Final Project
Industrial Engineering

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

ANNEX A: Technologies used
ANNEX B: Wired data transmission protocols
ANNEX C: Noteworthy C++ procedures
ANNEX D: Transition from breadboard to perfboard
ANNEX E: Enclosure manufacturing
ANNEX F: External libraries
ANNEX G: Custom made libraries
ANNEX H: C++ code
ANNEX I: Bill of materials (BOM)
ANNEX J: Schematics
ANNEX K: CAD blueprints

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

Escola Tècnica Superior d'Enginyeria Industrial de Barcelona

ETSEIB
UPC
Table of contents

Table of contents .......................................................................................................................... 1
List of figures and tables ............................................................................................................... 4

A. Technologies used .................................................................................................................. 7
   A.1. The Arduino platform ......................................................................................................... 7
       A.1.1. Introduction .................................................................................................................. 7
       A.1.2. Debugging with Arduino ............................................................................................. 9
       A.1.3. Creating external libraries .......................................................................................... 10
       A.1.4. The EEPROM ............................................................................................................. 12
   A.2. Bluetooth .......................................................................................................................... 13
   A.3. 433 MHz radiofrequency .................................................................................................. 15
   A.4. RFID ................................................................................................................................ 17
   A.5. Microcontroller sleep ......................................................................................................... 18
   A.6. Interrupts ............................................................................................................................ 19

B. Wired data transmission protocols ......................................................................................... 21
   B.1. Serial ................................................................................................................................ 21
   B.2. I^C ...................................................................................................................................... 21
   B.3. SPI .................................................................................................................................... 22
       B.3.1. Using two SPI devices together .................................................................................. 23

C. Noteworthy C++ procedures .................................................................................................. 26
   C.1. Syntax for creating external libraries .................................................................................. 26
   C.2. Static methods in libraries ................................................................................................. 27
   C.3. Writing a function that returns an array ............................................................................. 27
   C.4. Sending an array over RF .................................................................................................. 28

D. Transition from breadboard to perfboard .............................................................................. 30

E. Enclosure manufacturing ........................................................................................................ 32
   E.1.1. 3D printing ...................................................................................................................... 32
Development of a domestic or industrial alarm system for low volume, highly customizable production

F. External libraries

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F.1</td>
<td>EEPROM.h</td>
</tr>
<tr>
<td>F.2</td>
<td>Printf.h</td>
</tr>
<tr>
<td>F.3</td>
<td>Wire.h</td>
</tr>
<tr>
<td>F.4</td>
<td>RTClib.h</td>
</tr>
<tr>
<td>F.5</td>
<td>SPI.h</td>
</tr>
<tr>
<td>F.6</td>
<td>RF24.h</td>
</tr>
<tr>
<td>F.7</td>
<td>RF24Network.h</td>
</tr>
<tr>
<td>F.8</td>
<td>LiquidCrystal.h</td>
</tr>
<tr>
<td>F.9</td>
<td>MFRC522.h</td>
</tr>
<tr>
<td>F.10</td>
<td>Keypad.h</td>
</tr>
<tr>
<td>F.11</td>
<td>Sleep.h</td>
</tr>
<tr>
<td>F.12</td>
<td>EnableInterrupt.h</td>
</tr>
</tbody>
</table>

G. Custom made libraries

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>G.1</td>
<td>AnalogButton.h</td>
</tr>
<tr>
<td>G.2</td>
<td>AnalogKeyboard.h</td>
</tr>
<tr>
<td>G.3</td>
<td>Communications.h</td>
</tr>
<tr>
<td>G.4</td>
<td>Display.h</td>
</tr>
<tr>
<td>G.5</td>
<td>Settings.h</td>
</tr>
<tr>
<td>G.6</td>
<td>StateMachine.h</td>
</tr>
<tr>
<td>G.7</td>
<td>Sounds.h</td>
</tr>
<tr>
<td>G.8</td>
<td>Pitches.h</td>
</tr>
</tbody>
</table>

H. C++ code

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>

I. Bill of materials (BOM)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I.1</td>
<td>Components</td>
</tr>
<tr>
<td>I.2</td>
<td>Tools</td>
</tr>
<tr>
<td>I.3</td>
<td>Software</td>
</tr>
</tbody>
</table>
J. Schematics ................................................................. 58

K. CAD blueprints ......................................................... 65
List of figures and tables

Fig. A.1 - Arduino Nano board. ................................................................. 8
Fig. A.2 - Arduino UNO board. ................................................................. 8
Fig. A.3 - Arduino MEGA board. ............................................................... 8
Fig. A.4 - Locations in which the compiler will look for included libraries. .......... 12
Fig. A.5 - Various models of the Bluetooth module. .................................... 14
Fig. A.6 - 433 MHz RF receiver (left) and emitter (right) with no antennae soldered. 15
Fig. A.7 - Soldered antenna. ................................................................. 16
Fig. A.8 - RFID tag. ............................................................................. 17
Fig. A.9 - RFID tag inserted inside a cloth item. It allows for quick identification and theft detection. ................................................................. 18
Fig. A.10 - Yellow chip. It is worn by runners in organized races to accurately track their times................................................................. 18
Fig. A.11 - Cards with integrated RFID tags. They speed up financial transactions, provide access control in gyms, provide membership proof in libraries, etc. ........ 18
Fig. A.12 - Classification of interrupts.......................................................... 19
Fig. B.1 - I2C communications between a master and three slaves. .................... 21
Fig. B.2 - SPI connections with one master and two slaves ......................... 22
Fig. B.3 - SPI protocol for sending data from the master to a slave through the Master Out Slave In (MOSI) line. ................................................................. 23
Fig. B.4 - Recommended way to wire multi-slave SPI interfaces........................ 25
Fig. D.1 - Detail of the Arduino UNO. It is a PCB with SMD components soldered....... 30
Fig. D.2 - Perfboard with some through-hole components soldered. .................... 30
Fig. D.3 - Soldering station equipped for perfboard soldering ......................... 30
Fig. D.4 - Equivalence between the Arduino UNO and ATmega328p pins. Image provided by Arduino. ................................................................. 31
Fig. D.5 - Buzzer Node in a breadboard without the Arduino. This was a test for the perfboard version................................................................. 31
Fig. D.6 - Buzzer Node in a perfboard (Beta version). ..................................... 31
Fig. E.1 - BCN3D+ 3D printer................................................................... 32
Fig. E.2 - Screenshot of Solidworks with the enclosure design. ........................................34
Fig. E.3 - Screenshot of Repetier Host with print preview........................................34
Fig. E.4 - Printed Buzzer Node enclosure box.........................................................35
Fig. E.5 - Detail of the Buzzer Node enclosure box..................................................35
Fig. E.6 - Buzzer Node enclosure with electronics placed inside...............................35
Fig. E.7 - Buzzer Node enclosure with cover..........................................................35
Fig. H.1 - Percentage of total number of lines of code in each node. Image generated with the program LocMetrics.........................................................49

