



POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Elettronica

Tesi di Laurea Magistrale

Design of versatile and innovative wireless nodes for smart monitoring applications

Relatori

Prof. Daniele Trincherò

Ing. Mattia Poletti

Candidato

Simone TRINCHERO

ANNO ACCADEMICO 2017-2018

*† Ai miei genitori, Aldo
e Cecilia, e a mia nonna
Maria. A tutti coloro che
mi hanno sostenuto e a
tutti quelli che mi hanno
remato contro. E infine,
last but not least, a me
stesso.*

† *Le tecnologie aprono
orizzonti incredibili.*

*Internet non dà
informazione, dà
soltanto dati.*

*L'informazione si ha
quando i dati sono stati
assimilati dal cervello.*

Federico Faggin

Summary

The project described in this thesis conceived with the aim to realize a wireless device (SwiRem) to remotely monitor electrical equipments. Thanks to Wireless Sensor Network (WSN) technologies, it was possible to identify a pervasive monitoring infrastructure, with a considerable number of miniaturized and independent nodes, which sends information regarding the monitored system to a central collector. The integrated ad-hoc WSN solution can remotely monitor the power consumption and switch ON/OFF the power supply of a wide number of electrical devices, deployed in an extensive outdoor area. Moreover, the monitoring system must be separated and independent from the monitored one and needs an alternative network to send data and to receive commands. To face the problems of design, construction and testing, it was necessary to create a team of two students, who carried out the whole job together. The work was divided in four fundamental steps:

- Deep analysis of the problem.
- Analysis of suitable RF transceiver
- Firmware and PCB Computer-Aided Design (CAD)
- Assembly and on-field tests

TI CC1310 SwiRem system

The SwiRem devices fall into the definition of Short Range Devices (SRD) and in Europe must use the 433MHz or 868MHz industrial, scientific and medical (ISM) spectrum and must operate between 25mW to 100mW of Emitted Radiation Power (ERP), depending on the frequency band and on the transmission duty cycle, according to the European Conference of Postal and Telecommunications Administrations (CEPT). Among many integrated transceivers analysed, two solutions were identified: the Texas Instruments (TI) CC1310 868MHz, and the Murata LoRaWAN CMWX1ZZABZ. The SwiRem system with the TI CC1310 transceiver includes two boards designed in a modular way, to allow easy upgrade and to minimize costs and implementation time. Technically speaking, the boards are the

structural devices of a single-hop WSN and so they were named Master and Node in relation to the function they perform. The Master operates as collector/gateway and the Node manages the on-site sensing functions. Both the Master and the Node have a common RF hardware design, then they have different sensors, actuators and peripherals. The master can be connected to a Single Board Computer (SBC), like the RaspberryPi, to implement more complex activities and to have an Internet connection.

Both master and node boards use the TI CC1310 transceiver. The main parameters of this Integrated Circuit (IC) are the following:

- Wireless M-Bus and IEEE 802.15.4g PHY
- Sensitivity -110 dBm at 50 kbps
- Output power level up to $+14$ dBm
- Low power features (Standby: $0.6\mu A$)
- Power supply range: $1.8 - 3.8V$

This IC transceiver can achieve high RF and computational performances without compromising the low-power management of the hardware resources, thanks to an integrated power management system. A fully working Printed Circuit Board (PCB) was designed following, among the many, these main strategies:

- A 4-layers stack-up PCB, to separate signal path and power path and to minimize Electromagnetic Compatibility (EMC) problems.
- An accurate design of the RF path to match a $50\ \Omega$ antenna
- An accurate power management to accept wide voltage power supply

Texas Instruments implemented his proprietary wireless stack protocol, named TI SimpleLink, with an IEEE 802.15.4g PHY layer. To manage the RF drivers in the firmware, TI provides a complete suite of Application Programming Interface (APIs), named EasyLink, that allows a deep interaction with the hardware. The TI EasyLink APIs and others TI tools were used during the firmware design with Code Composer Studio software. After the analysis of a wide range of possibilities, it was implemented a firmware that manages a handshake behaviour, to guarantee the exchange of data between the master and the nodes. The communication starts when a node broadcasts a `wake_up` packet that includes his address. Afterwards, when the master receives this packet, it sends to the node a specific command or a data request and waits for an acknowledge packet that confirms the success of the communication.

Node board

The node is the core of the system, it monitors the specific Device Under Test (DUT) and holds all the needed actuators and sensors. This board can be powered both from an external power supply unit or from two AA 1.5V batteries, to guarantee the highest deployment versatility. The TPS2111A Auto-Switching Power Mux handles the switching between the two power sources. In this first realization, the node holds two main devices on it: a normally close Relay, able to manage a resistive load of 1A @ 30VDC and the IC INA226 I2C Current Shunt and Power monitoring. The firmware sets the node in a low power mode, then wakes it up every 3 seconds (this time interval is customizable) and waits for a command from the master. It executes the command and, if needed, it sends back data to the master.

Master board

The master board is the arbiter of the system and it represents the collector of the single-hop WSN. In this version it does not include any sensor or actuator, but it was arranged to be updated as needed. This board accepts different power supply, alternatively from the power grid or from the 5V of a specific RaspberryPi pin. The master manages two main tasks: the communication with nodes and the communication with the RaspberryPi through Serial Peripheral Interface (SPI). The master stores commands received from SPI in a queue and converts them in RF packets to be sent. Moreover, it implements the opposite conversion: it sends back, through SPI, the RF received data. Furthermore, it must memorize possible warnings about nodes that does not respond to commands, to report them.

LoRa SwiRem system

Due to the fast growth of new WSN IoT solutions, the SwiRem system faced the challenge to reach larger audience with enhanced reliability. For this reason, the CMWX1ZZABZ LoRaWAN transceiver from Murata was introduced. The main parameters of this System-On-a-Chip (SOC) are the following:

- LoRa PHY Spread Spectrum technology
- Sensitivity down to -148 dBm
- Output power level up to +20 dBm
- Low power features

- Power supply range: 2.4 – 3.6V

This SoC, physically designed and commercialised by Murata, integrates two ICs: a Semtech SX1276 LoRa transceiver and a STMicroelectronics ARM Cortex M0 STM32L0 microcontroller. A system based on this transceiver has some fundamental differences compared to the one based on the TI CC1310: it is necessary to design only the nodes because the wireless communication in a LoRaWAN network, based on the Long Range characteristics of the LoRa physical layer, allows a single-hop link between nodes and one or many gateways that acts as transparent bridge between RF packets and Internet IP packets and vice versa. This approach improves the reachable distances and facilitates the connection of nodes with Internet servers, simplifying data collections, storage and usage. A group of the LoRa Alliance community, named The Things Network, implemented a secure and redundant collaborative LoRaWAN Internet of Things network that allowed to test LoRa nodes exploiting public gateways. The Semtech LoRa spreading spectrum modulation has better communication and noise rejection behaviour compared to other transceivers, including the CC1310. The implementation of the SwiRem LoRa version is in a development phase, the CAD schemes were realized for this thesis.

Tests

After some hardware-software compatibility tests, performed in a controlled indoor environment, the CC1310 SwiRem system was tested in real environment. The chosen area was the one around the Santuario di Crea at Serralunga di Crea (AL). Several measurements have been taken, increasing progressively the distance between the two boards. The master was put on the top of a bell tower and a node was put in various positions in the valley. For every point the RSSI was measured and compared with theoretical values. For every test the measurements were coherent, and the system worked properly: the node correctly received the commands from the master and executed them. Furthermore, the master, received back data from node when necessary. The greatest distance reached was around 10 Km.

Conclusion

The work presented in this thesis shows the design, realization and testing of an ad-hoc WSN monitoring system. This monitoring approach will be certainly useful in various applications, reducing the time response to a possible breakdown of the device under control. This system allows to electrically turn off a device, without going to the physical place, for power saving aims or to force a power on reset. Furthermore, it can collect important data about power usage. In conclusion, the testing process highlighted the correct behaviour of the system and confirmed the achievement of expected results.

Contents

List of Figures	13
1 Introduction	18
1.1 Preface	18
1.2 Project background	19
1.3 Overview and innovative contributions	20
1.4 Thesis's structure	21
2 Survey on background knowledge	23
2.1 Internet of Things (IoT)	23
2.2 Wireless Sensor Networks	24
2.3 868 MHz wireless communication	26
2.4 Main standards on the market	27
2.4.1 IEEE 802.15.4g PHY	29
2.4.2 Wireless M-Bus	30
2.4.3 LoRa WAN	30
2.4.4 ZigBEE	30
2.4.5 Z-Wave	31
2.5 Fundamental parameters to evaluate radio communication	32
2.6 Main radio modulation techniques	32
2.6.1 FSK modulation	32
2.6.2 PSK modulation	33
2.6.3 ASK modulation	34

2.6.4	Spreading Spectrum Modulation	34
2.7	ICs communication protocols	35
2.7.1	SPI protocol	35
2.7.2	I2C protocol	36
2.7.3	UART	38
2.7.4	JTAG standard	39
3	SwiRem system overview	40
3.1	Specifications of the system	40
3.2	General description of the system	41
4	Analysis of possible RF platforms	44
4.1	WiMOD LoRa	44
4.1.1	iM880A LoRa characteristics	46
4.1.2	WiMOD LR Studio software and tools	47
4.2	Murata LoRa	48
4.3	TI SimpleLink CC1310	50
4.3.1	TI CC1310 characteristics	51
4.3.2	TI software and tools	52
4.4	CC1310 vs LoRa and final choice	53
5	CC1310 preparatory concepts	54
5.1	Hardware features	54
5.2	Software features	56
6	Firmware design	59
6.0.1	Firmware of the Master	61
6.0.2	Firmware of the Node	67
7	Hardware design	71
7.1	Hardware design of the node	71
7.1.1	Power supply	72

7.1.2	CC1310	78
7.1.3	Load monitoring	82
7.1.4	The up level of the system	85
7.2	Hardware design of the Master	88
7.2.1	Power supply	88
7.2.2	CC1310	89
7.2.3	The up level of the system	89
7.3	Bill of Materials (BOM)	91
8	PCB realization	96
8.1	PCB stack manager properties	96
8.2	PCB Shape	99
8.3	Footprint and library	101
8.4	Placement and routing guideline	102
8.5	Final PCB	105
8.5.1	Node	106
8.5.2	Master	107
8.6	Gerbers, Eurocircuit and Costs	108
9	Assembly and debug	110
9.1	Assembly	110
9.2	Debug	113
10	Further improvements and system evolution	116
10.1	LoRa Alliance "The Things Network" services	116
10.2	SwiRem LoRa Murata Version	117
11	Testing	120
11.1	Controlled environment	120
11.2	Real environment	122
11.3	Evaluation of LoRa Murata board	126
12	Conclusions	127

Bibliography	128
A Firmware Code	131
A.1 B1_nodo_RADIO_v2.0.c	132
A.2 B1_nodo_RADIO_v2.0.h	139
A.3 B1_nodo_TASK_v2.0.c	140
A.4 B1_nodo_TASK_v2.0.h	144
A.5 B2_master_RADIO_v2.0.c	145
A.6 B2_master_RADIO_v2.0.h	152
A.7 B2_master_TASK_v2.0.c	153
A.8 B2_master_TASK_v2.0.h	159

List of Figures

2.1	Visual representation of Internet of Things potentiality	23
2.2	Wireless Sensor Network	25
2.3	Single-hop a Multi-hop structure	26
2.4	Distance vs. main wireless standards applications [8].	27
2.5	Comparison between the main wireless standards [8].	29
2.6	General stack of a wireless protocol based on IEEE 802.15.4 PHY, like on CC1310	29
2.7	An example of binary FSK	33
2.8	An example of a basic PSK modulation	34
2.9	An example of OOK	35
2.10	A typical SPI timing	36
2.11	A common SPI scheme	36
2.12	A typical I2C timing	37
2.13	A common I2C scheme	37
2.14	A common JTAG scheme	39
3.1	Functional scheme of the whole system	42
4.1	LoRa Development Kit.	45
4.2	The WiMOD IM880 transceiver	46
4.3	The LoRa Murata IC from [23]	48
4.4	CC1310 Development Kit.	50
4.5	The TI Dev Kit CC1310 module	51
5.1	The internal structure of CC1310 IC ©Texas Instrument	55

5.2	The 7x7 RGZ Package pin-out of CC1310 ©Texas Instrument . . .	55
5.3	Code structure with TI-RTOS. ©Texas Instrument	56
5.4	CC1310 software structure ©Texas Instrument	57
6.1	Flow diagram of Master-Node radio interaction	60
6.2	Firmware's Structure	61
6.3	Flow diagram of Master-Node firmware radio protocol	62
6.4	Firmware flow diagram of the master	63
6.5	Firmware flow diagram of the node	68
7.1	The schematic of Node power supply section.	73
7.2	Simple circuit to handle backup power supply	73
7.3	The pin configuration of TPS2111A from datasheet.	74
7.4	The true table of MUX from datasheet.	75
7.5	The power divider at input of the TPS211A.	75
7.6	The pin configuration of TPS63050 from datasheet.	76
7.7	The schematic of TPS63050 from datasheet.	77
7.8	The schematic of radio part and all the component around CC1310.	78
7.9	Equivalent LC circuit integrated of a Sub-GHz impedance matched balun from datasheet.	79
7.10	The pin-out of the Johanson Technology's balun from datasheet.	79
7.11	The pin-out of the two crystal quartz oscillator from datasheet.	80
7.12	The functional scheme of load monitoring.	82
7.13	The schematic of load monitoring with INA226 and the relay HY1Z-3V.	83
7.14	The pin configuration of INA226 from datasheet.	84
7.15	The schematic of INA226 from datasheet.	84
7.16	The internal schematic of Panasonic HY1Z-3V (bottom view) from datasheet.	85
7.17	The up level schematic of the whole system of the node.	86
7.18	The schematic of Master power supply section.	89
7.19	The up level schematic of the whole system of the master.	90
7.20	The pin-out of the Raspberry PI 2.	90

7.21	The Bill Of Materials of the Master Board part 1.	92
7.22	The Bill Of Materials of the Master Board part 2.	93
7.23	The Bill Of Materials of the Node Board part 1.	94
7.24	The Bill Of Materials of the Node Board part 2.	95
8.1	A generic standard 2 and 4 layer structure.	96
8.2	Texas Instruments design cross section chart of PCB stack for the CC1310.	97
8.3	The Eurocircuit design cross section chart of layers stack for a "stan- dard pool" board.	98
8.4	Altium layer stack manager.	100
8.5	Dimensions of the Gewiss GW44205 case from datasheet.	100
8.6	A) Shape of the node board. B) Shape of the master board.	101
8.7	Footprint extracted by Ultra Librarian Tool.	102
8.8	Guideline structure for the balun chip.	104
8.9	PCB realization rule of a 50 Ω trace.	104
8.10	Guidelines routing of the INA226 from datasheet.	105
8.11	Routing guidelines of the TPS63051 from datasheet.	106
8.12	Layout of the node.	106
8.13	3D model of the node.	107
8.14	Layout of the node.	107
8.15	3D model of the node.	108
8.16	The Eurocircuit final document.	109
9.1	The final panel from Eurocircuit.	110
9.2	The syringe ready for the pneumatic dispenser	111
9.3	The pick and place at work.	112
9.4	Left: the final Master Board v. 1.0. - Right: the final Node Board v. 1.0.	113
9.5	A sample of the node board inside the Gewiss Case	114
9.6	Clear connection scheme between the TI XDS100V3 programmer on the SmartRF06 Dev Board and SwiRem boards.	115

10.1 Eagle CAD schematic of LoRa Murata Transceiver.	117
10.2 Eagle CAD SwiRem Board	118
10.3 Radio design on a iXem PCB	119
11.1 The configuration used to test the system in controlled environment.	121
11.2 The configuration used to test the system in controlled environment with the Raspberry.	121
11.3 Geographical test point 1	124
11.4 Geographical test point 2	124
11.5 Geographical test point 3	124
11.6 Geographical test point 4	124
11.7 Geographical test point 5	125
11.8 Geographical test point 6	125

Chapter 1

Introduction

1.1 Preface

The experimental work presented in this Master Thesis took place in the framework of an important and innovative laboratory in the Department of Electronic and Telecommunication (DET) of Politecnico di Torino, the iXem Labs.

The iXem Labs, supervised by Professor Daniele Trincherò, deals with various and innovative content with a R&D aim:

- the study of wireless communication systems
- the research of advanced wireless communication techniques
- the promotion of wireless technology for the realization of telecommunication networks all over the world
- the overcome of the digital divide in Italy and between Occidental and developing countries
- the exploitation of advanced, reliable, low cost and efficient wireless technology to improve the living conditions, the environmental protection and sustainable production processes
- realization of Wireless Sensor Network (WSN) for Internet of Things (IoT).

The points just mentioned describe well the mission of the laboratory. Some of the most important known projects are: design and testing of dipole antennas with balun, precision farming systems, a smart hydrophone to find losses in a water pipe and the world distance communication record reached with the link between two

antennas at 5 GHz.

To approach the project presented in this thesis it was necessary to create a team strongly guided by Professor Daniele Trinchero and by supervisors, composed by two Master degree students: Donato Mirolli (211457) and Simone Trinchero (211453). This was necessary to face all the problems of design and test of a wireless system. The contribution to this project by the two Master students was equivalently engaged in all issues regarding the project like: the case study, the project choices, the assembly, the testing process etc and the work was equally divided. **In all the cases where was necessary to test the wireless devices, the collaboration between minimum two person was mandatory. The complexity of PCB design of two boards including microcontrollers and transceiver requires a strong cooperation regarding hardware and firmware knowledge. In most of the cases every member of the team resolved a technical issue and after that there was a discussion to solve any potential problem, to match every project choice thus ensuring the correct progress of the project.** One example is the division of the work regarding the firmware writing: Simone wrote some functions in the while loop of the master board and some others to handle a sensor in the node; vice versa Donato wrote some functions in the while loop of the node board and some others to handle the SPI interfacing for the master. Same approach for the hardware design.

1.2 Project background

Nowadays the Internet of Thing is becoming more and more present, ensuring additional benefits in a wide range of situations. The increasing interest in this new computing application forced a lot of technical and political institutions to take an active part in this "new industrial revolution".

Strong signs, that evince the fast growing of IoT technologies, came from the most important sectoral world conferences. In the 44th ECC Plenary meeting (Electronic Communication Committee of the European Conference of postal and telecommunications Administrators) in 2017, were approved a range of deliverables and made decisions on spectrum issues covering in particular Machine-to-Machine (M2M)/Internet of things (IoT) applications, short ranges devices and wireless broadband communications.[1] Also in 2018, the IEEE 4th World Forum on Internet of Things in February, introduced fundamental innovations for all the various fields of applications like Industrial IoT, smart cities, agriculture and many others. The new edition of World Forum on Internet of Things in 2019 is already planned.

Another important sprint for IoT development, mainly managed by the Ngnm Alliance[2] (Next Generation Mobile Networks), is the imminent advent (by 2020) of the fifth generation wireless networks (the 5G technology) that, thanks to the implementation of ipv6, will be able to manage a bigger volume of users and handle with millions and millions of connected devices. According to that, a new dedicated network NB-IoT (Narrow-Band Internet of Things) have been implemented by the main telecommunications operators to face the increasing usage and need of IoT devices.

Very interesting and useful is the possibility to sense and monitor the environment with smart distributed systems implemented with a wireless sensor network (WSN) connected to the internet to exchange information. These WSN can be used in various monitoring applications like: domotic, building automation, environmental monitoring, protection, monitoring of energy consumption, smart agriculture and many others.

Following the iXem mission, with a strong desire of innovation, the aim of this work is the development of a versatile and innovative wireless system for smart monitoring applications. The development carried forward in this thesis include the hardware and firmware design of the two main devices of a wireless sensors network.

A secondary objective of this thesis was to use some innovative wireless transceiver, on a new project, to study and understand its potential to evaluate its introduction in some existing iXem systems for a technological upgrade.

This work starts with the analysis and study of existing wireless protocols, WSN structures and various wireless transceiver technologies, to select a suitable one. Then, as main goal, the focus was on: the firmware design for the chosen Integrated Circuit (IC) transceiver, the hardware design, the realization and testing of the two devices. The work ended with the physical realization of two Printed Circuit Board (PCB) that integrate the ICs radio transceiver and a selection of actuators and sensors that perform the needed monitoring activities. The system was implemented with two different transceiver technologies to allow useful comparisons, as mentioned before: for the first choice (the TI CC1310) the complete design flow was done and for the second one (the Murata LoRa) the flow ended with the CAD design. For the second technology (Murata) the physical realization of the prototype is not already produced. All the details about the system, the specifications and the testing environment are provided in the following chapters.

1.3 Overview and innovative contributions

The interest and the importance of the Environment is becoming very important in every business sector. The evolution of technology, that allow the realization

of new and innovative wireless applications, can improve the sensibility and the safeguarding of the environment we live in. As said before, a lot of applications of IoT are wireless sensors networks with a monitoring aim. A first example came as from iXem[3] as from other innovative realities and it is the intelligent farming monitoring, for example to reduce the usage of pesticides in grape crops.

Another example is the concept of city smart grid, implemented to meter the energy, water, gas consumption of the citizens. One concrete example about this, that allow to understand the importance of smart monitoring, is a TIM project in Torino, in collaboration with Olivetti and the Società Metropolitana Acque Torino. They have realized the first IoT intelligent water meter capable to send in real time the measures through the NB-IoT net. In this contest, a strong boost came from the power industry, which is now upgrading the power grid. Here IoT WSN technologies are playing a fundamental role in safety monitoring over power transmission and transformation equipment with the usage (or substitution) of billions of smart meters [4]. The project done for this thesis fits well the idea of power monitoring devices to evaluate their consumption and to avoid waste. This is a generic vision of this work, because the devices designed for this thesis are far away from a complete metering system for power saving.

A wide number of solutions exist on the market, that implement this idea for private homes, like the Bticino, Gewiss, Arduino and many others that use modern wireless protocols and standards, open source or proprietary. Also, at industrial level, there are many solutions, but this project wants to challenge existing systems by wagering on the concept of monitoring indoor or outdoor devices distributed over medium-extended geographic areas using free bandwidth wireless systems. In other words, the designed system have the mission to create a distributed monitoring network of sensors mainly independent from WiFi, Cellular connection and with a low power consumption. The aims of this monitoring are many, like: the necessity to harvest power consumption data for statistical or decisional aims, avoid power waste turning on/off a device when is not needed or force a power on reset on it, monitoring the power supply state of a device with a UPS system and many others. This is the innovative contribution that this thesis wants to give.

1.4 Thesis's structure

This section give to the reader a brief explanation about the structure of this thesis and why was structured in this way. It was decided, thanks to the suggestions of supervisors, to organize the sequence of chapters giving the same flow followed during the whole period spent in the design of this project, reporting the same steps followed to reach the final results. This was to ensure a better understanding

of the flow followed to approach all the problems encountered. In fact, after the introduction, is possible to find the following informations:

- In Chapter 2, a survey of all the important concepts needed to understand the world of WSN an IoT, useful to understand the following topics.
- In Chapter 3 starts the explanation of the SwiRem system, describing the general structure.
- In Chapter 4 and 5 there is the analysis of the wireless technology taken into account.
- **The core of the thesis** is concentrated in the Chapters 6, 7, 8, 9 where are explained in details all the firmware, hardware and assembly design choices about the CC1310 SwiRem devices. The succession of the topics covered in these chapters follow the same order in which they were be addressed during the work.
- The Chapter 10 briefly expose the initial upgrade steps of the designed system thanks to the introduction of the new Murata LoRa Transceiver and by the usage of an ecosystem of cloud tools implemented by LoRa Alliance partners.
- In Chapter 11 are exposed all the test and the results reached with the SwiRem system.
- A final Chapter, the 12th, contain the conclusions and reflections about the work done and about some possible future improvement that can be taken.

Chapter 2

Survey on background knowledge

2.1 Internet of Things (IoT)

The actual idea of connecting physical things together and to the Internet is an old idea. The term "Internet of Things" was, almost certainly, born in 1999 as the title of a Kevin Ashton presentation about the introduction of RFID in the Procter & Gamble supply chain.[5] Then for few years this concept remained in the dark until 2010, when the theme of Europe's biggest Internet conference LeWeb was the "Internet of Things". In the same years important magazines like Scientific American, Wired, Forbes started using IoT in their vocabulary to describe this concept of internet everywhere and in everything.

The ideas beyond the first concept of Internet of Thing, that also are the ideas that made this concept so important for the technological progress, are that we leave in a physical world and in our lives, we interact with things, not with ideas and information. So, the vision is the one where the Internet extends into the real world making possible to computers to sense by their own the environment, learn from it and help people in a infinite variety of situations.

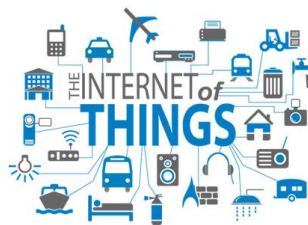


Figure 2.1. Visual representation of Internet of Things potentiality

The mission of IoT is becoming true, in the last 3 or 4 years, thanks to the huge and fast technology development that allow the integration of wireless embedded systems with a huge variety of smart object. This integration can guarantee a new observation of the real world at a new level of detail and at negligible cost. This new "under control" world with the capability to sense in real time data and react in automatic will be the scenario of many opportunities for new applications that can improve the quality of our lives, can deal with difficult situations and reduce waste of resources.

It is now clear how important IoT is and where is it from, but under a technical point of view, the Internet of Things is not a single new technology but the integration and cooperation of several technical developments that taken together make possible the bridge between the virtual and physical world. Some of the main capabilities that are needed to create an IoT device are: Communication and cooperation, Addressability, Identification, Sensing, Actuation, Embedded information processing, Localization, User interfaces. To realize an IoT device are needed lots of different technical solutions, starting from wireless protocols, then microcontroller solutions, perfect hardware and firmware cooperation for an embedded result, software and network solutions and many others. Some of this aspect are described in this chapter to be used in the following during the design explanations of this work. For a more detailed comprehension of what IoT is and how it works is possible, among others, refer to "From the Internet of Computers to the Internet of Things" [6]

The concept of IoT, intended as "sensors and actuators embedded in physical objects linked through wired and wireless networks" is intrinsically related with Wireless Sensor Network concept, sow is necessary to study it in deep.

2.2 Wireless Sensor Networks

As said before, the technology is now ready, thanks to the miniaturization based on micro-electro-mechanical systems (MEMS), to create low power, low cost and multifunction sensor objects with a very small size and that can communicate in short and medium distances. This kind of sensor can be, in cooperatively way, configured in a network that allow to cover and monitor a large and complex environment: it is a Sensor Network. Introducing a wireless communication between nodes and to a base station it is obtained a Wireless Sensor Network (WSN), more flexible than the wired solution. The development of WSNs come from military needs, like surveillance in conflict zones, but today it consists of distributed and independent devices that use sensors to monitor the physical world in a wide range of applications, as said before, like automation, industrial infrastructure, traffic, health and many consumer areas. A WSN system include several technical components

and infrastructures like a gateway, that manage wireless connectivity to the various point of the network and give access to acquired information to the wired world and distributed nodes that implement the wanted monitoring task. The wireless protocol used to communicate by nodes to each other and to the gateway depends on your application requirements. The most important protocols are explained in this chapter. A general WSN structure, with its components and infrastructures, is shown in Figure 2.2.

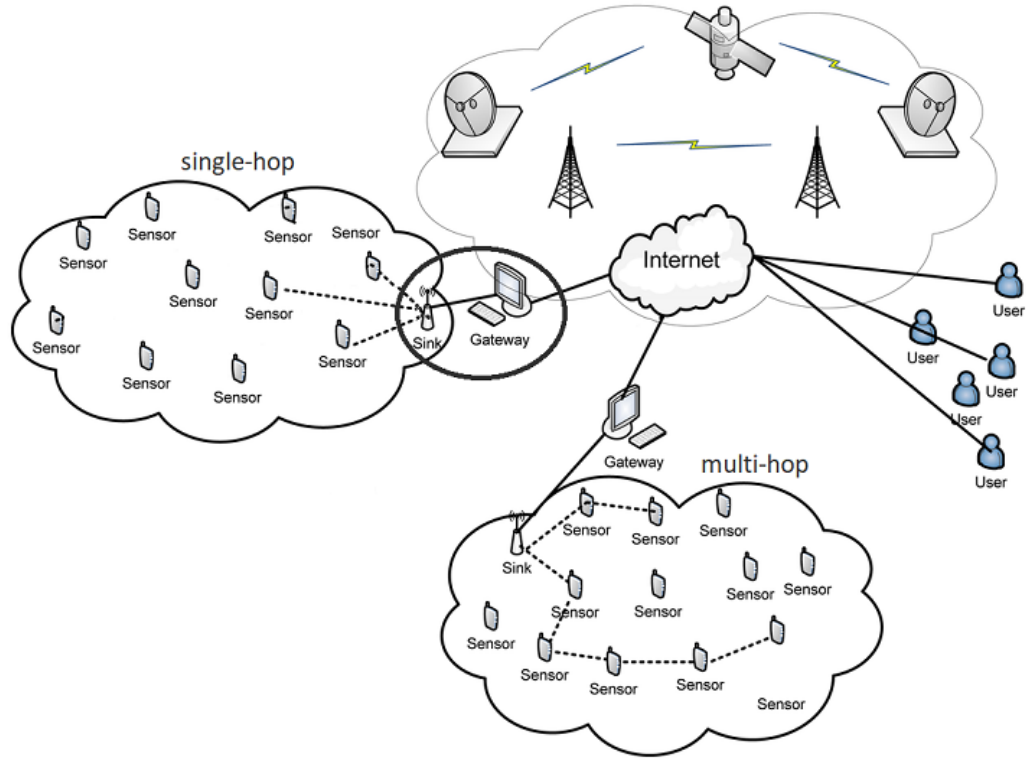


Figure 2.2. Wireless Sensor Network

It is possible to identify two main way to implement a WSN structure: multi-hop and single-hop. In figure Figure 2.3 are shown these two conceptual structures. The structure used in the implemented system is a single-hop communication because the characteristics of transceiver chosen and the morphology of the environment guarantee equally to reach greater distances without using aggregation node (multi-hop structure).

Since one of the goals of a WSN is the power conservation, the communication protocols and the whole firmware must consider the power management has one of the main parameters. To better understand the concept behind the WSN is

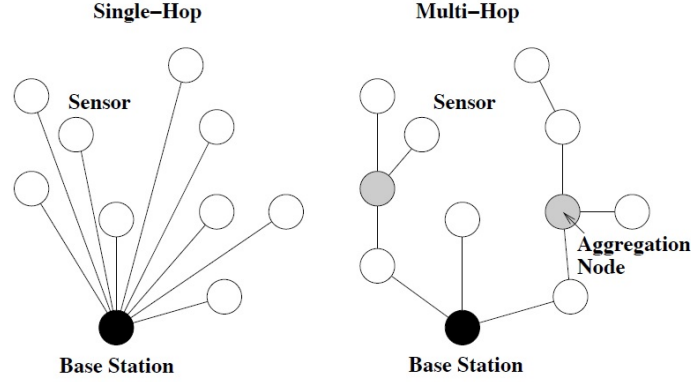


Figure 2.3. Single-hop a Multi-hop structure

possible to read the IEC white paper [4] and an interesting survey by the Georgia Institute of Technology [7] .

2.3 868 MHz wireless communication

To better understand why IoT devices use the 868 MHz band is necessary to investigate inside the European Institutions that decide the usage of the RF spectrum. IoT devices and WSN fall into the definition of "Short Range Devices" (SRD) that is an ECC Recommendation 70-03 (Electronic Communications Committee). This recommendation describes the radio frequency transmitter devices used in telecommunication for the transmission of information, which have low capability of causing harmful interference to other radio equipment. The ECC is a European institution that is responsible for radiocommunications and telecommunications matters and is formed by the merger of ECTRA and ERC (European Radiocommunications Committee) in September 2001. Going into detail, the Short-range devices are low-power transmitters typically limited to 25–100 mW effective radiated power (ERP) or less, depending on the frequency band, which limits their range to few hundred meters, and do not require a license from its user. Applications for short-range wireless devices include power meters and other remote instrumentation, RFID applications, radio-controlled models, fire, security and social alarms, vehicle radars, wireless microphones and earphones, traffic signs and signals (including control signals), remote garage door openers and car keys, barcode readers, motion detection, and many others. The European Commission mandates through CEPT and ETSI the allocation of several device bands for these purposes, restricts the parameters of their use and provides guidelines for avoiding radio interference. In Europe, 863

to 870 MHz band has been allocated for license-free operation using FHSS, DSSS, or analog modulation with either a transmission duty cycle of 0.1%, 1% or 10% depending on the band, or Listen Before Talk (LBT) with Adaptive Frequency Agility.

2.4 Main standards on the market

According to the specific application, WSNs have to evaluate the access technology to implement the wireless communication. Today a wide variety of different wireless standard technologies are remarkable to be used in this kind of applications, so it was necessary to resume the main ones and to understand how they work.

Considering the distance and speed of access, the access technologies can be grouped, following an increasing range order, in four categories: Personal Area Network (PAN), Wireless Local Area Network (WLAN), Wireless Neighbourhood Area Network (WNAN), Wireless Wide Area Network (WWAN). In Figure 2.4 are exposed some wireless access technologies grouped taking into account what category they belong to, and consequently in relation to the theoretical distance achievable. Are also heightened, for each category, the principal applications.

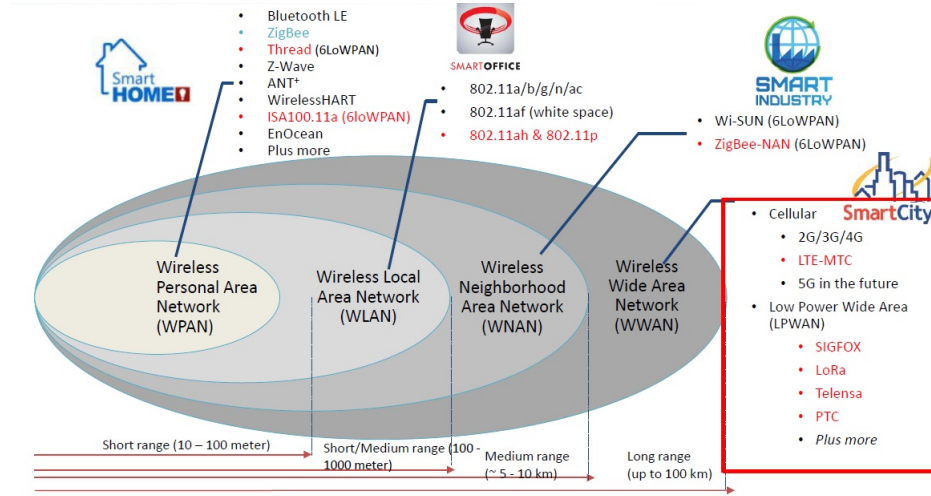


Figure 2.4. Distance vs. main wireless standards applications [8].

Before talking about the main standards on the market, used also in this project, it was necessary to introduce the OSI model for the communication functions of a telecommunication or computing system. The world of standards, regulations and trademarks is very complex and to understand well how a communication protocol

work is mandatory to know the structure of a wireless protocol. The model is a product of the Open Systems Interconnection project at the International Organization for Standardization (ISO), maintained by the identification ISO/IEC 7498-1. This conceptual model divides a protocol of communication in seven abstraction layers, where a layer serves the layer above it and is served by the layer below it. These layers, in order from one to seven are: Physical Layer, Data Link Layer, Network Layer, Transport Layer, Session Layer, Presentation Layer, Application Layer. This structure is also called stack of a protocol. It is important to focus on the two first layers, but to have a more complete description of the OSI model refer to [9] or to the ISO/IEC 7498-1.

The physical layer explains the physical and electrical specifications of the data connection. It defines the relationship between a device and a physical transmission medium (radio frequency link in this case). This includes the layout of pins, voltages, line impedance, cable specifications, signal timing and similar characteristics for connected devices and frequency and modulations for wireless devices. It is responsible for transmission and reception of unstructured raw data in a physical medium. The bit rate control is done at the physical layer. Furthermore, the data link layer provides node-to-node data transfer link between two directly connected nodes. It manages the detection of errors that may occur in the physical layer and try to correct them. At this stage the protocol to establish and terminate a connection between two physically connected devices is defined.[9]

Not all the standards use all the seven layers, but the first two are present in every wireless stack structure, so is important to focus on the standard that is the base for the upper protocols, that interest this project and in general the WSN for IoT devices. One among many is the IEEE 802.15.4 low data rate, low power networks standard. It defines the physical (PHY) and Medium Access Control (MAC) layers and forms the basis for numerous upper layer networking specifications. Other important PHY layer specifications are LoRa and WM-BUS.

