests done is the **Blinking Test**, focused on making a LED in order to take control over the PCB. To achieve a good performance here, parameters like frequency are relevant to control the LED's timing of . After the code, an is used to check the correct blinking frequency.

Since the PCB used has five LEDs, the first thing to do is deciding which one will perform this testing task. As seen in Figure B.1, two LEDs follow the I2C , one follows the SPI and another is connected to the power supply so it remains switched on all the time. Hence, the chosen LED is the one flagged as *SYSTEM_ON*, which is connected to the 95 of the MCU.

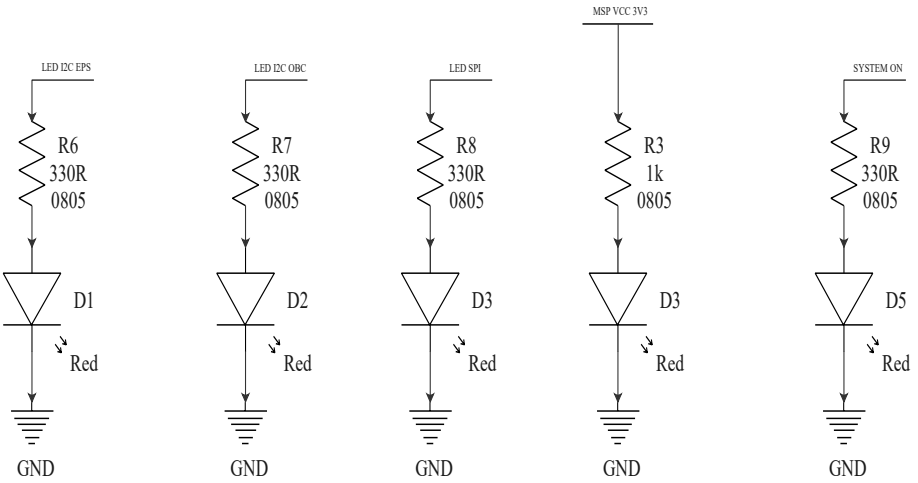


Figure B.1 – *LEDs system*

With the testing LED chosen, the next task to do is to make it . Therefore, a simple script is written and presented in Listing B.1. To make the comprehension of this code easier, in Figure B.2 the is shown:

