UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

DISEÑO E IMPLEMENTACIÓN DE UNA PASARELA DE COMUNICACIONES ENTRE UN TELÉFONO MÓVIL INTELIGENTE Y UNA RED DE SENSORES INALÁMBRICOS

DESIGN AND IMPLEMENTATION OF A COMMUNICATIONS GATEWAY BETWEEN A SMARTPHONE AND A WIRELESS SENSOR NETWORK

Luis Cavo Núñez

Julio 2015

TRABAJO FIN DE GRADO

Título:	DISEÑO E IMPLEMENTACIÓN DE UNA PASARELA DE COMUNICACIONES ENTRE UN TELÉFONO MÓVIL INTELIGENTE Y UNA RED DE SENSORES INALÁMBRICOS
Autor:	LUIS CAVO NÚÑEZ
Tutor:	ALBA ROZAS CID
Ponente:	ALVARO ARAUJO PINTO
Departamento:	DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

TRIBUNAL

Presidente:	Octavio Nieto-Taladriz García
Vocal:	Georgios Kontaxakis Antoniadis
Secretario:	Alvaro Araujo Pinto
Suplente:	Rubén San Segundo Hernández

CALIFICACIÓN:

Madrid, a de de 2015.

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

DISEÑO E IMPLEMENTACIÓN DE UNA PASARELA DE COMUNICACIONES ENTRE UN TELÉFONO MÓVIL INTELIGENTE Y UNA RED DE SENSORES INALÁMBRICOS

DESIGN AND IMPLEMENTATION OF A COMMUNICATIONS GATEWAY BETWEEN A SMARTPHONE AND A WIRELESS SENSOR NETWORK

Luis Cavo Núñez

Julio 2015

Resumen

En este Trabajo Fin de Grado se ha desarrollado una pasarela de comunicaciones a través de la cual es posible establecer una conexión y comunicarse con una red de sensores inalámbrica utilizando un teléfono móvil inteligente. Para ello, este trabajo consta principalmente de tres partes.

En primer lugar, este TFG cubre la fase de análisis que se ha llevado a cabo previamente al desarrollo del sistema. Durante esta fase se presenta al lector una perspectiva general del sistema, justificándose algunas de las más importantes decisiones que se han tenido en cuenta a lo largo de las siguientes fases.

Posteriormente el trabajo cubre el desarrollo de una aplicación Android para un teléfono móvil inteligente. El teléfono móvil se conecta a un sistema hardware, desarrollado también en este trabajo, a través de USB. Por lo tanto ha sido necesario el diseño e implementación de una aplicación móvil, que recibe el nombre de PGN Network Manager, capaz de realizar dicha tarea.

Tras esto, este TFG cubre las fases de diseño e implementación, tanto hardware como software, del sistema hardware desarrollado, el cuál recibe el nombre de PGN (Portable Gateway Node). Dicho sistema actúa a modo de pasarela hacia una red de sensores inalámbrica. Cuando se recibe un mensaje procedente de la aplicación móvil desarrollada, este dispositivo se encargará de procesar el mensaje recibido y encaminarlo a un nodo de una red de sensores si es necesario. Además, también es capaz de recibir mensajes de la red y encaminarlos hacia el terminal móvil del usuario de este sistema.

Una vez llevadas a cabo las fases de diseño e implementación de la aplicación móvil y del PGN, se ha comprobado el correcto funcionamiento del sistema en su conjunto a través de un escenario de prueba.

Palabras clave: redes de sensores inalámbricas, nodo, nodo pasarela, USB, Android, *smartphone*, aplicación, radio frecuencia, interfaz radio.

Abstract

In this End of Degree Project we have developed a system capable of accessing, managing and communicating with a wireless sensor network using a smartphone. To do so, we have divided this work in three main parts.

The first part of this project provides a general perspective of the different subsystems or modules that have been developed. In this part of the document, we focus on examining the different connections that will be established in this system, and the possible ways of managing the communications that take place over these connections.

Afterwards, the following phase covers the design and implementation of an Android application for a smartphone. The smartphone has to be able to connect to a hardware system, which has also been developed in this project, called the Portable Gateway Node (PGN). The connection to the PGN is done by using the USB interface available in the mobile phone. Therefore, in this phase of the project we cover the development of an Android application, called PGN Network Manager, which is capable of sending and receiving messages over USB.

Furthermore, this project describes the design and implementation phases of the PGN. This hardware system is capable of acting as a gateway from the end user's smartphone to a wireless sensor network. When the PGN receives a message coming from the developed application, this device will process the received message and route it to its destination. In addition, this device is also capable of routing messages received from the network to the user's smartphone.

Finally we have created a test scenario in which the operation of the whole system has been tested. This scenario provides a way of demonstrating the correct operation of the different subsystems developed.

Key words: wireless sensor networks, node, USB, Android, smartphone, application, app, gateway, radio frequency, radio interface.

List of Acronyms

ACM: *Abstract Control Model*

AOSP: *Android Open Source Project*

AP: *Access Point*

API: *Application Programming Interface*

CDC: *Communications Device Class*

CLK: *Clock*

CTS: *Clear To Send*

DC: *Direct Current*

DIP: *Dual In-Line Package*

ED: *End Device*

EEPROM: *Electrically Erasable Programmable Read-Only Memory*

EMI: *Electromagnetic Interference*

ESD: *Electrostatic Discharge*

FET: *Field Effect Transistor*

FTDI: *Future Technology Devices International*

GPIO: *General Purpose Input/Output*

HID: *Human Interface Device*

LDO: *Low-Dropout Regulator*

LED: *Light-Emitting Diode*

MCU: *Microcontroller Unit*

MRFI: *Minimal Radio Frequency Interface*

OS: *Operating System*

PCB: *Printed Circuit Board*

PGN: *Portable Gateway Node*

PHY: *Physical Layer*

RAM: *Random Access Memory*

RF: *Radio Frequency*

RTS: *Request To Send*

RX: *Reception*

SPI: *Serial Peripheral Interface*

TFG: *Trabajo Fin de Grado*

TI:	*Texas Instruments*
TX:	*Transmission*
UART:	*Universal Asynchronous Receiver-Transmitter*
UI:	*User Interface*
USB:	*Universal Serial Bus*
USCI:	*Universal Serial Communications Interface*
WSN:	*Wireless Sensor Network*
XTAL:	*Crystal*

Index

1. Introduction

1.1 Presentation

A wireless sensor network (WSN) is a system of distributed nodes that are responsible for interacting with the environment by capturing information of any kind. This is done using the available sensors, processing the information, and communicating with other nodes to make a decision, which may be taken using actuators. Such communication between nodes takes place in a wireless manner. Applications of these networks typically cover areas such as home (home automation, tele-assistance services), industry (efficient building management), safety and environment (measurement of parameters on burnt land, monitoring earthquakes), among others.

WSNs are one of the main research lines within the B105 - Electronic Systems Lab, a research group part of the Department of Electronic Engineering at the Universidad Politécnica de Madrid.

In this context, the core of this work is to develop a system capable of accessing, managing and communicating with one of these WSNs using a smartphone. Now the question is: why would the usage of a smartphone imply any benefits? Well, the reason for this is that working with a smartphone allows the user to perform management tasks quickly and conveniently, given the portability of these devices, without the need for large equipment and without requiring an external power supply. Furthermore, as we see in [1] the penetration of smartphones in countries like the United States, the United Kingdom, Germany or Spain is close to two thirds of the countries' population, making the results provided by this project accessible to a wide range of potential users.

The described scenario is represented in the following figure:

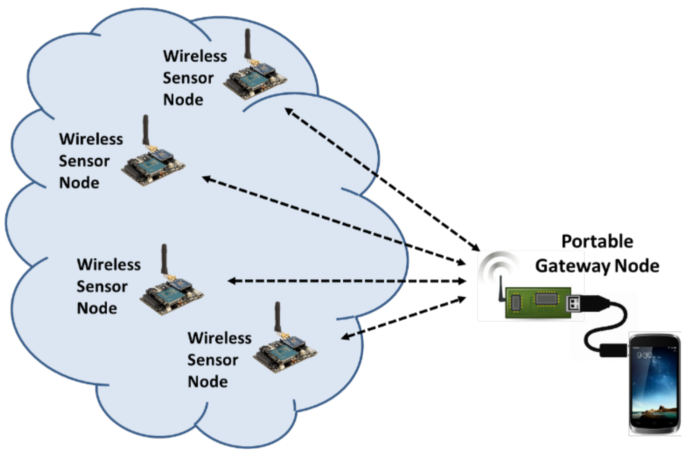


Figure 1 PGN use case scenario

As we observe in the previous figure, we have named the hardware system that we are going to develop Portable Gateway Node (or PGN from now on). By connecting a smartphone to the PGN, the smartphone acts as an interface to the end user or network manager, which can send and receive messages to or from the sensor network. Therefore, with a portable system like this, the user could determine if the deployment of a WSN has been done successfully, and it permits to carry out diagnostics or network configuration, all this in a semi-remote way. In addition, thanks to the wireless broadband connections available in smartphones, this system could be used as a gateway to transfer the acquired information over the internet.

1.2 Objectives

The main objective of this work is therefore to **design and implement a communications gateway between a smartphone and a wireless sensor network**. This objective implies several different tasks, which are presented below:

- Training: study of the necessary documentation to become familiar with application programming for the Android operating system (OS) and microcontroller programming. Get familiar with all the necessary programs that will be used throughout the project, as well as previous studies on the topic that have been conducted in the B105 lab.
- Design of the electronic system called PGN. This node must offer a Universal Serial Bus (USB) interface to connect to the smartphone as well as a radio frequency (RF) interface to be able to communicate with a WSN.
- Implementation of the PGN. The printed circuit board (PCB) layout will be designed, and all the necessary components will be soldered.
- Design and implementation of the PGN's firmware. This program will capture the data coming from the smartphone or RF interface, process this incoming data and reply using the corresponding interface.
- Design and implementation of an application for the smartphone. This application controls the PGN in order to establish the communication with a WSN, and will also enable the user to visualize the network.
- Testing: The PGN's hardware will be tested in different milestones established during the soldering phase and at the conclusion of this phase, to check the correct operation of the whole system. The PGN's firmware and the Android application will also be tested to verify their correct operation. Finally the whole system will be tested in a real scenario.

1.3 Project Structure

Section 2 presents an analysis of the project. On section 2.1 we will first give a general perspective of the whole system. Afterwards, section 2.2 will carry out a brief analysis concerning the advantages of using the Android OS in this project. Section 2.3 will analyze the main elements that compose the PGN.

Section 3 covers the design and implementation of the Android application. In section 3.1 the design is covered whereas in section 3.2 we will pass onto the implementation of the application.

Section 4 covers the design and implementation of the PGN. Section 4.1 will cover the design of the PGN's hardware and firmware. Section 4.2 will cover the implementation phase of the PGN.

Section 5 presents a test scenario in which we will test the correct system operation. In addition, this test scenario will be useful to demonstrate how the system works.

Section 6 presents the conclusions drawn and the future work lines.

Section 7 details the bibliography.

2. Analysis

In this section, we are first going to present a general perspective of the system as a whole. We will start with a brief analysis of the operating systems available when developing an application for a mobile phone. Afterwards, we will analyze in detail the different options we have considered when designing the PGN.

For this reason, some finished projects of the B105 lab have been taken into consideration. In addition, some commercial options have been useful in determining some of the guidelines followed during this project.

2.1 System Overview

The following figure shows the complete system with its main components and the main tasks that are going to be performed:

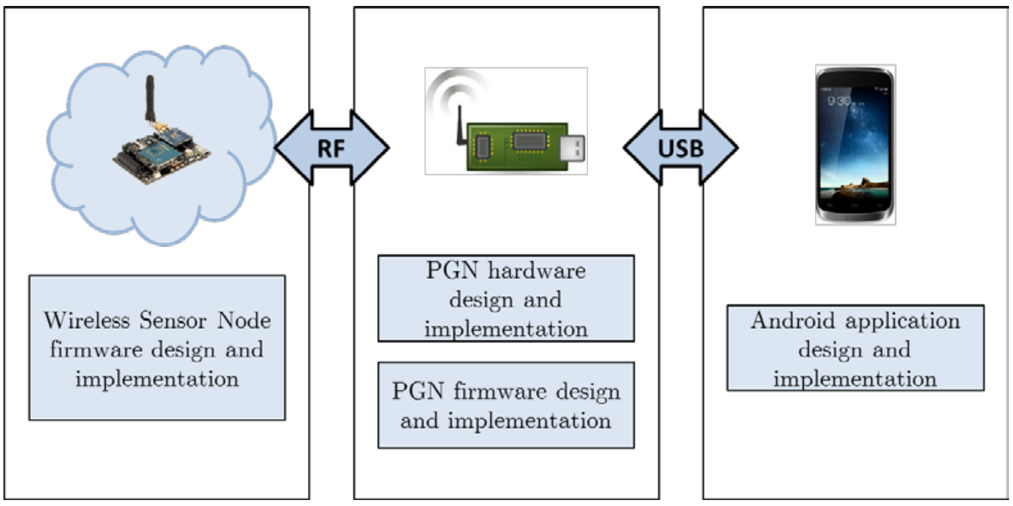

RF
USB
Wireless Sensor Node firmware design and implementation
PGN hardware design and implementation
PGN firmware design and implementation
Android application design and implementation

Figure 2 Project overview

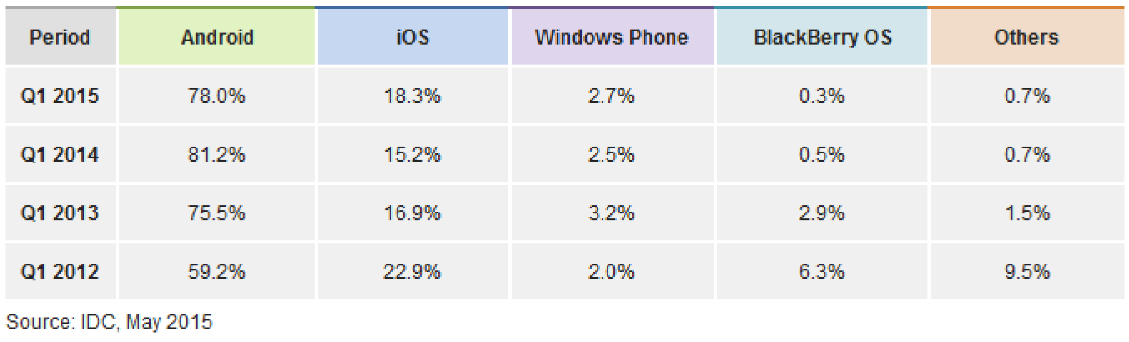
The main block in this project is the PGN, in the center of Figure 2. The PGN consists in a system that provides the necessary hardware and software to communicate with a WSN as well as the end user's smartphone.

The Android application is aimed at providing a user interface (UI) for the user to manage the PGN and, in this way, communicate with a WSN. This application is executed in a mobile device, having the benefits already mentioned in section 1.1.

Finally, an existing wireless sensor node available in the B105 lab will be programmed to enable a communication with such device. The firmware of this WSN will be modified to incorporate our newly developed communications protocol.

2.2 Android Platform

The four main operating systems for mobile phones are Android, iOS, Windows Phone and Blackberry OS. The next figure, taken from [2] shows the market share for each of these mobile operating systems:



Period	Android	iOS	Windows Phone	BlackBerry OS	Others
Q1 2015	78.0%	18.3%	2.7%	0.3%	0.7%
Q1 2014	81.2%	15.2%	2.5%	0.5%	0.7%
Q1 2013	75.5%	16.9%	3.2%	2.9%	1.5%
Q1 2012	59.2%	22.9%	2.0%	6.3%	9.5%

Source: IDC, May 2015

Table 1 Mobile phone OS market share

These operating systems have already been studied in other research projects in the B105 lab. In [3] we can find one of these analyses, which concludes that Android OS seems to be the most adequate platform for developing low-level applications. The following reasons are mentioned in that project:

1. Android is Open Source, and encourages third-party applications. In addition, developers may access the source code of Android native applications by means of the AOSP (Android Open Source Project) [4].
2. Market share: Android dominated the market with a 78% market share in the first quarter of 2015. This means that the vast majority of mobile phone owners could use this system by just installing the application that is going to be developed.
3. Low level programming: using the Android OS we can work directly with the device's USB interface, without the need of rooting the device.