Starting from the above-mentioned physical layer protocols, in the following are listed with a brief explanation the most common and useful protocols of interest. Obviously, some of these protocols are used by devices employed in this project. In the Figure 2.5 are reported some useful parameters to resume the main differences.








	NFC RFID	Bluetooth® Bluetooth low energy	Proprietary 2.4 GHz	ZigBee®	Wi-Fi®	6LoWPAN	Proprietary Sub-1 GHz
Network type	Identification 	Personal connection 	Customizable 	Mesh 	Existing infrastructure 	IP Mesh 	Customizable 
Range	Proximity	Personal area networks		Local area networks			Neighborhood area networks
Key differences	Data • Up to 848 Kbps • No battery to coin cell	Data or voice • Up to 3 Mbps • Coin cell to AAA	Data • Up to 1 Mbps • Coin cell	Data • Up to 256 Kbps • Energy harvesting to AAA	Voice or video • Up to 100 Mbps • AA battery	Data • Up to 256 Kbps • Energy harvesting to AAA	Data • Up to 1 Mbps • Coin cell
Industrial applications	• Device configuration / Firmware upgrade	• Lighting • Wire replacement • Beacons • Asset tracking • Factory automation	• Building and factory automation • Beacons	• Smart energy • Building automation • Lighting networks • Industrial Internet	• Assets tracking • Remote control of machinery • Sensors • Building automation	• Smart energy • Building automation • Lighting networks • Low-power Industrial Internet- gateways	• Metering • Smart grid • Alarm and security • Environmental monitoring

Figure 2.5. Comparison between the main wireless standards [8].

2.4.1 IEEE 802.15.4g PHY

IEEE 802.15.4-2006 is a technical standard that deal with the low-level protocol definitions corresponding to the OSI model physical and link layers. It was created for low-power devices in the 868 MHz, 915 MHz, and 2.45 GHz frequency bands in low-rate wireless personal area networks. Some of the most known standard of wireless communication like ZigBee, 6LoWPAN, etc. are built upward in the protocol stack and correspond to the network and transport layers, like shown in Figure 2.6, and it is the one present in the TI CC1310.

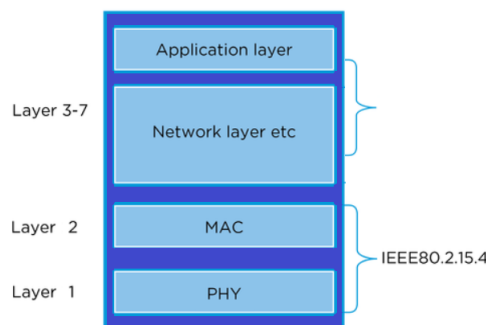


Figure 2.6. General stack of a wireless protocol based on IEEE 802.15.4 PHY, like on CC1310

2.4.2 Wireless M-Bus

The Wireless Meter Bus is a variant of the M-Bus or Meter-Bus which is a European standard defined under EN 13757. M-BUS cover all the layers of the communication stack protocol, starting from the physical one. It is becoming widely accepted in Europe for smart metering or Advanced Metering Infrastructure (AMI) applications. Wireless M-Bus was originally targeted to operate only in the 868 MHz band, which gives a good trade-off between RF range and antenna size. This stack can be implemented with the TI CC1310 transceiver.

2.4.3 LoRa WAN

LoRa is a proprietary spread spectrum specifications scheme that is derivative of Chirp Spread Spectrum modulation (CSS) and which can trade data rate for sensitivity within a fixed channel bandwidth. It implements a variable data rate, utilizing orthogonal spreading factors, which allows the system designer to trade data rate for range or power, to optimize network performance in a constant bandwidth.

The spread spectrum modulation is a method by which a signal with a particular bandwidth is spread in the frequency domain, resulting in a signal with a wider bandwidth. These techniques are used for a variety of reasons, including the establishment of secure communications, increasing resistance to natural interference, noise and jamming, to prevent detection, and to limit power flux density [10].

The forward error correction (FEC) or channel coding is a technique used for controlling errors in data transmission over unreliable or noisy communication channels. The central idea is that the sender encodes the message in a redundant way by using an error-correcting code (ECC).

LoRa is a PHY layer implementation and is separate with the higher-layer implementations. This allows LoRa to coexist and interoperate with existing network architectures. For more details refer to [11].

2.4.4 ZigBEE

ZigBee is a wireless networking standard that is aimed at remote control and sensor applications which is suitable for operation in harsh radio environments and in isolated locations. ZigBee technology builds on IEEE standard 802.15.4 which defines the physical and MAC layers. Above this, ZigBee defines the application and security layer specifications enabling interoperability between products from different manufacturers. In this way ZigBee is a superset of the 802.15.4 specification. With the applications for remote wireless sensing and control growing rapidly it is estimated that the market size could reach hundreds of millions of dollars as

early as 2007. This makes ZigBee technology a very attractive proposition for many applications. For more details refer to [12].

2.4.5 Z-Wave

The Z-Wave technology, working in the sub-1GHz band, is based on a low-power RF radio circuitry which is embedded into home electronics devices and systems, its strength is the interoperability. Z-Wave is used in a wide range of wireless home automation areas including lighting, entertainment systems and household appliances, residential access control. With many more home devices becoming remotely controlled, Z-Wave technology will have large market opportunities, especially in the contest of the Internet of Things. The Z-Wave MAC and PHY layers are defined by ITU-T Recommendation G.9959. In order to manage a hierarchical wireless network, various types of Z-Wave device are needed: Controller, slave, Routing slave. For more details refer to [13].

2.5 Fundamental parameters to evaluate radio communication

In this section are briefly described some of the main parameters that allow to evaluate the radio communications.

The following parameters are essential to characterise the quality of a radio transmission:

- PER - Packet Error Rate = $100 \cdot \left(1 - \frac{ReceivedPackets}{TransmittedPackets}\right)$
- RSSI - Received Signal Strength Indicator = measurement of the power present in a received radio signal
- SNR - Signal to Noise Ratio
- Latency - The latency is the time introduced by the signal travelling the geographical distance as well as over the various pieces of communications equipment
- Noise Floor - Is the measure of the signal created from the sum of all the noise sources and unwanted signals within a measurement system, where noise is defined as any other signal respect the one being monitored
- Budget Link - Is accounting all of the gains and losses from the transmitter, through the medium (free space, cable, wave-guide, fiber, etc.) to the receiver in a telecommunication system.

2.6 Main radio modulation techniques

In this section are described the main physical radio modulations used by a transceiver hardware. More in detail the focus is: FSK, PSK and ASK.

2.6.1 FSK modulation

Frequency-shift keying (FSK) [14] is a frequency modulation scheme in which digital information is transmitted through discrete frequency changes of a carrier signal. The simplest FSK is binary FSK (BFSK). BFSK uses a pair of discrete frequencies to transmit binary (0s and 1s) information. With this scheme, the "1" is called the mark frequency and the "0" is called the space frequency. The time domain of an FSK modulated carrier is illustrated in the Figure 2.7.

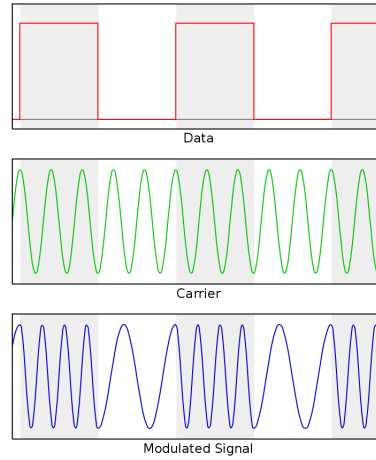


Figure 2.7. An example of binary FSK

Rather than directly modulating the frequency with the digital data symbols, "instantaneously" changing the frequency at the beginning of each symbol period, Gaussian frequency-shift keying (GFSK) filters the data pulses with a Gaussian filter to make the transitions smoother. This filter has the advantage of reducing sideband power, reducing interference with neighbouring channels, at the cost of increasing intersymbol interference.

2.6.2 PSK modulation

PSK or Phase-shift keying [15] is a digital modulation that encode data by changing the phase of a reference signal called the carrier. The modulation consists of the variations of the reference signal varying from sine to cosine and vice-versa in precise slots time.

A finite number of phases is used to represent data to form a unique pattern of binary digits. In other words, each pattern of bits is represented by a particular phase that is called symbol.

A simpler implementation of ordinary PSK is DPSK (Differential PSK), since there is no need for the demodulator to have a copy of the reference signal to determine the exact phase of the received signal.

The IEEE 802.15.4 (2.4.1) exploiting PSK use two frequency bands: 868–915 MHz with BPSK and at 2.4 GHz with OQPSK. Both QPSK and 8PSK are widely used in IoT technologies.

A basic PSK modulation is shown in the Figure 2.8.

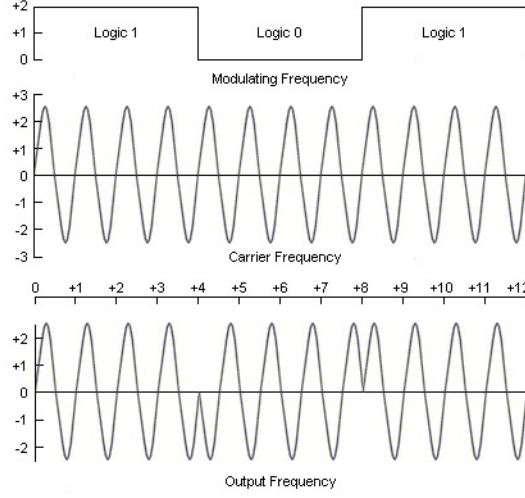


Figure 2.8. An example of a basic PSK modulation

2.6.3 ASK modulation

On-off keying (OOK) denotes the simplest form of amplitude-shift keying (ASK) [16] modulation that represents digital data at the presence or absence of a carrier wave. In its simplest form, the presence of a carrier for a specific duration represents a binary one, while its absence for the same duration represents a binary zero. Some more sophisticated schemes vary these durations to convey additional information. It is analogous to unipolar encoding line code. OOK is more spectrally efficient than frequency-shift keying, but more sensitive to noise when using a regenerative receiver or a poorly implemented superheterodyne receiver. The OOK modulation is shown in the following Figure 2.9.

2.6.4 Spreading Spectrum Modulation

The Spread Spectrum Modulation is a method to spread in the frequency domain the bandwidth of a signal, obtaining a signal with a wider bandwidth. These techniques are used for a variety of reasons, including the establishment of secure communications, increasing resistance to natural interference, noise and jamming, to prevent detection, and to limit power flux density.

Forward Error Correction (FEC) or Channel Coding is a technique used for controlling errors in data transmission over unreliable or noisy communication channels. The central idea is that the sender encodes the message in a redundant way by using an Error-Correcting Code (ECC).

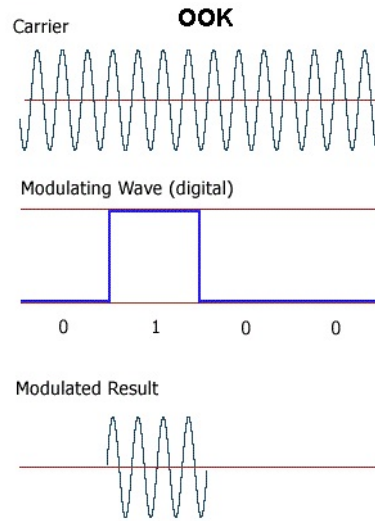


Figure 2.9. An example of OOK

Spread Spectrum Modulation and Forward Error Correction are techniques that increase the range and robustness of radio communication links compared with traditional FSK or OOK based modulation.

2.7 ICs communication protocols

Serial bus gives several advantages in the device's wiring. For this reason, they are widely present on chips like sensors, converters, analog switches, memory, etc. Below is reported a brief description of the main wired communication protocols and interconnection standards used in this thesis work.

2.7.1 SPI protocol

The Serial Peripheral Interface [17] (SPI) introduced by Motorola, is widely used for communications between ICs. It is a master-slave protocol that use only four wires. These wires are: one clock (SCLK), two data lines (Master Output Slave Input - MOSI and Master Input Slave Output - MISO) and a Slave Select (SS'). The most common realization of this protocol is the one in Figure 2.10.

There are many ways to connect the master device with the slave devices, but the most common one is the Figure 2.11.

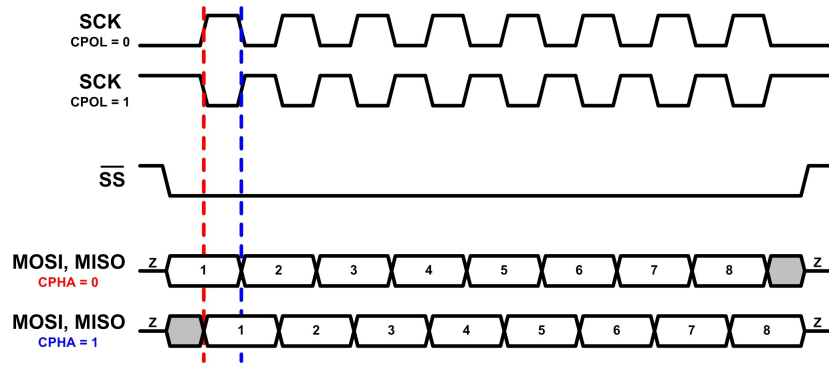


Figure 2.10. A typical SPI timing

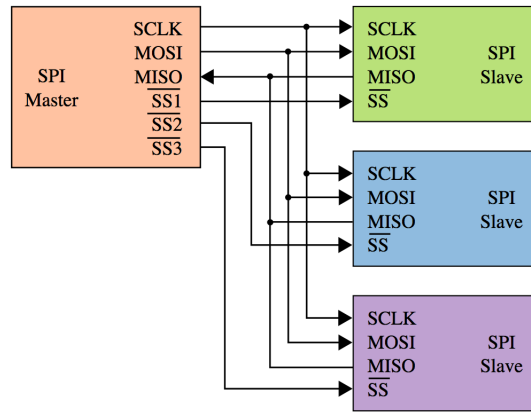


Figure 2.11. A common SPI scheme

2.7.2 I2C protocol

The Inter Integrated Circuit [17] (I2C) is a serial interface bus created by Philips (now NXP Semiconductor). This protocol uses two wires which are connected to all slaves ICs and there is not a separate chip select like the SS' in SPI protocol. Since there is not a SS', before sending data, is necessary to send an address to identify the correct slave; this address is sent on the same line of data. It is important to notice that the I2C is half-duplex and any device connect to the bus can become a master when the current master leaves the control. In Figure 2.12 is shown the typical timing protocol.

The two wires of this bus are a clock line (SCL) and data line (SDA). Both are connected by pull-up resistors to the supply voltage. A typical connection is shown in the Figure 2.13.

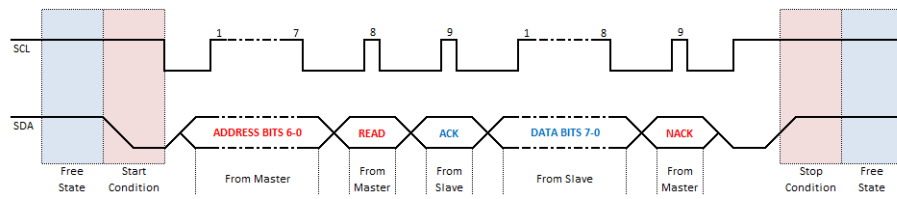


Figure 2.12. A typical I2C timing

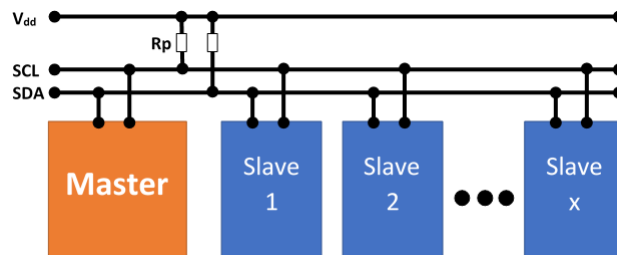


Figure 2.13. A common I2C scheme

2.7.3 UART

The Universal Asynchronous Receiver Transmitter [18] (UART) is an asynchronous serial communication protocol created by Gordon Bell. It is a hardware device that perform the translation of parallel data in to serial forms. UARTs are generally used with RS-232 standard. The electric signal levels and methods (such as differential signal etc.) are handled by a driver circuit external to the UART. The UART takes bytes of data and transmits the individual bits in a sequential fashion. At destination, a second UART reassembles the bits into complete bytes. Each UART contains a shift register, which is the fundamental method of conversion between serial and parallel forms. Serial transmission of digital information (bits) through a single wire or other medium is less costly than parallel transmission through multiple wires. A UART usually contains the following components:

- a clock generator, usually a multiple of the bit rate to allow sampling in the middle of a bit period
- input and output shift registers
- transmit/receive control
- read/write control logic
- transmit/receive buffers (optional)
- parallel data bus buffer (optional)
- First-In, First-Out (FIFO) buffer memory (optional).

2.7.4 JTAG standard

The Joint Test Action Group [17](JTAG) standard, also known as IEEE 1149.1 [19]. It became necessary to deal with the surface mount IC. JTAG give the possibility to look at the register and the data path of the chip to understand which is the problem that necessary to debug. In fact, it is widely used to program and debug microcontroller (e.g. ARM). The JTAG is a four wire serial bus: a clock line (TCK), a mode select line (TMS), data in (TDI) and data out (TDO). Very often, when JTAG is used to program a microcontroller, is necessary to include the RESET signal. A curiosity is that the "T" in the name of the signals stay for "Test", giving a precise idea about the scope of this interface. In the Figure 2.14 are shown a basic scheme and timing of JTAG.

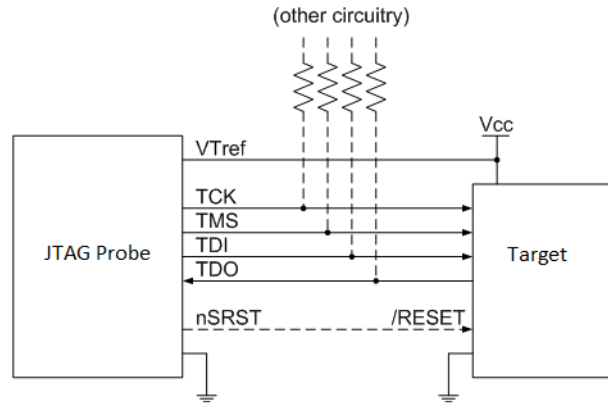


Figure 2.14. A common JTAG scheme

Chapter 3

SwiRem system overview

3.1 Specifications of the system

The idea to implement a WSN based on a custom system conceived with the need to electrically control and monitor a great number of devices, distributed in an extensive territory and often situated in uncomfortable places to be reached. So, there was the need to implement a stand-alone system that performs the task of remote control, monitor and meter of electrical characteristic of these distributed devices, from now called Devices Under Test (DUT).

The main idea was to have a simple and versatile interface to remotely control the DUTs without going in place. The system allows to know if one DUT has a problem, to know how much energy it was consuming and many other implementable features.

The core characteristics that the new system must have are:

- Monitoring of power consumption
- Sense the present of the main power supply of the DUT
- Possibility to turn it on or off remotely
- Low power consumption and battery supply for nodes
- Wireless 868 MHz connectivity
- Internet connectivity to provide collected data to users

The above characteristic implies the realization of two unit, a node and a base station or master ,as the teory of wireless sensor network explains. This structure is a general one, it is possible that in some cases is sufficient to design and realize only the node (that is the sensing object) because the so called "base station" can be already present, in function of the chosen communication system. To start the design of the system the problems were addressed as follows:

- Study of some transceiver technology, in particular Texas Instruments SimpleLink and LoRa Alliance products.
- Definition of the system's functionality and its related design specifications
- Firmware development
- Hardware development
- PCB realization, debug and testing
- Possible improvements with other technologies

3.2 General description of the system

The realized system is composed by two or more devices, one master and one or more nodes, to realize a single-hop WSN 2.2 . The nodes, that can be power supplied from battery or from power grid, are deployed where there is a DUT to be controlled and they perform the needed monitoring task, in function of the characteristic of the specific DUT. For example, a possible DUT can be a router board on a church bell tower that manage a radio link system or a Gateway deployed near a vineyard.

Whereas, the master has the function of gateway/access point and it is connected to the internet thank to a single-board computer (SBC), in this case a RaspberryPi with a 3G pen-drive. The master communicates via wireless with a node to send instructions of what to do and to collect the needed data about the monitored parameters of the remote device. The collected data are saved in an on-line database and they could be visualized from the end-users thanks to a GUI (Graphic User Interface). The back end processes (database and GUI) use an existing and working solution not aim of this project.

Due to a power saving strategy, periodically the node wakes up from sleep and starts a communication with the master, that is always in a reception mode (RX). Obviously, to avoid collisions, every node have a slightly different time to wake up and an anti-collision system procedures. At this point, the master has to

decide what the node has to perform, taking into account predefined situations and possible instructions coming from the user from an internet connection thanks to a Single Board Computer (SBC) like RaspberryPi.

The node, in this first realization, has the capability to: turn on/off a generic DUT to which it is connected, measure voltage and current (to monitor the power consumption) and sense the present of the main power supply. To better understand the configuration of the system is useful to look at Figure 3.1.

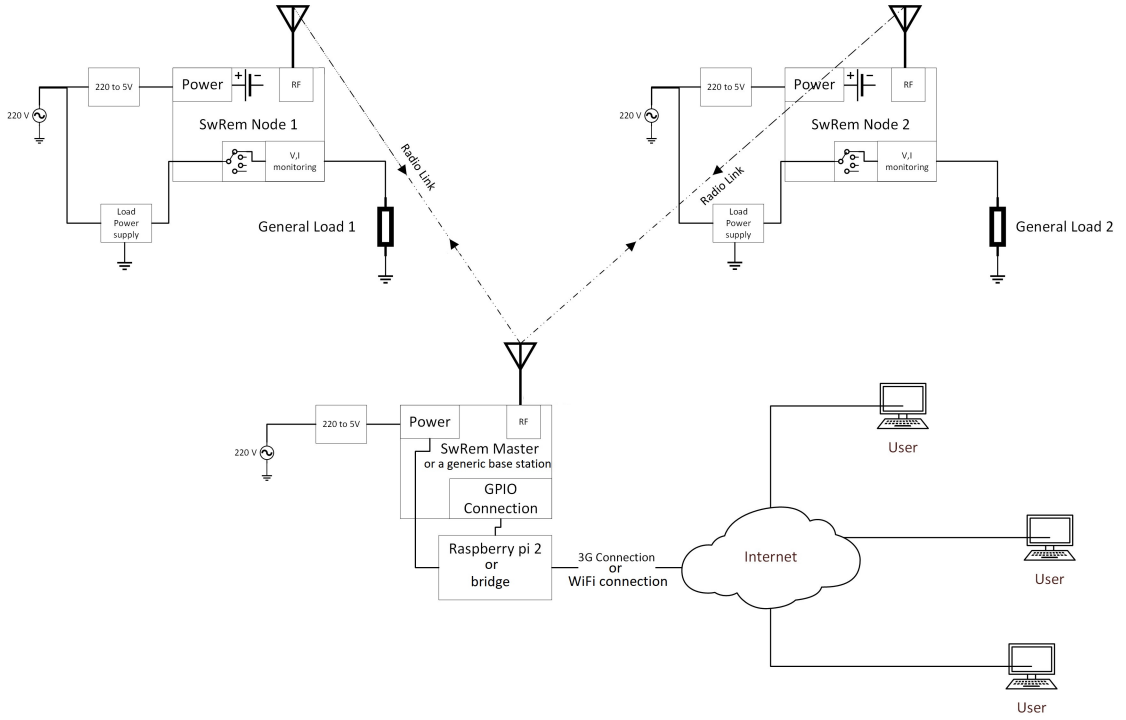


Figure 3.1. Functional scheme of the whole system

In the system schemes is easy to distinguish a general topology of two nodes attached to the respective DUT and the master collector, with a internet connection to reach the end users. Obviously, a single master can handle with a larger number of nodes, but the precise number is not yet defined. From the technical documents of the analysed transceivers, its possible to manage 50 nodes with a single base station. Other configurations, like the LoRaWAN, can handle hundreds of nodes. In order to realize an independent IoT wireless sensor network, every device of the network has the capability to understand the analysed phenomena and the autonomy to decide when to send data to the master, that become a collector. This is to ensure, for example, the immediate detection of an event, like the absence of the power supply of the DUT. For this reason, every node starts the communication by sending a

wakeup to the master with fixed intervals or when it detects a particular event. The master can handle the various requests coming from the nodes to supervise their operations. These choices are in agreement with the collaborative node structure in WSN theory. Than the master can decide, also processing information coming from user, which action a particular node must do.

In conclusion, this is the general solution elaborated to satisfy to specifications required, that in synthesis are: a remote switch and a power meter with wireless connectivity, possibility to upgrade the system, internet connection to collecting and saving data, management of distributed DUT in a not easily accessible environment.

Chapter 4

Analysis of possible RF platforms

At the very beginning of the project was necessary to study a wide number of wireless IoT oriented platforms, inside the definition of SRD that works at 868MHz. For the choice were taken into account several parameters like: distances to be reached, sensitivity in relations to the environment characteristics (mainly countryside), versatility of development tools related to the development and the time to market and possibly to expand and upgrade some existing iXem platforms. At the end of a selection process, three suitable solutions were analysed more in details: The LoRa WiMOD solution, the SimpleLink TI CC1310 solution (evolution of the TI CC110 already used in many iXem devices) and the LoRa Murata module. For the LoRa module the WiMOD solution was analysed because of the presence, in the laboratory, of the development kit, already used in other tests by iXem. On the market are now present different solutions that use a proprietary Semtec LoRa transceiver, like Arduino, Raspberry, Libelium, Murata etc. In the following, before exposing the final choice and explaining in detail the used product, is shown a brief description of analysed System on Chip (SOC), with a focus on the technical characteristics and on the available tools and development kits.

4.1 WiMOD LoRa

The WiMOD-iM880 is given with an evaluation kit that helps the test phase, and with a software that accelerates the settings of parameters and consequently the communication with the transceiver.

The main features present on the evaluation boards are:

- USB interface for communication with a PC
- Power supply by USB or battery
- 3 push buttons and 2 DIP switches
- 4 LED indicators
- Temperature sensor with 2-Wire Serial Interface
- Buzzer and potentiometer
- Expansion port (I/O connectors)

To study in deep and understand better the Evaluation Board refer to [20]

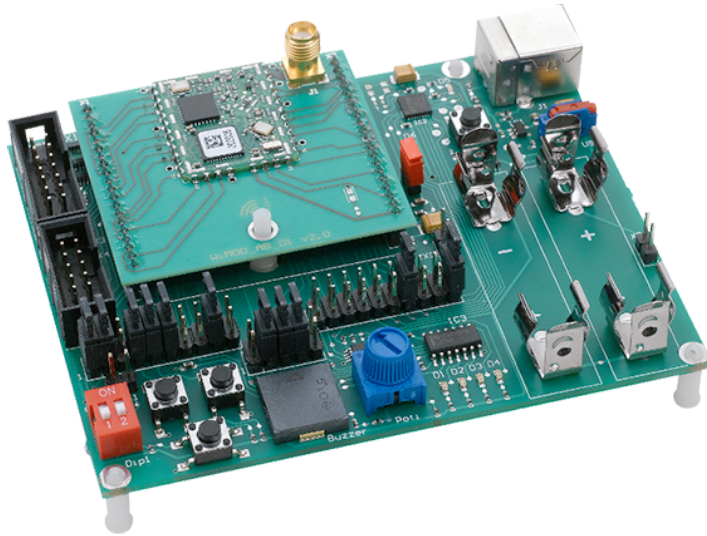


Figure 4.1. LoRa Development Kit.

4.1.1 iM880A LoRa characteristics

To better focus on the iM880A Long Range radio transceiver potentialities are reported the main characteristics of this system, considering that on the PCB solution designed by WiMOD are present two devices: a Semtec LoRa transceiver [21] and a ST microcontroller.

- Dimensions: 20.0 x 25.0 x 2 mm
- LoRa™ modulation technology ('**L**ong **R**ange' communications using very low power levels.)
- ST uC STM32L151Cx
- Semtech SX1272 transceiver
- Sensitivity down to -137 dBm
- UART, SPI and I^2C interface
- Analog and digital inputs – digital outputs
- Supply voltage range from 2,4 to 3,6 V
- RF interface optimized to 50Ω
- Output power level up to +19 dBm
- High link budget up to 156 dB
- Range up to 15000m (Line of Sight)
- Pre-Certified according to EN 300 220
- It operates in the license free 868 MHz SRD frequency band



Figure 4.2. The WiMOD IM880 transceiver

The iM880A uses a Semtech's patented LoRa module that, from the RF transmission point of view, it combines spread spectrum modulation and forward error correction techniques to increase the range and robustness of radio communication

links compared with traditional FSK or OOK based modulation according to the LoRa modulation as mentioned in section 2.4.3

In LoRa mode the iM880A offers three bandwidth options of $125kHz$, $250kHz$, and $500kHz$ with spreading factors ranging from 7 to 12. The spreading factor is the ratio between the nominal symbol rate and the chip rate; it represents the number of symbols sent per bit of information.

When two nodes communicate it is possible to identify a Local and a Peer. The direction from local to peer device is called Downlink. The peer device counts the number of received and transmitted packets and returns this status back to the local device within the so-called Uplink timeslot.

The local device communicates through serial COM with the software.

4.1.2 WiMOD LR Studio software and tools

The WiMOD LoRa studio software offers a wide range of testing tool, to easily interact and change the main parameters of the IM880 and the board. It is possible to identify a section where the transceiver parameters can be read and changed and a section where there are the real testing tools.

For the measurements we focus on the most important two tools.

Radio Link Test: The Radio Link Test can be used to verify the radio link quality between two iM880A modules. This test application measures THE PACKET ERROR rate by counting the number of transmitted and received packets on the local connected device and the peer device.

It is possible to modify the RF *Packet size* (from 15 byte to 64) and the *number of RF packet* (from 100 to 50000). It is also possible to save in an external log file all the measured values.

Data Link Service: The Data Link Service can be used to send radio messages from one iM880A to another in a comfortable way. This feature uses the "send unreliable radio message" service of the radio firmware. The user can define, by a dedicated box or by files, a payload to transmit (of maximum 245 bytes), and it is possible to set an automatic periodic transmission, setting the Transmit period. The tool shows the received payload (the receiver must be the one connected to the pc) and the measured *RSSI*, *SNR* and *RX time* of the received message.

For more information about the usage of the software please refer to the WiMODLR Studio User Guide [22]

4.2 Murata LoRa

The LoRa WAN IC CMWX1ZZABZ by Murata is a new version of IoT device that integrates in the same IC a Semtec SX1276 LoRa transceiver and a ST ARM Cortex M0 STM32L0 microcontroller (different from the WIMOD version that hold the two devices on a PCB). It is possible to test this IC with a DEV Kit by ST that allow an easy and rapid test of the most important features. The ST DEV Kit hold also the programmer that can b used to upload your own firmware on the transceiver. In the following are listed the main characteristics of this device:

- Dimensions: 12,5 x 11,6 x 1,76 mm
- LoRaLPWAN modulation technology ('**Long Range**' communications using very low power levels.)
- ST uC STM32L0 series
- Semtech SX1276 transceiver
- Sensitivity down to -135 dBm
- UART, SPI and I^2C interface
- includes 192kB flash and 20kB RAM
- Analog and digital inputs – digital outputs
- Supply voltage range from 2,2 to 3,6 V
- RF interface optimized to 50Ω
- Output power level up to +14 dBm (Max 20dBm)
- It operates in the license free 868 MHz SRD frequency band
- compatibility with "The things network" IoT system

 **LoRa Alliance**



Figure 4.3. The LoRa Murata IC from [23]

This innovative module is used in particular for smart metering, wearable, IoT edge nodes. The power of this system is the compatibility with a server infrastructure, The thing network [24] that eases the interconnection of IoT devices.

4.3 TI SimpleLink CC1310

The Texas Instruments CC1310 is the second transceiver taken into account. It is given with an evaluation kit that help to speed the comprehension of the CC1310 working. The kit includes two boards (that fits with various TI transceiver), two CC1310 Evaluation Modules 779-930 MHz (CC13xxEM-7XD-7793-4L), two W5017 IP-65 Pulse Antennas (2 dBi gain). The main features present on the evaluation boards are:

- USB interface for communication with a PC
- Power supply by USB or battery
- 4 push buttons and 2 DIP switches
- 4 LED indicators
- Accelerometer
- Ambient light sensor
- Micro SD Card reader
- LCD (to allow on board range test i.e.)

To study in deep and understand better the Evaluation Board refer to [25]



Figure 4.4. CC1310 Development Kit.

4.3.1 TI CC1310 characteristics

To better focus on the TI CC1310 radio transceiver potentialities are reported its main characteristics, taking into account that on the IC module are present many devices and two microcontroller: the main one is a Cortex M3 ARM and then there is an ARM Cortex M0 for the RF Core.

- RoHS-Compliant Package 7-mm \times 7-mm RGZ VQFN48 (30 GPIOs)
- Wireless M-Bus and IEEE 802.15.4g PHY
- Single-Ended or Differential RF Interface
- item Worldwide Radio Frequency Regulations: ETSI-EN 300 220, EN 303 131, MUX EN 303 204 (Europe)
- Powerful ARM Cortex-M3
- 48-MHz Clock Speed, 128KB Flash, 8KB of SRAM for Cache, 20KB of Ultralow Leakage SRAM
- A dedicated Radio Controller Cortex-M0 that handles low-level RF protocol commands
- Sensitivity -110 dBm at 50 kbps
- UART, SPI and I^2C interface
- True Random Number Generator, AES-128 Security Module
- RF interface optimized to 50Ω
- Supply voltage range from 1,8 to 3,8 V
- Output power level up to $+14$ dBm
- Low power features (Standby: $\mu A0.6A$)
- On-Chip Internal DC-DC Converter

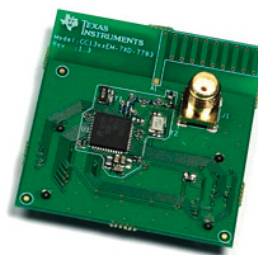


Figure 4.5. The TI Dev Kit CC1310 module

The CC1310 is a complex and highly technological Integrated Circuit (IC), with a high grade of integrations. It is classified in a Sub-1-GHz family, ultra-low power wireless MCUs. The CC1310 device combines a flexible, very low power RF

transceiver with a powerful 48-MHz Cortex- M3 microcontroller in a platform supporting multiple physical layers and RF standards. A dedicated Radio Controller (Cortex-M0) handles low-level RF protocol commands that are stored in ROM or RAM, thus ensuring ultra-low power and flexibility. The low-power consumption of the CC1310 device does not come at the expense of RF performance; the CC1310 device has excellent sensitivity and robustness (selectivity and blocking) performance.

4.3.2 TI software and tools

The TI give a complete suite of Software around the Smart RF system, that help the analysis and comprehension of all the CC1310 features in a fast way. The main tools are: SmartRF Studio 7, Sensor Controller studio and Flash Programmer 2. Over and above there are the CCS suite, but it will be considered separately from the previous cited TI tools.

SmartRF Studio 7 SmartRF Studio 7 is a PC software that can be used to manage several Texas Instruments development kits of the CCxxxx family. It helps designers, at an early stage in the design process, to evaluate in an easy way the RF-IC system for choosing the right product for the radio systems. The program provides a user-friendly graphic interface to understand and operate on all the chip's radio configuration registers, and it is very helpful for functional testing and for finding the appropriate radio settings. The direct access to the RF-IC's chip registers and radio-related features, and their XML storage help the designer not only to set and test the designed IC, but to understand and reuse important parameter that have to be included and set in the firmware. With SmartRF Studio it is easy to measure the signal strength (RSSI) and Packet Error Rate. There are 4 modes to set a Rf communication, in function of what is needed: Continuous RX/TX and Packet RX/TX.

Sensor Controller studio The Sensor Controller Studio is a software used to write, test and debug in an easy way code for the CC1310 Sensor Controller subsystem. The tool generates a Sensor Controller Interface driver, which is a set of C source files that are compiled into the System CPU (ARM Cortex-M3) application. The Sensor Controller is a small CPU core that is highly optimized for low power consumption and efficient peripheral operation and it is located in the CC1310 auxiliary power/clock domain, and can perform simple background tasks autonomously and independent of the System CPU and MCU domain power state. These tasks include such as: Analog sensor polling using the ADC or comparator, digital sensor polling (using bit-banged SPI, I2C or other protocols), capacitive sensing (using the

integrated current source, comparator and time-to-digital converter). The Sensor Controller is user programmable with a language that is very similar to C, and allows for sensor polling and other tasks to be specified as sequential algorithms. This approach is very powerful to give the possibility to develop very complex, low power features with CC1310.

