Luís Miguel Henriques Antunes

Modular hardware platform for monitoring and control at small office and home office

Master's degree thesis in Physics Engineering, presented to Faculty of Sciences and Technology of University of Coimbra, under the supervision of Prof. Dr. Jorge Afonso Cardoso Landeck and Rafael Patrício Balaus Jegundo, Whitesmith.

September, 2015

• U C •

UNIVERSIDADE DE COIMBRA

Dedicado a
José Carlos Marques dos Santos e
Maria Aldina de Jesus Henriques dos Santos

Resumo

Este trabalho consistiu no desenvolvimento de uma plataforma modular de hardware para controlo e monitorização, criando, deste modo, a base para a rápida prototipagem de produtos e sensores capazes de se ligarem à Internet.

Utilizam-se produtos e dispositivos disponíveis comercialmente para o público em geral na criação da plataforma, como é o caso do Raspberry Pi e de Arduinos, interligados pelos módulos de rádio NRF24L01+.

A plataforma desenvolvida foi usada na construção do Qold, um produto para monitorizar temperaturas de forma automática, sem fios e totalmente integrado com uma aplicação online, tendo sido testado num cenário de utilização real, mostrando-se um sistema fiável (80.7% e 74.8% de up time no principal instalação feita). Um total de 5 gateways e 14 nodos foram instalados.

Abstract

In this project it was developed a modular hardware platform for monitoring and control. This platform will allow the faster development of products and sensors able to be connected to the Internet.

We used commercial off the shelf products, such as Arduino and Raspberry Pi, and we connected them using the NRF24L01+ radio modules.

The created platform was used in the development of Qold, a product designed to automatically measure temperatures and that is totally integrated with a web application. Qold was tested in real situation scenarios, with a reliable performance (with up times of 80.7% and 74.8% in the main pilot tested). A total of 5 gateways and 14 sensor nodes were installed.

Acknowledgments/Agradecimentos

Primariamente, gostaria de agradecer ao Professor Jorge Landeck pela sua visão e objectividade, tanto nas suas aulas como nas reuniões tidas ao longo do projecto.

Ao Rafael Jegundo, por me ter convidado a trabalhar na Whitesmith e me ter dado a oportunidade de desenvolver este projecto de forma autónoma, mas apoiada.

À Fátima, por desde cedo me ter incutido o gosto pelo conhecimento, me ter apoiado incondicionalmente e ser um exemplo a seguir. Ao João, pelo carinho e afecto demonstrado desde sempre. A ambos estarei sempre eternamente grato.

Aos meus sobrinhos, Guilherme, Gabriel e André, por tornarem as idas a casa momentos de maior alegria.

Ao André Martins e Bruno Alves, pela amizade e companheirismo.

Às pessoas que conheci em Coimbra e que serão sempre recordadas com carinho e saudade, especialmente Andoni Santos, Daniel Mano, Emanuel Barata, Xavier Carvalho, Saulo Ramos, Marco Pinto, João Araújo, Vasco Elói, Ricardo Faleiro, Jorge Morais, Vanessa Marques, António Almeida, Sérgio Pereira, Pedro Silva, Carlos Henriques e Mariana David.

Um especial agradecimento ao Afonso Sousa e Mário Ribeiro pela sua amizade.

Às pessoas da jeKnowledge, da SAC e da Whitesmith com quem tive o prazer de trabalhar e conviver.

A todos aqueles com quem me cruzei ao longo desta jornada.

Aos meus tios, José Carlos e Aldina, por serem mais do que uns pais para mim.

À Natacha, por ser especial.

Contents

List of Figures

List of Tables

Abbreviations

ADC	Analog to Digital Converter
ALU	Arithmetic Logic Unit
BLE	Bluetooth Low Energy
DAC	Digital to Analog Converter
DDR	Data Direction Register
DIY	Do It Yourself
GPIO	General Purpose Input Output
HTTP	Hypertext Transfer Protocol
I^2C	Inter Integrated Circuit
IDE	Integrated Development Environment
IoT	Internet of Things
ISM	Industrial, Scientific and Medical
ISR	Interrupt Service Routine
MISO	Master In Slave Out
MOSI	Master Out Slave In
PAN	Personal Area Network
PRX	Primary Receiver
PTX	Primary Transmitter
PWM	Pulse Width Modulation
RF	Radio Frequency
RFID	Radio Frequnecy Identification

RTC	Real Time Clock
SCI	Serial Communication Interface
SCK	System Clock
SCL	Serial Clock Line
SDA	Serial Data Line
SPI	Serial Peripheral Interface
SS	Slave Select
SSH	Security Shell
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
WSN	Wireless Sensor Networks

It is our choices . . . that show what we truly are, far more than our abilities.

J. K. Rowling

Chapter 1

Introduction

This chapter provides an overview of what one can expect from this Master thesis. It will be explained the motivations behind the choice of the theme, the context in which this project is developed and the main goals of it. In the end of the chapter it will be possible to see this report structure, allowing a preview of what will be discussed in each chapter and section.

1.1 Motivation

There are many examples that illustrate the importance of Information Technologies in the modern society: they are being used as a provider of solutions to one of the biggest concerns of present days: environmental challenges [1]; as the basis for economical growth [2]; as a method of improving the productivity [3] and helping increasing the quality of life [4]:

Smart grid and smart power systems incentive the final consumer to reduce their home power consumption through a more transparent relationship between utilities and final consumers: utilities can selectively modify the amount of electricity they supply, and consumers can adjust their electricity use to take advantage of market price conditions [5].

Smart buildings and home automation systems take advantage of a series of technologies that combined can provide a more energy-efficient and comfortable buildings as well as the capacity of monitoring and controlling [6].

Smart transportation systems can optimize and increase the efficiency of both freight and passenger transport and may also contribute to an overall reduction of vehicles on the roads. On one hand there is less need to deliver goods if those goods can be delivered electronically. On the other hand, the use of information technologies allows a better management of the transport routes and traffic [7].

These concepts have something in common: all of them rely on sensors and actuators technology and more often than not on sensors and actuators networks.

1.1.1 Sensors and Sensors Networks

We can define a sensor as an electronic device that converts a measurable physical phenomenon into an electrical signal that can be read and analyzed by the user. Consequently, sensors represent the interface between the physical world and the world of electrical devices, such as computers. The reverse is represented by actuators, which convert electric signals into physical phenomena [8].

A sensor network can be viewed as a network of devices denoted as *nodes* that can sense the environment and communicate the information gathered from the monitored field to a central device (*sink* that can use it locally, or is connected to other networks (e.g., the Internet) through a gateway [9]. The communication between the sensors and sink can be made using wires or be wireless.

This subject will be extensively developed in the following chapters.

1.2 Whitesmith

Whitesmith[1] - the company where the work of this thesis was developed - is a Coimbra based technological *startup*. Although external software projects represent the core business of the company, Whitesmith is also deeply interested in developing its own products, such as Unplugg[2] and Qold[3]. The latter one was one of the main reasons behind the work done in this thesis.

1.2.1 Qold

Qold is an integrated and automated system of temperature monitoring. It consists in a network of sensors that measures the temperature in the desired places and then sends the registered values through wireless to a central gateway. This gateway then sends the values to an Internet server where the data is saved and then displayed to the final user. Using this product the final user has access to the temperature of as many places as he wishes in an automated, effortless way.

As we can see this products relies heavily on a hardware platform to overcome the requirements of the product:

- sensor nodes to measure and register the temperature;
- reliable communication between sensors and a central gateway;
- gateway capable of communicate with sensors and with Internet.

[1]http://www.whitesmith.co/
[2]http://unplu.gg/
[3]http://www.qold.co/

1.3 Project Goals

The main goal of this thesis is to create and develop a modular hardware platform for monitoring and control at Home office and Small office. It will be validated with the creation of two different prototypes: one to measure temperature and the other to measure power consumption.

This platform should be able to address one of the main concerns in home and building automation: the non existence of a reliable and standard platform able to make the bridge between the sensors we have connected in our local network and the Internet [10, 11]. In the end of this project we should have a system capable of connecting a generic sensor to a remote server, allowing us to easily prototype products such as Qold (1.2.1).

This modular hardware platform will be divided in the following items:

Sensor nodes to measure one or more physical quantities and then transmit them to a gateway. These sensor nodes should have the following characteristics [12, 13]:

- low cost and small form factor;
- scalable architecture and efficient protocols;
- resource-efficient design;
- self-configuration and self-organization sensors;
- localized processing;
- application specific design;
- secure design.

Gateway to make the bridge between sensors in our network and the Internet. It also should fulfill some requirements [14]:

- wide range of access capability;
- manageability;
- protocol interworking;
- low cost and small form factor;

All of these features and requirements will be explained in more detail in the following chapters. For now I would like to emphasize that this platform will be the base for the future hardware products of the company so the low cost factor is one of the most important factors to have in consideration when the time to do choices arrives.

1.4 Scope

This section will help to clarify what is expected to be developed and accomplished during the course of the project. The scope of this thesis falls under the following:

Design and build a modular, battery powered sensor node compatible with a wide variety of sensors and with the most popular radio modules.

Program the sensor nodes to measure the desired physical quantity and to transmit the data to a gateway.

Choose a gateway compatible with the most popular radio modules and communication protocols.

Configure the gateway to accept data from the sensor node and to transmit it to a Web server.

The Web server interface of the platform designed to receive, store and display the data sent by the gateway is not expected to build since it is already done, as well as the visualization dashboard.

As monitoring is the first, critical step for control, and a challenge to do reliably on its own, it was decided to focus the efforts on the development of a reliable monitoring platform.

1.5 Report Structure

This report offers an overview of the developed work, explaining not only what was done but also the concepts and the state of the art behind it. So we divided it in chapters to a better organization of the document.

Hence, chapter 1 gives an overview of the project, explaining the motivation, the goals and the objectives.

Chapter 2 is about the technology behind wireless sensor networks, including microcontrollers, sensors, communication interfaces and of course, an overview of wireless sensor networks per se.

In chapter 3, we offer an overview of the Do It Yourself trend and some concepts are explained, such as Internet of Things and open source products.

Chapter 4 is used to describe the developed product architecture, explaining hardware and design choices of the sensor nodes and gateway.

Chapter 5 displays some applications of the designed platform, with some of them already deployed.

Finally, on chapter 6, we have the conclusion of this report, with some analysis about the developed work and an insight of future work.

Chapter 2

Sensoring and Telemetry

Thanks to the great advance in the semiconductor industry (the transistors went from 10 *microns* technology in the 1970s to the actual 14 *nanometers* [15]), it is now possible to put all digital parts of a computer in a single chip, called a microcontroller.If we also take into account that these microcontrollers are becoming less expensive and more powerful through the time we can see why systems capable of interacting with the physical world are becoming so popular. Besides that and hence the hardware allows the use of standard Internet protocols, monitoring and controlling can be done over the Internet [16].

This new paradigm of technology allow us to develop things such as a network of smart and autonomous sensors where each one of them sends the measured value directly to a visualization platform accessible everywhere through internet.

2.1 Microcontrollers

A microcontroller is a multipurpose device that combines a processor, memory and a whole lot of other components integrated into one single chip [17]. Unlike a microprocessor, a microcontroller needs not only to be able to compute, but also to have some intrinsic features such as excellent input/output capability to be able to interact with external devices and environment [18]. Today, billions of microcontrollers are produced every year and the controllers are integrated into many appliances we all use in a daily basis, such as

- household appliances;
- telecommunications;
- automotive industry;
- aerospace industry;
- industrial automation;
- home automation;

- industrial, scientific and medical (ISM) environments.

In the context of this project, microcontrollers are extremely important since they are the basis of the sensor node.

2.1.1 Microcontroller Families

Since microcontrollers are so widely used and to do substantially different tasks, there is a question of which microcontroller is the appropriate choice to overcome the designed task. The rule of thumb, specially in industry, is to choose the cheapest device that matches the application needs. However, even after this primary selection, there is a wide variety of microcontrollers to choose from. The first important decision to make is the controller family since it defines the controller's architecture. All controllers of a family are similar to each other, containing the same processor core and hence are code-compatible, but they differ in the additional components such as the number of timers, the amount of memory or number of general purpose input/outputs (GPIO).

The are many different microcontroller brands such as PIC[1], Atmel[2] or Texas Instruments[3] just to name some of the most popular. All of them offer a wide variety of families, but in the end all of them share the general microcontroller architecture.

Microcontroller Architecture

The general architecture of every microcontroller is pretty similar and can be seen in figure 2.1. All parts of the device are connected to each other via an internal bus and are all integrated on one single chip. The connection to the physical world is made using the GPIO pins.

The components of the microcontroller are explained in the rest of the section.

2.1.2 Processor Core

The processor core (CPU) is the main part of any microcontroller. It is constituted by the arithmetical logic unit, the control unit and by the registers.

Arithmetic Logic Unit (ALU) is the part responsible to perform logical computations. It expects two inputs and returns the result of the operation as an output.

Control Unit has the function to determine which instructions the ALU executes next. There are two different designs of control units, each one with its advantages and disadvantages.

[1]`http://www.microchip.com/pagehandler/en-us/products/picmicrocontrollers`
[2]http://www.atmel.com/
[3]`http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/overview.page`

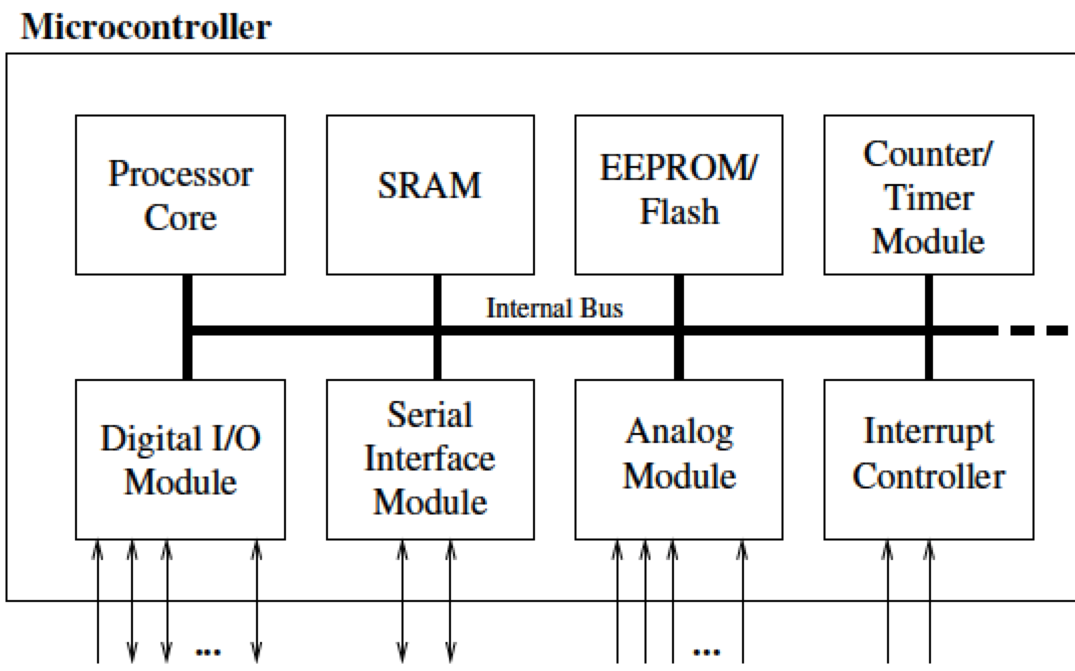


Figure 2.1: Basic architecture of a microcontroller [19].

RISC: Short for *Reduced Instruction Set Computer* is an architecture with simple, hard coded programs that takes few clock cycles to execute. The execution of instructions is very fast.

CISC: Short for *Complex Instruction Set Computer*, can have the instructions programmed in its microcode, allowing to feature more complex programs. Hence its superior code complexity the instructions set are more powerful when compared with RISC, but are also slower.

Register file stores the working registers of the CPU, with the last one taking the operands for the ALU from the file and storing the result back into the register file after doing the computing.

2.1.3 Memory

There are different kinds of memory in a microcontroller and each kind has a specific function. Besides the register file already mentioned (2.1.2) it is possible to categorize the memory in the following categories:

Data memory for long terms storage. This memory is generally external to CPU and is larger than the register file.

Instruction memory is also larger that the register file and here are stored the programs. Depending on whether we are dealing with Von-Newmann-Architecture or

Memory							
Volatile		Non Volatile					
SRAM	DRAM	ROM	PROM	EPROM	EEPROM	Flash EEPROM	NVRAM

Table 2.1: Classification and types of memory.

Harvard Architecture, this memory can be in the same physical memory as the data memory [20].

Because of the compact nature of microcontrollers, the memory is integrated in the same chip of the CPU which leads to restrictions in size.

Physical and Electronic Categorization

The previous classification was based on the functionality of the memories that are used. However we can also separate the different kinds of memory based on its electronic and physical properties.

The major difference between memories would be between *volatile* and *non volatile* memories. The first ones mean that the memory will be permanently lost when the power is turned off.

Inside each category we can also separate different kinds of memory. We can see this classification is the table 2.1.

Volatile Memory keeps its content only as long as the system is powered on. Although it may seem a big con, volatile memory is extremely useful because of its faster access time when compared with non volatile memory.

Volatile memory is generally known as RAM - Random Access Memory. The word random means only that any memory location can be accessed in the same amount of time, regardless of its position in the memory.

There are two types of volatile memory.

Static RAM chips are formed by an array of cells, each one capable of storing one bit of data. It uses a process called *flip-flop*, which consists in six transistors, to store the information [21]. The information is kept as long as power is applied to the chips, hence the name *static*.

Dynamic RAM chips use capacitors to store an amount of charge, where the charge level represents either a 1 (if the capacitor is charged) or a 0 (if the capacitor is empty). The *Dynamic* part refers to the fact that the chip needs to be refreshed every once in a while so the capacitors don't lose their charge due leakage currents [22].

Each kind of RAM has its advantages and disadvantages. DRAM can store much more information (about 4 times more [21]) using the same area since it only needs one transistor and one capacitor *per* cell. However, since it needs to be constantly refreshed, there is some inaccessible time where the DRAM is not of use. On the other hand, SRAM is quite faster but has high limitations in terms of memory size, since large memory chips are quite expensive and occupy an wide area.

In the case of microcontrollers we usually find SRAM because only small amounts of memory are required.

Non Volatile Memory holds data permanently, even if power is down. This is a great advantage since we don't need to worry about a possible fail in the power supply to the chip. It comes along with some disadvantages too: writing in these kind of memories is much slower when compared with SRAM and DRAM [23].

There are different types of Non Volatile Memory:

ROM , or Read Only Memory, were the first type of non-volatile semiconductor memories [19]. As the name implies, it is not possible to write in these kinds of memory. The programs have to be hard coded by the manufacturer at the time of the making. Many embedded systems and microcontrollers use ROM since in most cases the programs are never changed during the lifetime of the device.

Programmable ROM allows the memories to be programmed once, and only once, by the final user. The memory cells are formed by a fuse and a transistor. Initially, when the fuses are intact, all cells are read as a 1. When we select a memory cell and apply a short pulse, the fuse is destroyed and the cell is now read as a 0. Once the fuse destroyed, it can't be reverted, hence the one-time programming.

PROMs are well suited for middle range production, when the number of demanded chips isn't high enough [19, 23].

Erasable PROM can be programmed more than once, unlike PROM. The memory is stored in the gate of field effect transistors [24], and it is possible to program the cell by applying a high voltage in the drain pin. This high drain voltage leads to a process called avalanche injection that will charge the gate [25] and then closing the transistor switch. This means that no fuses are blown or irreversible processes occur.

Once the high voltage is removed the charge is trapped in the gate pin and it should be expected to the electrons to remain in the gate indefinitely, however it is not what is verified. Due to current leakage the charge drops through the time. When the gate loses enough electrons, the cell state is reverted to the original state. The amount of time required for this to happen is specified in the data sheets of the manufacturer. Although this current leakage is a limitation of EPROMs, it is used as an advantage by the manufacturers: by accelerating the process - exposing the silicon chip to a direct ultra-violet light source - it is possible to complete erase the memories: once the electrons in the gate absorb enough energy to break the $3.2eV$ barrier between the gate and the substrate they leave the gate [23]. Because of this feature, the packages containing these memories have a seal covered glass window. Once the seal is removed the memory will be erased in few minutes.

The process of programming and then erasing the memories is quite dull and time consuming, so more often than not, EPROMs are used as an One Time Programmable memories. In this case simple, cheaper packages are used.

Electrically EPROM is the natural evolution of EPROMs, bringing one long desired feature: the ability to easily reprogram memories. Light sources are not needed to erase the memories since no special voltage is required. As the name suggests, the erasing of the memories is made electrically, using the Fowler-Nordheim [26] tunneling effect to some extent. This tunneling effect is reversible, allowing the EEPROMS to be user several times. EEPROMs also have limitations: the number of write/read cycles is limited (around 100000 [19]) and they can't store the information indefinitely, either. This type of memory is widely used in microcontroller applications but due it's limitations, it is often used as longer term storage rather than as scratch memory.

Flash EEPROM has a really peculiar feature: when it erases data it is not possible to specify which cells you want to erase. This means that all the information is erased once the process of *flashing* the memory is completed. This way, the internal logic of the chips can be simplified and consequently it is cheaper.

Flash EEPROM are used essentially to store the microcontrollers' Flash Program since it is something that is not expected to be reprogrammable often.

Non Volatile RAM combines the advantages of both volatile and non volatile memories. This ideal solution can be achieved through two methods:

- adding a small battery to an SRAM device;
- combining a SRAM device with an EEPROM in one package.

2.1.4 Digital Input/Output Pins

The ability to directly monitor and control hardware and interact with external devices is the main reason why microcontrollers are so widely used. They interact with hardware through their digital input/output pins so it is expected to find digital input/output pins in every microcontroller. Most of them have between 8 and 32 pins [19]. In general, pins are grouped in *ports*, each containing 8 pins which allows for them to be accessed and programmed using a single byte.

Most digital pins are also called general multipurpose input/output (GPIO) pins since they have have other functionalities beyond their digital input/output capabilities: communication pins, analog modules, timers are all digital are alternate functions of digital pins.

The behavior of digital pin is controlled by three registers:

Data Direction Register (DDR) controls if pins are configured to be an input pin or output pin. Usually, each bidirectional port has its own DDR. Each pin of the port can be controlled independently. This means that we can have all of them as inputs, all of them as outputs, half as inputs and half as outputs or any other desired combination [27, 28].

Port Register (PORT) is used to store the logical state of the output pins. When a pin is configured to output, if the corresponding pin in PORT is set, the pin will be in a high state; if the pin is clear, the pin will be low [27, 28].

Port Input Register (PIN) is used to generally used read the state of the input pins, but it also registers the state of the output pins which allows to verify if the PORT operation worked as expected [27, 28].