For these reasons, the Android operating system has been selected for developing our application.

2.3 Portable Gateway Node

It is necessary to define some of the requirements before entering the design phase of the project. In the first place, we have clarified what is the objective of the project, and have introduced its main components. Now we will determine what this system has to be capable of by defining some requirements or requisites.

From a user's perspective, this system has to provide a way of connecting to a WSN using his or her smartphone device and therefore it has to be able to provide the following physical interfaces:

- A USB interface to communicate with the smartphone.
- A radio frequency interface to communicate with a wireless node.

Beside these interfaces, a microcontroller unit (MCU) must be used to acquire the incoming data and perform the necessary processing.

The following picture represents these modules:

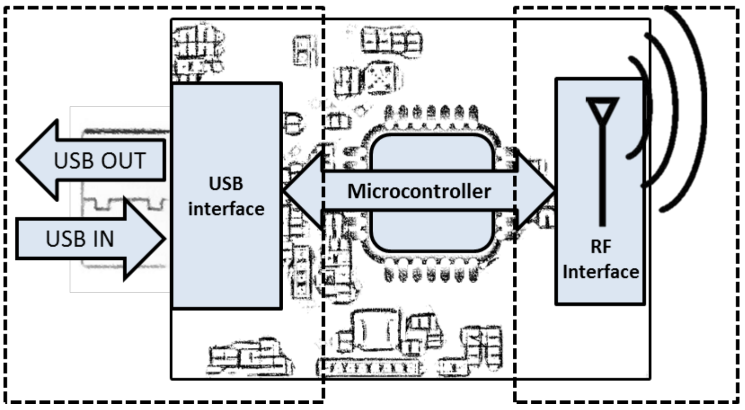


Figure 3 PGN communication interfaces

A great amount of projects have been executed in the B105 lab with the combination of an MSP430 microcontroller and a RF transceiver from the same manufacturer: Texas Instruments (TI). This provides the opportunity to test our design with a finished project in the laboratory. Therefore, using these components would facilitate the testing phase of the system, making the design useful right away.

To keep the design modular and simple, an adequate division of the elements has to be made. These blocks will then be described in more detail in the subsequent sections. The PGN has been divided into the following modules:

- A USB module that allows a physical interconnection between the mobile device and the PGN.
- A microcontroller module.
- A radio frequency module, consisting of a RF transceiver which allows the PGN to send and receive messages over the RF interface.
- A power management module, which converts the 5 V DC voltage from the USB line to the required voltage for the system. This voltage will be defined by analyzing the operating conditions for each selected component.

2.3.1 Universal Serial Bus Interface

For any clarification regarding the concepts related to the Universal Serial Bus it is recommended to refer to the USB 2.0 Specification [5].

The first step in designing the USB interface module consists in studying all related projects carried out in the B105 lab as well as searching for any commercial solutions, like experimenter boards, that could help in designing the system.

Commercial products from Texas Instruments will be analyzed because it is the manufacturer selected for the MCU as well as the RF transceiver.

The first design studied was the MSP-EXP430G2 LaunchPad [6] from TI, which we happened to have already available. This inexpensive evaluation kit features an integrated DIP target socket that allows an MSP430G2xx value line series of microcontrollers to be plugged into the LaunchPad.

The following figure represents the solution provided by TI in the MSP-EXP430G2 LaunchPad. In this approach, the USB line is connected to a USB-to-UART protocol converter, which is a device that provides bridging between a USB port and a Universal Serial Asynchronous Receiver-Transmitter (UART) serial port. This component is then connected to the MSP430 target's device UART port. A voltage regulator is used to convert the 5 V input provided by the USB interface to a lower-level voltage, which consists of a 3.6 V voltage in this solution.

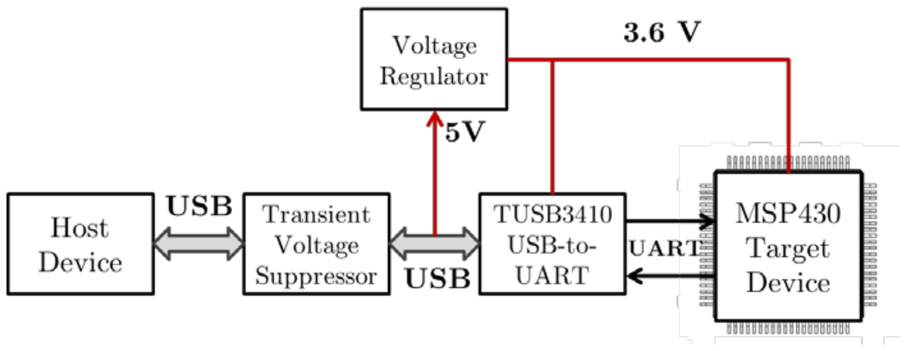


Figure 4 MSP-EXP430G2 USB-to-UART solution

In addition, another LaunchPad from Texas Instruments was studied: the MSP-EXP430F5529LP. This development kit, quite similar to the previous one, features a MSP430F5529 MCU.

The interesting part about these development kits is that they provide an application or "backchannel" UART, which provides communication with a USB Host via the USB interface available in the board. The following figure, taken from the EXP430F5529LP User's Guide [7] illustrates how this application UART works:

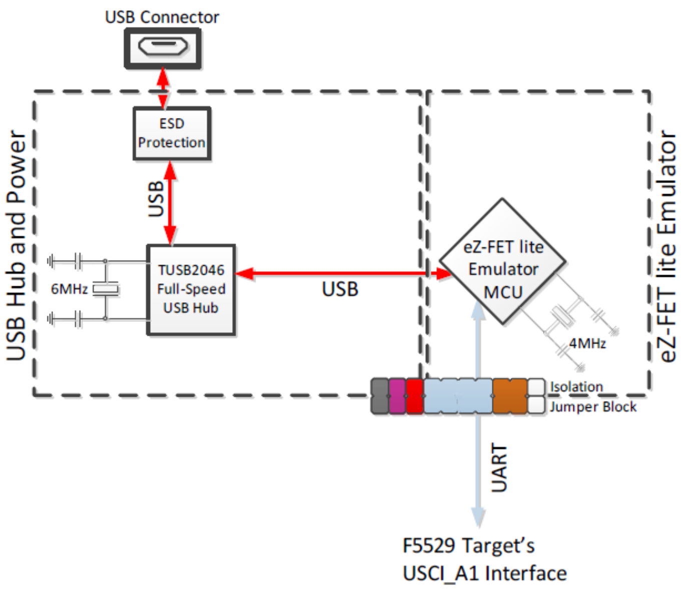


Figure 5 MSP430F5529LP backchannel UART pathway

In the MSP-EXP430G2 LaunchPad, this is the only way to communicate over the USB interface with the microcontroller. On the other hand, the MSP430F5529 MCU, available in the EXP430F5529LP evaluation kit, also provides an integrated USB capability. Thus, this evaluation kit is able to communicate with the microcontroller via one of these paths, or even communicating through both data communication paths.

This scenario is shown on the following figure taken from the User's Guide:

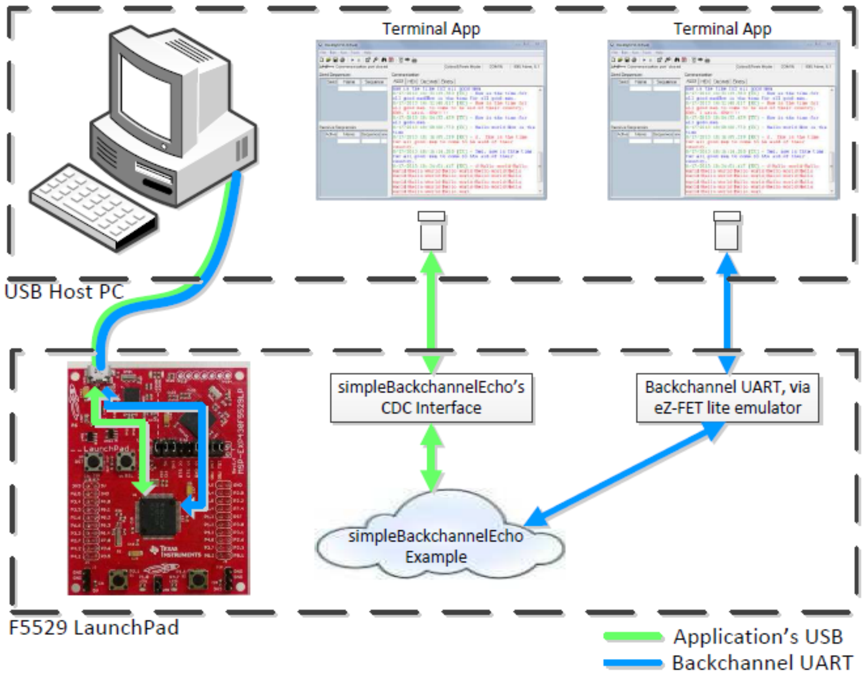

Terminal App
Terminal App
USB Host PC
simpleBackchannelEcho's CDC Interface
Backchannel UART, via eZ-FET lite emulator
simpleBackchannelEcho Example
F5529 LaunchPad
Application's USB
Backchannel UART

Figure 6 Two different USB communication paths

The flexibility offered by this last solution led towards the usage of a similar approach in the design phase. However, it is important to notice that selecting a full USB implementation requires integrating its corresponding driver into the RF network protocol stack. This issue will be dealt with in section 4.1.

2.3.2 Microcontroller

The microcontroller family studied was the MSP430 family of ultra-low-power microcontrollers. This family is optimized to achieve extended battery life in portable applications. Although the latter is not a critical design requirement given that it will not be a battery-powered device, it is important to remark that the PGN will drain its power from the smartphone's battery. Being the battery duration one of the most important and limiting design considerations for mobile phone designers, it is important to make the design as energy efficient as possible. Furthermore, this family of MCUs has been widely used in the B105 lab, achieving outstanding results in the projects developed with this MCU, and for this reason an MSP430 will be used in this system.

2.3.3 Radio Frequency Transceiver

The RF transceiver will handle the communication with the wireless nodes that form the network. There are many transceivers available from many manufacturers, but we have focused our interest on the CC1101 from Texas Instruments.

This sub-1GHz RF transceiver has been used in many research projects executed in the B105 lab in conjunction with a MSP430 low power MCU, providing excellent results. TI provides several reference designs and evaluation modules that feature a MSP430 MCU and a CC1101 RF module.

2.3.4 Network protocol

The RF communication capability is offered by the transceiver module. We have identified two ways of handling the communication between the PGN and another wireless device:

- Implement a custom network protocol and ensure the compatibility of this protocol with the selected hardware.
- Implement SimpliciTI's protocol stack on our PGN.

We are going to analyze the second option to examine the capabilities of this Texas Instruments simple RF network protocol.

2.3.4.1 SimpliciTI overview

SimpliciTI is a simple low-power RF network protocol aimed at small RF networks, developed and offered by Texas Instruments. This protocol is designed to provide the customer with a simple way to implement and deploy a network. From the customer's development perspective the application programming interface (API) provides the means to initialize a network, link devices, and send messages.

According to [11], SimpliciTI's basic stack supports the following roles:

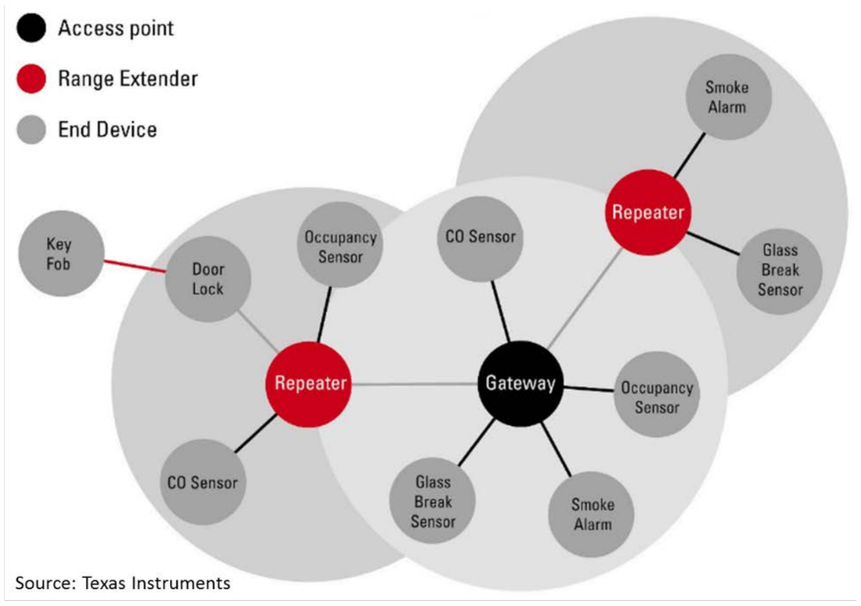


Figure 7 Example network scenario

- **Access Point (AP):** this object is in charge of the network management, which consists of functions such as store-and-forward functionality for sleeping devices, encryption key management, and frequency agility management. Only one AP per network is permitted.

- **End Device (ED):** they are the locus of most of the sensors/actuators in the network.
- **Range Extender:** these devices are intended to extend the radio range of the network. Range extenders replay all the received frames, this way expanding the radio range of the frame sender.

2.3.4.1.1 Network discipline with AP

The first action takes place during the initialization of SimpliciTI, and is a side effect of the initialization process itself. During this action, the node that wants to join a network sends a join message containing a join token to an AP. If this join token matches the token established in the AP (which is established at build time), then the AP responds with the necessary information to successfully establish a connection with the joining device. This reply message contains a link token, which will be used in the following phase: linking.

Linking is supported by the SimpliciTI API, and it is the phase by which a peer-to-peer connection is established. The link message contains a link token, used by the listener to validate the peer.

The following example scenario, illustrates the join and link actions:

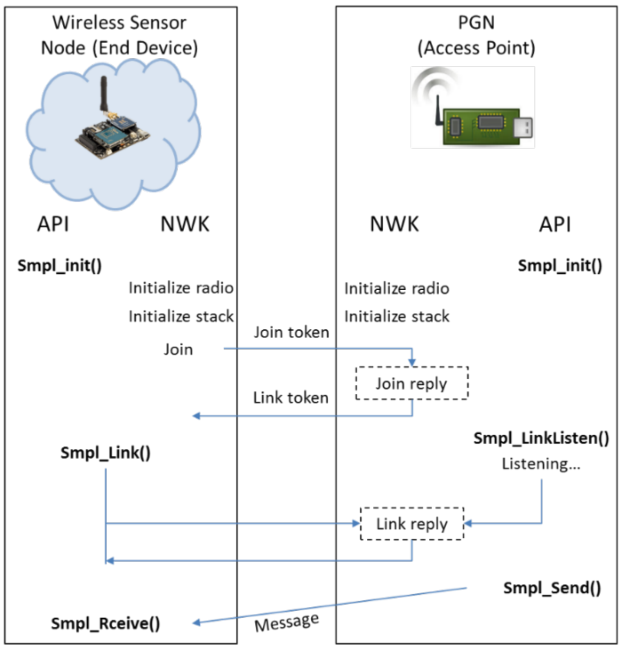


Figure 8 Link and join on AP scenario

To sum up, SimpliciTI is a simple low-power network protocol that provides a lot of flexibility by providing simple star topology networks with three kinds of roles. Furthermore, the five command API makes the implementation of this protocol really manageable.

Another main advantage of using SimpliciTI in this project is that this protocol was designed for easy implementation and out-of-the-box deployment on several TI RF platforms, one of these platforms being the MSP430F5529 USB Experimenter's Board + CC2520EMK. This Experimenter's Board is equipped with the MSP430F5529 MCU and therefore the adaptation of the protocol stack to our design will be much more straightforward.

In conclusion, it is clearly visible that the most versatile and simple solution is to implement the SimpliciTI protocol stack, whereas the first solution presented, developing a custom protocol, would be more time-consuming and would probably provide worse performance. More information on SimpliciTI will be given on Section 4.1.2.