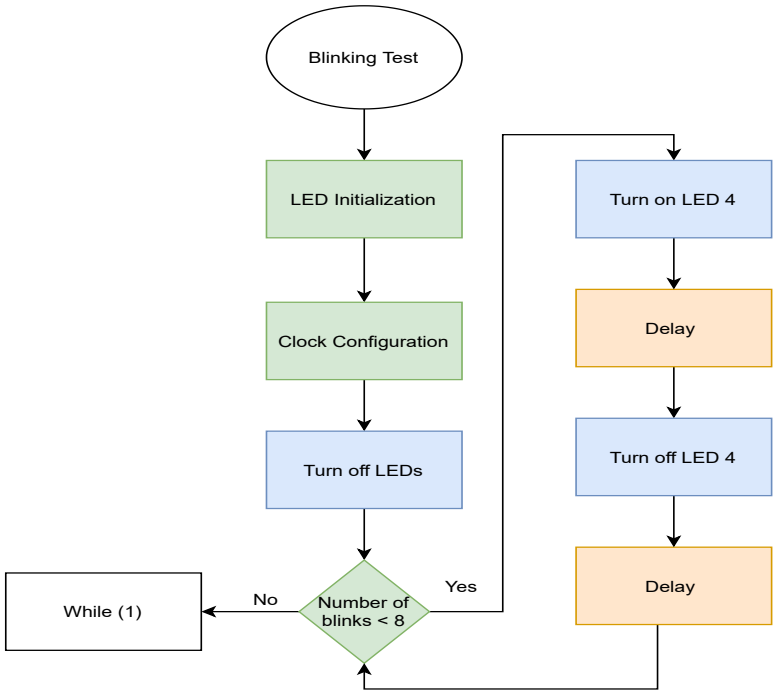


Figure B.2 – *Blinking Test Flow Diagram*

```
int main(void){
    WDTCTL = WDTPW | WDTHOLD; ///<  Stop watchdog timer

    int blink;  ///< Variable that will increase its value each time the main loop is run

    init_LEDs();  ///< Function that initializes the LEDs.
    configure_clock(8);  ///< Function to configure the clock at 8 MHz.

    turnOffLED(LED1);    ///< Function that turns off the LED1
    turnOffLED(LED2);    ///< Function that turns off the LED2
    turnOffLED(LED3);    ///< Function that turns off the LED3
    turnOffLED(LED4);    ///< Function that turns off the LED4


   for(blink=1;blink<8;blink++){
        turnOnLED(LED4);  ///< Function that turns on the LED4
        _delay_cycles(8000000);  ///< LED on during 800000 cycles. This value is used due to the clock is at 8 MHz
        turnOffLED(LED4);
        _delay_cycles(8000000);
    }

   while(1);
}
```

Code B.1 – *Blinking test main code*

After stopping the and initializing the LEDs of the PCB, it is necessary to set the microcontroller's clock frequency. In this test, the chosen value is 8 MHz, but it can be another up to 16 MHz, as seen in Subsection 3.1.1.

The following action is to turn off (see Listing B.4) every LED but the one connected to the power supply just to make the recognition of the blinking LED easier.

Finally, there is a *for* to turn off and on the LED4 seven times, which are enough iterations for this test. In this there are two s to keep the current state during 8000000 cycles. This value allows to see the blinking during about one second, according to the clock of the microcontroller.

In this code there are some relevant functions that will be shown in the following Listings.

```
void init_LEDs(){
    GPIO_setAsOutputPin(LED1);
    GPIO_setAsOutputPin(LED2);
    GPIO_setAsOutputPin(LED3);
    GPIO_setAsOutputPin(LED4);
    PMM_unlockLPM5();
}}
```

Code B.2 – *LEDs initialization*

```
void configure_clock(int8_t freq){
    if(freq == 1)
        CS_setDCOFreq(CS_DCORSEL_0,CS_DCOFSEL_0);
    else if (freq == 4)
        CS_setDCOFreq(CS_DCORSEL_0,CS_DCOFSEL_3);
    else if (freq == 8)
        CS_setDCOFreq(CS_DCORSEL_0,CS_DCOFSEL_6);
    else if (freq == 16)
        CS_setDCOFreq(CS_DCORSEL_1,CS_DCOFSEL_4);

    //Set ACLK = VLO with frequency divider of 1
```

```
12      CS_initClockSignal(CS_ACLK,CS_VLOCLK_SELECT,CS_CLOCK_DIVIDER_1);
13
14      //Set SMCLK = DCO with frequency divider of 1
15      CS_initClockSignal(CS_SMCLK,CS_DCOCLK_SELECT,CS_CLOCK_DIVIDER_1);
16
17      //Set MCLK = DCO with frequency divider of 1
18      CS_initClockSignal(CS_MCLK,CS_DCOCLK_SELECT,CS_CLOCK_DIVIDER_1);
19  }
```

Code B.3 – *Clock configuration*

```
1  void turnOffLED(uint8_t port, uint8_t pin){
2      GPIO_setOutputLowOnPin(port, pin);
3  }
```

Code B.4 – *Turn off LEDs*

```
1  void turnOnLED(uint8_t port, uint8_t pin){
2      GPIO_setOutputHighOnPin(port, pin);
3  }
```

Code B.5 – *Turn on LEDs*

All these functions need the s and s of the LEDs in the PCB. Therefore, it is necessary to define them in the . To find out about the s and s that belong to each LED, see code Listing B.6.

```
1  #define LED1 GPIO_PORT_P6,GPIO_PIN1   ///< Port 6,  Pin 1
2  #define LED2 GPIO_PORT_P6,GPIO_PIN5   ///< Port 6,  Pin 5
3  #define LED3 GPIO_PORT_P2,GPIO_PIN6   ///< Port 2,  Pin 6
4  #define LED4 GPIO_PORT_P10,GPIO_PIN0  ///< Port 10, Pin 0
```

Code B.6 – *LEDs definition in PINMAP*

B.2 Results

Once the code is understood, the following step is to see if the results agree with the expected performance. To do so, watching the 's output might be a good choice:

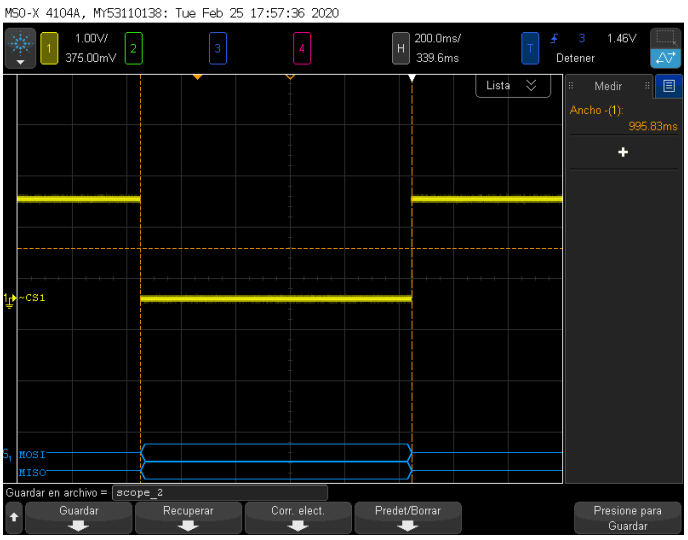


Figure B.3 – *Pulse High*

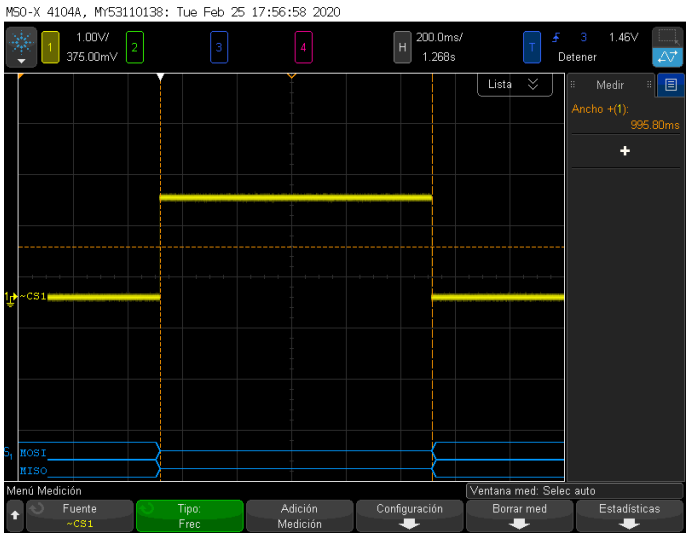


Figure B.4 – *Pulse Low*

Watching Figures B.3 and B.4, on the one hand the have a width of 995.80 ms. On the other hand, the have a width of 995.83 ms. This results are the expected because the clock frequency and the s of in each state have the same value.

Appendix C

Dummy Packets Sending Test

C.1 Code and explanation

After having taken control over the LEDs, the following step is sending s successfully. The wanted performance is to a LED when sending a . Furthermore, another LED will switch on once all the s had been sent.

The script used to do this is shown in Listing C.1, being its in Figure C.1, presented below the code.

```
1  int transmission(void){
2      WDTCTL = WDTPW | WDTHOLD; ///< Stop watchdog timer
3
4      init_LEDs();         ///< Function that initializes the LEDs.
5      configure_clock(8);  ///< Function to configure the clock at 8 MHz.
6
7      turnOffLED(LED1);    ///< Function that turns off the LED1
8      turnOffLED(LED2);    ///< Function that turns off the LED2
9      turnOffLED(LED3);    ///< Function that turns off the LED3
10     turnOffLED(LED4);    ///< Function that turns off the LED4
11
12     spiInit();          ///< Function that initializes the SPI protocol
13
14     rf4463_init('B');       ///< Function that initializes the RF4463 module
15
16     rf4463_enter_standby_mode('B');     ///< Function for entering the standby mode of the RF4463 module.
17
18     // Initialization of the buffer with dummy packets
19     uint8_t* tx_buf[]={0x11, 0x22, 0x33, 0x44, 0x55, 0x66};
20     uint8_t tx_buf_len = sizeof(tx_buf);    ///< Buffer length
21
22     int pointer=0;   ///< Variable that will be increased until the end of the buffer
23
24     while(pointer<6){
25         rf4463_tx_packet('B', tx_buf[pointer], tx_buf_len); ///< Packet transmission.
26
27         turnOnLED(LED4);      ///< Function that turns on the LED4
28         _delay_cycles(4000000);  ///< Delay used to keep the LED on for a while to notice about a transmitted
             packet
29         turnOffLED(LED4);
30         pointer++;
31
32     }
33
34     turnOnLED(LED1);   ///<Switch on a different LED to inform about the end of the transmission
35
```

```
36      while(1);
37
38  }
```

