Eduard Tiron Martínez

Industrial Engineering

June 2015 (plan 94)

Final Project

Industrial Engineering

Development of a domestic or business security system for low volume, highly customizable production

MEMORY

Author: Eduard Tiron Martínez
Director: Oriol Boix Aragonès
Date: June 2015

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

1. Resum

Aquest treball pretén cobrir alguns dels passos involucrats en la creació d'una start-up especialitzada en el disseny, producció i comercialització de sistemes de seguretat modulars destinats a particulars i PIMEs. Es dóna èmfasi, principalment, en els aspectes tècnics del desenvolupament d'un prototip funcional.

El prototip esmentat consisteix en un sistema de seguretat relativament senzill però que ofereix, apart de les funcions típiques dels sistemes de seguretat, una base per incorporar-hi noves funcions que el client especifiqui (sempre i quan siguin tecnològicament viables). És per això que es considera un sistema de seguretat personalitzable.

El model de disseny d'aquest prototip bàsic és el d'una xarxa centralitzada de nodes independents. Cada node és un sensor o actuador que es connecta al node central, que és qui determina quines accions es realitzen en funció de la informació enviada pels sensors i la configuració introduïda per l'usuari. Un node és, essencialment, un dispositiu electrònic encapsulat en una carcassa, que realitza una funció determinada i es comunica sense fils. La xarxa és modular i pot acomodar qualsevol nou node que l'usuari vulgui incorporar.

Per al desenvolupament d'un prototip es fa ús de la plataforma de desenvolupament Arduino i d'un nombre elevat de components electrònics fàcilment adquiribles en volums petits. També s'utilitzen eines típiques d'una caixa d'eines i una impressora 3D. En el disseny hi intervenen suites de software tals com KiCad, Solidworks, Repetier Host, Slic3r o Autocad. És un objectiu primordial del projecte investigar i aprendre suficient sobre aquestes tècniques per desenvolupar la idea.

El treball es desenvolupa centrant-se primer en les funcions primordials d'un sistema de seguretat i deixant pel final les més supèrflues. En la memòria s'hi pot trobar una progressió d'apartats que dediquen una explicació a cada detall del desenvolupament. El llenguatge utilitzat és pragmàtic - atén les raons pràctiques d'enginyeria abans que les ciències de fons -, i pretén ser llegit com un treball de referència per projectes similars.

El resultat tangible creat és un sistema de seguretat format per cinc nodes diferents: un node central, un node per introduir codis de desactivació (manualment o amb un clauer amb tecnologia RFID), un sensor de moviment, un sensor d'obertura de portes i finestres, i una brunzidor per quan es dispara l'alarma. Tots ells són prototipus muntats en protoboards. El sistema és totalment funcional i està governat per 2.900 línies de codi C++ sense incloure llibreries externes.

El brunzidor, a més, també s'ha redissenyat en la seva versió Beta, amb el circuit electrònic soldat i inserit en una carcassa impresa en 3D.

Amb tot, aquest projecte ha estat una gran font de coneixements relacionats amb electrònica, C++, *embedded development*, soldadura amb estany, disseny CAD, impressió 3D i *supply chain*.

2. Table of contents

1. Resum 1
2. Table of contents 3
3. List of figures and tables 5
4. Glossary 8
5. Preface 9
   5.1. Motivation 9
   5.2. Scope of the project 9
6. Introduction 11
   6.1. Workflow 11
   6.2. Structure of the project 11
7. Development 12
   7.1. Arduino 12
   7.2. Wireless Communications 12
      7.2.1. nRF24l01+ module 13
   7.3. Node #1: The Central Node 18
      7.3.1. Prototyping space and power distribution 19
      7.3.2. Controller board 19
      7.3.3. LCD Screen 20
      7.3.4. Button keypad 25
      7.3.5. The StateMachine.h library 30
      7.3.6. The Communications.h library 33
      7.3.7. The Settings.h library 34
      7.3.8. Real Time Clock 36
   7.4. Node #2: The Movement Detector Node 37
      7.4.1. PIR sensor module 37
      7.4.2. Wireless communications protocol 39
      7.4.3. LED indicator 39

7.5. Node #3: Buzzer alarm ... 40
7.5.1. Wireless communications protocol ... 40
7.5.2. Buzzer ... 41
7.6. Node #4: The Key Tray Node ... 41
7.6.1. Numpad module ... 42
7.6.2. RFID module ... 43
7.6.3. Key Tray Node's finite state machine ... 45
7.6.4. Interaction feedback ... 46
7.7. Node #5: The Window Node ... 47
7.7.1. Running from batteries ... 48
7.8. Developing new nodes ... 49
7.9. Complete security system ... 50
8. Financial analysis ... 51
8.1. Components ... 51
8.2. Tools ... 51
8.3. Software ... 52
8.4. Design and manufacture costs ... 52
8.5. Total costs ... 52
9. Possible future improvements ... 54
10. Potential environmental affectations and measures taken to reduce impact ... 56
10.1. Fabrication ... 56
10.2. Use ... 56
10.3. Disposal ... 56
11. Results and conclusions ... 58
12. Bibliography ... 59
13. Other bibliographic references ... 62

3. List of figures and tables

Fig. 7.1 - Two compatible versions of the nRF24l01+ module with breakout boards. .. 13

Fig. 7.2 – nRF24l01+ module with PA (Power Amplifier) and LNA (Low Noise Amplifier), plus a detachable external antenna. 14

Fig. 7.3 – Custom connector (left) that allows the nRF24l01+ (right) to be plugged into a breadboard. 14

Fig. 7.4 – Structure of a packet sent over the RF24Network. The type specifies the size of the message (in accordance to a pre-established protocol). 16

Fig. 7.5 – Process followed to send a message. 16

Fig. 7.6 – Process followed by a receiver node to correctly read a received message, using an example of a predefined protocol. 17

Fig. 7.7 – Central Node prototype. 18

Fig. 7.8 - Breadboard with YwRobot power supply and power pins already connected with jumper cables. 19

Fig. 7.9 - Power supply with selectable output voltage. 19

Fig. 7.10 - Connected LCD screen. 20

Fig. 7.11 – Visual representation of the libraries involved in the screen display. 21

Fig. 7.12 – Flicker cause by calling the clear() method too often (at every loop). 22

Fig. 7.13 - Bug caused by not calling the clear() function when changing menus. 22

Fig. 7.14 - Bug caused by power surge. 23

Fig. 7.15 – Keyboard configuration in prototype. 26

Fig. 7.16 – Hypothetical keyboard configuration in final product. 26

Fig. 7.17 - Digital connection for a pushbutton in positive logic. 26

Fig. 7.18 - Digital connection for a pushbutton in negative logic. 26

Fig. 7.19 - Cross-section of a generic electronics device. 27

Fig. 7.20 - Analog keyboard prototype. 27

Fig. 7.21 - Analog keyboard schematic. 27

Fig. 7.22 – Behavior of mechanical switches. 28

Fig. 7.23 – State machine that acts as the motor of the AnalogButton.h library. 29

Fig. 7.24 – Finite state machine designed for the Central Node menu navigation. 31

Fig. 7.25 – Visual representation of the libraries involved in the wireless communications among the different alarm nodes. 33

Fig. 7.26 - ds1307 Real Time Clock module. 36

Fig. 7.27 - Movement detector node prototype. 37

Fig. 7.28 – PIR sensor. 37

Fig. 7.29 – PIR sensor module with detached diffuser. 37

Fig. 7.30 – PIR sensor module breakout board with components. 37

Fig. 7.31 – Internal working of the PIR sensor. Image provided by GLOLAB [25] 38

Fig. 7.32 – Buzzer Node prototype. 40

Fig. 7.33 – Two types of piezo buzzer. 41

Fig. 7.34 – Key tray node prototype. 42

Fig. 7.35 – Different versions of number pads. 42

Fig. 7.36 - Internal circuitry of a number pad, with detail of a bridge where a trace has to jump over two other traces. 43

Fig. 7.37 – RC522 RFID reader. 44

Fig. 7.38 – RFID tag, in a convenient keychain package. 44

Fig. 7.39 - Finite state machine governing the Key Tray Node behavior. 45

Fig. 7.40 – Window Node prototype. 47

Fig. 7.41 - Base for the development of new nodes. 49

Fig. 7.42 - Set of five nodes in Alpha version, plus the Buzzer Node in Beta version (upper right). 50

Fig. 9.1 – ENC28J60 Ethernet module (left) and ESP8266 WIFI module (right). 55

Fig. 9.2 – Relay module with two independently addressable relays. 55

Fig. 9.3 – Electrical diagram of a hypothetical Relay Node. 55

Fig. 10.1 – Solder paths made with solder. 57

Fig. 10.2 – Solder paths made with retention wire and the component bent legs. 57

Tab. 7.1 – Representation of the network's tree structure. 16

Tab. 7.2 - Characters that can be displayed by the LCD. 21

Tab. 7.3 - Representation on the LCD of the various displays, along with an explanation on how the Display.h library draws them. Notice that this is a multiple page table.. 25

Tab. 7.4 - Protocol followed for communications among nodes. 34

Tab. 7.5 – EEPROM addresses of the different settings stored in the Central Node. .. 35

Tab. 7.6 – Optimal hardware configuration for indoors PIR module on the Movement Detector Node.. 39

Tab. 8.1 - Summary of project costs. 53

All images have been created by the author of this project unless stated otherwise.

4. Glossary

AC: Alternate current.

ADC: Analog to digital converter.

BOM: Bill of materials.

BT: Bluetooth.

CRC: Cyclic redundancy check.

DC: Direct current.

EEPROM: Electrically erasable programmable read-only memory.

FSM: Finite state machine.

GNU LGPL: GNU Lesser general public license.

I^2C: Inter integrated circuit.

IC: Microcontroller.

IDE: Integrated development environment.

IR: Infrared.

LCD: Liquid crystal display.

LED: Light emitting diode.

OOP: Object oriented programming.

PCB: Printed circuit board.

PIR: Passive infrared receiver.

PLA: Polylactic acid.

PLC: Power line communications.

RF: Radio-frequency.

RFID: Radio-frequency identification.

RoHS: Restriction of hazardous substances directive.

RTC: Real time clock.

SMD: Durface mount device.

SPI: Serial periphereal interface.

UID: User identification.

5. Preface

5.1. Motivation

For some years I have been living alone in a flat in Barcelona. Even though it is not in an area prone to robberies, I still feel uneasy about leaving my belongings unguarded for many hours every day. Some personal projects and digital data are the product of countless hours of work and it would be very troublesome to lose them in a burglary. Moreover, some family members also leave their homes unguarded, and one of them lives in a very isolated area with a considerable risk of intrusions (there has been at least one attemp).

Normally, the solution would be to hire the service of a security firm and install one of their kits. However, these kits often are hardly customizable and expensive in the long run. I have been interested in electronics for two years, in which I have learned how to use many components similar to those used in professional alarm kits I have previously disassembled. This leads me to believe that, with the tools I have, I can prepare a kit which covers most security needs of people in the aforemented situations.

5.2. Scope of the project

The main objective is the design and manufacture of a functional prototype for a security system that serves as a deterrent to intruders (either in an apartment, a house or a small business), while not requiring third party services with recurrent payments. It must also hold a standard of usability for the lay user (who is not expected to know about electronics). Its interface should be user friendly and its normal operation should not cause frustration (unreliable communications, undetected user input, false triggers, etc.).

Apart from the Central Node, four other modules should be designed:

- Movement Detector Node: Sensor that detects the presence of intruders with a PIR sensor.
- Buzzer Node: Actuator that alerts bystanders with a loud noise.
- Key Tray Node: Entrance panel to arm/disarm the system by detecting an RFID tag in the user’s keychain, or entering a passcode with a numpad.
- Window Node: Sensor that detects when a door or window is opened.

An example of a full release plan for this product would require the design to be separated in three phases:

- Alpha version: The electronic design is tested on a breadboard until any bugs are corrected and the device works satisfactorily.
- Beta version: The electronics are soldered on a perfboard and a custom enclosure is 3D printed. This is done to test the user experience.
- Final version: The electronics are professionally redesigned into a PCB and the enclosure is made with plastic injection.

The scope of this project encompasses only the Alpha version of the nodes, plus the Beta version of one of the nodes. Therefore all nodes will be presented on breadboards, and at least one of them will be additionally presented on a perfboard inside a 3D printed enclosure.

The system is intended for both particular homes and small business, so proper measures have to be taken so that the network can communicate across the entire local.

In addition, the system must be modular enough so that new types of nodes can be added to the network or new functions can be added to existing nodes as per customer requests.

6. Introduction

6.1. Workflow

These are the major steps needed to complete the project, ordered chronologically. While some need to be sequential, others can be done simultaneously.

1. Analysis of the Arduino platform and its capabilities.
2. Selection of a proper communications channel between nodes.
3. Development of the Central Node's basic functionality.
4. Parallel development of the other nodes.
5. Development of a communications protocol between nodes.
6. Further addition of typical functions in a security system. The typical ones have priority over the most superfluous ones. For example, the ability to activate and deactivate the alarm is more important than configuring a piezo to play a melody on certain events.
7. Choose one of the nodes in alpha version to develop into a Beta version.

Additionally, every step includes debugging by default. If a bug is found, it is necessary to solve it before proceeding.

6.2. Structure of the project

The workflow specified in the previous section is developed in section 7, divided into successively smaller and smaller subsections which can then be addressed easily.

For each of these small subdivisions, after a brainstorming and investigating session, one or more possible solutions are presented for each little problem or need in the project. Some options are quickly discarded due to major inconviniences, while others are analised more in depth. Even if a solution is ultimately discarded, some testing is presented in this work in order to show the reason. In the end, the most practical solution is selected and developed further.

7. Development

7.1. Arduino

Arduino is a platform intended for agile electronics prototyping. This is thanks to the use of readily available boards that already take care of power management, providing I/O pin connectors, surge protection, etc. It is also due to the provided software which encapsulates much of the low-level code necessary to program microcontrollers. Finally, it is also thanks to community that has put online solutions to most problems that can arise.

For more information about the Arduino project, see Annex A.1.

For information on how external libraries are created for Arduino and other C++ projects, see Annexes A.1.3.

7.2. Wireless Communications

Each node in the network (central node, movement detector, buzzer…) will be physically separated within a building. There are many ways to establish communication between them. The first choice is the physical support:

- Cabled connection along the wall: A vast network of cables using a protocol such as Serial, SPI or I2C. This is an inconvenient system since the different nodes are supposed to have a certain degree of freedom. Having to remove and install new cables just to move a movement detector from one room to another would be too inconvenient for the user. However, if the cables are properly secured it is the method that offers most security.
- PLC (Power Line Communications): This protocol sends a signal along a building's AC power network without interfering with power transmission. While not actually wireless, it offers a biggest degree of freedom. However, if any of the different nodes was able to run on batteries, PLC would still restrict its position next to an outlet. Moreover, commercial solutions are difficult to modify and expensive.
- Wireless connection: Since they offer the highest degree of freedom of movement, wireless signals will be used. These offer great versatility but could lead to possible security issues.

For wireless communication, there are many different technologies:

- Infrared: Quickly discarded, since IR signals cannot go through walls.
- Bluetooth: Very attractive, since it encapsulates many difficult tasks.
- 433 MHz radiofrequency: Very cheap and simple.
- Xbee: Arguably the best networking line of components, albeit it's the most expensive option. It has been discarded due to its price.
- nRF24L01+ module: Even though it's complex to use, it allows for a network similar to those of Xbee at a much lower price.

The available options, then, are Bluetooth, 433 MHz RF and nRF24L01+.