3. Android Application

The design phase of the Android application is covered in section 3.1. In this section the application's design will be described in detail, the requirements will be outlined and then a complete application diagram will be presented. In addition, the elements defining the application will be sketched out.

Section 3.2 covers the implementation of this application. In this section we will select the best way to implement the design considerations outlined in the design phase. The output of this section will be the Android application called PGN Network Manager, used to control the PGN.

3.1 Design

The Android application acts as an interface between the end user and the WSN. The main functionality of the application is to encapsulate user-selected network commands into a particular protocol frame structure following a designer-defined protocol. After this has been done, the application has to manage the sending and reception of messages over/from the USB line.

Before going into deeper detail about the actual structure of the application, it is necessary to define other important requirements and basic functionalities that the application has to offer the user. These functional requirements are:

- The application shall detect a connected USB device and allow the user to establish a connection with this device.
- It shall allow for changes in the USB line parameters such as baud rate or frame format (number of data bits, stop bits, and parity bits).
- It will establish a default line configuration.
- The application must allow the user to send and receive messages over/from the USB interface.
- It must provide a simple communications protocol in order to successfully communicate with the PGN; this protocol is therefore to be designed. The protocol will have some pre-defined commands that will always follow the same structure.

As an addition, we have given a lot of importance to designing the end-user side of the system (the Android application) in a "user-friendly" way. Thus, it will present the data about the number of connected devices to the system, as well as information regarding these nodes in a simple and well-organized manner.

This last point is to a certain degree unconnected to the previous ones because this requirement consists basically on user interface modifications, which Android OS makes really easy to accomplish.

For this reason, we have decided to divide the application's software in two different frameworks or domains: a UI domain designed to make the application "user-friendly", and a functional domain designed to meet the first four requirements.

In addition, the application will be developed in such way that the user does not have to root the device. Rooting is a process that allows the user to attain privileged control (known as root access) over various Android subsystems. Nevertheless, rooting an Android device is an intricate process that not all smartphone owners could be able to carry out, what would have considerably limited this project's penetration had it been necessary.

3.1.1 Architecture

As we have just explained, the application's architecture has been divided into two distinct domains to separate as much as possible the UI domain from the classes that offer control over the USB API.

The UI domain comprehends principally mere layout capabilities or other functionalities that make the application easier to handle by the final user, manage how data is presented to the user or provide better UI performance. On the other hand, the second domain corresponds to a set of classes that will enable a successful communication with the PGN.

The UI domain will likely consist in a screen over which a list of options will be rendered. These options will consist of:

- Selection of a USB device.
- USB communication with a connected device.
- Configuration of the USB line parameters.
- Network Management.

When one of these items on the list is clicked, another screen will appear showing the details of the option selected.

The following scheme shows this architecture:

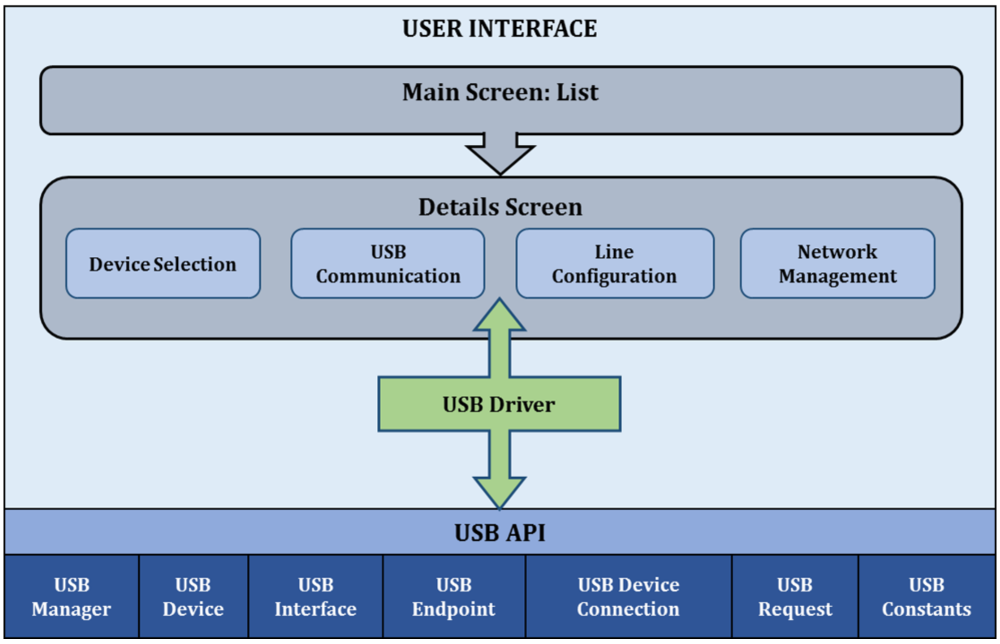


Figure 9 Android application architecture

Additionally, the functional domain will take care of enabling the communication with the Android USB API, and is represented in the previous diagram by the USB Driver module.

3.1.1.1 Device Selection

This UI element will allow the user to select a USB device connected to the smartphone, e.g. the PGN.

3.1.1.2 USB Communication

This element will be capable of sending and receiving messages over the USB line. The design will be similar to a terminal which shows the sent and received data.

3.1.1.3 Line Configuration

This UI will consist of a set of user-selectable parameters that will be used to configure the USB line.

3.1.1.4 Network Management

This UI element will act as the true user interface towards the network. If the PGN has been connected and a network has been created following the guidelines described in section 2.3.4, this UI element will present basic information about the network, and will permit the user to establish a communication with its components.

3.1.1.5 USB Driver

All the UIs that provide access to the USB API will do so by means of the USB Driver module, which will contain the necessary functions and objects to establish a communication with a USB device.

3.2 Implementation

There are many possibilities of implementing the architecture defined for the application. A valid approach would be to implement each of these UI elements by creating an activity associated with each view. However, Android introduced fragments in Android 3.0 (API level 11). Besides providing some UI benefits, the reason for using fragments in this project has been the ease with which fragments handle the pass of information between screens. For any clarification regarding the concepts related to Android it is recommended to refer to Android developers' webpage [12].

With activities the objects passed between the different screens in the application would have to be serialized, and then sent to another activity using an intent [13]. However, by making each screen a separate fragment this data-passing implementation is completely avoided. Since fragments always exist within the activity's context, you can always access that activity and in that way the objects that you have stored in that activity. Another reason for using fragments is that Android is encouraging the use of fragments on its applications.

To sum up, the implementation of the aforementioned screens has been done using 2 activities and 4 fragments. The first and launcher activity is called Layout Activity and only presents the user with a table containing a list of functionalities offered by the application:

- Select a USB device, if this device has been physically connected to the smartphone via a USB cable.

- Send and receive messages over/from the USB line.
- Configure the serial line.
- Initialize a network and establish a communication with this network.

Layout Activity has no other functionality than presenting a simple list of possibilities to the user, and in this manner it is the second activity that takes on greater importance. The second activity, called Details Activity acts as a container for the selected fragment on the list presented in Layout Activity. When an item in this list is selected, Details Activity will be launched, and the fragment that takes care of implementing such functionality will contribute with its own layout to the activity.

In addition to these four UI fragments, one more fragment will be added during the execution of Network Manager Fragment when the PGN has been connected and if a connection to a WSN is available. This fragment will not provide a user interface, and it is designed to provide background behavior for the application. This fragment is called USB Receive Fragment, and it will implement the following worker threads:

- A thread that will manage the reception of data from the USB line.
- A thread that will process the received data.

The reason for creating another fragment for receiving and processing the received data is to provide a background worker that is present regardless of which UI fragment is presented to the user.

Finally, six additional Java classes have been created. These classes will provide the additional intelligence needed in the application, and are called: Node, Access Point, End Device, Network, USB Message and Message Queue. These classes are explained in section 3.2.1.2.

3.2.1.1 USB CDC library

This class takes care of storing USB objects created during the initialization and set up of the connection to a USB device. These objects will be accessed whenever they are required by the application:

```
private UsbDeviceConnection mConnection;
private UsbDevice mDevice;
private UsbInterface mIntf;
private UsbEndpoint mEndPointRead;
private UsbEndpoint mEndPointWrite;
```

All these objects have their respective setters and getters. Besides these objects, this class implements the following functions:

```
public String getConfigurationDescriptor();
private String parseConfigDescriptor(byte[] buffer);

private String nameForClass(int classType);
private String nameForEndpointType(int type);
private String nameForDirection(int direction);

public void setMessage(byte[] mess);
public byte[] getMessage();
public int USBCDC_sendData();
public void setControlLineState();
public void setParameter(int baudRate, int dataBits, int stopBits, int
parity);
public int sendAcmControlMessage(int request_type, int request, int value,
byte[] buf, int delay);
```

3.2.1.2 Network Objects

As we have already mentioned, six Java classes have been used to provide the application with its main functionality. We want the application to be able to identify the PGN as the access point of the network and to identify the devices connected to the PGN as end devices. This is the reason for creating the class Node and its subclasses: Access Point and End Device.

The relationship that exists between these objects is explained with the following diagram. This diagram is expanded in the following sections.

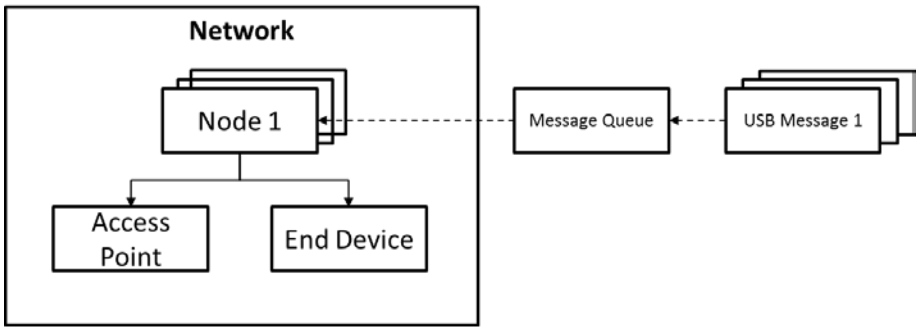


Figure 10 Network objects relationship

3.2.1.2.1 Network Object

A Network is composed of Node objects. Of these, only one is the Access Point of the network, and this role is played by the PGN. The other nodes are End Devices and will be stored in an array of End Devices.

A Network class has the following attributes:

```
private int numberEndDevices;
private AccessPoint PGNode;
private EndDevice[] networkEndDevices;
private boolean bIsNetworkInitialized;
```

And the following functions are implemented in the Network class:

```
public Network(AccessPoint APnode);
public AccessPoint getAccessPoint();
public void setAccessPoint(AccessPoint ap);
public int getNumberEndDevices();
public EndDevice[] getArrayEndDevices();
public void addNodeToNetwork(EndDevice dev);
public void deleteNodeFromNetwork(byte nodeID);
public void terminateNetwork();
public void initializeNetwork();
public boolean isInitialized();
public boolean isNetworkComponent (byte id);
public EndDevice findEndDeviceById (byte id);
```

3.2.1.2.2 Node Object

A Node represents a unique element in the network. The Node class is the superclass of Access Point and End Device, which inherit from Node its methods and attributes. Access Point and End Device will also have their own subclass attributes and methods.

A Node will have the following attributes:

```
private byte linkID;
private int numSensors;
private byte[] sensorsType;
private String typeOfDevice;
private MessageQueue messageQueue;
```

And implement the following functions:

```
public Node (byte id, int nSens, byte[] type, String tod)
public Node (byte id, String tod);
public void setNumSensors(int num);
public int getNumSensors();
public void setLinkID(byte id);
public byte getLinkID();
public void setArraySensorsType(byte[] sens);
public String linkIDtoString();
public String getDeviceName();
public byte[] getArraySensorsType ();
public void addSensor(byte sensType);
public void deleteSensor(int sensorPos);
public MessageQueue getMessageQueue();
```

3.2.1.2.2.1 Access Point Object

An Access Point will define one additional attribute with respect to its parent class Node. This attribute stores the AP's USB descriptor:

```
String descriptor;
```

It will also implement the following subclass methods:

```
public String getUsbDescriptor();
public void setUsbDescriptor(String des);
public String[] getAvailableCommands();
```

3.2.1.2.2.2 End Device Object

The End Device class will not have any additional attributes with respect to its parent class Node, however it does implement the following subclass methods:

```
public String getDeviceDescription();
public String sensorsToString(byte[] sens);
public String[] getAvailableCommands();
```

3.2.1.2.3 Message Queue Object

Each Node object (either an Access Point or an End Device) will have its own Message Queue. A Message Queue is composed of USB Message objects which are just a concatenation of byte arrays.

This object has the following attributes:

```
private int capacity;
private int position;
private int nAvailableMessages;
private UsbMessage[] messages;
private boolean bEmptyBuffer;
```

The Message Queue class also implements the necessary functions to manipulate a Message Queue object (constructor method, add a message, retrieve a message, etc.).

3.2.1.3 Device Selection Fragment

This fragment takes care of setting up the communication with the PGN, and for this reason it must be the first screen that the user opens. To do so, this module should follow the next execution sequence:

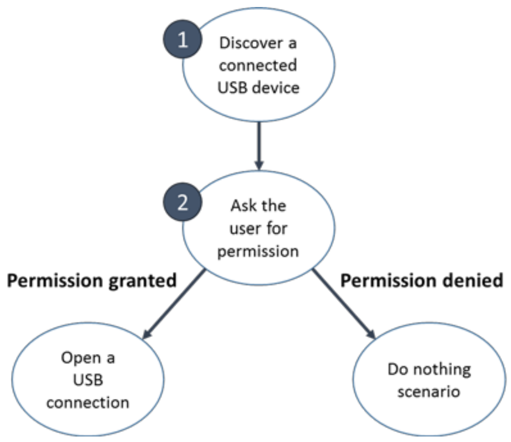


Figure 11 Device Selection execution sequence

1. In the first step, it will be necessary to add an intent filter in the manifest. This intent filter contains an action that "informs" the application if a USB device is attached to the smartphone. If this situation takes place, this intent is captured in Device Selection and a dialogue asking the user for permission will appear.
2. If the user grants permission to access the connected USB device, then a broadcast receiver [14] will notify that this permission has been granted.

When the connection has been granted by the user, a series of events take place. The figure below tries to explain this event queue:

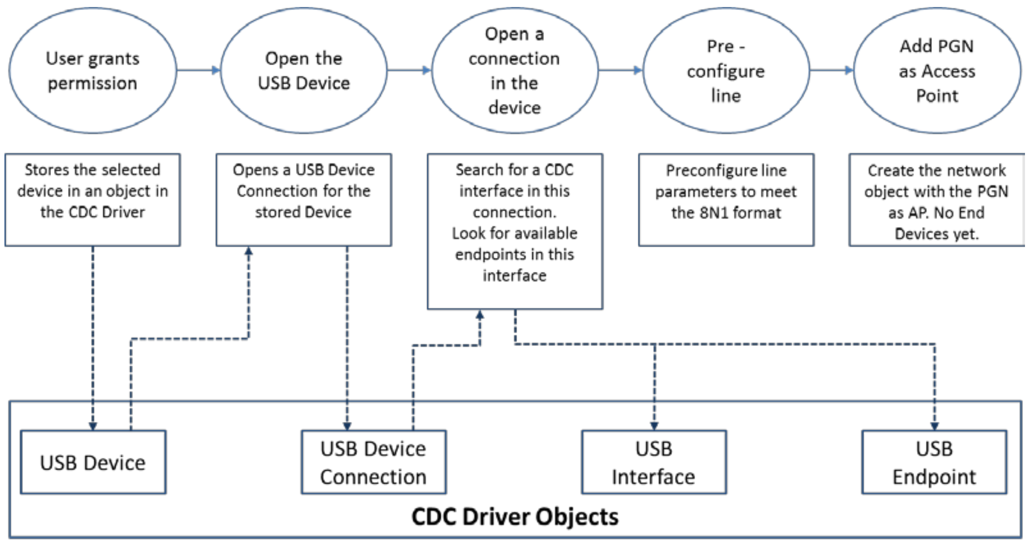

User grants permission
Open the USB Device
Open a connection in the device
Pre - configure line
Add PGN as Access Point
Stores the selected device in an object in the CDC Driver
Opens a USB Device Connection for the stored Device
Search for a CDC interface in this connection. Look for available endpoints in this interface
Preconfigure line parameters to meet the 8N1 format
Create the network object with the PGN as AP. No End Devices yet.
USB Device
USB Device Connection
USB Interface
USB Endpoint
CDC Driver Objects

Figure 12 Device Selection event queue

At the conclusion of this process, the connection with the device will have been established and the necessary USB endpoint and its associated Communications Device Class (CDC) interface to carry out any communication will be available.