Digital Input

Digital input pins are used whenever the signal we want to measure should be interpreted digitally, that is, only has two possible states: logical ‘1’ or logical ‘0’, ‘HIGH’ or ‘LOW’ respectively.

Since voltage is the only thing a microcontroller can measure, there is a range of measured voltages interpreted as ‘HIGH’ and a range of voltages interpreted as ‘LOW’. Depending on the logic level used by the microcontroller, theses values can vary.

An example of these values can be seen in figure 2.2.

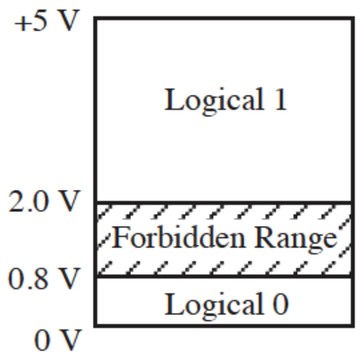


Figure 2.2: In this example, logical 0 is assumed when the voltages are between [0-0.8] volts and logical 1 is assumed when voltages are between [2-5] volts. These values are common on 5V transistor-transistor logic (TTL) [20].

As we have seen in figure 2.2 there is a range of values where the signal is neither a logical 1 nor a logical 0. This is expected to happen, since it reduces the risk of a false logical 0 or a false logical 1.

Digital Output

Just as digital input pins allows to interact with systems whose output is restricted to two states, digital output pins allows to interact with systems that also expect two states, but this time as inputs. They are particularly useful to control things such as LEDs or to turn something ON or OFF.

Once the DDR of a pin is set to output, the logical state is controlled by the PORT register. The values of the tension for each state depend on the type of logic used. Figure 2.3 shows an example of possible output values.

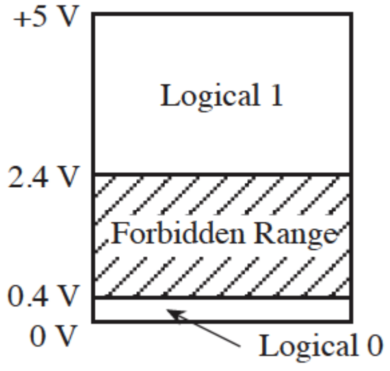


Figure 2.3: In this example, logical 0 is assumed when the voltages are between [0-0.4] volts and logical 1 is assumed when voltages are between [2.4-5] volts. These values are common on 5V transistor-transistor logic (TTL) [20].

2.1.5 Analog Input/Output Pins

Microcontrollers are used essentially to interact with the physical world and, unlike microcontrollers, the physical world is fully analogical. So microcontrollers need a way to convert the information given by a world full of analog signals and into digital information so it can understand and deal with. The analog module of the microcontroller is the one responsible for that.

There are two basic types of converters: digital-to-analog (DAC) and analog-to-digital (ADC) converters.

DAC

DACs have the responsibility of transforming a n-bit digital value N, in the range of $[0, 2^n - 1]$ and generate a proportional analog value V_O [29].

Although it is undoubtedly something useful, many microcontrollers lack a dedicated DAC. If the cases where an analog output signal is needed, there are two options: use an external DAC or use a functionality called pulse width modulation (PWM).

PWMs generate an intermittent signal whose HIGH/LOW time ratio is proportional to the digital value N. Since the HIGH/LOW transition takes some time due to an RC circuit, the output average PWM signal is equal to an analog value proportional to the value N. We can see this effect in figure 2.4.

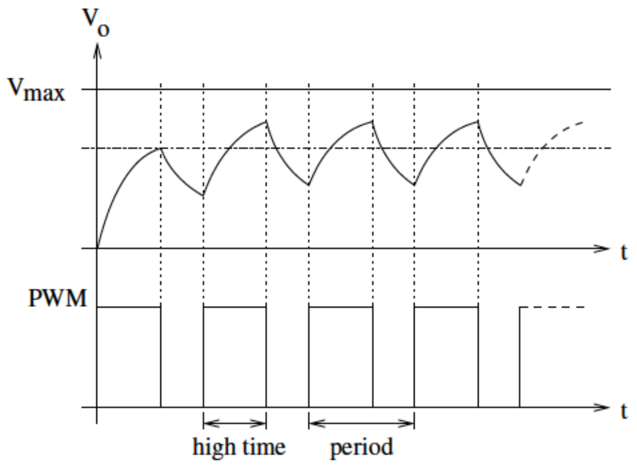


Figure 2.4: In this figure we can see an example of a PWM signal. The average voltage output signal is proportional to the high time period ratio [19].

To generate this kind of signal, microcontrollers use a dedicated timer and the resulting output signal is not as good as an obtained using a dedicated DAC since it needs to stabilize and the output value is always oscillating around the average value. Nevertheless this solution is good enough to use in many situations, such as motor control or light dimming.

ADC

When it is needed to convert analog values to its digital form, we use analog to digital converters. They are specially useful in situations where we want to measure the voltage of a signal and display it. We can find an ADC in almost every microcontroller.

ADCs accept an analog value as an input and converts it to a digital word that can be read by the microcontroller. The analog input voltage range $[GND, V_{ref}]$, where V_{ref} is the maximum voltage value that the ADC can tolerate, is mapped to 2^n individual values, where n represent the number of bits of the ADC. Typical values are 8 or 10 bits ADCs.

The number of bits allows us to determine the resolution of the ADC, that is, the smallest voltage difference that can be distinguished reliably [19]. This voltage difference is

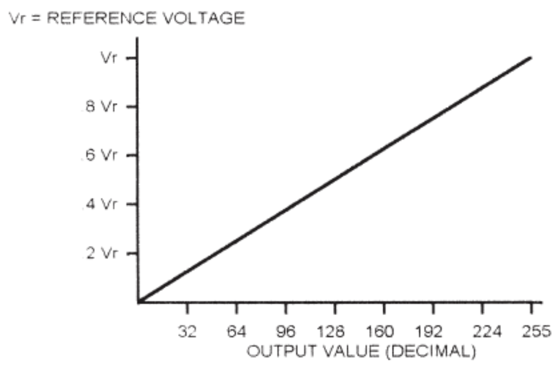


Figure 2.5: Example of what output values of a 8 bit ADC would be [29].

given by equation 2.1. Figure 2.5 shows the basic principles of analog-to-digital conversion.

$$\text{Smallest Voltage Difference} = \frac{V_{ref}}{2^n} \tag{2.1}$$

The digitalization of analog signals have some important aspects that we need to have into consideration when using an ADC:

- voltage differences smaller than ADC resolution go unnoticed and are represented by the same output value.
- conversions take time which means that we can not sample the whole signal. To be able to reconstruct a reliable representation of the signal the sampling frequency,f_s, should be at least twice of the maximum frequency, f_{max}, of the input signal.

$$f_{max} = \frac{f_s}{2} \tag{2.2}$$

This principle is known as Nyquist criteria [30].

2.1.6 Interrupts

Microcontrollers can be used to monitor events that are expected to happen at some point but are not periodical: pushing a button that turns a light on or automatically shut down the system if someone enters in the working area of the machine are some examples.

Assuming that the occurring event originates a change of state in an input pin, we need to assure that the microcontroller is monitoring this specific pin at the moment of the event. This can be done following two different approaches:

Polling the input signal, that is, periodically checking the input pin for state changes. Although it may be seem as a reasonable solution, this method has some flows:

- waste of processor time in events that may even not occur;

- the event may occur when the microcontroller is doing some other task, hence not being able to detect the change of state.

Interrupts that only come into action when the event is detected, making the microcontroller polling the signal and pausing the execution of the main program. As long as no events occur the microcontroller simply executes the main program. This is done calling an interrupt service routine (ISR).

Interrupt Control

Interrupts are controlled by two bits: *interrupt enable* bit and *interrupt flag* bit. The first one allows the interruption to occur and is set by the application programmer. The second one is set by the microcontroller itself when the event occurs.

Interrupt Priorities

In the cases where more than one type of interruptions can occur at the same time, the microcontroller has to be able to choose which one is to be dealt with first. This choice is made after consulting the vector table. This table contains an entry for each kind of ISR available and its priority.

2.1.7 Timer

Timer modules are present in every microcontroller, and most of them provide one or more 8 bit or 16 bit resolution timers. Timers can be used to generate PWM signals (2.1.5), to trigger interrupt the program (2.1.6) after a certain amount of clock ticks, or, it its most basic function, to act as counters.

Counters

Timers can be seen as counters whose value is either increased or decreased after every clock tick. This value can be read from the count register and specified by the user. 8 bit and 16 bit counters are the most common to find.

Even 16 bit counters can only have 65535 increments before it overflows. This number is well below that of the typical number of clock ticks per second doable by the microcontroller (typical values are in the order of the millions of clock ticks per second - MHz). To overpass this limitation, microcontrollers provide a feature called prescaler.

Prescaler is basically a nested counter attached to the first that increments with each system clock tick. When the prescaler overflows, one value is increased or decreased in the first counter and the prescaler is reset. Prescalers can be adjusted to match the need of the programmer.

2.1.8 Other Features

Watchdog Timer

The watchdog timer is used to monitor the execution of the main code. When enabled, it starts counting down. The counting keeps going until the main program resets the watchdog timer, meaning that the code was properly executed. Otherwise, when the counter reaches 0 the microcontroller is reset.

This can be viewed as a safety mechanism that is triggered when the program deviates from its normal behavior.

Power Consumption and Sleep

Microncontrollers are often powered by mobile batteries, and more often than not it is not trivial to change them. Because of that a low power consumption is a must have for any microcontroller.

There are some techniques that can be implemented to lower the energy consumption:

Clock frequency can be adjusted to minimize the energy consumption since they are proportional [31]:

$$E \propto f. \tag{2.3}$$

As long as the time requirements of the final application allows it, the microcontroller's frequency can be reduced without any major concerns.

Operation voltage is related to the energy consumption according to equation 2.4

$$E \propto U^2. \tag{2.4}$$

Reducing the operation voltage allows to drastically reduce the energy consumption of the chip. Unfortunately it is not possible to reduce this voltage arbitrarily, since microcontrollers need a minimum voltage to operate at a given frequency clock. It means that reducing frequency clock allows to reduce the operation voltage, which leads to even further energy consumption reduction.

Minimum operation voltages for a given frequency can be consulted in the microcontroller's data sheet.

Shutdown or sleep modes disable the functionalities of the microcontroller that are not being used. Since each modules draws some power to operate and not all of them are needed at the same time, this can be done without compromising the global goal of the application.

Usually we can find different sleep modes in a microcontroller, each one disabling a different module. Some sleep modes can even go further and shutdown all the modules. Then, the microcontroller can be waked from its sleep state using an interrupt (2.1.6).

2.2 Sensors

A sensor is an electronic device that measures phenomena of the physical world, converting the physical phenomenon into a proportional electric signal. Hence, sensors work as a bridge between the physical world and the world of electrical devices, such as microcontrollers (2.1) [8, 32]. Sensors allied with microcontrollers form the so called 'smart devices', that because of technology democratization have become largely used in everyday products [33, 34].

Sensors are usually classified based on the physical property it is designed to measure. Some of the physical properties measured include:

- temperature;
- humidity;
- pressure;
- position;
- electrical current;
- acceleration, shocking and vibration;
- light intensity.

Inside each one of the physical properties we have a wide selection of sensors we can choose, as we can see in table 2.2. It would be exhaustive to cover all of them and beyond the scope of this thesis.

Besides classification based on what a sensor will measure, we can also classify them based on their output: we have analog sensors and digital sensors.

There are also some features common to all sensors that will be discussed.

2.2.1 Performance Characteristics

When choosing a sensor, independently of the physical property we want to measure, there are some characteristics that we have in consideration when choosing the right sensor. This information can be easily consulted on the data sheets of the sensors.

Transfer function specifies the relation between the input signal and the output signal. This can be displayed either in the form of an equation or in the form of a graph, as can be seen in figure 2.6.

Sensitivity of the sensor is the minimum input of physical parameter capable of creating a detectable output change. It can also be viewed as the slope of the output characteristic curve.

Physical Property	Sensor	Output
Temperature	Thermocouple	Voltage
	Silicon	Voltage/Current
	Resistance temperature detector (RTD)	Resistance
	Thermistor	Resistance
Force/Pressure	Strain Gauge	Resistance
	Piezoelectric	Voltage
Acceleration	Accelerometer	Capacitance
Flow	Transducer	Voltage
	Transmitter	Voltage/Current
Position	Linear Variable Differential Transformers	AC Voltage
Light Intensity	Photodiode	Current

Table 2.2: Examples of sensor types and their outputs [1].

Dynamic range corresponds to the interval between the minimum and the maximum value of the measured variable that can be converted to an electrical signal by the sensor.

Accuracy can be defined as the largest expected error between the real value of the physical property and the value given by the signal output. This can be represented either in terms of absolute value or in terms of percentage.

Linearity corresponds to the deviation of the transfer function when compared to an ideal curve.

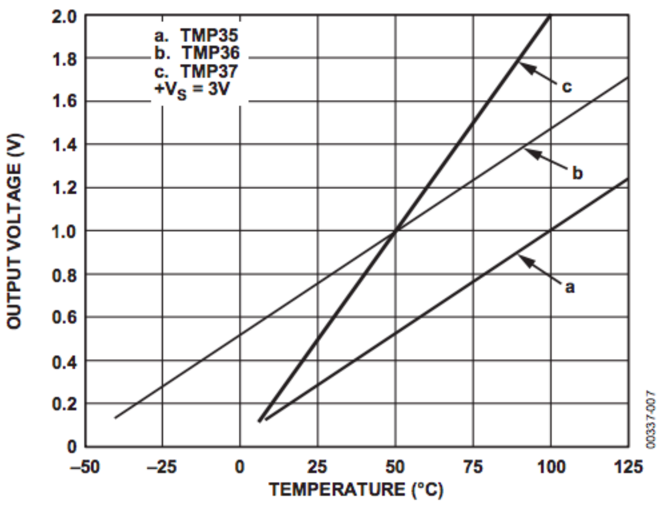


Figure 2.6: Transfer function of three different temperature sensors: TMP35, TMP36 and TMP37 [35].

Noise is the amount of output signal that is generated by the sensor. In most cases the noise produced by the sensor is less than the next element in the electronics, or less than the natural variation of the input signal, so in most cases it is not important [8].

Resolution of the sensor is the minimum detectable signal variation.

Offset error of the sensor is defined as the output signal that will be detected by the sensor when it should be zero.

Hysteresis happens when the signal is cycled up and down, causing the sensor to display different outputs depending on whether the signal is rising or falling.

2.2.2 Analog Sensors

Analog sensors convert the measured property to a continuous, proportional electrical signal. This signal can be a voltage or a current, whose typical values are the following [36].

- Voltage
  - 0 to 2.5V
  - 0 to 4V
  - 0 to 5V
  - ± 2.5V
  - ± 4V
  - ± 5V
- Current
  - 0 to 20mA
  - 4 to 20mA

After the conversion is done by the sensor, an ADC is needed to convert the output signal. As we have seen before, most microcontrollers have a built in ADC so in most cases we can connect directly the output of the sensor to an analog input of the microcontroller. However this is not always possible, either because the output signal is out of the allowed values by the microcontroller or because the signal output alone is not enough to give us the information we need. In these situations we need to provide a signal conditioning circuit between the sensor output and the input of the ADC. We can see a representative scheme of this process in figure 2.7.

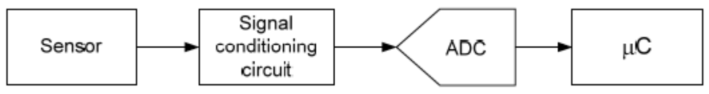


Figure 2.7: Classical analog sensor electronic interface [37]

.

2.2.3 Digital Sensors

Digital sensors convert the measured physical property to a discrete, two-state signal. This signal represents either a boolean state directly, such as, a switch that is pressed, the position of a tilt sensor, or can be the digital representation of a continuous value measured by the sensor.

This digital signal can be read directly by a microcontroller, connecting the digital output pin of the sensor to a digital input pin of the microcontroller.

Depending on how the data is transmitted, that is, parallel or serial, synchronous or asynchronous, data bus or point to point communication, full-duplex or half duplex, master-slave or equal partners, different protocols of communication are used. These communication interfaces will be explained on section 2.3.

2.3 Communication Interfaces

Microcontrollers are able to communicate with other devices, such as other microcontrollers or sensors, interfaces being thus one of its great advantages. This communication is made through one of its communication interfaces.

These interface can be implemented using different techniques and can be categorized based on several properties, as seen above (2.2.3).

It is common for microcontrollers to have several communication interfaces and sometimes even more than one instance of a particular interface [19].

2.3.1 Data transmission

The transmission of data across a link can be made either in parallel mode or serial mode. In parallel mode, several data lines are used to send multiple bits at a time. In serial mode, data is sent sequentially, one bit at a time. While there is only one way to transmit parallel data, serial data can be transmitted through two different techniques: synchronous or asynchronous.

Parallel Transmission

Binary data, consisting of 1s and 0s, is organized in groups of n bits each. The number of bits that are transferred in parallel varies, but widths of 4 and 8 bit are particularly useful, since they represent half a byte and a byte, respectively.

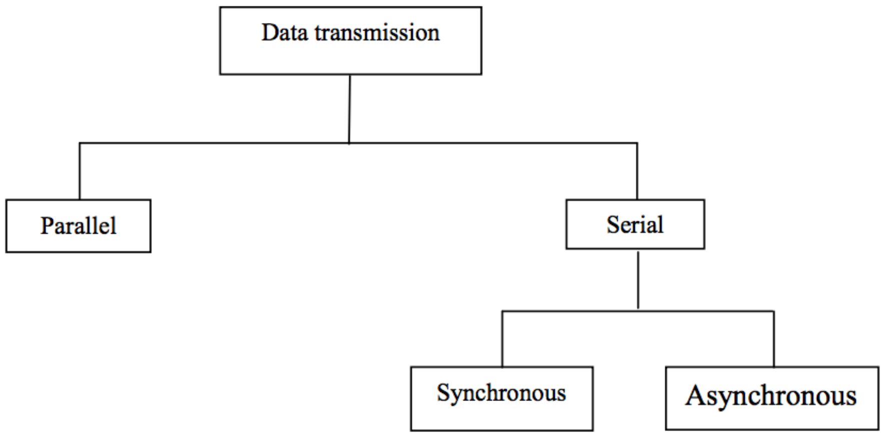


Figure 2.8: Overview of data transmission processes [38]

.

In parallel mode, each bit has one assigned wire, meaning that we need as many wires as bits that will be transmitted. Then, each bit of each line is transmitted at the same time, allowing the groups of bits to be transmitted at each clock pulse from one device to other.

Figure 2.9 is a representative illustration of this process.

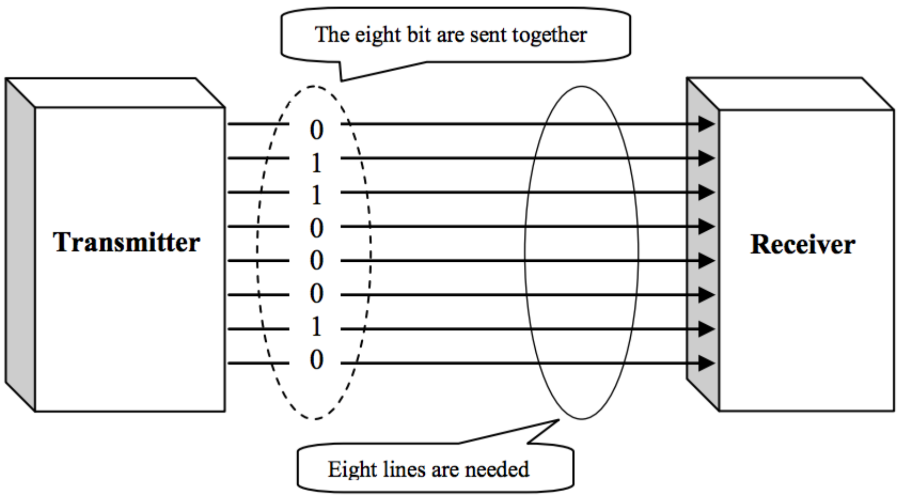


Figure 2.9: Parallel transmission of 8 bits [38]

.

Serial Transmission

A serial interface sends data sequentially, one bit at a time. This method has the advantage of only needing one wire, making this communication method resource efficient. On the other hand, the number of bits that can be sent per second is lower that when using a parallel interface.

Serial transmissions can be either asynchronous or synchronous.

Asynchronous transmission means that the *sender* and *receive* clock are not connected. As the receiver and the transmitter are not synchronized, the receiver must know in advance that there will be data coming from the transmitter. This implies that special techniques must be used to assure an efficient communication.

At the start of the transmission, a special bit is added to the beginning of the message, usually a 0. This is called the **start bit**. At the end of the message, other special bit is added, the **stop bit**. This bit is usually a 1.

In asynchronous transmission, at least 10 bits are sent for each message: 1 start bit, 1 stop bit and 8 bits on the message itself. There maybe also a time interval between each transmission - the gap.

Once the acknowledgment that a message is coming is done, the receiver sets its clock to the same frequency as that of the sender. It means that some synchronization is needed, but only for the duration of a single byte. After it receives a stop bit, the receiver will ignore all other bits until a new start bit is received.

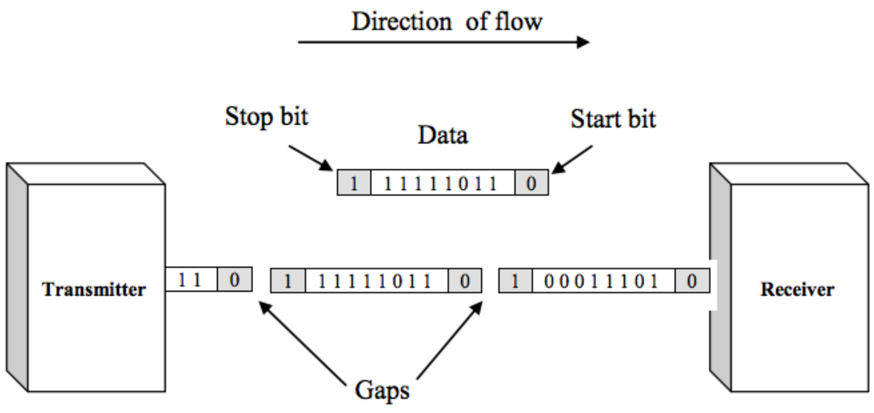


Figure 2.10: Asynchronous transmission [38]

.

Synchronous transmission means that the receive clock is linked to the transmitter clock. This is done be either adding an additional clock that controls both send and receive unit, or by using a data format whose clock signal can be reconstructed by the receiver.

It allows for the byte to be sent without any gap between it and the next one, and without the need of start or stop bits. Bits are sent sequentially and is the responsibility of the receiver to group the bits.