Code C.1 – *Transmission of dummy packets*

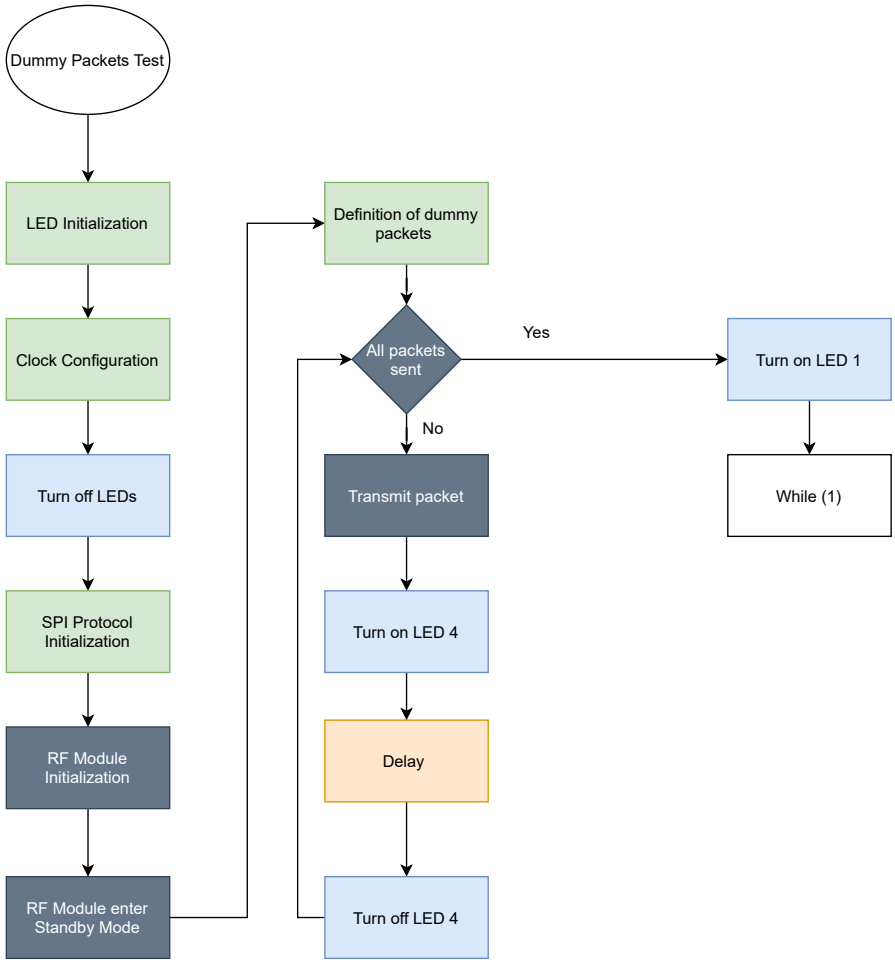


Figure C.1 – *Dummy packets Test Flow Diagram*

Beginning just like the *Blinking Test* script explained in Appendix B, after initializing the LEDs and the microcontroller's clock, it is needed to initialize the RF module. Firstly, using the spiInit() function (see Listing C.2), the SPI is set in the microcontroller. The next steps are to initialize the RF module and to enter in the standby mode to save energy. For these actions, functions rf4463_init() and rf4463_enter_standby_mode() have been used, respectively. Both have as input the slave that the s are being sent to. This input can be "B" (Module) or "T" (Module). In this case, the selected slave is the Module.

Next, s that will be transmitted are defined. Hence, a vector with s is created. Since this is a simple test, six s are enough to check if the transmission is being performed. The following line just establishes the length of the , parameter that will be useful after.