3.2.1.4 USB Communication Fragment

This fragment will present a layout to the user consisting of two terminal-like Android Edit Text views. The user will write a message on the sender terminal, and will be able to send this message by pressing a "send" button. If any message is received, this message will be showed in the reception terminal.

In comparison with the Network Manager Fragment, which launches a "background worker" fragment to handle the reception and processing of received frames, this fragment will not launch an additional fragment. Instead, this fragment will just provide a receptor thread that receives data and "pushes" this data onto the screen.

The drawback of this implementation is that the user has to either know or look up which messages can be sent to the PGN so they can be processed. Thus, the frame will have to be coded following certain guidelines.

Another constraint that this implementation has is that it won't process or store any data received, so this should be done by the user.

3.2.1.5 Line Configuration Fragment

Although the line is preconfigured during the connection phase, this fragment allows the user to change the line parameters by selecting the baud rate, stop bits, data bits and parity bits. The way this has been implemented is by selecting these parameters using Android Spinners. Spinners are drop down menus that provide a quick way to select one value from a set. After all the parameters have been set, the user must click on the Configure button to send this configuration message. The preconfigured line parameters are: 9600 baud rate and a frame format of 8 data bits, 1 stop bit and no parity bit. This configuration will match the line configuration established on the PGN side.

3.2.1.6 Network Manager Fragment

This fragment plays the most important role in the actual management of a network. Network Manager is a transformation of the USB Communication Fragment that provides a clearer and more guided way of connecting, sending and receiving messages to/from a network.

With this implementation, the different drawbacks and constraints that USB Communication Fragment has will be avoided. Network Manager will provide the application with the simplest but necessary intelligence to take control over the PGN.

Therefore, any additional improvement of the Android application would more likely be reflected on this fragment.

The following scheme shows this fragment's layout and how it works:

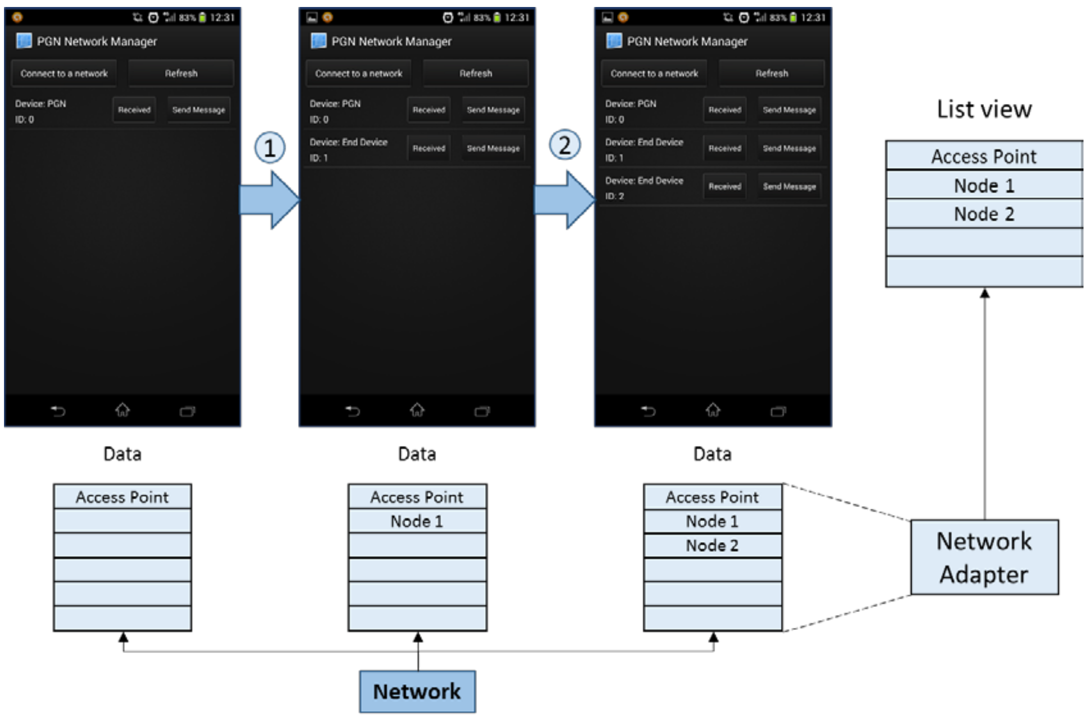


Figure 13 Network manager showing a connection with two EDs

The first image shows the situation in which only the PGN is connected. When the user presses the button "connect to a network", the PGN connects to the existing end devices. The second image represents the situation in which only 1 ED is connected to the PGN. Finally, the image in the right represents the situation in which two EDs are connected to the PGN.

We observe that this process is managed using an array adapter [15] defined in Network Adapter class that transfers the data stored in the Network object into the list view presented to the user.

3.2.1.7 USB Receive Fragment

As we already know, this fragment takes care of receiving and processing the messages received in the USB interface. It creates two different threads for these issues, where the processor thread is the only one active when a message has been received.

We have defined 3 states during the reception of a message. The first state (state = 0) is when the first byte of a message has been received. This byte contains the length of the rest of the message. The second state is when the rest of the protocol frame is being received (state = 1). Finally, when all the payload has been received, the state is updated (state = 2) and the processing thread is launched.

The following scheme shows a typical receive and process situation:

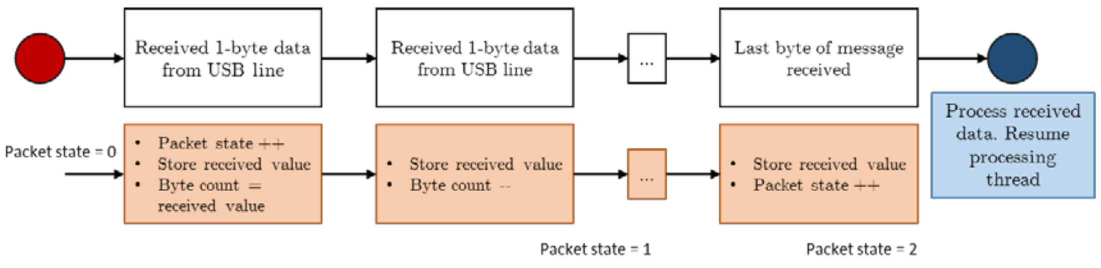

Received 1-byte data from USB line
Received 1-byte data from USB line
...
Last byte of message received
Process received data. Resume processing thread
Packet state = 0
Packet state ++
Store received value
Byte count = received value
Store received value
Byte count --
...
Store received value
Packet state ++
Packet state = 1
Packet state = 2

Figure 14 Receiving a protocol frame

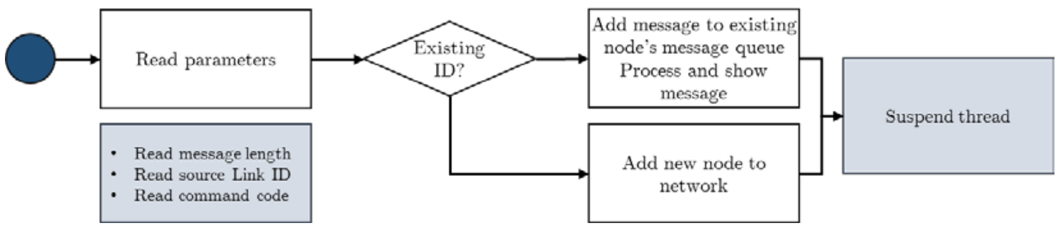


Figure 15 Processing the received protocol frame

4. Portable Gateway Node

4.1 Design

The following sections will cover the design and implementation phases of the PGN. Section 4.1 covers the design phase of this electronic system, differentiating between the hardware design of this system and the firmware design. Section 4.2 covers the implementation of the PGN.

4.1.1 Hardware design

In the analysis section of the PGN we already discussed the modules that this system will present. We will now cover in detail these modules selecting the necessary components to fulfill the system objectives and requirements.

4.1.1.1 Hardware architecture

In the analysis section some questions were left unanswered:

1. The first one is how we are going to treat the received data from the USB line, or in other words if we are going to convert this data to a UART-type serial protocol.
2. The second is what the appropriate voltage supply for our system is, keeping in mind that the USB interface provides 5 V to our circuit.

The first question, raised up at the end of section 2.3.1 is more of a designer's choice, which can be answered by evaluating the advantages and/or disadvantages of using a full USB scheme or converting the USB protocol into a UART-type serial protocol.

Putting together the information presented in [16] [17] and [18] we have created the following table which summarizes the main advantages and disadvantages:

Advantages	Disadvantages
API for MSP430 USB MCUs designed to speed up development time.	Still more intricate implementation than UART or SPI.
More robust protocol.	Need for a Descriptor Tool to generate the necessary USB descriptors in Windows.

Table 2 Integrated USB scheme advantages and disadvantages

Finally, we have decided to provide the design with both possibilities although only one of these solutions will be implemented. For the sake of making the design simpler on the software side, it has been decided to use a USB-to-UART Protocol Converter, which is a device that provides bridging between a USB port and a UART serial port.

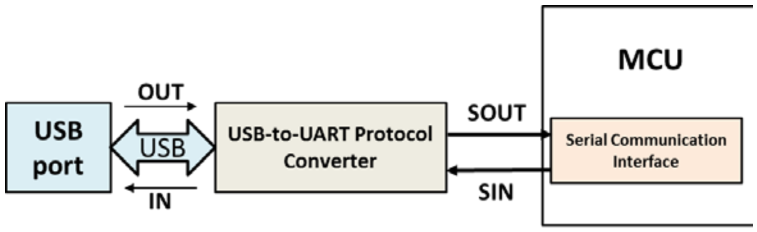

USB port
OUT
USB
IN
USB-to-UART Protocol Converter
SOUT
SIN
MCU
Serial Communication Interface

Figure 16 USB-to-UART protocol converter solution

The hardware will be adapted to support both possibilities, so that the MSP430 USB API could be implemented in the future if necessary. To adapt the hardware to enable both possibilities a USB hub will be used in the same way as the USB hub used in [7].

The second question proposed depends basically on the components used in the circuit and will be studied on section 4.2.1.5.

The following figure shows a preliminary hardware architecture for the PGN. In addition to the main components or modules that make up this system, we also show the main communication interfaces that interconnect each module.

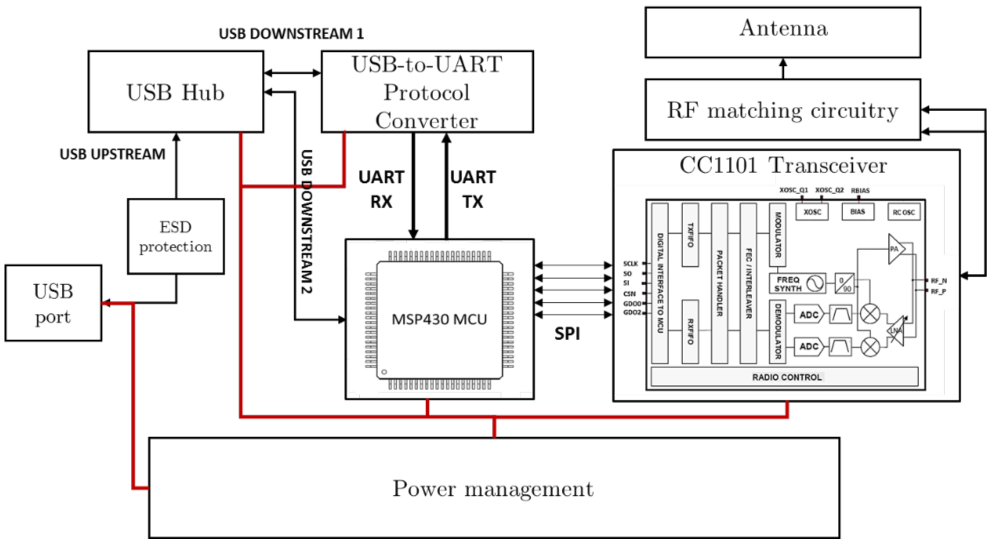


Figure 17 Preliminary PGN architecture

As we see in this figure, only two of these components, the MCU and the RF transceiver were selected during the analysis section (sections 2.3.2 and 2.3.3 respectively). Regarding the MCU, only the microcontroller family to be used (MSP430 family) has been selected. The following sections cover the design of the different PGN modules.

4.1.1.2 Modules

If we recall the module division we did in section 2.3, we will now pass onto the design of the USB hub module and the USB-to-UART protocol converter module (which form the USB module mentioned in section 2.3), the MCU module, which we already know that consists of a MSP430 MCU, the RF module, consisting of a CC1101 transceiver, and the power management module.

4.1.1.2.1 Microcontroller Module

The MCU must satisfy some requisites regarding available interfaces and some other characteristics like the amount of flash or RAM available. The SimpliciTI network protocol uses, according to [10], less than 8 KB of flash and 1 KB of RAM, depending on the configuration used. Besides this, the MCU must also feature a Universal Serial Communication Interface (USCI) module that can operate in both UART mode, to communicate with the smartphone, and an USCI module operating in SPI mode, to communicate with the CC1101 transceiver. The number of MSP430 microcontrollers that match these criteria is very large: 54 ultra-low power MCUs. This number is increased by the 274 low power MCUs that also satisfy these criteria. For this reason, we have considered to review the already studied MCUs as well as the most used MCUs in the B105 lab to check if they meet the requirements. This is shown in the following table:

Requirement	Comment	Purpose	F5529	F2274	G2553
Integrated USB MCU	Mandatory*	Sending messages to app	✓	✗	✗
USCI module – UART mode	Mandatory	Sending messages to app	✓	✓	✓
USCI module – SPI mode	Mandatory	Communicate with CC1101 transceiver	✓	✓	✓
Flash	At least 8KB	SimpliciTI requirement	✓	✓	✓
RAM	At least 1KB	SimpliciTI requirement	✓	✓	✗

Table 3 MCUs comparison

The MSP430F5529 is a great option since it covers all the requirements imposed, being this microcontroller the only one to integrate USB capability directly into the MCU. Likewise, a development kit is available for this device and thus this device is likely to have more documentation, more community support and more example applications to start with than many other MCUs.

In addition, checking out the documentation available for this microcontroller, we found the application report [8] which explains the required hardware, and the necessary software applications for the MSP430-EXP430F5529LP and MSP430-EXP430F5529 boards to work with Android mobile devices.

The applications described in this document are all found in Google Store except for the application TI MSP430 HID, which is a vendor specific HID tool. The source code for this HID application is available at TI webpage [9] and will serve as a useful reference when designing our Android application.

* In section 4.1.1.1 we decided to use an integrated USB MCU to support both solutions, although only the USB-to-UART solution will be implemented.