Tab. A.1 - EEPROM size in different ICs..................................................................12
Tab. A.2 - Example of EEPROM state. A char, a bool, a long and an int have been stored in addresses 1, 2, 5 and 11, respectively, using the EEPROM.put() function. . 13
Tab. A.3 - Electrical characteristics of the 433 MHz radio. Current for the transmitter represents a peak. .................................................................15
Tab. A.4 - Reliability test inside a building of the 433 MHz radio with antennae already soldered.................................................................16
Tab. B.1 - SPI connections for an Arduino MEGA board with two slaves. ..............24
Tab. B.2 - Maximum currents of the two 3.3 V devices on the board.................24
Tab. I.1 - BOM: Central Node (Alpha version)....................................................50
Tab. I.2 - BOM: Movement Detector Node (Alpha version)..................................51
Tab. I.3 - BOM: Buzzer Node (Alpha version).........................................................51
Tab. I.4 - BOM: Key Tray Node (Alpha version)......................................................52
Tab. I.5 - BOM: Window Node (Alpha version).......................................................53
Tab. I.6 - BOM: Buzzer Node (Beta version)...........................................................54
Tab. I.7 - BOM: Tools for electronics prototyping..................................................55
Tab. I.8 - BOM: Tools for embedded programming.................................................55
Tab. I.9 - BOM: Tools for soldering........................................................................56
Tab. I.10 - BOM: Tools for 3D printing...................................................................56
Tab. I.11 - BOM: Embedded development software...............................................57
Development of a domestic or industrial alarm system for low volume, highly customizable production

Tab. I.12 - BOM: Enclosure manufacturing software.......................................................... 57
Tab. I.13 - BOM: Documentation software. ................................................................. 57

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

The following subsections give an in depth explanation of different technologies that have been used along the project.

A.1. The Arduino platform

The brain of most nodes is an electronics board which can be programmed to monitor and control different inputs and outputs in any manner the developer wants it to. These boards are known as Arduino boards.

A.1.1. Introduction

Before 2005, at the Interaction Design Institute Ivrea (IDII) in Ivrea, Italy, students used expensive BASIC Stamps to study electronics engineering. One of those students, Hernando Barragán, started working on an electronics prototyping platform that was economically accessible for both hobbyists and students thanks to its low cost, ease of use, and user-friendly IDE (which was based on the Processing language’s own IDE, made by another student). This platform was known as Wiring.

Shortly after, Massimo Banzi and other IDII’s members took upon the project with the intention to improve on it. After a successful prototype, they decided to make the project open source following news of the institute’s imminent closure. They gave it the name Arduino [28].

Word of Arduino quickly spread online, even with no marketing campaigns. Nowadays, their factory in Ivrea produces 100-3000 boards per day with a strict quality control. There are also around 150 developers selling variation of the original board (with extra capabilities such as climate monitoring, integrated GPS, etc.). This practice is actively encouraged by the creators in accordance with the open source ideals. Moreover, countless counterfeit boards are being produced in Asian factories with slightly lower quality control and no license.

The Arduino project is composed of three components:

- The hardware: There are many different official Arduino boards, but they all consist of a PCB with:
  - A microcontroller storing an algorithm that governs its behavior.
  - Components for power distribution (power jack, voltage regulators, filters…).
  - Standard I/O pins for connecting external components such as motors, screens, other IC, etc.
  - USB-to-serial adapter chip and USB connector for communicating with a PC.
  - Other components such as LEDs, a quartz crystal and a reset button.

  The most popular and versatile boards are the Arduino Nano (Fig. A.1), UNO (Fig. A.2) and MEGA (Fig. A.3). There are other boards intended for certain uses: Fio facilitates wireless communications, Lilipad can be attached to clothes, Mini and Gemma are very small…

- The software:
Development of a domestic or industrial alarm system for low volume, highly customizable production

- Based on the original Processing IDE, Arduino’s own IDE is simple and intended for small and medium scale projects.
- The language used when coding is C++\(^1\), with a few special functions that encapsulate low level code required for using microcontrollers. C++ can be used when coding libraries.
- The user-written algorithm is compiled by AVRDUDE (AVR Downloader/Uploader) into a hex file. This file is then uploaded to the onboard microcontroller via USB.
- The microcontroller comes from the factory with a special firmware installed (the bootloader) that ensures it understands the compiled code.

- The community: Coders working on the Arduino IDE can make use of many public libraries for their own projects. Usually these are prepared by carefully studying individual components’ datasheets and writing code that encapsulates common functions users might need. Additionally, the number of users of this platform is so big that most basic problems have been solved and published online already, on forums and blogs [8].