Bluetooth has been discarded because it cannot communicate simultaneously with the rest of the nodes and has a small number of maximum links. The whole analysis is presented in Annex A.2.

433 MHz RF has proved to be too unreliable at sending messages from typical distances. The whole analysis is presented in Annex A.3.

7.2.1. nRF24l01+ module

The Nordic nRF24L01+ (pictured in Fig. 7.1) is a 2 Mbps RF transceiver IC for the 2.4 GHz ISM (Industrial, Scientific and Medical) band. With advanced power management, and a 1.9 to 3.6 V supply range, the nRF24L01+ provides a very low power solution enabling months of battery life from AA batteries or similar. It allows for both sending and receiving data, and is low cost ($1.10 for 5+ quantities). In addition, it has automatic error checking and reception acknowledgement incorporated.

With the right software, it can substitute the popular Xbee modules and their networking capabilities at a small fraction of the price. Moreover, an nRF24L01+ module at 3.3 V can reach as far as 433 MHz module at 12 V, or even farther with an external antenna.

This hardware is the most complex to use out of the previously mentioned wireless options. However, it offers the best results in range and reliability. Because of this, it has been chosen as the best option.

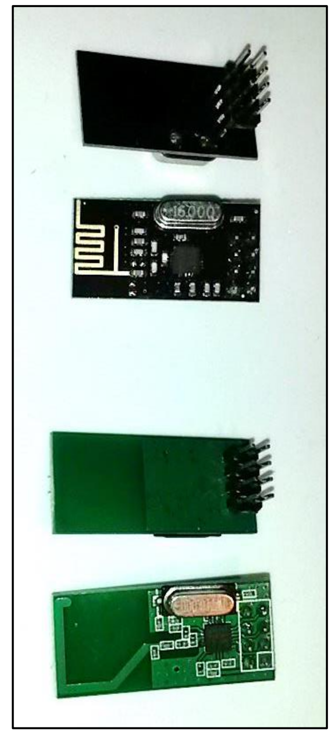


Fig. 7.1 - Two compatible versions of the nRF24l01+ module with breakout boards.

The nRF24L01+ module receives its name from the eponymous chip soldered to its surface. It is available in a compact 20-pin 4 x 4mm QFN package (the black squares seen in Fig. 7.1). It is commonly sold in different variations of breakout boards, which have compatible pinouts. Some versions have a traced antenna, while others have an external antenna (Fig. 7.2).

Fig. 7.2 – nRF24l01+ module with PA (Power Amplifier) and LNA (Low Noise Amplifier), plus a detachable external antenna.It is compatible with versions without antenna.

To increase the range of transmissions in a network of nodes, each node could have the version with antenna. However, due to its price (twice that of regular modules) this is not a good idea. Instead, since the network is a centralized network (all messages arrive or part from the same node), only the Central Node requires an antenna to increase transmission range. Of course, this relies on the fact that all three tested modules are compatible, which is true in this case according to testing.

This module has some hardware/electrical problems:

- Two pins can be assigned through software, while the rest are board-specific. Depending on whether the board is an Arduino UNO, Nano, MEGA… it will connect to different pin numbers.
- It must be fed 3.3 V. If it was run at 5 V, the IC would be irreversibly damaged. Fortunately Arduino boards have a 3.3 V output. The data pins, on the other hand, are 5 V tolerant.
- Its 2x4 pinout does not allow for any way to be plugged into a breadboard. Prototyping must be made with a custom connector (Fig. 7.3).
- When transmitting, the module requires a burst of current that the Arduino 3.3 V voltage regulator might not be able to provide. That is why a 4.7 or 10 μF capacitor is inserted between the V_{cc} and GND pins of the custom connector. It acts as a pool of readily available energy for power peaks (or in other words, a filter).

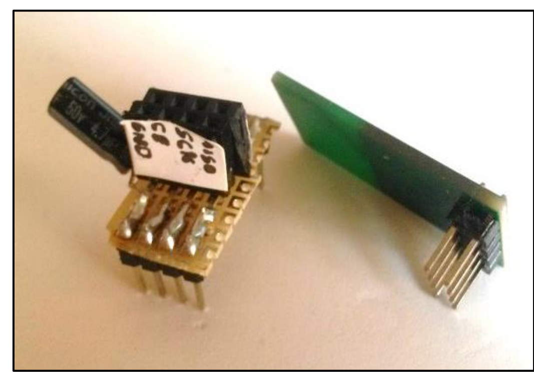


Fig. 7.3 – Custom connector (left) that allows the nRF24l01+ (right) to be plugged into a breadboard.
It must have a 4.7 or 10 μF capacitor soldered in its power pins as a reserve of energy for power peaks during transmissions.

The hardware is controlled via SPI thanks to the RF24 library. To control a network of multiple nodes, the RF24Network library is used.

7.2.1.1. RF24 library.h

The low level driver for the nRF24L01+ is the library RF24.h by J. Coliz, freely available under GNU General Public License v2. Once included in a project, it allows for direct one-to-one communication between a pair of modules [**14**, **16**]. There are many parameters that can be changed:

- Channel: useful to prevent interference with other 2.4 MHz devices such as Wi-Fi routers.
- CRClength: Changes the cyclic redundancy checksum length. It is a value calculated from the message and appended at the end of it. The receiver compares the received CRC and message to make sure no raw data was altered during transmission. A long CRC length improves error detection but decreases range.
- Data rate: The maximum data rate these modules can provide is 2 Mbps, but it can be changed to 1 Mbps or 250 kbps for greater range.
- PA level: Power amplification level. Increasing it means the module will consume more current but also increase its range.
- Retries: Any message will be sent up to a certain number of retries until it times out. The module can be configured to attempt up to 15 retries. To increase this number even further, the developer can simply call the sending function multiple times.
- Delay between retries: Increasing this value reduces noise and improves range.

One problem with this library is that sending a message blocks all other processes until it receives acknowledgment or times out. Time out happens after 1.5 s, so sometimes the Central Node might become unresponsive for 15 s (1.5 s x 10 retries) if the receiver has disappeared.

7.2.1.2. RF24Network.h library

J. Coliz has also made available, under the same license, a network layer library which works a level above RF24.h. After an RF24 instance is created, an RF24Network instance is created using the previous one [**17**].

The network generated by this library has a tree-like structure (see Tab. 7.1), in which each node is assigned an octal number as a number. When a node sends a message, it can only directly send it to its parent or children. To send it to any another node, it will automatically be rerouted through other nodes [**15**].

The largest node address is 05555, so 3,125 nodes are allowed on a single channel. For this project, however, only levels 0 and 1 are used, leaving the maximum number of nodes at eight. This is a very low number, but such limitation can be addressed with the notes in section 9, page 54.

Still, two nodes of the same type would count as one, since independent addressing for nodes of the same type has not been implemented (again, see section 9).

<table>
<tr><th>Level</th><th colspan="10">Nodes</th><th>Used by</th></tr>
<tr><td>Level 0</td><td colspan="10">0</td><td>Central Node</td></tr>
<tr><td>Level 1</td><td colspan="3">01</td><td colspan="3">02</td><td>…</td><td colspan="3">07</td><td>Other nodes</td></tr>
<tr><td>Level 2</td><td>011</td><td>…</td><td>017</td><td>021</td><td>…</td><td>017</td><td>…</td><td>071</td><td>…</td><td>077</td><td>Not used</td></tr>
</table>

Tab. 7.1 – Representation of the network's tree structure.

When a node has to send a message to another, it first creates an RF24NetworkHeader object, which acts as the "envelope" of the message (Fig. 7.4). It contains the receiver address and the message type, among other members. The type member is useful because the receiver cannot know in advance the size of the message (it could be a bool, an int, a 60 byte string…) [**19**]. It then sends the header and message together. The whole sending process is represented in Fig. 7.5.

Then, when the receiver detects a new message, it first reads the type field from the header, and depending on which type it is, the receiver will know what the size of the message is and which data-type to use in order to read it, based on a predefined protocol (see section 7.3.6, page 33). The whole receiving process is represented in Fig. 7.6.

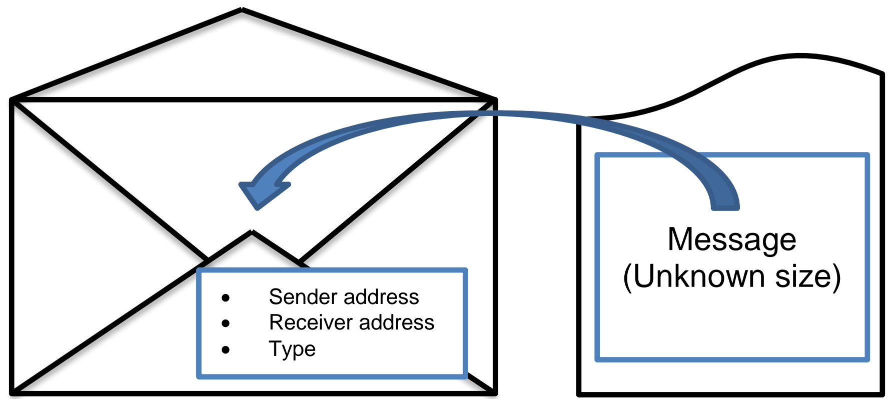


Fig. 7.4 – Structure of a packet sent over the RF24Network. The type specifies the size of the message (in accordance to a pre-established protocol).

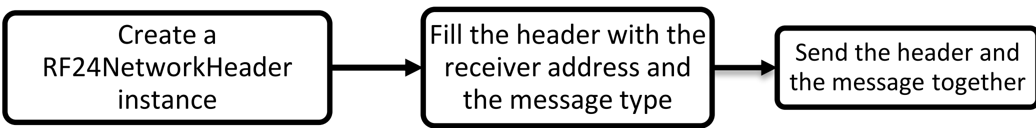


Fig. 7.5 – Process followed to send a message.

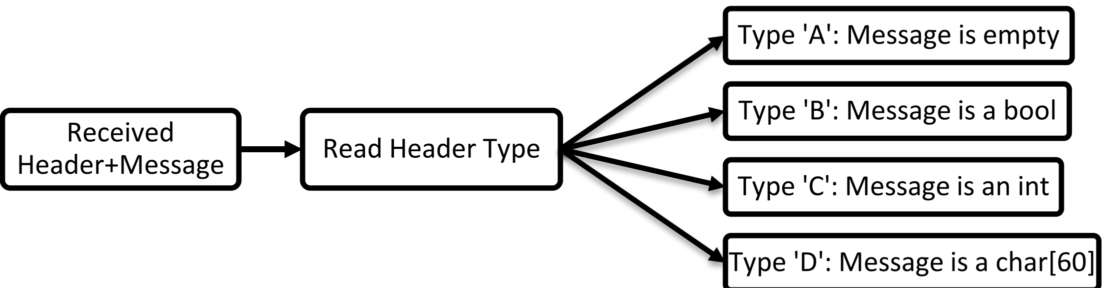


Fig. 7.6 – Process followed by a receiver node to correctly read a received message, using an example of a predefined protocol.

The type member of the header is of data-type char (1 byte long), so messages can be classified with up to 256 (2^8) different types.

The RF24network.write() method, which is used to send these messages, returns a Boolean true if the message has been correctly transmitted, thanks to the nRF24l01+ module's automatic transmission acknowledgement. Because of this it is not necessary to create redundant types of messages to make sure the messages arrive successfully. Additionally, the write() function is always called 10 times to increase the chance of success.

7.2.1.3. Corrective actions to improve range

On the first tests, the range between two antennae was short (about 10 m) and prone to errors. The following changes improved reception to almost no failed transmissions between any two points inside a walled 90 m^2 floor.

- Used module with external antenna at Central Node.
- Positioned the Central Node antenna vertically, since it has a blind cone oriented with its axis.
- Changed the RF channel to a recommended one with less interference.
- Set data rate to the minimum available (250 kbps).
- Set cyclic redundancy checksum to 8 bits, the minimum available before disabling it.
- Set the antenna power to its highest setting.
- One single call of the method RF24network.write() will try to send a message up to 15 times before reporting a failure. This has been circumvented to allow many more tries by calling the function many times.
- Replaced the original RF24 library by J. Coliz with a derived library (also known as *fork*). Since the original author stopped implementing bug solutions and improvements, other developers made modifications to the library which were later implemented in a single update and released online [**33**]. Also, a few lines in the library RF24Network were changed because of compatibility problems.

Some of these changes had to be implemented by manually editing the external library RF24Network, since the author did not provide adequate public methods.

7.3. Node #1: The Central Node

The Central Node is a device that acts as the decision-maker for the whole system. It stores the user settings, stores valid passcodes, and communicates with the rest of the network. All messages go through the Central Node. For example, if a Movement Detector Node is triggered, it sends a message to the Central Node, which in turn checks if the alarm is currently activated and, if it is, sends an outgoing message to the Buzzer Node. Since it communicates with the entire network wirelessly, its antenna is a more powerful version than those of other nodes.

The Central Node has an interface for the user to navigate a series of menus and submenus. The internal logic of this system (the backend) is a finite state machine. The visual output (the frontend) is presented in a 4x20 character LCD screen. The user uses five buttons to interact with the menus. It also contains a real time clock module to keep track of the time and date.

Since the rest of the system is useless without this node, it is supposed to be located at the center of the building to provide maximum coverage and deter possible tampering by intruders.

Its embedded code is by far the most complex of all node types. In order to keep all of it organized and easy to maintain, each piece of hardware is controlled by an external library. Additionally, each abstraction of its workflow is also encapsulated in its own library (communications protocols, saving and loading settings, analysis of user input, etc.).

The resulting prototype engineered is shown in Fig. 7.7.

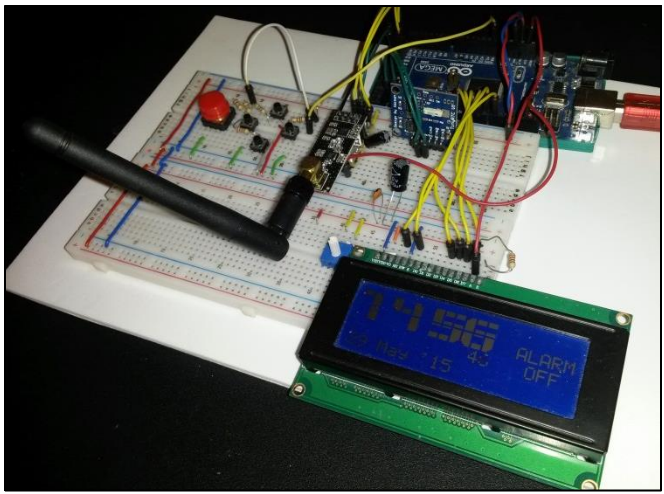


Fig. 7.7 – Central Node prototype.

7.3.1. Prototyping space and power distribution

The prototyping space consists of a pair of long typical breadboards. Its power distribution consists of four columns supplying V_{cc} (5 V) and four columns connected to ground. V_{cc} can be indistinctly harvested from:

- 5 V pin from the Arduino board, connected to a PC USB plug (which can supply no more than 0.5 A).
- YwRobot breadboard power supply. This will provide power if the Arduino cannot cover the device's demands. It takes V_{in} and regulates it to 5 V. In addition, a single pin has been soldered on the board which lets us bypass the voltage regulator and provide V_{in} when necessary. It is connected to a wall socket via a power supply with selectable output voltage V_{in} = [6, 7.5, 9, 12] V.

A 220 µF capacitor connected to V_{cc} and ground acts as a power supply filter to allow DC current but block any unwanted signals.