4.1.1.2.2 Radio Frequency Module

The main features of the CC1101 low-power sub-1 GHz RF transceiver are:

- High sensitivity:
  - o -116 dBm at 0.6 kBaud, 433 MHz, 1% packet error rate
  - o -112 dBm at 1.2 kBaud, 868 MHz, 1% packet error rate
- Low current consumption (14.7 mA in RX, 1.2 kBaud, 868 MHz)
- Excellent receiver selectivity and blocking performance
- Frequency bands: 300-348 MHz, 387-464 MHz and 779-928 MHz

The transceiver will be connected to the MCU through a SPI interface. The following diagram shows the CC1101's typical application circuit:

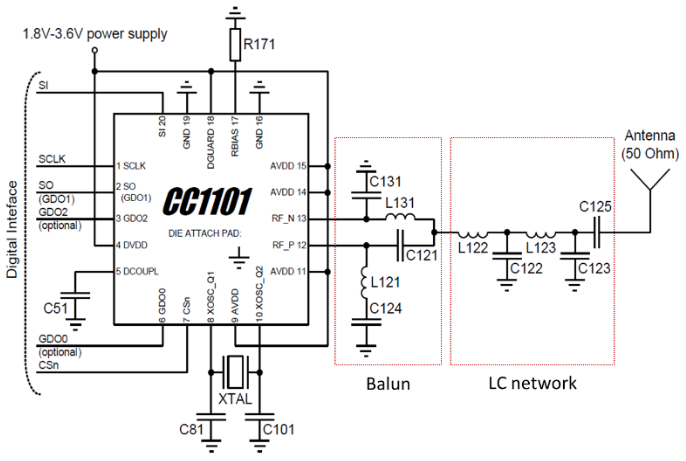
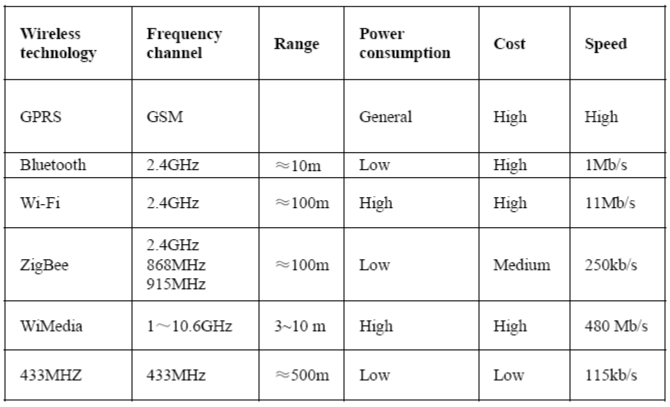


Figure 18 CC1101 application circuit

To select the operating frequency we have based our decision on prior design selections in projects carried out in the B105 lab. In [19] some advantages of using the 433 MHz band for short range wireless communication are presented, being the range provided one of the most important benefits of using the 433 MHz band.

The following table, taken from the same document presents a nice comparison of the 433 MHz, 868-915 MHz and 2.4 GHz bands.



Wireless technology	Frequency channel	Range	Power consumption	Cost	Speed
GPRS	GSM		General	High	High
Bluetooth	2.4GHz	≈10m	Low	High	1Mb/s
Wi-Fi	2.4GHz	≈100m	High	High	11Mb/s
ZigBee	2.4GHz 868MHz 915MHz	≈100m	Low	Medium	250kb/s
WiMedia	1～10.6GHz	3~10 m	High	High	480 Mb/s
433MHZ	433MHz	≈500m	Low	Low	115kb/s

Table 4 Frequency bands comparison

We observe that concerning the range, power consumption, and cost the 433 MHz band provides better performance. Nevertheless, this frequency band provides less speed, but the speed provided is more than enough for our system.

Likewise, the antenna used in our design must operate in the 433 MHz frequency band. The application note [20] describes some important parameters to consider when designing what kind of antenna to use in short range applications. This application note, together with the measurements and results provided in [21] recommend the usage of a helical wire antenna for the 433 MHz band. The technical datasheet of the antenna used is available at [22].

4.1.1.2.3 USB hub Module

Since the idea of using a USB hub was obtained from [7], which uses the TUSB2046 from TI, we decided to use this same product in our design. However, the TUSB2046 is no longer available and has been replaced by the TUSB2046B and so this device has been analyzed. This USB hub's features are:

- Fully compliant with the USB Specification as a full-speed hub.
- 3.3 V powered. Two power source modes available: self-powered and bus-powered.
- 4 downstream ports.
- Power switching and overcurrent reporting is provided ganged or per port.

Our design only needs two downstream ports, so we thought of the possibility of finding a similar device with the only difference of featuring two downstream ports instead of four.

The TUSB2036 is a practically identical device that can provide two or three downstream terminals, selected by input pins as seen on Table 1 of [23]. The features described above are the same for this device except for the number of downstream ports and so, in conclusion, the TUSB2036 will be used as USB hub.

4.1.1.2.4 USB-to-UART Protocol Converter Module

The selection of a USB-to-UART protocol converter has been based on the quality of the documentation available for the device as well as the components characteristics and prize.

Based on these requirements, the following devices have been analyzed:

Manufacturer	Component	Prize/unit (€)	Baud rate (bps)	Buffer (bytes)	Bus powered (V)	Self-powered (V)
FTDI Chip	FT230X	1.94	300 – 3M	512	5	3.3 – 5.25
	FT232RL	4.27	300 – 3M	128 - 256	5	3.3 – 5.25
Microchip	MCP2200	2.11	300 – 1M	64	5	3.3
Silicon Labs	CP2104	1.69	300 – 2M	576	4 – 5.25	3 – 3.6

	CP2110	1.73	300 – 1M	480	4 – 5.25	3 – 3.6
TI	TUSB3410	4.24	50 – 921.6K	64	5	3.6

Table 5 USB-to-UART protocol converter comparison

FT230X from Future Technology Devices International (FTDI): this family of USB-to-UART protocol converters uses a vendor specific USB class. As a consequence, the USB CDC ACM class cannot be used and, instead, a vendor-specific driver provided by FTDI should be modified to meet the application's requisites.

MCP2200 from Microchip: this device implements the USB protocol composite device. This device implements Communications Device Class (CDC) for communications and configuration and Human Interface Device (HID) for I/O control.

CP2104 from Silicon Labs: only implements the USB HID class.

The MCP2200 has been selected as the USB-to-UART Protocol converter. The following figure shows the device's block diagram.

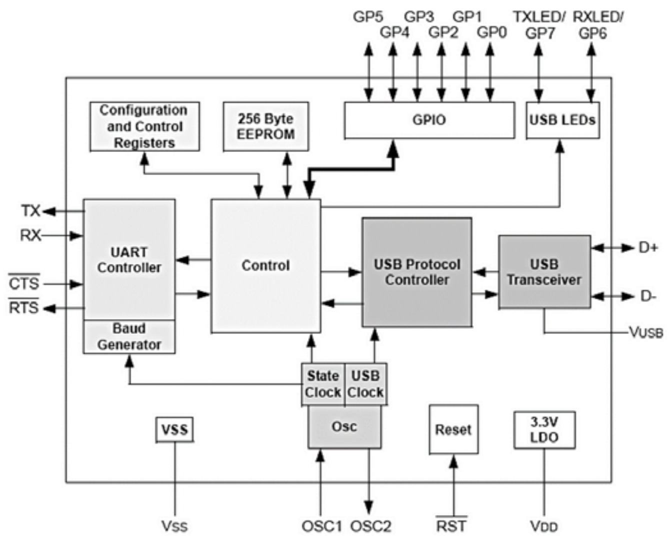


Figure 19 MCP2200 block diagram

The MCP2200 will use an external crystal oscillator, connected to pins OSC1 and OSC2 to provide the proper frequency for the USB module. The UART controller provides the TX, RX, CTS and RTS signals. The hardware flow control signals are not going to be used in this design. The TX signal will be connected to the RX signal in the MCU, whereas the RX signal of this device will be connected to the TX signal in the MCU side.

Additionally, multiple GPIO pins are available. Two of these GPIO, called RxLED and TxLED are used to signal the reception of data coming from the USB host and the transmission of data to the USB host respectively.

The MCP2200 Configuration Utility [24] is used to configure this device. This configuration utility connects to the device's HID interface, and is capable of writing special commands using this interface.

4.1.1.2.5 Power Management Module

The energy consumed by the system is proportional to the voltage the system uses, so in order to achieve higher energy efficiency it will be beneficial to use a power converter that outputs only the required voltage and current for the connected loads. This required voltage and current depend on the components characteristics, provided in their technical datasheet. For instance, it will be necessary to convert the stable 5 V coming from the USB line to a voltage that suits all components operating conditions. The following table shows the minimum, recommended and maximum supply voltages for the ICs selected in the previous sections.

Component	Minimum supply voltage (V)	Nominal supply voltage (V)	Maximum supply voltage (V)
MCU	2.4	-	3.6
USB hub	3	3.3	3.6
RF transceiver	1.8	-	3.6
USB-to-UART converter	3.0	-	5.5

Table 6 Supply voltage on each module

From the previous table, we conclude that the operating voltage has to be one in between 3 V and 3.6 V. A 5 V to 3.3 V regulator is chosen due to the wide availability of this output voltage in commercial voltage regulators.

4.1.1.2.5.1 Voltage regulator selection

A linear regulator controls the supplied output voltage V_0 by continuously adjusting a bipolar or field effect transistor (FET), which operates in its linear mode. Thus, the transistor works as a variable resistor in series with the output load. To adjust this resistor in order to obtain the desired output voltage, a feedback network is used.

A switching mode power supply increases the efficiency operating in switching mode instead of linear mode, where the transistor operates as a high frequency switch. High efficiency, low power dissipation as well as being able to supply a higher output voltage than the input voltage (boost converter) are the main reasons for a designer to user a switching mode power supply. If a switching mode power supply is to be used, then a step down or buck converter has to be included.

The following table gathers the main advantages and disadvantages of using a linear regulator or a buck converter.

Type of regulator	Advantages	Disadvantages
	Simple	Poor efficiency

Lineal	Low cost solution	
DC-DC	High efficiency	More complex solution

Table 7 Voltage regulator comparison

The conversion efficiency obtained by a linear regulator in a 5 V to 3.3 V conversion is shown in the following table. The results are obtained using a Low Dropout Voltage (LDO) linear regulator from Microchip, the MCP1700. The technical datasheet for this component [25] provides the necessary data for the calculations. The results are compared with those obtained for a switching mode power supply, the TPS62203 from TI [26].

Component - Manufacturer	Prize($)	Approximate efficiency (%)
MCP1700 - Microchip	0.3	30 - 35
TPS62203 - TI	0.9	90

Table 8 Prize and efficiencies for selected regulators

To calculate the required output current for the regulator, the maximum current consumption for each IC has been obtained from their respective datasheets, obtaining this way the worst case current consumption, which corresponds to the case in which all ICs will consume the maximum rated current.

Component - Manufacturer	**Typical current consumption**	**Maximum current consumption**	**Total (maximum)**
MSP430F5529 - TI	4.6 mA[*]	-	163.8 mA
TUSB2036 – TI	40 mA	-	
CC1101 - TI	16 mA	29.2 mA	
MCP2200 - Microchip	-	90 mA	

Table 9 Approximate system consumption

The MCP1700 power dissipation, calculated using equation 6.1 of the component's datasheet [25]: P_{LDO} = (5-3.3) × 164 = 278.8 mW, giving an approximate efficiency (P_{IN}/P_{OUT}) of 34%.

The TPS62203 step-down converter datasheet indicates an efficiency of 90% with an input voltage of 5 V and an output current of 200 mA. Therefore, with a 3.3 V output voltage the power dissipation of this step-down converter is 73 mW.

Consequently, the TPS62203 step-down voltage regulator is chosen as the power regulator for the system.

[*] Active mode, f_{MCLK} =12 MHz, V_{CC} =3 V.

4.1.2 Firmware design

As we have seen throughout this document, the microcontroller module has to be able to communicate with two other modules, the transceiver module and the USB module, in order to establish the bidirectional communication, which goes end to end from the user's smartphone to a network's node.

Thus, besides implementing the SimpliciTI protocol stack, it will be necessary to implement a UART driver as well.

The SimpliciTI network protocol stack, which is compatible with the selected MCU and transceiver, has been chosen for this system. Hence, the first step we are going to take is to port the SimpliciTI code of the evaluation boards available in the SimpliciTI compliant protocol stack to the hardware of our PGN.

The experimenter board chosen as a starting point is the MSP-EXP430F5529, which has been chosen considering it features the same MCU – MSP430F5529 – that is included in the PGN.

The guidelines to port the SimpliciTI protocol stack are available at [27]. This porting process will be detailed in the implementation chapter.

SimpliciTI does not strictly follow the OSI reference model as a result of not implementing a formal PHY or Data Link Layer. Conceptually, SimpliciTI supports the following 3 layers:

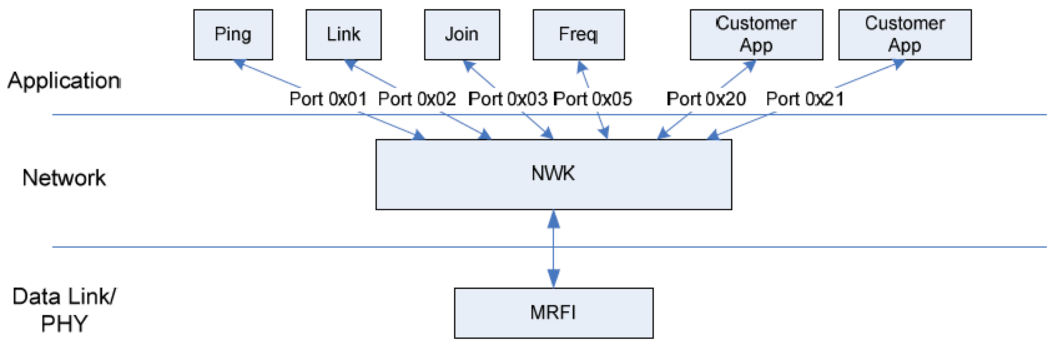


Figure 20 SimpliciTI layers

4.1.2.1 Data Link/PHY layer

The data is received directly from the radio already framed, and therefore the radio performs the framing functions. The Minimal RF Interface (MRFI) is a layer that abstracts whatever support is necessary to interact with the radio. Thus, different radios require different implementations but the basic interface offered is the same for all RF transceivers.

4.1.2.2 Application Layer

The SimpliciTI API offers the following functions:

```
smplStatus_t SMPL_Link (linkID_t *linkID);
smplStatus_t SMPL_UnLink (linkID_t linkID);
smplStatus_t SMPL_LinkListen(linkID_t *linkID);
smplStatus_t SMPL_SendOpt (linkID_t lid, uint8_t *msg, uint8_t len, uint8_t opt);
smplStatus_t SMPL_Send (linkID_t lid, uint8_t *msg, uint8_t len);
smplStatus_t SMPL_Receive (linkID_t lid, uint8_t *msg, uint8_t *len);
```

Using the last three functions above, it will be possible to perform the communication within the network once the links have been established and the directions have been assigned.

As we saw in section 2.3.4, SimpliciTI aims at providing a simple implementation and deployment of small wireless networks. The radio and network configuration is encapsulated and hidden from the application layer. It is driven by build-time configuration assisted by a tool such as Smart-RF Studio.

The network configuration can be changed from the default configuration by modifying the join and link tokens. These tokens are used to establish a connection between an ED and an AP, both ED's and AP's join and link tokens have to be the same in order to successfully establish a connection. These tokens are changed so the EDs used in the test scenario only connect to the PGN, and not to any other AP available in the surroundings.

As we said in the beginning of 4.1.2, apart from the basic transmit-receive functionality it is necessary to provide the application layer with the following functionalities:

1. Introduce a UART driver based on the driver supplied in Design Note DN117. Develop UART reception and transmission handling routines.
2. Develop network transmission and reception routines, by using the available API commands SMPL_Send and SMPL_Receive.
3. Develop processing routines to route the incoming data from the UART or network to its destination executing the necessary processing.

The messages sent from the End User's Android application have to be "translated" in some way into RF messages. It will be necessary to define a frame structure for each of the previous links: the USB and RF links.

This frame structure will be slightly different for each of the aforementioned links. This is mainly due to the fact that the SimpliciTI protocol sends and receives messages without the need to send the length of the message encapsulated in the frame. Besides this, the link ID of an end device communicating with the Access Point is passed as a parameter in the callback of the reception routine and therefore it will always be accessible.