Besides serial transmission or parallel transmission, data transmission can be categorized in respect of:

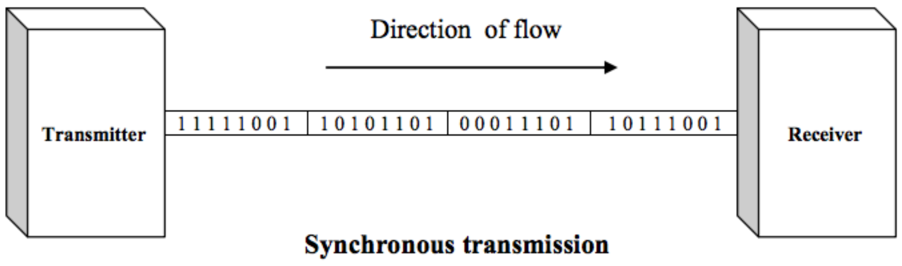


Figure 2.11: Synchronous transmission [38]

.

Number of devices: In a *bus* topology, two or more devices can be connected to the same communication medium. Some kind of addressing is required to select a particular device. On the other hand, *point-to-point* communications allow only two devices to communicate with each other and, as such, no addressing is needed.

Timing of communication: More often than not, the communication between two devices is bi-directional. So it is important to know if both devices can communicate at the same time or not. In *full-duplex* communication, both sides can transmit messages at the same time. In *half-duplex* communication, only one mode can transmit at a time.

Communication hierarchy: It is important to know if all the devices can initiate a transmission or if only some devices can. In *master-slave* systems, only the master can start a transmission. The slave only transmits when inquired by the master. On the other hand, in systems where all devices can transmit data, no device is more important than other. Only some kind of arbitration is needed to manage concurrent access to the transmission medium.

2.3.2 SCI (UART)

The *Serial Communication Interface* (SCI) provides an asynchronous communication interface (*Universal Asynchronous Receiver Transmitter*, UART). This communication interface relies in two wires, one to transmit data (TX) and one to receive data (RX). The TX from one device should go to to RX of the other, and vice-versa. In figure 2.12 we can see the internal structure of a UART module.

The UART inside a microcontroller allows the application to control the behavior of the serial communication, configuring its parameters:

Data chunk: The number of bits carried in each message. The most common value is 8 bits, but values between 5 and 9 bits are allowed.

Parity bit: This is a very simple, low level form of error checking and the user is able to decide whether there should be a parity bit or not, and, in case of yes, whether the

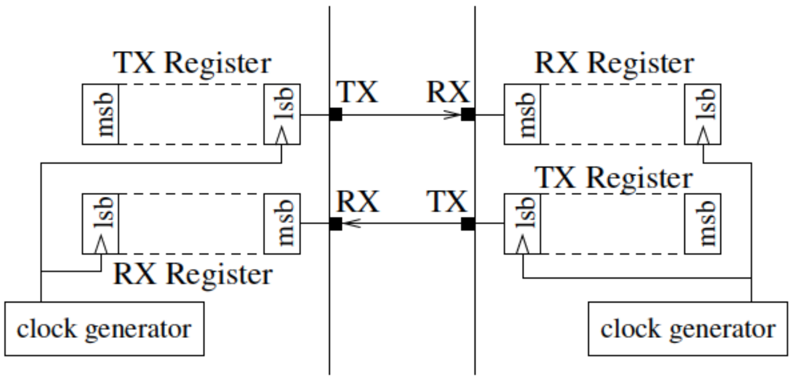


Figure 2.12: Basic structure of a UART module [19]

.

parity should be even or odd.

To test for a parity bit, all of the bits of the message are added up, and is the evenness of the sum that indicates if the parity is even or odd.

Synchronization bits: There are two or more special bits that are transmitted within each message: the start bits and the stop bits. As the number of start bits is always one, the number of stop bits can be configured by the user.

Baud rate: The transmission speed, given in bits per second (bps), can be selected from a set of pre-defined values. These values are within the range from 9600 to 115200 for many microcontrollers. Inverting the value of baud rate allows us to know how long it would take to transmit a bit.

Both devices must operate in the same baud rate.

Message Transmission

Each block of data (usually a byte) is actually sent in a *frame* of bits. This frame is constituted by a start bit, by the bits encoding the message, by an option parity bit and by one or more stop bits. Figure 2.13 shows the general frame format of a UART packet.

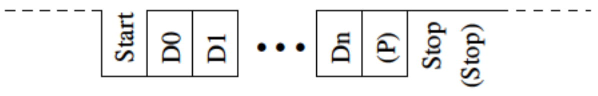


Figure 2.13: UART frame format: In its idle state, the line is high. After receiving the start bit, the line goes low, meaning that a new frame is coming. After the start bit, data bits are transmitted, followed by an optional parity bit. The frame is concluded with one or more stop bits [19]

.

2.3.3 SPI

The *Serial Peripheral Interface*(SPI) is a point-to-point interface communication based on a master-slave principle. It allows full-duplex transmissions between a master (usually a microcontroller) and one or more slaves (sensors or other peripheral devices). This interface relies on four single-ended lines:

MOSI (Master Out, Slave In): Line used to the master send data to the slave.

MISO (Master In, Slave Out): Line used to the slave send data to the master.

SCK (System Clock): Line used for the master to send the clock signal.

SS (Slave Select): Line used for the master to select a slave.

Data Transmission

In SPI there is a clock signal, always generated by the master, that is responsible to synchronize all the communication. When the master sends data to a slave, the message is transmitted through the MOSI line. If some response is needed, the master keeps generating a clock signal with a predefined number of clock cycles, allowing the slave to transmit data back, this time using the MISO line.

In this interface, the master always knows when and how much data the slave has to transmit.

Slave Select

The master must always ask for the slave to communicate, setting its SS pin to LOW. This allows for more than one slave to be connected to the same master, as long as the SS pins of all the slaves but one are HIGH, the master will communicate with the slave whose SS pin is LOW. This, of course, requires a separate SS line for each slave.

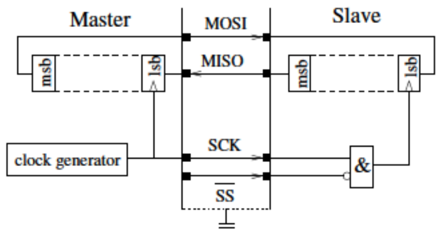


Figure 2.14: SPI interface [19]

.

Advantages And Disadvantages

SPI has many advantages when compared with simple asynchronous serial communication. It allows higher baud rates, supports multi slaves and the receive hardware can be a simple shift register. On the other hand, it requires more signal lines, the communications must be well-defined in advance and only the master can control all the communication.

2.3.4 I^{2}C

The Inter Integrated Circuit (I^{2}C) is a synchronous bus that operates on a master-slave principle. It requires only two wires to communicate: SCL (Serial Clock Line) and SDA (Serial Data Line).

There are three different modes of operation, based on the data speed transmission: *standard mode* for speed until 100 $kbits/s$, *fast mode* for speeds until 400 $kbits/s$ and finally *high-speed mode* for transmission rates to 3.4 $Mbits/s$ [39].

I^{2}C, unlike SPI, allows the presence of multiple masters, due to arbitration mechanisms. Usually the microcontroller is the master, with the peripheral devices or sensors falling to the slave category. If many microcontrollers are present in the system, it is possible to choose if all of them are master, if only one is master and the rest slaves and which ones are which. The only condition is that there must be at least one master in the system.

This interface allows to easily add devices to the system: one only needs to connect them directly on the bus. Figure 2.15 illustrates this process.

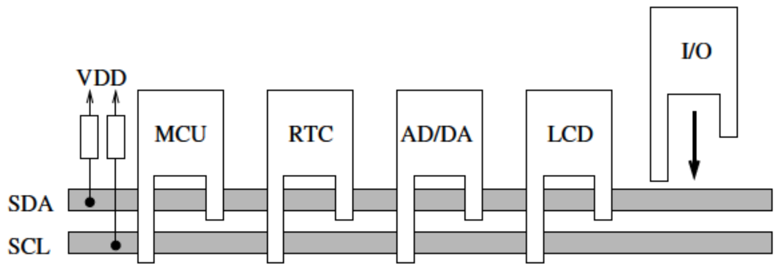


Figure 2.15: Basic configuration of the I^{2}C bus [19]

.

The bus supports both 7-bit and 10-bit addresses, thus up to 1008 devices can be connected to a single bus line when using 10-bit addresses (the bit addresses 0000000XXX and 111111XXX are system reserved, meaning that instead of 1024 addresses (2^{10}) we have only 1008 ($2^{10} - 2^3 \times 2$) available). When using 7-bit addresses, the reserved bits are 0000XXX and 1111XXX, hence the number of available addresses drops to 112 ($2^{10} - 2^3 \times 2$).

Data Transmission

In I^{2}C,voltage levels are defined with respect to a common ground. The low level voltage is in the range of $-0.5V$ to $0.3V_{DD}$ while the high level voltage is around $0.7V_{DD}$ and $V_{DD} + 0.5V$ [39]. Low level voltages are coded as 0 and high level voltages as 1.

While idle, both SDA and SCL lines remain as HIGH. When the master initiates the transmitting, it generates a starting condition (S) on the SDA line.

The following step is for the master to put the address on the bus and indicating if it is a read or write operation. After the R/$\bar{W}$ bit is received by the slave, it sends an acknowledgement bit back, stating that it has recognized the address. After this process is done, data can be transmitted by either the master or the slave, being each bit acknowledged by the receiver.

After all data is transmitted, the master generates either a stop condition (P), indicating that line is now free, or a repeated start condition, indicating that the current transmission is over but starting immediately a new one.

Start and Repeated Start: These conditions are started by leaving the SCL HIGH while pulling SDA LOW. The repeated start uses the same process, while saving one clock cycle. This is particularly useful on systems with multi master, since it allows to keep the ownership of the line. The start condition is represented on figure 2.16.

Only the start and repeated start bits are the only bits allowed to change the level of SDA line while the SCL state is HIGH. All other operations are done when the SCL state is LOW.

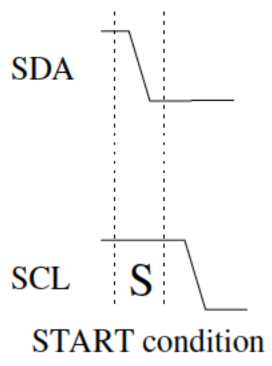


Figure 2.16: Start condition [19]

.

Address Frame: This address frame is the first one to be transmitted in new communications. For a 7-bit address, the first bits are the most significant ones. Like regular data bits, these bits are changed during the low phase of the SCL line.

Direction Control: Right after the 7-bit address is transmitted, the byte is completed by sending an eighth bit specifying the directions of the communication. If the R/$\bar{W}$

bit is HIGH, then the master will want to read data from the chosen slave. If the bit is LOW, it will send data to the slave.

Acknowledgment: The ninth bit of the transmitted frame is the acknowledged ($A\bar{C}K$) bit, sent by the receiver to indicate that it received the transmitted data. This $A\bar{C}K$ is achieved by setting SDA low. If something happens and the receiver does not pull the SDA line to low, it means that either the receiver did not receive the message or does not know how to decode the data.

Data: After the address frame have been sent, the data bits are transmitted like any other bits, and each bit sent must be acknowledged by the receiver. Depending if the R/$\bar{W}$ bit is set to HIGH or LOW, the data is put on the SDA line by either the master or by the slave, respectively.

Stop: The stop condition (P) is given by the master if it wants to free the communication line. It is achieved by pulling the SDA line from LOW to HIGH while the SCL line is kept HIGH. Figure 2.17 shows how this condition is managed.

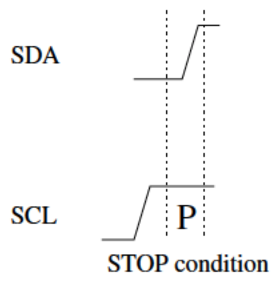


Figure 2.17: Stop condition [19]

.

2.4 Wireless Sensor Networks

In recent years, wireless sensor networks (WSN) have gained high visibility, receiving significant attention from academia, industry, and standards development organizations. The popularity of WSN also means that many business and industries are created around it, with consequences on its market value, which is expected to grow from the actual $450 million to $2000 million in 2022 [40].

A WSN can be defined as a network of (ideally) small size, low cost, low power devices, called *sensor nodes* that are spatially distributed and can sense the environment and then communicate the information gathered through wireless links. The data gathered is then sent to a different device, the *gateway*, that either uses the data locally or is connected to other networks, such as Internet.

Recent advances in many technological areas, such as in micro-electro-mechanical-systems (MEMS), wireless communications and digital electronics have contributed to the massive explosion of low cost, small size, low power and multifunctional sensor nodes [41].

2.4.1 Sensor Node

Sensor nodes are the basic element of any wireless sensor network.

The basic, functional block diagram of a sensor node can be seen in figure 2.18: the microcontroller is the core element of the system, controlling all the operations. Then, one or more sensors are responsible to take data from the environment. The radio-frequency (RF) transceiver whose function consists in receive data from the microcontroller and sending it wireless to either other sensor node or to a gateway.

A modular, flexible design is an essential approach when building these modules, since it allows to easily build and change the device to quickly adjust them to the needs of the user: depending on the physical variable to measure, different sensors can be used and the sensor signal conditioning module can be re-programmed or replaced. In the same way, the radio link may be swapped for other due to range limitation or standard and protocol incompatibilities.

One key feature of any sensor node is its power consumption: as they are remote, independent and wireless (no wires neither to communicate nor to power supply), they need to minimize the power consumed by the system as much as possible. The radio modules and the microcontrollers are the elements that need the largest amount of power, so it is common to put them in a shutdown or sleep mode and only wake them up when needed to make a measurement or transmit data.

Usually the number of sensor nodes is higher than any other element in the network, therefore its costs shall be kept as low as possible.

2.4.2 Main Features

Wireless sensor networks have to face challenges that are not present in traditional, ad-hoc wireless networks:

- resource constraints;
- data redundancy;
- unreliable wireless communication;
- no global identification for sensor nodes;
- prone to node failure;
- large scale deployment.

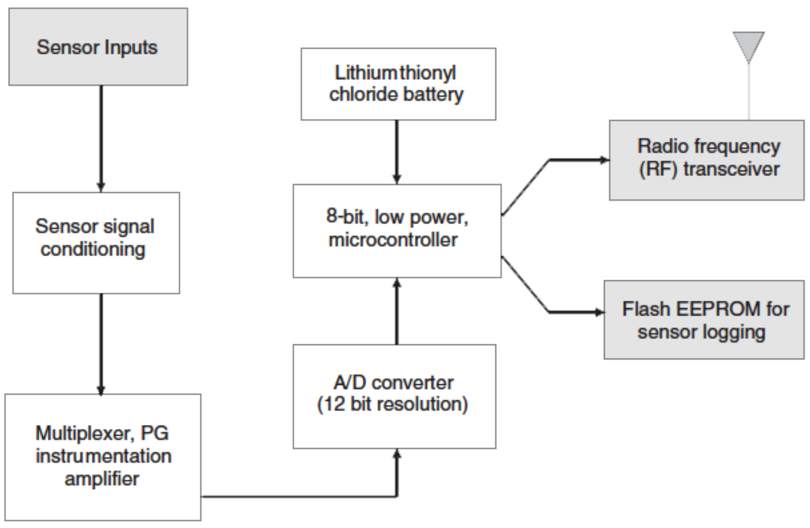

Sensor Inputs
Sensor signal conditioning
Multiplexer, PG instrumentation amplifier
Lithium thionyl chloride battery
8-bit, low power, microcontroller
A/D converter (12 bit resolution)
Radio frequency (RF) transceiver
Flash EEPROM for sensor logging

Figure 2.18: Wireless sensor node functional block diagram [8]

.

Hence, WSNs need to have some special features to allow them to overcome those challenges. The ideal wireless sensor network should have the following characteristics:

- energy efficiency;
- data fusion and localized processing;
- reliability;
- self-organization;
- self-healing;
- scalable network, with low cost, small size and low complexity nodes.

2.4.3 Applications

Sensor networks may consist of many different kinds of sensors, covering this way a wide variety of environment variables, such as

- temperature;
- humidity;
- lightning condition;
- pressure;
- noise levels;
- mechanical characteristics of objects,

which allows them to be used in many application scenarios:

Military: Since sensor networks are based on the dense deployment of small, low cost, disposable sensor nodes, the loss of one sensor does not affect an operation as much as the destruction of a traditional sensor. Sensors can be used to track the overall status of friendly forces, including troops, ammunition and equipment. Battlefield surveillance, reconnaissance of opposing forces and terrain, targeting are other possible applications.

Environmental: Things like forest fire detection, animal tracking, soil monitoring, pollution mapping can be done recurring to wireless sensor networks. Since sensor nodes can be randomly, widely and densely deployed in a open area, they are perfect for this kind of application. For example, in a case of a forest fire, sensor nodes could alert immediately if a fire had started and indicate the approximated location of the origin of it, allowing for a quick intervention.

Health: Sensor networks are mostly used to monitor the disabled: patients can be remotely, real time monitored, with sensors indicating their vital signals and sending the information directly to a doctor, for example.

Home: Home automation and smart environment contribute for a higher quality of life, with sensors embedded in house appliances and communicating with each other.

Commercial applications: WSNs can be used to monitor material fatigue, managing inventory, monitoring product quality, constructing and controlling smart office spaces, interactive museums, process control.

2.4.4 Network Architectures

Different topologies can be applied when building wireless sensor networks. These topologies define how and if the sensor nodes communicate with each other.

Star Network

On the star network topology (figure 2.19), the node devices are only allowed to send or receive messages from a single base station, thus not being allowed to communicate with each other. Nonetheless, the base station communicates with all sensor nodes inside the network.

The advantage of this topology is its simplicity, since designer only needs to worry nodes communicating with the base station and not with each allows, allows to keep the energy consumption at the minimum level, as the sensor node can be in a sleep or shutdown mode almost all the time, only waking up when prompted by the base station to transmitting data. On the other hand, this topology implies that the base station must be within radio transmission range of all sensor nodes.

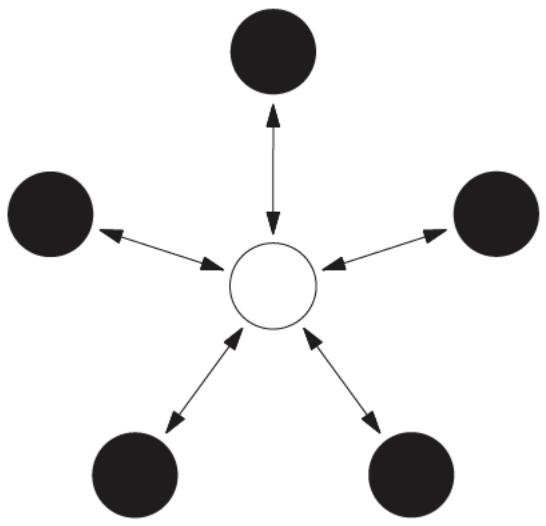


Figure 2.19: Star network topology.

Mesh Network

On mesh networks (figure 2.20) all the nodes on network can communicate with each other, providing they are within the radio transmission range. This allows for what is known as multihop communications, that is, it is possible for a node to send a message to any element in the network, even if out of limits due radio transmission range, using intermediate nodes to transmit the message through.

Despite being more complex, this topology has the advantage of the redundancy and scalability: even if one remote node fails, other nodes can still communicate with other nodes in range, which in turn, are able to forward the message to the desired node or base station. The negative side of this architecture is the power consumption of the individual nodes being higher than the ones in star topology, and the time that message takes to go from one node to the final destination can also be rather high.

Hybrid Star - Mesh Network

Hybrid star-mesh networks (figure 2.21) are the best of two worlds: allows to keep a robust and versatile network with a wide range while keeping the power consumption to a minimum. To achieve this, some nodes are not enabled with the ability to forward messages, being only allowed to communicate with the one node. The other nodes keep the multihop capability, though. This way, they are still able to forward the messages from the low power nodes.

Usually, the nodes with the multihop capability are higher power and, if available, are often plugged into the electrical main line.

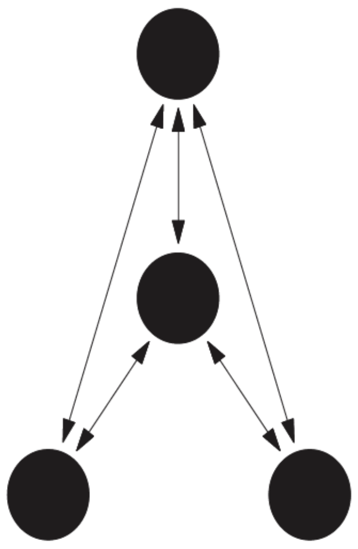


Figure 2.20: Mesh network topology.

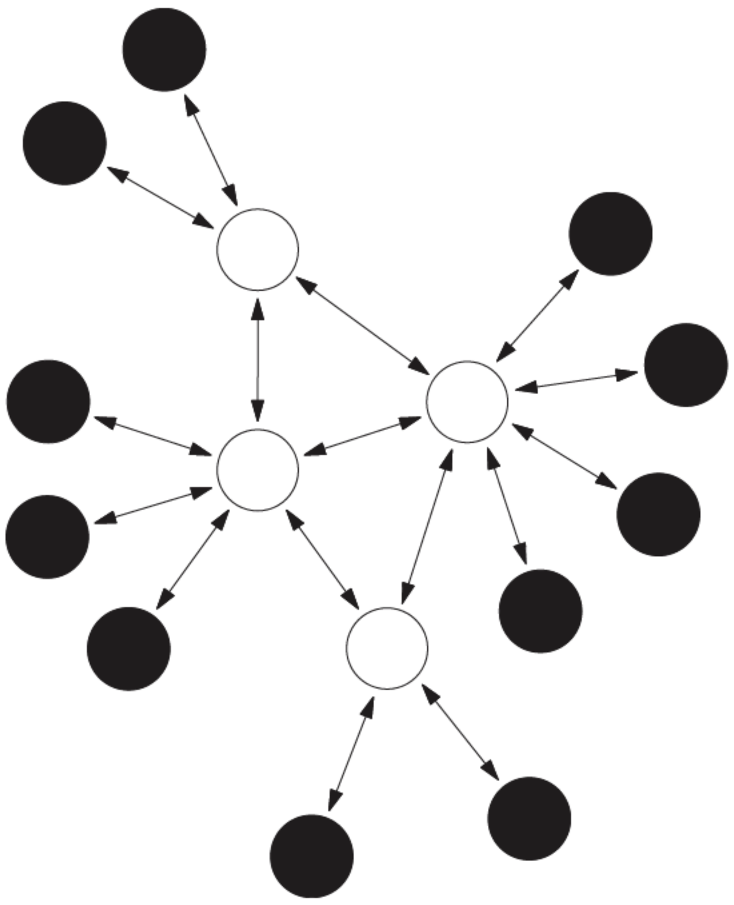


Figure 2.21: Hybrid network topology.