4.4 CC1310 vs LoRa and final choice

The long and accurate analysis of this two different RF technology which, as mentioned above, has focused on: RF protocols, hardware and low power features, development tools and documentations.

Taking into account the specific task needed some fundamental differences were identified, that led to the choice of TICC1310:

- **TICC1310 can be used to improve and upgrade some iXem system that already use TI Transceiver (CC1110)**
- Range Tests, for the specific application, are similar
- TICC1310 have is more versatile respect to the WiMOD LoRa
- CC1310 can implement different IEEE802.15.4 based RF communication protocol
- SimpleLink products compatibility with little firmware changes.
- LoRa WiMOD have a proprietary firmware that impose the usage on another uC.

During the development of the system with the chosen transceiver, the CC1310, a new Semtech LoRa devices come out and communities of the alliance developed new infrastructures and new powerful devices like the Murata one. This evolution allowed the upgrade of the SwRem system with this new technologies to reach a larger audience with enhanced reliability. More details follows in 10.

Chapter 5

CC1310 preparatory concepts

From this chapter on starts the most important section of the thesis with strongly technical information. Purpose of this section was to deeply analyse the general information given in the previous one (Chapter 4) about CC1310. The main aspects pointed out will be useful characteristic of the hardware, software, features and structures that will be widely exploited during the design of the SwiRem boards. All the concepts here explained were allocated in a separate chapter because they regard a general view of CC1310 and can be exploited in other projects.

5.1 Hardware features

To design a system that includes the CC1310, as central unit and transceiver, it was necessary to analyse and understand some useful hardware characteristics, that are spread in a large number of datasheets and manuals. In the following are resumed the most important informations, essential for this work.

First of all it was important to understand generally the internal structure of the CC1310. It has four main sections inside like shown in Figure 5.1. There is the main CPU, the RF core, the general Peripheral and the sensor controller (the one explained in section 4.3.2)

The CC1310 have three different RGZ package dimension: 4x4, 5x5, 7x7 that make available respectively 10, 15 and 30 GPIO pins. In this project was chosen the 7x7 one, because of the prototype aim of the boards and because the ease to be handled. It was important to identify the pin-out of the CC1310 and their electrical characteristics and absolute maximum ratings.

In Figure 5.2 is possible to see the pins order and their usage. There are five main types of pins: GPIO and reset (all the DIO_N and pin 35), the power supply pins (13, 22, 23, 33, 34, 44, 45, 48), the RF pins (1 and 2), the oscillator pins (4,

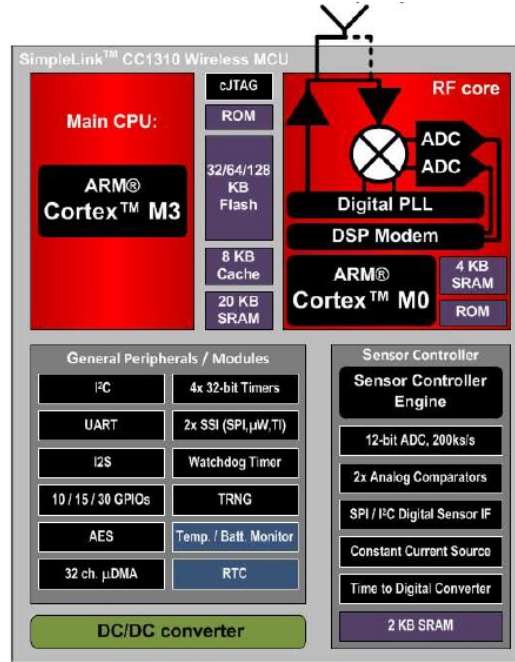


Figure 5.1. The internal structure of CC1310 IC ©Texas Instrument

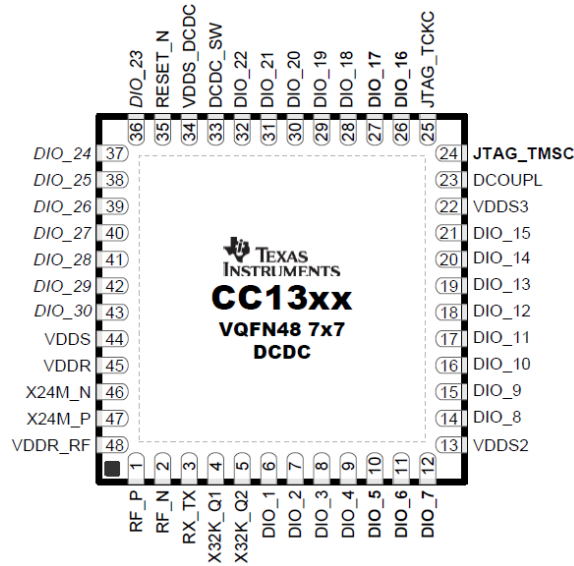


Figure 5.2. The 7x7 RGZ Package pin-out of CC1310 ©Texas Instrument

5, 46, 47) and JTAG pins (24, 25, 26, 27). The GPIO at pin 10, 11, 12, 26, and 27 have a high drive capability that is the capability to drive a maximum current of 8mA instead of only 4mA like the other GPIO.

5.2 Software features

The CC1310 is a complex environment with various level of abstraction that allow the designer to handle the hardware more quickly. A useful scheme to understand how a complete code structure of a CC1310 firmware based on TI-RTOS is shown in Figure 5.3 taken from the TI-RTOS User Guide [26].

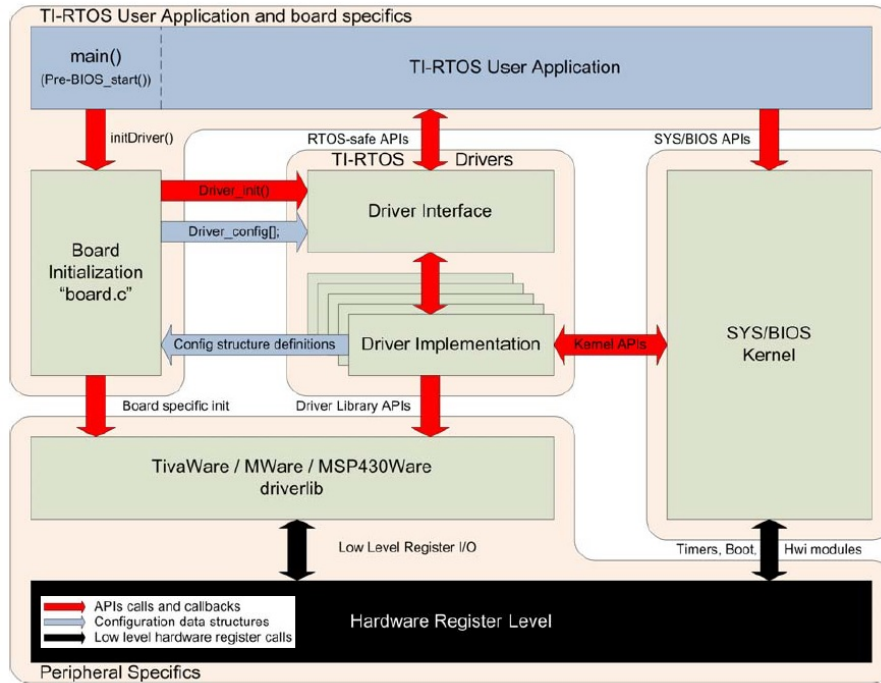


Figure 5.3. Code structure with TI-RTOS. ©Texas Instrument

The application firmware is based on the Texas Instrument Real Time Operation System, TI-RTOS. TI-RTOS is a scalable, one-stop embedded tools ecosystem for TI devices. It scales from a real-time multitasking kernel (SYS/BIOS) to a complete RTOS solution including additional middle-ware components and device drivers. By providing essential system software components that are pre-tested and pre-integrated, TI-RTOS enables the designer to manly focus on the application instead

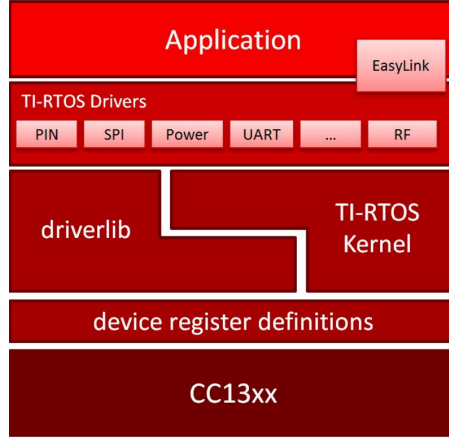


Figure 5.4. CC1310 software structure ©Texas Instrument

of focusing on low level code. TI-RTOS contains its own source files, pre-compiled libraries, examples and several components within its "products".

TI-RTOS provides various software instruments that are utilized for the writing of a complex firmware code. A stack structure of these tools is shown in Figure 5.4. These powerful tools are part of a very complex ecosystem that is articulated on various level of abstraction and is presented through dozens of technical papers from TI. After a careful analysis of these information, it was decided to report only the essential tools that were used for the design of the firmware for this application. In particular the focus is on the following user applications level tools:

- SimpleLink
- Power management

SimpleLink is the trade mark chosen by Texas Instruments to identify the RF TI 15.4 communication Stack. The CC1310 SimpleLink includes the software stack from TI, that implements the standard IEEE 802.15.4e and 802.15.4g specifications 2.4.1. More in detail, at a operational level, to realize a radio communication are used the EasyLink functions and SmartRF structures. EasyLink is a set of functions that perform all the needed radio operations. As an example, to send a packet it was possible to use, at high level code, the function *EasyLink_Status EasyLink_transmit(EasyLink_TxPacket *txPacket)*. Inside the EasyLink APIs are specified the packet structures used to collect and send the wanted data, like show in the following code.

```
/* Copyright (c) 2015-2016, Texas Instruments Incorporated
All rights reserved. */

/// \brief Structure for the TX Packet
typedef struct
{
    uint8_t dstAddr[8];        /// Dst Address
    uint32_t absTime;          ///Absolute time to Tx packet (0 for immediate)
    ///Layer will use last SeqNum used + 1
    uint8_t len;               ///Payload Length
    uint8_t payload[EASYLINK_MAX_DATA_LENGTH];    ///Payload
} EasyLink_TxPacket;

/// \brief Structure for the RX'ed Packet
typedef struct
{
    uint8_t dstAddr[8];        ///Dst Address of RX'ed packet
    int8_t rssi;               ///rssi of RX'ed packet
    uint32_t absTime;          ///Absolute time to turn on Rx when passed
    ///(0 for immediate), Or Absolute time that
    ///packet was Rx
    ///when returned.
    uint32_t rxTimeout;        ///Relative time in ticks from Rx start to Rx
    Timeout                    ///a value of 0 means no timeout
    uint8_t len;               ///length of RX'ed packet
    uint8_t payload[EASYLINK_MAX_DATA_LENGTH];    ///payload of RX'ed packet
} EasyLink_RxPacket;
```

These structures are filled with incoming or ready to send data and are managed by EasyLink. As interface, similar structures are created by the designer and are useful to manage the data before the usage of EasyLink functions.

Moreover, TI gives a versatile method to set all the dozens of registers to provide the correct functionality the RF hardware. All these parameters were also exportable from the SmartRF Studio Software 4.3.2.

Chapter 6

Firmware design

To handle with a micro-controller based device is necessary to design the firmware to control the hardware and allow to perform the wanted operations. This is a very important part, because the firmware gives the final intelligence to the device, but it is not possible to write a good firmware without deeply knowing the hardware to be controlled, they are strictly related. The firmware was realized with the IDE Code Composer Studio, that allow to manage all the configurations files needed to program the CC1310. As first approach, to create the two firmware for the two boards, the development kit was used. It was necessary to test and understand deeply which functions were needed, using a properly working hardware. All the firmware is based on TI-RTOS, the real time operation system of Texas Instrument, explained in the section 5.2.

The first basic operation designed was the structure of the radio communication protocol, to allow the interaction between two transceivers. Using the literature, the TI guidelines and the given examples, it was implemented a radio communication protocol to handle the exchange of information between the two boards. After many configurations, the final protocol that manage the correct succession of operations was created, with a handshake like structure. In Figure 6.1 is shown a schematic representation of the designed flow of operations.

To use and interact with the CC1310 RF driver the Texas Instrument SimpleLink family provides some useful APIs called EasyLink, as explained in the section 5.2. This APIs give the possibility, to the designers, to create a sub 1-GHz protocol with a good abstraction from the hardware, considering the concepts explained in section 2.4.1. This methodology is very modern and useful to guarantee an advantageous time to market and the possibility to a multi platform implementation.

The code is structured in a hierarchical file structure, like in the Figure 6.2. More in detail, this general structure allows a pre-emptive multitasking thank to

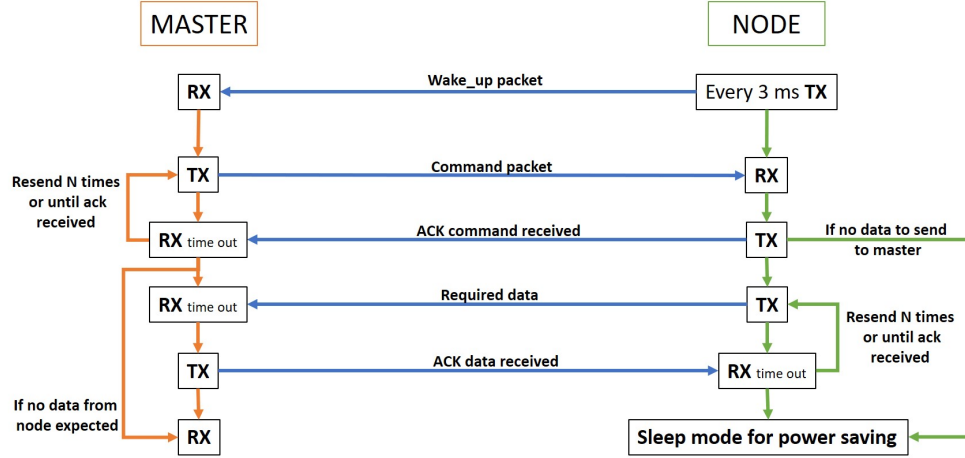


Figure 6.1. Flow diagram of Master-Node radio interaction

TI-RTOS Kernel. The task is a TI-RTOS kernel service: independent, pre-emptible thread of execution that has its own stack and can yield the processor. Every task, entirely defined in a separate .c file, are initialized by calling a *init()* function that provide to start it. In fact, in the main file there are only two *init()* fuction (one for main-task and one for radio-task) and the *BIOS_start()* function that invoke the startup sequence of SYS/BIOS (TI-RTOS kernel). The radio-task initialise the TASK that handle with the radio issues. In detail it initialise the task steak, the event queue and all the TI-RTOS OS functions needed in this TASK. The main-task initialise the TASK that handle with all the actions that the uC unit have to do. In detail it initialises the task steak, the event queue, SPI, and all the TI-RTOS OS functions needed in this TASK.

The core structure of the designed task is a loop that wait an event (thanks to the function *Event_pend()*) to decide how to evolve the operations. The event is a resource used to guarantee the communication between the different part of the task and tasks. Each function set an event to give the permission to the system to go ahead with the execution. To better explain this concept, it is necessary to use a practical example: when the function used to sense the present of a new radio packet in a RF driver identify the arrival of a new packet, it set an event to communicate the present of this new radio data to allow the prosecution of the code flow. In other words, the task implements a state machine system.

So, having understood the TIRTIOS functionality it was possible to realize the radio protocol between the two boards, shown in Figure 6.3

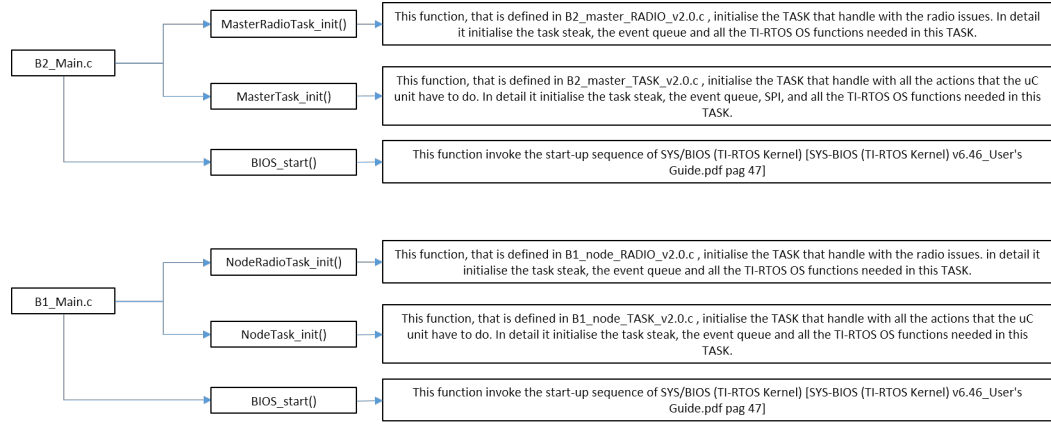


Figure 6.2. Firmware's Structure

6.0.1 Firmware of the Master

The flow diagram of firmware, shown in Figure 6.4, is formed by two tasks, one that handles the radio operations and one that handles all the operations of the micro-controller.

After power-on, TI-RTOS starts and all the values needed by the task are initialized. More in detail, are created the main structures of the task: the semaphore, the event queue, the task priority and the stack size (state M2 and M3). After the initialization, all the radio parameters are set (like modulation, frequencies, address, RX-filter, and so on) and the master goes in a synchronous receiving mode without time-out (M4 state) and it remains in Rx-mode until it receives a radio-packet from a node. If the received packet is a valid packet and a Wakeup flag_wake_received is set in M6 state otherwise it goes back in M4 state. In M7, the Radio-Task notifies to the Master-Task the node's information received and waits for a command coming from SPI. When a command comes from SPI (M13), the radio task sends it to the node and comes back in Rx-mode, but this time with a time-out (M8). The time-out is necessary to wait an acknowledge from the node, to be sure that it has received the command. A resend structure is implemented to send again n-times the command, until a valid acknowledge is received (M9). If any acknowledge is received, during n-resends, the transmission is considered failed and this failure is saved in an internal database that memorized this data for each node (M10 and M14). Otherwise, if a valid acknowledge is received, the communication is considered valid and the master come back in Rx-mode without time-out (M4). Now it can wait for another wakeup or a data packet from the node, if the initial command asked for the return of some data from the node (M5 and M14).

The firmware is structured to automatic memorize a new node inserted in the


```
MR4) enum MasterRadioOperationStatus MasterRadioTask_sendCommandData(uint16_t data
    )
```

The MR4 function is used, by main task, to communicate to the radio task which command the master have to send to a node. Furthermore, this function there is on of the control on a received wakeup that allow to send a command only if a wakeup is received. The semaphore TIRTOS tool guarantee the successful sending of the only one command at a time without encounter conflicts on rf driver request.

```
MR5) static void returnRadioOperationStatus(enum MasterRadioOperationStatus result
    )
```

This function collects the radio operations result and to decide how to set the semaphore for rf-driver and allow or not the permission to start or not a new rf communication.

```
MR6) static void sendCommandPacket(struct CommandPacket cmdPacket, uint8_t
    maxNumberOfRetries, uint32_t ackTimeoutMs)
```

The MR6 function realize a transmission of a generic packet (in this case the master send only a acknowledge or command packet) with an acknowledged waiting. Also, here are set the maximum number of retries and the wanted acknowledge time-out. Immediately afterwards a successful transmission the transceiver is set in reception with time-out.

```
MR7) static void sendAck(uint8_t MasterAddress)
```

This function implements the simply sending of an acknowledge packet.

```
MR8) static void resendPacket()
```

The MR8 function is similar to the MR7 function but it deals with the resending of an unsuccessful sent packet. It takes count the number of retries before set.

```
MR9) static void notifyPacketReceived(union MasterPacket* latestRxPacket)
```

The MR9 function assists the function MR1 to transmit the received data from a node. In particular it notifies the receipt of a package and physically give the data received to the main task also exploiting the function MT12.

```
MR10) static void rxDoneCallback(EasyLink_RxPacket * rxPacket, EasyLink_Status
      status)
```

This function that act as a callback active by the detection of an incoming rf transmission. This is the core of the radio protocol, in-fact it deals with the identification of the received packet type (analysing the packet header) and communicate it to the task while loop in MR2 with an appropriate event. It understands also the end status of every reception.

Functions of Master Task

```
MT11) static void masterTaskFunction(UArg arg0, UArg arg1);
```

The MT11 is the core function of the Master Task. Is divided in two main section: an initialization section and the task while loop. The main initialization regards the SPI and the reset of the database where are stored all the known node. In the while loop is implemented the decision process that guarantee the correct evolution of the algorithm thanks to the "event" tools.

```
MT12) static void packetReceivedCallback(union MasterPacket* packet, int8_t rssi)
      ;
```

The function MT12 is used by function MR9 as explained before. In particular it takes the packet data received from a node and copy each value received in a local structure to be used in this task.

```
MT13) void notify_toTask_addNode(int8_t address);
```

When the Radio Task identify a wake-up from a node, it uses this function to notify to the main task the address of the particular node. This is to allow the main task to know which node have talked and to decide which action to do. Moreover, it is responsible for starting the procedures for sending via the SPI the address of the received node.

```
MT14) void notify_node_notRespond(int8_t address);
```

When the Radio Task identify a not responding node it uses this function to notify to the main task the address of the particular node. This is to allow the main task to know which node not have talked and memorize it inside of the database.

```
MT15) static uint8_t isKnownNodeAddress(uint8_t address);
```

The function MT15 search inside the database if the talking node is already known.

```
MT16) static void updateNode(struct NodeDataPacket* node);
```

The function MT16 after the execution of MT15 function update some useful data (like RSSI of the last communication) inside the database if the node is known.

```
MT17) static void addNewNode(struct NodeDataPacket* node);
```

This function adds a new node to the database when it is found.

```
MT18) void spiInizializzazioni_blocking(int8_t address);
```

The MT18 function initialize all the need parameters to use SPI driver in blocking mode.

```
MT19) static void spiInizializzazioni_callback();
```

The MT19 function initialize all the need parameters to use SPI driver in callback mode.

```
MT20) static void spiCallback(SPI_Handle handle, SPI_Transaction *Trans);
```

This is the function the physically manage the SPI transmission saving the data receiving in a local structure.

6.0.2 Firmware of the Node

The flow diagram of firmware, shown in Figure 6.5, is formed by two tasks like in the master. Also in this case, after power-on, TI-RTOS starts and all the value needed by the task are initialized. More in detail, are created the main structure of the task: the semaphore, the event queue, the task priority and the stack size (state N2). After the initialization, all the radio parameters are set (like modulation, frequencies, address, RX-filter, and so on) and the power policy is set to give the permission to the system to go in deep sleep if there aren't any ongoing operations that denied it (N3). Obviously, a timer interrupt has to be used, to set when to wakeup, if any other operation does it. In this first approach the node wakes up every three seconds, but this value is customizable. When the timer reaches the prefixed value, the Node wakeup and immediately go in transmission mode and send a wakeup packet to the master and then it goes in Rx-mode with time out to wait an eventual command from the master (N4 and N5). If an invalid or any packet is received the node, after the rx time out, come back in sleep mode (N6). Otherwise, if a valid packet is received the node transmit an acknowledge to the master (N7). At this point, the node task that was waiting (N13) for a new command it is notified from the radio task about the presence of a new command (N8). Now the command is evaluated and execute (N14). If the operation required by the command does not need to send back data to the master (for example switching relay) after execution the node can go back to sleep. Instead, if it is necessary to send data, the radio task received the data to be sent (N9) and transmit it to the master. After the transmission the node goes in rx mode with time out for waiting an acknowledge from the master (N10). As explained in the master here is implemented a resend structures that try to send the data to the master n-times until it receives the acknowledge (N11). If the acknowledge is correctly received the power policy allow to go back to sleep.

Functions of Node Radio Task

```
NR1) void NodeRadioTask_registerPacketReceivedCallback(  
    ConcentratorRadio_PacketReceivedCallback callback);
```

The NR1 function implements the connection between the radio task and the main task of the node. More in detail it was used to connect the pointer to the two structures. It transforms the reception of a packet to a callback event.

```
NR2) static void NodeRadioTaskFunction(UArg arg0, UArg arg1);
```

The NR2 is the core function of the Node Radio Task. Is divided in two mains section: an initialization section and the task while loop. The main initialization


```
NR4) enum NodeRadioOperationStatus NodeRadioTask_sendCommandData(uint16_t data);
```

The NR4 function is used, by main task, to communicate to the radio task which data the node have to send to a master. The semaphore TIRTOS tool guarantee the successful sending of the only one data at a time without encounter conflicts on rf driver request.

```
NR5) static void returnRadioOperationStatus(enum NodeRadioOperationStatus result);
```

This function deals with the collection of the radio operations result and to decide how to set the semaphore for rf-driver and allow or not the permission to start or not a new rf communication.

```
NR6) static void sendCommandPacket(struct NodeDataPacket Packet, uint8_t  
    maxNumberOfRetries, uint32_t ackTimeoutMs);
```

The NR6 function realize a transmission of a generic packet (in this case the node send only a acknowledge or data packet) with an acknowledged waiting. Also, here are set the maximum number of retries and the wanted acknowledge time-out. Immediately afterwards a successful transmission the transceiver is set in reception with time-out. Before starting the transmission is necessary to disable the low power policy and then reactivate it after the transmission.

```
NR7) static void resendPacket();
```

The NR7 function is similar to the MR7 function but it deals with the resending of an unsuccessful sent packet. It takes count the number of retries before set.

```
NR8) static void sendWake(uint8_t MasterAddress);
```

This function implements the simply sending of an wakeup packet.

```
NR9) static void sendAck(uint8_t MasterAddress);
```

This function implements the simply sending of an acknowledge packet.

```
NR10) static void notifyPacketReceived(union ConcentratorPacket* latestRxPacket);
```

The NR10 function assists the function NR1 to transmit the received data from a master. In particular it notifies the receipt of a package and physically give the data received to the main task also exploiting the function NT14.

```
NR11) static void rxDoneCallback(EasyLink_RxPacket * rxPacket, EasyLink_Status
    status);
```

This function that act as a callback active by the detection of an incoming rf transmission. This is the core of the radio protocol, in-fact it deals with the identification of the received packet type (analysing the packet header) and communicate it to the task while loop in NR2 with an appropriate event. It understand also the end status of every reception.

```
NR12) void Clok_Radio_TimeoutCallback(UArg arg0);
```

This is essential to set the time interval in which the node wakeup sleep from power policy.

Functions of Node Task

```
NT13) static void NodeTaskFunction(UArg arg0, UArg arg1);
```

The NT13 is the core function of the Node Task. In the while loop is implemented the decision process that guarantee the correct evolution of the algorithm thanks to the "event" tools. In particular based on the received command in this section is performed the request action.

```
NT14) static void packetReceivedCallback(union ConcentratorPacket* packet, int8_t
    rssi);
```

The function NT14 is used by function NR10 as explained before. In particular it take the packet data received from a node and copy each value received in a local structure to be used in this task.

```
NT15) void readADCvalue();
```

This function manage all the operation to eventually read a value from internal ADC.

Chapter 7

Hardware design

The core of this work was the hardware design of the two boards with Computer-aided design (CAD) tools. The CAD software used was Altium Designer, and following the recommendations of a good CAD design a hierarchical organization of the schematic sheets was used. In other words, for every section of the project, was made a separate schematic sheet seen as black block. Then every block was linked together to create the final design. The choice to realize, as first prototype, an integrated Printed Circuit Board (PCB) was due to the necessity to test, as soon as possible, the reliability of a board including the TI CC1310 transceiver. Doing this, it was possible to test the radio capability and other parameters to evaluate the goodness of this IC separately from the development kit, over and above realizing the needed task of power monitoring. First of all, to manage the project of a complex system, it was necessary to understand and define a high level design of the necessary sub-systems that must be present in the PCB. The main design areas that it was possible to identify are: the power supply management, the radio management, the peripheral sensor/actuator management and the micro-controller management. In this case the RF hardware and the micro-controller are integrated in the same IC, the CC1310.

As explained before, the system is composed by two boards. This two boards have some common hardware components, like the transceiver, and some differences. In the next sections every design decisions are expanded, starting from the node.

7.1 Hardware design of the node

The node is the core of the system, is the board that have to monitor the DUT, in fact is the board that hold all the needed actuators and sensors. How explained

above, the board is characterized by three main sections, let's analyse them in detail.

7.1.1 Power supply

The main idea behind the designed power supply chain in the board is forced by the necessity to manage two voltage supply: the primary source and the secondary one, which guarantees the recovery power supply in absence of the primary one. This is basically the idea of a UPS system. To decide how to supply the board and every device inside it, it was chosen, as reference, the voltage supply of the CC1330, that is the most critical component. In this way it was possible to realize a single supply voltage board instead of a multi voltage one. Since a multi supply voltage PCB have a higher design complexity and the advantages are unnecessary in this case, the decision was done to simplify the design to minimize costs and farther to realize a robust power supply path. How explained in the section dedicate to the CC1310, the range of the supply voltage is 1.8 to 3.8 V. Since that the maximum voltage needed in the board is about 3.3 V (considering all the devices) it was chosen to use an external AC-DC power supply to transform the voltage of electrical grid into fixed 5 V. In this way, on the board, there is not the presence of high voltage which requires more complex approaches. Instead, for the secondary power supply was chosen to use the two 1.5 V batteries in series of 3 V overall. To reach the correct supply voltage, starting from the external supply, and to guarantee its goodness in term of stability, it was necessary to introduce some power conditioning circuitry. In the Figure 7.1 is reported the schematic of the designed power supply management for the Node

The design of this section required two main devices: an auto-switching power mux and a single inductor buck-boost, the Texas Instruments TPS63051.

Auto-switching power mux - TPS2111A

To manage the automatic switching between two power supply, the main one coming from the power grid and the other from battery, it was necessary to introduce an auto-switching power mux.

In electronic, is very common to handle with multi power supply, for example to guarantee power supply in the absence of the primary source or for many other reasons. In the literature is possible to find many circuital solutions, starting from the simplest, with discrete diodes and transistors, to more integrated and complex circuits, has the chosen one. In Figure 7.2 is shown the simplest circuit possible to handle with a backup power supply that have to feed the circuit when the main

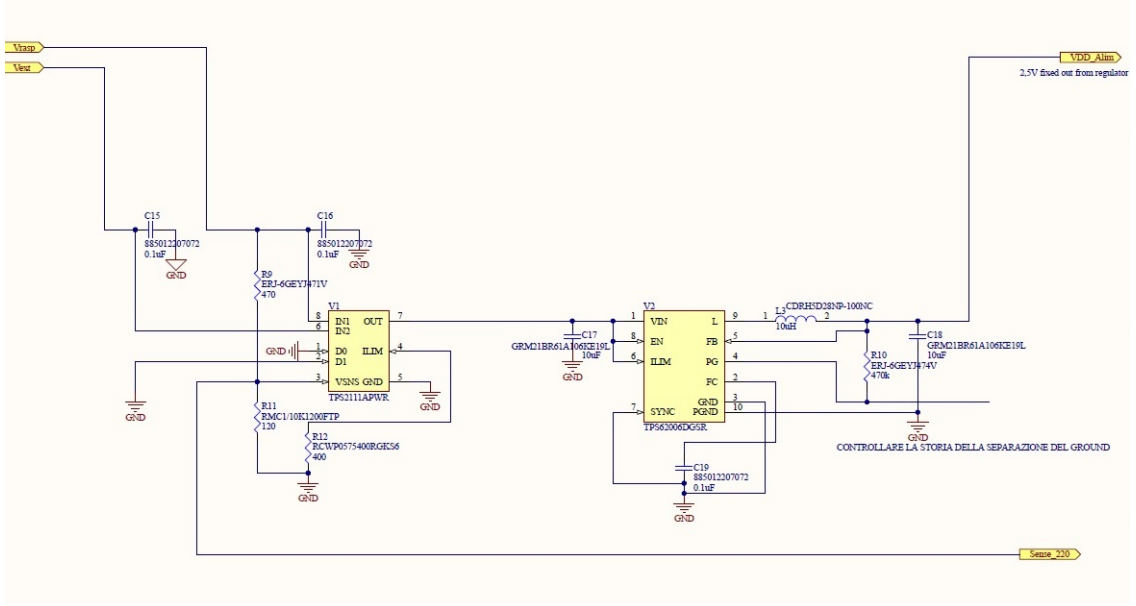


Figure 7.1. The schematic of Node power supply section.

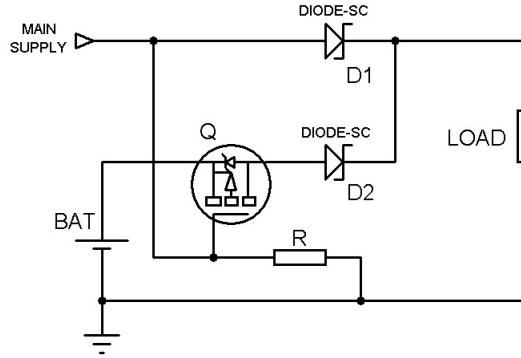
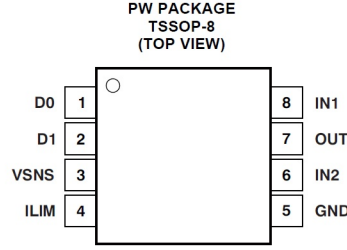


Figure 7.2. Simple circuit to handle backup power supply

one is down. From this theoretical idea was started the search of a more complex, integrated and versatile solution like the one explained in the following.

The chosen solution is an IC auto-switching power multiplexer (mux) that allow the transition between two power supplies, each operating in a range between 2.8 V to 5.5 V, and it can deliver up to 1 A. To better understand the schematic is necessary to introduce in Figure 7.3 the pin configuration description of the TPS2111A.

**Table 1. TERMINAL FUNCTIONS**

TERMINAL		I/O	DESCRIPTION
NAME	NO.		
D0	1	I	TTL- and CMOS-compatible input pins. Each pin has a 1- μ A pull-up. The Truth Table illustrates the functionality of D0 and D1.
D1	2	I	
GND	5	Power	Ground
IN1	8	I	Primary power switch input. The IN1 switch can be enabled only if the IN1 supply is above the UVLO threshold and at least one supply exceeds the internal V_{DD} UVLO.
IN2	6	I	Secondary power switch input. The IN2 switch can be enabled only if the IN2 supply is above the UVLO threshold and at least one supply exceeds the internal V_{DD} UVLO.
ILIM	4	I	A resistor (R_{ILIM}) from ILIM to GND sets the current limit I_L to $250/R_{ILIM}$ and $500/R_{ILIM}$ for the TPS2110A and TPS2111A, respectively.
OUT	7	O	Power switch output
VSNS	3	I	In the auto-switching mode ($D0 = 1$, $D1 = 0$), an internal power FET connects OUT to IN1 if the VSNS voltage is greater than 0.8 V. Otherwise, the FET connects OUT to the higher of IN1 and IN2. The Truth Table illustrates the functionality of VSNS.

Figure 7.3. The pin configuration of TPS2111A from datasheet.

Besides advantages to have only one integrated device that perform the needed task, the TPS2111A offers many other advantages: thermal protection, inrush current control, seamless supply transition, cross-conduction blocking and reverse conduction blocking. The most interesting features of this component is that it can operate in two mode: auto-switching and manual-switching. To decide the operating mode is necessary a hardware configuration of D0 and D1 pins. For this application it was chosen the auto-switching mode following the indications reported in the true table shown in the Figure 7.4.

According to the truth table, D0 was linked to high value (in this case V_{main}) through a resistor, to limit the sink current, and D1 to the ground. If the voltage on pin VSNS is less than 0.8 V the mux selects as output the greater input between IN1 and IN2 (in this case the V_{bat}). Instead if the voltage of VSNS is greater than 0.8 V, the mux selects as output IN1 (in this case V_{main}).

At node A from Figure 7.5, are connected two resistors of a voltage divider to set the threshold that discriminate the decision between the two power supply. This is to guarantee that if V_{main} reaches 4 V the voltage in A is 0.8 V, in other words the control of the supply is managed by the level of a primary supply: if it drop under 4V the mux switch to the other supply voltage.