---

\(^1\) Many sources state that the language used is C. This is because C++ was designed to be backwards compatible with C.
The Arduino hardware relies on the fact that it is easily accessible to beginners and very useful for prototyping. However, there are some inconveniences when using it:

- When developing a commercial project, each component should be chosen to comply with a certain restriction (cost, size, efficiency, battery life...). An Arduino board comes with predefined components which usually offer better specs than are needed, bringing the project's price up unnecessarily. That is why it is useful to make a prototype using Arduino and then recycle its design and strip all the unnecessary components.
- The Arduino bootloader simplifies working with microcontrollers such as ATMEGA328P-PU, but requires a certain amount of IC memory that could be critical for large coding projects.
- The libraries used to control a variety of components encapsulate low level code that, depending on the project, should be looked in detail to find possible optimizations. For example, the bootloader ignores the existence of certain microcontroller mechanisms such as sleeping, which is essential in battery powered applications. The user can, however, use low level code to access these functions.

Arduino can be used in commercial products under certain conditions [3]. An engineer can:

- Embed a full Arduino board inside an appliance.
- Derive a product from the Arduino schematics. It must then follow the Creative Commons Attribution Share-Alike license.
- Use the Arduino bootloader and official libraries. This requires releasing the source code so that clients may reupload the firmware should the bootloader or libraries be updated.
- The Arduino name and logo cannot be used commercially, except when designing shields and board variations. These must be marketed as ‘[…] for Arduino’ or ‘[…] (Arduino-compatible)’.

A.1.2. Debugging with Arduino

When prototyping, it is common for developers to keep the board connected to a computer via USB. On one hand, it usually gives enough power to the board\(^2\) and lets the developer quickly upload new iterations of code. On the other hand it allows for sending data back to the PC and reading it in a console, which is very convenient for debugging. One of the core libraries of Arduino is the Serial library. While it is very commonly used, it has the major drawback of not supporting concatenation.

Concatenation is the practice of joining together two or more strings or data-types which can be converted to strings.

A library called Printf.h, written by J. Coliz, enables the function printf(), which is used frequently in C++ code. It acts just like Arduino’s own Serial.print(), but allows for concatenation of strings, ints, bools... which can greatly reduce the number of lines of code needed for debugging.

\(^2\) The USB 2.0 specification provides up to 500 mA, and the USB 3.0, up to 900 mA.
Input over Serial:

```cpp
Serial.print("First value is ");
Serial.print(x);
Serial.print("and second value is ");
Serial.print(y);
Serial.printf('.');
```

Input over Printf:

```c
printf("First value is %f and second value is %u.", x, y);
```

Output:

First value is 23.21 and second value is 47.

Literal strings intended to be sent over Serial are stored in SRAM along with the variables. This is problematic since SRAM is a critical resource. In order to store them in flash memory (where the compiled program is), the printf() function is used in its _P version, and the literal string is wrapped in the function PSTR() [4, 18]:

```c
printf_P(PSTR("Message"));
printf_P(PSTR("Message %u"), var1);
```

A.1.3. Creating external libraries

Usually, Arduino is used to program relatively simple projects. This makes it possible to write all the code on the main sketch. As a project becomes longer and complex, storing all the code in a single file becomes difficult to understand and maintain. This is why it is possible to divide different functional areas in different files (external libraries).

C++ code can be written in two main program structures: Procedural Programming and Object Oriented Programming (OOP).

- Procedural programming consists of writing a program as a list of instructions to be carried out sequentially. This step-by-step approach is intuitive and easy to apply for small programs. Most old languages are procedural (Fortran, COBOL, C…).

- OOP consists in applying to programming the real life notion that everything is an object. Each object has its own properties and methods, such as a TV that has the properties channel and volume, and methods that change the channel and the volume. The actual code that changes these properties is encapsulated (hidden) and a developer interacts with the TV object through its methods.

In Arduino, OOP external libraries are usually used to control hardware, such as a servo motor. In these cases the developer would declare an instance of the class Servo, and control its position easily with its provided methods without delving in the
low-level code that is actually understood by the servo. The alarm prototype uses many external libraries to control many hardware devices.

Still, a project such as this requires thousands of lines of code. Even if the code that controls hardware is encapsulated in external libraries, a lot of procedural programming code is still present and makes it very inconvenient to develop and maintain. The following distinction has been made:

- **Central Node:** Since it controls the behavior of the whole network, this one is the most complex node. All of its functionality is encapsulated in external libraries, both hardware control and abstract functionality (communications protocol, menu navigation, settings management…). This leaves the main sketch with a series of objects that simply update themselves regularly in the background.
- **Other nodes:** Procedural programming in a single main sketch suffices for the simplest ones. Those with more complexity use a combination of both.

There are some considerations to the creation of custom libraries in the Arduino environment:

- **Main sketch file (.ino):** contains the starting point of the program, setup(), and a function that will be called repeatedly, loop(). This is the basic structure of all Arduino programs.
- **Library header files (.h):** They contain constant variables and class definitions.
- **Library source files (.cpp):** They contain constructors and method declarations (the actual code of the library).

Arduino makes it impossible, as per design, to use included external libraries inside user-created libraries. This is merely a matter of which folders are seen by the compiler at compile time [20]. Sadly, the resulting compile error doesn’t make this explicit. In Fig. A.4, Library2 could be a custom library that uses some functionality from Library3. In order to solve this, there are two options:

  - Both Library2 and Library3 must be moved alongside MyProject inside the same working directory. This defeats the purpose of having a unique directory for libraries, with a single copy of each library, and accessible by any project (a library of libraries).
  - The main sketch, MyProject, must have an #include directive for Library3 even if it does not directly use it. Then, another #include directive must be placed in the actual library that makes use of it, Library2.

The second option is used. Having many scattered copies of the same library along a series of folders of different programs is not viable for this project.
The actual C++ procedures to write an external library are explained in section C.1.

**A.1.4. The EEPROM**

The Arduino’s microcontroller contains a small, permanent memory called EEPROM (Electrically Erasable Programmable Read-Only Memory). Its total size varies between microcontrollers (see Tab. A.1).