2.4.5 Standards

The physical radio layer of the OSI model [42] defines the signal modulation, the transmitting frequency and the hardware interface of the radio system.

Only node devices working under the same physical layer are able to communicate with each other. Thus standards are a must in this field, since it simplifies the process of creating a wireless sensor network, and modifying it at our will. While using the same physical layer, even products of different companies can be integrated in the same network.

Despite all that, standard protocols may not always be the best choice for all the situations. There are many low power proprietary radio modules from companies such as ATMEL[4], Microchip[5], Nordic Semiconductors[6] or Texas Instruments[7], that are also valid choices for wireless sensor networks. The final decision will always have to be taken by the network designer after weighting all the facts.

IEEE 802.11x

IEEE802.11, also known as Wifi, is a standard meant for local area networks, with high data transfer needs. The most recent 802.11ac allows top transfer speeds starting on 433 Mbps (mega bits per second) till several Gbps (gigabits per second) and the range is typically around 100 meters in line of sight, with a standard antenna.

As is easily perceptible, these values of range and speed imply a high power consumption, making this standard ideal to connect computer and multimedia devices due to its high data transfer rate but not to sensor nodes.

IEEE 802.15.4

IEEE 802.15.4 is a standard defined by IEEE 802.15.4 Working Group for data communication devices operating in Low Rate Wireless Personal Area Networks (LR-WPANs) [40]. This standard is specially targeted to sensing network applications due to its low cost, low power, low data-rate and short range communication features which privileges battery operated devices with short-range needs.

There are two kinds of nodes in this network: full-function device (FFD) and reduced-function device (RFD). RFDs are limited devices, with low processing power and low memory, used mainly as end devices of the network and are able to communicate only with FFDs. FFDs are able to fully implement the standard and act as coordinators of the personal area network (PAN) and are able to communicate with both RFDs and FFDs. Based on the characteristics of the devices, two topology architectures are supported by the standard: star networks or mesh networks.

[4]`http://www.atmel.com/default.aspx`

[5]`http://www.microchip.com/`

[6]`http://www.nordicsemi.com/`

[7]`http://www.ti.com/`

The original 2003 version of this standard supports 868/915 MHz low bands, with data rates of 20 and 40 *kbps*, and 2.4 GHz high bands with a rate of 250 *kbps*. The current version of the IEEE standard is 802.15.4-2006. It improves the maximum data rates 868/915 MHz bands up to 100 and 250 *kbps* respectively [40].

The physical and medium access layers of the standard are implemented through the following protocols:

Zigbee: The Zigbee^{TM} Alliance[8] is an association of companies working together to achieve reliable, cost-effective, low-power, wireless networked systems products based on an open global standard. This protocol stack is composed of four main layers of the OSI model: physical layer, medium access layer, network layer and application layer [43].

Besides Zigbee end devices, a Zigbee coordinator and a Zigbee router are required to build a network. By default, the end devices and Zigbee coordinator lack an Internet Protocol (IP) gateway to communicate with an IP network. Hence, Zigbee is not suited to deal with applications that require to interface with IP devices.

6LoWPAN: The name stands for IPv6 over Low power Personal Wireless Area Networks and is a protocol that has been defined by Internet Engineering Task Force[9] (IETF) to adapt IPv6 communications on top of IEEE 802.15.4 networks [44].

This protocol allows IPv6 packet transmission over low power and low rate IEEE 802.15.4 structures and assures compatibility with other IP devices, allowing them to communicate directly with each other.

IP for Smart Objects (IPSO) Alliance[10] is one of the main forces behind this protocol, promoting the use of 6LowPAN and embedded IP solutions in smart objects.

WirelessHART: Wireless Highway Addressable Remote Transducer (HART) adds wireless capabilities to the existing HART[11] protocol, while maintaining compatibility with default HART devices. Hence, this protocol is widely used in industrial applications.

WirelessHART operates in the 2.4 GHz ISM band and uses a method called frequency hopping to prevent interference from other applications.

Bluetooth and Bluetooth Low Energy

Bluetooth was previously regulated by the IEEE 802.15.1, but this standard is no longer maintained. Instead, Bluetooth is managed by the Bluetooth Special Interest Group, which adopted Bluetooth Core Specification Version 4.0 in 2010.

[8] `http://www.zigbee.org/`
[9] `https://www.ietf.org/`
[10] `http://www.ipso-alliance.org/`
[11] `http://en.hartcomm.org/`

This protocol is designed for short range, high data-rate communications, allowing data-rates till 3 Mbps and covering a range of 10-100 meters [40].

The most recent version of the protocol brought us Bluetooth Low Energy (BLE) technology, providing low power capabilites to the standard Bluetooth protocol. Thus, new devices using BLE technology and able to operate for months with a coin-cell battery are now possible. Hence, BLE is a well suited protocol for WSNs.

BLE operates in the same 2.45 GHz ISM frequency as classic Bluetooth, but has some functional differences. Instead of 1 Mhz channel, BLE uses 2 Mhz channels: practical effects are a reduced power consumption, lower data-rate (up to 1 Mbps) and higher range (within 200 meters).

Depending on the mode of operation (single-mode or dual-mode), BLE devices may be standard Bluetooth retro-compatible (dual-mode) or not (single-mode).

Z-Wave

Z-Wave was developed by Zensys[12] and is maintained by Z-Wave Alliance[13].

When compared to IEEE 802.15.4 standard, Z-Wave has the advantage of operating on the less overloaded under 1 GHz band (around 900 MHz), not being affected for interferences from the popular 2.4 GHz band. In Europe, due community regulations, Z-Wave operates on the 868 Mhz frequency.

Z-Wave supports star and mesh networking topologies and data rates of 9.6 *kbps* and 40 *kbps*.

Overview

The IEEE 802.15.4 standard is specifically designed for wireless sensor networks communication, due to low cost, low data rate and low power characteristics.

If we compare this standard with the widely used 802.11 standard (Wifi), this one provides faster data rates and a wider range but at costs of a considerable high power consumption, which turns it incompatible with sensor networks.

Bluetooth Low Energy (BLE) is one standard alternative to wireless sensor networks applications that demand a higher data rate, but short ranges.

2.4.6 Proprietary RF Links

Proprietary radio frequency modules can also be used to create WSNs. They tend to be quite cheap and small and can provide communication ranges between 10 meters and 1 kilometer, depending on the transmission power and antenna module. The data rate is variable between manufacturers and sometimes can even be chosen by the network designer, but typical values range from 250 *kbps* to 2 $Mbps$.

[12]`http://www.zensys.com/main.html`

[13]`http://z-wavealliance.org/`

When comparing with standard protocols, proprietary modules lack the versatility and adaptability of the first ones, making them more difficult to interact with an existing WSN: usually devices using proprietary radio modules are only able to communicate with devices using the same proprietary protocol.

Nonetheless, if the ultimate goal is to create a WSN that we previously know that all the devices will be using the same radio module, proprietary RF modules can be a perfectly acceptable choice.

Chapter 3

Prototyping Internet Of Things Devices

The term Internet of Things (IoT) was first introduced by Kevin Ashton, in 1999 [45], and although almost every author has its own definition of what IoT is, the broad vision is the pervasive presence of objects or *things* around us - such as sensors, actuators, wearables, Radio-Frequency IDentification (RFID) tags - which through appropriate addressing schemes are somehow connected to each other through the biggest network of all: the Internet [46]. In other words, IoT allows to connect the Internet to the physical world, and vice-versa. It is expected that more than 50 thousand millions of objects will be connected to the Internet by 2020, making the IoT one of the most potentially disruptive technological revolution of our lifetime [47].

With so many devices and products becoming available, rapid prototyping is an essential quality to face the new challenges and opportunities brought by the IoT vision [48].

The democratization of technology [34] made this prototyping possible to everyone, giving the opportunity to people like hobbyists, electronic enthusiasts or even hardware start-ups to create their own products [49].

3.1 Basic Concepts

Independently of the definition of IoT, the real concept behind it does not change: devices are part of the virtual world of Internet and interact with it by tracking, sensing, and monitoring objects and their environment. So, a device should be able to:

Collect and transmit data: Equipped with sensors, IoT devices are able to sense the environment and either act based on the reading or transmit it to a different device or directly to the Internet.

Actuate based on triggers: It can be programmed to take actions based on pre-defined conditions set by the device designer.

Receive Information: Besides collecting and transmitting data, devices are also able to receive data from other devices present in the network they belong to.

Communication assistance: Devices part of mesh networks are also responsible to help transmitting message, forwarding received data to the final destination.

3.2 Do It Yourself

Do It Yourself (DIY) refers to a cultural movement of people doing and making things themselves [50]. With the rapid proliferation of IoT capable devices and the so called technology democratization, many people started to make their own products, using existing technology. This is the idea behind the *maker movement* [51].

Maker Movement

The maker movement is based in three pillars: curiosity, creativity and community [49]. It is focused in creating products or developing projects while learning in the process and sharing knowledge between the pairs. This is a technological movement that started in the 1960s, when Stewart Brand launched the Whole Earth Catalog[1], a revolutionary[2] publication focused on Do It Yourself guides and how-to's. This book inspired the creation of communities based on the DIWO (do it with others) moto, which later contributed to the free and open source software and hardware movements.

In our days, the *maker movement* is still present and growing in popularity. DIY communities, such as Instructables[3] or Adafruit[4] have a large base of active users and events like Maker Faire[5], focused in hobbyist projects, have increased in popularity, going from 200 exhibits and 20000 attendees in 2006 to 900 exhibits and more than 120000 attendees in 2013 [49]. Also, community-driven Mini Maker Faires [6] have been organized world-wide. Lisbon, for example, had its first Mini Maker Faire on September 2014 and the second edition is already being prepared to September 2015[7].

Contributing Factors

Some factors have changed over the years that allowed the use the DIY concept as a base to commercial products and solutions, to both individual maker and hardware start-ups:

Rapid Prototyping: Advances in prototyping technologies have drastically changed the process of taking an idea from paper to the physical world:

[1]`http://www.wholeearth.com/index.php`
[2]`http://www.theguardian.com/books/2013/may/05/stewart-brand-whole-earth-catalog`
[3]`http://www.instructables.com/`
[4]`https://www.adafruit.com/`
[5]`http://makerfaire.com/`
[6]`http://makerfaire.com/mini/`
[7]`http://makerfairelisbon.com/en/`

- 3D printing machines, capable of building plastic, ceramic and metal high precision products, have become really popular in the last years and its price has dropped significantly [52];
- prototyping boards such as Arduino[8], Raspberry Pi [9] or BeagleBone[10] are low cost products with a huge community of hobbyist and makers, making the electronic prototyping accessible to everyone;
- IoT programmable, focused products such as Electric Imp[11] Particle[12] or Pinoccio[13] allow to effortlessly create a full IoT platform while keeping the costs low;
- CAD software has become more sophisticated and easier to use. Fritzing[14] is an example.

Inexpensive Components: Discrete electronic component prices have come down, in a similar way of the prototyping boards. As this, acquiring sensors, leds, batteries, actuators and all of the typical prototyping material has never been easier. Besides that, many new companies dedicated to maker business have been created and are quite popular inside the maker community, providing not only products but also tutorials and guidance. Makerbot[15], Sparkfun Electronics[16] or Adafruit[17] are some of the most popular worldwide. In Portugal, companies as InMotion[18], PTRobotics[19] or Leds & Chips[20] are positive example of this kind of companies.

Small-batch manufacturing: The minimum number of product units needed to secure a manufacturer contract used to be quite high, hence quite expensive. But this paradigm is changing, with factories allowing to do small-batch runs of a product.

This way the initial investment is more affordable and even if a run is ruined due to bad hardware or software iteration, the amount of money lost is reduced.

Open Source Hardware: Open source hardware platforms are a rapid, easy and affordable way to prototype products and test them before a proprietary board is made. Arduino is probably the biggest and most successful example of an open source hardware platform. Raspberry Pi, though not totally open source, benefits from similar popularity and allows to easily make the connection between the physical world and the many high-level languages available in the computers world.

[8] `http://arduino.cc`
[9] `https://www.raspberrypi.org/`
[10] `http://beagleboard.org/bone`
[11] `https://electricimp.com/`
[12] `https://www.particle.io/`
[13] `https://pinocc.io/`
[14] `http://fritzing.org/home/`
[15] `http://www.makerbot.com/`
[16] `https://www.sparkfun.com/`
[17] `https://www.adafruit.com/`
[18] `http://inmotion.pt/`
[19] `http://www.ptrobotics.com/`
[20] `http://ledsandchips.com/Home/`

Open Hardware is promoted by The Open Source Hardware Association[21].

Open Source IoT platforms: Open source platforms like Freeboard[22] or EmonCMS[23] allow to quickly create and visualize dashboards for Internet of Things devices. These platforms are highly configurable and can be run locally or in web servers, allowing to create products and applications around them.

Online communities: Web based communities like Instructables allow people around the world to share their projects online and to learn from others. On the other hand, crowdfunding platforms, such as Kickstarter[24] or Indiegogo[25], have become rather successful in the last years, helping many individual makers and start ups to achieve the needed funds to advance with a submitted project.

3.3 Arduino

Arduino is an open-source prototyping platform based on flexible, easy-to-use hardware and software. It is constituted for both the physical microcontroller boards and the Integrated Development Environment (IDE) software. The Arduino platform has become quite popular in the *maker* community, due mainly to its simplicity and affordability: no additional hardware is needed to program most of the boards beyond an Universal Serial Bus (USB) cable and ready-to-use official boards are available from €16,00.

3.3.1 Boards

At the present date there is a total of 12 different boards available to buy in the official store[26], each one of them serving different purposes and applications.

Arduino ready-to-use boards are basically formed by a microcontroller, some additional electronic components, such as a voltage regulator or a Universal Serial Bus (USB) to Serial interface, and some socket headers to directly connect jumper wires.

Microcontroller

Arduino uses public available AVR microcontrollers in all of their boards but one (Arduino Due uses an ARM processor), with no modifications whatsoever in terms of the microcontroller itself. The only thing that differentiates Arduino microcontrollers is the custom boot loader, which is specially designed to allow microcontrollers to be programmed through the Arduino IDE.

The following microcontrollers can be found on Arduino boards:

[21]`http://www.oshwa.org/`
[22]`http://www.freeboard.io`
[23]`http://emoncms.org/`
[24]`https://www.kickstarter.com/`
[25]`https://www.indiegogo.com/`
[26]`https://store.arduino.cc/category/11`

ATmega328p [53]: the core microncontroller of the Arduino platform. It features a 8-bit CPU, 32 kB flash memory, 23 Input/Output pins and frequencies up to $20MHz$. Though being neither the fastest nor the most complete microcontroller, it is more than enough for many projects.

ATmega2560 [54]: The bigger vesion of ATmega328p: more Input/Output pins, more memory, more available communication interfaces.

ATmega32U4 [55]: This microcontroller is pretty similar to Atmega328p, with the big difference being ATmega32U4 featuring integrated USB-to-Serial into the chip.

Table 3.1 gives an overview comparison between these microcontrollers.

3.3.2 Arduino IDE

Arduino IDE is the software used to write and upload programs to Arduino boards or Arduino *bootloaded* compatible microcontrollers, that is, microcontrollers with the Arduino boot loader flashed.

It is an open source project, compatible with all major operative systems: Windows, Linux and OS X.

Arduino IDE uses the 'Arduino programming language' which is based on Wiring. Although it may seem strange to implement its own programming language to write software to AVR microcontrollers, that is not the case. Arduino IDE just offers an out of the shelf interface to the language used to program generic microcontrollers: C and C++. With many out of the shelf functions available, it makes the process of one writing its own first program easier. On the other hand, all the features and advantages of C and C++ languages are also there: it is possible to upload a program written exclusively in C or C++ to an Arduino, or combine both.

One of the disadvantages of the IDE is that it lacks some of the advanced features of other commercial IDEs, such as a dedicated debugging tool.

3.3.3 Arduino Success

Arduino platform is a great product and that alone could be a the explanation for its popularity and acceptance inside the maker community. However, Arduino offers some other unique characteristics [56]:

Affordable

Official board have prices starting on €16,00. This value, though low, is still high when compared with bare microcontollers, whose prices are around a couple of euros. But, since arduino is an open source hardware platform, it is possible to replicate the hardware which leads to many distributors selling ready to use arduino boards really cheap. Besides that,

	ATmega328p	ATmega2560	ATmega32U4
CPU	8-bit AVR	8-bit AVR	8-bit AVR
Pin Count	32	100	44
Max I/O Pins	23	86	26
Ext Interrupts	24	32	13
Flash Memory ($KBytes$)	32	256	32
EEPROM ($Bytes$)	1024	4096	1024
SRAM ($KBytes$)	2	8	2.5
Maximum Frequency (MHz)	20	16	16
Operating Voltage (Vcc)	1.8 to 5.5	1.8 to 5.5	2.7 to 5.5
SPI	2	5	2
I^2C	1	1	1
UART	1	4	1
ADC channels	8	16	12
ADC resolution (bits)	10	10	10
ADC speed (ksps)	15	15	15
PWM channels	6	15	8
Timers	3	6	4
Watchdog	Yes	Yes	Yes
USB interface	No	No	Yes
Price (USD)	1000 @ USD 1.88 each	1000 @ USD 10.45 each	1000 @ USD 3.51 each

Table 3.1: Comparison between ATmega328p, ATmega2560 and ATmega32U4 microcontrollers.

it is possible to buy individual microcontrollers, and flash them with Arduino boot loader, and easily build our own boards.

Multiplatform

As previously said, Arduino IDE is available to download to all main operative systems. It is not common for microcontrollers IDEs to support Mac OS X and Linux, since usually they were available exclusively to Windows.

Libraries

Arduino has a great repository of libraries available, most of them written and shared by and to the community as open source projects itself. This turns the process of dealing with common hardware and objects really easy, since it is possible that someone has already designed a library for what we want to do.

Performance

There are no downsides in performance for programming Arduino compatible microcontrollers with its dedicated IDE and language over pure C or C++. The code is compiled before being uploaded and then runs directly on the chip.

Sensors

Arduino boards are compatible with almost every sensor in the market: its analog inputs makes it able to read data from analog sensors. Its communication interfaces, such as I^2C and SPI make it compatible with digital sensors. And beyond hardware compatibility, if it happens that we are dealing with a popular sensor or devices, there is a high chance that a library to easily deal with it already exists.

Open Source

Besides the cost advantages, being an open source hardware platform makes Arduino available to be used for everyone, even for companies in commercial products. Furthermore, the development of the project is not dependent of any company, meaning that there is no risk of it being discontinued and the software gone. Also, if we want a different feature, we can always add it by ourselves, starting from a solid base.

Shields

If all the available official boards are not enough for a project, there is the possibility to add shields to Arduino. There are shields to add Ethernet to arduino boards; there are shields to add Wifi; there are shields to help dealing with relays, for example. If something demands more than what Arduino can offer in terms of hardware, maybe it can be done recurring to a shield.

3.4 Raspberry Pi

The Raspberry Pi is a low cost, credit card sized, single board computer introduced in 2012 by Raspberry Pi Foundation [57, 58]. It was created with the main objective of providing the basic computer skills for future computer science undergraduate applicants of the Cambridge's University, reincarnating what happened on early 1990s, where applicants used to have those skills, due to 1980s home computers, which required programming and were open to hacking [59].

At the moment there are three different Raspberry Pi boards: Model A, Model B and Model B+, whose characteristics can be seen on table 3.2.

	Model A	Model B+	Model B 2
System on Chip	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, one USB port	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, one USB port	Broadcom BCM2836 (CPU, GPU, DSP, SDRAM, one USB port
CPU	700 MHz single-core ARM1176JZF-S	700 MHz single-core ARM1176JZF-S	900 MHz quad-core ARM Cortex-A7
SDRAM ($MBytes$)	512	512	1024
USB 2.0 ports	1	4	4
On-board storage	MicroSD slot	MicroSD slot	MicroSD slot
On-board network	No	Ethernet	Ethernet
Operating Voltage (Vcc)	5	5	5
Number of Pins	40	40	40
Number of generic GPIO	17	17	17
SPI	Yes	Yes	Yes
I^{2}C	Yes	Yes	Yes
UART	Yes	Yes	Yes
ADC channels	No	No	No
Price	€20	€25	€35

Table 3.2: Comparison between different Raspberry Pi models.

3.4.1 Hardware

Looking at the table 3.2 we can find both specifications that we usually find on end user personal computers and specifications that we typically find on microcontrollers. With CPUs clocked at 700 or 900MHz, 512 $MBytes$ or 1$GByte$ of RAM, LAN connectivity, storing capacity in the order of *Gigabytes* - due to Memory Card expansion - Raspberry Pi has enough processing power to run a complete operative system. On the other hand, it offers direct access to 40 GPIO header pins. These pins are no different from the ones we can find on traditional microcontrollers, with digital input and output generic pins and pins dedicated to communication interfaces such as SPI, I^2C and UART.

Its small form factor and versatility makes it the ideal choice for projects where we need both the versatility of a microcontroller and the processing power and connectivity of a personal computer.

3.4.2 Software

Raspberry Pi is no different that an ordinary personal computer in terms of Inputs and Outputs, that is, we can connect a keyboard and a mouse as input devices and have a monitor as output. The big difference between Raspberry Pi and traditional personal computers is the CPU's architecture, with Raspberry Pi using ARM architecture CPUs instead of the traditional X86 or X86-64 architecture. It means that operative systems and all programs have to be compiled with this in mind, or otherwise they won't work on Raspberry Pi.

Fortunately the list of compatible Operative Systems and software is vast enough and well documented, making the process of installing a bare board quite easy and straightforward.

Operative System

Raspberry Pi use dedicated Linux distributions as its operative system. It means that many users are ready to work and explore Raspberry Pi even before putting their hands in one. The most used operative system is Raspbian, which is based on the Debian distribution.

Programming

Having a standard operative system as its base allows Raspberry Pi to use high-level language and functions, being Python the most common choice. Most distributions come with ready to use libraries that help dealing with the GPIO pins available.

3.4.3 Communication

Some Raspberry Pi models come with an integrated Ethernet port, allowing the device to be connected directly to the Internet, due to automatically receive the details it needs to access the Internet when it loads its operating system through the Dynamic Host Configuration Protocol. This gives an Internet Protocol address on the network to the Raspberry Pi. Models where this port does not exist, it is also possible to connect the Raspberry Pi to the Internet using a Wifi or 3G dongle connected to one of the device USB ports. This operation can be a little more complicated since it may imply to manual install the drivers of the new peripheral. On the bright side, it is a process that is done only once.

Device to Device communication

Along with Internet connectivity it is often useful to communicate with other devices or peripherals, such as microcontrollers. Raspberry Pi is able to do that using its communication interfaces: UART, I^{2}C and SPI. This feature gives Raspberry Pi great advantages to be used as an IoT gateway: it is easily connected to the Internet and to microcontrollers, which are the core of sensor nodes [60].