Single inductor buck-boost - TPS63051

The second main device is a single inductor buck-boost. This device is a voltage regulator that perform an auto switch between buck and boost mode depending on input voltage. The input range voltage is 2.5 V to 5.5 V. Considering the maximum GPIO V_{OH} of CC1310, in relation to power supply, to be able to drive the necessary peripherals (like led, relay, the INA sensing etc.) and all the devices present on the board, the power supply of the whole system was set to 3.3 V. To guarantee a stable 3.3 V, having as input a wide range of voltage supplies (5 V from the power grid and 3 V for the battery), it was necessary to use a buck-boost DC-DC converter. For this reason, it was chosen the TPS63050.

To better understand the schematic is necessary to introduce the pin-out description of the TPS63050, reported in Figure 7.6.

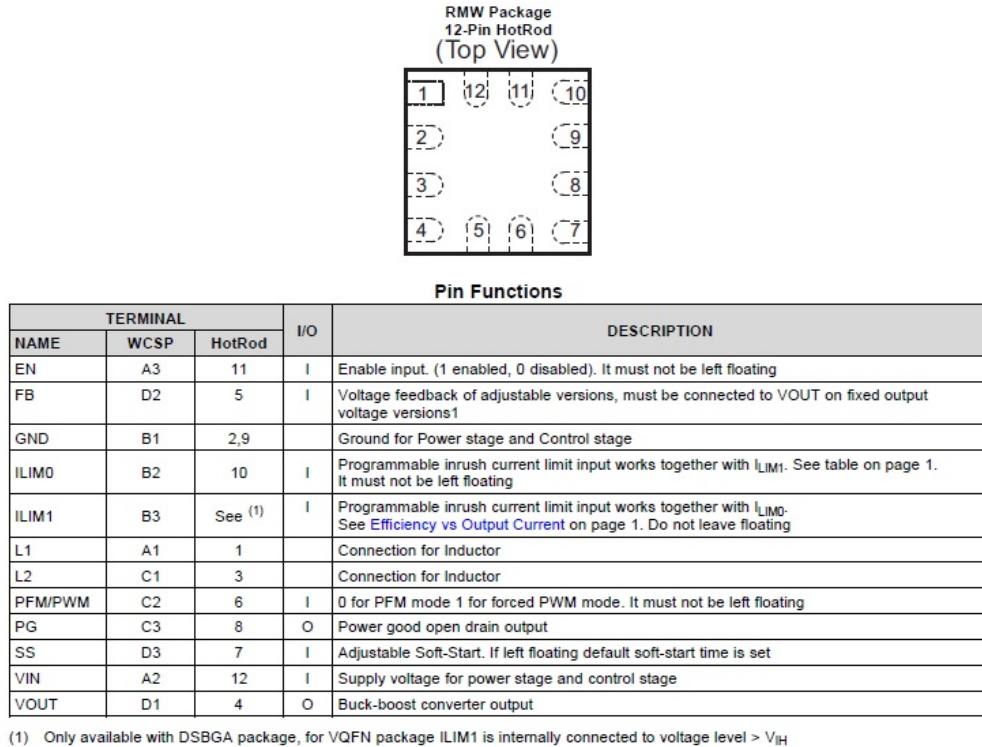


Figure 7.6. The pin configuration of TPS63050 from datasheet.

This device offers some interesting advantages: more than 90% of efficiency in boost mode and 95% in buck mode, buck-boost transition without interruption, soft start, load disconnection during shut-down and over temperature protection.

In the Figure 7.7 is shown the schematic of the DC-DC converter with its components. These components were chosen according to the TPS63050 datasheet [28].

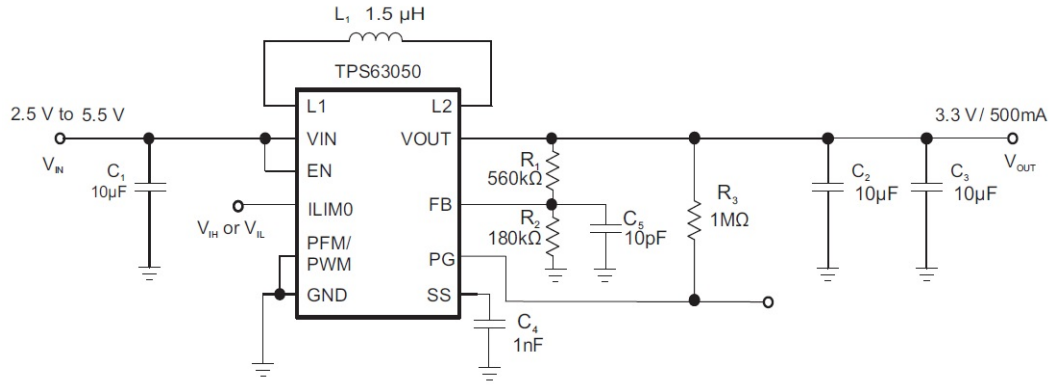


Figure 7.7. The schematic of TPS63050 from datasheet.

7.1.2 CC1310

In this section are described the CC1310 hardware design. In Figure 7.8 were reported all the components around CC1310 including: oscillators, radio antenna connector, V_{DD} conditioning, reset circuit and balun. Every connection decision and every consideration from now on widely uses the concepts exposed on the hardware explanation section of the CC1310 in 5.1

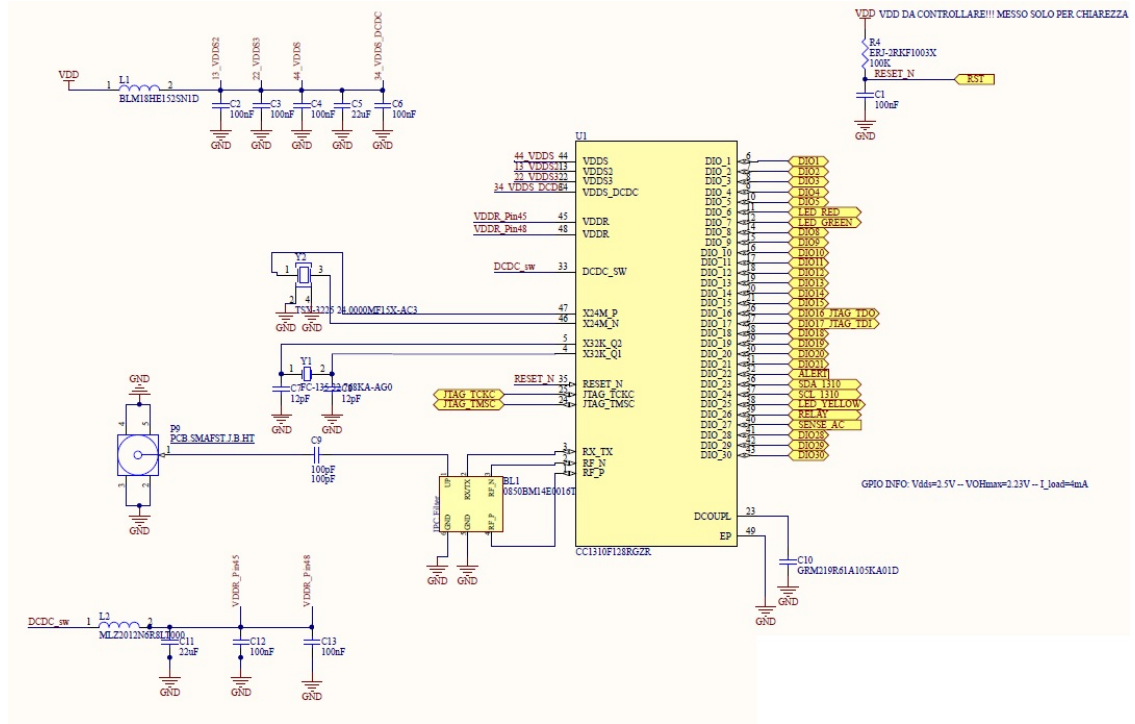


Figure 7.8. The schematic of radio part and all the component around CC1310.

Balun - Johanson 0850BM14E0016

The balun (balanced-unbalanced) is a LC circuit used to convert a balanced signal (two signals working against each other where ground is irrelevant) to unbalanced signal (a single signal working against ground or pseudo-ground) and it is fundamental to allow the transceiver to transmit and receive signals. In other words, it is necessary to adapt the RF signal in output from CC1310 (pin1 positive and pin2 negative) with the signal provided to the antenna. In this project, to improve stability, to avoid the presence of about 11 discrete components (Inductors and

capacitors) and to avoid problems concerning the routing it was decided to introduce, as balun, the IC 0850BM14E0016 Sub-GHz impedance matched balun of the Johanson Technology [29]. As shown in Figure 7.9, all the LC circuit is integrated inside the Johanson component.

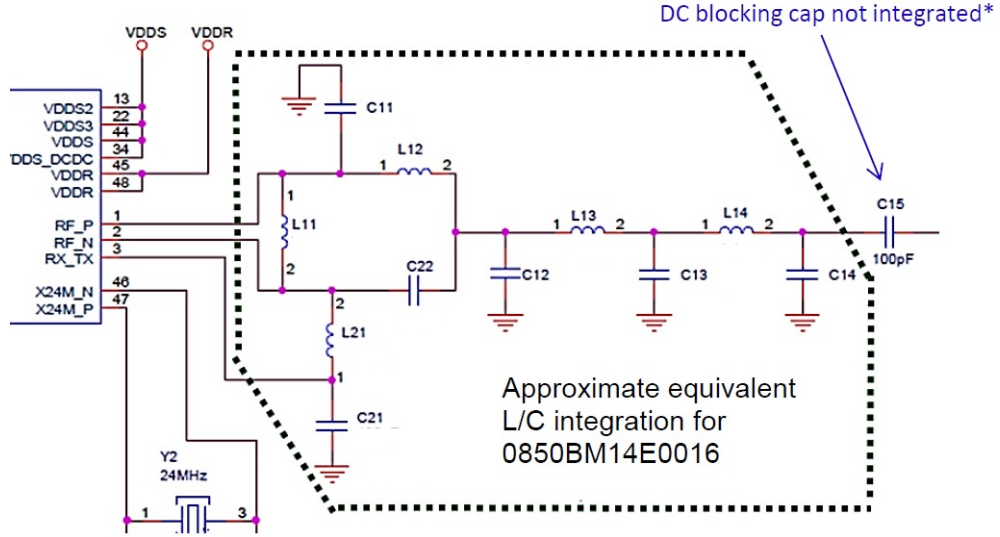


Figure 7.9. Equivalent LC circuit integrated of a Sub-GHz impedance matched balun from datasheet.

In the Figure 7.9 is reported the pin-out of the Johanson's Technology balun.

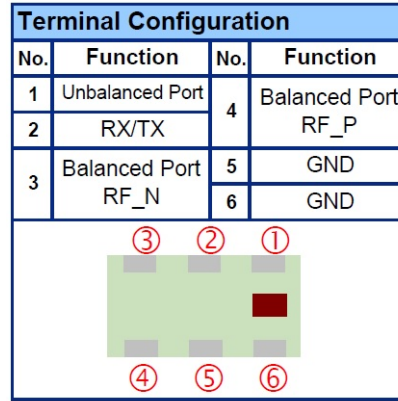


Figure 7.10. The pin-out of the Johanson Technology's balun from datasheet.

Following the pin-out table, pin2, pin3 and pin4 are connected to the respective

CC1310 pins and pin1 is connected to the antenna connector (Teoglas Antenna Solution Female SMA PCB Antenna Jack).

Crystal Oscillators

The CC1310 has two integrated crystal oscillators circuits and Texas Instrument, in the Crystal Oscillator and Crystal Selection guideline document [30], gives the possibility to the designer to choose a low cost quartz crystal to be externally connect to the correct pins of CC1310 (pin4, pin5, pin46 and pin47). The system uses one high-frequency oscillator at 24 MHz and a low-frequency one, at 32.768 kHz. The first is used to generate the reference clock for the Rf blocks and for the microcontroller unit. It is very important the precision of this clocks, to guarantee the correct operations of the whole system (correct and not degraded RF performances, the correct regulatory requirements, etc.). The second clock is used to manage some operations during low-power mode, when the high-frequency oscillator is turned-off.

The two selected crystal unit are the Seiko Epson Corporation, in particular the TSX-3225 24.0000MF15X-AC3 for the 24 MHz [31] and the FC-135 32.7680KA-AG0 for the 32.768 kHz [32].

In the Figure 7.11 is reported the pin-out of the two crystal quartz oscillators to better understand the schematic of this section, shown in Figure 7.8.

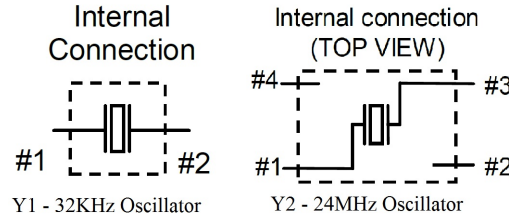


Figure 7.11. The pin-out of the two crystal quartz oscillator from datasheet.

Power supply pins

The CC1310 have six power supply pins that are pin13, pin22, pin34, pin44, pin45 and pin48. The output of the TPS63050 of the power management section (Figure 7.1) is directly taken through a LC conditioning circuit to supply the following internal subsection of CC1310 via the following pins:

- VDDS_DCDC 34: supply pin for the internal DCDC

- VDDS3 22: DIO supply
- VDDS2 13: DIO supply
- VDDS 44: main chip supply

The other two power pins, VDDR and VDDR_RF, are powered with a 1.7 to 1.95 V supply taken from the output of the internal DCDC at pin33. The path from pin33 to VDDR and VDDR_RF pass through another LC conditioning circuit. It is also possible to supply these two pins with an external supply source but it is not the case for this project. Under the CC1310 there is a ground via layout, that in the schematic is represented by pin49 (EP,) that is not a real pin and it will be explained better in PCB design section REF.

Reset pin

The pin RESET_N (pin 35) needs an active-low reset signal. Since there is not an internal pull-up resistor, it was necessary to introduce it. For this project the reset signal is user-managed by the pressure of a reset button, that generate the signal needed for the reset. The pin was grounded through a capacitor, to give the correct slew rate to the reset signal. To be valid, the reset pin must remain low for at least 1 μ s, so the pressing of the button combined with the designed RC circuit with a $\tau = 1ms$ of time constant is sufficient to guarantee a sure reset. The circuit is shown in Figure 7.1 but the push button is present in Figure 7.17.

7.1.3 Load monitoring

In this section is explained the circuit that performs the monitoring of a generic DUT load referring, as shown in Figure 7.12, to the situation used to define the system. The general configuration of a monitored system is the one with a generic UPS (Uninterruptible Power Supply) that powers the DUT.

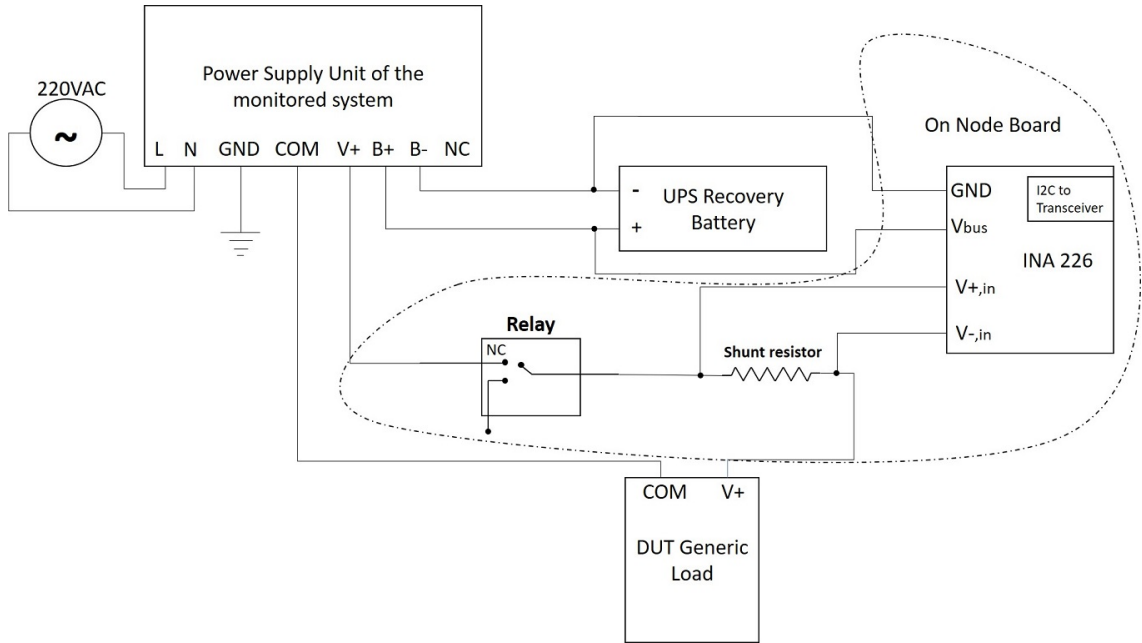


Figure 7.12. The functional scheme of load monitoring.

The two main devices of this section, shown in Figure 7.13, are: the Texas Instruments INA226 bi-directional current and power monitoring and the HY1Z-3V Panasonic relay.

Current shunt and power monitoring IC with I2C interface - INA226

The INA226 is a current shunt and power monitoring IC with I2C interface. The device measures the voltage of a line (from 0 V to 36 V) and a shunt voltage drop (from -81.92 V from 81.92 V). Thanks to the possibility to program parameters like conversions time, calibration value, averaging and internal multiplier, the INA226 gives back the measures of current in amperes and power in watts. This device can operate in a wide range of temperatures from -40°C to 125°C . Its supply voltage range is 2.7 V to 5.5 V and it absorbs 330 μA . The INA226 can operate in two modes: continuous and triggered. It is possible to choose between them in real

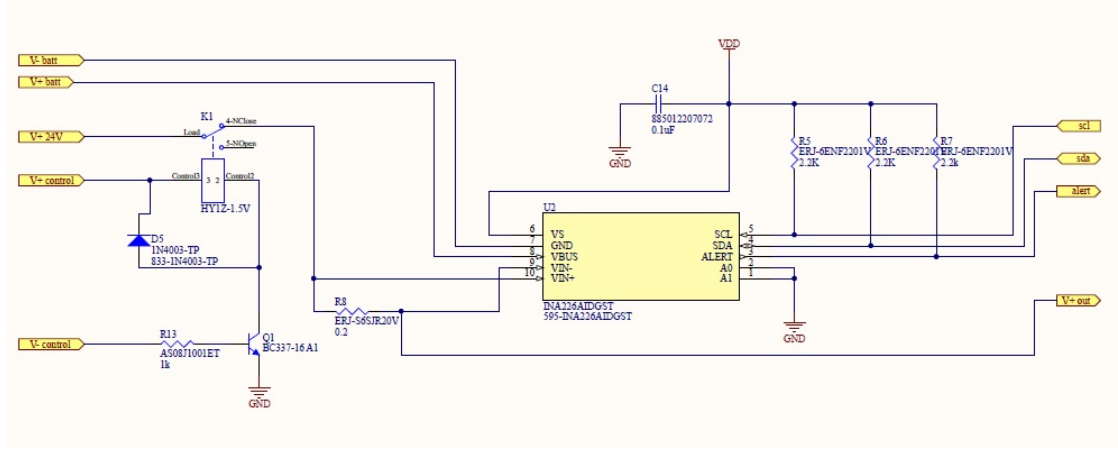


Figure 7.13. The schematic of load monitoring with INA226 and the relay HY1Z-3V.

time, according to the specific application needed. The choice is done by sending a specific setting command through I2C.

To better understand the schematic is necessary to introduce the pin configuration of the INA226 shown in Figure 7.14.

The two pins A0 and A1 have the function to set the serial bus address of the INA226. To choose the wanted slave address is necessary to follow the indications on the address pin table presented in the datasheet [33]. They were set at ground to have as address: "1000000".

The INA226 can operate only as slave in I2C bus. To use the open drain SCA and SCL lines, it was necessary to introduce some pull-up resistors. In this project, considering power supply and the current limitations of the INA226, they were set at 2.2 k Ω .

This device can measure the current shunt in two ways, with respect to the load position: high-side and low-side, like shown in Figure 7.15. In this case was chosen a high-side current sensing for two main reasons: first of all, with high-side shunt it is possible to directly monitor the current delivered by the supplier to the load with the possibility to detect an eventual short circuit, second is that low-side sensing may introduce disturbances to the system load's ground potential.

Relay - Panasonic HY1Z-3V

The second main device present on the board is the Panasonic relay HY1Z-3V. This relay has the capability to manage a resistive load of 1 A at 30 V DC. This electrical parameters were chosen in relation with the family of the load devices

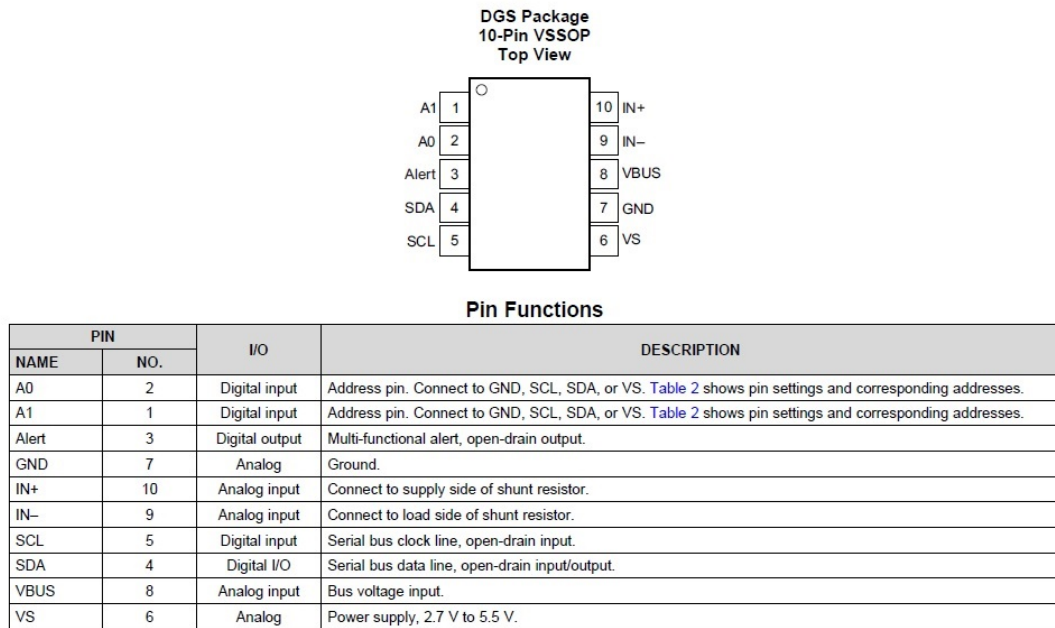


Figure 7.14. The pin configuration of INA226 from datasheet.

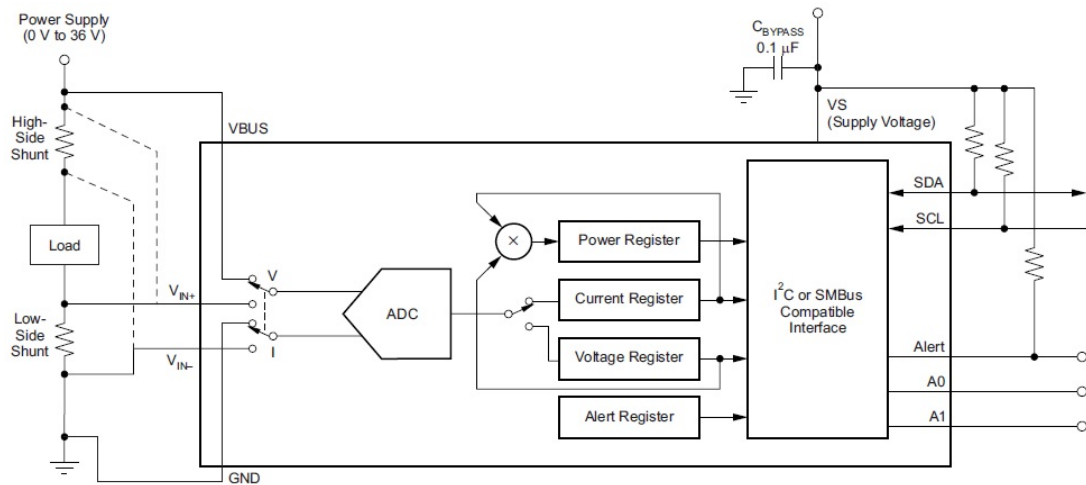


Figure 7.15. The schematic of INA226 from datasheet.

that will be used (for example a MikroTik router-board supplied at 24 V). It was chosen a normally close relay, instead a normally open, because the main necessity was to disconnect, for a short time, the load from power supply. This was the

main reason, but a secondary one was that the load has to remain connected to the power supply also if the board fails. In Figure 7.16 is reported the mechanical and electrical schematic of the Panasonic HY1Z-3V.

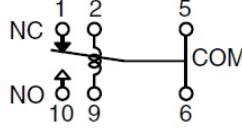


Figure 7.16. The internal schematic of Panasonic HY1Z-3V (bottom view) from datasheet.

The relay is controlled by a GPIO pin of CC1310 (pin 39) through a N-P-N transistor, the Taiwan Semiconductor BC337-16A1. It realizes a low-side switch, the best configuration for the usage of a BJT as switch. Obviously, the relay needs a free-wheeling diode to avoid eventual voltage spikes due to inductive effects produced by a fast current variation. So, it was added the diode (MCC 1N4003) in parallel to relay coil as shown in the the schematic in Figure 7.13.

7.1.4 The up level of the system

In Figure 7.17 is shown the schematic that include all the subsystems explained in the previous sections (Power management, Load monitoring and CC1310) and all the general component needed on the board, like LEDs and connectors.

LEDs

On the board are present four LEDs, all of them are Bival SMD LEDs with different colors (green, yellow and red). Three of them (D1, D2, D3) are used as state LEDs for the firmware operations, like described in the firmware section. This three LEDs are connected to GPIO pins 11, 12, 38 of CC1310 and they are active high. To guarantee a satisfying brightness, according to the led datasheet, were put some resistors to fix at 4 mA the sink current, and to limit it to acceptable values for the cc1310. The Voh provided by the GPIO for a load of maximum 4mA is about 2,75V (value related to the power supply and estimated, in the design phase, using data on the CC1310 datasheet[34]). So the chosen resistor value is 200 Ω . The fourth one, LED D6, is used as "power good" indicator, in fact is directly connected to the VDD plane and it turns on as soon as the exit of the DCDC give supply to the board.

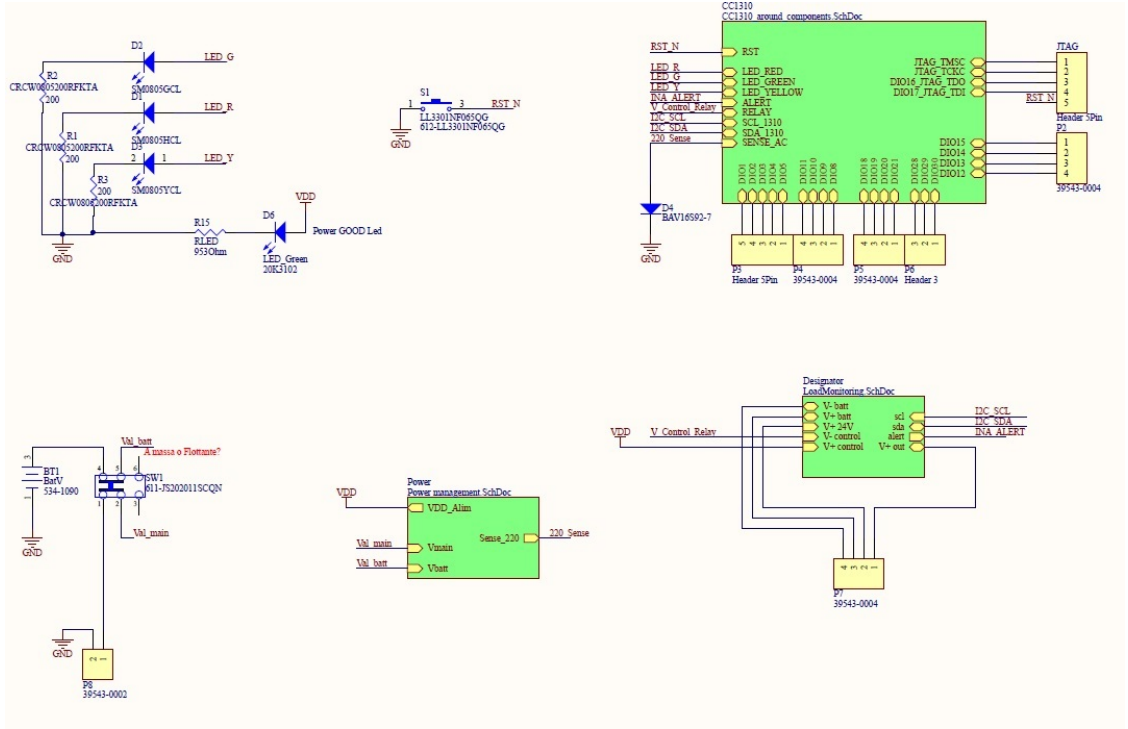


Figure 7.17. The up level schematic of the whole system of the node.

board ON/OFF switch

To turn on/off the board was put a SMD sub-miniature slide switch, the JS202011SCQN of C&K Components. This switch has two input, so it can connect two inputs to two outputs. This was necessary to manage the two power supply needed in the system, as explained in power supply section. Probably this component will be eliminated in a future realisation of the board. In the Figure 7.17 (down left) is clearly possible to see that one supply input come from the batteries and one come from a connector. Indeed, to make a more user friendly connection procedure, all the main external inputs can be connected to the board through some headers.

Connectors

For the main power supply and for the connection of the load where chosen two separated Molex connectors: one with two contacts (power supply connections) and one with four contacts (load connection). These two Molex have a 5.00 mm Eurostyle Pitch with a vertical fixed mount stile. The portion that holds the cable has a screw pressure clamp. Another important contact is the one for the JTAG

pins, used to program the uC. This contacts where made as planted true halls, to allow to insert the programming cables or to solder a header connector, as needed. Due to the fact that this board is also a development board, it was decided to expose all the GPIO contacts to allow any further experimentation and test. In a final version the unused pins will not be exposed.

7.2 Hardware design of the Master

The master board is the arbiter of the system. It has to communicate to the RaspberryPi through SPI to send the collected data to the users and has to send the request of the users to nodes. This board has less components respect to the node, but every common part presented the same problems that in the node. All the common hardware components were not be repeated in this chapter.

7.2.1 Power supply

The structure of the power supply chain of the master board must guarantee the possibility to power the system with two main voltage supplies: one from the power grid and one from the RaspberryPi. The structure is very similar to the node board but conceptually is different. In this case there is not an integrated UPS system (the battery) because the master is supposed to be installed in a non critical zone from the point of view of power supply. In other words the master board is located near an external UPS system. As the node, also for the master it was used an external AC-DC power supply to have 5 V on board.

It is present a generic 40 pins GPIO header that permit to interface with a SBC with a standard 40 pins header, in this case a Raspberry PI 2. Thanks to this header, it was possible to take the secondary voltage from the external board. In summary, the input of the mux TPS2111A 7.3 are the following: at the IN1 is connected the supply from RaspberryPi 2 and at IN2 the external AC-DC power supply. All the considerations about the voltage, devices, etc. are the same made for the node in section 7.1.1. The only difference is the absence of the pick point to sense the present of main power supply as shown in the highlighted red circle in Figure 7.18.

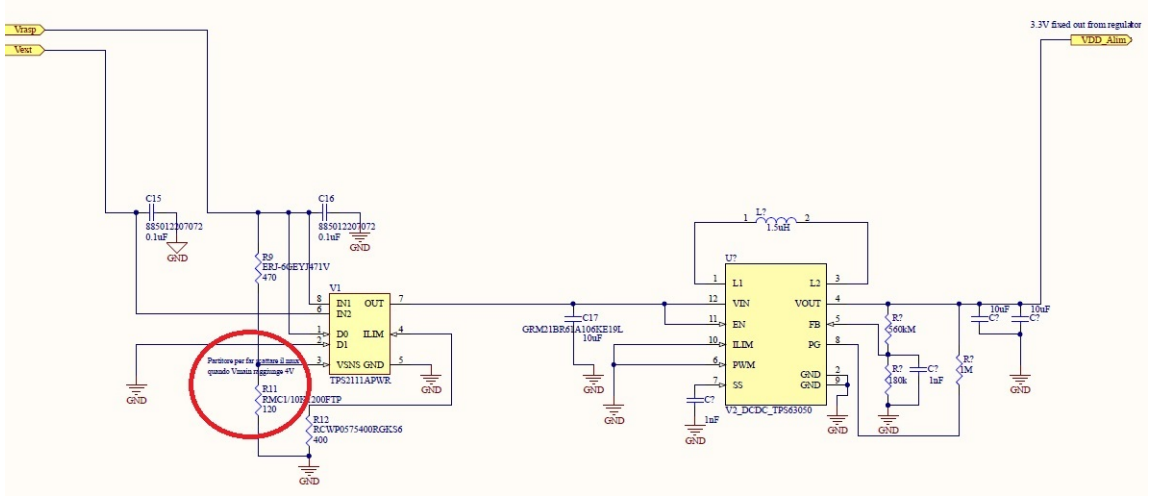


Figure 7.18. The schematic of Master power supply section.

7.2.2 CC1310

Both the boards have the same CC1310 section, so also for the master are valid the same considerations made in 7.1.2.

7.2.3 The up level of the system

In this section is shown the schematic that includes all the subsystem explained in the previous subsections (Power management and CC1310) and all the general components needed on the board, like LEDs and connectors, like shown in Figure 7.17.

The 40 pin header is used to connect the master board with the RaspberryPi that perform the on-line transmission of the data. It is possible to see in Figure 7.20 the pin-out of the Raspberry PI 2.

Starting from this configuration, it was pointed out the necessary pin for the needed operations. These pins are: the SPI pins, the I2C pins, the UART pins, the 5 V pins and the ground pins. Also here, it is present the on-off SMD sub-miniature slide switch, the JS202011SCQN of C&K Components. As input it receives the 5 V from Raspberry or 5 V from the electrical power grid.

In perspective to use these boards as a development kit, it was decided to introduce various SMD pads to facilitate the access to the various pins like I2C, SPI, all GPIO, and others.

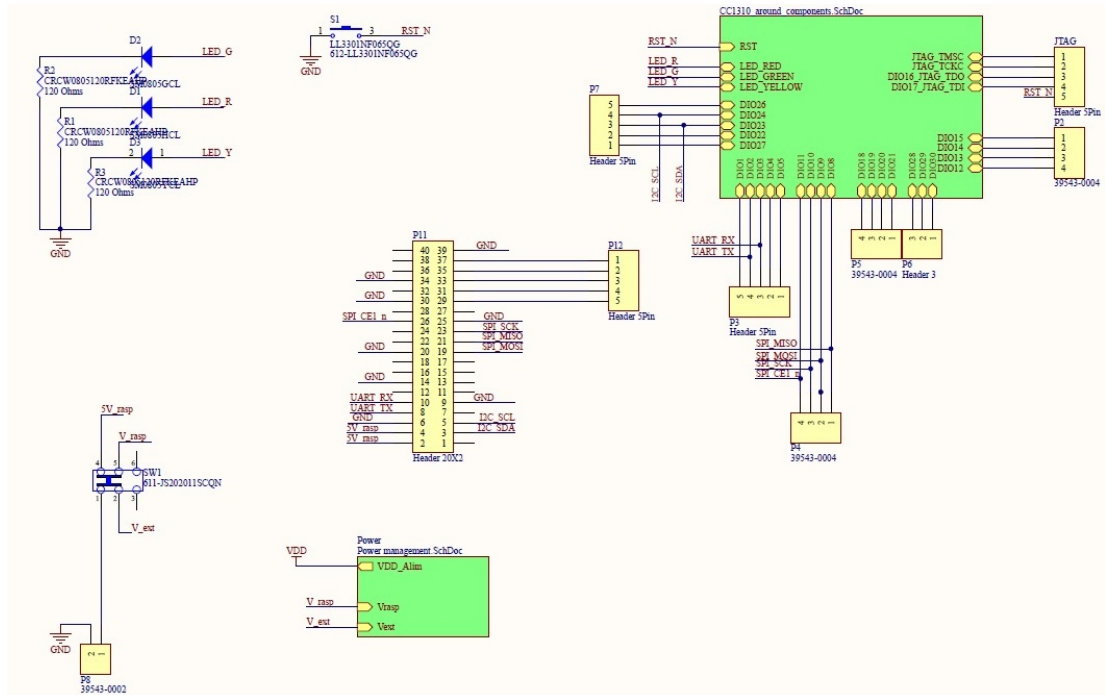


Figure 7.19. The up level schematic of the whole system of the master.