<table>
<thead>
<tr>
<th>Board</th>
<th>Microcontroller</th>
<th>EEPROM size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino MEGA</td>
<td>ATmega2560</td>
<td>4096</td>
</tr>
<tr>
<td>Arduino UNO</td>
<td>ATmega328</td>
<td>1024</td>
</tr>
</tbody>
</table>

Tab. A.1 - EEPROM size in different ICs.

Arduino’s EEPROM.h library provides methods to write and read individual bytes, as well as data-types bigger than one byte. The developer must specify the address in which to save or retrieve the data, and at the same time take care not to store data in overlapping positions [2, 19]. Tab. A.2 shows an example of how an EEPROM could be used.
Each time an EEPROM cell is written it undergoes physical wear and tear. At a certain point, the cell wears out and reading it reports an error or a false value. The datasheet for ATmega328 reports a minimum lifespan of 100,000 write cycles for each single cell. This acknowledges situations of extreme environmental conditions, and a simple destructive experiment in normal conditions usually yields a value an order of magnitude higher [9].

The alarm’s code does not need to prevent EEPROM failure since these values are far greater than needed for normal use. The write speed (3.3 ms/byte) is slow but not critical since it is used sporadically.

EEPROM read operations can be used indiscriminately since they do not have such limitation. Moreover, they are also quite fast (2 µs/byte).

### A.2. Bluetooth

Bluetooth is a wireless technology standard for exchanging data between devices over short distances using short-wavelength radio waves in the 2.4 to 2.485 GHz band.

Online retailers usually sell three components capable of establishing a BT link:

- **HC-04**: Can establish a slave connection only. It is a very small component sold without breakout board, making it hard to work with.
- **HC-05**: Can establish a master/slave connection. It is sold with a JY-MCU breakout board, making it easy to plug in breadboards or solder in perfboards.
- **HC-06**: Can establish a slave connection only. It is sold with a JY-MCU breakout board.

Fig. A.5 shows these models.
Development of a domestic or industrial alarm system for low volume, highly customizable production

The three of them operate at 3.3 V and communicate with the Arduino by Serial, making it imperative to connect a voltage divider between Arduino’s TX and BT’s RX. \( V_{cc} \) should come from the 3.3V pin or a similarly regulated power source.

Once two BT devices are connected, they can communicate as they would with a simple cabled Serial connection (in its conception, Bluetooth technology aimed to become a wireless alternative to Serial; see section B.1, page 21, for more information). These devices have an AT mode in which the programmer can change parameters such as its name, pin, baud-rate… by sending certain commands.

They have two major inconveniences for which BT has been discarded:

- They have the limitation of being able to remember only 7 slaves, since they assign 3 bit long addresses \( 2^3 = 8 = 1 \) master + 7 slaves).

- One master can only be connected to one slave at a time. For the alarm system, a master should be able to listen to all slaves simultaneously.
A.3. 433 MHz radiofrequency

These components are sold in pairs of sender and receiver (Fig. A.6), and are very inexpensive (as low as 1.50 € when buying a single pair online).

The transmitter could be used for a sensor and the receiver for an actuator. The Central Node would then use both. Another approach could be using both on every node in order to have communication feedback. They present the electrical parameters shown in Tab. A.3. The transmitter idle current is not important, since it can simply be powered down to save energy until it has to send a message. It has a wide range of possible input voltages. The higher the voltage, the longer the transmission range is. However, the rest of the components used for the entire alarm project are mostly 3.3 V or 5 V tolerant, so it would be difficult to derive more than one voltage level from a single power source.

<table>
<thead>
<tr>
<th>Transmitter</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage (V)</td>
<td>3 - 12</td>
</tr>
<tr>
<td>Current (mA)</td>
<td>40 at 12 V</td>
</tr>
</tbody>
</table>

Tab. A.3 - Electrical characteristics of the 433 MHz radio. Current for the transmitter represents a peak.

Ideally, the DATA pins on both emitter and receiver can be used as a direct cabled-like connection. However, the receiver is very sensitive to external noise around 433 MHz and needs signal processing [32].

Their range is much less than advertised (>500 m), even after soldering an antenna to both emitter and receiver with the following traits:

- Soldered directly on the ANT position in the PCB.
- 22 AWG cable or similar.
- Uncoiled length equal to Wavelength/4, which is 17 cm (Eq. 1).
- Oriented parallel to other antennas due to blind cones.
- Coiled to save space.
The table at Tab. A.4 shows tested reliability values. A good result should be 95% or more in typical use. At 9 V, the 90% reception rate would still be acceptable if it was not for the aforementioned difficulty in providing different voltages to different components of the same board. A 70% reception rate for 5 V is simply not reliable enough.

<table>
<thead>
<tr>
<th>Emitter Voltage (V)</th>
<th>Distance</th>
<th>Obstacles</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10 m</td>
<td>1 wall + 2 doors</td>
<td>70%</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>90%</td>
</tr>
</tbody>
</table>

Tab. A.4 - Reliability test inside a building of the 433 MHz radio with antennae already soldered.

The RCSwitch library encapsulates many protocols for sending codes. The simplest one sends a string of 1s and 0s. If the user has to send a message in the form of a string, such as "Movement detected in area 6", it must first be converted to binary: "01001101011011101101100110...". However, working with strings of characters wastes SRAM. A custom protocol should be designed that sends information in the least possible amount of bits. Shorter messages reduce the probability of a failed broadcast.

The VirtualWire library acts in a similar way, but also implements easy Text-Binary conversion and Server-Client communication for centralized networks [30].