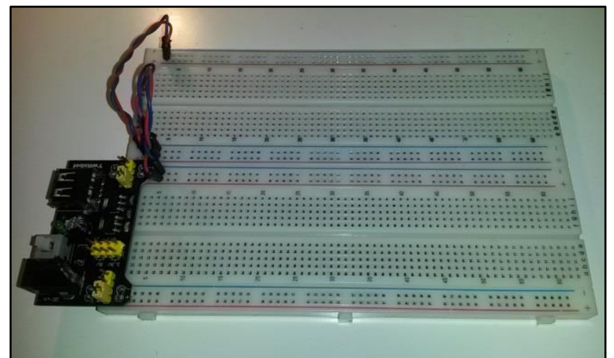


Fig. 7.8 - Breadboard with YwRobot power supply and power pins already connected with jumper cables.

Fig. 7.9 - Power supply with selectable output voltage.

7.3.2. Controller board

While the first choice for prototyping was an Arduino UNO, its small number of I/O pins was a factor which would have proved difficult to cope with. Basic functionalities like the LCD screen and the buttons would already take up most of the inputs, and solutions like multiplexors would free pins at the cost of increased development time and code complexity. The Arduino MEGA is more fitted for this purpose due to its high number of I/O pins.

One problem with Arduino MEGA is that the SMD chip can't be taken off and placed in a custom perfboard or PCB as can be done with the Arduino UNO's chip. After the prototyping phase there will be the following options:

- Include the whole Arduino MEGA board in the product (very expensive and big).
- Buy standalone ATmega2560 chips, a breakout board, and a Serial programmer (very time consuming, complex and mildly expensive).
- Make the necessary modifications so that the code can be compiled and uploaded to the Arduino UNO's chip, ATMEGA328. To cover the lack of I/O pins, multiplexors and

demultiplexors can be used. This chip can be extracted from the board and mounted on perfboard (very time consuming but very inexpensive).

From a practical point of view, the last option could be feasible. There are some ways to operate an LCD screen with just 3 pins (using a shift register plus the LiquidCrystal595 library, for example). Additionally, there is a way to connect a group of pushbuttons to just a single analog input.

7.3.3. LCD Screen

This is a common, Hitachi HD44780 driver compatible, 4x20 character, backlit LCD screen. The LiquidCrystal.h library, which is part of the Arduino core libraries, is used to control it.

The screen has a total of sixteen connections, but four of them can remain unused at the cost of data transfer speed, which is not important for this application. Two other pins are the anode and cathode for the backlight LED.

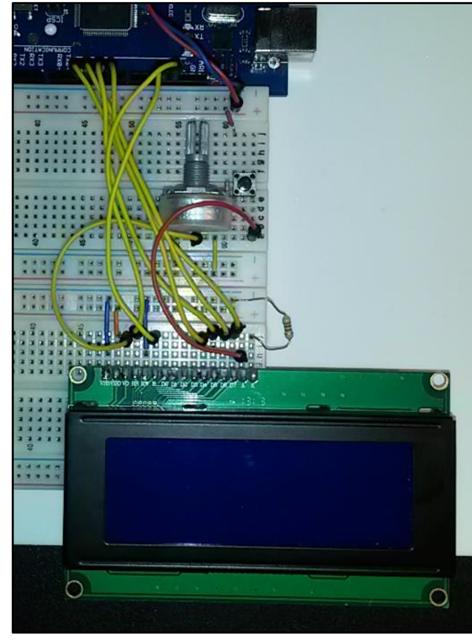


Fig. 7.10 - Connected LCD screen.

7.3.3.1. The LiquidCrystal.h library

This library is included in the Arduino environment. It allows an Arduino board to control liquid crystal displays (LCDs) based on the Hitachi HD44780 (or a compatible) chipset, which is found on most text-based LCDs. It is an Object-oriented library. Therefore, to use it a LiquidCrystal object is needed [**29, p.363-385**].

To print text on the display, the code must do two actions: first, use a function to move a cursor to the desired position, and then print a string of text there. If a character is not present on the keyboard, it can still be displayed by referencing its byte value (see Tab. 7.2).

There are other functions to create special characters by drawing them pixel by pixel in a 5x8 px glyph. Up to 8 custom characters can be stored in the LCD’s RAM.

Another useful method is the clear() method. It writes a blank space on all positions of the screen, effectively clearing it.

b7-b4 / b3-b0	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
0000	CG RAM (1)		0	@	P	`	p		ー	タ	ミ	α	p
0001	(2)	!	1	A	Q	a	q	。	ア	チ	ム	ä	q
0010	(3)	"	2	B	R	b	r	「	イ	ツ	メ	β	θ
0011	(4)	#	3	C	S	c	s	」	ウ	テ	モ	ε	∞
0100	(5)	$	4	D	T	d	t	、	エ	ト	ヤ	μ	Ω
0101	(6)	%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü
0110	(7)	&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
0111	CG RAM (8)	'	7	G	W	g	w	ァ	キ	ヌ	ラ	g	π
1000	CG RAM (1)	(	8	H	X	h	x	ィ	ク	ネ	リ	√	x̄
1001	(2)	)	9	I	Y	i	y	ゥ	ケ	ノ	ル	⁻¹	y
1010	(3)	*	:	J	Z	j	z	ェ	コ	ハ	レ	j	千
1011	(4)	+	;	K	[	k	{	ォ	サ	ヒ	ロ	ˣ	万
1100	(5)	,	<	L	¥	l	|	ャ	シ	フ	ワ	¢	円
1101	(6)	-	=	M	]	m	}	ュ	ス	ヘ	ン	£	÷
1110	(7)	.	>	N	^	n	→	ョ	セ	ホ	゛	ñ	
1111	CG RAM (8)	/	?	O	_	o	←	ッ	ソ	マ	゜	ö	█

Tab. 7.2 - Characters that can be displayed by the LCD.The first column holds 8 custom characters. Taken from the HD44780 driver datasheet.

7.3.3.2. The Display.h library

The main purpose of this library is to encapsulate the low-level methods of the LiquidCrystal.h library and provide methods to display, on the LCD screen, entire screens particularly useful to display information about the alarm. Since the state machine seen in chapter 7.3.5 is the backend, this can be considered the frontend.

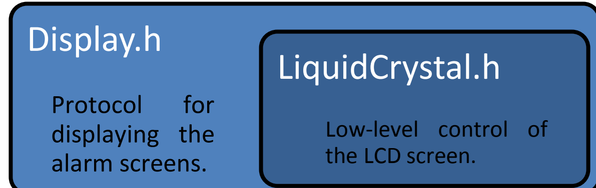


Fig. 7.11 – Visual representation of the libraries involved in the screen display.

A Display instance is created and its method update() is regularly called. It then requests information about the state machine. Depending on which is the current state of the FSM, the Display instance will display one screen or another. It also checks the cursor position to correctly draw it on those screens in which a cursor is present.

On every loop, the display draws a new screen. There are two ways to do this depending on how the LiquidCrystal.clear() is used:

One way is clearing the screen on every loop with LiquidCrystal.clear(), and then draw whichever new screen is needed. This causes the screen to flicker (Fig. 7.12).

Another way is to draw each new screen over the previous one without deleting it. This has the inconvenience of causing bugs when the user moves between different menus, since there might be some characters placed in positions where the new screen does not draw anything (Fig. 7.13).

A compromise has been made to prevent both errors. The clear() method is mostly avoided, and only used when the state machine has changed (either because the state has changed or because the cursor has moved).

Fig. 7.12 – Flicker cause by calling the clear() method too often (at every loop).

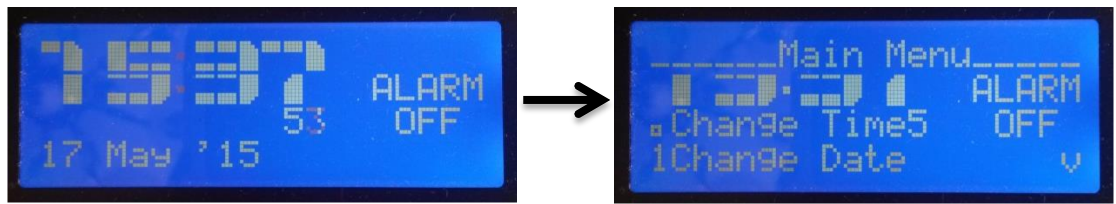


Fig. 7.13 - Bug caused by not calling the clear() function when changing menus.For example, when the user is in the Main Screen and accesses the Menu Root screen, the empty positions on the LCD do not get deleted and continue displaying the old characters until overwritten.

While prototyping, accidentally touching the wires occasionally causes a bug in the display (Fig. 7.14). This is likely due to power surges or poor contacts in the breadboard. In the final product, a PCB fit inside an enclosure would not have such severe problems, although a filter would still be recommendable. For the prototype, a filter has been added using two capacitors:

- 220 μF electrolytic capacitor as power supply filter (already seen in section 7.3.1, page 19).
- 22 pF ceramic capacitor to filter small, fast ripples.

The filter has been added close to the LCD power pins. Additionally, the power cables coming from the Arduino board have been twisted in order to cancel out any possible external electromagnetic interference.

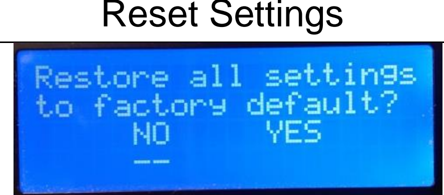
Fig. 7.14 - Bug caused by power surge.

7.3.3.3. Visual representation of each menu

The list in Tab. 7.3 shows the different menus the user can access, with notes regarding their display. For more information about their behavior, see section 7.3.5, page 30.

Loading Screen	
	Shown only when booting. The bar currently holds no purpose, since it runs synchronously and blocks execution of others tasks. However, it gives the user a useful visual cue. It is not part of the state machine and the user does not interact with it in any way.

Screen	Description
Main Screen 15 37 53 ALARM OFF 17 May '15	The hours and minutes are printed in big numbers made with eight custom characters. The colon appears and disappears every two seconds. The alarm state can change between ON and OFF.
Root Menu ______Main Menu_____ Change Time ^ Change Date Add passcode v	The list scrolls vertically as the user moves the cursor, which does not move. When the cursor is in the first position, the line 2nd is empty. When it is at the last position, the 4th line is empty. Additionally, two arrows are displayed to let the user know he can move up and down. They disappear when the user reaches the first or last entries.
Change Time _____Change time____ 16 : 08 OK BACK --	The cursor moves horizontally and is always made with two hyphens, even when below BACK.
Change Date _____Change date____ 17/05/2015 OK BACK --	Similar to the Change Time screen.
Add Passcode With alarm off press # on Key Tray Node and enter new pass- code/RFID tag.	Just displays a set of instructions, since this action is done from another node with more convenient hardware.
Delete Passcode Delete passcode/RFID 5 6 5 6 5 6 ^ 7 7 7 7 7 7 1 2 3 4 5 6 v	Similar to the Root Menu screen, but the list entries are the stored passcodes.
Change Backlight Mode ___Backlight mode___ Always on On button press Always off	Similar to the Root Menu screen, but since there are only three options it is easier to make the cursor move up and down than moving the entries.

Reset Settings	
 	The cursor moves horizontally between the two options.

Tab. 7.3 - Representation on the LCD of the various displays, along with an explanation on how the Display.h library draws them. Notice that this is a multiple page table.

7.3.3.4. The LCD backlight

The Display.h library, apart from taking care of what is drawn on the LCD, also manages its LED backlight. This light has three modes of operation: ‘Always on’, ‘On button press’ and ‘Always off’.

The ‘Always on’ and ‘Always off’ modes are self-explanatory. Otherwise, if it is set at ‘On button press’, the function that controls it does three actions:

- When the alarm is powered on, keep the backlight on for 10 s and then turn it off.
- When the user presses any button, turn it on.
- When 10 s have passed since the last button press, turn it off.

7.3.4. Button keypad

On the LCD screen the user should be able to open and navigate different menus with options to change. These are just two examples of how these menus are expected to work:

- Change time: the user moves the cursor over the hours or minutes, and increases or decreases the value. The he moves the cursor to an OK, presses a key and the time is stored.
- Delete passcodes: the user moves the cursor through a list of stored passwords and selects which one he wants to delete.

The minimum number of pushbuttons for a user input depends on the following factors:

- In order to use the bare possible minimum number of components, two pushbuttons would be enough as long as every menu has a ‘Back’ entry. One button would be used to advance through a list / increase a value and the other would be used to select an option / save a value. This method has the inconvenience of being very tedious for the user.

- For a user-friendly interface, five buttons would be ideal: Select, Up, Down, Left and Right. This would allow for a much more fluid experience when navigating menus and changing settings.

Since this is a highly customizable product, the user is expected to use the interface frequently enough to warrant the use of five pushbuttons instead of two. For prototyping, they are arranged in the configuration presented in Fig. 7.15. However, in the final, professionally manufactured enclosure they would follow the one seen in Fig. 7.16.

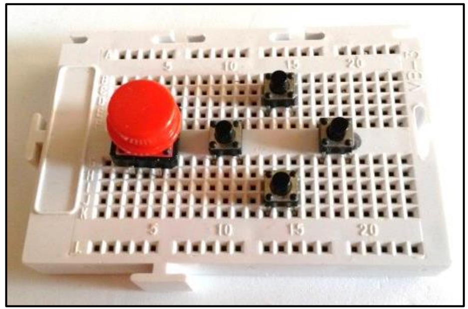


Fig. 7.15 – Keyboard configuration in prototype.

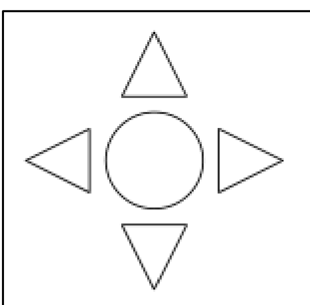


Fig. 7.16 – Hypothetical keyboard configuration in final product.

7.3.4.1. Connection to board

A pushbutton is usually connected to a single digital pin on the Arduino board, along with a pull-up or pull-down resistor (see Fig. 7.17 and Fig. 7.18). Fortunately, the resistor can be avoided since it is already incorporated into the Arduino's microcontroller [**29, p.154-155**].

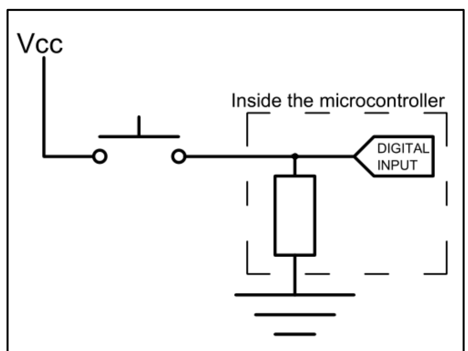


Fig. 7.17 - Digital connection for a pushbutton in positive logic.

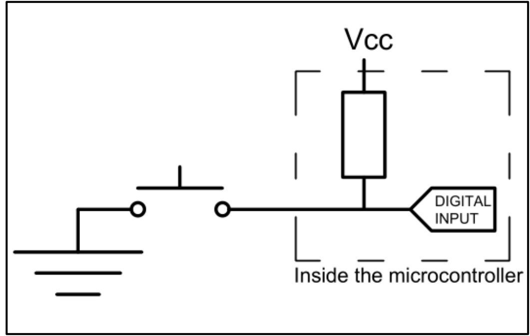


Fig. 7.18 - Digital connection for a pushbutton in negative logic.

With positive logic, a read on the pushbutton pin will yield a high value when the button is pressed and a low value when it is not. The opposite happens if it is connected in negative logic.