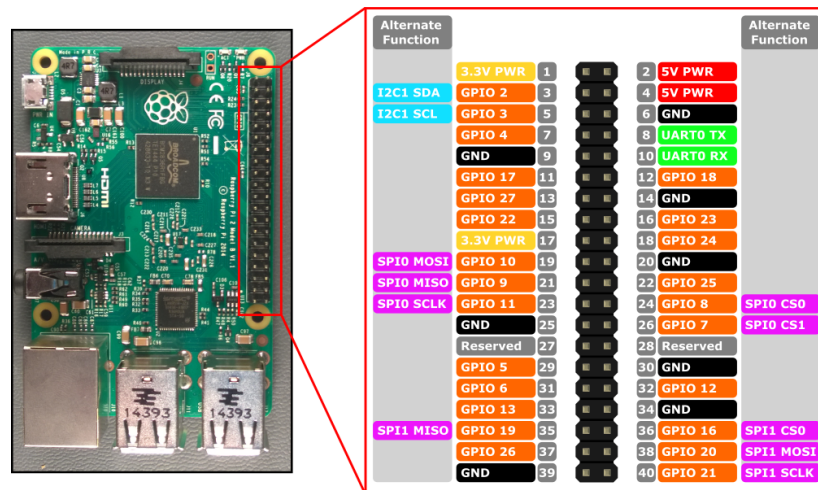


Figure 7.20. The pin-out of the Raspberry PI 2.

All the concepts about the reset, the power-on LED, the status LEDs and external connectors are the same designed for the node board.

Even if, both the boards have many common parts, every design choice was independently evaluated to guarantee the correct work of this system.

Every schematics previously described represent only the first step of the design: in reality, every component corresponds to a physical footprint with placing, routing and physical dimensions problems, that will exhaustively described in chapter 8.

7.3 Bill of Materials (BOM)

To resume all the components chosen for this system are reported, for the master first and then for the node, the final BOM generated by Altium through the Output Job Files. The BOM file is an important tool for the designer that allow to buy the correct quantity of the needed components from a supplier shop. In this case the chosen supplier was Mouser Electronics. During the realization of the component library, as explained in section 8.3, all the Supplier Part Number were inserted in Altium, among other, to create the BOM.

Comment	Description	Designator	Footprint	LibRef	Quantity
0850BM14E0016T	BALUN ipc filter johanson tec. for RF CC1310 fo RF pins	BL1	BALUN_FOOTPRINT_ SOP	FL2_BALUN	1
100nF	Capacitor 100nF, RESET, Capacitor 100nF, VDDS path, Capacitor 100nF, VDDS path, Capacitor 100nF, VDDS path, Capacitor 100nF, VDDS path, Capacitor 100nF, VDDS path	C1, C2, C3, C4, C12, C13	CAPACITOR_0402	C351_100nF, C131_100nF, C221_100nF, C441_100nF, C451_100nF, C481_100nF	6
22uF	Capacitor 22uF on VDDS path bw pin 44 & 34, Capacitor 22uF on DCDC path	C5, C11	CAPACITOR_0603	C341_22uF, C331_22uF	2
100nF	Capacitor 100nF VDDS path pin 34	C6	CAPACITOR_0603	C342_100nF	1
12pF	Capacitor 12pF, on oscillator Y1	C7, C8	CAPACITOR_0402	C51_12pF, C41_12pF	2
100pF	Capacitor 100pF, on RF path	C9	CAPACITOR_0603	C15_100pF	1
1uF	Capacitor 1uF, DC Coupl pin 23	C10	CAPACITOR_0402	C231_uF	1
885012207072	Capacitor 0.1uF, on input of V1 Power Mux, Capacitor 0.1uF, on input of V1 Power Mux, Capacitor 0.1uF, on V2 DCDC	C15, C16, C19	CAPACITOR_0805	C602_0.1uF, C601_0.1uF, C605_0.1uF	3
GRM21BR61A106KE 19L	Capacitor 10 uF, on input of V2 DCDC, Capacitor 10uF, on out V2 DCDC	C17, C18	CAPACITOR_0805	C603_10uF, C604_10uF	2
SM0805HCL	Typical RED LED	D1	LED_0805SMD_FOO TPRINT	LED_Red	1
SM0805GCL	GREEN LED	D2	LED_0805SMD_FOO TPRINT	LED_Green	1
SM0805YCL	Typical YELLOW LED	D3	LED_0805SMD_FOO TPRINT	LED_Yellow	1
Header 5Pin	Header, 5-Pin	JTAG, P3, P7, P12	HDR1X5	Header 5Pin	4
BLM18HE152SN1D	FILTER, EMI, 1500@100MHz on VDDS path	L1	FL1 EMI FILTER	FL1	1
MLZ2012N6R8LT000	Induttance 6.8uH on DCDC SW path	L2	INDC0805	L331_6.8uH	1
CDRH5D28NP- 100NC	Induttance 10uH on V2 output DCDC	L3	IND_POWER_SUMID A	L601_10uH	1
39543-0004	Header, 4-Pin	P2, P4, P5	HDR1X4	Header 4Pin	3

Figure 7.21. The Bill Of Materials of the Master Board part 1.

Comment	Description	Designator	Footprint	LibRef	Quantity
Header 3	Header, 3-Pin	P6	HDR1X3	Header 3	1
39543-0002	Header, 2-Pin	P8	MOLEX 2 - NO 45	Header 2Pin	1
PCB.SMAFST.J.B.HT	RF Coaxial PCB Connector, MMCX; Thru-Hole, Vertical Mount Plug, 50 Ohm Impedance	P9	MCX5.08-H5	COAX-F	1
Header 20X2	Header, 20-Pin, Dual row	P11	HDR2X20-PIN-CONNECTOR-RASP	Header 20X2	1
CRCW0805120RFKEAHP	Resistor 120 Ohm per i LEDs	R1, R2, R3	Resistor_6-0805_N	RLED_120	3
ERJ-2RKF1003X	Resistor 100K on RESETn	R4	RESISTOR 0402	R351_100K	1
ERJ-6GEYJ471V	Resistor 470 ohm, on input of power mux V1 TPS2111A	R9	RESISTOR_6-0805_N	R601_470	1
ERJ-6GEYJ474V	Resistor 470 Kohm, on output of DCDC V2 TPS62006	R10	RESISTOR_6-0805_N	R604_470k	1
RMC1/10K1200FTP	Resistor 120 ohm, on input of power mux V1 TPS2111A	R11	RESISTOR_6-0805_N	R602_120	1
RCWP0575400RGKS6	Resistor 400 ohm, on input of power mux V1 TPS2111A	R12	RESISTOR_6-0805_N	R603_400	1
LL3301NF065QG	Switch Push Button	S1	PUSH BUTTON FOOTPRINT	Push_Button	1
JS202011SCQN	Slide Switches to power on the system	SW1	SWITCH	SLIDE_SW	1
CC1310F128RGZR	CC1310 7x7	U1	RGZ0048A	CC1310F128RGZR	1
TPS2111APWR	Dual In / Single Out Autoswitching Power Mux, 2.8 to 5.5 V, -40 to 85 degC, 8-pin TSSOP (PW), Green (RoHS & no Sb/Br)	V1	V1_MUX_PW8_N	V1_MUX_TPS2111APW	1
TPS62006DGSR	Buck Step-Down Regulator with 2 V to 5.5 V Input and 2.5 V Output, -40 to 85 degC, 10-Pin MSOP (DGS), Green (RoHS & no Sb/Br)	V2	V2_DCDC_DGS10_N	V2_DCDC_TPS62006DGS	1
FC-135 32.768KA-AG0	32,7KHz Range Crystal Oscillator	Y1	3.5X2.8X1.9	Y1_32KHz_Oscillator	1
TSX-3225 24.0000MF	24MHz Oscillator	Y2	Y2_OSCILLATOR	Y2_24MHz_Oscillator	1

Figure 7.22. The Bill Of Materials of the Master Board part 2.

Comment	Description	Designator	Footprint	LibRef	Quantity
0850BM14E0016T	BALUN ipc filter johanson tec. for RF CC1310 fo RF pins	BL1	BALUN_FOOTPRINT_ SOP	FL2_BALUN	1
BatV	AA Battery	BT1	BATTERY HOLDER KEYSTONE	Battery	1
100nF	Capacitor 100nF, RESET, Capacitor 100nF, VDDS path, Capacitor 100nF, VDDS path, Capacitor 100nF, VDDS path, Capacitor 100nF, VDDS path, Capacitor 100nF, VDDS path	C1, C2, C3, C4, C12, C13	CAPACITOR_0402	C351_100nF, C131_100nF, C221_100nF, C441_100nF, C451_100nF, C481_100nF	6
22uF	Capacitor 22uF on VDDS path bw pin 44 & 34, Capacitor 22uF on DCDC path	C5, C11	CAPACITOR_0603	C341_22uF, C331_22uF	2
100nF	Capacitor 100nF VDDS path pin 34	C6	CAPACITOR_0603	C342_100nF	1
12pF	Capacitor 12pF, on oscillator Y1	C7, C8	CAPACITOR_0402	C51_12pF, C41_12pF	2
100pF	Capacitor 100pF, on RF path	C9	CAPACITOR_0603	C15_100pF	1
1uF	Capacitor 1uF, DC Coupl pin 23	C10	CAPACITOR_0402	C231_uF	1
885012207072	Capacitor 0.1uF on DCDC path, Capacitor 0.1uF, on input of V1 Power Mux, Capacitor 0.1uF, on input of V1 Power Mux	C14, C15, C16	CAPACITOR_0805	C701_0.1uF, C602_0.1uF, C601_0.1uF	3
GRM21BR61A106KE 19L	Capacitor 10 uF, on input of V2 DCDC, Capacitor 10 uF, on input of V2 DCDC, Capacitor 10uF, on out V2 DCDC	C17, C18, C19	Capacitor_0805	C603_10uF, C603_10uF, C604_10uF	3
VJ0603A100JXACW1 BC	Capacitor 10pF, on V2 DCDC	C20	Capacitor_0603	C607_10pF	1
06035F102K4Z4A	Capacitor 1nF, on V2 DCDC	C21	Capacitor_0603	C606_1nF	1
SM0805HCL	Typical RED LED	D1	LED_0805SMD_FOO TPRINT	LED_Red	1
SM0805GCL	GREEN LED	D2	LED_0805SMD_FOO TPRINT	LED_Green	1
SM0805YCL	Typical YELLOW LED	D3	LED_0805SMD_FOO TPRINT	LED_Yellow	1
BAV16S92-7	Silicon Switching Diode for High- Speed Switching	D4	DIODE_FOOTPRINT	Diode BAS16	1
1N4003-TP	Silicon Switching Diode for High- Speed Switching	D5	DIODE_RELAY	Diode Relay	1
LED_Green	GREEN LED	D6	LED_0805SMD_Foot print	LED_Green	1
Header 5Pin	Header, 5-Pin	JTAG, P3	HDR1X5	Header 5Pin	2
HY1Z-3V	Single-Pole Dual- Throw Relay	K1	Relay2	Relay-SPDT	1
BLM18HE152SN1D	FILTER, EMI, 1500@100MHz on VDDS path	L1	FL1 EMI FILTER	FL1	1
MLZ2012N6R8LT000	Induttance 6.8uH on DCDC SW path	L2	INDC0805	L331_6.8uH	1

Figure 7.23. The Bill Of Materials of the Node Board part 1.

Comment	Description	Designator	Footprint	LibRef	Quantity
CV201210-1R5K	Induttance	L3	INDC0805	L600_1.5uH	1
39543-0004	Header, 4-Pin	P2, P4, P5	HDR1X4	Header 4Pin	3
Header 3	Header, 3-Pin	P6	HDR1X3	Header 3	1
39543-0004	Header, 4-Pin	P7	MOLEX 4 - NO 45	Header 4Pin	1
39543-0002	Header, 2-Pin	P8	MOLEX 2 - NO 45	Header 2Pin	1
PCB.SMAFST.J.B.HT	RF Coaxial PCB Connector, MMCX; Thru-Hole, Vertical Mount Plug, 50 Ohm Impedance	P9	MCX5.08-H5	COAX-F	1
BC337-16 A1	NPN Bipolar Transistor	Q1	BJT-FOOTPRINTTO-226-AA	NPN BJT	1
CRCW0805200RFT A	Resistor 200 Ohm per i LEDs	R1, R2, R3	RESISTOR_6-0805_N	RLED_200	3
ERJ-2RKF1003X	Resistor 100K on RESETn	R4	RESISTOR 0402	R351_100K	1
ERJ-6GEYJ302V	Resistor 3K I2C pull up resistor	R5, R6, R7	RESISTOR_6-0805_N	R702_3K	3
ERJ-S6SJR20V	Resistor 0.2 ohm, R shunt for INA	R8	RESISTOR_6-0805_N	R701_200m	1
ERJ-6GEYJ471V	Resistor 470 ohm, on input of power mux V1 TPS2111A	R9	RESISTOR_6-0805_N	R601_470	1
ERJ-3GEYJ564V	Resistor 560Kohm	R10	RESISTOR 0603	R607_560K	1
RMC1/10K1200FTP	Resistor 120 ohm, on input of power mux V1 TPS2111A	R11	RESISTOR_6-0805_N	R602_120	1
RCWP0575400RGKS 6	Resistor 400 ohm, on input of power mux V1 TPS2111A	R12	RESISTOR_6-0805_N	R603_400	1
AS08J1001ET	Resistor 1K on relay BJT	R13	RESISTOR_6-0805_N	R705_1K	1
CRCW08056M04FKE A	Resistor 240 ohm, on input of power mux V1 TPS2111A	R14	Resistor_6-0805_N	R605_6M	1
RLED	Resistor 953 Ohm per i LEDs	R15	Resistor_6-0805_N	RLED	1
CRCW06031M00FKT A	Resistor 1Mohm	R16	RESISTOR 0603	R606_1M	1
ERJ-PA3F1803V	Resistor 180Kohm	R17	RESISTOR 0603	R608_180k	1
LL3301NF065QG	Switch Push Button	S1	PUSH BUTTON FOOTPRINT	Push_Button	1
JS202011SCQN	Slide Switches to power on the system	SW1	SWITCH	SLIDE_SW	1
CC1310F128RGZR	CC1310 7x7	U1	RGZ0048A	CC1310F128RGZR	1
INA226AIDGST	Bi-Directional Current and Power Monitor with I2C Compatible Interface	U2	INA226	INA226_DGS_10	1
V2_DCDC_TPS63050	Imported	U3	V2_DCDC_TPS63051	V2_DCDC_TPS63050	1
TPS2111APWR	Dual In / Single Out Autoswitching Power Mux, 2.8 to 5.5 V, -40 to 85 degC, 8-pin TSSOP (PW), Green (RoHS & no Sb/Br)	V1	V1_MUX_PW8_N	V1_MUX_TPS2111APW	1
FC-135 32.768KA-AG0	32,7KHz Range Crystal Oscillator	Y1	3.5X2.8X1.9	Y1_32KHz_Oscillator	1
TSX-3225 24.0000MF	24MHz Oscillator	Y2	Y2_OSCILLATOR	Y2_24MHz_Oscillator	1

Figure 7.24. The Bill Of Materials of the Node Board part 2.

Chapter 8

PCB realization

8.1 PCB stack manager properties

Starting the physical design of the PCB was necessary to face some issues about materials, stratigraphy, dimensions and rules to guarantee the correct realization of the printed circuit.

Different PCB stack structures

The first main choice, when proceeding to the realization of a PCB, was to choose how many layers to use in the stratigraphy of the device. In this case the choice was between 2-layers and 4-layers. In Figure 8.1 are shown the two generic standard structures.

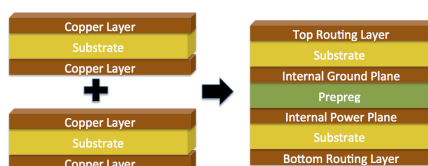


Figure 8.1. A generic standard 2 and 4 layer structure.

Generally, with a two layers board, it can be realized everything that is possible to realize with a multilayer structure, but the multilayer board has many improvements and technical advantages. Usually, the two internal layers of a 4-layers structure, are used as power plane and ground plane, providing easy routing. In this way, in addition to the possibility to create a more complex and dense design, a lot of electromagnetic compatibility problems are reduced. The plains usage

guarantees many advantages like: easy decoupling, separation between power path and signal path, increasing of thermal dissipation. To exploit all these advantages, and taking care of the presence of a RF path, the ground plane is placed under the top plane.

Stack structure

The organization of the layers started at an early stage of the project, due to the presence of a transceiver with a RF path, that has rigid design rules imposed by the producer (Texas Instruments in this case). The Texas instrument rules are shown in Figure 8.2.

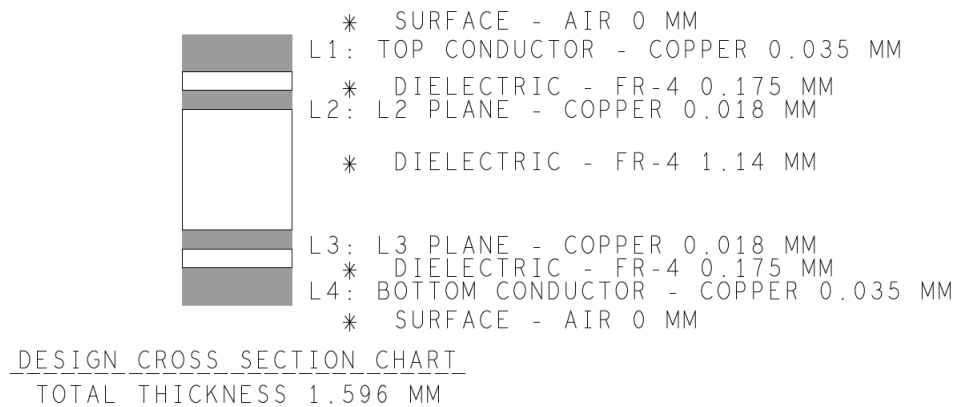


Figure 8.2. Texas Instruments design cross section chart of PCB stack for the CC1310.

It was strictly suggested to follow as much as possible these guidelines, so the 4-layers PCB was chosen.

To better understand the specifications it is useful to underline that a PCB is composed by two main structures: a substrate (the board) and printed layer that holds the wires (the copper traces). The substrate provides the solid structure that physically holds the components and the printed wires and provides electrical insulation between conductive parts. A common type of substrate is FR4, which is a fiber-glass– epoxy laminate. Substrates are made also by Teflon, ceramics and other special polymers [35].

Multilayer boards are composed by one or more substrate that can have copper on one or both sides. These substrates are called cores.

The cores are glued together thanks to one or more sheets of a partially cured epoxy called PREPREG. It is simple to recognize the described structure, in Figure 8.1 and in Figure 8.2 .

The order of GND and VDD plane is also fixed, because under the signal RF path must be present a ground plane to improve the quality of the RF signal as explained before.

To realize a RF 50 Ω matched line, it was very important to evaluate the ratio between the width of the line and its distance from the ground plane.

Stack properties

To decide the values of the various layers thickness, were taken into account both the specifications of the CC1310 layers stack and the specifications of the chosen manufacturer, Eurocircuit, that realized the two boards.

Unfortunately, the TI specifications and the one accepted by Eurocircuit didn't match at all. Eurocircuit gave, at a reasonable cost, different maximum and minimum specifications for layer thickness and hole diameters. Among the various possibilities offered by Eurocircuit, the most suitable one for this project, was the so called "standard pool PCB" [36]. This service covers most of the needs for standard PCB production.

In Figure 8.3 is reported the Eurocircuit cross section chart of the layers stack of a "standard pool" board.

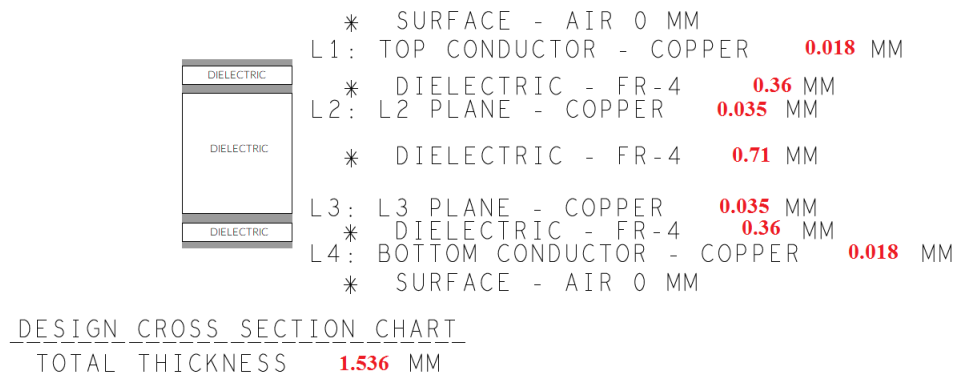


Figure 8.3. The Eurocircuit design cross section chart of layers stack for a "standard pool" board.

Analysing the given Eurocircuit specifications, with respect to the TI ones, many differences arose.

It was necessary to contact the TI technical customer service to discuss and evaluate if it was possible to use these different parameters of the stack layer and which kind of problems may occur. From the discussion with a TI technical employee, the following issues were identified:

- The type of substrate is a key parameter and fortunately it was respected
- The distance between L1 and L2 of Eurocircuit is wider than the TI one. These parameters is very critical, in fact increasing this distance the parasitic capacitance become smaller but the inductance of the ground vias becomes larger. This it changes the impedance the Rf path and therefore changes the performances. Thanks to a simulation, made by the TI technical customer service, the impedance variations is relatively small.
- The reduced copper thickness of the outer layer provided by Eurocircuit don't cause any problem, it may only reduces reliability.

After these analysis, the recalculation of the width of the RF path and the distance between the rounding mass plane was done. At the end of this adjustment process, was decided that using Eurocircuit parameters don't compromise the system functionality and the properly working of the transceiver was guaranteed.

In Figure 8.4 are listed the final values inserted in the layer stack manager of Altium Designer.

8.2 PCB Shape

The physical board layout realization, starting from the schematics project, faces some placement and routing problems.

First of all, it was necessary to evaluate the mechanical properties of the boards, taking into account the real working environment where they have to be installed. For this application, taking into account that the device has to work outside exposed to meteorological events, the usage of a case was necessary. It was chosen the IP56 Gewiss GW44205 case. In Figure 8.5 are shown the dimensions of the case.

To fit this case, the physical structure of the board has to be designed as shown in Figure 8.6. It was necessary to cut the four corners and to drill some fixing holes to fix the board at the case. To remain independent from the hole position on the case, the usage of a interfacing metallic bracket between the board and the case was introduced. The two boards have different fixing holes because the master board is suitable to support a tower-fashion mounting to guarantee the connection to an external SBC like the RaspberryPI. This solution may or may not be adopted

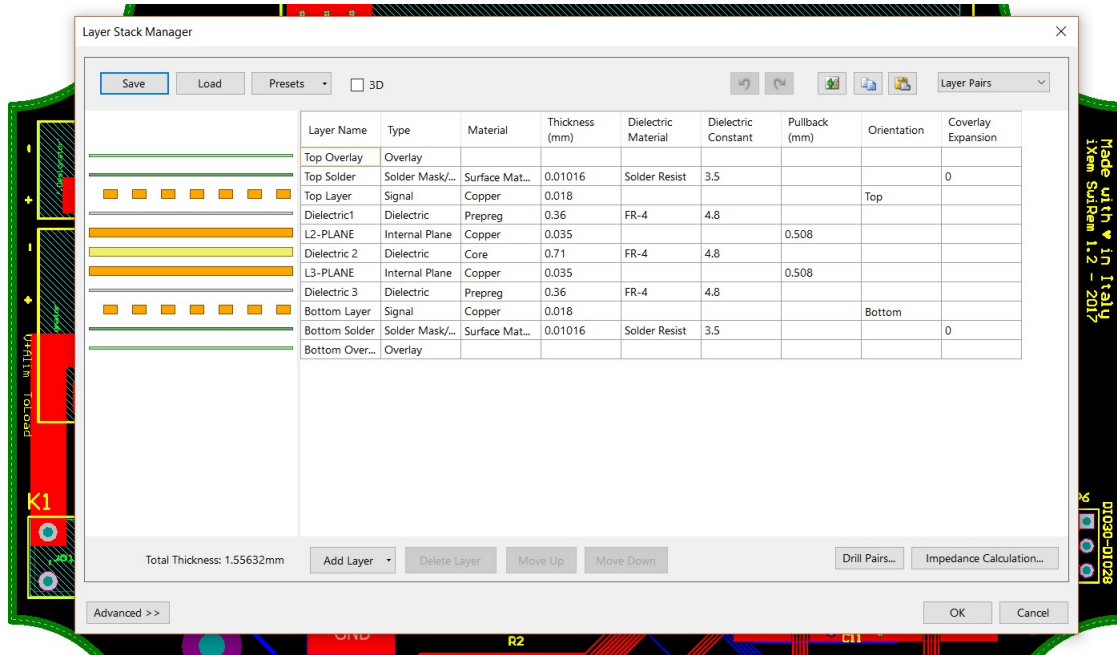


Figure 8.4. Altium layer stack manager.

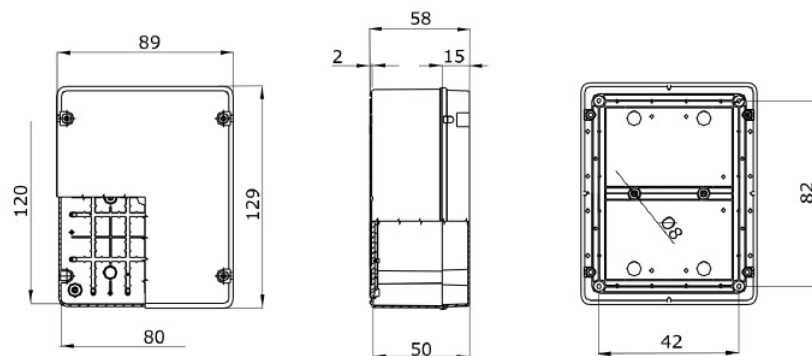


Figure 8.5. Dimensions of the Gewiss GW44205 case from datasheet.

and probably in future implementations will be discarded, but for this prototype this was a good choice to remain independent with respect to the specific case. The corners are rounded with a circle of 13 mm of radius.

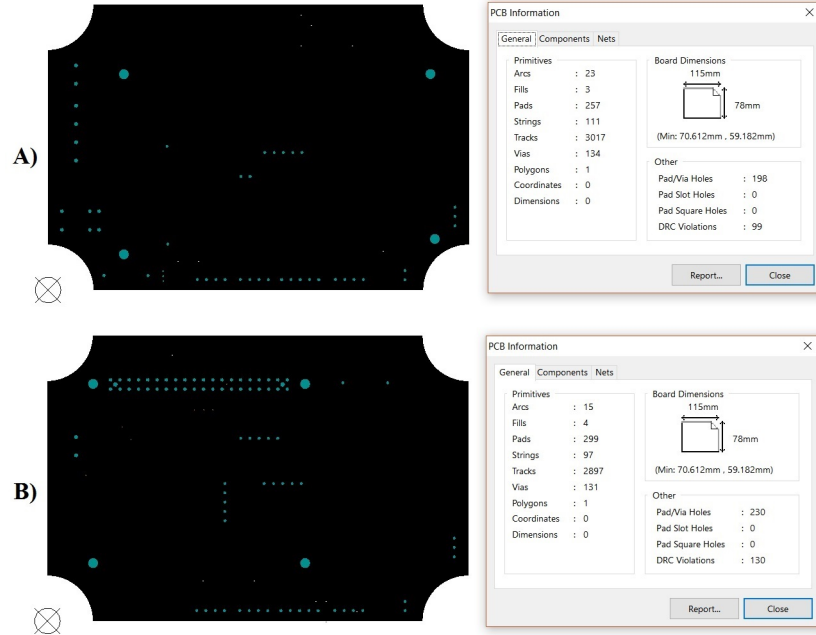


Figure 8.6. A) Shape of the node board. B) Shape of the master board.

8.3 Footprint and library

To develop a PCB project, using a CAD tool, is fundamental to use and eventually to create, a library that contains every components and device used. In Altium Designer, a library is composed mainly by two distinct sections: one for the schematic description and one for the footprint structure. Many components are provided, by the manufacturers, with the complete library structure, so the work of the designer is to check the correctness of the provided files and to adapt them to the specific needs. For example, to use the CC1310 footprint given by TI, it was necessary to download template file from the TI website and process it with the Ultra Librarian Software, that permit to extract the Altium footprint and the schematic files. It was also necessary to modify the footprint ground of the component because the original one had some errors. In Figure 8.7 is possible to see, in the left the Ultra Librarian interface and in the right the same footprint in Altium Designer.

To realize an exhaustive library, it was necessary to associate at each component the supplier part number. This procedure is to guarantee the correct association of all the component parameters through the Supplier Search Tool and to allow the creation of the BOM to buy the needed component.

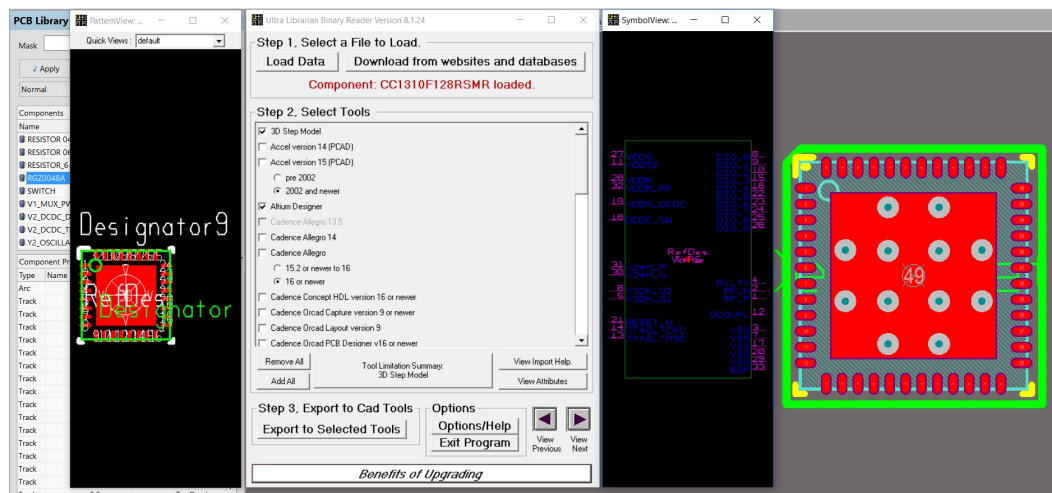


Figure 8.7. Footprint extracted by Ultra Librarian Tool.

8.4 Placement and routing guideline

Another important aspect of the layout realization, was the choice of where to collocate each component. Extremely important was to design precise footprints and to follow the routing guidelines, to ensure the proper working of the circuits and to minimize noise and EMC problems. To face the organization of the components on the top and bottom layer, the boards were divided in three main areas: the radio area, the power area and the sensors/actuators area.

CC1310 routing

Since that the CC1310 is one of the most critical device present on the boards, obviously the layout must respect several guidelines [8]:

- The area under the CC1310 is used for ground connection and shall be connected to the ground plane with several vias for good thermal performances and sufficiently low inductance to ground
- It is recommended not to place any digital trace below/near the crystal oscillator
- It is highly recommended to have a continuous and un-interrupted ground plane in any RF design

- A continuous ground plane underneath the top layer provides easy connection to the ground by allowing to drop vias from the pads to the ground. This also eliminates the need for any additional traces that would worsen the performance by increasing the inductance of the ground connections.
- On the top plane it is usually a good idea to fill the unused area with ground plane and then connect this top fill with the ground plane below with several vias
- To prevent high frequency noise from reaching the power supply pins every power connection of the CC1310 is bypassed to the ground plane using a capacitor, which provides a low impedance path to the high frequency noise.

RF routing

First of all, to design the radio section, it was necessary to reserve a correct and precise area where to put the transceiver and all the rounding components. This layout dimensions are strictly recommender on the TI guidelines. To maximize power transfer, for optimum performance, it was important that impedance matching is considered carefully in radio frequency designs. The main rules to take care of are the following:

- It is mandatory a solid ground plane
- Any signal traces have to be place underneath the RF path
- RF longer traces must have 50 Ω impedance
- Decoupling capacitor as close as possible to the CC1310

The balun has a precise structure, as shown in Figure 8.8, that must be precisely reproduced to avoid any disturbances and any phase alterations on the filter characteristics.

A good choice was to put far from RF area all possible source of noise, like the relay, transistors, DC-DC converters, LEDs. For this reason, the layout was organized to put all external connectors as far as possible from the transceiver area.

For delivering the maximum power out of the balun-filter section to the antenna the impedance of the RF trace was a critical factor. The impedance of a PCB trace at RF frequencies depends from: the thickness of the trace, its height above the ground plane, and the dielectric constant. All these parameters, for the PCB realization, were set in the *PCB rules and constraints editor* in Altium Design introducing value provided by Eurocircuit. In particular, as shown in Figure 8.9,

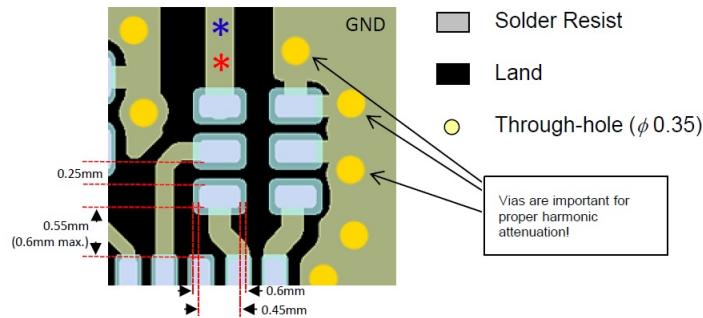
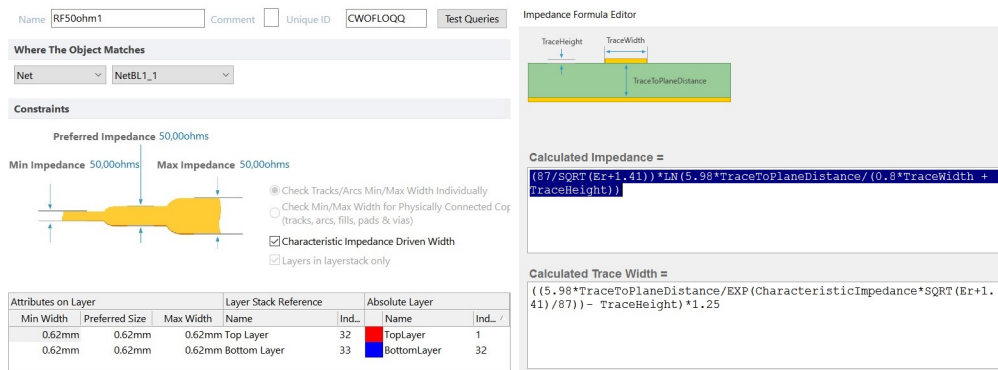


Figure 8.8. Guideline structure for the balun chip.

it is possible to read the rules that manage the trace width to maintain a fixed impedance of $50\ \Omega$. Every sensible device is surrounded by a ground plane on the surface of PCB that act as guard ring.

Figure 8.9. PCB realization rule of a $50\ \Omega$ trace.

INA226 routing

In Figure 8.10 is shown what must be done to the routing of this particular device. More in detail, it was important to avoid an additional resistance between the input pins, realizing a good routing. So, it must be used a 4-wire connection to ensure that only the current-sensing resistor impedance is detected between the input pins. It is also important to place the power-supply bypass capacitor as close as possible to the supply and ground pins.

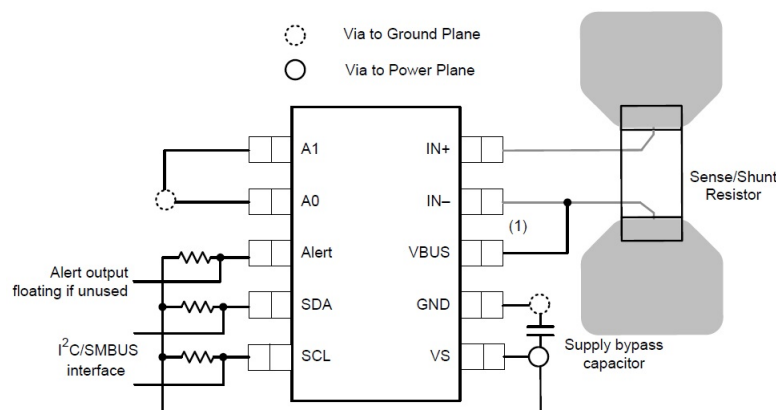


Figure 8.10. Guidelines routing of the INA226 from datasheet.