Ultimately, this component is very cheap and simple to use. It also requires no extra hardware and the minimum amount of connections. Sadly, it is very unreliable on 5 V and typical distances inside a building, even with a closed loop system with reception acknowledgement (emitter + receiver on each module). In some cases, waiting for a confirmation would cause a timeout because of the low reliability and high number of retries. At higher voltages it performs better, but this would complicate the power distribution (expensive batteries, more losses, etc.). It would be usable at shorter distances, such as inside a single room, but not for an entire home. This is why it has been discarded.
A.4. RFID

RFID or Radio Frequency IDentification is a technology used for contactless automatic identification and data capture. It is composed of:

- An RFID tag. Typically, it contains a chip storing data, plus a coiled antenna traced on the PCB (Fig. A.8). It can be passive (powered by external inductance, short range) or battery powered (long range). The data stored inside can be read only or read/write.
- A reader, which emits an inductive RF signal that powers on passive tags and reads the data they emit (or edits the data on the tag).

These components can also be referred to as PICC (Proximity Integrated Circuit Card) and PCD (Proximity Coupling Device), respectively.

![RFID tag](image_url)

This system has been implemented in a great variety of scenarios since its conception:

- Access control: A person puts a card close to a surface and is granted access by a machine. The same concept has been applied to this module using an RFID-RC522 reader and tags inside keychains. Some examples are a gym or a workplace (Fig. A.11).
- Time tracking for runners: For races with many runners, it is difficult to keep track of each and every one of them. Manual data entry often results in inaccurate times since runners might have to queue for having their number registered when reaching a control point or the end. The Yellow Chip is a small, personal RFID tag that can be tied to the runners’ shoelaces. In every control point, a reader is mounted on the floor which is capable of detecting every passing chip with a negligible margin of error and in real time (Fig. A.10).
- Financial transactions: Credit cards recently started incorporating RFID tags. They work like they did before, but also allow for the user to merely pass the card close to the reader to begin the transaction (Fig. A.11).
- Theft prevention: Since RFID tags can be miniaturized and hidden in clothes and other items, they offer a higher degree of protection against theft. Long range readers at the shop entrance alert of any RFID tag that leaves the shop without first being registered as sold (Fig. A.9).
Development of a domestic or industrial alarm system for low volume, highly customizable production

- Animal tagging: RFID tags, mounted inside tiny capsules, can be inserted in an animal skin. This is useful to manage livestock in farms or lost pets.

Fig. A.9 - RFID tag inserted inside a cloth item. It allows for quick identification and theft detection.

Fig. A.10 - Yellow chip. It is worn by runners in organized races to accurately track their times.

Fig. A.11 - Cards with integrated RFID tags. They speed up financial transactions, provide access control in gyms, provide membership proof in libraries, etc.

A.5. Microcontroller sleep

Sleeping is a common method to save energy. By deactivating unneeded internal parts of a chip, it will consume a fraction of what it consumes at full power. The Arduino board is not advisable for low energy consumption, since even when its ATmega328p or similar microcontroller is asleep, other components will still consume a lot of energy (the voltage regulator and the Serial to USB converter, mostly). A standalone chip is better suited for low power devices.

The ATmega328p microcontroller has many internal sleep modes which can be activated with methods from Sleep.h, a library included in the Arduino core libraries and released by Atmel, the manufacturer of these chips. Each mode dictates which parts of the chip will deactivate when it goes to sleep. Disabling more parts reduces consumption but also reduces functionality of the chip while sleeping.

For example, there is the Brown Out Detector (BOD), which detects and prevents damage to the chip from being powered by excessively low voltages. Deactivating it means a longer battery life at the cost of this protection.

These are the five sleep modes available on standard 8-bit AVRs [22]:

- SLEEP_MODE_IDLE (least power savings)
- SLEEP_MODE_ADC
- SLEEP_MODE_PWR_SAVE
- SLEEP_MODE_STANDBY
- SLEEP_MODE_PWR_DOWN (most power savings)
At complete power down mode, the consumption is reduced from between 10 and 20 mA to merely 0.1 µA or less. The only way to wake up the chip from this sleep mode is with a hard reset or with an interrupt (section A.6). The interrupt must be enabled before the microcontroller goes to sleep and its ISR must contain the code that wakes up the chip.

A.6. Interrupts

Interrupts are an internal mechanism of many microcontrollers which execute an asynchronous block of code when triggered by certain internal or external events.

When an interrupt is run, the program flow is interrupted and a predefined function is run, which is the interrupt service routine (ISR). After the ISR has ended, the main code execution is resumed.

The events that trigger an interrupt can be either internal or external. Internal ones deal with timers that trigger after a certain amount of time. External ones react to the following events on a pin.

There are two kinds of external interrupts: hardware interrupts and pin change interrupts [23]. Fig. A.12 shows the full classification.

![Classification of interrupts](image)

For the ATmega328p microcontroller, only two pins (INT0 and INT1 on Fig. D.4) are purposefully prepared to use hardware interrupts. These allow the detection of the following events on said pins:

- Being low.
- Rising edges (low to high).
- Falling edges (high to low).
- Either rising or falling edges.
Hardware interrupts grant full monitoring of which event has happened and on which pin.

On the other hand, pin change interrupts can be used on any pin of the microcontroller except power pins, and AREF. This leaves the impressive amount of 23 available interrupt pins. There are two disadvantages, though:

- Only two events can be monitored: rising and falling edges. The interrupt cannot differentiate which one of them happened, so it runs the same ISR on both cases. The developer can, however, keep track of the state of the pins to know whether it was a rising or a falling edge.
- Pins are grouped in groups of seven or eight pins, so when an edge happens on a pin, the microcontroller will only know in which group the event happened, but not which pin. The developer can, once again, keep track of all pins between interrupts or, in other words, take consecutive snapshots and compare them.

With proper code, pin change interrupts can be used with the same versatility that hardware interrupts have. The library EnableInterrupt.h, by Michael Anthony Schwager, provides this functionality. It is released under the Apache License Version 2.0.

To use an interrupt with this library, the developer first has to enable it by attaching a pin, the event associated to that pin, and a user defined function that will be used as the ISR. If the interrupt is no longer needed, it can be easily disabled. The ISR does not allow for debugging since interrupts conflict with the Serial library (see sections A.1.2 B.1).