This configuration is very simple, but requires one digital input for each pushbutton. This can cause problems if the microcontroller does not have too many inputs.

Additionally, having too many cables can cause mechanical problems. On many commercial products, there is a separate PCB mounted on a frontal panel with a few buttons. To connect this PCB with the rest of the electronics, a few long cables go from one PCB to the other (Fig. 7.19). This can be acceptable if there are few cables and they are well connected, or if they are ribbon cables. However, dangling cables tend to suffer wear due to friction and they are generally hard to maintain.

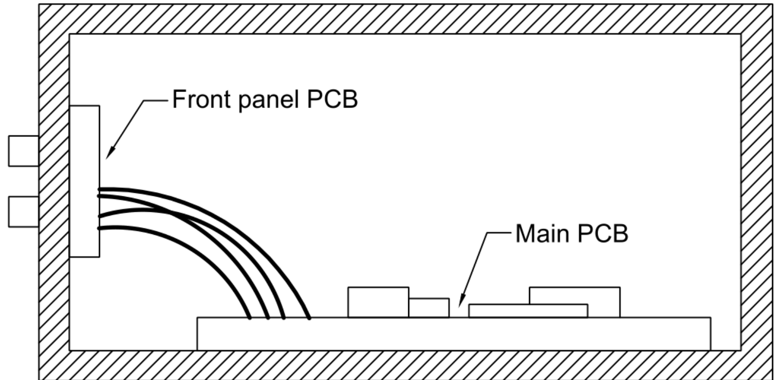


Fig. 7.19 - Cross-section of a generic electronics device.Connecting various separate PCBs with dangling cables is a valid solution, but having too many of them can cause problems.

Fortunately, there is a different approach which only uses a single analog pin to read many pushbuttons.

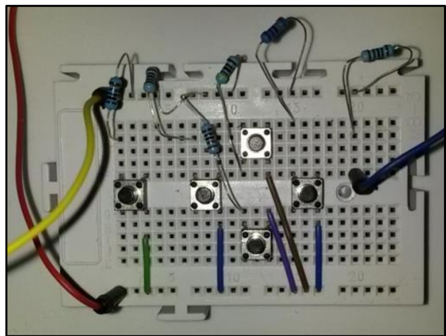


Fig. 7.20 - Analog keyboard prototype.

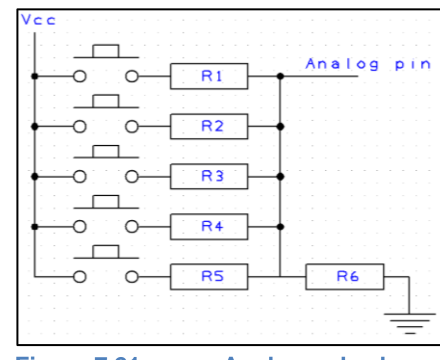


Fig. 7.21 - Analog keyboard schematic.

With the circuit mounted on Fig. 7.20 with the schematic from Fig. 7.21, a single read of the analog pin yields a voltage value that can identify which button is pushed. Each button connects a different voltage divider that sets the output line at a different voltage [**29**, **p.170-172**].

Its main advantage is that, with three pins (V_{cc}, ground and output), it is possible to read the state of many pushbuttons simultaneously. This technique has been tested and works well with at least 10 pushbuttons, more than enough for this project. However, the maximum number would depend on:

- The selected resistors' tolerance.
- The input voltage fluctuations. When the button is pressed, the output voltage oscillates for a short time before reaching a definitive value.
- The Analog to Digital Converter (ADC) module in the Arduino can only return a discrete number between 0 (0 V) and 1024 (5 V), with an error of ±2.

Its drawbacks are:

- Bigger number of components (extra resistors).
- Slower reads. Reading an analog value is an order of magnitude slower than reading a digital one. Even more so when making redundant reads to obtain an averaged value.
- Inaccuracy on extreme temperatures. On very hot or cold days, the resistance of each resistor can vary and modify the expected output voltage.
- Added difficulty when dealing with multiple buttons pressed at the same time. Extra care must be taken when selecting the resistors, in order to ensure that no combination of buttons causes the same voltage drop as another combination.

An example where this cable configuration is used is in the connections from the car's steering wheel to the rest of the car. Since the cables inside the axis are continuously moving, fewer cables decrease the risk of failure.

7.3.4.2. Debouncing

Debouncing is needed when using any mechanical switch due to very small bounces that occur when pressing and might register as multiple presses in a short time, as seen in Fig. 7.22.

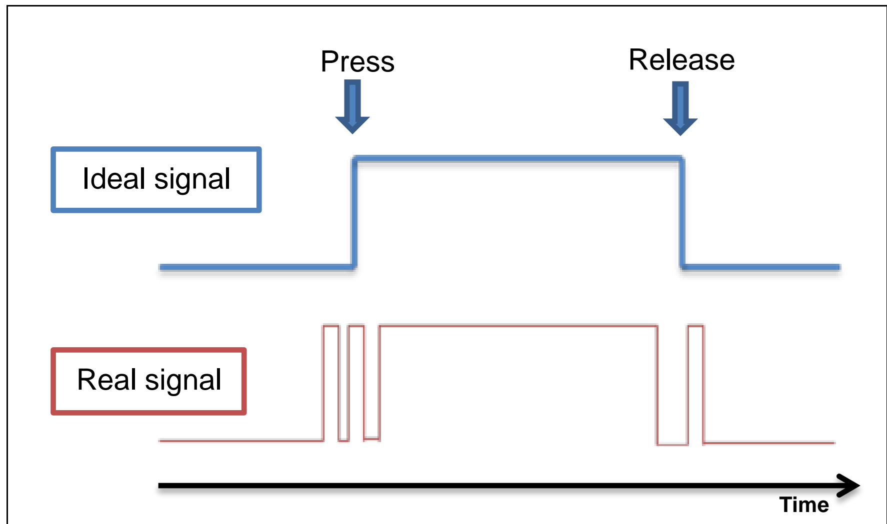


Fig. 7.22 – Behavior of mechanical switches.

7.3.4.3. AnalogButton.h library

Even though there are a few libraries online to control pushbuttons in an analog connection, they do not behave as required. They work by assigning a single function to an event (press, release, double click…). In this project, the library should only provide information about the state of the buttons, but not perform any action since they are performed by other parts of the code.

A new library was built from scratch to address this. It monitors the state of a single button thanks to a state machine (Fig. 7.23).

The user has to create an instance of the AnalogButton class and pass it the minimum and maximum voltages that can be detected when pressing the button (obtained empirically), plus the debounce time (recommended is around 30 ms). The voltage range should be expanded to compensate for the drawbacks stated in section 7.3.4.1 (mostly the temperature related drawback).

The state machine has two states that indicate whether the button is pressed or not, plus two extra states to address debouncing by making sure that potential presses and releases are not just jittering of the pushbutton contacts.

At each loop of the program, the AnalogButon instance has to be updated with the update() method. Afterwards, other methods can be used to check whether a button was pressed (transition 1-2) or released (transition 3-0) in the last loop.

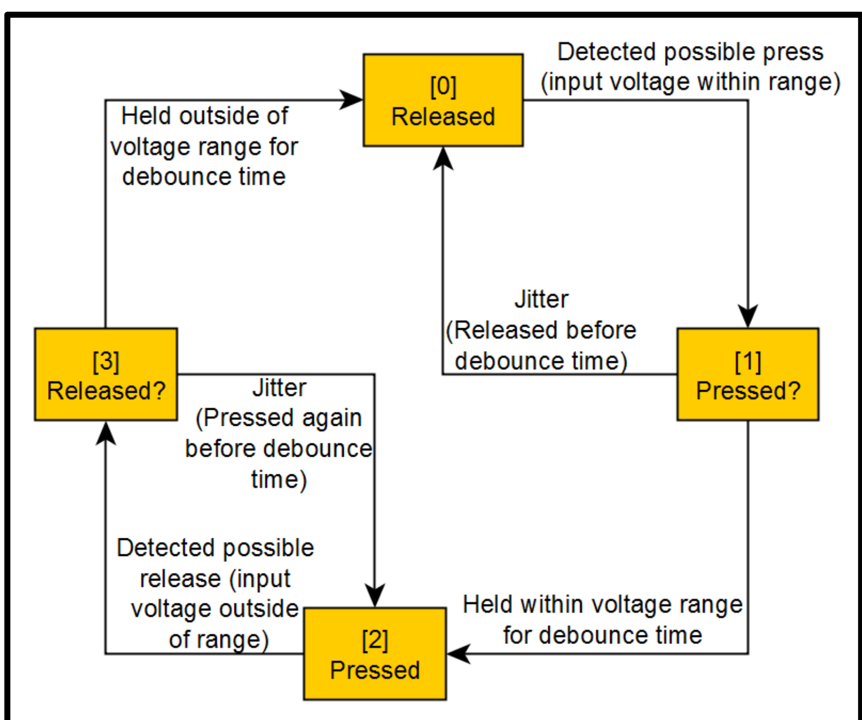


Fig. 7.23 – State machine that acts as the motor of the AnalogButton.h library.

7.3.4.4. AnalogKeybard.h library

The library in the previous section, AnalogButton.h, permitted the management of a single pushbutton. However, there are five of them in the Central Node and the main sketch would be cluttered with an instance for each of them.

Instead, another custom library called AnalogKeyboard.h was created. It encapsulates the AnalogButton objects, but still keeps them public so that they are addressable from the main code. Therefore, the code to declare the instances:

```
AnalogButton BTNSelect(917,926,30);
AnalogButton BTNUp(691,698,30);
AnalogButton BTNRight(602,611,30);
AnalogButton BTNDown(507,517,30);
AnalogButton BTNLeft(833,842,30);
```

Becomes just:

```
AnalogKeyboard analogKeyboard(A0);
```

On the other hand, it also provides methods to easily check if any key has been pressed, without necessarily specifying which one. One method checks if any of the four directional keys has been pressed (to understand its usefulness, see the transition from state 0 to 1 in Fig. 7.24, page 31). Another method checks if any of all five keys has been pressed (useful to detect when to turn on the LCD backlight, as seen in section 7.3.3.4, page 25).

7.3.5. The StateMachine.h library

The Central Node allows the user to navigate through a series of menus on an LCD screen using the directional keypad. StateMachine.h is an object oriented library that takes care of analyzing the user input and updating a series of variables that indicate which is the current menu by characterizing them as states in a finite state machine. Its conceptual structure is shown in Fig. 7.24.

It is important to note that the code in this library is purely backend. StateMachine updates the state machine and executes some necessary actions during transitions such as saving settings. It only does background logic and does not draw anything on the LCD. Separating the internal logic from the display logic is considered good praxis.

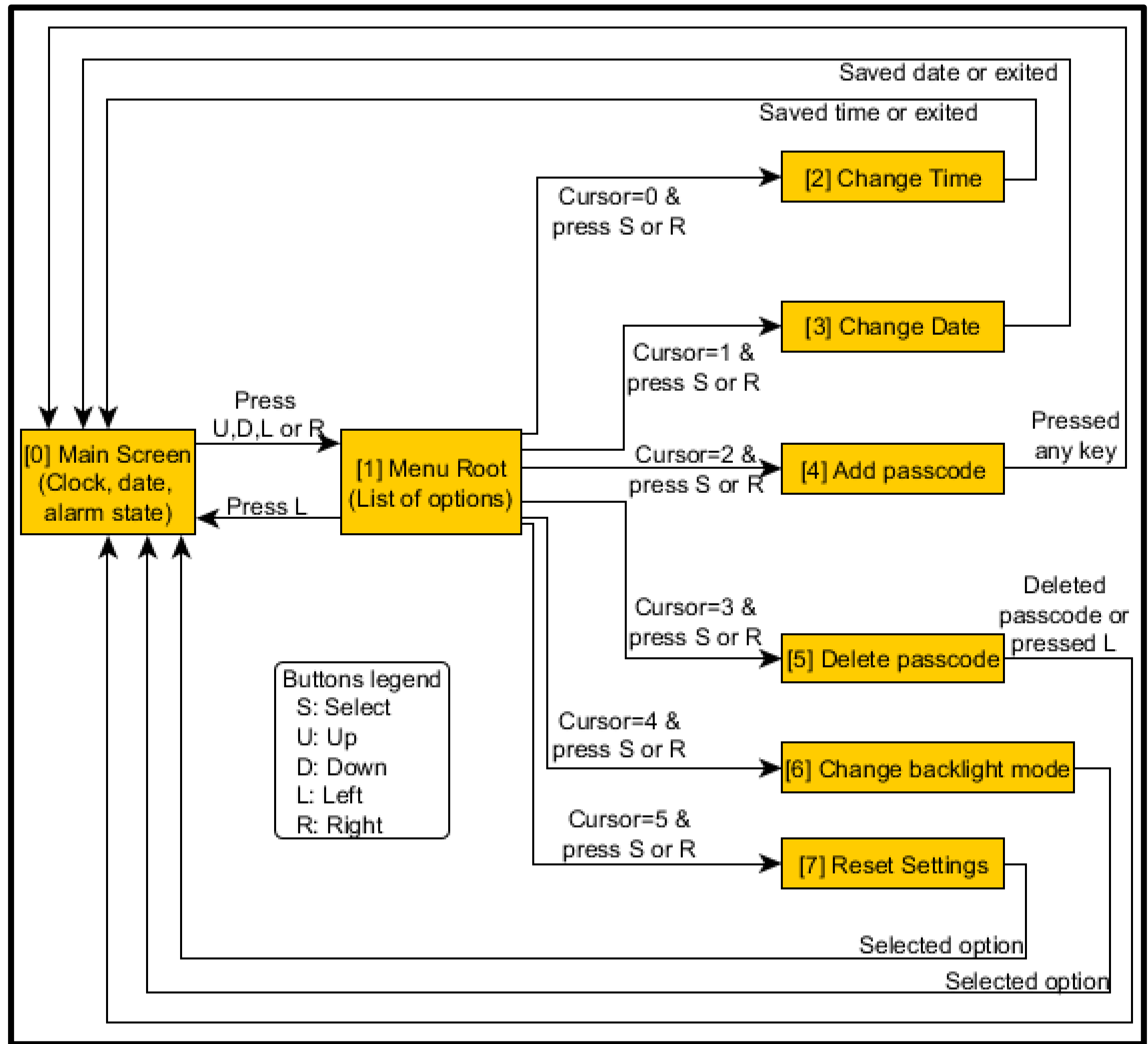


Fig. 7.24 – Finite state machine designed for the Central Node menu navigation.A transition can be triggered by a simple event (pressing a button) or a more complex one (navigating through a list and selecting an option).

The state of the FSM is stored in a variable. Additionally, since some menus allow for the user to move a cursor through a list of options, a second variable holds the position of the cursor.

The code uses a switch() structure to represent the FSM:

```
switch(state){

  case 0:
    if(transitionCondition1){
      DoSomething();
      state=1;
    }
    else if(transitionCondition2){
      DoSomethingElse();
      state=2;
    }
```

```
    break;

  case 1:
  […]
}
```

These are the explanations of how each state works:

- Main Screen: The default screen which will show the time, date and state of the alarm (activated/deactivated).

- Root Menu: It is a list of submenus. The user can move the cursor through the list. When he presses the Select button, the FSM makes the corresponding transition based on which entry the cursor was located.

- Change Time: The user can move the cursor among four positions (hour, minute, Ok and Back). Placing it under the hour or minute allows him to increase or decrease by pressing up or down. They loop when reaching the maximum or minimum. As the user increases or decreases the hour or minute value, the changes are stored in temporal variables. When the user moves the cursor to OK and presses the Select button, the clock is reconfigured with these temporal values and the state machine returns to the Main Screen. The user can also select Back to return to the Main Screen without saving any change. The seconds cannot be manually changed, but they are automatically set to zero whenever a new time is set.