3.4.4 Flexibility

Raspberry Pi can be used connected to a monitor, mouse and keyboard or can be used 'headless', that is, without any peripherals connected. The interaction with the device is made through a *Security Shell* (SSH) connection, and if needed, using a Virtual Network Connection for graphical access. This allows to place the Raspberry everywhere, as long as a wall power socket is present.

Since it has been released, in 2012, Raspberry Pi has been chosen to feature many different applications: private home server to backup files, multimedia device, 3D printers, IoT applications, home weather stations. Like Arduino, many third-party shields are available to help in specific projects.

3.4.5 Overview

Raspberry Pi is an excellent affordable device, and its advantages can be summarized as follows [61]:

- complete, small and independent computer, running a fully featured Linux based operative system;
- large amounts of RAM memory when compared with microcontrollers;
- expandable memory to store data, due to its memory card slot;
- speeds from 700 MHz to 900 MHz;
- native support for USB 2.0 ports;
- Wifi, Bluetooth, 3G support, via USB dongles or shields;
- interaction with microcontrollers and other electronic devices, through its communication interfaces: UART, SPI and I^{2}C;
- high level languages can be used to program the Raspberry Pi;
- can be run in server mode ('headless');

Although these are great features for the price, there are some useful features lacking:

- does not have a real time clock (RTC) with a backup battery;
- does not have any built in Analog to Digital Converters;
- when buying a Raspberry Pi, along the board itself, we need to also buy the basic needed accessories: SD card and power supply.

3.5 Other Products

Besides Arduino and Raspberry Pi, there are other boards available on the market that are being used to prototype IoT devices or were made with that single purpose.

3.5.1 MSP430

MSP430[27] is an affordable prototyping board developed by Texas Instruments. It features a MSP430G2553 microcontroller, whose main characteristics include a 16 *MHz* CPU, 16 *KBytes* of Flash memory, 512 *Bytes* of RAM, eight 10 bits ADCs and up to 1 I^{2}C, 2 SPI and 1 UART.

Texas Instruments offer the possibility to download its IDE, the Code Composer Studio, for free. Unfortunately the IDE is only available to Windows and Linux platform. On the other side, there is an open source project called Energia that is trying to bring the Arduino framework to the Texas Instruments MSP430 based boards, which supports the Windows, Linux and OS X.

When compared with Arduino boards, this board is cheaper, with prices starting on €10,00. However, it lacks the support and huge amount of existing code given by Arduino community and the third-party shields.

3.5.2 Beaglebone Black

Beaglebone Black[28] is a open source hardware board owned by Beagle. This board is similar to the Raspberry Pi in terms of philosophy, since they both can be described as being low cost, credit card size computers designed with the developers and hobbyists in mind.

When compared to Raspberry Pi, Beaglebone Black offers some advantages, such as more GPIO pins, more communication interfaces, built in ADC channels and storage memory, which is pre-programmed with an operative system. These features make Beaglebone Black an excellent choice to electronic projects.

In terms of drawbacks, this board is more expensive, with prices starting at around €50,00.

[27]http://www.ti.com/ww/en/launchpad/launchpads-msp430.html
[28]http://beagleboard.org/black

3.5.3 Electric Imp

Electric Imp is a wifi module combined with a Cortex-M3 processor core and 6 flexible I/O pins supporting UART, I^2C, SPI, analog in and out, PWMs and all of this fitted into a SD card size case [62]. The development of this board is always done via wireless communication, since Electric Imp has a dedicated web IDE to program and interact with the device. Even the initial configuration is done without using additional hardware, using a smartphone to optically transfer the credentials of the wifi network.

Since all the development and interaction is done via a web IDE, we are already in the presence of an out of the shelf connected device.

In terms of price, each Electric Imp module costs around €30,00.

3.5.4 Pinocc.io

Pinoccio is a open hardware and software platform, based on Arduino [63]. It provides both the hardware devices and the necessary web services to easily create a wireless sensor network.

Pinoccio devices, the *Scouts*, use an ATmega microcontroller, 802.15.4 radio modules and wifi modules to create a network of devices connected to each other and to the Internet. The *Lead Scout* has both wifi and 802.15.4 radio modules, while *Field Scouts* are only equipped with radios.These devices cost around €140,00 and €60,00 each, respectively.

3.5.5 Photon

Photon is Particle's Internet of Things hardware development kit. This device combines a ARM Cortex M3 microcontroller with a Broadcom wifi chip in a tiny thumbnail-sized module called the PØ (P-zero) [64]. This module as a series of GPIO pins that allow rapid prototyping and testing of products.

This platform offers its own IDE, which can be used either locally or via web browser. It is also possible to deploy the firmware over the air, as long as the device is connected to the Internet.

There are also an integrated development language - ParticleJS - and a Mobile SDK to help building web and mobile applications.

Photons can be bought for around €25.

3.5.6 ESP8266

ESP8266 is a cheap wifi module, able to act in stand alone mode or connected to a microcontroller [65]. These modules have some of the following features:

- 802.11 b/g/n wifi;
- integrated low power 32-bit MCU;

- integrated 10-bit ADC;
- SPI, UART, I^2C, IR Remote Control, PWM, GPIO;
- integrated TCP/IP protocol stack;
- supports antenna diversity;
- wifi 2.4 GHz, support WPA/WPA2;
- deep sleep power <10 μA, Power down leakage current <5 μA;
- wake up and transmit packets in <2 ms;
- standby power consumption of <1.0 mW;
- +20 dBm output power in 802.11b mode.

Since it was introduced, in 2014, it has caught the attention of the maker community, because it represents an easy and low cost alternative to add wifi to any microcontroller, since these modules can be bought for prices around €5.

When compared with the other products mentioned, ESP8266 has the disadvantage of not having any big company supporting the product. Also, the official documentation is scarce, difficulting the interaction with the module. On the other hand, it has a growing and active community, having inclusive developed an Arduino IDE specifically designed to interact with ESP8266 [66].

Chapter 4

System Architecture and Implementation Choices

The developed system was designed and built to create a modular platform able to easily connect sensor node devices to the Internet, allowing to remotely monitoring, registering and displaying a wide number of physical quantities. This goal was ultimately achieved creating a wireless sensor network of low cost, small form factor, mainly open hardware, commercial off-the-shelf components.

Building a wireless sensor network system requires development and integration of many hardware and software components. Figure 4.1 shows the overall architecture of the wireless sensor network system developed. It follows the typical IoT application architecture that can be divided in three layers [14]:

Perception Layer: In this layer, the system aims to acquire, collect and process the data from the physical world, which consists of two parts: the sensor device and wireless sensor networks. The former one includes the microcontroller, the sensors used and the radio transceiver. The latter is a self-organizing wireless network using a high number of sensor nodes distributed in a large area.

Transmission Layer: In the transmission layer, the system aims to transfer the data, collected from the perception layer, to a large area or long distance. Once the data is collected, it can be transferred to a remote location. Thus, this layer relies heavily on mobile broadband communication network and wifi.

Application Layer: In the final destination, the data from transmission layer will be used to data processing and to service providing, the two major purposes of the application layer.

In this scenario, my tasks were the following:

- choose the appropriate components to build the wireless sensor network main components: node devices, gateway and the RF link;

- make sure that gateway could properly communicate with each sensor node: identify the node with which it was communicating and receive a valid message;
- configure the gateway to send a valid request to the web application responsible to store the values.

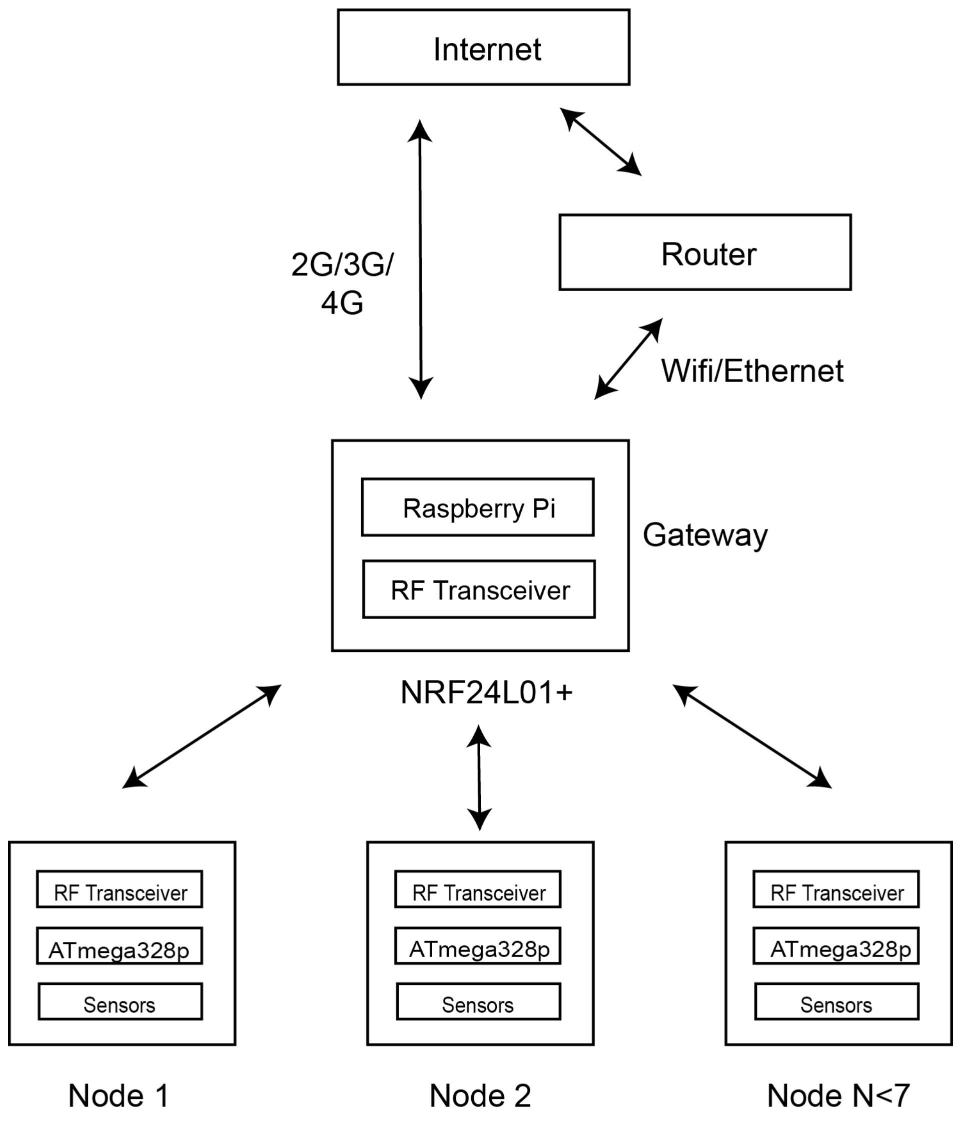


Figure 4.1: Overall WSN system architecture.

4.1 Sensor Node

In this architecture, sensor nodes are responsible to measure the desired physical quantity, process the information and then transmit it to the gateway.

In terms of components, sensor nodes were constituted for the following material:

- microcontroller;

- radio frequency transceiver;
- sensors;
- batteries.

4.1.1 ATmega328p

The microcontroller's choice fell on ATmega328p, from Atmel. Atmel and its ATmega family were the obvious choice from the beginning, since using the Arduino platform was planned, so we needed to be able to flash the Arduino bootlader on the chip.

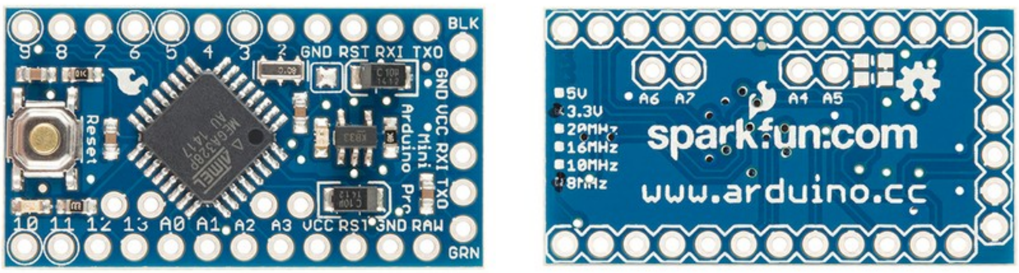
The fundamental characteristics of this microcontroller can be seen on table 3.1. From the compared microcontrollers, this was the one with less Input/Output pins, which can be important in terms of scalability, less timers and less SRAM memory. On the other hand, this chip was the cheapest of the trio.

As it is intended to create a generic sensor node, the reduced number Input/Output pins of the ATmega328p is not a problem, since it is enough for most of the situations. In cases where more Input/Output is needed, it is possible to just change the microcontroller. Furthermore, the price is a decisive factor: as long as the microcontroller is fit for the task, the chosen one will always be the cheapest one.

ATmega328p is also the chip present in many of the boards sold by Arduino and other third party retailers, with Arduino boot loader already flashed.

It is possible to buy individual, ready to use boards, such as the Arduino Pro Mini, with prices around €5. Usually there are two versions available to buy: one operating at 5 Volts and 16 MHz and the other one operating at 3.3 Volts and 8 MHz. Our nodes used the 3.3 Volts version, since it implies a lower power consumption.

Figure 4.2 shows the board used during the development of the platform.



(a) Front view (b) Back view

Figure 4.2: Board used during the prototyping [67].

4.1.2 Sensors

A general overview of sensors can be seen in section 2.2. The sensor nodes should be compatible with as many sensors as possible, contributing this way to the modularity of the system. Since the nodes have 6 ADC channels, 1 UART, 1 SPI and 1 I^2C line, and 23 GPIO pins it is safe to say that most of the sensors are possible to use as sensing components in our system.

4.1.3 Radio Frequency Transceiver

We initially aimed to open source technology and open standards, thus options such as Zigbee or BLE, whose features are explained in the subsection 2.4.5, would be our intended choice. But all of these alternatives had a cost: radio modules supporting them do not have the lowest prices and have limitations [68]:

- Zigbee radio transceiver modules are limited to 250 *kbps* and typically have a price between €0.8 and €12. Popular modules such as CC2530 from Texas Instruments [69] or MRF24J40 from Microchip [70] are available from €6.89 and €8.78 respectively, at the Digikey[1] store.
- Bluetooth Low Energy radio transceivers have a maximum speed of 1 *Mbps*, and prices range from €5 to €15. BLE modules such as CC2540 from Texas Instruments [71] have prices starting on €4.78 in the same store.

In the end, the choice fell on NRF24L01+ module [72]. This is a proprietary radio transceiver module, operating in the 2.4 *GHz* band, with software programmable radio transfer speeds: 250 *kbps*, 1 *Mbps* or 2 *Mbps*. These modules can be bought on Digikey from €2.97.

As we can see on table 4.1, this module has the highest data rate transfer speed, the lowest receiving and transmitting current consumption and the lowest price, which makes it a perfectly valid option.

A detailed description of this module can be found on section 4.2.

4.1.4 Device Software

The device was programmed with Arduino IDE, and the generic program applied to each device can be divided in the following parts:

Importing libraries: Libraries are modules that can be imported and that provide extra functionality for use in code, such as working with hardware and manipulating data. Arduino IDE has a set of built in libraries [73].

In our case, we use two base libraries: *SPI*, a built in library for communicating with devices using the SPI devices; *NRF24L01*, a library developed by *Maniacbug* and made available to everyone as an open source project. This library is specifically designed to interact with the NRF24L01+ radio module [74].

[1] `http://www.digikey.pt/`

	CC2530	MRF24J40	CC2540	NRF24L01+
Manufacturer	Texas Instruments	Microchip Technology	Texas Instruments	Nordic Semiconductor
RF Standard	802.15.4	802.15.4	Bluetooth	General ISM/SRD > 1 GHz
Protocol	Zigbee	Zigbee	Bluetooth v4.0 Low Energy	Proprietary
Frequency	2.4 GHz	2.4 GHz	2.4 GHz	2.4 GHz
Data Rate (Max)	250 $kbps$	250 $kbps$	1 $Mbps$	2 $Mbps$
Power Output	4.5 dBm	0 dBm	4 dBm	0 dBm
Sensitivity	-97dBm	-94dBm	-93 dBm	-94 dBm
Serial Interface	SPI, UART	SPI	SPI, UART	SPI
Voltage Supply	2 V - 3.6 V	2.4 V - 3.6 V	2 V - 3.6 V	1.9 V - 3.6 V
Current Receiving	24 mA	19 mA	22.1 mA	13.5 mA
Current Transmitting	29 mA	23 mA	31.6 mA	11.3 mA
Price (at Digikey and referred to individual units)	€6.89	€8.78	€4.78	€2.97

Table 4.1: Comparison between different radio module transceivers.

With regard to the consumption of the microcontroller, we also import three built in libraries to help us manage the different power modes of the chip: *avr/sleep.h*, *avr/power.h* and *avr/wdt.h.*

Additional modules can be imported depending on the sensors we are using.

Define constants: Next we define a set of constants that will be used during the execution of the code:

- microcontroller pins assigned to the CE and CSN pins of the radio module;
- set the pipe addresses used by the NRF24L01+ transceiver;
- define a *struct* object with 22 bytes of size.

Define functions: Functions are a great tool to easily access and modify parts of the code without changing the general structure of the program. In our case, the following functions are present:

- function to read the V_{CC} voltage of the chip, hence the voltage level of batteries, comparing it with the $1.1V$ internal reference voltage of the microcontroller. It is needed since we are using a variable voltage input source (the batteries' voltage drop through the time) and allows us to do so without the need of an extra electronic circuit;
- function to read the variable we want to measure and return a valid result. There may be more than one of these functions.

Setup: Initialize and configure the radio module, defining the radio frequency channel, the auto acknowledgement mode, number of retries to send a message, data rate, payload size, power mode, reading and writing pipe addresses and initiating the listening mode; starting the serial communication between the microcontroller and the developer's personal computer and defining the baud rate of the communication; set the watchdog timeout prescaler value.

Main loop: This part of the program is constantly and cyclically running:

1. assigning values to the struct object (named *payload*):
   - payload.magic - a control letter, to identify the product this device is associated with. Typically we would send a 'Q', from Qold.
   - payload.major - a number, referring to the major version of the program in use;
   - payload.minor - a number, referring to the minor version of the program in use;
   - payload.reserved - a control number;
   - payload.qid - identification number of the device;
   - payload.qauth[0] - four least significant bytes of the authentication token;
   - payload.qauth[1] - four most significant bytes of the authentication token;
   - payload.temp - value given by the return of the function we are using to measure the desired physical variable. In this particular example, that function would return a number representing a temperature.
   - payload.batt - value of the percentage level of the batteries' capacity. Assigned by calling a function;
2. power up the radio;
3. stop radio listening mode;
4. transmit the struct payload;
5. start listening mode;

6. set loop counter to 0;
7. power down the radio;
8. set the microcontroller to the power down sleep mode;
9. watchdog wakes the system periodically, depending on the value of the timeout prescaler. Since the prescaler has a maximum timeout value of 8 seconds, if we want the microcontroller to be in power down mode for more than that, we need to make a cycle where this method is called multiple times, until we achieve the intended value. This is done by setting a value that is incremented every time the watchdog is called. Once this value is higher than a preset value, the system exits power down sleep mode;
10. exits power down sleep mode;
11. back to 1.

4.1.5 Batteries and Power Consumption

According to the microcontroller data sheet ([53]), when active and working at 8 MHz ($V_{CC} = 5V$), ATmega328P has a typical current consumption of 5.2 mA, with a maximum value of 9 mA. However, this chip has shutdown and sleep modes, which allow it to consume much less current when inactive.

Nevertheless, the board we chose was not constituted by the microcontroller alone, as it had some more elements consuming current. Hence, the power indicator LED and the voltage regulator are some of these extra elements [75]. Removing them could save us some current consumption:

- the power LED status is useless, since the board will be inside a closed box;
- the voltage regulator is not needed because we wanted to power it directly with a 3 V source.

The radio transceiver has a current consumption of 13.5 mA when transmitting and 11.3 mA when receiving. Also, when in stand by and in power down mode, NRF24L01+ has a current consumption of 26μA and 900 nA, respectively [72].

We built the circuit in figure 4.3 to measure the current consumed for both the micro-controller and the radio module when active and when inactive.

Batteries

The need of a wireless device is also extended to the power supply, so batteries as the power source are a must have. Having into consideration the power consumption of the sensor node, the voltage inputs of both microcontroller and radio transceiver and the physical size of the device, we concluded that two AA, nominal 1.5 V batteries in series would be enough to power up the system.

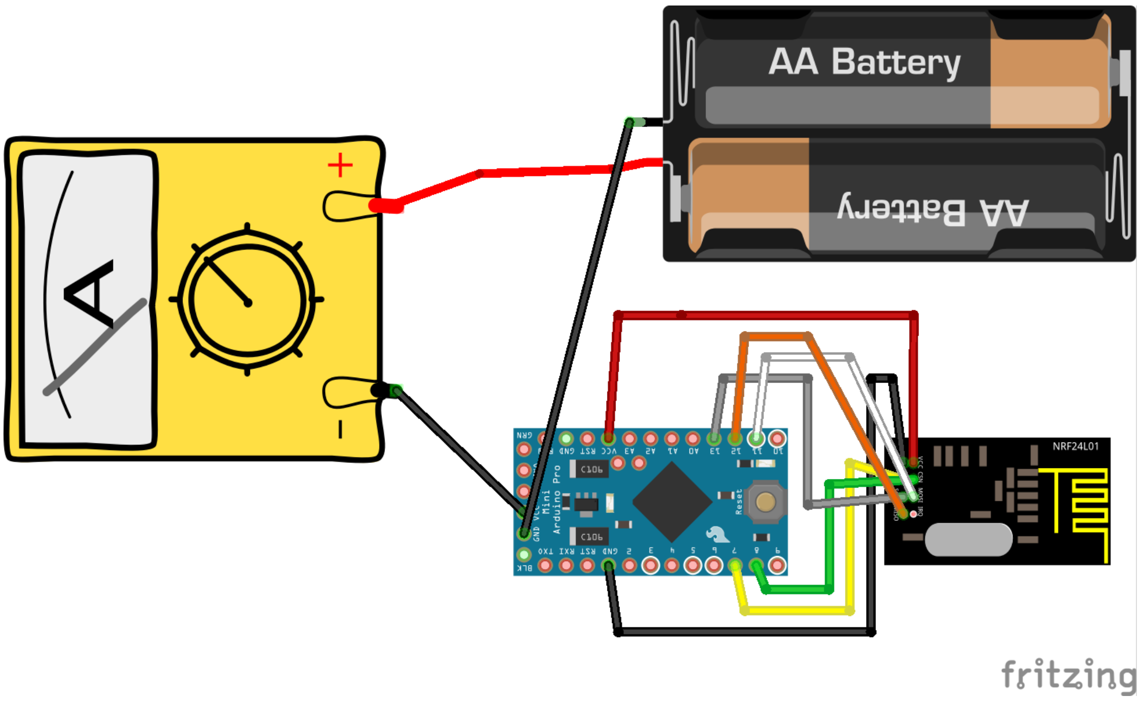


Figure 4.3: Circuit built to measure the current consumption of the Arduino Pro Mini and the NRF24L01+ radio transceiver. Before removing the components, we measured 23.1 mA when the microcontroller was in the active mode and the radio module transmitting/receiving. When in sleep mode and not transmitting/receiving, we measured 0.295 mA. After removing the components, the circuit had a current consumption of 18.3 mA when in active mode and 0.22 mA in sleep mode. Figure made using the Fritzing software.