Interrupts are useful for waking up sleeping microcontrollers, since external events can't be monitored while the main code is paused. By writing an ISR which includes the code that wakes up the microcontroller from sleep, the node can go to sleep indefinitely, wake up when it detects, for example, that a pin goes high, and then continue executing code.
B. Wired data transmission protocols

B.1. Serial

Synchronous serial communication, often referred as Serial or RSR-232, describes a type of wired communication protocol in which data is sent from one device to another in a continuous stream of bits at a constant rate. To achieve this, both devices must have synchronous clocks.

Arduino implements Serial communications with the computer through a Serial to USB converter chip. This is useful for debugging (See section A.1.2). All nodes of the alarm are set at a clock of 57600 baud. Baud stands for bits per second. Additionally, any other device can communicate over Serial with Arduino through pins 0 and 1.

B.2. I²C

The Inter-integrated Circuit (I²C) protocol allows multiple slave devices to communicate with one or more master device. It is intended for short distance communications, and only requires two signal wires plus ground to exchange information.

Unlike Serial, it has a clock line to allow for communication among devices with different internal clocks. Slaves only listen to the data line once for every pulse of the clock line.

In Fig. B.1, the data line is used, first of all, to send an identifier. Each slave has a different identifier, and the one that matches it will start listening. After the slave is selected, the master begins sending it data.

![Fig. B.1 - I²C communications between a master and three slaves.](image)

In the alarm, I²C is used by an external library to control the ds1307 Real Time Clock (see Memory, section 7.3.8).
Development of a domestic or industrial alarm system for low volume, highly customizable production

B.3. SPI

The Serial Peripheral Interface (SPI), developed by Motorola, is used for communications between ICs. It works in a master-slave configuration, with one master and multiple slaves. It uses three different cables plus an additional one for each slave (plus ground).

It is a synchronous protocol, so there must be a clock line. Only one device, the master, can give a clock signal. Multiple slaves can be introduced, and they all share three cables (SCLK, MOSI and MISO). Additionally, no handshake is required (a master will send data even to a nonexistent slave) [26].

- SCLK: The clock line. Only the master can give this signal. The slaves listen to it and read/send bits only when the clock signal rises.
- MOSI: Master Out Slave In. Data line for information going from the master to a slave.
- MISO: Master In Slave Out. Data line for information going from a slave to the master.
- SS: Slave Select. There is one for every slave. It selects which slave is currently communicating and works at negative logic (must pull low to select a slave).

Since there are two data lines, communications can be bidirectional and at the same time. Typically, at every rising edge of the clock signal, the master sends a bit through the MOSI line. Whichever slave has its SS low reads that bit.

Sending data from a slave to the master is not as easy. The master has to know how the slave works and predict when it will have stored data awaiting to be sent (SPI slaves tend to be sensors and their behaviors are explained in their datasheets). Additionally, data is sent with a delay of one clock cycle, since the slave can't predict when the first clock cycle will arrive.

![Fig. B.2 - SPI connections with one master and two slaves](image-url)
SPI is a common protocol implemented in many ICs which come from factory with high level functions that address communications. However, there might be differences between devices that force the engineer to ignore them and write low level code. This is known as bit-banging. One possible difference could be, for example, using the falling edge instead of the rising edge from the clock signal.

Arduino users can implement bit-banging or make use of the SPI.h library, which takes advantage of the SPI hardware built into the microcontroller. The first method lets the user select the SCLK, MOSI and MISO lines, while the second one does not since they are hardwired into the microcontroller.

B.3.1. Using two SPI devices together

A major setback when setting up the hardware for the Key Tray Node was using both the RFID-RC522 and the nRF24l01+ modules in the same board. Both use SPI communications protocol, which usually makes such a scenario easy. However, in order to set it up a careful study of their respective libraries was done in order to find any incompatibilities.

Confusing pinout

First of all, wiring them to the correct pins was complicated because:

- Arduino UNO and Arduino MEGA use different chips with different SPI hardware. Because of this, they use different pins. Most examples for both libraries use the Arduino UNO pinout without warning MEGA users that it will not work on their boards.
- The MFRC522 library documentation is wrong for the Arduino MEGA board. For different SPI devices communicating with the same IC, the Clock, MOSI and MISO lines have to be the same.
- The pin names printed on the PCB vary from one device to the other, leading to confusion.
Development of a domestic or industrial alarm system for low volume, highly customizable production

### RFID module - RC522 -

<table>
<thead>
<tr>
<th>SPI line</th>
<th>Name traced on PCB</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>SCK</td>
<td>52</td>
</tr>
<tr>
<td>MOSI</td>
<td>MOSI</td>
<td>51</td>
</tr>
<tr>
<td>MISO</td>
<td>MISO</td>
<td>50</td>
</tr>
<tr>
<td>Chip Select**</td>
<td>SDA</td>
<td>32</td>
</tr>
<tr>
<td>Reset**</td>
<td>RST</td>
<td>33</td>
</tr>
</tbody>
</table>

### Wireless Comms module - nRF24l01+ -

<table>
<thead>
<tr>
<th>Name traced on PCB*</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCK</td>
<td>52</td>
</tr>
<tr>
<td>MOSI</td>
<td>51</td>
</tr>
<tr>
<td>MISO</td>
<td>50</td>
</tr>
<tr>
<td>CSN</td>
<td>49</td>
</tr>
<tr>
<td>CE</td>
<td>48</td>
</tr>
</tbody>
</table>

Tab. B.1 - SPI connections for an Arduino MEGA board with two slaves. Notice that some lines are shared.

* Not all nRF24l01+ modules have pin names traced on their PCB. They are still available on their datasheets.

** These are not set by hardware and can be changed programmatically.

#### Insufficient power for both devices

Additionally, both devices operate at 3.3 V. The Arduino MEGA can provide this voltage, but not at the needed current to feed both devices. Because of this the RFID reader kept failing a lot.