For the transmission of the s, the function needed is rf4463_tx_packet, shown in Listing C.5. Using a while condition, the s will be released until the end of the . After every sent , LED4 will to notice the user. Through the same technique used in the *blinking test*, keeping s between turnOnLED and turnOffLED functions, this performance can be achieved. However, this time shorter s are set up to make this test quicker. Finally, when every has been sent, LED1 turns on.

All these functions that have been used can be seen followingly:

```
void spiInit(){
    // Configure pins
    GPIO_setAsOutputPin(SDN_B);
    GPIO_setAsInputPin(nIRQ_B);
    GPIO_setAsOutputPin(GPIO0_B);
    GPIO_setAsOutputPin(GPIO1_B);

    GPIO_setAsOutputPin(SDN_T);
    GPIO_setAsInputPin(nIRQ_T);
    GPIO_setAsOutputPin(GPIO0_T);
    GPIO_setAsOutputPin(GPIO1_T);

    GPIO_setAsOutputPin(RF_SS_T);
    GPIO_setAsOutputPin(RF_SS_B);
    GPIO_setAsPeripheralModuleFunctionInputPin(RF_MOSI,
                                               GPIO_PRIMARY_MODULE_FUNCTION);
    GPIO_setAsPeripheralModuleFunctionInputPin(RF_MISO,
                                               GPIO_PRIMARY_MODULE_FUNCTION);
    GPIO_setAsPeripheralModuleFunctionInputPin(RF_CLK,
                                               GPIO_PRIMARY_MODULE_FUNCTION);

    GPIO_setAsPeripheralModuleFunctionOutputPin(EPS_SDA,
                                                GPIO_PRIMARY_MODULE_FUNCTION);
    GPIO_setAsPeripheralModuleFunctionOutputPin(EPS_SCL,
                                                GPIO_PRIMARY_MODULE_FUNCTION);

    GPIO_setAsPeripheralModuleFunctionInputPin(OBC_SCL,
                                               GPIO_PRIMARY_MODULE_FUNCTION);
    GPIO_setAsPeripheralModuleFunctionInputPin(OBC_SDA,
                                               GPIO_PRIMARY_MODULE_FUNCTION);
    PMM_unlockLPM5();

    //Initialize Master
    EUSCI_A_SPI_initMasterParam param = {0};
    param.selectClockSource = EUSCI_A_SPI_CLOCKSOURCE_SMCLK;
    param.clockSourceFrequency = CS_getSMCLK();
    param.desiredSpiClock = 1000000;
    param.msbFirst = EUSCI_A_SPI_MSB_FIRST;
    param.clockPhase = EUSCI_A_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT;
    param.clockPolarity = EUSCI_A_SPI_CLOCKPOLARITY_INACTIVITY_LOW;
    param.spiMode = EUSCI_A_SPI_3PIN;
    EUSCI_A_SPI_initMaster(EUSCI_A1_BASE, &param);

    GPIO_setOutputHighOnPin(SDN_T);
    GPIO_setOutputHighOnPin(SDN_B);

    EUSCI_A_SPI_initMaster(EUSCI_A1_BASE, &param);
    EUSCI_A_SPI_enable(EUSCI_A1_BASE);
}
```

Code C.2 – *SPI Protocol Initialization*

```
uint8_t rf4463_init(char slave){
    // I need to init SPI protocol outside this function

    // Reset the RF4463
```

```
    rf4463_power_on_reset(slave);

    // Registers configuration
    rf4463_reg_config(slave);

    rf4463_clear_interrupts(slave);

    // Set max. TX power
    rf4463_set_tx_power(slave, 127);

    // Check if the RF4463 is working
    if (rf4463_check_device(slave) == STATUS_SUCCESS){
        return STATUS_SUCCESS;
    }
    else{
        return STATUS_FAIL;
    }
}
```

Code C.3 – *RF Module Initialization*

```
bool rf4463_enter_standby_mode(char slave){
    uint8_t data = 0x01;

    return rf4463_set_cmd(slave, RF4463_CMD_CHANGE_STATE, &data, 1);
}
```

Code C.4 – *RF Module Standby Mode*

```
bool rf4463_tx_packet(char slave, uint8_t *data, uint8_t len){
    // Setting packet size

    rf4463_fifo_reset(slave);        // Clear FIFO
    rf4463_write_tx_fifo(slave, data, len);
    rf4463_clear_interrupts(slave);

    uint16_t tx_timer = RF4463_TX_TIMEOUT;

    uint8_t time;

    rf4463_enter_tx_mode(slave);

    while(tx_timer--){
        if (rf4463_wait_nIRQ(slave)){    // Wait packet sent interruption
            return true;
        }

        __delay_cycles(2000);// Should be around 10 us
    }

    // If the packet transmission takes longer than expected, resets the radio.
    rf4463_init(slave);

    return false;
}
```

Code C.5 – *Transmission of packets through RF Module*

Functions for turning the LEDs on and off are explained in Appendix B.