Power routing

The PCB layout is an important step to maintain the high performance of the TPS63051 device. Some precautions must be taken into account:

- Place input and output capacitors as close as possible to the IC
- Traces need to be kept short
- Routing wide and direct traces to the input and output capacitors results in low-trace impedance and in a low parasitic inductance
- Use a common-power GND
- The sense traces connected to FB must be kept away from L1 and L2 nodes
- A capacitor must be added between FB node and ground to filter ground noise and to match efficiency results

In Figure 8.11 is reported the layout suggested in the TI datasheet [28].

The traces that manage the power signals must be realized wider than the other traces, to support a high power.

8.5 Final PCB

Keeping in mind all the remarks explained in the previous sections, it was placed all the components on the board and it was done the manual (to optimize the

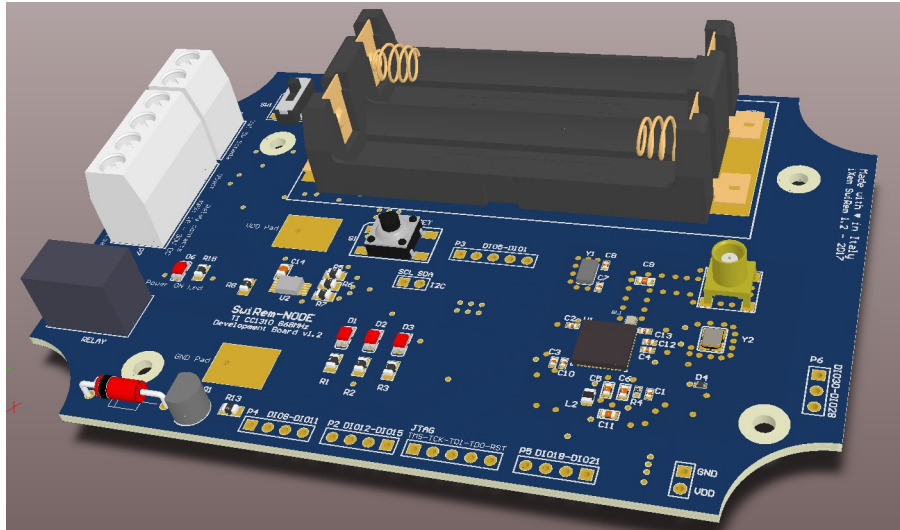


Figure 8.13. 3D model of the node.

8.5.2 Master

In Figure 8.14 is reported the Altium Designer layout of the node board and in figure Figure 8.15 is also reported the 3D model.

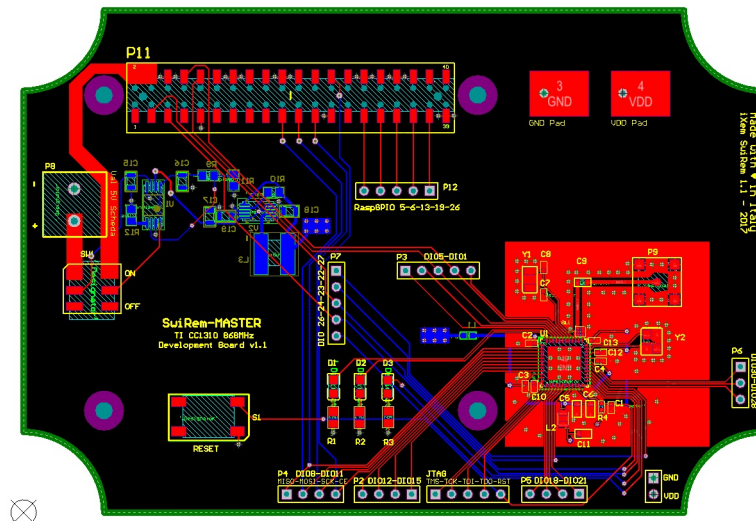


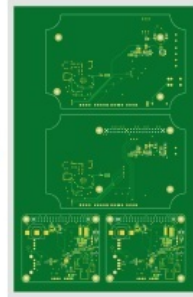
Figure 8.14. Layout of the node.

PCB images

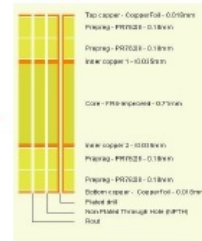
Top view :



Bottom view:

**Buildup & Mechanical plan**

Board buildup

**Board definition**

Number of layers	4
PCB width (X)	0 mm
eC-registration compatible	No

Delivery format	Customer, according to customer specific panel rules
PCB height (Y)	0 mm

Panel definition

Panel width	150.00 mm
Horizontal (X) PCB separation method	Breakrouting
Panel without cross outs	No

Panel height	237.50 mm
Vertical (Y) PCB separation method	Breakrouting
Panel outline	Routed

Board definition

Top soldermask	Green
Top legend	White
Surface finish	Che Ni/Au selective
Bare board testing	Yes

Bottom soldermask	Green
Bottom legend	White

Board technology

Pattern class	6
Outer layer trackwidth (OL-TW)	0.150 mm
Outer layer isolation distance (OL-TT-TP-PP)	0.150 mm
Outer layer annular ring (OAR)	0.125 mm
Inner layer trackwidth (IL-TW)	0.150 mm
Inner layer isolation distance (IL-TT-TP-PP)	0.250 mm
Inner layer annular ring (IAR)	0.125 mm

Drill class	Drill D
Hole density	<1000/dm2
Holes <= may be reduced	0.45 mm

Material definition

Board thickness	1.55 mm
Base material	FR4IMP
Outer layer copper foil	18µm(End-Cu +/-35µm)
Extra PTH runs	0
Reversed buildup	No

Board buildup	Standard buildup
Material Tg	145-150°C
Inner layer copper foil	35µm
Extra press cycles	0
Inner layer core thickness	Standard

Advanced options

Copper up to board edge	No
Specific tolerances	No
Press-fit holes	No
Round-edge plating	No

Plated holes on the board edge	No
Specific marking	No
Depth routing	No
Chamfered mechanical holes	No

Pricing**Printed circuits**

Basket nr.	Delivery term	Quantity	Unit price	Transport price	Transport mode	Total price	VAT	Gross
80932800	7 Working days	2	144.34 EUR	3.68 EUR	Express	292.36 EUR	0.0 %	292.36 EUR

Figure 8.16. The Eurocircuit final document.

Chapter 9

Assembly and debug

9.1 Assembly

In this section is explained everything about the procedures used to mount all the components on the PCB. In the Figure 9.1 is shown the panel received from Eurocircuit.

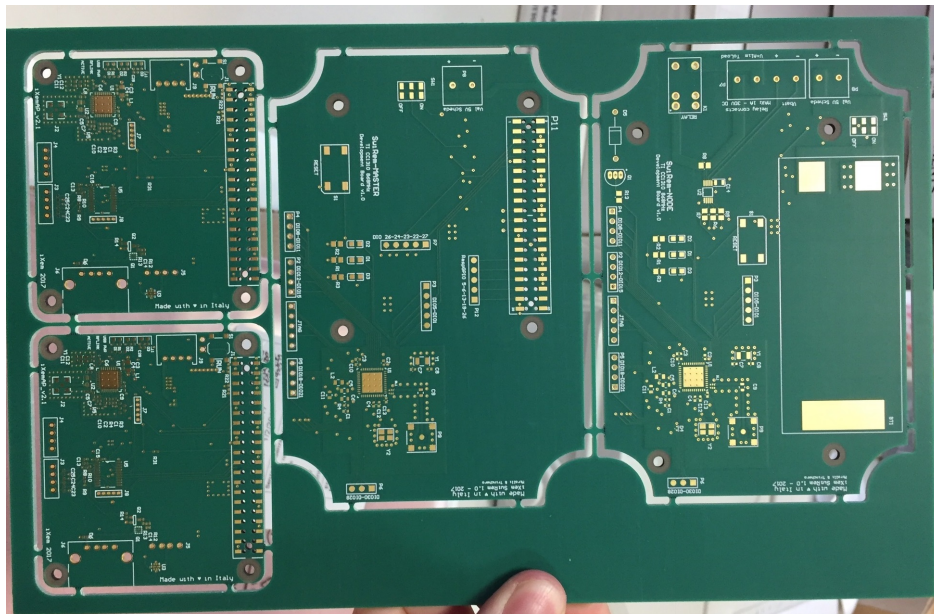


Figure 9.1. The final panel from Eurocircuit.

Before illustrating the welding process, it is important to list the materials and the equipments used.

More in detail:

- Low temperature Solder Past - Sn42Bi57.6Ag0.4 No-Clean T5
- High temperature Solder Past - Sn96.5Ag3.0Cu0.5 No-Clean T5
- Needle for solder past
- Pneumatic pick and place - Essemtec
- Microscope - i-Tronik
- Pneumatic solder past dispenser - Clever-Dispenser-04
- Reflow soldering oven - Infrared IC Heater T-962A
- Professional soldering station - JBC DDE 2 tools

After separating the various boards from the panel and from each other, as first step of the soldering process, it was necessary to put on the gold pads of the PCB the adequate quantity of solder past, contained in a syringe shown in Figure 9.2. To dispense the solder past with a pneumatic dispenser, the correct needles have to be chosen. The needles have to be chosen in relation of the pads dimensions and consequently, in relation to the physical characteristic of the solder past.



Figure 9.2. The syringe ready for the pneumatic dispenser

When there is the need to populate a PCB board with two side is mandatory to use two different solder pasts: one that solidifies at high temperature (for the first baking process) and the second one that solidifies at lower temperature for the second baking process. In this way, with the second baking process, it is avoided that the first soldered components unstick.

After depositing the solder paste on all the pads, was necessary to place all the components on the board thanks to the pneumatic pick and place machine, as shown in Figure 9.3.

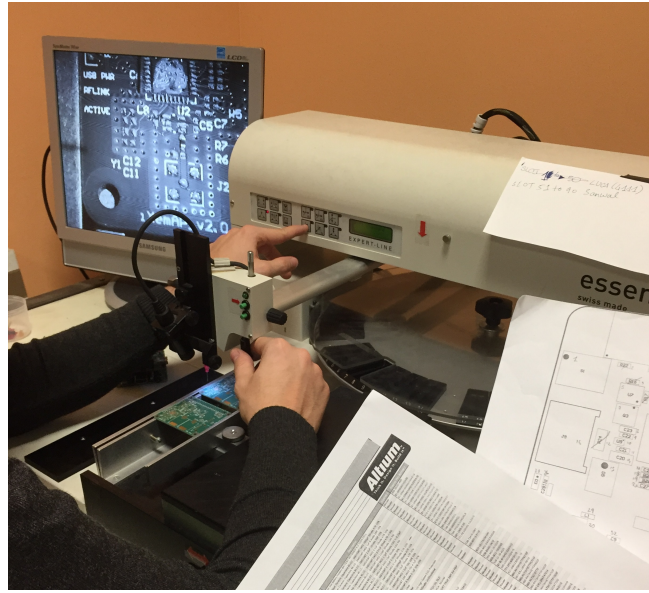


Figure 9.3. The pick and place at work.

When all the components on one side of the board were placed, it was possible to proceed with the first baking process of the PCB in the oven. To guarantee the correct soldering it was necessary to choose the correct baking profile in relation to the solder past used (the first one is always the one at higher temperature).

All the steps above are repeated for the second side of the PCB board with the low temperature solder past. The last passages was the manual polishing thanks to a specific solvent and a brush.

Some components, in particular the true-hole, were manually soldered after the baking process by means of a professional soldering station.

At this stage arose a problem about the value of the true hole diameter of some components, in particular the holes of Molex connectors were too tight. To resolve this problem it was necessary to manually drill the PCB to enlarge the holes. Obviously this error was immediately corrected in the Altium PCB project for the future physical realization.

In the Figure 9.4 are shown the two complete boards (version 1.0) ready to be tested, with all the components mounted above. In the Figure 9.5 is possible to see the node board in a Gewiss case.

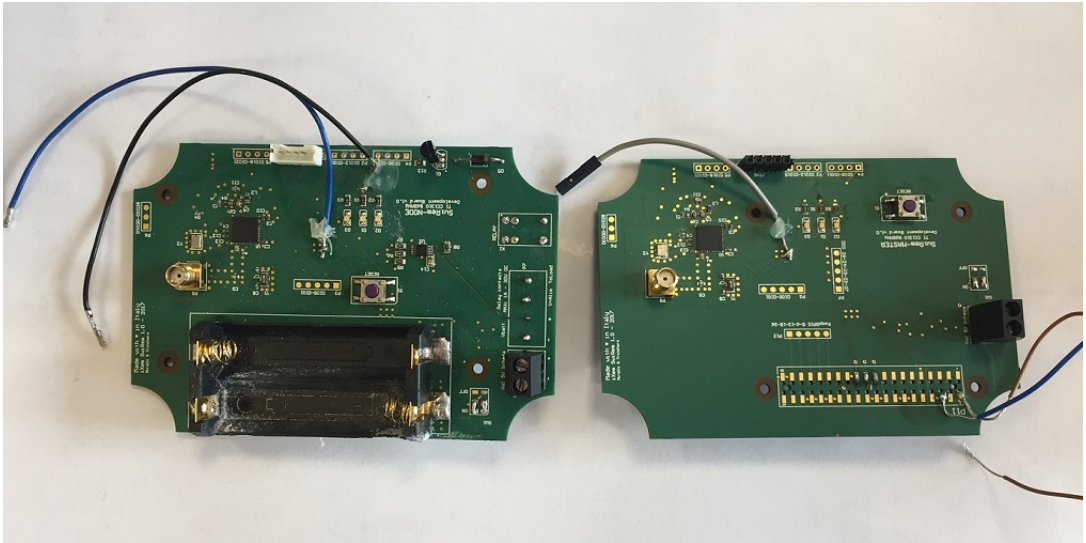


Figure 9.4. Left: the final Master Board v. 1.0. - Right: the final Node Board v. 1.0.

In conclusion, it is well known that there are many industries that provide, to the designer, soldered and tested PCB boards, with precision and robustness higher than the one reached with the procedure just described. For this first prototype it was decided to make all the steps in laboratory for both a didactic and practical purpose.

9.2 Debug

The first type of debug done was the visual control of the solders with the help of the electronic microscope, to verify the alignment of the various components leads with the under pads and the uniform distribution of the solder. In some cases it was necessary to de-soldering and re-soldering manually a component because the previous solder was unsuccessful.

Successively it was necessary to control some connections with a digital multi-meter, without giving power supply, to control the physical short circuit between two points.

Afterwards, it was possible to verify the presence of the correct voltage in some strategic points, like the network of the power supply. Thanks to this measures on the power supply network, a problem was founded on the control pins of the Auto Switching Multiplexer.



Figure 9.5. A sample of the node board inside the Gewiss Case

In detail, it was set the state value wrongly, putting to zero voltage the configuration pin D0 instead of VDD, like explained in 7.1.1.

The previous measurements have been repeated many times before reaching the correct functionality, from the electrical point of view.

After the preliminary testing and before proceeding to the cross hardware-software testing, it was necessary to update the Altium PCB project adding some pads, for Ground and VDD, to facility the connection of the testing measure instruments.

Obviously, all these mistakes are present in PCB version 1.0 (the one explained in this chapter) but in the Hardware Design and PCB realization chapters (chapter 7 and 8) all the explanations refer to the correct version 1.1 of the boards.

After solving all the hardware bugs, it was possible to load the firmware into the TI CC1310 by means of the Texas Instruments Programmer XDS100V3 on the SmartRF06 Board with the JTAG connector for both the boards. The connection used between programmer and SwiRem board is shown in Figure 9.6.

Thanks to Code Composer Studio (CCS), leaving connected the SwiRem board

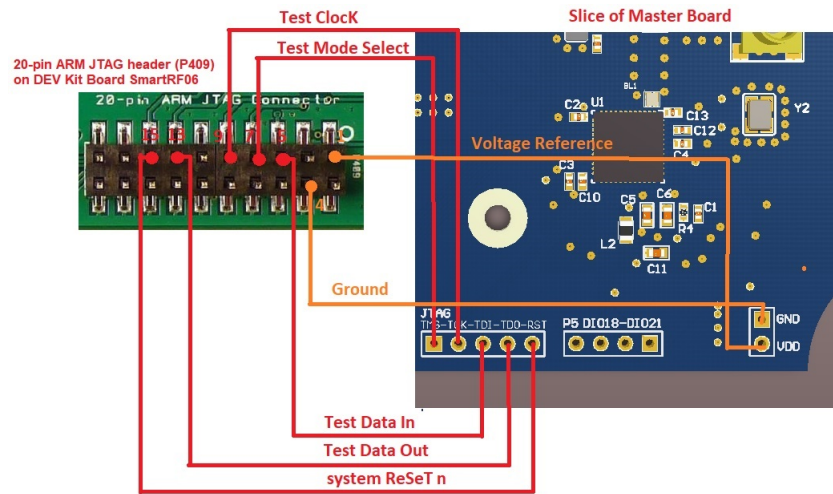


Figure 9.6. Clear connection scheme between the TI XDS100V3 programmer on the SmartRF06 Dev Board and SwiRem boards.

to the Debugger Module, it was possible to control the success of the firmware loading and its proper functioning. It was also possible to make a true debug, very useful to understand various firmware mistakes. After few changes, it was possible to make sure that the hardware responds properly to the software instruction.

Now the design process is completed, the two boards correctly handle with the tasks for what they were designed for.

Chapter 10

Further improvements and system evolution

Due to the high innovation rate of IoT technologies it was decided to improve the system using the new structure based on LoRa Alliance products, that include hardware, firmware and server solutions. It is a very different approach respect to the WSN TI CC1310 based because:

- It is necessary to design only the Nodes.
- As base station is possible to use the Sentiur RG1XX LoRa Enabled Gateway from Laird
- The Laird Gateway realize a private LoRaWAN network for end-to-end node control.
- The possibility to use a cloud service called "The Things Network" that allow easily link nodes distributed in long distances exploiting public Gateways already installed on the territory.
- LoRa Murata CMWX1ZZABZ has better communication and noise rejection behaviour certified by various iXem tests.

10.1 LoRa Alliance "The Things Network" services

The Things Network is a community that has as mission the realization of a secure and redundant collaborative LoRa WAN Internet of things network to improve the

proliferation of new IoT application. The network use only LoRa WAN technology without using 3G or WiFi, to link the various distributed "things".

This system provides some important tools as firmware APIs, Server applications and hardware products to allow any certified LoRa WAN devices to use long range Gateway and to have access to a decentralized network. This infrastructure allow the exchange of data with applications.

To exploit all the advantages of The Things Network with the SwiRem system, the previous design was converted to use the new LoRa Murata transceiver. In this thesis work the firmware for this product was not implanted yet. The iXem team developed the firmware for a LoRa WAN Node based on the Murata and the server management.

10.2 SwiRem LoRa Murata Version

For the SwiRem LoRa Murata board was necessary to reimplement all the radio part around the new transceiver. Deeply analysing the datasheet [37], in collaboration with other iXem teams, all the necessary components were selected and introduced in Eagle CAD to create the schematic sheet, as shown in Figure 10.1

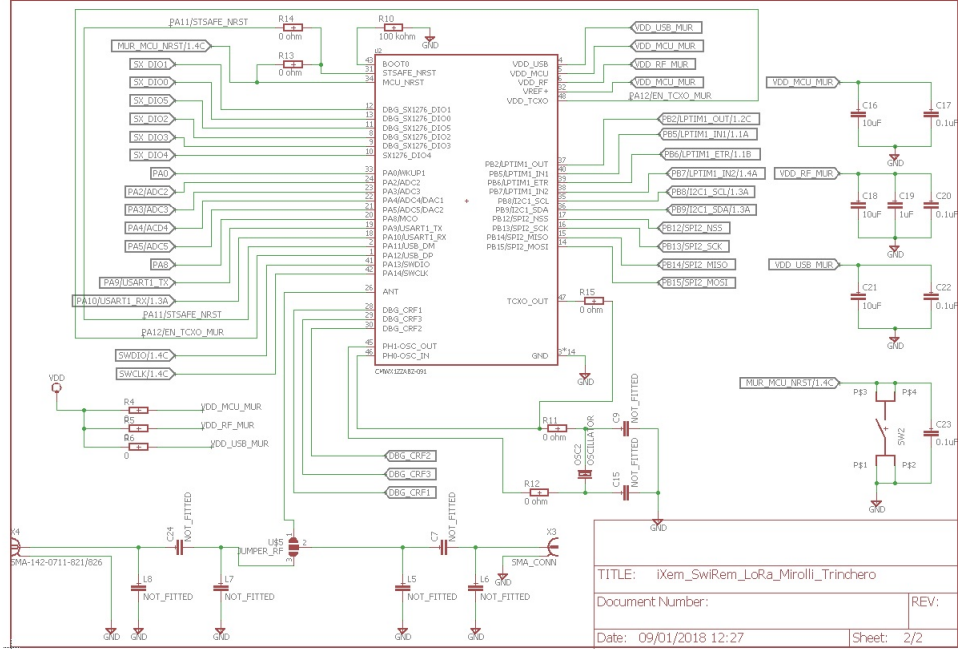


Figure 10.1. Eagle CAD schematic of LoRa Murata Transceiver.

To make compatible all the previous actuators and sensors, it was necessary to choose the suitable pins of the new transceiver and design a new routing for the connections to guarantee a functional design, as shown in Figure 10.2

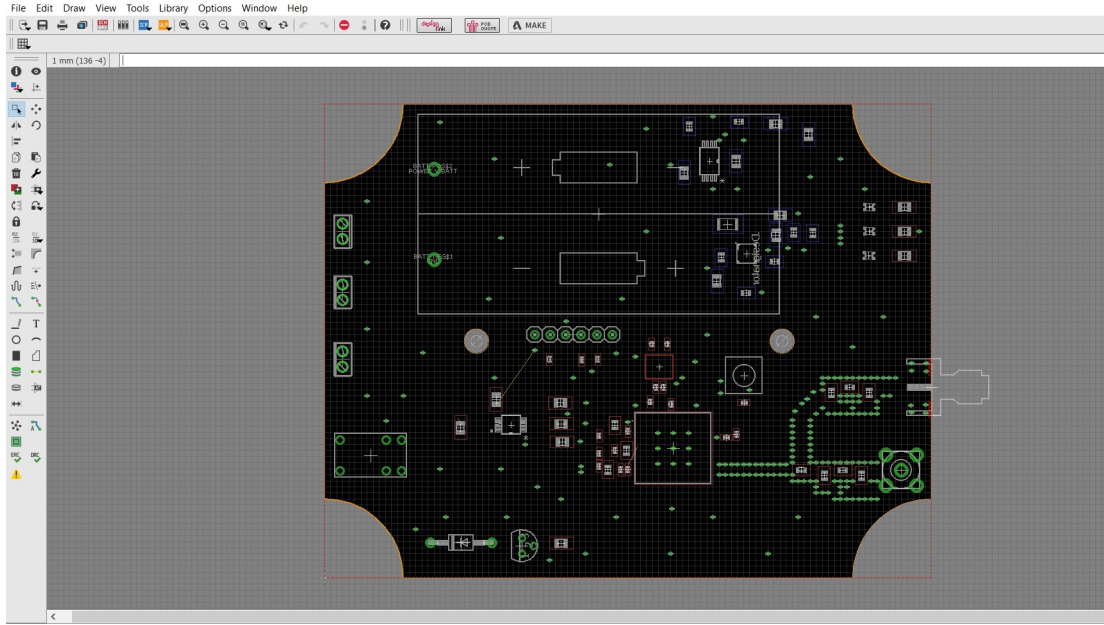


Figure 10.2. Eagle CAD SwiRem Board

This board is not printed yet because is in a development phase and the RF hardware it was tested on another iXem board, to be sure about the radio design.

In Figure 10.3 is possible to see, on a PCB done in collaboration with another iXem team, the realization of the radio part (so not in a SwiRem version, that is not printed yet, but it has almost the same radio design). The photo is only a portion because the entire board is still a project under development.

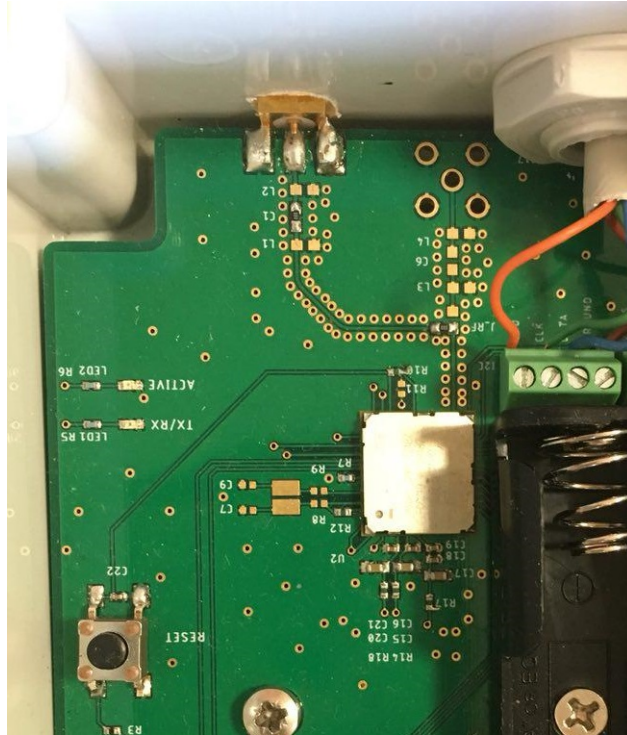


Figure 10.3. Radio design on a iXem PCB

Chapter 11

Testing

When the prototype was ready to use, after software and hardware debug, the most important work was testing the functionalities of the devices and of the whole system. These analysis were essential to understand the correctness of the system, to verify the presence of HW or SW bugs and to test the performances. The tests where done first of all in a controlled environment and then in a real environment.

11.1 Controlled environment

As first approach, the preliminaries tests were done in a controlled environment, inside Politecnico di Torino, to verify the correct behaviour of the system without taking into account any performance parameters. The configuration used is the one shown in the Figure 11.1.

The test was successfull, the implemented radio protocol took place properly, when the node received from the master the command to switch the relay it correctly turn on or off the connected generic load, that in this case was a MikroTik routerboard. Furthermore, every time the node was asked to send back data to the master, the communication was done correctly, completing the planned data exchange cycle. When the master received the packet data, it was possible to collect it through SPI thanks to Bus Pirate universal serial interface [38], and see them through a serial monitor on the PC. During another test, done between the two bridge of Politecnico, with a distance of about 190 meters, the RSSI of the radio link was evaluated. It was possible to enrich the configuration replacing the Bus Pirate Universal Serial Interface with the Raspberry PI (with a proper running script) like shown in the Figure 11.2.

Also in this test the exchange of data between master and node and the commands execution took place correctly . In the following table 11.1 are reported the

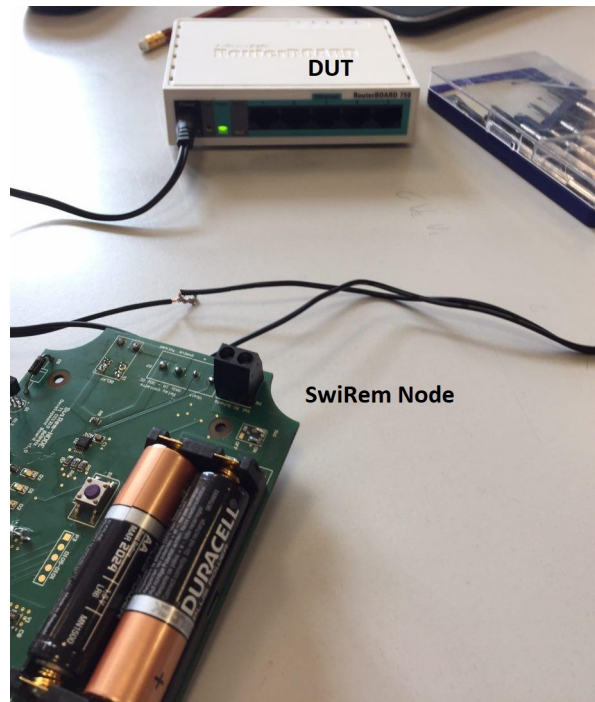


Figure 11.1. The configuration used to test the system in controlled environment.

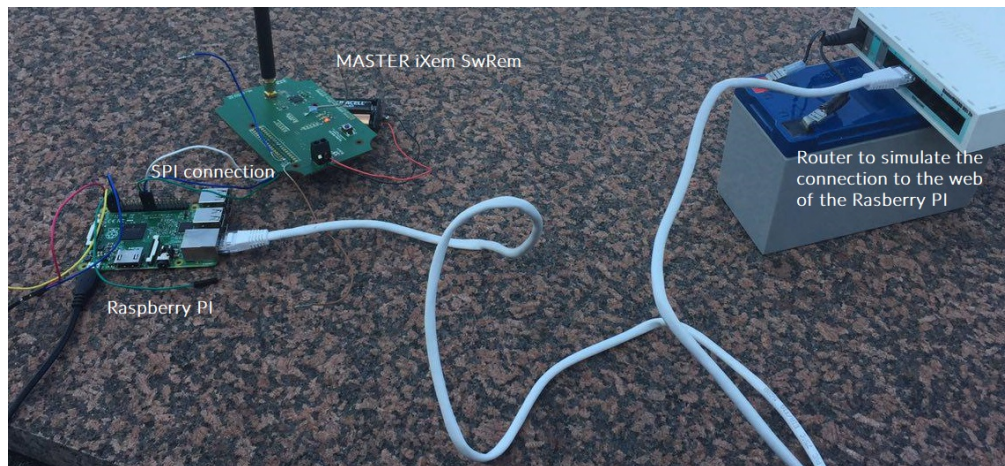


Figure 11.2. The configuration used to test the system in controlled environment with the Raspberry.

measures taken in this context.

The transmission parameters were the following:

1)

SwiRem: 50Kbps - 2-GFSK - 14dBm Tx power

TI Dev Kit: 50Kbps - 2-GFSK - 14dBm Tx power

Wi-MOD LoRa Module: LoRa Mod - SF:12 - BW:125Khz - EC 4/5 - 20dBm Tx power

2)

SwiRem: 625bps - LoRa - 14dBm Tx power

TI Dev Kit: 625bps - LoRa - 14dBm Tx power

Wi-MOD LoRa Module: LoRa Mod - SF:7 - BW:125Khz - EC 4/5 - 20dBm Tx power

N°	RSSI WiMOD [dBm]	RSSI CC1310[dBm]	RSSI SwRem [dBm]	Friis [dBm]
1	-62	-70	-73	-59
2	-63	-70	-74	-59

Table 11.1. RSSI Measures of SwiRem, CC1310 Dev Kit and LoRa between Politecnico bridges

The RSSI measured values differ from the RSSI calculate with the Friis Equation (-59 dBm with 14 dBm of Tx Power and -52 dBm with 20 dBm of Tx Power) because Friis gives a theoretical value without taking into account many parameters like the altitude of the antenna respect to the floor, that influences a lot the performance of a radio transmission.

11.2 Real environment

To understand the reachable performances and to better understand the behaviour of the system it was necessary to do some tests in an environment the most similar to the one where the system will be used. This test contest is one of the main situation where was necessary to work in team, because it was necessary to place in different locations the two board and taking data on their behaviour. To approach an environment the most similar to the real one, it was chosen as testing area the Santuario di Crea at Serralunga di Crea (AL).

In this frame were realized six measures, increasing progressively the distance between the two boards. These measures were done in the same conditions and in the same places with respect to the ones done with the development kit of CC1310 and WiMOD Module. To take these measures, the organization was the following:

the master was attached to the PC (through Bus Pirate universal serial interface) in a fixed geographical position, on the bell tower of the "Santuario Diocesano di Crea" on the top of a hill. Then the node was put in various points with a crescendo of distance, around the Sanctuary. For every measure the main focus was to understand if the communication between master and node took place, in detail the master sent the command to turn on or off the relay to the node, as explained before, and the evidence of the correct communication was the relay commutation. Then, to evaluate performances, the RSSI of the radio link was measured and compared with the one measured with the Texas Instrument Development Kit and with the one of the WiMOD Module.

In the following Figures, from 11.3 to 11.8, are shown the geographical points (named from 1 to 6) where the measurements were taken.

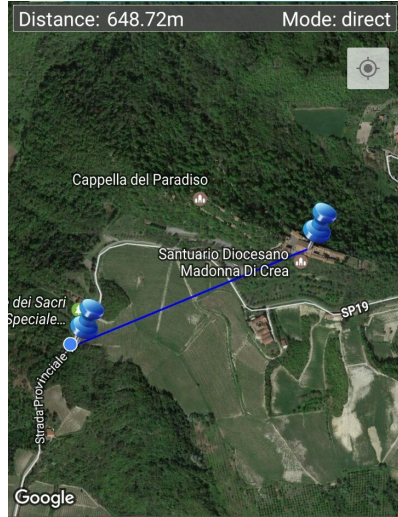


Figure 11.3. Geographical test point 1

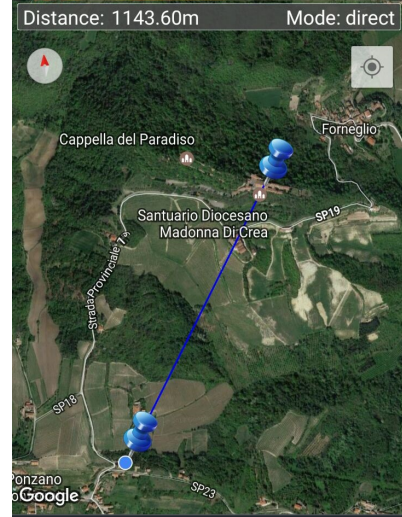


Figure 11.4. Geographical test point 2

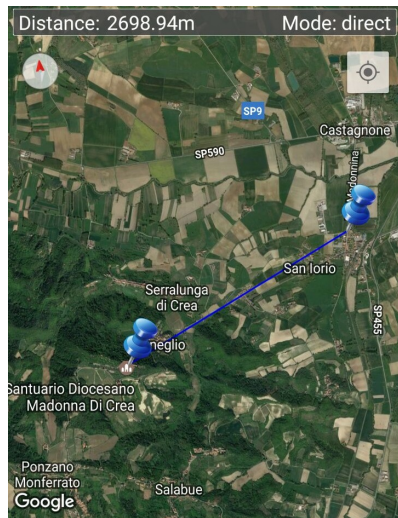


Figure 11.5. Geographical test point 3

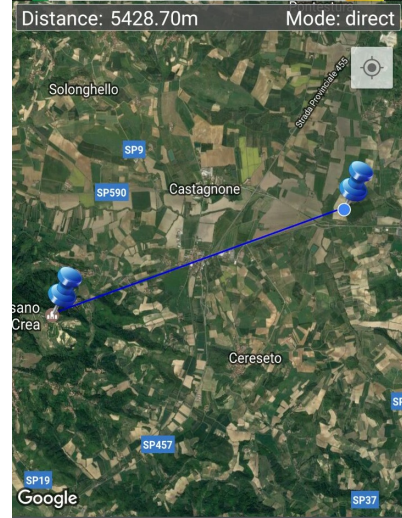


Figure 11.6. Geographical test point 4

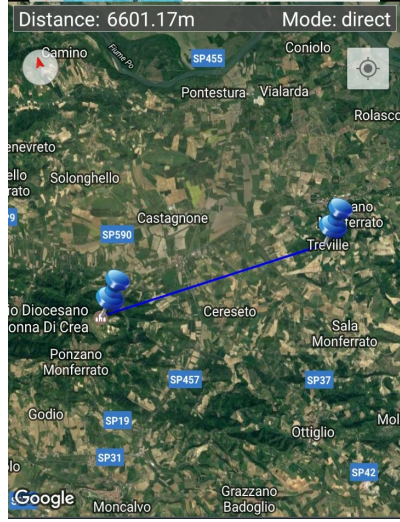


Figure 11.7. Geographical test point 5

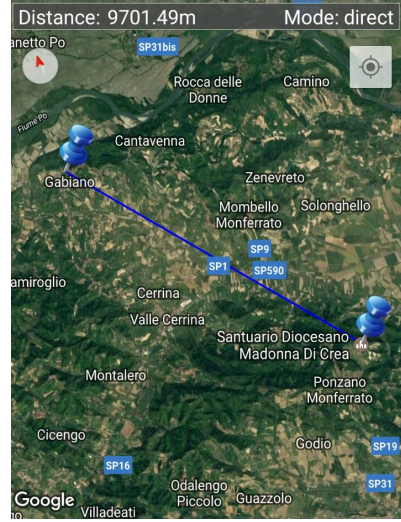


Figure 11.8. Geographical test point 6

RSSI measurements

In the following are reported the RSSI measurements related to the various geographical positions of the node. For completeness are reported, in the same table 11.2, all the values obtained from SwiRem system, the Texas dev. kit and the WiMOD Module. All the measurements are ordered with increasing distance and the cardinal number X refer to the respective "test point X"

	Distance [m]	RSSI WiMOD [dBm]	RSSI CC1310[dBm]	RSSI SwRem [dBm]	Friis [dBm]
1	649	-80	-82	-83	-69
2	1144	-80	-81	-83	-74
3	2697	-110	-123	-122	-81
4	5429	-96	-97	-96	-88
5	6601	-96	-95	-96	-90
6	9701	-95	-95	-97	-93

Table 11.2. RSSI Measures of SwRem, CC1310 Dev Kit and LoRa in Monferrato countryside

At the end of these tests it was possible to certify that the main aspects of the system work properly. It is necessary to further test a ongoing working cycle to evaluate some online errors that may not occur in a brief period of usage.