- Change Date: Similar to Change Time, but allowing the change of day, month and year. The year value starts at 2000 and does not loop. There are no prevention measures for erroneous dates such as 30/02/2015.

- Add Passcode: This functionality is not implemented in the Central Node. It will be implemented in the Key Tray Node since it will have a more adequate hardware. If the user selects this option, he is presented with instructions on how to add passcodes from the Key Tray Node.

- Delete Passcode: The user is presented a list of passcodes sored in the database. This menu works like the Root Menu. By scrolling down the cursor and selecting a passcode it will be removed from the database.

- Change Backlight Mode: This menu lets the user scroll through the two backlight options: always on or only when a button is pressed (see section 7.3.3.4, page 25).

- Reset Settings: Offers an option to wipe stored user data and restores it to factory defaults. It sets the alarm state to deactivated, the backlight mode to 'Always on', and empties the passcode list. The time and date are not affected. The user can move a cursor horizontally to either confirm or exit this mode without performing the reset.

7.3.6. The Communications.h library

The code inside this library is in charge of reading any incoming message and acting accordingly. For example, if a sensor sends an alert, the Central Node will decide to send a message to the Buzzer Node to activate.

It encapsulates the RF24.h and RF24Network.h libraries, and works a level above them (as seen in Fig. 7.25).

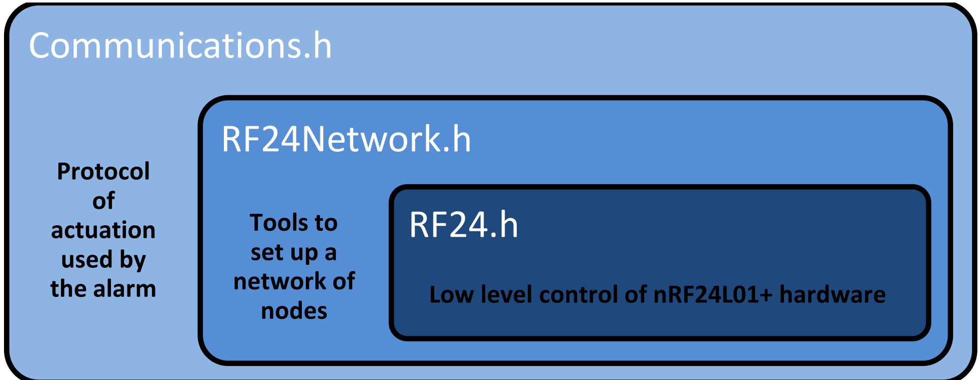


Fig. 7.25 – Visual representation of the libraries involved in the wireless communications among the different alarm nodes.

At the beginning of the program, it initializes two instances of RF24 and RF24Network as private members. Then, on every loop, the public method Communications.update() uses these instances to check for any new message. If a message has been received, the first action to do is read its header in order to know its type and what is the size of the message in bytes (as seen in section 7.2.1.2, page 15).

The custom protocol shown in Tab. 7.4 has been created in order to let the receiver know how to treat each type of message. Again, see section 7.2.1.2 to understand the importance of such protocol.

Type	From	To	Data type	Explanation
‘A’	Key Tray Node	Central Node	⌀	The user has pressed the alarm activation key in the Key Tray Node. It has sent notice to the Central Node.
‘B’	Key Tray Node	Central Node	char[6]	The user has entered a passcode or passed a RFID tag to the Key Tray Node while the alarm is activated. The code has been sent to the Central Node for verification.
‘C’	Central Node	Key Tray Node	bool	Confirmation/Denial of access.
‘D’	Key Tray Node	Central Node	char[6]	New passcode sent to be added to database (unless database is full or passcode is repeated).
‘E’	Movement Detector Node	Central Node	⌀	A Movement Detector Node has been triggered and it has sent notice to the Central Node.
‘F’	Central Node	Buzzer Node	⌀	The Central Node gives the order to the Buzzer Node to start emitting a loud noise.
‘G’	Window Node	Central Node	⌀	A Window Node has been triggered and it has sent notice to the Central Node.

Tab. 7.4 - Protocol followed for communications among nodes.It is used in order to let the receiver know the size of the message before reading it, thanks to the associated type.

After the Communications.update() method detects a new message and reads its type, it executes a switch(type) statement. For each case it does a different action:

- Type ‘A’: Set the alarm setting to ‘Activated’.
- Type ‘B’: Compare the received passcode with those in the database. Then return a message of type ‘C’ with permission or denial of access.
- Type ‘D’: Add new passcode to database.
- Type ‘E’ or ‘G’: If the alarm is activated, send a message of type ‘F’ to the Buzzer Node.

7.3.7. The Settings.h library

Its main function is to provide methods to save and retrieve user settings. These settings are stored in the microcontroller’s permanent memory, or EEPROM (see Annex A.1.4), so that they can be recalled even if the board is powered off.

This class has no data members, and its method members are static. Therefore, it is not necessary to create an instance of the Settings class. For more information on static members, see Annex C.2.1.

The library provides public methods to work with the following settings:

- Alarm state: a Boolean set to true if the alarm is activated and false otherwise. The library includes methods to get and set this setting.
- Backlight mode: a byte is set to 0 if the backlight is set to ‘Always on’, 1 if it is set to ‘On button press’ and 2 if it is set to ‘Always off’. The library includes methods to get and set this setting.
- Passcode database: an array of up to 10 valid passcodes. The library includes, among others, methods to:
    - Check if a given passcode is valid by comparing it with those in the database.
    - Add a new passcode.
    - Delete a passcode.

There is also a public method to restore all settings to factory defaults. It sets the alarm as deactivated, the backlight mode as ‘Always on’, and deletes all stored passcodes.

Additionally, there is a public method that retrieves a single passcode from a position in the database. This is useful when the Display.h library needs to retrieve a passcode to draw it on the screen. As a function returning an array, the code for this method has a certain degree of complexity (see Annex C.3).

The implemented settings have fixed addresses on the EEPROM, as seen in Tab. 7.5.

Contents	Data type	Size (bytes)	First byte address	Last byte address
Alarm state (activated / deactivated)	Bool[1]	1	0	0
Backlight mode (Always on / On button press / Always off)	Byte	1	1	1
10 valid passcodes (each passcode is a 6 byte array)	Byte[6][10]	60	100	159

Tab. 7.5 – EEPROM addresses of the different settings stored in the Central Node.

[1] Bool variables theoretically need only one bit of memory. Unfortunately, the EEPROM cannot access individual bits so an entire byte must be used to hold the variable.

7.3.8. Real Time Clock

The Arduino board can keep track of time in a rudimentary way: calling the function millis() gets the number of milliseconds since the board was activated. The downside is that this method relies on the board being permanently powered on. Otherwise the time is reset on each reboot.

The real time clock (RTC) ds1307, shown in Fig. 7.26, is a small board with its own coin cell that solves this problem. It can be setup with a time and date and it will keep itself running for as long as the battery holds a charge. The board incorporates a charging circuit that keeps the cell charged. If external power is interrupted, the coin cell will last 1 year.

Silicon circuit speeds are affected by temperature. This one is no exception, but the error is reported as ±2 s/day. It should not be a problem, but if it were, there are alternatives with an incorporated temperature sensor and code with speed compensation. The time could also be corrected periodically with an online source if an internet connection is implemented.

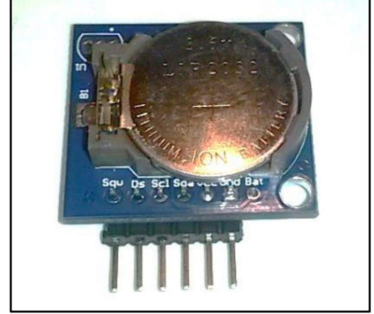


Fig. 7.26 - ds1307 Real Time Clock module.

JeeLabs provides, under public domain license, the RTClib.h library, which is used to control this component. It encapsulates the protocol used to communicate with the module, I^2C (see Annex B.2). This library is very easy to use, so no further encapsulation is needed and the library can be used from the main sketch.

One useful resource is that Arduino's IDE ability to grab the date and time of the PC to configure the ds1307 using this line:

```
Clock.adjust(DateTime(__DATE__, __TIME__));
```

When the code is compiled, `__DATE__` and `__TIME__` are automatically replaced by corresponding values which the compiler requests from the PC's own clock. This means that every time this line is run (when the Central Node is restarted) the ds1307 will be configured back to the same time. To prevent this, the code must be uploaded once with this line and another time immediately afterwards with the line commented.

While the RTC is only used for aesthetic purposes on the LCD screen, it could prove useful if the alarm was given the ability to log events.

7.4. Node #2: The Movement Detector Node

This sensor module is supposed to detect any person that walks inside its frontal cone. Upon detection it sends a notification to the Central Node.

Prototyping is done on an Arduino Nano. It is simple enough that no OOP is used, and all the code is written on a single file (excluding external libraries). The result of the development shown in this section is the prototype in Fig. 7.27.

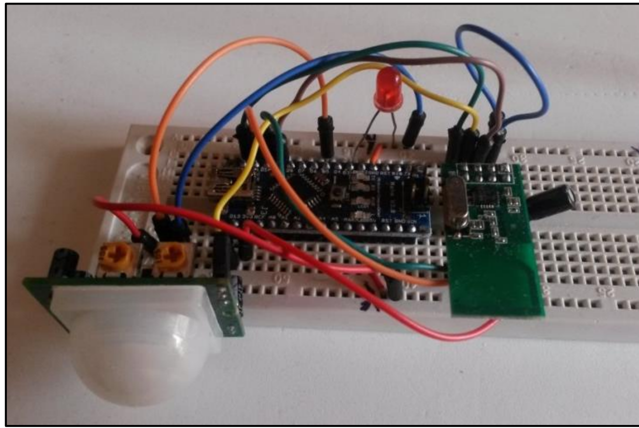


Fig. 7.27 - Movement detector node prototype.

7.4.1. PIR sensor module

The characteristic component of this node is the ubiquitous Pyroelectric Infrared sensor, found in the vast majority of security systems. It can detect small changes in the ambient infrared light. Since all matter at more than 0 K radiates a certain amount or IR light (particularly living beings), this sensor can detect when a person moves inside a cone in front of it. The whole module is actually formed by:

- PIR sensor
- BISS0001 microcontroller
- Breakout board + other components
- Diffuser

Fig. 7.28 – PIR sensor.

Fig. 7.29 – PIR sensor module with detached diffuser.

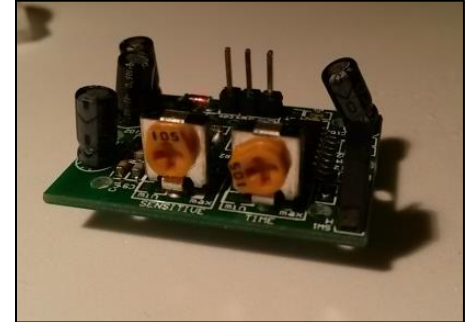


Fig. 7.30 – PIR sensor module breakout board with components.

The PIR sensor is a package with two sensing elements side by side. A body passing in front of the sensor will activate first one and then the other element whereas other sources will affect both elements simultaneously and be cancelled. This arrangement cancels signals caused by vibration, temperature changes and sunlight, elements that would cause many false triggers. The radiation source must pass across the sensor in a horizontal direction so that the pyroelectric elements are sequentially exposed to the IR source [**25**].

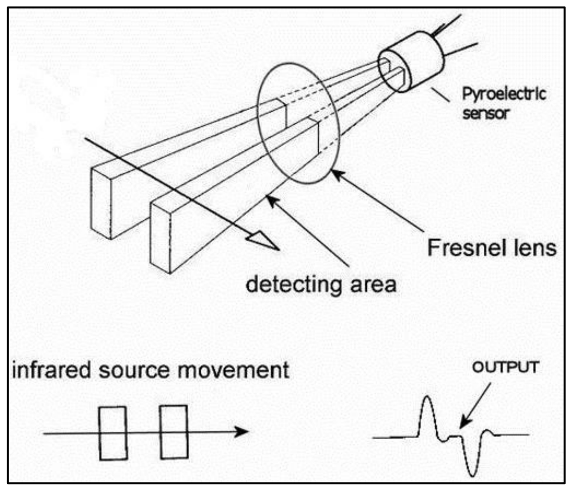


Fig. 7.31 – Internal working of the PIR sensor. Image provided by GLOLAB [25]

Its pinout consist of three lines: 5 V, OUT and GND. When proper power is supplied, the OUT signal will remain low until movement is detected, at which point it will go high. It will go low some time later (the actual time depends on hardware settings explained later).

When the PIR module is powered on, it is recommended to wait a few seconds for it to calibrate to ambient IR light. During this time, nobody should stand, walk or project shadows in front of the PIR module.

The BISS0001 microcontroller is quite complex, and allows various modes of operation. The pinout on the breakout board does not provide any means to configure these modes through software. However, there are a few components soldered in the breakout board that permit the module configuration through hardware [**6, 24**]:

- Retriggering jumper: A set of three pins and a jumper. Depending on the position of the jumper, the chip will enter H or L mode.
    - L mode, non-retriggering: When movement is detected, the output will go high, wait n seconds, and then go low. If movement is still being detected, it will immediately go high again.
    - H mode, retriggering: When movement is detected, the output will go high and wait n seconds. If movement is detected, the timer will reset. Once the module stops detecting movement and the timer has reached zero, the output line will go low. This mode is useful to make sure that a single source of movement does not trigger the output multiple times.

- Sensitivity potentiometer: Changes the module's sensitivity.
  - Maximum, it can detect small movements from approximately 4 m.
  - Minimum, it only detects movement up to 1 m.

- Time: Determines how long the OUT line will remain high after the module detects movement.
  - Maximum: 20 s.
  - Minimum: 1 s.
- Holes for LDR: By soldering a light dependent resistor on these holes, the module can detect daylight. If used, it will remain inactive during the day and turn on only during nighttime.

The module has been manually configured to achieve optimal results indoors. The settings used, seen in Tab. 7.6, have been obtained by trial and error in a certain set of lightning and spatial conditions which will work for most scenarios. Still, the final product might be used in very different locations, so it is up to the user to play with the potentiometers until achieving the desired result.

Calibration time	Mode	Sensitivity	Time	LDR
5 s	H	80 %	0 % (1 s)	Not used

Tab. 7.6 – Optimal hardware configuration for indoors PIR module on the Movement Detector Node

Reflective surfaces such as mirrors or glass panels can reflect light in such a way that a person can trigger the PIR sensor without being in its frontal cone. This can be used strategically to provide a better coverage of a room or can be thought of an inconvenience. Either way, it has to be taken into consideration when selecting a location for the Movement Detector Node.

7.4.2. Wireless communications protocol

It uses the RF24.h and RF24Network.h classes as stated in section 7.2.1, page 13. The only message it sends is one of type 'E', which has a header but no message (like an empty envelope). It suffices since no information is sent to the Central Node, other than the actual detection of movement.

7.4.3. LED indicator

A red LED serves as an indicator of activity.

During calibration, it continuously changes its brightness in a heartbeat pattern, using a cosine function to calculate the PWM duty rate.

While idle, it remains off to save power. When movement is detected, it turns on.

7.5. Node #3: Buzzer alarm

This actuator node is meant to emit a loud sound to alert bystanders of any breach in security.

For prototyping, an Arduino Nano board is used. The resulting prototype for this section is shown in Fig. 7.32.