We can observe this difference in the following figure:

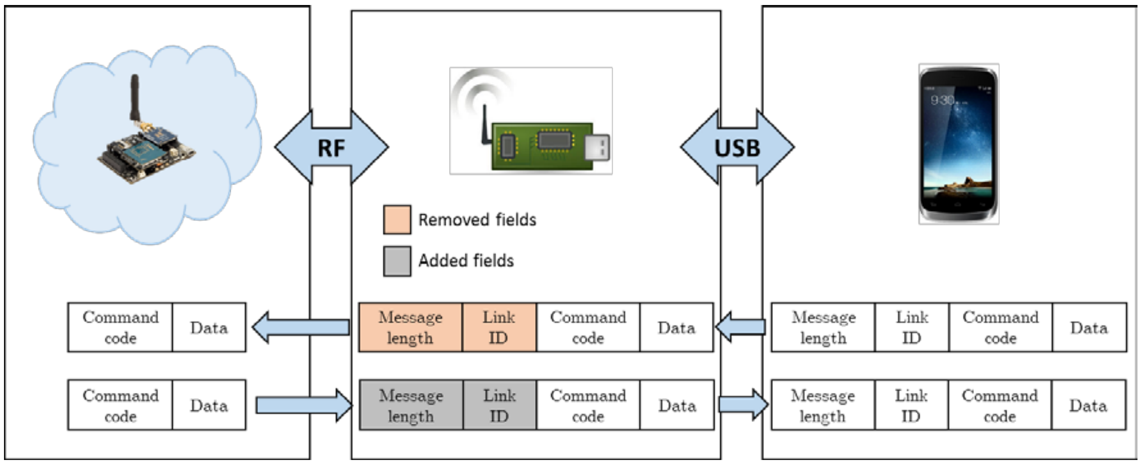


Figure 21 Frame structure in communication paths

4.1.2.3 Application frame structure

From the previous figure, we observe that the frame structure selected follows the scheme:

LENGTH	Link ID	COMMAND CODE	DATA
1 byte	1 byte	1 byte	X bytes
Frame length in bytes	Link ID of the destination device	Command code	Data payload

Table 10 Frame structure

There are two types of devices available in a network, which are called Access Point and End Device. To communicate with the Access Point we define the next commands:

COMMAND CODE	LINK ID	DESCRIPTION
0x00	PGN ID	Initialize a network connection with EDs
0x01	PGN ID	Get a description from EDs

Table 11 Command codes for an AP

The following commands are defined to communicate with an end device:

COMMAND CODE	LINK ID	DESCRIPTION
0x00	Device ID	Get the ED description message
0x01	Device ID	Get device's sensor 1 data
0x02	Device ID	Get device's sensor 2 data

...	Device ID	...
0x0A	Device ID	Toggle LED 1
0x0B	Device ID	Toggle LED 2

Table 12 Command codes for an ED

Thus, putting together what we observe in Figure 21 and Table 11, the frame structure of a *get ED description message* reply received by the Android application is as follows:

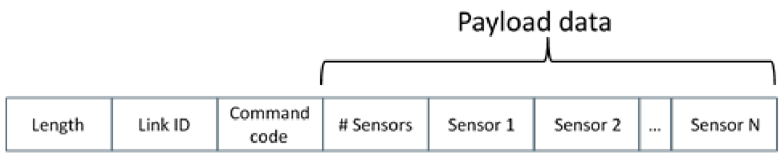


Figure 22 Get ED description response message

The command code will be 0x00 indicating that it is a response to a “Get description” command. The link ID will be the one corresponding to the End Device that is sending the response. This link ID is added by the PGN before sending the message to the application. The payload consists of the number of sensors available in the node as well as a byte for each sensor indicating the type of sensor.

4.2 Implementation

At this point, we can differentiate, like we did in the design phase, two different tasks. The first task covered is the hardware implementation of the PGN. The PCB layout and a picture of the PGN is seen on section 4.2.2.

Finally, the software implementation of the PGN is covered, as well as the implementation of a simple program to run on a generic node. This node will be used in the test scenario of this project.

4.2.1 Hardware Implementation

We have now designed the PGN to cover all the functionalities. The device now hosts the capabilities for computation, power supply, and connectivity using USB and a RF transceiver in the 433MHz band.

The communication with the Android mobile phone can be done either using the MCU’s direct USB capability or by the MCU’s USCI module in UART mode. The UART module will exchange data with a MCP2200 USB-to-UART protocol converter, which also hosts a USB interface that will be connected to the micro USB port through the TUSB2036 USB HUB.

The communication with the CC1101 RF transceiver is done using the USCI module in SPI mode. Power supply takes place through the USB interface’s 5 V line, which will then be converted to a 3.3 V stable voltage necessary for the system to operate in best conditions.

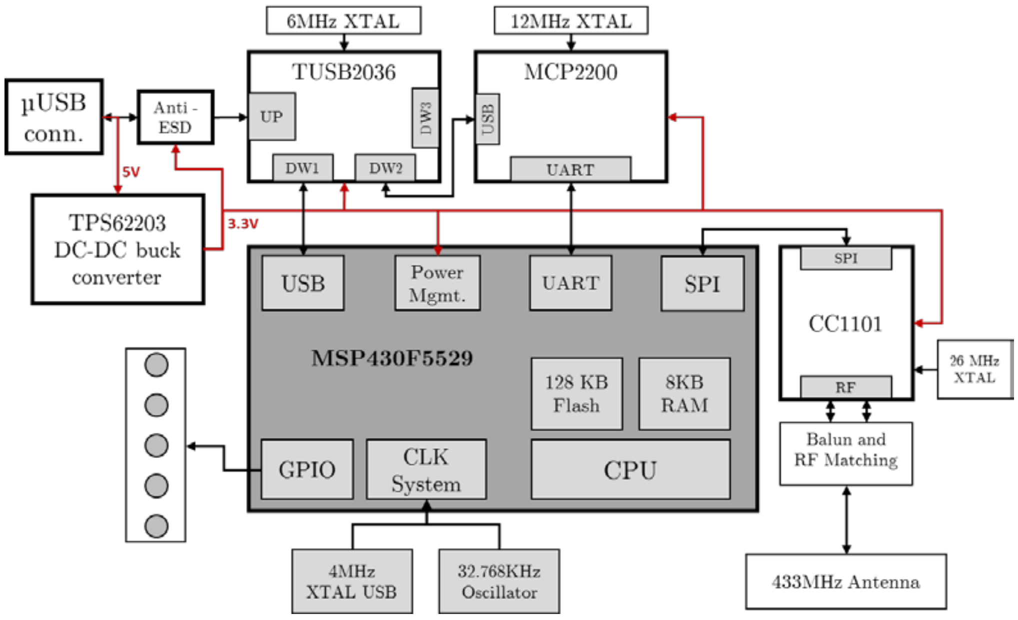


Figure 23 Architecture diagram of the PGN

4.2.1.1 Microcontroller Module

The connections for the MCU module have been done following the hardware design files provided for the MSP430F5529LP [29] as well as section 4 of TI's application report "Starting USB Design using MSP430 MCUs" [28], which provides a reference design for the MSP430 USB module.

We have created the following schematic file:

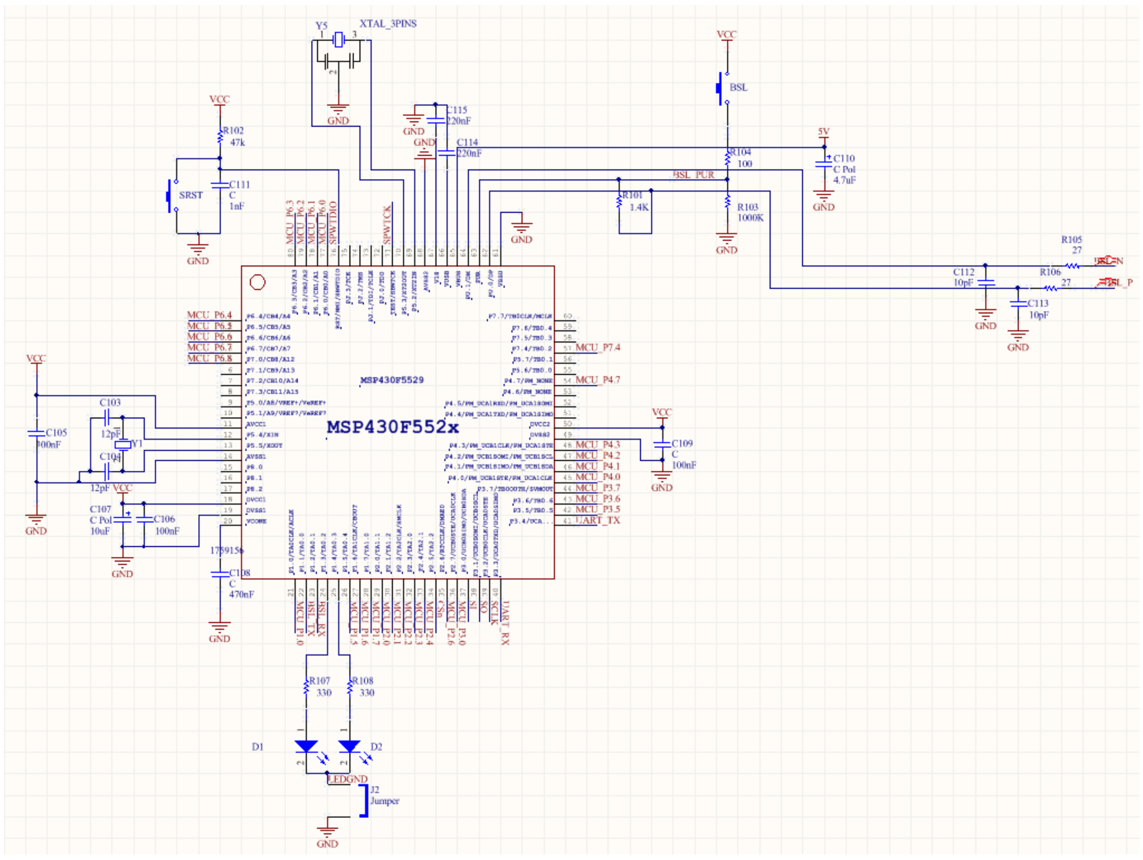


Figure 24 MSP430F5529 schematic

4.2.1.2 Radio Frequency Module

The CC1101 technical data datasheet [30] provides an application circuit and a bill of materials for this circuit operating in the 315/433 MHz frequency range. Thus, the transceiver module has been implemented following these recommendations.

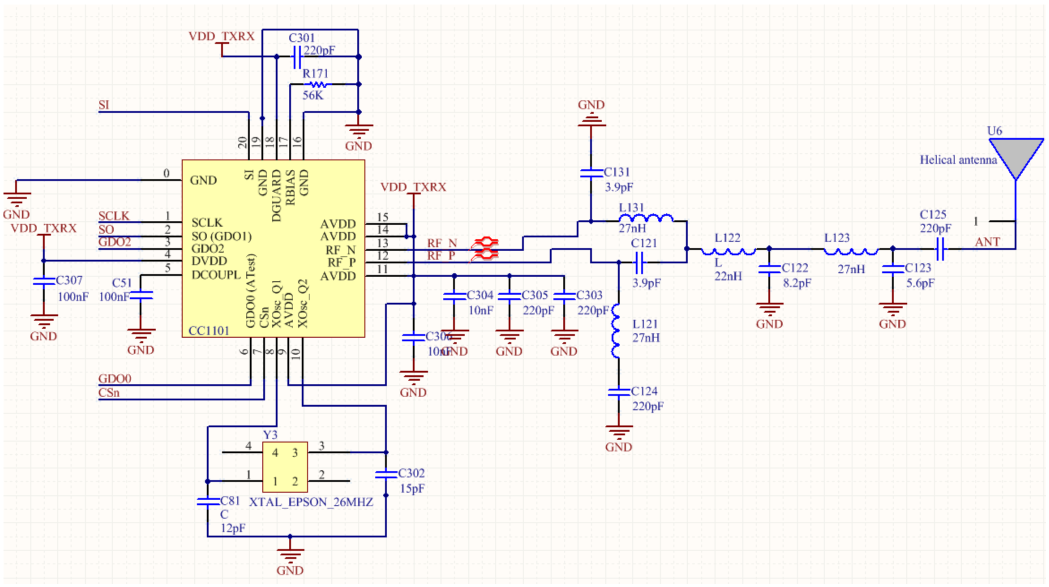


Figure 25 CC1101 schematic

The only difference with the circuit provided in the CC1101's datasheet [30] is the value of the crystal loading capacitors C81 and C302. According to section 7.3 of this document, the crystal loading capacitors value must satisfy:

$$C_L = \frac{1}{\frac{1}{C_{81}} + \frac{1}{C_{302}}} + C_{parasitic}$$

Where the typically parasitic capacitance is 2.5 pF. With the values provided by the component's datasheet, this equation turns $C_L = 16$ pF. With the values used in our design, the value is $C_L = 9.2$ pF which is closer to the crystal's load capacitance of 10 pF. The crystal oscillator selected is a 26 MHz crystal as recommended in section 4.4 of the CC1101's technical datasheet.

4.2.1.3 USB Hub Module

The implementation of the USB hub has been done carefully following the component's datasheet [23], the design guidelines for an evaluation module of the TUSB2036 [32], and a FAQ application report for TI USB hubs [31]. The schematic for the USB hub module is as follows:

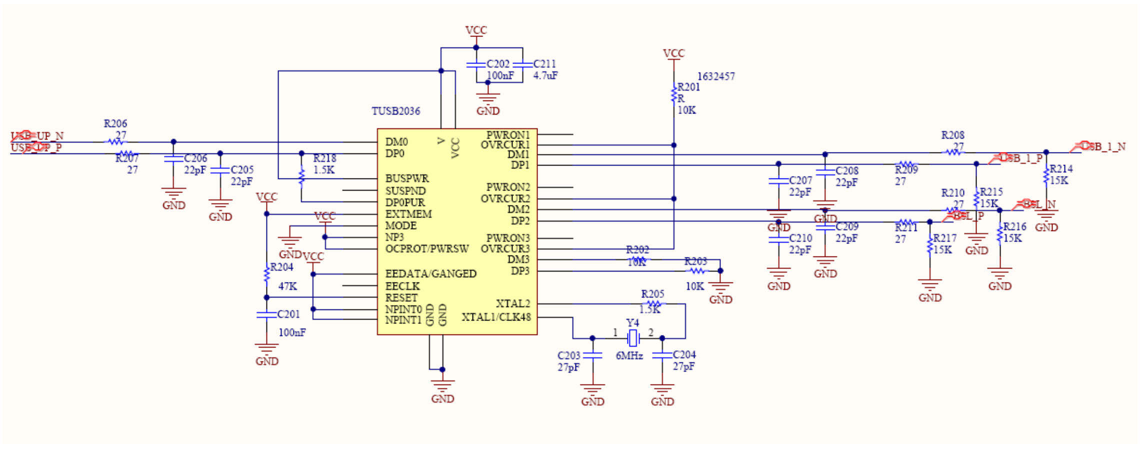


Figure 26 TUSB2036 schematic

4.2.1.4 USB-to-UART Protocol Converter Module

The design of this module is very straightforward following the design considerations of the component's technical datasheet [33]. The configuration utility [24] will be used to configure the UART parameters to: 9600 baud rate, 8 data bits, 1 stop bit and no parity bit, matching the UART configuration established for the Android application in section 3.2.

The MCP2200 will be powered from a 3.3 V external power supply, and for this reason the following scheme has to be followed:

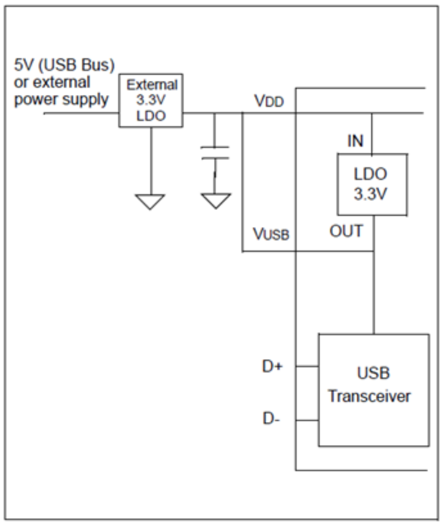


Figure 27 MCP2200 power from external supply

In addition, the recommended 12MHz crystal oscillator is a Murata CSTCE12M0G15L. The CTS and RTS pins of the device are left open so no hardware flow control will be used. The RST pin is tied to V_{DD} using a 1 kΩ to 10 kΩ resistor as indicated in [33], selecting a 4.7 kΩ resistor for this matter.