11.3 Evaluation of LoRa Murata board

The characterisation of the board with LoRa Murata was done on a board printed for a different project. It reached vary good goals in term of performances, link distances and time to market realization. The most important advantage, widely tested in the above mentioned board, was the exploitation of The Things Network community. The infrastructure, offered by the community, works properly, in fact a node can communicate with a public Gateway (or private) sending data to the iXem database on Amazon hosting. In the future further test will be done on a first prototype of SwiRem Lora Murata board.

Chapter 12

Conclusions

In conclusion, the team work done by Mirolli Donato and Trincherò Simone led to the creation of an ad-hoc wireless system capable to monitor power consumption and turning on and off a specific device under test.

The design process started from the specifications and concluded with the physical realization of two boards that implement a wireless sensor network. All the experimental test gave positive results about the working of this system that, even with some errors, behaves as expected. This monitoring approach will be certainly useful in various applications, reducing the time response to a damage of the device under control and to guarantee an improvement in power wasting, allowing to turning off a device, if not needed, without going in the physical place.

This work is open to future implementations and improvements. A first improvement was preliminary treated, at the end of the physical realization of the system with the TI CC1310 transceiver, introducing the new Murata LoRa SoC transceiver and some useful LoRa Alliance services, to exploit the advantages of the LoRaWAN network and the goodness of the LoRa PHY spreading spectrum technology.

Bibliography

- [1] Electronic Communication Committee (ECC). *THE 44TH ECC PLENARY MEETING, DUBLIN, 28 FEBRUARY-3 MARCH 2017*. 2017. URL: <http://www.cept.org/ecc/news/results-from-the-44th-ecc-plenary-meeting-dublin-28-february-3-march-2017/>.
- [2] Javan Erfanian Rachid El Hattachi. *5G White Paper*. Tech. rep. NGMN Alliance, 2015.
- [3] Abel Rodríguez de la Concepción, Riccardo Stefanelli, and Daniele Trincherio. “Ad-hoc multilevel wireless sensor networks for distributed microclimatic diffused monitoring in precision agriculture”. In: *Wireless Sensors and Sensor Networks (WiSNet), 2015 IEEE Topical Conference on*. IEEE. 2015, pp. 14–16.
- [4] MSB Member Dr. Shu Yinbiao Project Leader and Co. “Internet of Things: Wireless Sensor Networks”. In: International Electrotechnical Commission. 2014.
- [5] Kevin Ashton. “That ‘Internet of Things’ Thing - In the real world, things matter more than ideas.” In: *RFID Journal 22.7* (2009).
- [6] Christian Floerkemeier Friedemann Mattern. “From the Internet of Computers to the Internet of Things”. In: *Distributed Systems Group, Institute for Pervasive Computing - ETH Zurich* ().
- [7] Y. Sankarasubramaniam I.F. Akyildiz W. Su and E. Cayirci. “Wireless sensor networks: a survey”. In: *Computer Networks (2002)* (2002).
- [8] Suyash Jain. *Layout Review Techniques for Low Power RF Designs*. Tech. rep. Texas Instruments, Rev 2012.
- [9] Wikipedia. *Osi Model*. URL: https://en.wikipedia.org/wiki/OSI_model.
- [10] Wikipedia. *Spread spectrum*. 2017. URL: http://en.wikipedia.org/wiki/Spread_spectrum.
- [11] Lora Alliance. URL: <https://www.lora-alliance.org>.
- [12] Zigbee Alliance. URL: <http://www.zigbee.org>.

- [13] Z-Wave Alliance. URL: <https://z-wavealliance.org/>.
- [14] Wikipedia. *Frequency Shift Keying*. URL: https://it.wikipedia.org/wiki/Frequency-shift_keying.
- [15] Wikipedia. *Phase Shift Keying*. URL: https://en.wikipedia.org/wiki/Phase-shift_keying.
- [16] Wikipedia. *Amplitude Shift Keying*. URL: https://en.wikipedia.org/wiki/Amplitude-shift_keying.
- [17] Winfield Hill Paul Horowitz. *The Art of Electronics*. Cambridge University Press, 2015, pp. 1032–1037.
- [18] Wikipedia. *Universal asynchronous receiver/transmitter*. 2010. URL: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter.
- [19] Colin M. Maunder (chair, Vice Chair editor) Rodham E. Tulloss, and Various Authors. *IEEE Standard Test Access Port and Boundary-Scan Architecture*. Tech. rep. The Institute of Electrical and Electronics Engineers, Inc., 1993.
- [20] IMST GmbH. *WiMOD iM880A Datasheet - Document ID: 4100/40140/0063*. Tech. rep. IMST GmbH.
- [21] Semtech Corporation. *SX1272/73 DATASHEET*. Tech. rep. Semtech Corporation, Rev July 2014.
- [22] IMST GmbH. *WiMOD LR Studio User Guide Version 1.3 - Document ID: 4100/40140/0061*. Tech. rep. IMST GmbH.
- [23] Murata. *Murata IC LoRa LPWAN*. 2017. URL: <http://wireless.murata.com>.
- [24] LoRa Alliance. *The Things Network*. 2017. URL: <http://www.thethingsnetwork.org>.
- [25] Texas Instrument. *SmartRF06 Evaluation Board User's Guide - SWRU321A*. Tech. rep. Texas Instruments, Rev May 2013.
- [26] Texas Instrument. *TI-RTOS 2.20 User's Guide*. Tech. rep. Texas Instrument, June 2016.
- [27] Texas Instruments. *Autoswitching power MUX - TPS2110A*. Tech. rep. Texas Instruments, Rev 2010.
- [28] Texas Instruments. *TPS6305x Single Inductor Buck-Boost With 1-A Switches and Adjustable Soft Start*. Tech. rep. Texas Instruments, Rev 2015.
- [29] Johanson Technology. *Sub-GHz Impedance Matched Balun + LPF integrated Passive Component for Texas Instruments CC1310 Chipset*. Tech. rep. Johanson Technology, Rev 2016.

- [30] James Murdock and Danielle Griffith. *Crystal Oscillator and Crystal Selection for the CC26xx and CC13xx Family of Wireless MCUs*. Tech. rep. Texas Instruments, Rev 2016.
- [31] Seiko Epson Corporation. *FA-238V FA-238 TSX-3225*. Tech. rep. Seiko Epson Corporation.
- [32] Seiko Epson Corporation. *FC-135R FC-135 FC-255*. Tech. rep. Seiko Epson Corporation.
- [33] Texas Instruments. *INA226 High-Side or Low-Side Measurement, Bi-Directional Current and Power Monitor with I2C Compatible Interface*. Tech. rep. Texas Instruments, Rev 2015.
- [34] Texas Instruments. *CC1310 SimpleLink Ultralow Power Sub-1-GHz Wireless MCU*. Tech. rep. Texas Instruments, Rev 2015.
- [35] Kraig Mitzner. *Complete PCB Design Using OrCAD Capture and Layout*. Elsevier, 2007, pp. 23–24.
- [36] Eurocircuit. *Standard Pool*. URL: <http://www.eurocircuits.com/standard-pool-your-default-choice-for-your-pcb/>.
- [37] Murata. *LoRa module datasheet BP-ABZ-B*. Tech. rep. Murata.
- [38] Where Labs. *Bus Pirate US1*. URL: <http://dangerousprototypes.com/blog>.

Appendix A

Firmware Code

All the following firmware (from page 131 to 159), for the TI CC1310 transceiver, refers to the Texas Instruments Incorporated disclaimer reported below in green, and it was reused, modified, implemented and extended by Donato Mirolli and Simone Trincherio for this thesis in relation with the SwiRem project. Any trades marks that may appear in the code is property of Texas Instruments, as explained in the disclaimer below. In this code may were used and reused parts of some provided examples given to developers as development tools by Texas Instruments. There are no connections, no endorsements or no promotions by Texas Instruments for any part of work.

```
/* Copyright (c) 2015-2018, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * * Redistributions of source code must retain the above copyright
 *   notice, this list of conditions and the following disclaimer.
 *
 * * Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the distribution.
 *
 * * Neither the name of Texas Instruments Incorporated nor the names of
 *   its contributors may be used to endorse or promote products derived
 *   from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
```

```
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

/*Mirolli Trinchero
 *Firmware v2.0 per iXem SwiRem 2018
 */
```

A.1 B1_nodo_RADIO_v2.0.c

```
/****** Includes *****/
#include <B1_nodo_RADIO_v2.0.h>
#include <RadioProtocol_Nodo_v2.0.h>
#include <xdc/std.h>
#include <xdc/runtime/System.h>
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Event.h>
//Power management
#include <ti/drivers/Power.h>
#include <ti/drivers/Power/PowerCC26XX.h>
#include <ti/drivers/Power/PowerCC26XX_tirtos.c>
/* Drivers */
#include <ti/drivers/rf/RF.h>
#include <ti/drivers/PIN.h>

/* Board Header files */
#include "Board.h"
#include "easylink/EasyLink.h"

/****** Defines *****/
#define NODE_TASK_STACK_SIZE 1024
#define NODE_TASK_PRIORITY 3
#define RADIO_EVENT_ALL 0xFFFFFFFF
#define RADIO_EVENT_VALID_PACKET_RECEIVED (uint32_t)(1 << 0)
#define RADIO_EVENT_INVALID_PACKET_RECEIVED (uint32_t)(1 << 1)
#define RADIO_EVENT_COMMAND_PACKET_RECEIVED (uint32_t)(1 << 2)
#define EVENT_WAKE_UP (uint32_t)(1 << 3)
#define RADIO_EVENT_SEND_DATA (uint32_t)(1 << 4)
#define RADIO_EVENT_ACK_TIMEOUT (uint32_t)(1 << 5)
#define RADIO_EVENT_SEND_FAIL (uint32_t)(1 << 6)
#define RADIO_EVENT_NO_OPERATIONS (uint32_t)(1 << 7)
#define RADIO_EVENT_DATA_ACK_RECEIVED (uint32_t)(1 << 8)
#define NODERADIO_MAX_RETRIES 2
#define NODERADIO_ACK_TIMEOUT_TIME_MS (8000)
#define NODERADIO_COMMAND_TIMEOUT_TIME_MS (12000)
#define NODE_CLOCK_INTERVAL_TIMEOUT_MS 10000 //ms

/****** Type declarations *****/
struct RadioOperation {
    EasyLink_TxPacket easyLinkTxPacket;
    uint8_t retriesDone;
    uint8_t maxNumberOfRetries;
    uint32_t ackTimeoutMs;
    enum NodeRadioOperationStatus result;
};
```



```

/***** Variable declarations *****/
static Task_Params NodeRadioTaskParams;
Task_Struct NodeRadioTask; /* not static so you can see in ROV */
static uint8_t NodeRadioTaskStack[NODE_TASK_STACK_SIZE];
Event_Struct radioOperationEvent; /* not static so you can see in ROV */
static Event_Handle radioOperationEventHandle;
static Semaphore_Handle radioAccessSemHandle;
Event_Struct radioOperationEvent; /* not static so you can see in ROV */
static Event_Handle radioOperationEventHandle;
Semaphore_Struct radioResultSem; /* not static so you can see in ROV */
static Semaphore_Handle radioResultSemHandle;
Semaphore_Struct radioAccessSem; /* not static so you can see in ROV */
static uint16_t dataToSend;
static uint8_t flag_wake_moment=0;
static ConcentratorRadio_PacketReceivedCallback packetReceivedCallback;
static union ConcentratorPacket latestRxPacket;
static EasyLink_TxPacket txPacket;
static struct WakeUpPacket WakePacket;
static struct AckPacket ackPacket;
static uint8_t nodoAddress;
static int8_t latestRssi;
static struct RadioOperation currentRadioOperation;

/***** Prototypes *****/
static void NodRadioTaskFunction(UArg arg0, UArg arg1);
static void rxDoneCallback(EasyLink_RxPacket * rxPacket, EasyLink_Status status);
static void notifyPacketReceived(union ConcentratorPacket* latestRxPacket);
static void sendWake(uint8_t MasterAddress);
static void sendAck(uint8_t MasterAddress);
static void returnRadioOperationStatus(enum NodeRadioOperationStatus status);
static void sendCommandPacket(struct NodeDataPacket Packet, uint8_t
    maxNumberOfRetries, uint32_t ackTimeoutMs);
static void resendPacket();
static struct NodeDataPacket dato_da_inviare;
void Clock_Radio_TimeoutCallback(UArg arg0);
/* Pin driver handle */
static PIN_Handle ledPinHandle;
static PIN_State ledPinState;

/* Clock for the fast report timeout */
Clock_Struct RadioReportTimeoutClock; /* not static so you can see in ROV */

PowerCC26XX_Config power;

/* Configure LED Pin */
PIN_Config ledPinTable[] = {
    D2 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL |
        PIN_DRVSTR_MAX,
    PIN_TERMINATE
};
/***** Function definitions *****/
void NodeRadioTask_init(void) {

    /* Create semaphore used for exclusive radio access */
    Semaphore_Params semParam;
    Semaphore_Params_init(&semParam);
    Semaphore_construct(&radioAccessSem, 1, &semParam);
    radioAccessSemHandle = Semaphore_handle(&radioAccessSem);

    /* Create semaphore used for callers to wait for result */
    Semaphore_construct(&radioResultSem, 0, &semParam);
    radioResultSemHandle = Semaphore_handle(&radioResultSem);

```

```
        /* Open LED pins */
        ledPinHandle = PIN_open(&ledPinState, ledPinTable);
        if (!ledPinHandle)
        {
            System_abort("Error initializing GPIO\n");
        }

        /* Create event used internally for state changes */
        Event_Params eventParam;
        Event_Params_init(&eventParam);
        Event_construct(&radioOperationEvent, &eventParam);
        radioOperationEventHandle = Event_handle(&radioOperationEvent);

        /* Create clock object*/
        Clock_Params clkParams2;
        clkParams2.period = 0;
        clkParams2.startFlag = FALSE;
        Clock_construct(&RadioReportTimeoutClock, Clok_Radio_TimeoutCallback, 1, &
            clkParams2);
        RadioTimeoutClockHandle = Clock_handle(&RadioReportTimeoutClock);

        /* Create the node radio protocol task */
        Task_Params_init(&NodeRadioTaskParams);
        NodeRadioTaskParams.stackSize = NODE_TASK_STACK_SIZE;
        NodeRadioTaskParams.priority = NODE_TASK_PRIORITY;
        NodeRadioTaskParams.stack = &NodeRadioTaskStack;
        Task_construct(&NodeRadioTask, NodRadioTaskFunction, &NodeRadioTaskParams,
            NULL);
    }

    void NodeRadioTask_registerPacketReceivedCallback(
        ConcentratorRadio_PacketReceivedCallback callback) {
        packetReceivedCallback = callback;
    }

    static void NodRadioTaskFunction(UArg arg0, UArg arg1)
    {
        /* Initialize EasyLink */
        if(EasyLink_init(RADIO_EASYLINK_MODULATION) != EasyLink_Status_Success) {
            System_abort("EasyLink_init failed");
        }

        /* Set frequency */
        if(EasyLink_setFrequency(RADIO_FREQUENCY) != EasyLink_Status_Success) {
            System_abort("EasyLink_setFrequency failed");
        }

        /* Set node address */;
        nodoAddress = RADIO_NODO_ADDRESS;
        EasyLink_enableRxAddrFilter(&nodoAddress, 1, 1);

        /* Set up Wake up and Ack packet */
        WakePacket.header.sourceAddress = nodoAddress;
        WakePacket.header.packetType = RADIO_PACKET_TYPE_WAKEUP_PACKET;

        ackPacket.header.sourceAddress = nodoAddress;
        ackPacket.header.packetType = RADIO_PACKET_TYPE_COMMAND_ACK_RECEIVED;

        //set n* di clock to wake up node
        Clock_setTimeout(RadioTimeoutClockHandle, NODE_CLOCK_INTERVAL_TIMEOUT_MS * 1000
            / Clock_tickPeriod);
        Clock_start(RadioTimeoutClockHandle);

        //Set the power policy - PowerCC26XX_standbyPolicy
    }
```

```
power.policyInitFxn =NULL;
power.policyFxn = &PowerCC26XX_standbyPolicy;
power.calibrateFxn = &PowerCC26XX_noCalibrate;
power.enablePolicy=FALSE;
power.calibrateRCOSC_LF=FALSE;
power.calibrateRCOSC_HF=FALSE;

PIN_setOutputValue(ledPinHandle , D2,0);

while (1) {
    Power_enablePolicy();
    uint32_t events = Event_pend(radioOperationEventHandle , 0, RADIO_EVENT_ALL
        , BIOS_WAIT_FOREVER);

    //If timer tell to wake up
    if(events & EVENT_WAKE_UP)
    {
        flag_wake_moment=1;//for the RADIO_EVENT_ACK_TIMEOUT to verify if
            is a wake up or an other RX situation

        //Send wake up packet
        sendWake(RADIO_MASTER_ADDRESS);

        Power_setConstraint(PowerCC26XX_SB_DISALLOW);
        Power_setConstraint(PowerCC26XX_IDLE_PD_DISALLOW);

        //Go back to RX with time out to wait a valid command from master
        EasyLink_setCtrl(EasyLink_Ctrl_AsyncRx_TimeOut ,
            EasyLink_ms_To_RadioTime(
                NORERADIO_COMMAND_TIMEOUT_TIME_MS));
        if(EasyLink_receiveAsync(rxDoneCallback , 0) != EasyLink_Status_Success
            ) {
            System_abort("EasyLink_receiveAsync failed");
        }
    }

    if(events & RADIO_EVENT_COMMAND_PACKET_RECEIVED)
    {
        flag_wake_moment=0;
        sendAck(RADIO_MASTER_ADDRESS);
        /* Call packet received callback */
        notifyPacketReceived(&latestRxPacket);
    }

    if(events & RADIO_EVENT_SEND_DATA){
        dato_da_inviare.header.sourceAddress = nodoAddress;
        dato_da_inviare.header.packetType = RADIO_PACKET_TYPE_DATA_PACKET;
        dato_da_inviare.adcValue = dataToSend;

        sendCommandPacket(dato_da_inviare , NORERADIO_MAX_RETRIES ,
            NORERADIO_ACK_TIMEOUT_TIME_MS);

        System_printf("send command\n");
        System_flush();//serve per far stampare
    }

    //If we get an ACK command received from node,
    if (events & RADIO_EVENT_DATA_ACK_RECEIVED)
    {
        returnRadioOperationStatus(NodeRadioStatus_Success);
        Event_post(radioOperationEventHandle ,
            RADIO_EVENT_NO_OPERATIONS);
    }
}
```

```
    }

    //If we get an ACK timeout
    if (events & RADIO_EVENT_ACK_TIMEOUT)
    {
        if(flag_wake_moment==1){
            flag_wake_moment=0;
            Event_post(radioOperationEventHandle,
                RADIO_EVENT_NO_OPERATIONS);
        }else if (currentRadioOperation.retriesDone <
            currentRadioOperation.maxNumberOfRetries)
            resendPacket();
        }else
        {
            //Else return send fail
            Event_post(radioOperationEventHandle,
                RADIO_EVENT_SEND_FAIL);
        }
    }
    //If send fail
    if (events & RADIO_EVENT_SEND_FAIL)
    {
        returnRadioOperationStatus(NodeRadioStatus_Failed);
        Event_post(radioOperationEventHandle,
            RADIO_EVENT_NO_OPERATIONS);
    }

    /* If invalid packet received */
    if(events & RADIO_EVENT_INVALID_PACKET_RECEIVED) {
        Event_post(radioOperationEventHandle,RADIO_EVENT_NO_OPERATIONS);
    }
    //Give the permission to go to sleep and set the timer to wake up
    if(events & RADIO_EVENT_NO_OPERATIONS){
        EasyLink_setCtrl(EasyLink_Ctrl_AsyncRx_TimeOut,
            EasyLink_ms_To_RadioTime(0));
        Power_releaseConstraint(PowerCC26XX_SB_DISALLOW);
        Power_releaseConstraint(PowerCC26XX_IDLE_PD_DISALLOW);
        PIN_setOutputValue(ledPinHandle, D2,0);
        Clock_start(RadioTimeoutClockHandle);
    }
}

}

void done_or_do_nothing(){

    Event_post(radioOperationEventHandle,RADIO_EVENT_NO_OPERATIONS);

}

enum NodeRadioOperationStatus NodeRadioTask_sendCommandData(uint16_t data)
{
    enum NodeRadioOperationStatus status_enum ;
    // Get radio access sempahore
    Semaphore_pend(radioAccessSemHandle, BIOS_WAIT_FOREVER);
    // Save data to send
    dataToSend = data;
    // Raise RADIO_EVENT_SEND_COMMAND_DATA event
    Event_post(radioOperationEventHandle, RADIO_EVENT_SEND_DATA);
    // Wait for result
    Semaphore_pend(radioResultSemHandle, BIOS_WAIT_FOREVER);
    //Get result
    status_enum = currentRadioOperation.result;
}
```

```
        // Return radio access semaphore
        Semaphore_post(radioAccessSemHandle);

    return status_enum;
}

static void returnRadioOperationStatus(enum NodeRadioOperationStatus result)
{
    /* Save result */
    currentRadioOperation.result = result;
    /* Post result semaphore */
    Semaphore_post(radioResultSemHandle);
}

static void sendCommandPacket(struct NodeDataPacket Packet, uint8_t
    maxNumberOfRetries, uint32_t ackTimeoutMs)
{
    //Set destination address in EasyLink API
    currentRadioOperation.easyLinkTxPacket.dstAddr[0] = RADIO_MASTER_ADDRESS;

    // Copy ADC packet to payload
    //Note that the EasyLink API will implicitly both add the length byte and the
    //destination address byte.
    memcpy(currentRadioOperation.easyLinkTxPacket.payload, ((uint8_t*)&Packet),
        sizeof(struct NodeDataPacket));
    currentRadioOperation.easyLinkTxPacket.len = sizeof(struct NodeDataPacket);
    //Setup retries
    currentRadioOperation.maxNumberOfRetries = maxNumberOfRetries;
    currentRadioOperation.ackTimeoutMs = ackTimeoutMs;
    currentRadioOperation.retriesDone = 0;
    EasyLink_setCtrl(EasyLink_Ctrl_AsyncRx_TimeOut, EasyLink_ms_To_RadioTime(
        ackTimeoutMs));

    Power_disablePolicy();//disable the power policy because it interferes with
        the "receiveAsync"

    //Send packet
    if (EasyLink_transmit(&currentRadioOperation.easyLinkTxPacket) !=
        EasyLink_Status_Success)
    {
        System_abort("EasyLink_transmit_1_failed_1");
    }
    //do not insert anything here
    //Enter RX
    if (EasyLink_receiveAsync(rxDoneCallback, 0) != EasyLink_Status_Success)
    {
        System_abort("EasyLink_receiveAsync_1_failed");
    }
}

static void resendPacket()
{
    Power_disablePolicy();//disable the power policy because it interferes
        with the "receiveAsync"
    /* Send packet */
    if (EasyLink_transmit(&currentRadioOperation.easyLinkTxPacket) !=
        EasyLink_Status_Success)
    {
        System_abort("EasyLink_transmit_2_failed_2");
    }
}
```

```
//do not insert anything here

/* Enter RX and wait for ACK with timeout */
if (EasyLink_receiveAsync(rxDoneCallback, 0) != EasyLink_Status_Success)
{
    System_abort("EasyLink_receiveAsync_ failed");
}

/* Increase the retries number by one */
currentRadioOperation.retriesDone++;
}

static void sendWake(uint8_t MasterAddress) {

    /* Set destination Address, but use EasyLink layers destination address
       capability */
    txPacket.dstAddr[0] = MasterAddress;

    memcpy(txPacket.payload, &WakePacket.header, sizeof(WakePacket));
    txPacket.len = sizeof(WakePacket);

    /* Send packet */
    if (EasyLink_transmit(&txPacket) != EasyLink_Status_Success)
    {
        System_abort("EasyLink_transmit_ failed");
    }
}

////Send the ACK to master after it revived a command
static void sendAck(uint8_t MasterAddress) {

    /* Set destinationAddress, but use EasyLink layers destination address
       capability */
    txPacket.dstAddr[0] = MasterAddress;

    /* Copy ACK packet to payload, skipping the destination address byte.
       * Note that the EasyLink API will implicitly both add the length byte and the
       destination address byte. */
    memcpy(txPacket.payload, &ackPacket.header, sizeof(ackPacket));
    txPacket.len = sizeof(ackPacket);

    /* Send packet */
    if (EasyLink_transmit(&txPacket) != EasyLink_Status_Success)
    {
        System_abort("EasyLink_transmit_ failed");
    }
}

static void notifyPacketReceived(union ConcentratorPacket* latestRxPacket)
{
    if (packetReceivedCallback)
    {
        packetReceivedCallback(latestRxPacket, latestRssi);
    }
}

static void rxDoneCallback(EasyLink_RxPacket * rxPacket, EasyLink_Status status)
{
    union ConcentratorPacket* tmpRxPacket;

    /* If we received a packet successfully */
    if (status == EasyLink_Status_Success)
    {
        /* Save the latest RSSI, which is later sent to the receive callback */
        latestRssi = (int8_t)rxPacket->rssi;
    }
}
```

```
/* Check that this is a valid packet */
tmpRxPacket = (union ConcentratorPacket*)(rxPacket->payload);

/* If this is a known packet */
if (tmpRxPacket->header.packetType == RADIO_PACKET_TYPE_COMMAND_PACKET)
{
    /* Save packet */
    memcpy((void*)&latestRxPacket, &rxPacket->payload, sizeof(struct
        CommandPacket));

    /* Signal packet received */
    Event_post(radioOperationEventHandle,
        RADIO_EVENT_COMMAND_PACKET_RECEIVED);

} else if (tmpRxPacket->header.packetType ==
    RADIO_PACKET_TYPE_DATO_ACK_RECEIVED)
{
    /*Signal ACK packet received
    Event_post(radioOperationEventHandle,
        RADIO_EVENT_DATA_ACK_RECEIVED);

}

}

else if(status == EasyLink_Status_Rx_Timeout)
{
    /*Post a RADIO_EVENT_ACK_TIMEOUT event
    Event_post(radioOperationEventHandle, RADIO_EVENT_ACK_TIMEOUT);

}

else
{
    /* Signal invalid packet received */
    Event_post(radioOperationEventHandle, RADIO_EVENT_INVALID_PACKET_RECEIVED)
        ;
}

}

void Clok_Radio_TimeoutCallback(UArg arg0)
{
    PIN_setOutputValue(ledPinHandle, D2,1);
    Event_post(radioOperationEventHandle, EVENT_WAKE_UP);
}
```

A.2 B1__nodo__RADIO__v2.0.h

```
#ifndef TASKS_CONCENTRATORRADIOTASKTASK_H_
#define TASKS_CONCENTRATORRADIOTASKTASK_H_

#include "RadioProtocol_Nodo_v2.0.h"
#include "stdint.h"

enum ConcentratorRadioOperationStatus {
    ConcentratorRadioStatus_Success,
    ConcentratorRadioStatus_Failed,
    ConcentratorRadioStatus_FailedNotConnected,
};

enum NodeRadioOperationStatus {
```

```
NodeRadioStatus_Success,
NodeRadioStatus_Failed,
NodeRadioStatus_FailedNotConnected,
};

union ConcentratorPacket {
    struct PacketHeader header;
    struct CommandPacket rxCommand;
    struct AckPacket ack_master_data_recived;
};

typedef void (*ConcentratorRadio_PacketReceivedCallback)(union ConcentratorPacket*
    packet, int8_t rssi);

/* Create the NoderRadioTask and creates all TI-RTOS objects */
void NodeRadioTask_init(void);

/* Register the packet received callback */
void NodeRadioTask_registerPacketReceivedCallback(
    ConcentratorRadio_PacketReceivedCallback callback);

/* void done_or_do_nothing(void);
permete di comunicare al radio quando gli e' arrivato il comando torna a dormire o
quando ha eseguito un certo comando */

/* Sends an DATA value to the Master */
enum NodeRadioOperationStatus NodeRadioTask_sendCommandData(uint16_t data);

#endif /* TASKS_CONCENTRATORRADIOTASKTASK_H_ */
```

A.3 B1_nodo_TASK_v2.0.c

```
/* Includes */

#include <B1_nodo_RADIO_v2.0.h>
#include <B1_nodo_TASK_v2.0.h>
#include <RadioProtocol_Nodo_v2.0.h>
#include <xdc/std.h>
#include <xdc/runtime/System.h>
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Event.h>

/* Drivers */
#include <ti/drivers/PIN.h>
#include <ti/mw/display/Display.h>
#include <ti/mw/display/DisplayExt.h>

/* Board Header files */
#include "Board.h"

/* Driver Header files */
#include <ti/drivers/ADC.h>
#if defined(CC2650DK_7ID) || defined(CC1310DK_7XD)
#include <ti/drivers/PIN.h>
#endif

/* Defines */
#define NODE_TASK_STACK_SIZE 1024
```



```
#define NODE_TASK_PRIORITY    3
#define NODE_EVENT_ALL 0xFFFFFFFF
#define NODE_EVENT_NEW_COMMAND (uint32_t)(1 << 0)
#define NODE_EVENT_NEW_DATA (uint32_t)(1 << 1)
#define CONCENTRATOR_MAX_NODES 7
#define CONCENTRATOR_DISPLAY_LINES 8

#define NODE_CLOCK_INTERVAL_TIMEOUT_MS    30000 //ms

/***** Type declarations *****/
struct AdcSensorNode {
    uint8_t address;
    uint16_t latestAdcValue;
    uint8_t button;
    int8_t latestRssi;
};

/* Pin driver handle */
static PIN_Handle ledPinHandle1;
static PIN_State ledPinState1;

/* Configure LED Pin */
PIN_Config ledPinTable1[] = {
    D3 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL |
        PIN_DRVSTR_MAX,
    D1 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL |
        PIN_DRVSTR_MAX,
    Board_ALS_PWR | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL |
        PIN_DRVSTR_MAX,
    Board_DIO27_ANALOG | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW |
        PIN_PUSHPULL | PIN_DRVSTR_MAX,
    PIN_TERMINATE
};

/* ADC conversion result variables */
uint16_t adcValue0;

/***** Variable declarations *****/
static Task_Params NodeTaskParams;
Task_Struct NodeTask; /* not static so you can see in ROV */
static uint8_t NodeTaskStack[NODE_TASK_STACK_SIZE];
Event_Struct NodeEvent; /* not static so you can see in ROV */
static Event_Handle NodeEventHandle;
struct AdcSensorNode knownSensorNodes[CONCENTRATOR_MAX_NODES];
static struct CommandPacket commandReceived;
static uint16_t stato_relay;
static uint16_t Sense_220;

//Commands to control the node
uint8_t GET_RELAY = 0x61;//a
uint8_t SET_OPEN_RELAY = 0x62;//b
uint8_t SET_CLOSE_RELAY = 0x63;//c
uint8_t GO_BACK_SLEEP = 0x65;//e
uint8_t CONTROL_220 = 0x64;//d
uint8_t CONTROL_BATTERY = 0x05;
uint8_t CONTROL_I_LOAD = 0x06;
uint8_t READ_ADC_VALUE = 0x07;

/***** Prototypes *****/
static void NodeTaskFunction(UArg arg0, UArg arg1);
static void packetReceivedCallback(union ConcentratorPacket* packet, int8_t rssi);
void readADCvalue ();
```

```
void ClokTimeoutCallback(UArg arg0);

/**I2C**/
static I2C_Handle i2c_node_handle;
static I2C_Params i2c_node_param;
static I2C_Transaction i2cTransaction;
unsigned char i2cBuffer[3];
int8_t INA_READ_I;
int8_t INA_READ_V;

void i2cInizializzazione_blocking(int8_t address);
I2C_Config i2c_configuration;
I2C_FxnTable i2c_table;
static uint16_t INA_value;

/***** Function definitions *****/
void NodeTask_init(void) {

    /* Open LED pins */
    ledPinHandle1 = PIN_open(&ledPinState1, ledPinTable1);
    if (!ledPinHandle1)
    {
        System_abort("RR_Error_initializing_board\n");
    }

    /* Create event used internally for state changes */
    Event_Params eventParam;
    Event_Params_init(&eventParam);
    Event_construct(&NodeEvent, &eventParam);
    NodeEventHandle = Event_handle(&NodeEvent);

    /* Create the node radio protocol task */
    Task_Params_init(&NodeTaskParams);
    NodeTaskParams.stackSize = NODE_TASK_STACK_SIZE;
    NodeTaskParams.priority = NODE_TASK_PRIORITY;
    NodeTaskParams.stack = &NodeTaskStack;
    Task_construct(&NodeTask, NodeTaskFunction, &NodeTaskParams, NULL);
}

static void NodeTaskFunction(UArg arg0, UArg arg1)
{

    /* Register a packet received callback with the radio task */
    NodeRadioTask_registerPacketReceivedCallback(packetReceivedCallback);

    /* Enter main task loop */
    while(1) {
        /* Wait for event */
        uint32_t events = Event_pend(NodeEventHandle, 0, NODE_EVENT_ALL,
            BIOS_WAIT_FOREVER);

        /* If we got a new ADC sensor value */
        if(events & NODE_EVENT_NEW_COMMAND) {
            PIN_setOutputValue(ledPinHandle1, D1,1);

            if(commandReceived.command == SET_OPEN_RELAY){

                PIN_setOutputValue(ledPinHandle1, Board_ALS_PWR,1);
                PIN_setOutputValue(ledPinHandle1, D3,1);
                Clock_start(RadioTimeoutClockHandle);
                done_or_do_nothing();
                PIN_setOutputValue(ledPinHandle1, D1,0);
            }
        }
    }
}
```

```

    }else if(commandRecived.command == SET_CLOSE_RELAY){

        PIN_setOutputValue(ledPinHandle1, Board_ALS_PWR,0);
        PIN_setOutputValue(ledPinHandle1, D3,0);
        Clock_start(RadioTimeoutClockHandle);
        done_or_do_nothing();
        PIN_setOutputValue(ledPinHandle1, D1,0);

    }else if(commandRecived.command == GET_RELAY){

        stato_relay = (uint16_t)PIN_getOutputValue(Board_ALS_PWR);
        NodeRadioTask_sendCommandData(stato_relay);
        PIN_setOutputValue(ledPinHandle1, D1,0);

    }else if(commandRecived.command == GO_BACK_SLEEP){

        done_or_do_nothing();
        PIN_setOutputValue(ledPinHandle1, D1,0);

    }else if(commandRecived.command == CONTROL_220){

        Sense_220 = (uint16_t)PIN_getOutputValue(
            Board_DIO27_ANALOG);//pin 40/DIO_27 on CC1310, where
is possible to sense the presence of 220
        NodeRadioTask_sendCommandData(Sense_220);
        PIN_setOutputValue(ledPinHandle1, D3,0);

    }else if(commandRecived.command == CONTROL_BATTERY){
//read V
        i2cInizializzazioni_blocking(INA_READ_V);
        NodeRadioTask_sendCommandData(INA_value);

    }else if(commandRecived.command == CONTROL_I_LOAD){

//read I
        i2cInizializzazioni_blocking(INA_READ_I);
        NodeRadioTask_sendCommandData(INA_value);

    }else if(commandRecived.command == READ_ADC_VALUE){
//If it is necessary to read a value with ADC
        readADCvalue();
        NodeRadioTask_sendCommandData(adcValue0);
        PIN_setOutputValue(ledPinHandle1, D3,0);
    }
    else {
        Clock_start(RadioTimeoutClockHandle);
        done_or_do_nothing();
        PIN_setOutputValue(ledPinHandle1, D1,0);
    }
}
}

//I2C initialization
void i2cInizializzazioni_blocking(int8_t address){

    I2C_Params_init(&i2c_node_param);
    i2c_node_param.bitRate = I2C_100kHz;
    i2c_node_param.transferMode = I2C_MODE_BLOCKING;
    i2c_node_param.transferCallbackFxn = NULL;
    i2c_node_param.custom =(uintptr_t) NULL;
    I2C_init();