<table>
<thead>
<tr>
<th>Device</th>
<th>RFID-RC522</th>
<th>nRF24l01+</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. current (mA)</td>
<td>13</td>
<td>27.8</td>
<td>41.8</td>
</tr>
</tbody>
</table>

Tab. B.2 - Maximum currents of the two 3.3 V devices on the board.

The onboard 3.3 V power regulator can theoretically provide 150 mA. In practice, it has failed to do so. This could be due to the fact that it can provide the current but not as fast as needed. However, the 10 μF capacitor already used with the nRF24l01+ module breakout board (see Memory, section 7.2.1) does not solve this problem.

After testing with many other ways to power the devices, a TS1117 3.3 V power regulator was selected. It is fed 5 V from the Arduino MEGA board and provides 3.3 V to the RFID-RC522 and the nRF24l01+. It works well, so the probable cause of the problem is most likely that the 3.3 V regulator on the Arduino was insufficient.

#### Subpar SPI bus design

After solving the previous problems, the modules still acted strangely. During startup, after configuring the nRF24l01+ in software with the desired parameters, it somehow
reconfigured itself with completely different settings (different bandwidth, power settings) and could not communicate with the rest of the network.

Many software tests were tried: studying and modifying the SPI, MFRC522, RF24 and RF24Network libraries; changing the order of creation of instances and their initializing functions; whether the radio and the RFID reader worked together or not…

In the end, after much investigation, the culprit was found to be a bad wiring design. The Chip Select line has to be pulled high. This is usually ignored since it is unnecessary for projects with a single SPI slave. Not complying will result in instructions being sent to both devices at the same time because the Arduino SPI hardware does not do a correct job in pulling the Select line high when the component has to remain inactive [21].

This is why the nRF24l01+ reconfigured itself with a wrong configuration: it was reading and interpreting data meant for the RFID-RC522. After adding a 220 Ω resistor for each device, they both worked well.

The MISO should theoretically be protected with a Tri-state buffer, but apparently the node works well without it.

Fig. B.4 - Recommended way to wire multi-slave SPI interfaces
C. Noteworthy C++ procedures

The following subsections address C++ procedures that have been studied to solve different needs which have arisen during the embedded programming of the different alarm nodes. They differentiate themselves from the rest of the code because of their complexity or use of advanced C++ syntax.

C.1. Syntax for creating external libraries

Writing custom OOP libraries requires a deep knowledge of C++. There are many syntax rules to follow regarding namespaces, public and private fields and methods, declaration of instances and initialization [20]. For this project, custom libraries have been written using the following structure:

**Header file for Example library (Example.h):**

```cpp
#ifndef Example_h
#define Example_h

#include "Example2.h"

class Example {
    public:
        Example();
        int count;
        bool DoSomething();

    private:
        Example2 example2;
        void doSomethingElse();
};

#endif
```

Include guards are instructions to the preprocessor that prevent the library from being included twice in the project.

Another library is included from inside the current one.

The class definition has two sections:
- Public members can be accessed publicly:
  ```cpp
eexample2.DoSomething();
  ```
- Private members are only available from inside the same library.

Each section can have two types of members:
- Fields: variables such as int, bool, Example2, etc.
- Method prototypes, which are defined in the source file.

Moreover, one of these methods must be the constructor Example(), which is called when an instance of the class is created.
Source file for Example library (Example.cpp):

```cpp
#include "Example.h"

Example::Example()
    : example2(arg1, arg2)
{
    // Preparations
}

void Example::DoSomething()
{
    [...]
    doSomethingElse();
}

bool Example::doSomethingElse()
{
    example2.checkSomething(20, 4);
    [...]
    return (example2.checkSomething());
}
```

C.2. Static methods in libraries

When an object’s method does not need any data from the same object, it can be made static. A static method can be called without creating first an instance of its class. To make them static, the developer only has to add the keyword ‘static’ at the function declaration in the headers (.h) file:

```cpp
static bool isAlarmActivated();
```

Then, to call this function, it must be preceded by the namespace and two colons:

```cpp
Settings::isAlarmActivated();
```

This practice is used in the Central Node’s Settigs.h library (see Memory, section 7.3.7), and in the Key Tray Node’s Sounds.h library (see Memory, section 7.6.4).

C.3. Writing a function that returns an array

A new problem encountered was getting a function to return an array, which would be useful for returning a 6-byte passcode when the Display.h class needed to display it on the screen. This is usually not permitted in C++. However, in this language, an array is treated as a pointer to a location in memory which stores the first member.
Development of a domestic or industrial alarm system for low volume, highly customizable production

```c
int array[6]  ≡  int* array
```

Then, when accessing a member, the number inside the brackets is used to perform pointer arithmetic and find the correct location in memory.

```c
array[3]  ≡  *(array + 3*sizeof(int))
```

A static variable is one which is created inside a function the first time it is executed, but is not deleted from memory upon exit. Instead, it keeps its value on posterior executions.

This lets the developer write a function that returns a pointer to a static array created and filled inside the function.

```c
//Function that returns an array (or rather, a pointer to its location in memory)
byte* getStoredPasscode() {
    static byte storedPasscode[6];
    [...]  
    return(storedPasscode);
}
```

```c
byte* passcode = getStoredPasscode(); //passcode can now be treated as an array. 
//Example: cardID[2]
```

The static keyword in front of the variable is necessary for a correct behavior. It prevents the deletion from memory of the local variable when the function exits. Since it exits returning a pointer, this pointer would be useless if the data it points to was deleted.

### C.4. Sending an array over RF

As seen in the previous section, arrays are treated by C++ as pointers to their first member. Using the RF24Network method to send data wirelessly does not work well with arrays because it only sends the pointer, not the actual contents. Sending a pointer to another node is essentially useless.