Assuming that:

- batteries have a nominal capacity of 2100 mAh [76];
- system is active 1 second in every minute and then goes back to sleep.

The system has an average current consumption of

$$\frac{1 \times 18.3mA + 59 \times 0.22mA}{60} = 0.521mA \tag{4.1}$$

which together with a 2100 mAh battery gives the system a theoretical autonomy of

$$\frac{2100mAh}{0.521mA} = 4028.13h \text{ (hours).} \tag{4.2}$$

4.2 NRF24L01+

NRF24L01+ is a single chip 2.4 GHz transceiver designed for operation in the world wide ISM band frequency 2.400 - 2.525 GHz. It has an embedded baseband protocol engine (Enhanced Schockburst) and is suitable for ultra low power wireless applications [72]. The transceiver consists of an integrated frequency synthesizer, a power amplifier, a crystal oscillator, a demodulator and a modulator [77].

4.2.1 Features

An overview of the transceiver's features [72]:

- Radio
  - worldwide 2.4GHz ISM band operation;
  - 26 RF channels;
  - common RX and TX interface;
  - GFSK modulation;
  - 250$kbps$, 1 and 2$Mbps$ air data rate;
  - 1MHz non-overlapping channel spacing at 1$Mbps$;
  - 2MHz non-overlapping channel spacing at 2$Mbps$;
- Transmitter
  - programmable output power: 0, -6, -12 or -18dBm;
  - 11.3mA at 0dBm output power
- Receiver
  - fast AGC for improved dynamic range;
  - integrated channel filters;
  - 13.5mA at 2$Mbps$;
  - -82dBm sensitivity at 2$Mbps$;
  - -85dBm sensitivity at 1$Mbps$;
  - -94dBm sensitivity at 250$kbps$;
- RF Synthesizer
  - fully integrated synthesizer;
  - no external loop filer, VCO varactor diode or resonator;
  - accepts low cost ±60ppm 16MHz crystal;
- Enhanced ShockBurst^{TM}
  - 1 to 32 bytes dynamic payload length;
  - Automatic packet handling;
  - Auto packet transaction handling;
  - 6 data pipe MultiCeiver^{TM} for 1:6 star networks;
- Power Management

  - Integrated voltage regulator;
  - 1.9 to 3.6V supply range;
  - Idle modes with fast start-up times for advanced power management;
  - 26μA Standby-I mode, 900nA power down mode;
  - Maximum 1.5ms start-up from power down mode;
  - Maximum 130μs start-up from standby-I mode;

- Host Interface
  - 4-pin hardware SPI;
  - Maximum 10$Mbps$;
  - 3 separate 32 bytes TX and RX FIFOs;
  - 5V tolerant inputs;
- Compact 20-pin 4x4mm QFN package;

4.2.2 Radio Control

NRF24L01+ transceiver offers several options to configure the operation mode and the parameters used to control the radio.

Operational modes

The chip can be operated in four different modes: power down, standby, RX or TX mode [72].

Power down mode: In the power down mode, NRF24L01+ is disabled, reducing the current consumption to the minimum. All register values available are maintained and the SPI is kept active.

Standby mode: There are two standby modes available: standby-I mode and standby-II mode. In the former mode, only part of the crystal oscillator is active, contributing to a lower average consumption while maintaining short start up times; in the latter mode, extra clocks buffers are active and consumes more current than sandby-I mode. On the other hand, standby-II mode is quicker to wake up.

RX mode: This mode is active when the module is used as a receiver. The transceiver remains in RX mode unless it is told otherwise.

TX mode: TX mode is active when the radio transceiver is used to transmit a data packet. NRF24L01 remains in this mode until it finishes to transmit the data packet.

Air data rate

The data rate used to transmit and receive data packets can be set to $250kbps$, $1Mbps$ or $2Mbps$. Using a slower data rate increases the sensitivity of the receiver and has a theoretical better range. On the other hand, higher data rates gives lower average current consumption and reduced probability of on-air collisions.

RF channel frequency

In order to make two radio modules communicate with each other, both have to share the same RF channel frequency. This sets the center of the channel used by the NRF24L01+.

When transmitting at $250kbps$ or 1 $Mbps$, the channel occupies a bandwidth of $1MHz$, while a data rate of 2 $Mbps$ occupies 2 MHz.

The RF channel frequency (F_0) is set by a dedicated register (RF_{CH}) according to equation 4.3 [72]:

$$F_0 = 2400 + RF_{CH} \text{ [MHz]} \tag{4.3}$$

4.2.3 Enhanced ShockBurst^{TM}

Enhanced ShockBurst^{TM} is a packet based data link layer, featuring automatic packet assembly and timing, automatic acknowledgement and retransmissions of packets and automatic transaction handling [72]. It allows to easily implement a reliable bi-directional data link.

With automatic transaction handling, two transceivers, one acting as a primary receiver (PRX) and one acting as primary transmitter (PTX), can communicate with each other. The transaction is always initiated by the PTX, transmitting a data packet to the PRX and is terminated when the PTX has received an acknowledgement packet (ACK packet) from the PRX. PRX can attach a custom payload in the response packet, initiating this way a bi-directional data link.

Main features

- 1 to 32 bytes dynamic payload length;
- Automatic packet handling;
- Auto packet transaction handling:
  - Auto Acknowledgement with payload;
  - Auto retransmit;
- 6 data pipe MultiCeiver^{TM} for 1:6 star networks;

Packet format

The transmitted and received packet format contains preamble, address, packet control, payload and CRC field. Figure 4.4 shows the packet format.

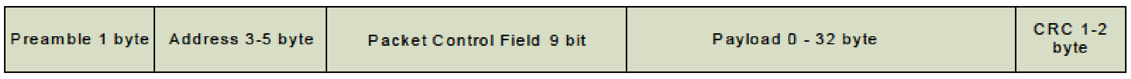


Figure 4.4: An Enhanced ShockBurst^{TM} packet with payload (0-32 bytes) [72].

Preamble: Is one byte long and is either 01010101 or 10101010. Its main function is to synchronize the receiver demodulator to the incoming bit stram.

Address: This is the address of the receiver. It can be configured to be 3, 4 or 5 bytes long.

Packet control field: It is constituted by a 6 bit payload length field, a 2 bit Packet Identify and a 1 bit NO_{ACK} field.

Payload: It is the user defined content of the packet. Its size can be set between 0 and 32 bytes and it is transmitted on air when is uploaded to NRF24L01+. In our case, we are using a 22 byte's size payload whose content is described on subsection 4.1.4.

Cyclic Redundancy Check (CRC): It is a mandatory error detection mechanism of the packet.

Automatic packet transition handling

Enhanced ShockBurst^{TM} features two functions for automatic packet transition handling: auto-acknowledgement and auto re-transmit.

Auto acknowledgement: This function automatically transmits an ACK packet to the PTX after it has received a valid packet.

Auto retransmission: If no ACK packet is received by the PTX after auto retransmission delay is elapsed, it keeps retransmitting the same packet n times, where n can be configured by the user.

MultiCeiver^{TM}

MultiCeiver^{TM} has a feature used in RX mode that allows six logical channels - data pipes - in the same physical RF channel. Each data pipe has its own address. This property allows the PRX to receive packets from six different PTXs transceivers without changing its frequency. Figure 4.5 shows an example of a PRX using MultiCeiver^{TM}.

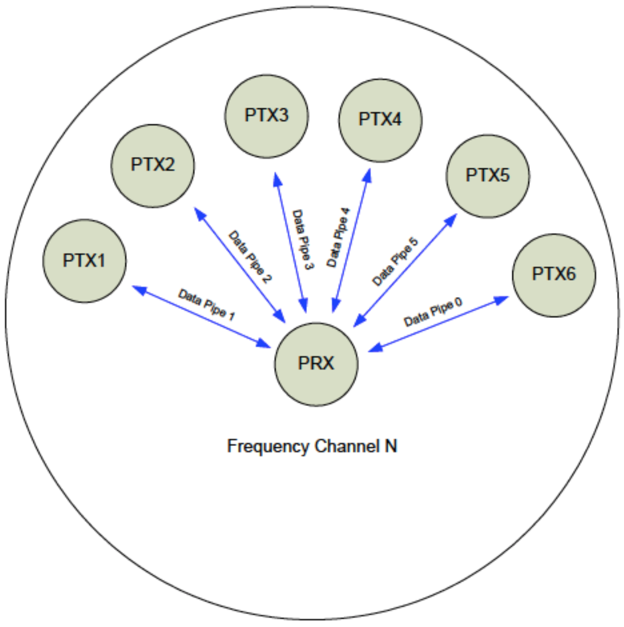


Figure 4.5: PRX using MultiCeiver^{TM} [72].

4.2.4 Data and control interface

All features of the NRF24L01+ chip can be accessed through the SPI interface.

Features

- 0-10 *Mbps* 4-wire SPI;
- 8 bit command set;
- easily configurable register map.

4.3 Gateway

Gateway makes the bridge between sensor networks and traditional communication networks. It not only acts as the primary receiver (PRX) in our architecture but also transmits the received data to the application layer.

It is constituted by two main elements:

- Raspberry Pi;
- NRF24L01+.

4.3.1 Raspberry Pi

A detailed description of the Raspberry Pi can be found on section 3.4.

In our case, as we are using the device as an IoT gateway, it should be able to fulfill the goals purposed in section 1.3. Hence:

Wide range of access capability [14]: The gateway should be compatible with a wide variety of wireless communication standards and radio transceivers. Since Raspberry Pi has a set of communication interfaces commonly found in any microcontroller, we can assume that if the radio module or standard can be implemented in a microcontroller, it can be implemented in the Raspberry Pi.

In our case, NRF24L01+ can be totally configured through its SPI pins, hence it is compatible with Raspberry Pi.

Manageability [14]: An IoT gateway should be able to be easily accessed, configured, upgraded and maintained. Since Raspberry Pi runs Linux and can be remotely accessed via a *SSH* connection, these requirements are easily fulfilled.

Protocol Interworking [14]: The IoT gateway should support both the WSN and the traditional networks. As the NRF24L01+ is the proprietary communication protocol used by the WSN, TCP/IP is the protocol used by Internet networks. Hence, Raspberry Pi should be able to make the data flow seamlessly from one network to another. Once again, it is easily done by the Raspberry Pi due to its connectivity interfaces: it is possible to connect the device to Internet using an ethernet cable, a wifi dongle or a 2G/3G/4G USB dongle.

4.3.2 NRF24L01+

The gateway and the sensor nodes need to use the same WSN protocol, so naturally we also find the NRF24L01+ modules connected to the Raspberry Pi. The radio modules communicate with the gateway device through its SPI communication interface.

4.3.3 Gateway Software

The software of the gateway aims to decode the data received from the radio module transceiver, process it and then transmit it to a web server, where the data will be stored and displayed to the users. Python was the programming language chosen to do that, while some bash scripts help configuring some features of the Raspberry Pi.

The main program can be divided in the following parts:

Importing libraries: There are lot of built in Python libraries available and accessible using a simple command in the terminal.

In our program, we need to use several libraries in order to be able to:

- control Raspberry Pi GPIO pins;
- control NRF24L01+ radio transceiver;
- configure SPI interface;
- unpack structured data received from the radio transceiver;

- make mathematical operations;
- interact with local databases;
- make HTTP requests to Aqora - a database as a service (DBaaS) developed by Whitesmith prior to this project.

The library to control the NRF24L01+ is a Python version from the original library developed by *Maniacbug*. This port was made by *JPBarraca*[2] and it is publicly available in his GitHub account under the terms of GNU General Public license [78].

All other libraries were already present in the operative system.

Define constants: Some constants needed to interact with the server are necessary: API keys, API secrets and Aqora url are defined importing them as environment variables.

Define functions: Functions to set the radio up, to unpack the data, to authenticate in the Aqora server and to make POST requests to it are defined.

Setup radio: Radio frequency channel, data pipe addresses, data rates and power output level is set. The radio transceiver is configured to be always in RX mode.

Main loop: The core of the program is responsible to do the following in a cyclic way:

1. parse value received from the radio;
2. construct a JSON object with the data parsed;
3. append the JSON object to a local database;
4. authenticate to the Aqora server;
5. try to make a POST request to Aqora, thus transmitting the JSON object as a payload to the server database:
   - if the response status code returns a '200', the data was successfully accepted by the server and the JSON object is flushed from the local database. Otherwise it is kept. Figure 4.6 shows an example of a JSON object.
6. back to 1.

4.4 Main features

Sensor nodes, radio transceivers and the gateway are the core elements of our architecture. Together they contribute to some of the system main features:

Adaptable: The system can be adapted to a wide variety of use cases, from temperature measurement to energy consumption. As long as the sensor we are using is compatible with the microcontroller, we can measure, store and display any physical variable.

[2]`https://github.com/jpbarraca/pynrf24`

```
{
  "data": [
    {
      "qid": "QOLD_QID",
      "qauth": "QOLD_QAUTH",
      "data": [
        {
          "t": "2015-01-19T20:15:32.846Z",
          "temp": 3.4
        },
        {
          "t": "2015-01-19T20:30:32.846Z",
          "temp": 3.7,
          "batt": 89
        }
      ]
    },
    {
      "qid": "QOLD_QID2",
      "qauth": "QOLD_QAUTH2",
      "data": [
        {
          "t": "2015-01-19T19:15:32.846Z",
          "temp": 3.2
        }
      ]
    }
  ]
}
```

Figure 4.6: Example of a JSON object posted to Aqora server.

Modular: The system is constituted for one gateway and up to six sensor nodes in a star network topology. As long as the nodes are within the communication range of the gateway, they can be placed anywhere.

Easy deployment: After having the gateway ready and configured and the sensor nodes built and with the right credentials, the process of installing a WSN is as easy as plug the Rasperry Pi to a power socket and place the sensor nodes in the desired place.

Easy to fix: In case of some hardware failure of a sensor node, it is easy to replace it with other. All we have to do is remove the faulty node and replace it with another flashed with the same program. There is no need to interact with the gateway.

Chapter 5

Platform Validation

As stated before, this system was developed to create Whitesmith's own modular hardware platform to be used in Qold[1] and, eventually, Unplugg[2]. Hence, to validate the developed platform and architecture choices, it was created two prototypes: one to measure temperature and the other to measure power consumption. The former was used to prototype sensor nodes to be used by Qold, being tested in different real application scenarios, with the most relevant results displayed and explained in section 5.1. The latter was only tested in a controlled environment, with the results explained and displayed in section 5.2.

5.1 Qold

Qold was already explained on subsection 1.2.1 as well as its requirements: a working WSN of sensor nodes and a gateway.

5.1.1 Sensor Nodes

In this product, sensor nodes are designed mainly to measure temperatures inside refrigerators and freezers. Thus, beyond the generic hardware present in the sensor node (section 4.1) we also have a temperature sensor present in the node.

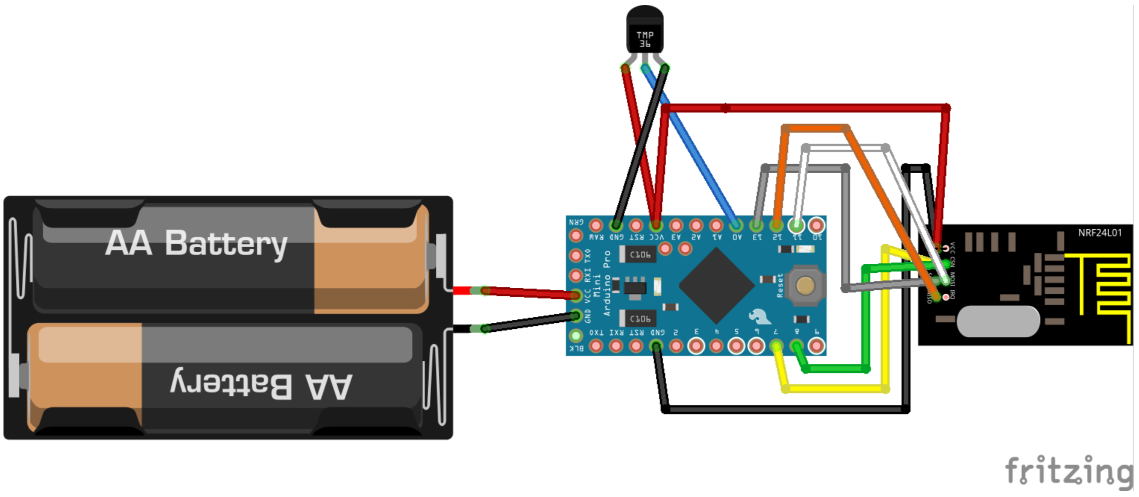
Temperature Sensor

We used and tested three different temperature sensors during the development of the product: TMP36 [35], DHT22 [79] and DS18B20 [80].

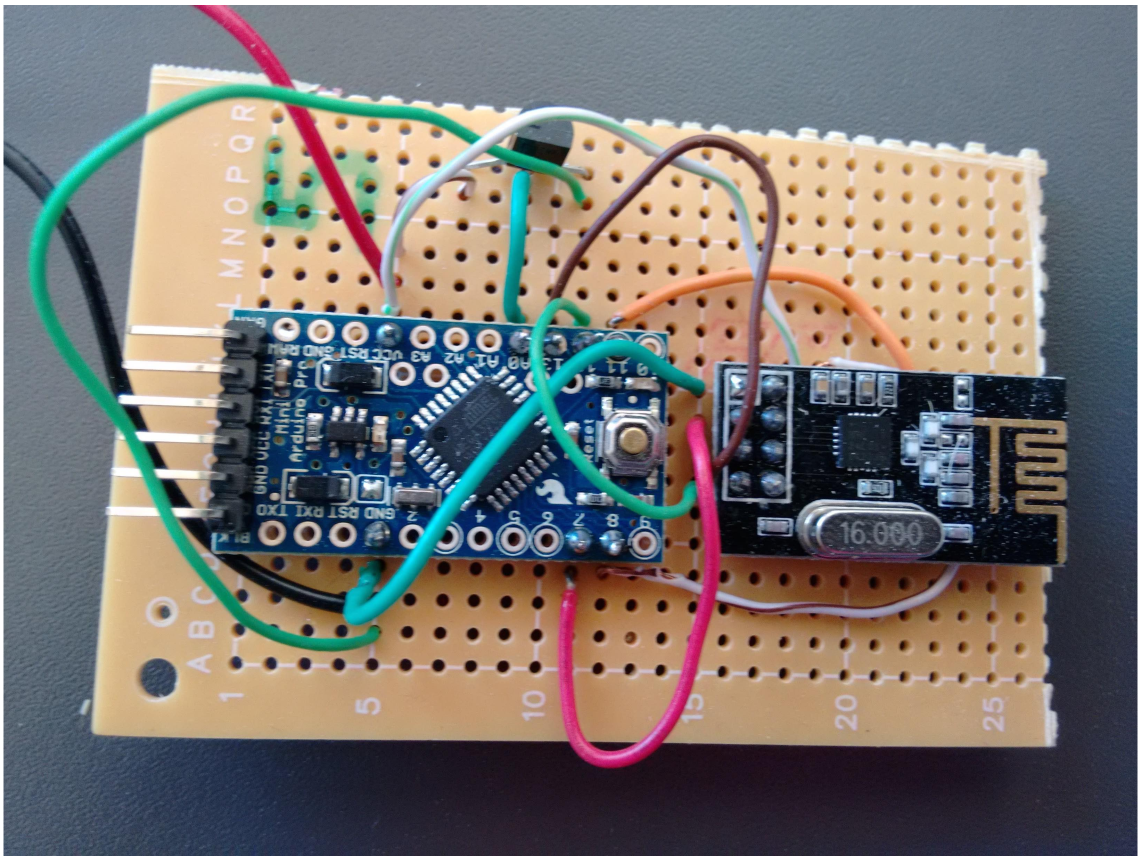
TMP36: This is a low cost analogical sensor, whose output voltage is proportional to the temperature. Figure 5.1 shows a schematic view of the sensor node as well as one of the prototypes built on a veroboard. As we can see, the device is constituted by the microcontroller on a Arduino Pro Mini Board, the NRF24L01+ radio transceiver,

[1]`http://www.qold.co/`
[2]`http://unplu.gg/`

the batteries and the temperature sensor. This was the sensor we used in most of our prototypes.



(a) Schematic version of the developed node.



(b) Sensor node built on a veroboard.

Figure 5.1: Node sensor with TMP36 temperature sensor. Figure made using the Fritzing software.

DHT22: This is a digital temperature and humidity sensor, with a single wire digital interface. This sensor is a bit more expensive when compared with TMP36, but adds the ability to measure humidity and the digital interface, which makes the integration with the microcontroller easier. Figure 5.2 shows the schematic version of this sensor node. As we can see, in terms of hardware, the only difference is the sensor used and the pins, with everything else remaining the same.

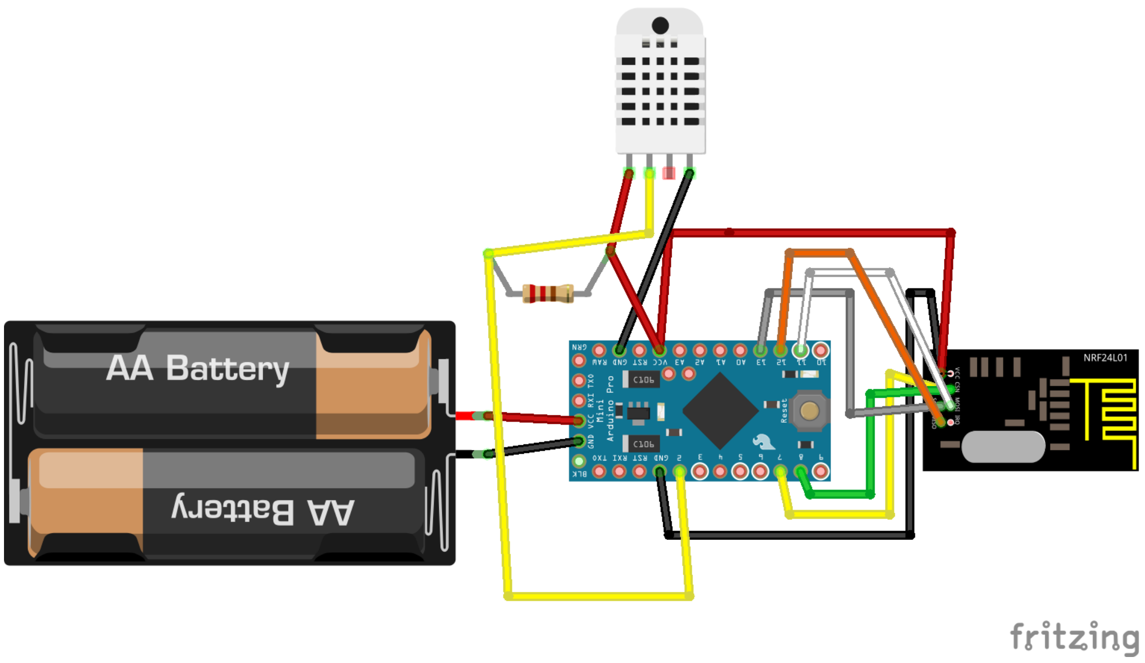


Figure 5.2: Sensor node with DHT22 temperature sensor. Figure made using the Fritzing software.