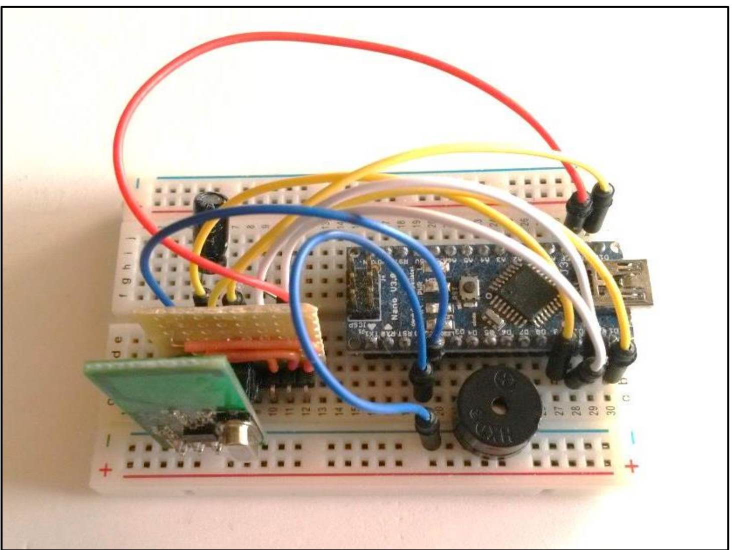


Fig. 7.32 – Buzzer Node prototype

7.5.1. Wireless communications protocol

It uses the RF24.h and RF24Network.h classes as stated in section 7.2.1, page 13. It only listens to incoming messages. When it receives a message, it first checks its type. If it is of any type other than ‘F’, the message is discarded, since according to the communications protocol, only ‘F’ messages should arrive at the Buzzer Node. Otherwise, it proceeds to activate the buzzer sequence.

7.5.2. Buzzer

Fig. 7.33 – Two types of piezo buzzer.

The buzzer is a component used to produce a loud noise with a fixed frequency. Two kinds of buzzer can be used for this purpose.

The electronic buzzer (small cylinder in Fig. 7.33) is the simplest to use. Inside, it has a piezo plus a resonator circuit.

The piezo is a component that vibrates when AC current is passed through it and emits sound with the frequency of the AC signal.

Piezos can be used by themselves, particularly when the developer wants to use them to play in more than one frequency to play melodies (see section 7.6.4, page 46 for an example) but they require the microcontroller to provide an oscillating signal (a square wave, for example). Electronic buzzers include a resonator circuit, which is a small circuit that can turn DC power into a single frequency AC signal which is used to excite the piezo, making the electronic buzzer very simple to use.

On the other hand, an electromechanical buzzer (rectangular enclosure in Fig. 7.33) has been used for the same purpose in the Beta version of the Buzzer Node (see Annex D) . It consists of a rapid switching electromagnet that attracts and releases a diaphragm which produces sound [**12**]. It requires a considerably bigger amount of current, so a transistor has been connected to drive it. The pN2222A has a low price and size, and is well rated for this purpose.

7.6. Node #4: The Key Tray Node

This module's main objective is providing the user with a way to activate and deactivate the alarm from the entrance of the building.

The Central Node is supposed to be in the center of the building for security purposes (it should be difficult to access by thieves) and to provide better range to all the other nodes. This is why a different node must be placed at the entrance. With it, the alarm can be deactivated without triggering any sensors.

The Key Tray Node provides two ways to activate or deactivate the alarm:

- A numerical keypad
- An RFID reader able to detect when the user's keys are placed on top of the node thanks to an RFID keychain. That is why this node, in its hypothetical final product conception, is shaped like a key tray.

For prototyping, an Arduino MEGA board is used. The resulting device is shown in Fig. 7.34.

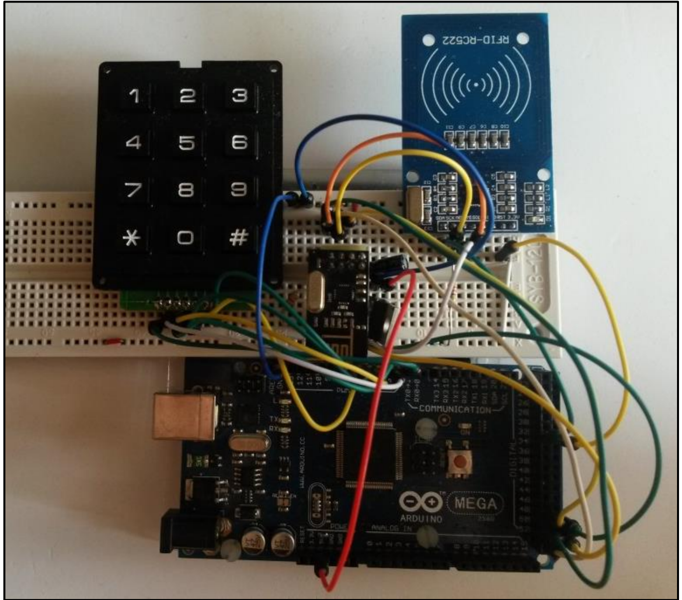


Fig. 7.34 – Key tray node prototype.

7.6.1. Numpad module

The first step involved is the setup of a number pad, or numpad for short. This is an array of labeled pushbuttons in a convenient package, very typical in conventional telephones. It is usually sold in a 3x4 array, which contains numbers 0 through 9, an asterisk and a number sign. It also found in a 4x4 array, with the additional column being letters A to D.

Fig. 7.35 – Different versions of number pads.

Numpads have no active components inside, only conductive traces. When a key is pressed, two pins corresponding to its row and column are shorted. By pulsing every row and reading every column (or vice versa), a microcontroller can find out if a key has been pressed [**29, p.163-165**]. This is accomplished with the library Keypad, by Mark Stanley and Alexander Brevig (distributed under the GNU Lesser General Public License) [**10**, **11**].

While this library takes care of the pulsing routine and debouncing, it is up to the engineer to map the pinout of the particular brand of keypad in his hands. To do so, a multimeter can be used to meticulously probe each pair of pins while pressing each button. However, it is important to use the multimeter in Ohm mode. Using it in continuity mode will not always work because some internal keypad traces have bridges with non-negligible resistance (100-400 Ω). This could make the engineer erroneously think that the keypad is defective.

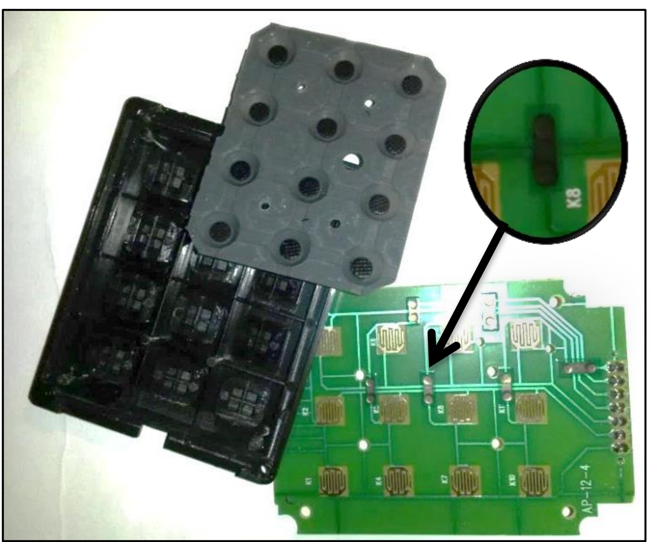


Fig. 7.36 - Internal circuitry of a number pad, with detail of a bridge where a trace has to jump over two other traces.

7.6.2. RFID module

RFID or Radio Frequency IDentification is a technology used for contactless automatic identification and data capture. More information is available in Annex A.4.

For the project in question, the smart alarm, this technology can let the user deactivate the alarm without entering a passcode. For this, the hardware needed is a set of a reader and a tag (or multiple tags, ideally).

The tag

As for the tags, two types have been tested: a small keychain (Fig. 7.38) and a blank card. Both work identically and have distinct Unique Identifications (UIDs).

The UID is a 4-byte code, and is usually presented in a hexadecimal codification such as 41 D9 1E C5. Each RFID tag contains an integrated chip with its UID number burned-in during

the manufacturing process. While an RFID tag can contain much more data, only its UID will be used verification in this project.

Some tags have a 7-byte UID. The alarm is configured for 4-byte long UIDs only. The last three bytes would then be ignored as per design.

No modification is needed on the tags, and users might find that their work or gym cards work as well as any tag included in the alarm.

The reader

A common RFID reader for Arduino projects is the RC-522 (Fig. 7.37). It uses the SPI communications protocol and has a range of about 5 cm, which is enough for detecting a keychain placed on a key tray.

The software used to control the reader is the MFRC522 library [**7**]. It is based on code by Dr. Leong, has been created by Miguel Balboa, rewritten and translated by Søren Thing Andersen and extended by Tom Clement. It has been released into the public domain.

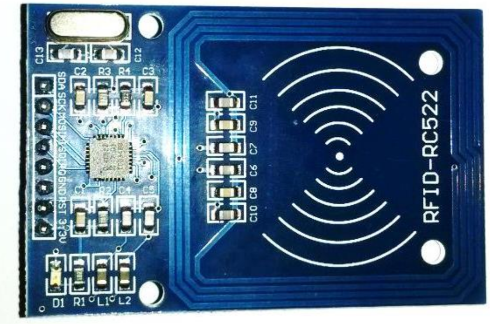


Fig. 7.37 – RC522 RFID reader.

Fig. 7.38 – RFID tag, in a convenient keychain package.

Some major setbacks were encountered when trying to make two SPI devices work side by side. They are explained in Annex B.3.1.

In addition, sending a passcode through RF to the Central Node was also challenging because the passcode is an array, and only the pointer to the array was being transmitted. The solution to this is explained in Annex C.4.

7.6.3. Key Tray Node's finite state machine

Much like the backend of the Central Node, this node also uses a finite state machine to advance between different states. It is pictured in Fig. 7.39.

There are two LEDs of different colors (green and red) and a piezo to provide feedback to the user about the current state of the FSM and many different events.

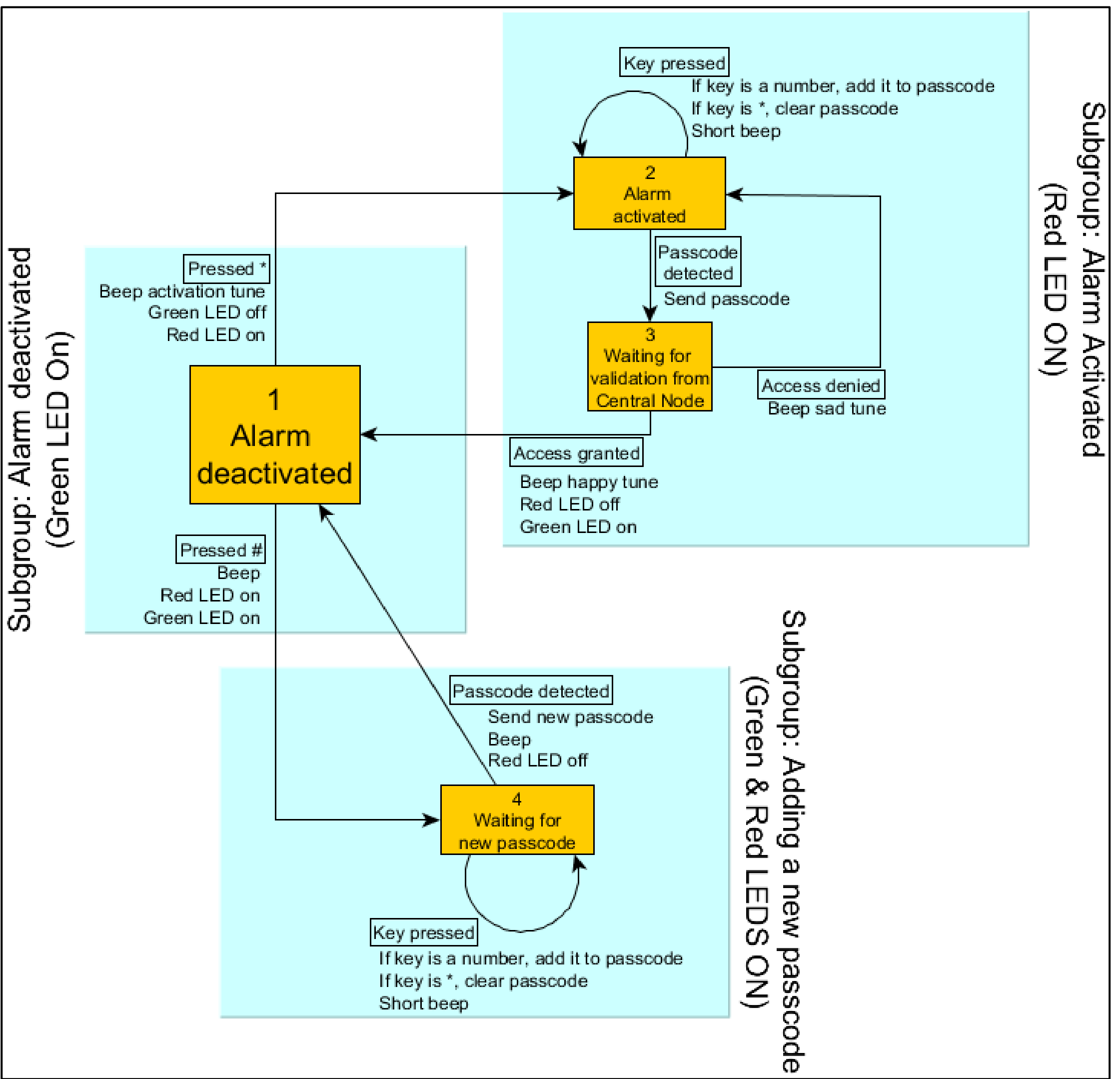


Fig. 7.39 - Finite state machine governing the Key Tray Node behavior.Each transition has an action that triggers it (inside a rectangle) and a list of actions to be done on that transition.

On startup, the system sets itself at state 0, or alarm deactivated. If the user presses the asterisk on the numpad, the FSM will transition to state 2 (alarm activated), provided that it successfully sends notice to the Central Node.

While in state 2, the node listens to button presses for numbers 0-9. If a number is pressed, it is added to a buffer passcode, and a counter is increased.[2] When the counter reaches six, the passcode is full. If the user presses '#' before entering six digits, the buffer will be emptied and the counter will go back to 0. Additionally, a full passcode can be automatically entered by placing an RFID tag close to the reader. Whichever input method the user selects, the full passcode is automatically sent to the Central Node for verification. The FSM then enters state 3.

While in state 3, the alarm is still activated, but the Key Tray Node is waiting for a message from the Central Node stating whether or not the previously sent passcode was found in its database. If it was found, the Central Node has deactivated the alarm and the FSM can go back to state 1. If it was an invalid passcode, the FSM will go to state 2 again since the alarm is still activated. This process happens very quickly, since it requires no user input and wireless messages are transmitted almost instantaneously.

While the alarm is deactivated (state 1), the user can also press '#' to access state 4. In this state, the user can enter a passcode through the numpad or an RFID tag just as in state 2, but this time it will be sent to the Central Node to be added to its database. If the message is sent successfully, the FSM goes back to state 1. No feedback has been implemented as of yet to inform on whether the passcode was really added after the transmission (the database may have been full or the passcode might have been repeated).

This node is neither a sensor nor an actuator, but a combination of both. It needs feedback from the Central Node so that Central Node and Key Tray Node are always synchronized on whether or not the alarm is activated.