In conclusion, the MCP200 will have the following schematic:

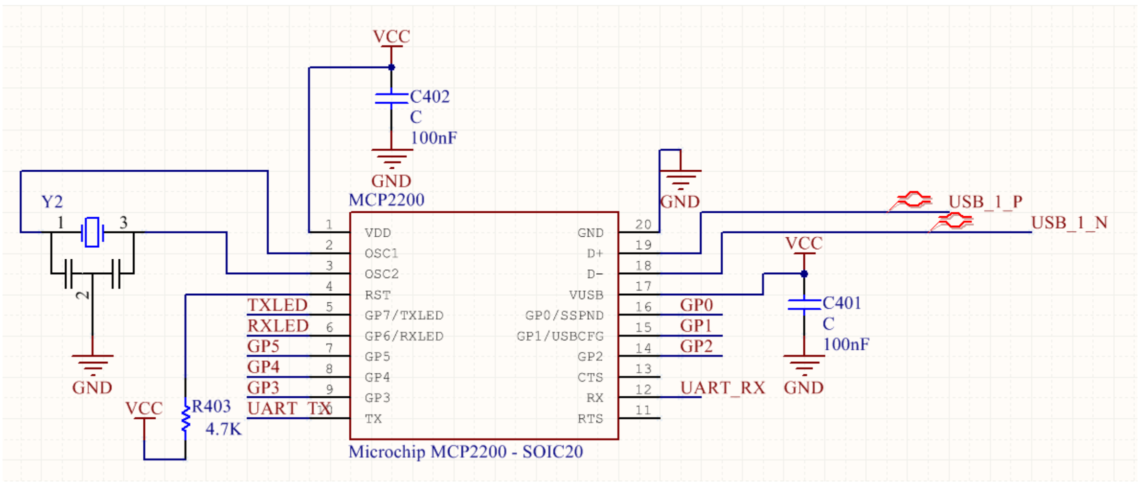


Figure 28 MCP2200 schematic

4.2.1.5 Power Management Module

This module is composed of the TPS62203 plus the necessary components to generate the required 3.3 V stable supply. To implement this design, the design guidelines in [26] have been followed. Besides this document, the TPS62203EVM-211 User's Guide [34] has been followed.

These component values have been compared with the component selection section in its technical datasheet. This component is optimized to operate with a fixed inductor value of 10 μH. The input capacitor will have a value of 4.7 μF for good input voltage filtering and must be positioned as close as possible to the V_{in} input. A low-ESR ceramic or tantalum capacitor is recommended as the output capacitor. The recommended capacitors are shown in Table 2 of [26].

Following these considerations, we designed the schematic below. We can observe that there are two additional 100 nF bypass capacitors, located in parallel to the input and output capacitors respectively.

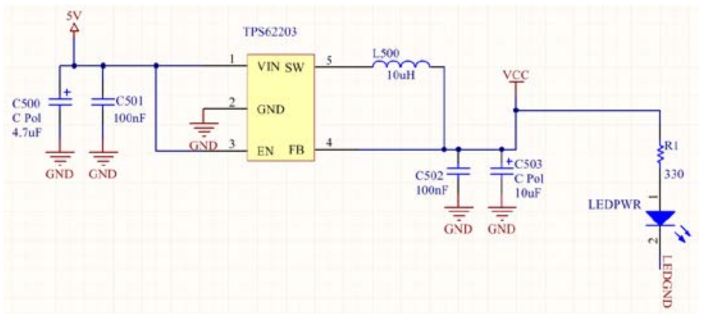


Figure 29 TPS62203 schematic

4.2.2 PGN Layout and Prototype

With the schematics presented in the previous section we have designed the following layout:

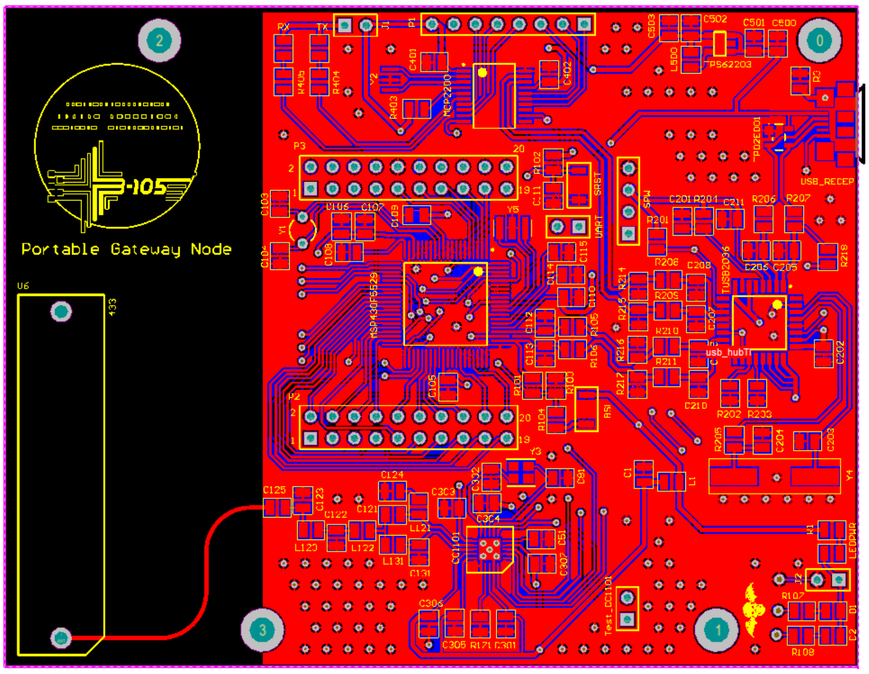


Figure 30 PGN layout

One of the most important considerations concerning the PGN's layout is to follow the USB specification when routing the USB differential pair. All these differential pairs require series resistors of approximately 27 Ω to ensure proper termination. Filter capacitors (which are optional) have been added for EMI suppression following section 7.1.6 of the USB specification. Another critical section in this layout is the RF transceiver. The layout of the balun and RF filtering components has to be carefully designed to have a 50 Ω line impedance in capacitor C125, which is connected to the antenna through a line with 50 Ω of characteristic impedance.

Finally, in the next picture we observe the PGN once the PCB was manufactured and all the components were mounted. A 3D box has been designed using Tinkercad [35] and printed using a 3D printer.

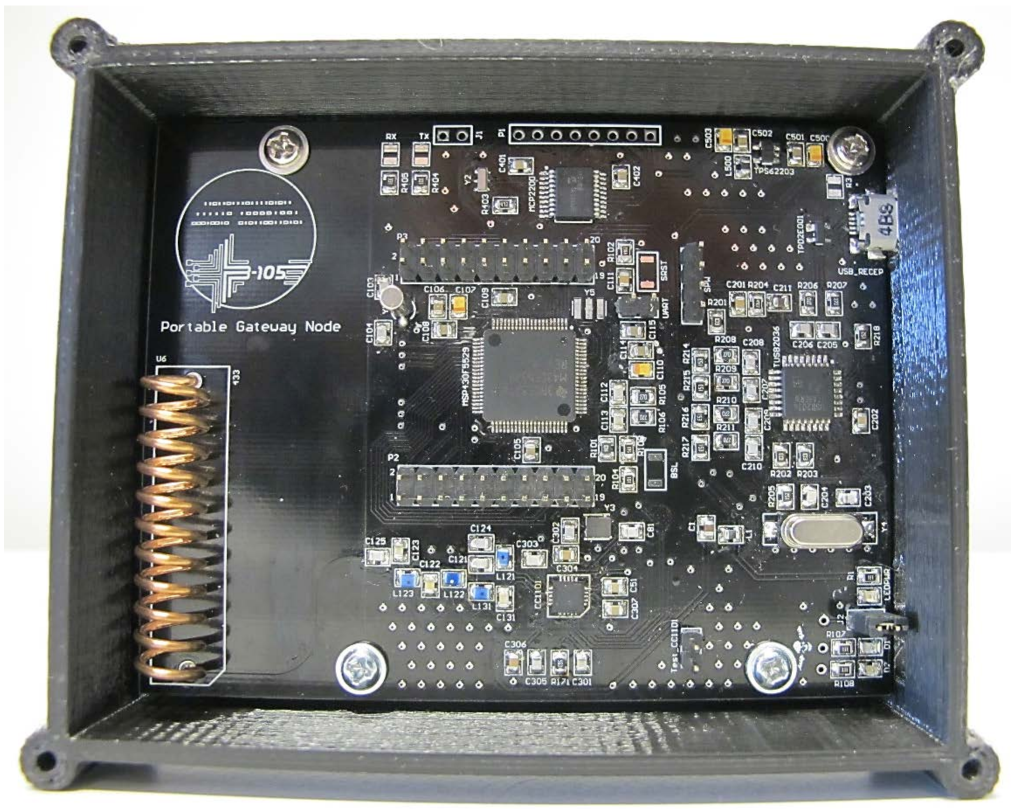


Figure 31 PGN with 3D-printed box

4.2.3 Firmware Implementation

In section 4.1.2 we covered the design phase of the PGN's firmware, the functionalities that are to be implemented are:

1. Port SimpliciTI to our PGN hardware and implement its protocol stack. Develop network transmission and reception routines, by using the available API commands SMPL_Send and SMPL_Receive.
2. Implement the UART driver based on the driver supplied in Design Note DN117. Develop the UART reception and transmission handling routines.
3. Develop the processing routines to route the incoming data from the UART or network to its destination executing the necessary processing.

4.2.3.1 Porting SimpliciTI

To port the SimpliciTI protocol stack, we have modified the code available for the MSP430F5529 USB Experimenter's Board + CC2520EMK available in [36], following the steps in [27]. This experimenter board features the same MCU we are using, so the changes will be the following:

- Edits in **bsp_board_defs.h**: changed the "Compile Time Integrity Checks" to enable compile time checking of the correct MSP430 platform: MSP430F5529
- Edits in **bsp_leds_defs.h**: this file contains the LED definitions available on the MSP430 platform. Therefore, we have to connect the LEDs to the correct MCU pins for our design: P1.3 and P1.4.
- Edits in **mrfi_board.c**: the mrfi_board.c contains the assignment of the GPIO ISR function. We had to change the statement because our GDO0 pin is connected to PORT2:

```
#if (MRFI_GDO0_INT_VECTOR != PORT1_VECTOR)
For:
#if (MRFI_GDO0_INT_VECTOR != PORT2_VECTOR)
```

- Edits in **mrfi_board_defs.h**: this header file defines the connections between the MSP430 and the RF module. We have to establish the following connections:

```
/* GDO0 Pin Configuration connect to P2.0 in header */
#define __mrfi_GDO0_BIT__    0
#define __mrfi_GDO0_PORT__   2

/* GDO2 Pin Configuration connect to P2.1 in header */
#define __mrfi_GDO2_BIT__    1
#define __mrfi_GDO2_PORT__   2

/* CSn Pin Configuration P2.5*/
#define __mrfi_SPI_CSN_GPIO_BIT__     5
/* Use P2DIR and P2OUT in the next statements */

/* SCLK Pin Configuration P3.2*/
#define __mrfi_SPI_SCLK_GPIO_BIT__    2
/* Use P3DIR and P3OUT in the next statements */

/* SI Pin Configuration P3.0*/
#define __mrfi_SPI_SI_GPIO_BIT__      0
/* Use P3DIR and P3OUT in the next statements */

/* SO Pin Configuration P3.1*/
#define __mrfi_SPI_SO_GPIO_BIT__      1
/* Use P3DIR and P3OUT in the next statements */

/* SPI Port Configuration */
#define MRFI_SPI_CONFIG_PORT() : change to P3SEL
```

- Edits in **smartrf_CC1101.h**: we used 433 MHz operation register values already available in the research lab. These values can be created using the tool SmartRF Studio, available at [37].

4.2.3.2 Program flow

The program flow in the PGN will be as showed in the following figure:

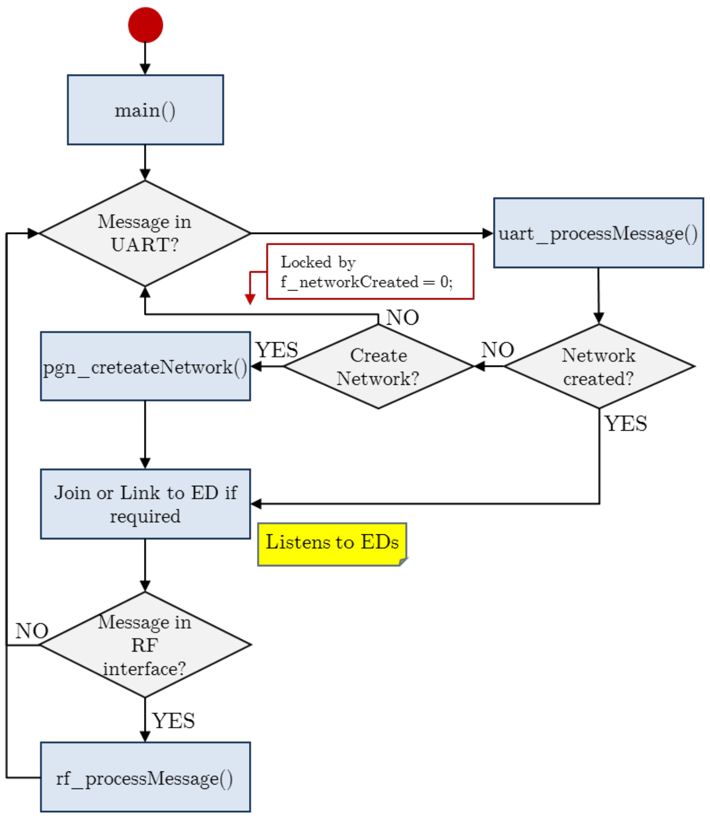


Figure 32 PGN firmware program flow

4.2.3.2.1 UART Driver and application UART processing

The UART has been configured following chapter 36 of the MSP430F5529 User's Guide [17]. We have configured the UART to work with the same speed and character format as the MCP2200 and the Android application, being this format: 9600 baud rate, eight data bits, one stop bit and no parity bit.