DS18B20: Also a digital temperature sensor, but this time it adds water proof characteristics, making it ideal to humid environments. Moreover, as it has a long wire, the sensor can be put inside the refrigerators and freezers while the rest of the node can be outside, improving the range of the radio transceiver. Figure 5.3 shows the schematic version of this sensor node.

5.1.2 Gateway

Qold's gateway is constituted by a Raspberry Pi and a NRF24L01+ radio module. In our case we used mostly the Raspberry Pi model A, due to its reduced cost when compared to the models. Also, its features are enough, as one USB port allows to connect either a wifi dongle or a 2G/3G/4G USB dongle. Figure 5.4 shows the schematic version of the gateway and one gateway that we are actually using, with a wifi dongle connected.

5.1.3 Real Scenario Application

Qold offers real-time temperature monitoring and is aimed mostly to small and medium-sized business, where this control is made manually, as most automated solutions are

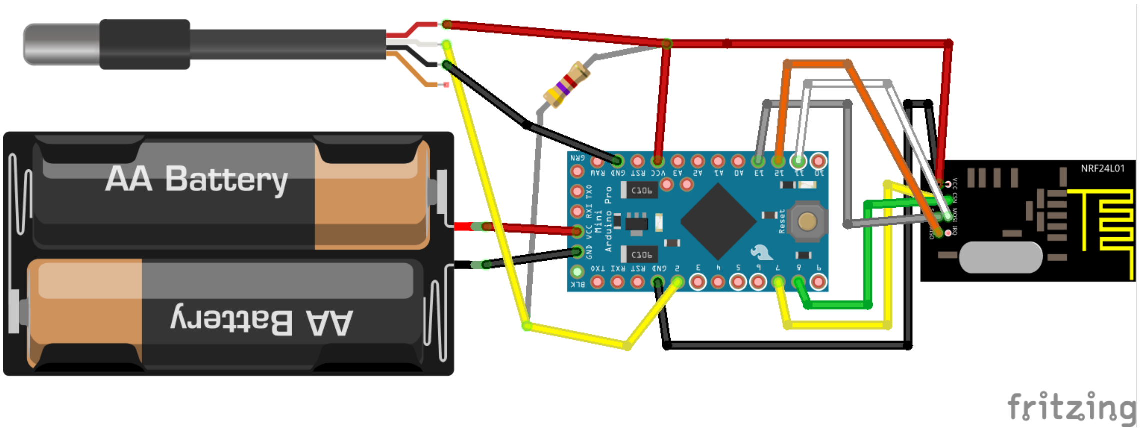


Figure 5.3: Sensor node with DS18B20 temperature sensor. Figure made using the Fritzing software.

expensive and require a significant upfront investment [81].

While developing the platform, we installed pilot devices in some potential future users of the product. The most relevant test was done in a commercial establishment in Coimbra.

Pilot

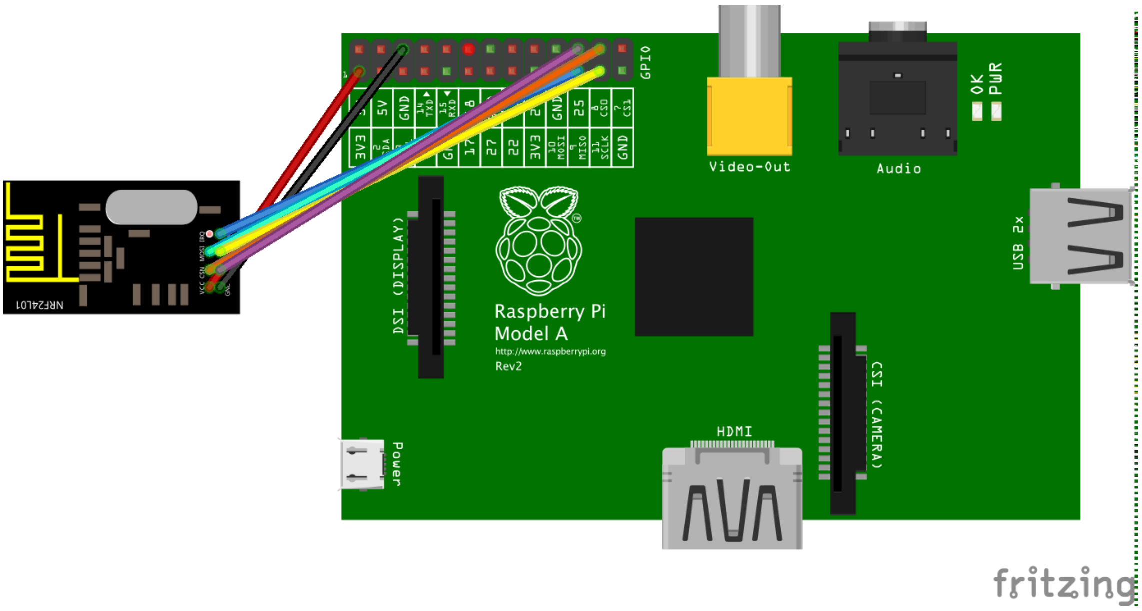
In this scenario we used two sensor nodes connected to a gateway. One was put in a refrigerator and the other inside a vitrine. Both should measure the temperature of refrigerator and vitrine every minute and send the values to the gateway and then to Aqora. The gateway was connected to the internet using a wifi USB dongle.

Device in the refrigerator has been measuring and transmitting data from March 17th to July 31st, as we can see in figure 5.5a.

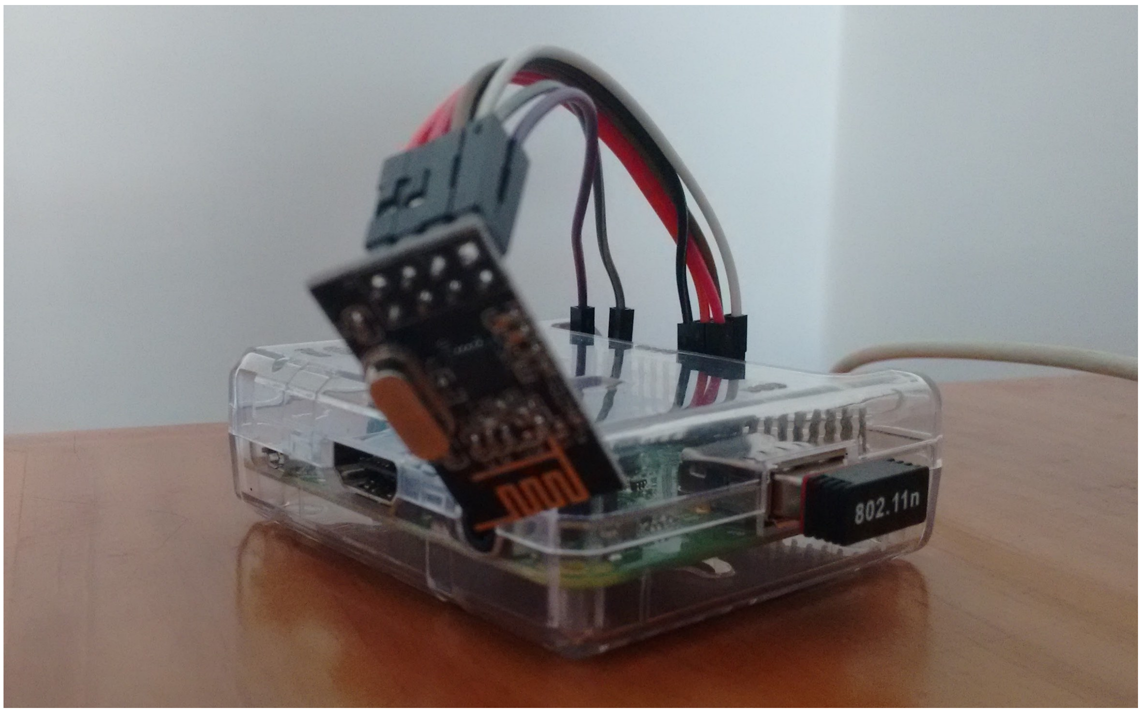
A total of 159436 values were registered out of the theoretically expected 197280, resulting in an average of one measurement every one minute and fourteen seconds and an up time of 80.7%. This number can be explained by some periods of time where the gateway didn't communicate with the Aqora at all, due mostly to problems with the wifi connection which were beyond our control. Although it was not possible to measure the intended mark of one value every minute, the obtained result is good enough to, for example, observe the pattern of the thermostat of the refrigerator, as it can be seen in figure 5.5b, meaning that it is also good enough for its typical scenario of application.

Device in the vitrine has been measuring and transmitting data from March 17th to July 31st as well, as we can see in figure 5.6a.

A total of 147800 values were registered out of the theoretically expected 197280, resulting in an average of one measurement every one minute and twenty seconds and an up time of 74.8%. This number can also be explained by some periods of time where the gateway didn't communicate with the Aqora at all, due mostly to problems with



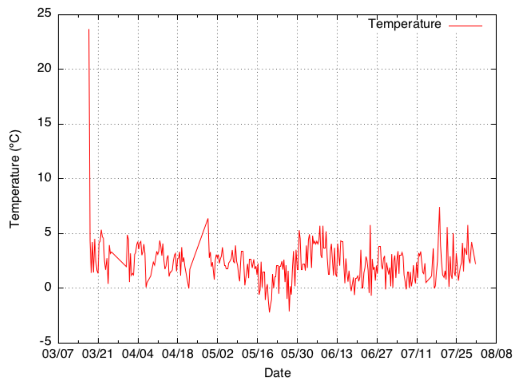
(a) Schematic version of the gateway. It is constituted by a Raspberry Pi and a NRF24L01+ radio transceiver.



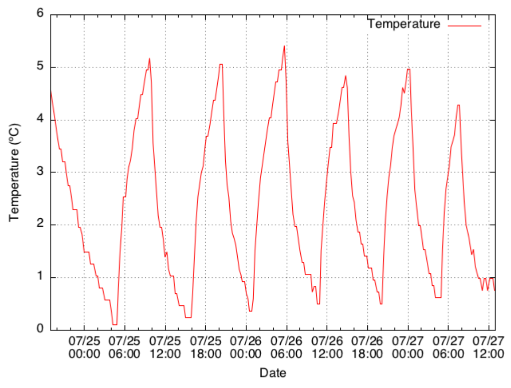
(b) One of the gateways used.

Figure 5.4: Gateway with a wifi USB dongle.

the wifi connection. However, the number of values received is lower when compared with the device inside refrigerator, which means that in the same network conditions, the gateway received more packets from one device than from another, which should not have happened.



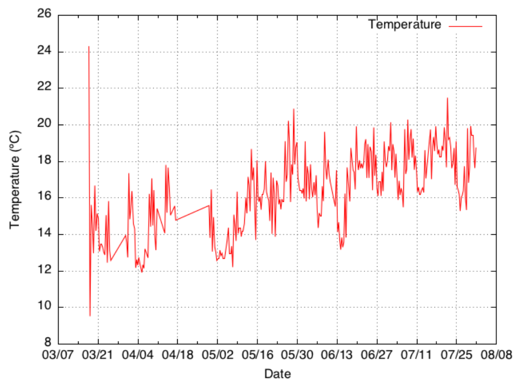
(a) Measured temperatures of the refrigerator from March 17th to July 31st. Data shown was down sampled to one point in every five hundred.



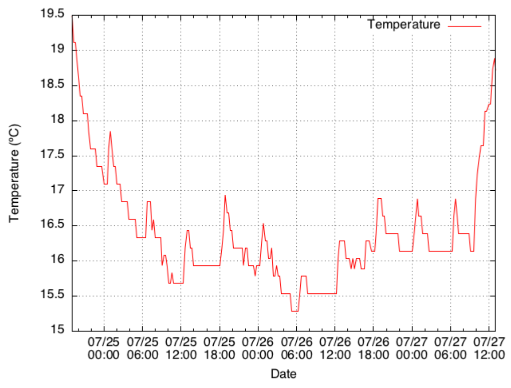
(b) Measured temperatures of the refrigerator from July 24th 19:00:00 to July 27th 13:00:00. Data shown was down sampled to one point in every fifteen.

Figure 5.5: Measured temperatures of the refrigerator.

Beyond that, in this case is also more difficult to understand patterns in the variation of the temperature inside the vitrine, as can be seen in figure 5.6b.



(a) Measured temperatures of the vitrine from March 17th to July 31st. Data shown was down sampled to one point in every five hundred.

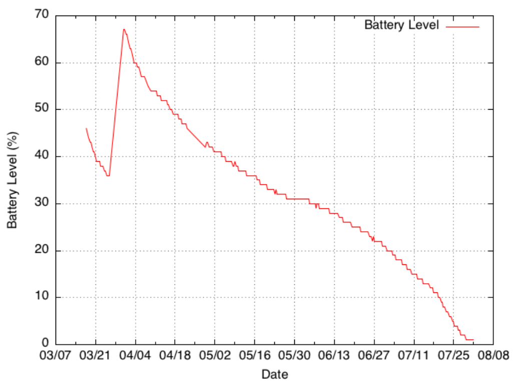


(b) Measured temperatures of the vitrine from July 24th 19:00:00 to July 27th 13:00:00. Data shown was down sampled to one point in every fifteen.

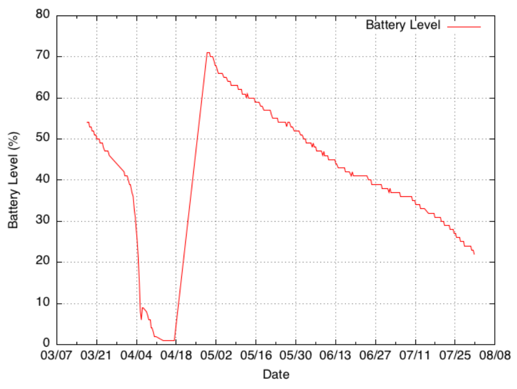
Figure 5.6: Measured temperatures of the vitrine.

The percentage level of the batteries used in the devices was also registered for both situations, as we can see in figure 5.7. It was done mapping the values of the voltage to percentage values. Thus, 100% was generically mapped to $3.3V$ $(2 \times 1.65V)$ and 0% to $2.1V$ $(2 \times 1.05V)$.

The information is useful to inform the potential client when is time to change the batteries. As we can see by the rise of battery level, we had changed them in both devices. In the case of the device placed in the vitrine, the batteries were almost empty at a certain moment (see figure 5.7b), which can be the reason why this device sent less values when compared with the other one.



(a) Battery level of the device placed inside refrigerator (in %).



(b) Battery level of the device placed inside vitrine (in %).

Figure 5.7: Battery level of both devices (in %).

Taking into consideration only the values of battery level registered after batteries have been changed, it is possible to estimate how long they would last, if the voltage drop was constant as seen in figure 5.8. The estimation was done making a simple linear regression fit to our data using the free software Gnuplot[3].

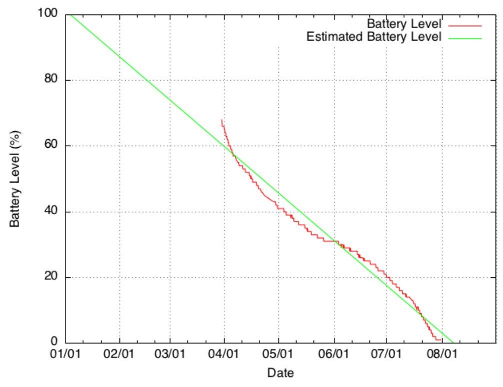
According to the graph, It is expected that both batteries last between seven and eight months. This value is a littler higher than the one theoretically estimated in subsection 4.1.5. This can be explained by the fact that the voltage drop of a battery is non linear, that is, it typically drops faster in the end of battery's life cycle [76]. Also, the reference voltage values to both 100 % and to 0 % have to be reconsidered if we want a more realistic information.The former one is too optimistic, as it represents the open circuit voltage of two AA batteries in series. When under a load, this value is lower. The latter has to be adjusted either, since the device in the vitrine was still able to transmit a series of packets indicating that it was with 0% of battery, which should not have happened.

Other tests

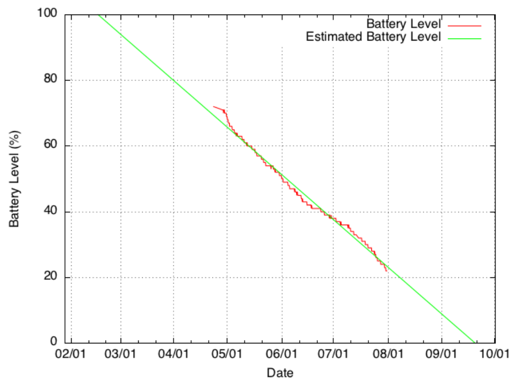
The success of the first installed pilot contributed to the acceptation of the platform as a valid solution to the use case of Qold. Hence, more pilots were installed in different locations:

- Two sensor nodes and a gateway connected to the internet through a 3G USB dongle, in an establishment located in the Municipal Market of Coimbra;
- Two sensor nodes and a gateway connected to the Internet through a 3G USB dongle, in a multinational company, Porto;

[3]`http://www.gnuplot.info/`



(a) Estimated battery life of the device placed inside refrigerator.



(b) Estimated battery life of the device placed inside vitrine.

Figure 5.8: Estimated battery life of both devices.

- Eight sensors and two gateways connected to the internet using wifi, in a commercial establishment, Aveiro.

Overall the results were good, as it allowed us to test different scenarios successfully:

- gateway communicating to the internet using two different methods: wifi and 3G;
- once in the place, we were able to quickly install the devices and the user had immediate access to the platform and to the transmitted data;
- in the situations where we had to change nodes, for example, to replace it with other one using a different temperature sensor, the process was quick and seamless, as all we had to do was replace one node with another, having the same flashed firmware.

It also allows us to understand some of the limitations of the platform:

- we still have not developed an easy process to connect the gateway to new wifi networks. We have to configure the credentials of the wifi beforehand;
- in situations where data is expected to be transmitted to Aqora but for some reason it is not being received, we still have some difficulties to understand if the problem lies in the sensor nodes or in the gateway.

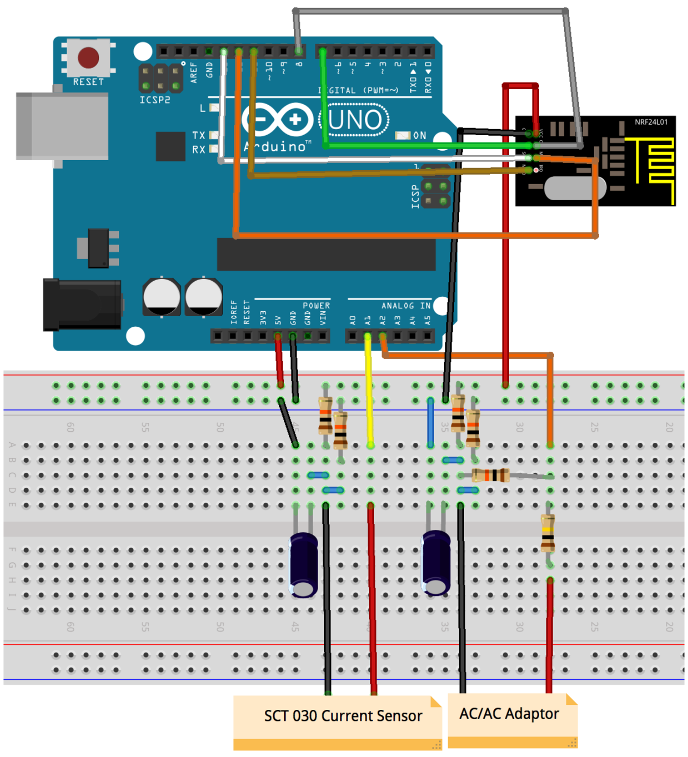
5.2 Energy Monitor

A device to measure the power consumption of house appliances was also developed. This prototype was only tested in a controlled environment, although it may be useful in the future to integrate in products such as Unplugg. As Qold, this device was constituted by a sensor node and a gateway. Although the gateway is the same used in Qold, the sensor node is different.

5.2.1 Sensor Node

This sensor node was designed to measure and transmit the values of house appliance power consumption and the grid voltage. The sensor node's circuit built was based on a guide from the Open Energy Monitor website [82], with our adaptations in order to add the NRF24L01+ radio module.

As it was not only tested internally but also designed to be near a socket power supply, we used a Arduino Uno [83] and a breadboard. The designed circuit can be seen in figure 5.9.



(a) Schematic version of the energy monitor. Figure made using the Fritzing software.

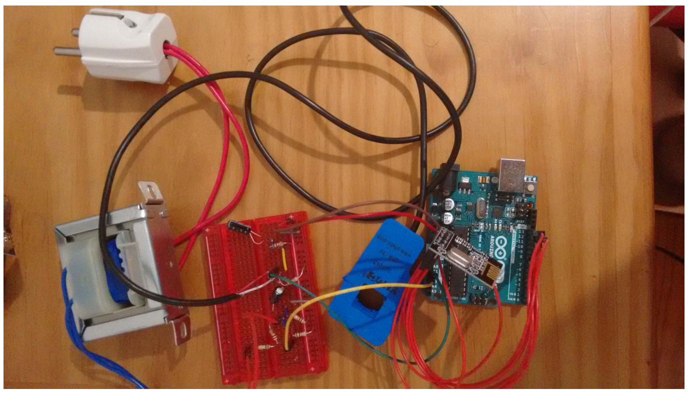

(b) Energy monitor built on a breadboard.

Figure 5.9: Energy monitor sensor node.