7.6.4. Interaction feedback

The Key Tray Node has a green LED and a red LED. These LEDs indicate the current state of the FSM, and the statements that turn them on and off are found exclusively in transitions. Each subgroup of states in Fig. 7.39 has an LED combination associated:

- Alarm deactivated (state 1): Green LED is on.
- Alarm activated (states 2 or 3): Red LED is on.
- Adding new passcode (state 4): Both LEDs are on.

Additionally, there is also a piezo present on the node. It is used to play different sounds and melodies on many events. Some are very brief, like the button press sound. Others consist of a melody with a few notes, like the 'Access granted' sound. The following are events that have a sound associated:

[2] This is not in accordance with the strict definition of a finite state machine. Each time a number is entered, a new state should be reached (such as 'Waiting for third digit' state). In this case, a digit counter is used because it reduces greatly the size and complexity of the FSM.

- When a key is pressed (only if it has a function in the current mode).
- When an RFID tag is detected while in state 2.
- When a messages arrives granting access.
- When a message arrives denying access.
- When the alarm is activated.
- When adding a new passcode.
- When a message broadcast is attempted after the previous one failed.
- When the Central Node is unresponsive.

The code that plays these tunes is inside a custom library called Sounds.h. It uses the core Arduino function tone(), which can provide a square wave of the desired frequency on any pin for a set period of time. It is asynchronous, so once it has been called, the code continues execution before the sound has finished. In other words, it is used to play a note [**5**].

A function called playMelody() has been created to play each node from an array with the notes, and array with the duration of each note. The notes, which represent the frequency of the square signal, are encoded in constant variables in a file called Pitches.h.

For each event with a sound associated, a function is created which declares these arrays, fills them with the correspondent melody associated with the event, and uses playMelody() to play it. There are exceptions, since some events require a single beep that can be accomplished directly calling tone ().

The Sounds.h library's methods are static, so it is not necessary to create a Sounds instance to call them (see Annex C.2).

7.7. Node #5: The Window Node

This sensor node acts as a detector for any door or window being opened. If it detects any, it sends an alert to the Central Node, which is configured to trigger the Buzzer Node.

It has been directly prototyped using an ATmega328p microcontroller, as explained in Annex D, but using a breadboard instead of a perfboard. The resulting prototype is seen in Fig. 7.40.

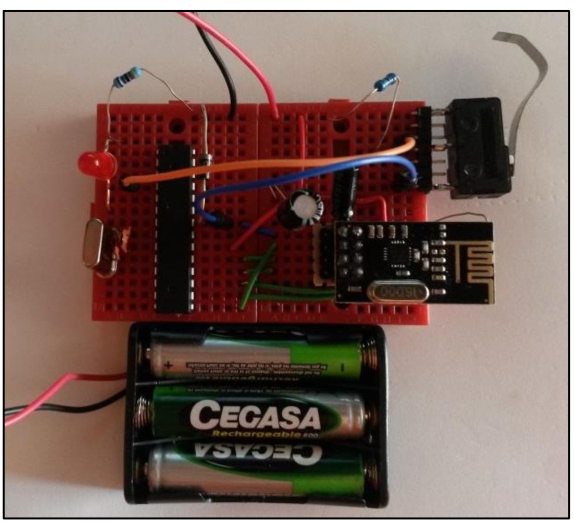


Fig. 7.40 – Window Node prototype.

This node recycles much of the hardware design of the Movement Detector Node. It is a sensor which is activated when a door or window is opened and briefly touches a momentary switch with a long arm. Of course, the node must be placed very close to the edge of the door or window.

Both the PIR sensor and a momentary switch connected to V_{cc} produce the same output when activated: a high signal. The main difference is that a momentary switch, while open, consumes no current. That is why it is interesting to run this node from batteries.

As is common with battery powered devices, a diode is placed to prevent the user from damaging the circuit by placing batteries in the reverse position. Convenient filters are also added.

An LED provides a visual cue when batteries are inserted and when the switch is pressed.

7.7.1. Running from batteries

At software level, two new concepts have been introduced: sleeping and interrupts. When used together, these techniques greatly reduce energy consumption and enable the node to run on batteries for a very long period of time.

Sleeping on the lowest consumption mode pauses the execution of code permanently and reduces the current required enormously (From between 10 and 20 mA when active to less than 0.1 µA when asleep). More on sleeping can be found in Annex A.5.

It is important to note that the nRF24l01+ will continue consuming a small amount of power. While not implemented, there is a way to put it to sleep in parallel with the ATmega328p and wake it up only when it has to send a message [**13**]. This is possible because this node is a sensor node and does not need to receive any incoming messages.

On this mode, the only practical way of waking up the microcontroller is with the use of interrupts (see Annex A.6). To wake up the microcontroller, an interrupt is configured to trigger when the switch is pressed. It then runs the code that wakes up the microcontroller.

Therefore, the microcontroller follows this routine:

1. Start when the batteries are inserted.
2. Go to sleep to save energy (code execution is paused indefinitely).
3. When the switch is triggered, run an interrupt that wakes up the microcontroller so that it resumes execution of the code.
4. Send an alert message to the Central Node and flash the LED.
5. Go back to 2.

7.8. Developing new nodes

Due to the intended customizable aspect of the alarm, a client could be offered the possibility to request new types of nodes. Instead of developing them from scratch, a *barebones* node can be prepared in advance with only the components and software common to all nodes (Fig. 7.41). Afterwards, new nodes can be developed on top of it.

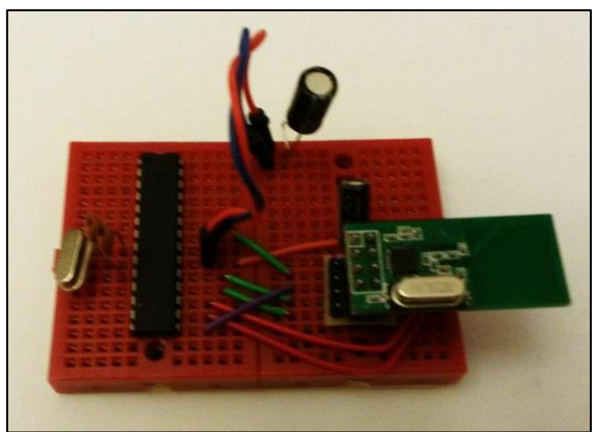


Fig. 7.41 - Base for the development of new nodes.

The basic design is the one used for the Window Node, but just incorporating a handful of components:

- ATmega328p microcontroller as the brain.
- 16 MHz crystal plus its filters (two 22 pF capacitors), for the microcontroller's clock.
- nRF24l01+ radio plus its filter, for communications.
- 220 μF capacitor as power filter.

Afterwards, other sensors or actuators can be added to the node depending on its mission.

Additionally, a power source is needed. The microcontroller and the radio can be put to sleep, so it depends on any other additional components whether the node can be powered from batteries or needs to be plugged into an outlet.

7.9. Complete security system

Fig. 7.42 presents the complete set of prototypes. Both the individual nodes and the whole security system work as intended. Communications are reliable at typical distances of approximately 10 m between the Central Node and any other node.

A total of 2,902 lines of C++ code have been written for this project (external libraries are not included in the count).

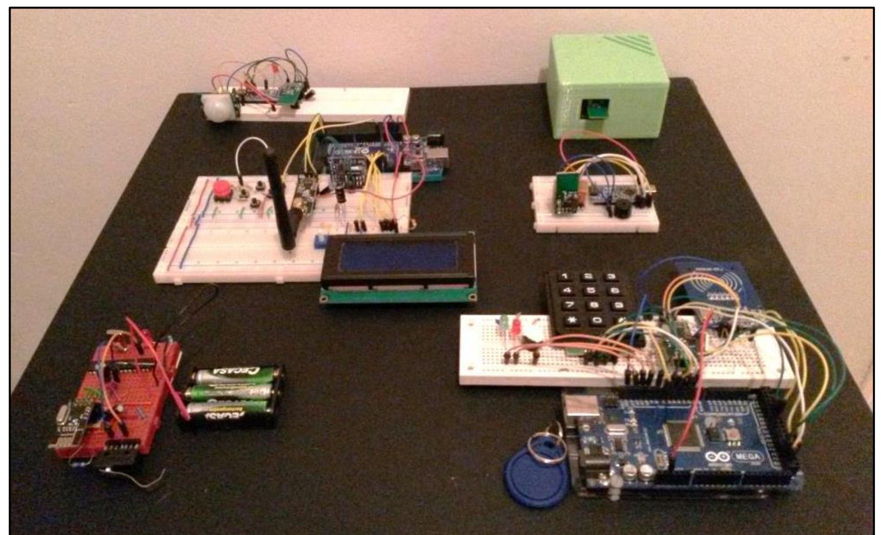


Fig. 7.42 - Set of five nodes in Alpha version, plus the Buzzer Node in Beta version (upper right).

8. Financial analysis

8.1. Components

The bill of materials (BOM) for the electronics components can be found in Annex I.1.

In the BOM, a list of each component used in the prototypes is presented, along with its price. Many components are bought in packs, from which their individual price is calculated. The size of the packs has been selected with the intention of fabricating just one prototype of the alarm system while still having spares for components that could be damaged.

All electrical components have been bought online on eBay and DealExtreme and come mostly from Hong Kong via free international shipping. This is the most economical way to buy components, albeit a very slow one (shipping usually takes a month). Due to time constraints, some components have been bought at the Diotronic and Onda Radio electrical hardware stores in Barcelona at a slightly higher cost than their Asian counterparts.

The total cost of a set with the five different nodes in Alpha version is **107.60 €**.

The transition from Alpha to Beta versions has only been done for the Buzzer Node. In this case, the price has remained approximately the same since the cost of the new components has been compensated by the removal of the Arduino board and the breadboard, which are the most expensive components. By extrapolating this relationship to the rest of the nodes, it could be assumed that an entire set of Beta version nodes would also cost around 108 €.

8.2. Tools

The BOM for the tools used along the development process can be found in Annex I.2. They are the minimal number of tools needed to fabricate the prototypes.

General purpose tools have been bought at generic hardware stores, and 3D printing related ones at RepRapBCN (later renamed as BCN3D Technologies).

The total cost of all tools is **1855 €**.

Obviously, most of these tools will last far longer than this single project. A 3D printer or a PC will become obsolete in 4 - 8 years, while a soldering iron or pliers can last decades. To compute the variable cost from tools usage which should be added to the product price, a certain formula would have to be applied to each tool (Eq. 1).

$$Variable\ cost\ associated\ to\ tools\ usage\ =$$

$$= \frac{Cost\ of\ the\ tool}{Number\ of\ uses} = \frac{Cost\ of\ the\ tool}{\left(Number\ of\ uses/_{day}\right) \cdot Life\ expectancy\ in\ days}$$

(Eq. 1)

8.3. Software

The BOM for the used software is presented in Annex I.3.

It contains the programs used through the development. Their price does not count towards the total, since commercial options can be substituted with free alternatives that can do the task at hand in most cases with satisfactory results.

8.4. Design and manufacture costs

The whole project has been made by one person under the following approximate time constraints:

- Period 1: 3 months, 4 days/week, 4 h/day. Equals 192 h.
- Period 2: 2 months, 6 days/week, 8 h/day. Equals 384 h.
- Period 3: 1 month, 7 days/week, 10 h/day. Equals 280 h.

Total is 856 h. At a normal rate of 8 h/day with no extra hours, and a salary of 15 €/h, it amounts to **12,840 €**.

8.5. Total costs

Therefore, the total cost for each of the investments is as shown in Tab. 8.1.

These numbers mostly show the investment required to prepare the Alpha version of the security system.

As is typical with electronics products, the actual cost of the materials is low, whereas the design and the infrastructure needed for manufacturing have the biggest cost. The worker or workers' wages, in this case, are responsible for an 87% of the investment.

Further development into Beta and Final versions would incur in additional fixed costs in order to reduce variable costs (the cost of the components). Fortunately, lots of spare components would still be available from the Alpha phase and the tools that were already

acquired would be perfectly usable. On the other hand, the time needed to create the designs would run in the hundreds or thousands of hours.

Concept	Cost (€)	Notes
Components (Only Alpha versions)	107.60	Subsequent Beta and Final versions would maintain and reduce the cost, respectively.
Tools	1855	Obviously, the investment would be useful for more production than just a prototype.
Software	0	Most software used is free, and there are free alternatives for the rest.
Human resources	12,840	-
Total	**14,802.60**	

Tab. 8.1 - Summary of project costs.

9. Possible future improvements

As said in section 5.2, page 9, the security system developed for this project is mostly an alpha version. Future improvements could include:

- Running the Movement Detector Node and the Buzzer Node from batteries. This would be possible since the PIR sensor consumes very little current and the nRF24l01+ radio has sleep capabilities.

- Battery level monitoring and notification of low battery for battery powered nodes. A cable can be connected from the battery to one of the Arduino's analog pins. If the voltage were to drop below a certain threshold, the node would send a message to the Central node, which would display a warning.

- Adding encryption in wireless communications. As of now, unencrypted passcodes are being sent over the air from the Key Tray Node to the Central Node. Any person with sufficient technical knowledge could capture this data.

- Making each node independently addressable. Right now, a message intended for a Buzzer Node is received by any and all Buzzer Nodes connected to the network.

- Ability to seamlessly add new independently addressable nodes to the network.

- Ability to change channel to avoid interference with other 2.4 MHz systems.

- Use the RF24Mesh library to increase the maximum number of nodes a network can include [**34**].

- Design new nodes:
    - Internet Node: Would provide a gateway to remotely interact with the security system's behavior and monitor events. This can be accomplished with the ENC28J60 Ethernet module or an ESP8266 WIFI module, for example (Fig. 9.1).

    - Relay Node: A device with a relay and a power outlet, capable of providing or cutting power to whichever appliance is connected to it. Ideal uses would be connecting a lamp to simulate that there is a person at home, or an IP camera preconfigured to stream video. A basic version of this node could make use of the component in Fig. 9.2 and the design in Fig. 9.3, which has two controllable AC outlets.

- Replace licensed software that cannot be commercially distributed (most of the external libraries).

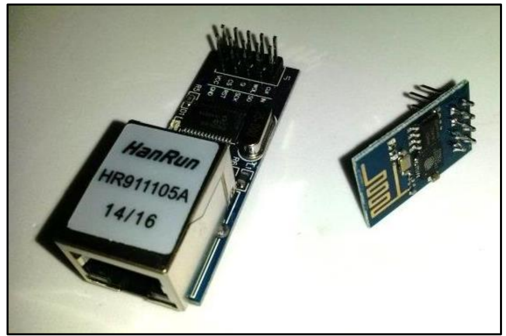


Fig. 9.1 – ENC28J60 Ethernet module (left) and ESP8266 WIFI module (right).

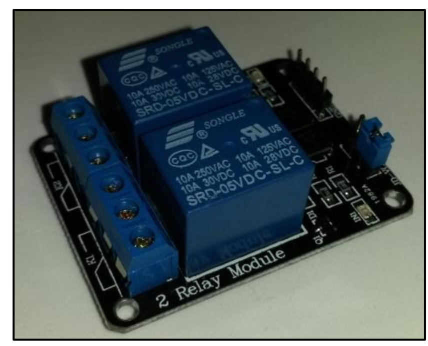


Fig. 9.2 – Relay module with two independently addressable relays.