The following schemes show the TX and RX scenario in the UART driver. They describe the actions that take place, the conditions that have to be met and the functions involved:

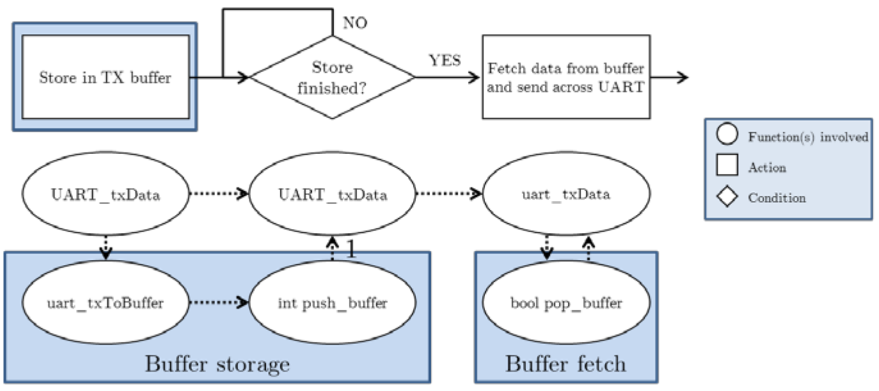
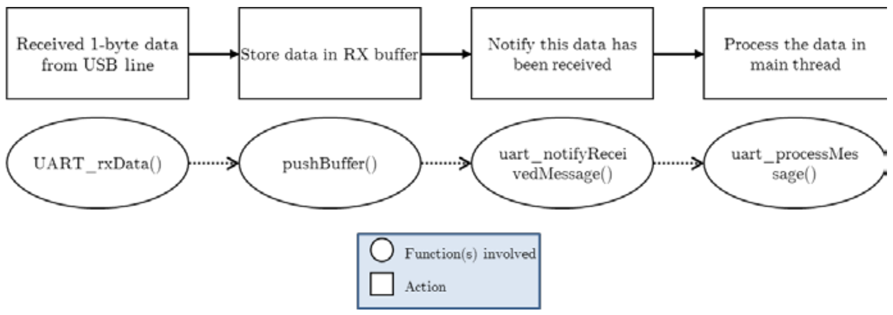


Figure 33 UART TX scenario



Received 1-byte data from USB line
Store data in RX buffer
Notify this data has been received
Process the data in main thread
UART_rxData()
pushBuffer()
uart_notifyReceivedMessage()
uart_processMessage()
Function(s) involved
Action

Figure 34 UART RX scenario

Finally, the following diagram shows how the main thread handles the reception of a protocol frame:

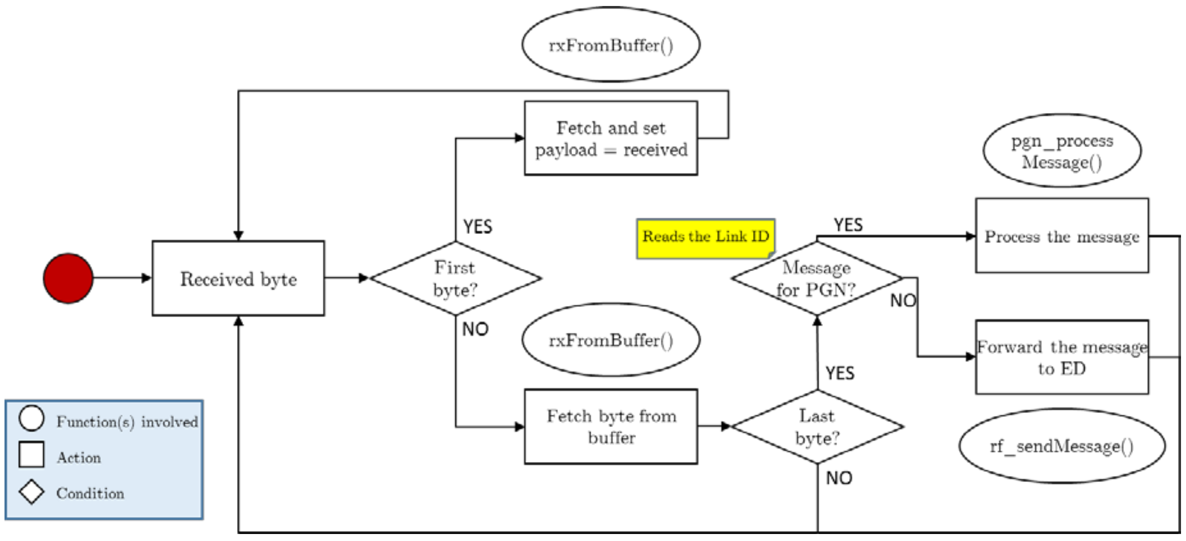


Figure 35 UART reception in main thread

5. Test scenario

This section will cover the test scenario that has been developed in order to demonstrate the correct operation of the system. It is important to note that the software found on the different subsystems; the Android application, the firmware on the PGN, and finally the firmware developed for the wireless sensor nodes, have been developed with this test scenario in mind.

Previous to this test phase, every subsystem has been tested on its own. This scenario will proof that all the components developed for this project are able to successfully interact with each other.

There are three elements in this test scenario, corresponding to the different subsystems developed for this project:

- An Android smartphone, in this case a Sony Xperia SP has been used. This device is capable of being used as a USB host.
- The PGN, which plays the role of an access point.
- Two wireless sensor nodes playing the role of end device. These nodes were developed for the project Prometeo [38] in the B105 lab. The Prometeo nodes have two LEDs (red and green) and a SHT75 digital temperature and humidity sensor that will be used in this test scenario.

In Table 12 we can see the necessary commands to communicate with an ED. The temperature and humidity requests as well as the toggle LED commands will be tested.

The first thing that we have to do is connect the Android device to the PGN using a USB OTG adapter cable + micro-USB to USB, seen on the next figure:

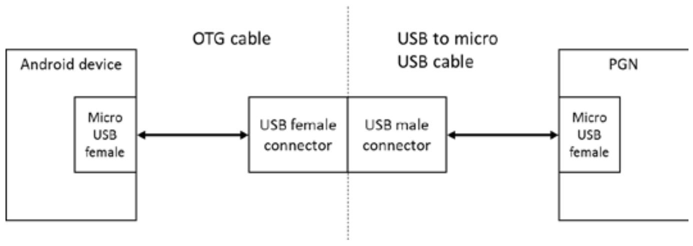
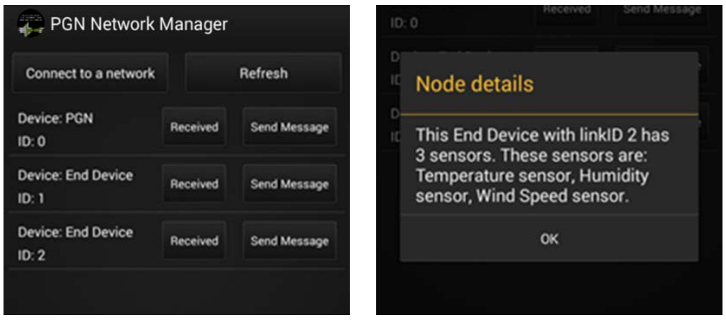


Figure 36 Android device and PGN connection

Then the Android application, which we have named PGN Network Manager, has to be launched. If we recall section 3 we must first connect to the PGN by pressing on Device Selection. On Device Selection we must search for the USB device connected, connecting this way to the connected PGN.

Afterwards, to request the PGN to connect to the surrounding end devices, we must go to Network Manager and click on Initialize. This will cause the necessary join and link actions to occur. After the PGN and the end devices are linked, we must send a command indicating the end devices to send their description. This is done by pressing on the "Send Message" in the PGN device on the table and selecting "Get EDs description". Once this is done we have the situation of Figure 37. We can also visualize the ED capabilities pressing on the field containing the ED's name and ID, as we see in Figure 37.


PGN Network Manager

Connect to a network | Refresh

Device: PGN
ID: 0 | Received | Send Message

Device: End Device
ID: 1 | Received | Send Message

Device: End Device
ID: 2 | Received | Send Message

Node details

This End Device with linkID 2 has 3 sensors. These sensors are: Temperature sensor, Humidity sensor, Wind Speed sensor.

OK

Figure 37 List of network nodes and ED details

Finally, it is time to check if an ED is capable of exhibiting a state-change on one of its LEDs or respond with a temperature or humidity lecture. To do this, we must click on “Send Message” on one of the EDs, identified by their link ID. By selecting one of the messages available like “Toggle LED 1”, “Toggle LED 2”, “Request temperature” or “Request humidity” we observe how the Prometeo nodes respond by toggling a LED or returning the temperature or humidity lecture, depending on the message sent:

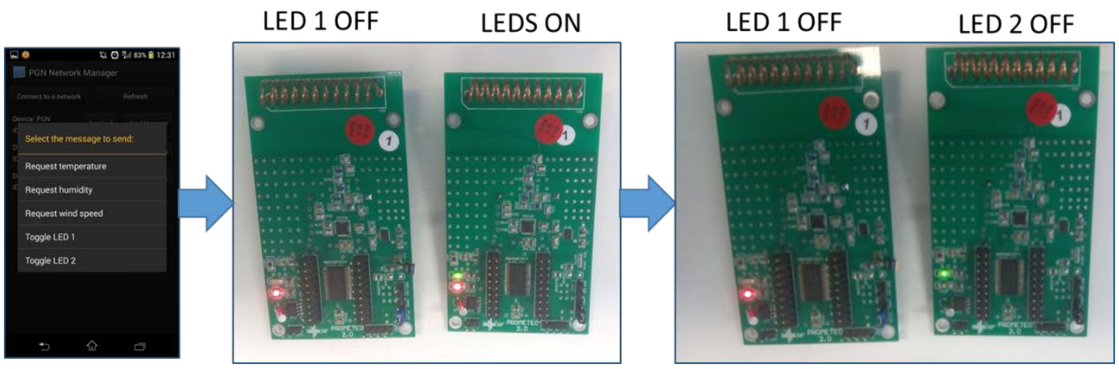
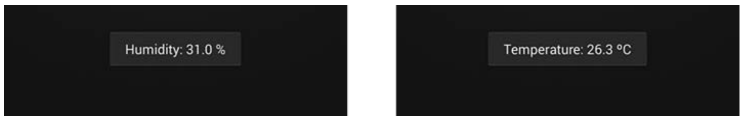


Figure 38 Communicating with two EDs


Humidity: 31.0 %

Temperature: 26.3 °C

Figure 39 Temperature and humidity lecture received from ED

6. Conclusions and future lines

6.1 Conclusions

We have successfully designed a system that is capable of connecting an end user to a WSN by just installing an application on their Android device and having a PGN available.

Regarding the Android application, working with the Android OS has been really comfortable due to the great amount of documentation that Android offers to developers. Furthermore, the USB API works really well and the documentation offered by Android is more than enough, although some example code would have been of help. This is true especially for the Android USB reception routines, which really took some time to debug.

The PGN has been designed to work as a gateway to this WSN. The software that has been designed, especially the PGN's software, has been designed with the purpose of being a part of a simple test scenario. The framing protocol developed is therefore simple but functional and has demonstrated to work properly. Furthermore this protocol can easily be expanded to include more functionalities.

Regarding the USB implementation, we decided to use a USB-to-UART solution in this project because we thought that the implementation would be simpler. We believe it was a really good decision given that the MCU UART routines were quite simple to implement. In addition, this solution has provided more than enough speed for our application. Nevertheless, this solution adds some additional components and also modifies the Android application's reception routine in comparison to a USB implementation. This is due to the fact that data arrives one byte at a time in our solution, whereas in a USB implementation the frames would arrive at the reception buffer of the smartphone all together, modifying what we see in Figure 14. As a future line, this project could be implemented using the full USB implementation.

On section 2.3.4 we decided to use SimpliciTI foreseeing that the implementation would be simpler than implementing a custom protocol. After using SimpliciTI, we conclude that this protocol works really well, and we like the fact that it abstracts from the programmer a lot of complexity of the network and physical layer. On the other side, SimpliciTI's basic stack only makes link IDs available to the programmer. With these identifiers you cannot make a correspondence between a link ID and a physical device, which could be necessary in some situations. We decided to overcome this situation by using ED descriptors, although some modifications to SimpliciTI's basic stack can be made to be able to use the ED's physical address.

To sum up, all the subsystems developed for this project have demonstrated, as we saw in the test scenario, to be really useful and work properly with devices already developed in the B105 lab. In conclusion, we think this project has proved to be a success.

6.2 Future lines

This project offers a big amount of improvements. Some of these are:

1. Develop a more complex application layer that expands or reinvents the one that has been developed for this system. This new approach could modify the existing frame structure. Other modifications could be the use of a more

reliable communications protocol by using acknowledgement messages (ACKs) or using features offered by SimpliciTI like the frequency agility mechanism.

2. Make the necessary modifications to make the PGN act as an ED in the network. This way, the PGN could use direct peer to peer communications, sending data to other EDs without the need of being an AP and having to always be present in the network.
3. Focusing on the hardware part, the PGN could be designed to occupy less space. The actual size of the PCB is 99.50 mm x 77.0 mm, making this design small enough to be portable, but improvable. Some other design defects could also be corrected.
4. The Android application can be modified in many ways to provide more stability. In addition, if the PGN's firmware is to be modified, the application should be modified in accordance.
5. Use the mobile device's network connection to be able to send the acquired data from the ED's over the internet.

7. Bibliography

[1] Business Insider smartphone report by country: http://www.businessinsider.com/smartphone-adoption-platform-and-vendor-trends-in-major-mobile-markets-around-world-2015-3

[2] IDC smartphone OS market share: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[3] Rozas Cid A., Análisis y diseño de escenarios para redes cognitivas en una plataforma móvil, PFC. Laboratorio de Sistemas Integrados, ETSIT-UPM, Sep. 2011.

[4] Android Open Source Project: https://source.android.com/

[5] USB 2.0 Specification: http://www.usb.org/developers/docs/usb20_docs/usb_20_060115.zip

[6] MSP-EXP430G2 LaunchPad Evaluation Kit User's Guide: http://www.ti.com/lit/ug/slau318f/slau318f.pdf

[7] MSP-EXP430F5529LP Development Kit User's Guide: http://www.ti.com/lit/pdf/slau533

[8] Android Applications With MSP430™ USB on Mobile Devices: http://www.ti.com/lit/an/slaa630/slaa630.pdf

[9] Android Applications With MSP430™ USB on Mobile Devices Source Code: http://www.ti.com/lit/zip/slaa630

[10] SimpliciTI Overview: http://www.ti.com/lit/ml/swru130b/swru130b.pdf

[11] SimpliciTI: Simple Modular RF Network Developers Notes: https://courses.cs.washington.edu/courses/cse466/10au/pdfs/SimpliciTI%20docs/SimpliciTI%20Developers%20Notes.pdf

[12] Android developers webpage: http://developer.android.com/guide/index.html

[13] Android reference on Intent: http://developer.android.com/reference/android/content/Intent.html

[14] Android reference on Broadcast Receiver: http://developer.android.com/reference/android/content/BroadcastReceiver.html

[15] Android reference on Array Adapter: http://developer.android.com/reference/android/widget/ArrayAdapter.html

[16] MSP430F551x, MSP430F552x Mixed Signal Microcontroller technical datasheet: http://www.ti.com/lit/gpn/msp430f5529

[17] MSP430x5xx and MSP430x6xx Family User's Guide: http://www.ti.com/lit/pdf/slau208

[18] Starting a USB Design Using MSP430 MCUs (Rev. A): http://www.ti.com/lit/pdf/slaa457

[19] Dai Q., Bao H., Liu Y., Liu Z., Zhou K., Wang J., 433MHz Wireless Network Technology for Wireless Manufacturing, 2008 Second International Conference on Future Generation Communication and Networking.

[20] Antenna Selection Guide: http://www.ti.com/lit/an/swra161b/swra161b.pdf

[21] CC-Antenna-DK and Antenna Measurements Summary: http://www.ti.com/lit/an/swra328/swra328.pdf

[22] ANT-433-HETH technical datasheet: http://www.linxtechnologies.com/resources/data-guides/ant-433-heth.pdf

[23] 2/3-port hub for the universal serial bus with optional serial eeprom interface technical datasheet: http://www.ti.com/lit/ds/symlink/tusb2036.pdf

[24] MCP2200 Configuration Utility v1.3.1: http://ww1.microchip.com/downloads/en/DeviceDoc/MCP2200%20Configuration%20Utility%20v1.3.1.zip

[25] MCP1700 technical datasheet: http://ww1.microchip.com/downloads/en/DeviceDoc/20001826C.pdf
[26] High Efficiency, SOT23 Step-Down, DC-DC Converters (Rev. E): http://www.ti.com/lit/gpn/tps62203
[27] MSP430 SimpliciTI Porting Guidelines: http://processors.wiki.ti.com/index.php/MSP430_SimpliciTI_Porting_Guidelines
[28] Starting a USB Desing using MSP430 MCUs: http://www.ti.com/lit/an/slaa457a/slaa457a.pdf
[29] MSP-EXP430F5529LP Hardware Design Files (Rev. B): http://www.ti.com/lit/zip/slar090
[30] CC1101 Low-Power Sub-1GHz RF Transceiver technical datasheet: http://www.ti.com/lit/ds/symlink/cc1101.pdf
[31] TI USB 1.1/USB 2.0 Hub FAQs: http://www.ti.com/lit/an/slla314/slla314.pdf
[32] TUSB2036 and TUSB2077A EVM User's Guide: http://www.ti.com/lit/ug/sllu190/sllu190.pdf
[33] MCP2200 Technical Data Sheet: http://ww1.microchip.com/downloads/en/DeviceDoc/22228B.pdf
[34] TPS62203EVM-211, TPS62200EVM-211 User's Guide: http://www.ti.com/lit/ug/slvu069/slvu069.pdf
[35] Tinkercad online application: https://www.tinkercad.com/
[36] Simpliciti IAR protocol stack 1.2.0: http://www.ti.com/lit/zip/swrc099
[37] SmartRF Studio: http://www.ti.com/tool/smartrftm-studio&DCMP=hpa_rf_general&HQS=Other+OT+smartrfstudio
[38] B105 laboratory Prometeo project: http://elb105.com/prometeo-tecnologias-para-el-combate-integral-contra-incendios-forestales-y-para-la-conservacion-de-nuestros-bosques/