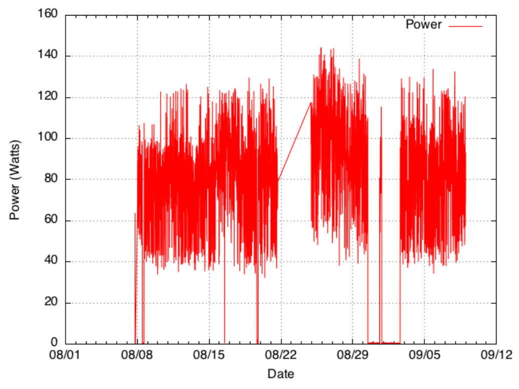
The current sensor used was a non invasive AC current sensor - SCT-013-30 [84]. This sensor can be clamped around the supply line of an electrical load. The voltage output of the sensor will be proportional to the current passing in the supply line.

To measure the grid voltage, we used an AC-AC 230V-9V transformer. It was necessary due to Arduino's limitations, as it only supports a maximum of 5V input. A voltage divider circuit and an offset was also added in order to only allow positive voltage to be read by the Arduino.

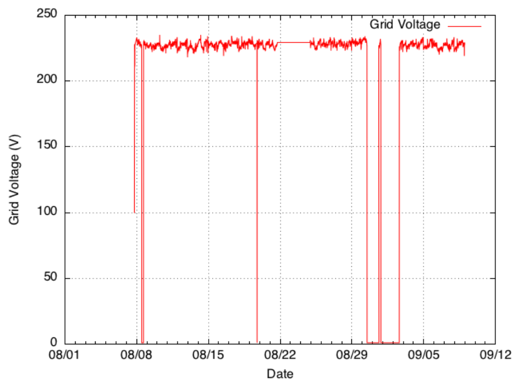
5.2.2 Results

We measured the power consumption of a refrigerator between August 8^{th} and September 8^th. It was also measured the grid voltage, essential to calculate the power consumption. The measurement was done with a rate of one point every minute. Figure 5.10 gives us an overview of the collected data.

A total of 42977 points were transmitted out of the theoretically possible 44958, meaning that we measured 95.6% of the possible values. Hence, we were able to transmit one value every one minute and three seconds, in average.



(a) Power consumption of a refrigerator.



(b) Grid voltage.

Figure 5.10: Power consumption of a refrigerator and grid voltage.

It is possible to observe periods of time where the power consumption was near zero. That happened because the AC adapter was unintentionally removed from the power socket, meaning that sometimes a 0V reading was made. Since the power consumption reading depends on both the current reading and the voltage reading, the zero voltage reading values lead to zero power consumption reading values. If we look carefully at both figure 5.10a and 5.10b, we can see that the values of zero watts power consumption and 0 volts grid voltage are measured at the same periods. Figure 5.11 allows us to see that.

Selecting a short period of time it is possible to observe the change of the power consumption of the refrigerator due to its thermostat being turned on and off, as is demonstrated in figure 5.12.

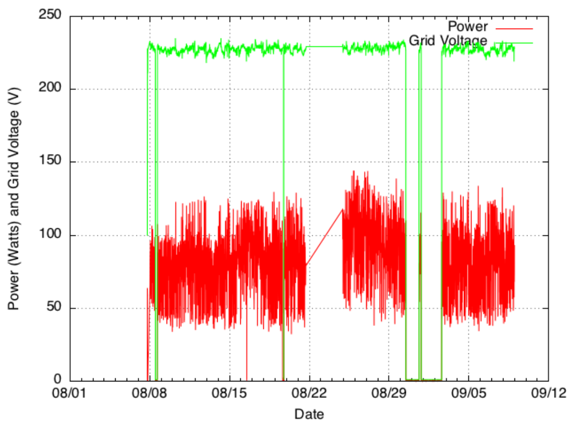


Figure 5.11: Superposition of the power consumption and grid voltage.

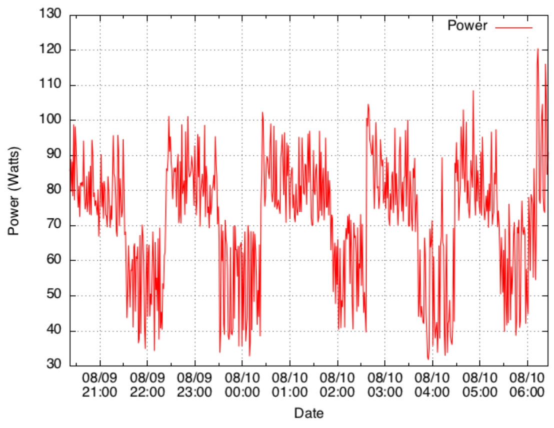


Figure 5.12: Variation of the power consumed by the refrigerator.

5.2.3 Overview

This prototype allows us to successfully test the platform in a different application scenario, measuring different physical variables. We were able to achieve 95% of up time, which was the best value so far.

Chapter 6

Conclusion

The work developed in the context of this thesis can be analyzed having into consideration the scope of this thesis, the initial project goals and the developed prototypes.

6.1 Scope Analysis

At section 1.4 we defined what was expected to be accomplished in the end of the project. Having that in mind, we can do a checklist:

- Design and build a modular, battery powered sensor node compatible with a wide variety of sensors and with the most popular radio modules ✓
- Program the sensor nodes to measure the desired physical quantity and to transmit the data to a gateway ✓
- Choose a gateway compatible with the most popular radio modules and communication protocols ✓
- Configure the gateway to accept data from the sensor node and to transmit it to a Web server ✓

6.2 Goal Analysis

As previously said, once this project was concluded we should have a modular platform capable of connecting a generic sensor to the internet, thus becoming the base hardware platform for future Whitesmith's products, such as Qold.

6.2.1 Sensor nodes

The requirements of this product regarding the sensor nodes were:

Low cost and small form factor: The cost of the device depends mostly on the cost of its components, at least while prototyping, hence the price of the node will always depend on the cost of the sensors used. However, the other components - the microcontroller and the radio module - are really affordable. For example, Qold sensor nodes costed around €6 each.

Regarding the dimensions of the product, the only requirement is for the the packaging containing the electronics to also be able to accommodate two AA batteries.

Scalable architecture and efficient protocols: It is easy to add sensor nodes to a established WSN, easily scaling the size of the network. However, there is a limitation regarding the number of devices that can communicate with each receiver. In situations where more than six sensor nodes are necessary, it means that either a different topology must be designed (mesh network, for example) or more gateways have to be used. Moreover, it is also easy to add different sensors to the node, increasing the adaptability of the product.

Resource-efficient design: As we have seen, the developed nodes have both a low price and a low power consumption - with an average power consumption of 0.521 mA it is possible to transmit one packet every minute.

Self-configuration and self-organizing sensors: When designing a WSN, sensor nodes should be able to be added or replaced without affecting the general objective of the application and without having to reconfigure the gateway. We were able to achieve that.

Localized processing: The sensor nodes only transmit the expected values, that is, all the raw collected data is processed locally. Qold sensor nodes, for example, transmit always a value of temperature in Celcius degrees, independently of the temperature sensor it is using.

Application specific design: Each sensor node is designed having into consideration its application.

Using the same base, we were able to use a node to transmit temperature values and other one to transmit power consumption values.

Secure design: This is an area where we still need to improve the product. There is not any kind of encryption or security mechanism in the packets transmitted between the sensor nodes and gateway.

6.2.2 Gateway

The defined goals of the gateway were:

Wide range of access capability: Raspberry Pi is compatible with most of the commercial off the shelf radio module transceivers available.

Manageability: Running Linux, a high level operation system, our gateway can be easily accessed, configured, upgraded and maintained.

Protocol interworking: Raspberry Pi supports both local WSNs and traditional networks.

Low cost and small form factor: A working gateway, such as the one used to test Qold, can be obtained for a price around €40.

6.3 Developed Prototypes

We tested the developed platform in two main scenarios: one where it was placed in a commercial establishment, being confronted with a real situation of application scenario and the other one where it was tested in a controlled environment.

In the first scenario, we were able to:

- measure temperatures every minute almost uninterruptedly for 5 months;
- transmit the temperature read values to the gateway and then to an online database;
- have two sensor nodes connected to the same gateway;
- validate the power consumption of the product and have a rough estimate the life time of the batteries;
- prototype a pilot for a commercial product - Qold;

In the second scenario, we were able to:

- measure the power consumption of a refrigerator, registering the grid voltage in the process;
- have one value transmitted every one minute and three seconds, in average;
- demonstrate the modularity of the platform;

Moreover, beyond these tests, we also installed our platform in three more places as a pilot product for Qold, having tested a total of 5 gateways and 14 sensor nodes. We had 6 different sensors connected to the same gateway with good results.

6.4 Future Work

The developed platform is far from being finished, having still room for improvement. The most important features to add:

- method to easily connect our gateway to new wifi networks;

- secure remote access to the gateway through the Internet, for diagnosis processes;
- flashing the sensor nodes over the air;
- security layer to the WSN used;
- sensor nodes with capacity of self-diagnosis;
- more realistic information about the remaining battery time.

Bibliography

[1] Verena Weber. Smart sensor networks: Technologies and applications for green growth. *The Organization for Economic Cooperation and Development*, 2009.

[2] Khuong M Vu. Ict as a source of economic growth in the information age: Empirical evidence from the 1996–2005 period. *Telecommunications Policy*, 35(4):357–372, 2011.

[3] Melisande Cardona, Tobias Kretschmer, and Thomas Strobel. Ict and productivity: conclusions from the empirical literature. *Information Economics and Policy*, 25(3):109–125, 2013.

[4] Marc-Eric Bobillier Chaumon, Christine Michel, Franck Tarpin Bernard, and Bernard Croisile. Can ict improve the quality of life of elderly adults living in residential home care units? from actual impacts to hidden artefacts. *Behaviour & Information Technology*, 33(6):574–590, 2014.

[5] Rolf Adam and Walter Wintersteller. What's so smart about the smart grid? `http://www.strategy-business.com/article/li00091?gko=aa0aa`, 2008. [Online; accessed 10-July-2015].

[6] David J Brooks. Intelligent buildings: an investigation into current and emerging security vulnerabilities in automated building systems using an applied defeat methodology. 2011.

[7] Robert D Atkinson and Daniel Castro. Digital quality of life: Understanding the personal and social benefits of the information technology revolution. *Available at SSRN 1278185*, 2008.

[8] Jon S Wilson. *Sensor technology handbook*. Elsevier, 2004.

[9] Roberto Verdone, Davide Dardari, Gianluca Mazzini, and Andrea Conti. *Wireless sensor and actuator networks: technologies, analysis and design*. Academic Press, 2010.

[10] Jacob Kastrenakes. The dumb state of the smart home. `http://www.theverge.com/2014/1/24/5336104/smart-home-standard-are-a-mess-zigbee-z-wave`, January 2014. [Online; accessed 15-July-2015].

[11] Alex Wood. The internet of things is revolutionising our lives, but standards are a must. `http://www.theguardian.com/media-network/2015/mar/31/the-internet-of-things-is-revolutionising-our-lives-but-standards-are-a-must`, March 2015. [Online; accessed 15-July-2015].

[12] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67. ACM, 2000.

[13] Vehbi C Gungor and Gerhard P Hancke. Industrial wireless sensor networks: Challenges, design principles, and technical approaches. *Industrial Electronics, IEEE Transactions on*, 56(10):4258–4265, 2009.

[14] Qian Zhu, Ruicong Wang, Qi Chen, Yan Liu, and Weijun Qin. Iot gateway: Bridging-wireless sensor networks into internet of things. In *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, pages 347–352. IEEE, 2010.

[15] Christian Märtin. Multicore processors: Challenges, opportunities, emerging trends. In *Embedded World Conference Proceedings*, 2014.

[16] Cuno Pfister. *Getting Started with the Internet of Things: Connecting Sensors and Microcontrollers to the Cloud.* " O'Reilly Media, Inc.", 2011.

[17] Alan Trevennor. A brief history of microcontrollers. In *Practical AVR Microcontrollers*, pages 3–11. Springer, 2012.

[18] Tim Wilmshurst. *Designing embedded systems with PIC microcontrollers: principles and applications*. Newnes, 2006.

[19] Gunther Gridling and Bettina Weiss. Introduction to microcontrollers. *Vienna University of Technology Institute of Computer Engineering Embedded Computing Systems Group*, 2007.

[20] Miles Murdocca and Vincent P Heuring. *Principles of computer architecture*. Prentice Hall, 2000.

[21] IBM. Understanding static ram operation. In *Applications Note*, 1997.

[22] IBM. Understanding dram operation. In *Applications Note*, 1996.

[23] Lu Shih-Lien. Memory devices. In *The Electronics Handbook*, pages 756–768. Springer, 2012.

[24] Curtis C Johnson, Stanley D Moss, and Jiri A Janata. Field effect transistor, May 3 1977. US Patent 4,020,830.

[25] Jagan Singh Meena, Simon Min Sze, Umesh Chand, and Tseung-Yuen Tseng. Overview of emerging nonvolatile memory technologies. *Nanoscale research letters*, 9(1):1–33, 2014.

[26] Navid Zeraatkar. Flash memory technology.

[27] The avr microcontroller digital i/o ports. `http://www.avr-tutorials.com/digital/about-avr-8-bit-microcontrollers-digital-io-ports`, 2010-2012. [Online; accessed 25-July-2015].

[28] Port registers. `https://www.arduino.cc/en/Reference/PortManipulation`. [Online; accessed 25-July-2015].

[29] Stuart R Ball. *Analog interfacing to embedded microprocessor systems.* Elsevier, 2004.

[30] Edward McConnell and David Jernigan. Data acquisition. In *The Electronics Handbook*, pages 1938–1967. Springer, 2012.

[31] Sergio Saponara, Luca Fanucci, and Pierangelo Terreni. Architectural-level power optimization of microcontroller cores in embedded systems. *Industrial Electronics, IEEE Transactions on*, 54(1):680–683, 2007.

[32] Charles Bell. *Beginning sensor networks with Arduino and Raspberry Pi.* Apress, 2013.

[33] Stefan Poslad. *Ubiquitous computing: smart devices, environments and interactions.* John Wiley & Sons, 2011.

[34] Eric A Von Hippel. Democratizing innovation. 2005.

[35] Low Voltage Temperature Sensors TMP35. Tmp36/tmp37 datasheet,© 1996–2010 analog devices. *Inc. All rights reserved.*

[36] Inc. BiPOM Electronics. Microcontroller to sensor interfacing techniques. `http://www.bipom.com/documents/lectures/Microcontroller%20to%20Sensor%20Interfacing%20Techniques.pdf`. [Online; accessed 29-July-2015].

[37] Ferran Reverter. The art of directly interfacing sensors to microcontrollers. *Journal of Low Power Electronics and Applications*, 2(4):265–281, 2012.

[38] Near East University. Transmission of digital data. `http://staff.neu.edu.tr/~shanableh/file/COM%20318-CH5-7.pdf`. [Online; accessed 10-August-2015].

[39] Frédéric Leens. An introduction to i 2 c and spi protocols. *Instrumentation & Measurement Magazine, IEEE*, 12(1):8–13, 2009.

[40] Priyanka Rawat, Kamal Deep Singh, Hakima Chaouchi, and Jean Marie Bonnin. Wireless sensor networks: a survey on recent developments and potential synergies. *The Journal of Supercomputing*, 68(1):1–48, 2014.

[41] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.

[42] Hubert Zimmermann. Osi reference model–the iso model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 28(4):425–432, 1980.

[43] Carles Gomez and Josep Paradells. Wireless home automation networks: A survey of architectures and technologies. *IEEE Communications Magazine*, 48(6):92–101, 2010.

[44] Gabriel Montenegro, Nandakishore Kushalnagar, Jonathan Hui, and David Culler. Transmission of ipv6 packets over ieee 802.15. 4 networks. Technical report, 2007.

[45] Kevin Ashton. That 'internet of things' thing. *RFiD Journal*, 22(7):97–114, 2009.

[46] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[47] Lieven Trappeniers, Mohamed Ali Feki, Fahim Kawsar, and Mathieu Boussard. The internet of things: the next technological revolution. *Computer*, 46(2):0024–25, 2013.

[48] Steve Hodges, Stephen Taylor, Nicolas Villar, James Scott, Dominik Bial, and Patrick Tobias Fischer. Prototyping connected devices for the internet of things. *Computer*, 46(2):26–34, 2013.

[49] Renee DiResta, Brady Forrest, and Ryan Vinyard. *The Hardware Startup.* " O'Reilly Media, Inc.", 2015.

[50] Dries De Roeck, Karin Slegers, Johan Criel, Marc Godon, Laurence Claeys, Katriina Kilpi, and An Jacobs. I would diyse for it!: a manifesto for do-it-yourself internet-of-things creation. In *Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design*, pages 170–179. ACM, 2012.

[51] David Mellis et al. Do-it-yourself fabrication of electronic devices. *Pervasive Computing, IEEE*, 13(3):22–29, 2014.

[52] Lilli Manolis Sherman. 3d printers lead growth of rapid prototyping. `http://www.ptonline.com/articles/3d-printers-lead-growth-of-rapid-prototyping`, August 2014. [Online; accessed 25-August-2015].

[53] Atmel. Atmega48a/pa/88a/pa/168a/pa/328/p. `http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf`. [Online; accessed 1-September-2015].

[54] Atmel. Atmel atmega640/v-1280/v-1281/v-2560/v-2561/v. `http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf`. [Online; accessed 1-September-2015].

[55] Atmel. Atmega16u4/atmega32u4. `http://www.atmel.com/images/atmel-7766-8-bit-avr-atmega16u4-32u4_summary.pdf`. [Online; accessed 1-September-2015].

[56] Philip Torrone. Why the arduino won and why it's here to stay. `http://makezine.com/2011/02/10/why-the-arduino-won-and-why-its-here-to-stay/`, February 2011. [Online; accessed 2-September-2015].

[57] What is a raspberry pi? `https://www.raspberrypi.org/help/what-is-a-raspberry-pi/`. [Online; accessed 3-September-2015].

[58] Raspberry Pi. Raspberry pi. *Raspberry Pi 1 HDMI 13 Secure Digital 34 Universal Serial Bus 56 Python (programming language) 84*, page 1, 2013.

[59] Andrew K Dennis. *Raspberry Pi home automation with Arduino, 2nd edition.* Packt Publishing Ltd, 2015.

[60] Mirjana Maksimović, Vladimir Vujović, Nikola Davidović, Vladimir Milošević, and Branko Perišić. Raspberry pi as internet of things hardware: Performances and constraints. *design issues*, 3:8, 2014.

[61] Maik Schmidt and Jaquelyn Carter. *Raspberry Pi: A Quick-start Guide.* Pragmatic Programmers, 2014.

[62] Electric Imp. specification: imp001. `https://electricimp.com/docs/attachments/hardware/datasheets/imp001_specification.pdf`. [Online; accessed 08-September-2015].

[63] Pinoccio. The schematics, board layout, and datasheets for the pinoccio board. `https://github.com/Pinoccio/hardware-pinoccio`. [Online; accessed 08-September-2015].

[64] Particle.io. Photon datasheet. `https://docs.particle.io/datasheets/photon-datasheet/`. [Online; accessed 08-September-2015].

[65] Espressif Systems IOT Team. Esp8266ex datasheet. `https://www.adafruit.com/images/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf`. [Online; accessed 08-September-2015].

[66] ESP8266. Arduino core for esp8266 wifi chip. https://github.com/esp8266/Arduino. [Online; accessed 09-September-2015].

[67] Arduino. Arduino pro mini. https://www.arduino.cc/en/Main/ArduinoBoardProMini. [Online; accessed 9-September-2015].

[68] João Paulo Barraca. Domótica diy (com comunicações m2m/iot). Presented at the 1st Lisbon Maker Faire, September 2014.

[69] Texas Instruments. Cc2530f32, cc2530f64 cc2530f128, cc2530f256 - a true system-on-chip solution for 2.4-ghz ieee 802.15.4 and zigbee applications. http://www.ti.com/lit/ds/symlink/cc2530.pdf. [Online; accessed 9-September-2015].

[70] Microchip. Mrf24j40data sheet ieee 802.15.4™ 2.4 ghz rf transceiver. http://ww1.microchip.com/downloads/en/DeviceDoc/39776C.pdf. [Online; accessed 9-September-2015].

[71] Texas Instruments. Cc2540f128, cc2540f256 - 2.4-ghz bluetooth® low energy system-on-chip. http://www.ti.com/lit/ds/symlink/cc2530.pdf. [Online; accessed 9-September-2015].

[72] Nordic. nrf24l01+ single chip 2.4ghz transceiver product specification v1.0. http://www.nordicsemi.com/eng/content/download/2726/34069/file/nRF24L01P_Product_Specification_1_0.pdf. [Online; accessed 8-September-2015].

[73] Duracell. Libraries. https://www.arduino.cc/en/Reference/Libraries. [Online; accessed 10-September-2015].

[74] Maniacbug. Arduino driver for nrf24l01 2.4ghz wireless transceiver. https://github.com/maniacbug/RF24. [Online; accessed 10-September-2015].

[75] Will Cooke. Howto: Very low power usage on pro mini v2 (arduino clone). http://www.whizzy.org/2015/06/howto-very-low-power-usage-on-pro-mini-v2-arduino-clone/. [Online; accessed 10-September-2015].

[76] Duracell. Alkaline-manganese dioxide battery. http://ww2.duracell.com/media/en-US/pdf/gtcl/Product_Data_Sheet/NA_DATASHEETS/MN1500_US_CT.pdf. [Online; accessed 10-September-2015].

[77] Zhu Yao-lin, Zhang Gao-qiang, Zhu Lei, and Xu Jin. Design of wireless multi-point temperature transmission system based on nrf24l01. In *Business Management and Electronic Information (BMEI), 2011 International Conference on*, volume 3, pages 780–783. IEEE, 2011.

[78] GNU Operating System. Gnu general public license. `http://www.gnu.org/licenses/gpl-3.0.en.html`. [Online; accessed 11-September-2015].

[79] GNU Operating System. Digital-output relative humidity & temperature sensor/-module dht22 (dht22 also named as am2302). `http://www.gnu.org/licenses/gpl-3.0.en.html`. [Online; accessed 11-September-2015].

[80] Ltd Aosong Electronics Co. Ds18b20 programmable resolution 1-wire digital thermometer. `https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf`. [Online; accessed 11-September-2015].

[81] Whitesmith. Qold - cold-chain monitoring for everyone. `http://soul-fi.ipn.pt/portfolio/whitesmith/`. [Online; accessed 12-September-2015].

[82] OpenEnergyMonitor. How to build an arduino energy monitor - measuring mains voltage and current. `http://openenergymonitor.org/emon/node/58`. [Online; accessed 12-September-2015].

[83] Arduino. Arduino / genuno uno. `https://www.arduino.cc/en/Main/arduinoBoardUno`. [Online; accessed 13-September-2015].

[84] XiDi Technology. Spilit-core current transformer. `https://www.arduino.cc/documents/datasheets/E000020_Non-invasiveAC%20currentSensor30A.pdf`. [Online; accessed 11-September-2015].