C.2 Results

There are two ways to check that the s have been released. The first one is connecting the blinking LED to the . Every time a is released, the will perform a (Figure C.2). The second way is receiving the s using a simulated on (Figure C.3) connected to a SDR.

This result is the expected. After transmitting a , there is a in the LED with a duration of almost 0.5 seconds. This is because of the relationship between the values of the microcontroller's clock frequency and the . The clock is set up at 8 MHz, while

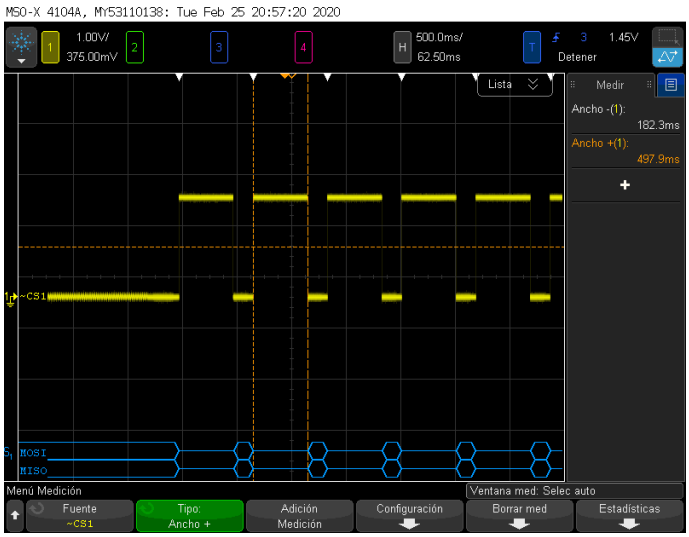


Figure C.2 – *Packets released*

the is at 4 MCycles. Taking into account this, it is understandable that the lasts for half a second.

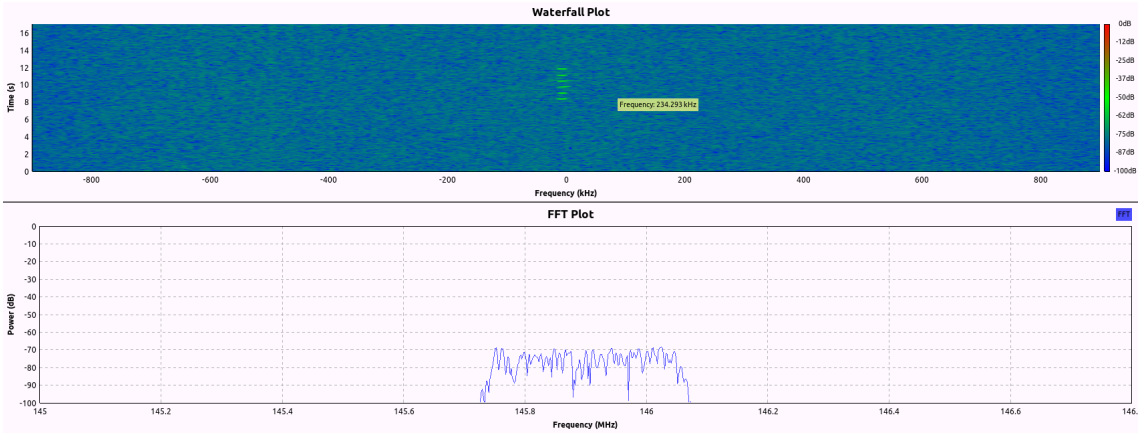


Figure C.3 – *Packet reception on GNU Radio*

As seen in Figure C.3, six s have been received in the simulated . These s are translated as six higher frequencies in the (in green).