```

```

i2cBuffer[0]=(char)address;

i2c_node_handle=I2C_open(Board_I2C , &i2c_node_param);
if (!i2c_node_handle){
    System_printf("I2C did not open");
}
i2cTransaction.readCount=2
i2cTransaction.slaveAddress = 0x40;//INA HW ADDRESS
i2cTransaction.writeBuf = i2cBuffer;
i2cTransaction.writeCount = 1;
i2cTransaction.readBuf = i2cBuffer;
i2cTransaction.readCount = 2;
I2C_transfer(i2c_node_handle, &i2cTransaction);
INA_value = i2cBuffer[1] << 8 | i2cBuffer[2];
}
//Open an ADC instance and get a sampling result from a one-shot conversion.
void readADCvalue ()
{
    ADC_Handle    adc;
    ADC_Params    params;
    int_fast16_t  res;

    ADC_Params_init(&params);
    adc = ADC_open(Board_ADC0 , &params);

    if (adc == NULL) {
        System_abort("Error initializing ADC channel 0\n");
    }
    else {
        System_printf("ADC channel 0 initialized\n");
    }
    /* Blocking mode conversion */
    res = ADC_convert(adc, &adcValue0); //adcValue0 is a global variable

    if (res == ADC_STATUS_SUCCESS) {
        System_printf("ADC channel 0 convert result: 0x%x\n", adcValue0);
    }
    else {
        System_printf("ADC channel 0 convert failed\n");
    }

    ADC_close(adc);

    System_flush();
}

static void packetReceivedCallback(union ConcentratorPacket* packet, int8_t rssi)
{
    /* If we recived an ADC sensor packet, for backward compatibility */
    if (packet->header.packetType == RADIO_PACKET_TYPE_COMMAND_PACKET)
    {
        commandRecived.command = packet->rxCommand.command ;

        Event_post(NodeEventHandle , NODE_EVENT_NEW_COMMAND);
    }
}

```

A.4 B1_nodo_TASK_v2.0.h

```
#ifndef TASKS_CONCENTRATOR_TASK_H_
```

```
#define TASKS_CONCENTRATOR_TASK_H_

/* Create the ConcentratorRadioTask and creates all TI-RTOS objects */
void NodeTask_init(void);

#endif /* TASKS_CONCENTRATOR_TASK_H_ */
```

A.5 B2_master_RADIO_v2.0.c

```
/****** Includes *****/
#include <B2_master_RADIO_v2.0.h>
#include <xdc/std.h>
#include <xdc/runtime/System.h>
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Event.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/drivers/Power.h>
#include <ti/drivers/power/PowerCC26XX.h>
/* Drivers */
#include <ti/drivers/rf/RF.h>
#include <ti/drivers/PIN.h>
/* Board Header files */
#include "Board.h"
#include <stdlib.h>
#include <driverlib/trng.h>
#include <driverlib/aon_batmon.h>
#include <RadioProtocol_Master_v2.0.h>
#include "easylink/EasyLink.h"

/****** Defines *****/
#define MASTERRADIO_TASK_STACK_SIZE 1024
#define MASTERRADIO_TASK_PRIORITY 3

#define RADIO_EVENT_ALL 0xFFFFFFFF //
#define RADIO_EVENT_SEND_COMMAND (uint32_t)(1 << 0) //
#define RADIO_EVENT_COMMAND_ACK_RECEIVED (uint32_t)(1 << 1)
#define RADIO_EVENT_ACK_TIMEOUT (uint32_t)(1 << 2)
#define RADIO_EVENT_SEND_FAIL (uint32_t)(1 << 3)
#define RADIO_EVENT_WAKEUP_RECEIVED (uint32_t)(1 << 4)
#define RADIO_EVENT_RETURN_RX (uint32_t)(1 << 5)
#define RADIO_EVENT_DATA_PACKET_RECEIVED (uint32_t)(1 << 6)
#define RADIO_EVENT_INVALID_PACKET_RECEIVED (uint32_t)(1 << 7)
#define MASTERRADIO_MAX_RETRIES 2
#define MASTERRADIO_ACK_TIMEOUT_TIME_MS (1000)

/****** Type declarations *****/
struct RadioOperation {
    EasyLink_TxPacket easyLinkTxPacket;
    uint8_t retriesDone;
    uint8_t maxNumberOfRetries;
    uint32_t ackTimeoutMs;
    enum MasterRadioOperationStatus result;
};

/****** Variable declarations *****/
static MasterRadio_PacketReceivedCallback packetReceivedCallback;
static Task_Params MasterRadioTaskParams;
Task_Struct masterRadioTask; /* not static so you can see in ROV */
```

```
static uint8_t masterRadioTaskStack[MASTERRADIO_TASK_STACK_SIZE];
Semaphore_Struct radioAccessSem; /* not static so you can see in ROV */
static Semaphore_Handle radioAccessSemHandle;
Event_Struct radioOperationEvent; /* not static so you can see in ROV */
static Event_Handle radioOperationEventHandle;
Semaphore_Struct radioResultSem; /* not static so you can see in ROV */
static Semaphore_Handle radioResultSemHandle;
static struct RadioOperation currentRadioOperation;
static uint8_t masterAddress = 0;
static struct CommandPacket cmdPacket;
static int flag_wake_recived=0; //flag per il controllo del wake up ricevuto
static int ppp=0;
static union MasterPacket latestRxPacket1;
static uint16_t commandToSend;
extern PIN_Handle ledPinHandle;
static struct AckPacket ackPacket;

static int8_t latestRssi;
static EasyLink_TxPacket txPacket;
//static int8_t latestRssi;
//static struct DualModeSensorPacket dmSensorPacket;
/* previous Tick count used to calculate uptime for the Sub1G packet */
//static uint32_t prevTicks;
/***** Prototypes *****/
static void MasterRadioTaskFunction(UArg arg0, UArg arg1);
static void returnRadioOperationStatus(enum MasterRadioOperationStatus status);
static void sendCommandPacket(struct CommandPacket cmdPacket, uint8_t
    maxNumberOfRetries, uint32_t ackTimeoutMs);
static void resendPacket();
static void rxDoneCallback(EasyLink_RxPacket * rxPacket, EasyLink_Status status);
static void rerutnInRX_noTimeOut();
static void sendAck(uint8_t MasterAddress);
static void notifyPacketReceived( union MasterPacket* latestRxPacket);
/***** Function definitions *****/
void MasterRadioTask_init(void) {

    /* Create semaphore used for exclusive radio access */
    Semaphore_Params semParam;
    Semaphore_Params_init(&semParam);
    Semaphore_construct(&radioAccessSem, 1, &semParam);
    radioAccessSemHandle = Semaphore_handle(&radioAccessSem);

    /* Create semaphore used for callers to wait for result */
    Semaphore_construct(&radioResultSem, 0, &semParam);
    radioResultSemHandle = Semaphore_handle(&radioResultSem);
    /* Create event used internally for state changes */
    Event_Params eventParam;
    Event_Params_init(&eventParam);
    Event_construct(&radioOperationEvent, &eventParam);
    radioOperationEventHandle = Event_handle(&radioOperationEvent);
    /* Create the radio protocol task */
    Task_Params_init(&MasterRadioTaskParams);
    MasterRadioTaskParams.stackSize = MASTERRADIO_TASK_STACK_SIZE;
    MasterRadioTaskParams.priority = MASTERRADIO_TASK_PRIORITY;
    MasterRadioTaskParams.stack = &masterRadioTaskStack;
    Task_construct(&masterRadioTask, MasterRadioTaskFunction, &
        MasterRadioTaskParams, NULL);

    System_printf("B2_masterRadio_ RadioTask_inizializzato\n");
    System_flush();
}
```

```
void MasterRadioTask_registerPacketReceivedCallback(
    MasterRadio_PacketReceivedCallback callback) {
    packetReceivedCallback = callback;
}

//MAIN RADIO TASK FUNCTION
static void MasterRadioTaskFunction(UArg arg0, UArg arg1)
{
    /* Initialize EasyLink */
    if(EasyLink_init(RADIO_EASYLINK_MODULATION) != EasyLink_Status_Success) {
        System_abort("EasyLink_init_ failed");
    }

    // Set frequency
    if(EasyLink_setFrequency(RADIO_FREQUENCY) != EasyLink_Status_Success) {
        System_abort("EasyLink_setFrequency_ failed");
    }
    masterAddress = RADIO_MASTER_ADDRESS;

    /* Set the filter to the generated random address */
    if (EasyLink_enableRxAddrFilter(&masterAddress, 1, 1) !=
        EasyLink_Status_Success)
    {
        System_abort("EasyLink_enableRxAddrFilter_ failed");
    }

    cmdPacket.header.sourceAddress = masterAddress;
    cmdPacket.header.packetType = RADIO_PACKET_TYPE_COMMAND_PACKET;

    System_printf("B2_masterRadio_ impost. _radio_ settate\n");
    System_flush(); //serve per far stampare

    // Start in receiving mode
    if(EasyLink_receiveAsync(rxDoneCallback, 0) != EasyLink_Status_Success) {
        System_abort("EasyLink_receiveAsync_ failed");
    }

    /* Initialize previous Tick count*/
    // prevTicks = Clock_getTicks();

    /* Enter main task loop */
    while (1)
    {
        // Wait for an event
        uint32_t eventsRadio = Event_pend(radioOperationEventHandle, 0,
            RADIO_EVENT_ALL, BIOS_WAIT_FOREVER);
        if (eventsRadio & RADIO_EVENT_WAKEUP_RECEIVED )
        {
            System_printf("B2_masterRadio_ while_ WAKE_UP_ RECEIVED\n");
            System_printf("1\n");
            System_flush(); //serve per far stampare
            flag_wake_recived=1; //setto che ho ricevuto un wake up da un
                                nodo
            notify_toTask_addNode(latestRxPacket1.header.sourceAddress);

            Event_post(radioOperationEventHandle, RADIO_EVENT_SEND_COMMAND);
        }
        if(eventsRadio & RADIO_EVENT_SEND_COMMAND)
        {
            System_printf("B2_masterRadio_ evento_ SEND_ COMMAND_ ricevuto\n");
            System_flush(); //serve per far stampare
        }
    }
}
```

```

        //cmdPacket.command = (uint8_t)commandToSend; //!
        PIN_getInputValue(Board_BUTTON0); ///USARE NUMERI
        DISPARI

    if(ppp==0){
        cmdPacket.command = 0x63;
        ppp=1;
    }else if(ppp==1){
        cmdPacket.command = 0x62;
        ppp=0;
    }

    sendCommandPacket(cmdPacket, MASTERRADIO_MAX_RETRIES,
        MASTERRADIO_ACK_TIMEOUT_TIME_MS);
    System_printf("B2_masterRadio_dopo_sendCommand\n");
    System_flush();//serve per far stampare
}

if (eventsRadio & RADIO_EVENT_DATA_PACKET_RECEIVED)
{
    sendAck(latestRxPacket1.header.sourceAddress);
    //sendAck(0x01);
    notifyPacketReceived(&latestRxPacket1);
    System_printf("B2_masterRadio_DATI_RICEVUTI_DAL_NODO_0x%02x\n",
        latestRxPacket1.rxCallBack_datiNodo.adcValue);
    System_flush();
    Event_post(radioOperationEventHandle, RADIO_EVENT_RETURN_RX);
}

if (eventsRadio & RADIO_EVENT_COMMAND_ACK_RECEIVED)
{System_printf("4\n");
    System_printf("B2_masterRadio_ACK_RECEIVED\n");
    System_flush();//serve per far stampare
    returnRadioOperationStatus(MasterRadioStatus_Success);
    rerutnInRX_noTimeOut();
}

//If we get an ACK timeout
if (eventsRadio & RADIO_EVENT_ACK_TIMEOUT)
{System_printf("5\n");
    //If we haven't resent it the maximum number of times yet,
    then resend packet
    if (currentRadioOperation.retriesDone <
        currentRadioOperation.maxNumberOfRetries)
    {
        resendPacket();
    }else
    {System_printf("6\n");
        System_printf("B2_masterRadio_ACK_TimeOut_
            raggiunto\n");
        System_flush();//serve per far stampare
        //Else return send fail
        Event_post(radioOperationEventHandle,
            RADIO_EVENT_SEND_FAIL);
    }
}

//If send fail
if (eventsRadio & RADIO_EVENT_SEND_FAIL)
{System_printf("7\n");
    System_printf("B2_masterRadio_radio_send_fallita\n");
    System_flush();//serve per far stampare
    returnRadioOperationStatus(MasterRadioStatus_Failed);
    notify_node_notRespond(latestRxPacket1.header.
        sourceAddress);
}

```



```

        Event_post(radioOperationEventHandle,
                    RADIO_EVENT_RETURN_RX);
    }

    if (eventsRadio & RADIO_EVENT_RETURN_RX)
    {System_printf("8\n");
     rerutnInRX_noTimeOut();
    }

    //se ho ricevuto un pacchetto non valido ritorno in ricezione
    if (eventsRadio & RADIO_EVENT_INVALID_PACKET_RECEIVED)
    {System_printf("9\n");
     rerutnInRX_noTimeOut();
    }
    System_printf("while\n");
}

}

void rerutnInRX_noTimeOut(){

    EasyLink_setCtrl(EasyLink_Ctrl_AsyncRx_TimeOut, EasyLink_ms_To_RadioTime
                     (0));
    /* Enter RX */
    if (EasyLink_receiveAsync(rxDoneCallback, 0) != EasyLink_Status_Success)
    {
        System_abort("EasyLink_receiveAsync failed");
    }
}

enum MasterRadioOperationStatus MasterRadioTask_sendCommandData(uint16_t data)
{
    System_printf("B2_masterRadio_enum chiamato da masterTask\n");
    System_flush();

    enum MasterRadioOperationStatus status_enum ;

    if(flag_wake_recived==1)
    {
        /* Get radio access semaphore */
        Semaphore_pend(radioAccessSemHandle, BIOS_WAIT_FOREVER);

        /* Save data to send */
        commandToSend = data;
        System_printf("master_radio_dentro_enum, comando:%d\n",
                      commandToSend);
        System_flush();
        receiveBufferPointer

        /* Raise RADIO_EVENT_SEND_COMMAND_DATA event */
        Event_post(radioOperationEventHandle, RADIO_EVENT_SEND_COMMAND);

        flag_wake_recived = 0;
        /* Wait for result */
        Semaphore_pend(radioResultSemHandle, BIOS_WAIT_FOREVER);

        /* Get result */
        status_enum = currentRadioOperation.result;
        /* Return radio access semaphore */
        Semaphore_post(radioAccessSemHandle);
        System_printf("B2_masterRadio_enum DOP0 semaforo send command
                      avvenuto\n");
        System_flush();
    }
}

```

```
    }
    return status_enum;
}

static void returnRadioOperationStatus(enum MasterRadioOperationStatus result)
{
    /* Save result */
    currentRadioOperation.result = result;

    /* Post result semaphore */
    Semaphore_post(radioResultSemHandle);
}

static void sendCommandPacket(struct CommandPacket cmdPacket, uint8_t
    maxNumberOfRetries, uint32_t ackTimeoutMs)
{
    //Set destination address in EasyLink API
    currentRadioOperation.easyLinkTxPacket.dstAddr[0] = latestRxPacket1.header.
        sourceAddress;
    //currentRadioOperation.easyLinkTxPacket.dstAddr[0] = 0x01;

    // Copy ADC packet to payload
    /* Note that the EasyLink API will implicitly both add the length byte and
        the destination address byte.
    memcpy(currentRadioOperation.easyLinkTxPacket.payload, ((uint8_t*)&cmdPacket),
        sizeof(struct CommandPacket));
    currentRadioOperation.easyLinkTxPacket.len = sizeof(struct CommandPacket);

    //Setup retries
    currentRadioOperation.maxNumberOfRetries = maxNumberOfRetries;
    currentRadioOperation.ackTimeoutMs = ackTimeoutMs;
    currentRadioOperation.retriesDone = 0;
    EasyLink_setCtrl(EasyLink_Ctrl_AsyncRx_TimeOut, EasyLink_ms_To_RadioTime(
        ackTimeoutMs));

    //Send packet
    if (EasyLink_transmit(&currentRadioOperation.easyLinkTxPacket) !=
        EasyLink_Status_Success)
    {
        System_abort("EasyLink_transmit_ failed_1");
    }

    //Enter RX
    if (EasyLink_receiveAsync(rxDoneCallback, 0) != EasyLink_Status_Success)
    {
        System_abort("EasyLink_receiveAsync_ failed");
    }
}

static void sendAck(uint8_t MasterAddress) {

    /* Set destination Address, but use EasyLink layers destination address
        capability */
    txPacket.dstAddr[0] = MasterAddress;

    ackPacket.header.sourceAddress = RADIO_MASTER_ADDRESS;
    ackPacket.header.packetType = RADIO_PACKET_TYPE_DAT0_ACK_RECEIVED;
    /* Copy ACK packet to payload, skipping the destination address byte.
```

```

    * Note that the EasyLink API will implicitly both add the length byte and the
      destination address byte. */
memcpy(txPacket.payload, &ackPacket.header, sizeof(ackPacket));
txPacket.len = sizeof(ackPacket);

/* Send packet */
if (EasyLink_transmit(&txPacket) != EasyLink_Status_Success)
{
    System_abort("EasyLink_transmit_␣failed");
}
System_printf("B2_masterRadio_␣send_␣ack\n");
System_flush();//serve per far stampare
}

static void resendPacket()
{
    System_printf("B2_masterRadio_␣dentro_␣resendPacket\n");
    System_flush();//serve per far stampare
    /* Send packet */
    if (EasyLink_transmit(&currentRadioOperation.easyLinkTxPacket) !=
        EasyLink_Status_Success)
    {
        System_abort("EasyLink_transmit_␣failed_␣2");
    }

    /* Enter RX and wait for ACK with timeout */
    if (EasyLink_receiveAsync(rxDoneCallback, 0) != EasyLink_Status_Success)
    {
        System_abort("EasyLink_receiveAsync_␣failed");
    }

    /* Increase retries by one */
    currentRadioOperation.retriesDone++;
}

static void notifyPacketReceived(union MasterPacket* latestRxPacket)
{
    if (packetReceivedCallback)
    {
        packetReceivedCallback(latestRxPacket, latestRssi);
    }
}

static void rxDoneCallback(EasyLink_RxPacket * rxPacket, EasyLink_Status status)
{
    union MasterPacket* tmpRxPacketByNodo; //modificato tipo di struttura

    /* If we received a packet successfully */
    if (status == EasyLink_Status_Success)
    {
        System_printf("B2_masterRadio_␣dentro_␣rxDoneCallback_␣easy_␣link_␣
            success\n");
        System_flush();//serve per far stampare

        /* Save the latest RSSI, which is later sent to the receive
           callback */
        latestRssi = (int8_t)rxPacket->rssi;

        /* Check that this is a valid packet */
        tmpRxPacketByNodo = (union MasterPacket*)(rxPacket->payload);

        /* If this is a known packet */ //fare i vari casi (ogni type un
           controllo)
    }
}

```

```
if (tmpRxPacketByNodo->header.packetType ==
    RADIO_PACKET_TYPE_WAKEUP_PACKET)
{
    System_printf("B2_masterRadio_rxDoneCallback_WAKEUP_
        RECEIVED\n");
    System_flush();//serve per far stampare
    /* Save packet */
    memcpy((void*)&latestRxPacket1, &rxPacket->payload, sizeof
        (struct WakeUpPacket));
    /* Signal packet received */
    Event_post(radioOperationEventHandle,
        RADIO_EVENT_WAKEUP_RECEIVED);
}
else if (tmpRxPacketByNodo->header.packetType ==
    RADIO_PACKET_TYPE_DATA_PACKET)
{
    memcpy((void*)&latestRxPacket1, &rxPacket->payload, sizeof
        (struct NodeDataPacket));
    Event_post(radioOperationEventHandle,
        RADIO_EVENT_DATA_PACKET_RECEIVED);
}
else if (tmpRxPacketByNodo->header.packetType ==
    RADIO_PACKET_TYPE_COMMAND_ACK_RECEIVED)
{
    memcpy((void*)&latestRxPacket1, &rxPacket->payload, sizeof
        (struct AckPacket));
    System_printf("B2_masterRadio_rxDoneCallback_ACK_RECEIVED\n
        ");
    System_flush();//serve per far stampare
    //Signal ACK packet received
    Event_post(radioOperationEventHandle,
        RADIO_EVENT_COMMAND_ACK_RECEIVED);
}

}
else if(status == EasyLink_Status_Rx_Timeout)
{
    System_printf("B2_masterRadio_rxDoneCallback_Timeout\n");
    System_flush();//serve per far stampare
    //Post a RADIO_EVENT_ACK_TIMEOUT event
    Event_post(radioOperationEventHandle, RADIO_EVENT_ACK_TIMEOUT);
}
else
{
    System_printf("B2_masterRadio_rxDoneCallback_invalid_packet_
        recived\n");
    System_flush();//serve per far stampare
    /* Signal invalid packet received */
    Event_post(radioOperationEventHandle,
        RADIO_EVENT_INVALID_PACKET_RECEIVED);
}
}
```

A.6 B2__master__RADIO__v2.0.h

```
#ifndef TASKS_NODERADIOTASKTASK_H_
#define TASKS_NODERADIOTASKTASK_H_

#include "RadioProtocol_Master_v2.0.h"
#include "stdint.h"
```

```
#define NODE_ACTIVITY_LED Board_LED0

enum MasterRadioOperationStatus {
    MasterRadioStatus_Success,
    MasterRadioStatus_Failed,
    MasterRadioStatus_FailedNotConnected,
};

union MasterPacket {
    struct PacketHeader header;
    struct WakeUpPacket rxCallBack_wakeup;
    struct AckPacket rxCallBack_Ack;
    struct NodeDataPacket rxCallBack_datiNodo;
};

typedef void (*MasterRadio_PacketReceivedCallback)(union MasterPacket* packet,
    int8_t rssi);
void MasterRadioTask_registerPacketReceivedCallback(
    MasterRadio_PacketReceivedCallback callback);

void notify_node_notRespond(int8_t address);
void notify_toTask_addNode(int8_t address);
/* Initializes the NodeRadioTask and creates all TI-RTOS objects */
void MasterRadioTask_init(void);

/* Sends an ADC value to the concentrator */
enum MasterRadioOperationStatus MasterRadioTask_sendCommandData(uint16_t data);

#endif /* TASKS_NODERADIOTASKTASK_H_ */
```

A.7 B2__master__TASK__v2.0.c

```
/* ***** Includes ***** */

#include <B2_master_RADIO_v2.0.h>
#include <B2_master_TASK_v2.0.h>
#include <xdc/std.h>
#include <xdc/runtime/System.h>

#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Event.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/drivers/PIN.h>
#include <ti/drivers/SPI.h>

/* Board Header files */
#include "Board.h"
#include "SceAdc.h"

/* ***** Defines ***** */
#define MASTER_TASK_STACK_SIZE 1024
#define MASTER_TASK_PRIORITY 3

#define MASTER_EVENT_ALL 0xFFFFFFFF
#define MASTER_EVENT_NEW_ADC_VALUE (uint32_t)(1 << 0) //FORSE DA CANCELLARE
#define MASTER_EVENT_NEW_COMMAND_VALUE (uint32_t)(1 << 1)
```

```
#define MASTER_EVENT_NEW_DATA_VALUE      (uint32_t)(1 << 2)
/* A change mask of 0xFF0 means that changes in the lower 4 bits does not trigger
   a wakeup. */
#define MASTER_ADCTASK_CHANGE_MASK      0xFF0

/* Minimum slow Report interval is 50s (in units of samplingTime)*/
#define NODE_ADCTASK_REPORTINTERVAL_SLOW      50
/* Minimum fast Report interval is 1s (in units of samplingTime) for 30s*/
#define NODE_ADCTASK_REPORTINTERVAL_FAST      1
#define NODE_ADCTASK_REPORTINTERVAL_FAST_DURATION_MS      30000
#define MAX_NODES      10

/***** Variable declarations *****/
static Task_Params masterTaskParams;
Task_Struct masterTask; /* not static so you can see in ROV */
static uint8_t masterTaskStack[MASTER_TASK_STACK_SIZE];
Event_Struct masterTaskEvent; /* not static so you can see in ROV */
static Event_Handle masterEventHandle;
//static uint16_t latestAdcValue; //da cancellare
static uint16_t latestCommandValue;
static struct NodeDataPacket dato_ricevuto_daNodo;
/* Clock for the fast report timeout */
//Clock_Struct fastReportTimeoutClock; //DA ELIMINARE // not static so you can
//see in ROV
//static Clock_Handle fastReportTimeoutClockHandle; //DA ELIMINARE

static void addNewNode(struct NodeDataPacket* node);
static void updateNode(struct NodeDataPacket* node);
static uint8_t isKnownNodeAddress(uint8_t address);

/* Pin driver handle */
static PIN_Handle buttonPinHandle;
static PIN_Handle ledPinHandle;
static PIN_State buttonPinState;
static PIN_State ledPinState;

//SPI//////////////////////////////////SPI//////////////////////////////////SPI
#define SPI_MAX_BYTE_TRANSACTION      4
static SPI_Handle spi_master_handle;
static SPI_Params spi_master_param;
static SPI_Transaction spiTransaction;
char transmitBufferPointer[SPI_MAX_BYTE_TRANSACTION];
char receiveBufferPointer[SPI_MAX_BYTE_TRANSACTION];
char init_mex_SPI[SPI_MAX_BYTE_TRANSACTION]="abcd";
bool ret;
//SPI//////////////////////////////////SPI//////////////////////////////////SPI

struct active_node_known{
    uint8_t address;
    uint8_t latest_RSSI;
    bool state;
    ///altri parametri da memorizzare del nodo
};
static struct active_node_known nodes[10];
int8_t rssi_recived;

PIN_Config pinTable[] = {
    NODE_ACTIVITY_LED | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL |
    PIN_DRVSTR_MAX,
    Board_LED1 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL |
    PIN_DRVSTR_MAX,
    Board_LED4 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL |
    PIN_DRVSTR_MAX,
```

```
        Board_LED3 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL |
        PIN_DRVSTR_MAX,
    PIN_TERMINATE
};

PIN_Config buttonPinTable[] = {
    Board_BUTTON0 | PIN_INPUT_EN | PIN_PULLUP | PIN_IRQ_NEGEDGE,
    PIN_TERMINATE
};

/***** Prototypes *****/
static void masterTaskFunction(UArg arg0, UArg arg1);
//void fastReportTimeoutCallback(UArg arg0);
//void adcCallback(uint16_t adcValue);
//void buttonCallback(PIN_Handle handle, PIN_Id pinId);
static void packetReceivedCallback(union MasterPacket* packet, int8_t rssi);
static void spiInizializzazioni_callback(void);
static void spiInizializzazioni_blocking(int8_t address);
static void spiCallback(SPI_Handle handle, SPI_Transaction *Trans);
SPI_Config spi_configuration;
SPI_FxnTable spi_table;

/***** Function definitions *****/

void MasterTask_init(void)
{
    /* Create event used internally for state changes */
    Event_Params eventParam;
    Event_Params_init(&eventParam);
    Event_construct(&masterTaskEvent, &eventParam);
    masterEventHandle = Event_handle(&masterTaskEvent);

    /*
    // Create clock object which is used for fast report time-out
    Clock_Params clkParams;
    clkParams.period = 0;
    clkParams.startFlag = FALSE;
    Clock_construct(&fastReportTimeoutClock, fastReportTimeoutCallback, 1, &
        clkParams);
    fastReportTimeoutClockHandle = Clock_handle(&fastReportTimeoutClock);
    */
    /* Create the node task */
    Task_Params_init(&masterTaskParams);
    masterTaskParams.stackSize = MASTER_TASK_STACK_SIZE;
    masterTaskParams.priority = MASTER_TASK_PRIORITY;
    masterTaskParams.stack = &masterTaskStack;
    Task_construct(&masterTask, masterTaskFunction, &masterTaskParams, NULL);
    System_printf("B2_masterTask_inizializzato\n");
    System_flush();
}

static void masterTaskFunction(UArg arg0, UArg arg1)
{
    /* Open LED pins */
    ledPinHandle = PIN_open(&ledPinState, pinTable);
    if (!ledPinHandle)
    {
        System_abort("Error_inizializing_board_3.3V_domain_pins\n");
    }
    PIN_setOutputValue(ledPinHandle, Board_LED4, 1);
}
```

```
int j;
for (j = 0; j < MAX_NODES; j++)
{
    nodes[j].address = 0x00;///azzerare tutti i nodi memorizzati, avviene solo
    con un reset o un riavvio
}

///SPI////////////////////////////////////////SPI
spiInizializzazioni_callback();
///SPI////////////////////////////////////////SPI

    System_printf("B2_masterTask░masterTaskFunction░n");
    System_flush();

/*DA ELIMINARE
// setup timeout for fast report timeout
Clock_setTimeout(fastReportTimeoutClockHandle,
    NODE_ADCTASK_REPORTINTERVAL_FAST_DURATION_MS * 1000 / Clock_tickPeriod);
// start fast report and timeout
Clock_start(fastReportTimeoutClockHandle);
*/

buttonPinHandle = PIN_open(&buttonPinState, buttonPinTable);
if (!buttonPinHandle)
{
    System_abort("Error░initializing░button░pins░n");
}

MasterRadioTask_registerPacketReceivedCallback(packetReceivedCallback);///
connect the pointers to receive a command from the radio task

while(1) {
    /* Wait for event */
    uint32_t eventsTASK = Event_pend(masterEventHandle, 0, MASTER_EVENT_ALL,
        BIOS_WAIT_FOREVER);

    /* If new command, send this command */
    if (eventsTASK & MASTER_EVENT_NEW_COMMAND_VALUE) {

        /* Toggle activity LED */
        PIN_setOutputValue(ledPinHandle, NODE_ACTIVITY_LED, !PIN_getOutputValue
            (NODE_ACTIVITY_LED));
        System_printf("B2_masterTask░while░Evento░NEW_ADC_VALUE░n");
        System_flush();///make print

        /* Send command value to nodo */ ///give the command to the radio task
        MasterRadioTask_sendCommandData(latestCommandValue);
    }
    if(eventsTASK & MASTER_EVENT_NEW_DATA_VALUE){

        if(isKnownNodeAddress(dato_ricevuto_daNodo.header.
            sourceAddress)){
            updateNode(&dato_ricevuto_daNodo);
        }else{
            addNewNode(&dato_ricevuto_daNodo);
        }

        System_printf("B2_masterTask░TASK ,░RICEVUTI░DAL░NODO░0x%02x░n",
            dato_ricevuto_daNodo.adcValue);
        System_flush();
    }
}
```



```
    }
}

static void packetReceivedCallback(union MasterPacket* packet, int8_t rssi)
{
    rssi_recived=rssi;
    /* If we received an ADC sensor packet, for backward compatibility */
    if (packet->header.packetType == RADIO_PACKET_TYPE_DATA_PACKET)
    {
        dato_ricevuto_daNodo.adcValue = packet->rxCallBack_datiNodo.adcValue ;
        dato_ricevuto_daNodo.header.sourceAddress = packet->rxCallBack_datiNodo.
            header.sourceAddress;
        Event_post(masterEventHandle, MASTER_EVENT_NEW_DATA_VALUE);
    }
}

void notify_toTask_addNode(int8_t address){

    //SPI_close(spi_master_handle);

    System_printf("B2_masterTask_notify_toTask_addNode: 0x%02x\n",address);
    System_flush();

    //spiInizializzazioni_blocking(address);

    SPI_close(spi_master_handle);

    spiInizializzazioni_callback();
return;

}

void notify_node_notRespond(int8_t address){

    uint8_t i;
    for (i = 0; i < MAX_NODES; i++)
    {
        if (nodes[i].address == address)
        {
            nodes[i].state = 0; //1 tutto ok, 0 non risponde
            System_printf("B2_masterTask_node_not_responding add:0x%02
                x,status:0x%02x\n", nodes[i].address,nodes[i].state)
                ;
            System_flush();
        }
    }
return;
}

static uint8_t isKnownNodeAddress(uint8_t address) {

    uint8_t found = 0;
    uint8_t i;
    for (i = 0; i < MAX_NODES; i++)
    {
        System_printf("B2_masterTask_nodi_conosciuti: 0x%02x, last_rssi 0x%02x,
            status:0x%02x\n", nodes[i].address,nodes[i].latest_RSSI,nodes[i].
            state);
        System_flush();
        if (nodes[i].address == address)
        {
            found = 1;
            break;
        }
    }
}
```

```
    }
    return found;
}

static void updateNode(struct NodeDataPacket* node) {
    uint8_t i;
    for (i = 0; i < MAX_NODES; i++) {
        if (nodes[i].address == node->header.sourceAddress)
        {
            nodes[i].latest_RSSI = rssi_recived;
            nodes[i].state = 1; //1 tutto ok, 0 non risponde
            break;
        }
    }
}

static void addNewNode(struct NodeDataPacket* node) {
    uint8_t i;
    System_printf("B2_masterTask add new node to vector");
    System_flush();
    for (i = 0; i < MAX_NODES; i++)
    {
        if (nodes[i].address == 0x00)
        {
            nodes[i].address=node->header.sourceAddress;
            nodes[i].latest_RSSI = rssi_recived;
            nodes[i].state = 1; //1 tutto ok, 0 non risponde
            break;
        }
    }
}

//SPI//////////////////////////////////SPI//////////////////////////////////SPI
void spiInizializzazioni_blocking(int8_t address){
    char buf_inviare;

    transmitBufferPointer[0]=(char)address;
    transmitBufferPointer[1]='s';
    transmitBufferPointer[2]='t';
    transmitBufferPointer[3]='u';

    SPI_Params_init(&spi_master_param);
    spi_master_param.frameFormat = SPI_POL1_PHA1;
    spi_master_param.mode = SPI_SLAVE;
    spi_master_param.transferMode = SPI_MODE_BLOCKING;
    //spi_master_param.transferTimeout = 0x0010;
    SPI_init();

    spi_master_handle=SPI_open(Board_SPI0 , &spi_master_param);
    if (!spi_master_handle){
        System_printf("SPI did not open");
    }

    System_printf("B2_masterTask spiInizializzazioni_blocking: 0x%02x\n",
        address);
    System_flush();

    spiTransaction.count = SPI_MAX_BYTE_TRANSACTION;
    spiTransaction.txBuf = transmitBufferPointer;
    spiTransaction.rxBuf = NULL;
    SPI_transfer(spi_master_handle , &spiTransaction);
}
```

```
static void spiInizializzazioni_callback(){

    /*Defaults values are:
    transferMode      = SPI_MODE_BLOCKING
    transferTimeout   = SPI_WAIT_FOREVER
    transferCallbackFxn = NULL
    mode              = SPI_MASTER
    bitRate           = 1000000 (Hz)
    dataSize          = 8 (bits)
    frameFormat       = SPI_POL0_PHA0
    */
    SPI_Params_init(&spi_master_param);
    spi_master_param.frameFormat = SPI_POL1_PHA1;
    spi_master_param.mode = SPI_SLAVE;
    spi_master_param.transferMode = SPI_MODE_CALLBACK;
    spi_master_param.transferCallbackFxn = spiCallback;
    SPI_init();

    spi_master_handle=SPI_open(Board_SPI0 , &spi_master_param);
    if (!spi_master_handle){
        System_printf("SPI did not open");
    }

    spiTransaction.count = SPI_MAX_BYTE_TRANSACTION;
    spiTransaction.txBuf = NULL;
    spiTransaction.rxBuf = receiveBufferPointer;
    SPI_transfer(spi_master_handle , &spiTransaction);
}

static void spiCallback(SPI_Handle handle , SPI_Transaction *Trans){

    ret=SPI_transfer(handle , Trans);
    if (!ret) {
        System_printf("Unsuccessful SPI transfer");
        System_flush();
        receiveBufferPointer
    }
    latestCommandValue =receiveBufferPointer[0];
    Event_post(masterEventHandle , MASTER_EVENT_NEW_COMMAND_VALUE);
    System_printf("SPI Callback , received %s\n", spiTransaction.rxBuf);
    System_flush();
}

//SPI//////////////////////////////////SPI//////////////////////////////////SPI
```

A.8 B2_master_TASK_v2.0.h

```
#ifndef TASKS_NODETASK_H_
#define TASKS_NODETASK_H_
/* Initializes the Node Task and creates all TI-RTOS objects */
void MasterTask_init(void);
#endif /* TASKS_NODETASK_H_ */
```