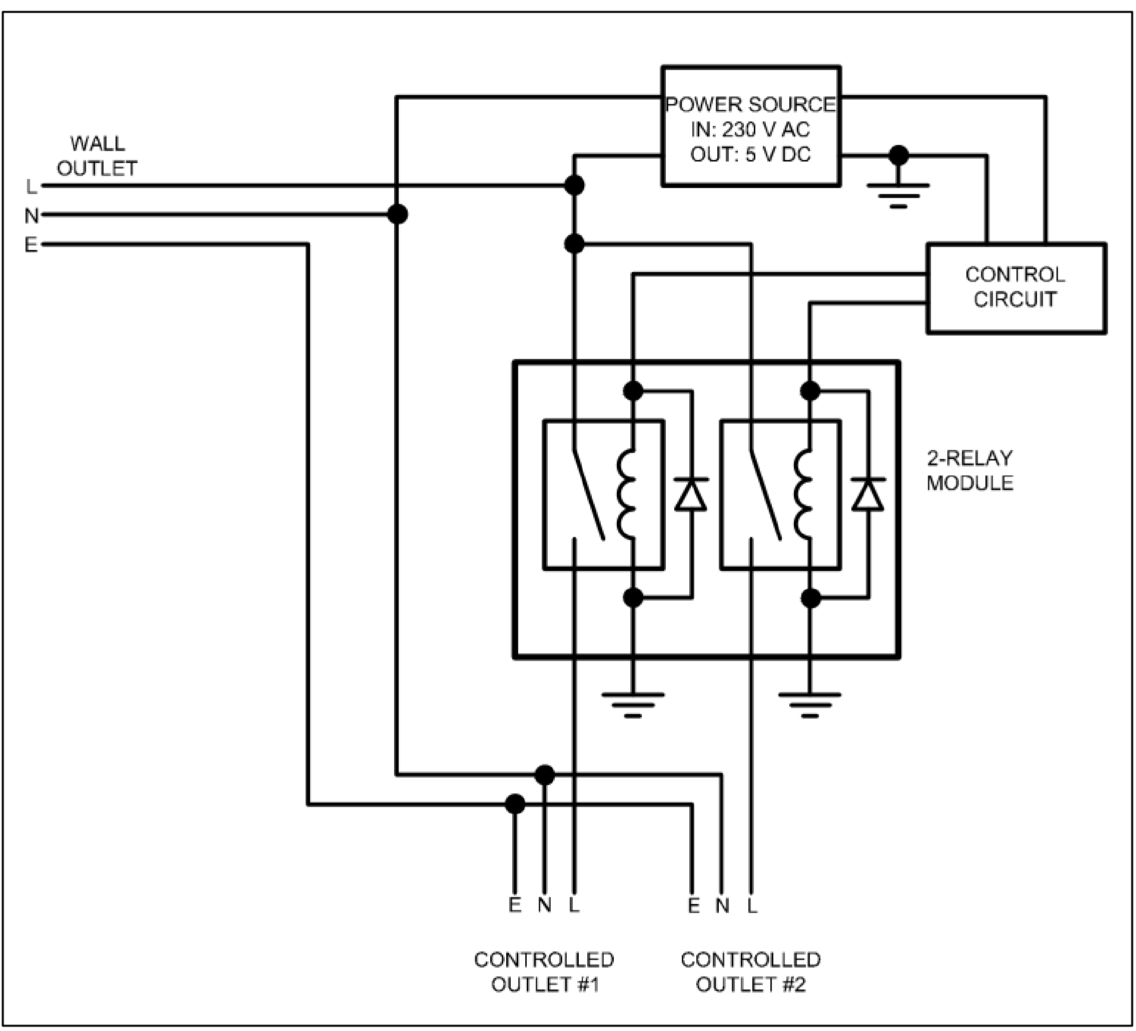


Fig. 9.3 – Electrical diagram of a hypothetical Relay Node.

10. Potential environmental affectations and measures taken to reduce impact

10.1. Fabrication

During fabrication, the following aspects are considered:

- Working with electronics means that few pieces go to waste. While it is difficult to buy the exact number of components needed, the spares are always stored with the perspective that they will be useful in the future. Of course, this is possible since they do not require much storing space.
- The soldering process requires the use of tin-lead alloy wire. The lead component causes toxic fumes to be released during soldering. This does not constitute a considerable air pollutant on a global level, but can be the cause of health problems for the worker. Therefore, all soldering work is done on an open room with forced air circulation. Protective glasses are also required. Lead free wire is available but very inconvenient for hand soldering.
- The use of an additive technology, 3D printing, means that no material is wasted on the enclosure. This is an advantage over subtractive technologies.

10.2. Use

The use of the alarm does not necessarily have any potential affectation, as long as the user is environmentally responsible when disposing of depleted batteries. The correct behavior would be using rechargeable ones on battery powered nodes, so it could be advised in a hypothetical user manual.

It has not been implemented in all nodes, but putting the microcontroller to sleep can reduce energy consumption considerably in the long run.

10.3. Disposal

Components involved follow the Restriction of Hazardous Substances Directive (RoHS), adopted in February 2003 by the European Union. All electronic products entering the market after July 1, 2006 must pass RoHS compliance in order to be sold within the EU.

RoHS specifies maximum levels for the following six restricted materials [**31**]:

- Lead (Pb): < 1000 ppm
- Mercury (Hg): < 100 ppm
- Cadmium (Cd): < 100 ppm
- Hexavalent Chromium: (Cr VI) < 1000 ppm
- Polybrominated Biphenyls (PBB): < 1000 ppm
- Polybrominated Diphenyl Ethers (PBDE): < 1000 ppm

RoHS compliance is specified in the datasheet of each component, although the title of the product on the retail store usually indicates whether or not it follows the directive.

There is an element in the design that contains lead: the solder wire, whose chemical formula is Sn60PBCu2. For the Beta version (components soldered in perfboard) of any of the nodes to be legally sold, it must have less than 1000 ppm of lead. Thankfully, the required connections between components can be made with retention wire instead of solder paths (Fig. 10.1 and Fig. 10.2). This greatly reduces the amount of solder, and therefore, lead used.

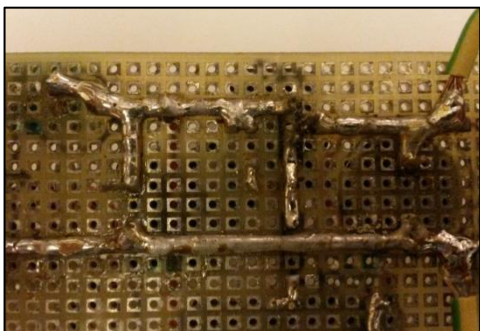


Fig. 10.1 – Solder paths made with solder. This practice is not recommended.

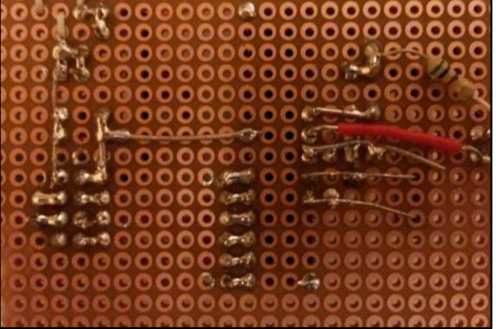


Fig. 10.2 – Solder paths made with retention wire and the component bent legs.

11. Results and conclusions

The tangible result of this project is an entire set of five nodes in Alpha version (Fig. 7.42). Even though it is just a prototype, it works well and can be used as a functional security system. A lot of work has been done to eliminate any bug that could cause frustration for the user, mostly by making wireless communications very reliable and by creating an easily navigable user interface with an LCD screen and buttons on the Central Node.

Additional work has been done to design and manufacture a Beta version of the Buzzer Node, which consists of an electronic circuit soldered on a perfboard and housed inside a 3D printed enclosure. This version closely simulates what a commercial product would aim to offer.

The idea of a customizable security system is realized on two fronts: on one hand, each node can be modified to accommodate user needs, both in hardware and in software.

On the other hand, new types of nodes can be created and seamlessly added to the network thanks to the fact that any new nodes can be developed on top of a previously made platform.

The development of a project of this type and magnitude could not be made without prior knowledge in many different areas: electronics, soldering, C++, 3D printing, component supplying… Each has its own learning curve.

Having a good foundation allows for progressing further and learning faster since one can turn big ideas into working prototypes without being stopped by beginner's problems. The following are aspects which were successfully studied and became invaluable skills:

- C++ Pointers.
- Object oriented programming, C++ specific syntax for writing OOP libraries.
- Turning complex flowcharts into easy to understand finite state machines, and coding them.
- Optimizing the number of used I/O pins in a microcontroller by using analog inputs instead of digital ones.
- Creating useful and easy to use user interfaces with buttons, screens, LEDs, piezos…
- Setting up radio networks.
- Using microcontrollers to their full potential: sleeping to save energy, interrupts to work outside the program flow, EEPROM to save data permanently...
- Successfully transitioning a circuit from a breadboard to a perfboard.
- Designing enclosures for electronics and manufacturing them with a 3D printer.
- The importance of knowing where and when to buy electronics components.
- Extensive knowledge on how to document all work done.

12. Bibliography

1. 3DPRINTING. *What is 3D printing?* [http://3dprinting.com/what-is-3d-printing, accessed May 28, 2015].

2. ARDUINO. *EEPROM Library.* [http://www.arduino.cc/en/pmwiki.php?n=Reference/EEPROM, accessed May 21, 2015].

3. ARDUINO. *Frequently Asked Questions.* [http://arduino.cc/en/Main/FAQ, accessed November 17, 2014].

4. ARDUINO. *Printf.* [http://playground.arduino.cc/Main/Printf, accessed May 19, 2015].

5. ARDUINO. *tone().* [http://www.arduino.cc/en/Reference/Tone, accessed May 8, 2015].

6. ARDUINOBASICS. *PIR Sensor (Part 2).* December 20, 2013. [http://arduinobasics.blogspot.com.es/2013/12/pir-sensor-part-2.html, accessed May 24, 2015].

7. BALBOA, M. *Arduino RFID Library for MFRC522.* January, 2012. [https://github.com/miguelbalboa/rfid, accessed April 30, 2015].

8. BANZI, M. *TED Talks: How Arduino is open-sourcing imagination.* June 27, 2012 [Video, https://www.youtube.com/watch?v=UoBUXOOdLXY, accessed November 17, 2014].

9. BOXALL, J. *Discovering Arduino's internal EEPROM lifespan.* May 11, 2011. [http://tronixstuff.com/2011/05/11/discovering-arduinos-internal-eeprom-lifespan/, accessed May 19, 2015].

10. BOXALL, J. *Arduino Tutorials - Chapter 42 - Numeric Keypads.* December 16, 2013. [http://tronixstuff.com/2013/12/16/arduino-tutorials-chapter-42-numeric-keypads/, accessed April 26, 2015].

11. BREVIG. A. *Keypad Library for Arduino.* December 12, 2014. [http://playground.arduino.cc/Code/Keypad, accessed April 26, 2015].

12. BRIGHT HUB ENGINEERING. *Make Yourself a Simple Homemade Electronic Buzzer.* September 25, 2013. [http://www.brighthubengineering.com/consumer-appliances-electronics/68808-make-yourself-a-simple-homemade-electronic-buzzer/, accessed May 29, 2015].

13. COLIZ, J. *Low-Power Wireless Sensor Node.* October 19, 2011. [https://maniacbug.wordpress.com/2011/10/19/sensor-node/, accessed April 18, 2015].

14. COLIZ, J. *Getting Started with nRF24L01+ on Arduino.* November 2, 2011. [https://maniacbug.wordpress.com/2011/11/02/getting-started-rf24/, accessed April 18, 2015].

15. COLIZ, J. *RF24Network for Wireless Sensor Networking.* March 30, 2012. [https://maniacbug.wordpress.com/2012/03/30/rf24network/, accessed April 18, 2015].

16. COLIZ, J. *Driver for nRF24L01(+) 2.4GHz Wireless Transceiver.* [http://maniacbug.github.io/RF24/index.html, accessed April 18, 2015].

17. COLIZ, J. *Network Layer for RF24 Radios.* [http://maniacbug.github.io/RF24Network/index.html, accessed April 18, 2015].

18. CPLUSPLUS. *Printf.* [http://www.cplusplus.com/reference/cstdio/printf/, accessed May 6, 2015].

19. CPLUSPLUS. *Variables and types.* [http://www.cplusplus.com/doc/tutorial/variables/, accessed April 20, 2015].

20. DANIELS, S. *Advanced Arduino – Including Multiple Libraries in Your Projects. Provide Your Own…* August 9, 2011. [http://provideyourown.com/2011/advanced-arduino-including-multiple-libraries/, accessed April 17, 2015].

21. DORKBOTPDX. *Better SPI Bus Design in 3 Steps.* November 24, 2014. [http://www.dorkbotpdx.org/blog/paul/better_spi_bus_design_in_3_steps, accessed May 6, 2015].

22. ENGBLAZE. *Hush little microprocessor… AVR and Arduino sleep mode basics.* [http://www.engblaze.com/hush-little-microprocessor-avr-and-arduino-sleep-mode-basics/, accessed May 21, 2015].

23. ENGBLAZE. *We interrupt this program to bring you a tutorial on… Arduino interrupts.* [http://www.engblaze.com/we-interrupt-this-program-to-bring-you-a-tutorial-on-arduino-interrupts/, accessed May 21, 2015].

24. FRIED, L. *Testing a PIR.* January 28, 2014. [https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/testing-a-pir, accessed May 24, 2015].

25. GLOLAB. *How Infrared motion detector components work.* [http://www.glolab.com/pirparts/infrared.html, accessed May 6, 2015].

26. GRUSIN, M., SPARKFUN. *Serial Peripheral Interface (SPI).* [https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi, accessed May 6, 2015].

27. HARIZANOV, M. *nRF24L01+ power consumption footprint.* May 16, 2013 [http://harizanov.com/2013/05/nrf24l01-power-consumption-footprint/, accessed May 5, 2015].

28. KUSHNER, D., IEEE SPECTRUM. *The Making of Arduino.* October 26, 2011. [http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino, accessed November 17, 2014].

29. MARGOLIS, M. *Arduino Cookbook (2nd edition).* United States of America, O'Reilly Media, Inc., 2012, p. 154-155, 163-165 170-172, 363-385.

30. MCCAULEY, M. *VirtualWire.* 2008. [https://www.pjrc.com/teensy/td_libs_VirtualWire.html, accessed November 19, 2014].

31. ROHS GUIDE. *RoHS Compliance Guide: Regulations, 6 Substances, Exemptions, WEEE.* [http://www.rohsguide.com/, accessed May 25, 2015].

32. SEEEDSTUDIO. *433 MHz RF link kit.* December 13, 2012. [http://www.seeedstudio.com/wiki/433Mhz_RF_link_kit, accessed November 19, 2014].

33. TMRH20S. *Arduino: Using the full potential of NRF24L01 radio modules.* [http://tmrh20.blogspot.com.es/2014/03/high-speed-data-transfers-and-wireless.html, accessed May 9, 2015].

34. TMRH20S. *Mesh Networking Layer for RF24 Radios.* [http://tmrh20.github.io/RF24Mesh/, accessed April 18, 2015].

13. Other bibliographic references

ACERA, M. A. *C/C++ Edición revisada y actualizada 2010.* Madrid, Anaya, 2010.

BOMBARDÓ, C., AGUILAR, M., BARAHONA, C. *Technical Writing. A Guide for Effective Communication.* Barcelona, Edicions UPC, 2008.

HOROWITZ, P., HILL, W. *The Art Of Electronics (3rd edition).* Cambridge, Massachusetts. Cambridge University Press, 2015.

STELLMAN, A., GREENE, J. *Head First C# (2nd edition).* United States of America, O'Reilly Media, Inc., 2010.

ETSEIB