To force it to send the whole array, it must be declared and passed as a global variable instead of an argument of a function.
Array passed as an argument (does not work):

```c
void sendPasscode(
    char array[LENGTH])
{
    [...] network.write(header, &array, sizeof(array));
    [...] }
}
```

Array passed as a global variable (works successfully):

```c
char array[LENGTH] = {...};
void sendPasscode()
{
    [...] network.write(header, &array, sizeof(array));
    [...] }
}
```

It is still important to say that global variables should be used sparsely to prevent unreadable code and wasted RAM. It is bad practice to have variables be accessible in functions in which they are not used since they can be accidentally modified and they occupy memory which could be freed.
D. Transition from breadboard to perfboard

Any electronics project on a breadboard is convenient to work with because of how easy it is to rewire. However, it cannot be put inside an enclosure and presented as a final product for many reasons: the long cables cause interferences, they might detach themselves, breadboards are relatively expensive...

To solve this, commercial products are usually presented as a Printed Circuit Board (PCB) with surface mount devices (SMD) soldered (Fig. D.1). While PCBs are the most economic option for large scale production, a small series product like the Arduino alarm can be soldered to a perfboard instead.

Perfboards (Fig. D.2) are the antecessor to PCBs in the industry of electronics manufacturing. A typical perfboard consists of a thin, rigid sheet with holes pre-drilled at standard 0.1” intervals across a two-dimension matrix. Each hole is surrounded by a copper pad which does not bridge to neighbor pads. Using tin-based solder, through-hole components can be manually soldered to these pads with relatively inexpensive tools (compared to the machines needed for PCB)(see Fig. D.3).
To test the viability of presenting a perfboard version of the previously developed prototypes, a single node (the Buzzer Node) has been redesigned in such a way.

The first step of the redesign was to get rid of the Arduino board and use only the ATMEGA328 microcontroller. After programming the microcontroller with an Arduino UNO, it can be removed and placed on a breadboard or perfboard. The table in Fig. D.4 is necessary to correctly connect the components. Apart from the microcontroller, there are a few extra components needed: a 16 MHz crystal, and two 22 pF capacitors.

To test the design, it was first laid out on a new breadboard (Fig. D.5). After being tested and deemed correct, it was finally soldered on a perfboard (Fig. D.6).

<table>
<thead>
<tr>
<th>Arduino function</th>
<th>Arduino function</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>(PCINT14/RESET)</td>
</tr>
<tr>
<td>digital pin 0 (RX)</td>
<td>(PCINT19/RXD)</td>
</tr>
<tr>
<td>digital pin 1 (TX)</td>
<td>(PCINT17/TXD)</td>
</tr>
<tr>
<td>digital pin 2</td>
<td>(PCINT18/INT0)</td>
</tr>
<tr>
<td>digital pin 3 (FWM)</td>
<td>(PCINT19/OC2B/INT1)</td>
</tr>
<tr>
<td>digital pin 4</td>
<td>(PCINT20/XCK/70)</td>
</tr>
<tr>
<td>VCC</td>
<td>VCC</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>crystal</td>
<td>(PCINT6/XTAL1/TO56C1)</td>
</tr>
<tr>
<td>digital pin 5 (FWM)</td>
<td>(PCINT21/OC2B/11)</td>
</tr>
<tr>
<td>digital pin 6 (FWM)</td>
<td>(PCINT22/OC2B/11)</td>
</tr>
<tr>
<td>digital pin 7</td>
<td>(PCINT33/AIN1)</td>
</tr>
<tr>
<td>digital pin 8</td>
<td>(PCINT9/OC2B/INT1)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>PC5 (ADC5/SCL/PCINT13)</td>
<td>analog input 5</td>
</tr>
<tr>
<td>PC4 (ADC4/SCL/PCINT12)</td>
<td>analog input 4</td>
</tr>
<tr>
<td>PC3 (ADC3/PCINT11)</td>
<td>analog input 3</td>
</tr>
<tr>
<td>PC2 (ADC2/PCINT10)</td>
<td>analog input 2</td>
</tr>
<tr>
<td>PC1 (ADC1/PCINT9)</td>
<td>analog input 1</td>
</tr>
<tr>
<td>PC0 (ADC0/PCINT8)</td>
<td>analog input 0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. D.4 - Equivalence between the Arduino UNO and ATmega328p pins. Image provided by Arduino.

Fig. D.5 - Buzzer Node in a breadboard without the Arduino. This was a test for the perfboard version.

Fig. D.6 - Buzzer Node in a perfboard (Beta version).
E. Enclosure manufacturing

In the previous section, the Buzzer Node is manufactured into a perfboard version. This electronics board needs an enclosure with the following requirements:

- Protection of the device against environment (liquids, hits, interference…).
- Protection of the user from electrical contact.
- Mounts for connectors, buttons, displays, etc.
- Adequate heat dissipation.
- Esthetic design.

The first step is selecting a technique to manufacture it.

- Manual fabrication: Cutting and joining sheets of wood, plastic or metal is very cheap, but only viable for very simple enclosure designs.
- Buying instead of fabricating: A vast offer of designs is available on electrical hardware stores. It is the cheapest option, but it forces the rest of the project to adapt to the enclosure, when it should be the opposite. It is also visually dull.
- Plastic injection molding: It is the standard in the industry, but it is not viable for low volume production. In accordance with the lean manufacturing model, the alarm can be redesigned frequently as users demand more or fewer functions from the alarm. Often, adding or removing hardware might cause the previous enclosure to become obsolete. Having to design, manufacture and implement a new mold for each iteration would greatly increase the cost of the process.
- Additive manufacturing: 3D printers offer results with just a fraction of the quality of injection molding, but compensate with an enormous versatility. Because of this, their use is justified in this case.

E.1.1. 3D printing

3D printing consists in the manufacture of a digital object into a physical one through the consecutive impression of stacked layers with a plastic filament extruder.

In recent years, the entry price for the required hardware has fallen from high end manufacturer to hobbyist level, and is expected to continue the trend [1]. The available printer used for the alarm is a BCN3D+. Its cost as a disassembled kit was 895 €, and PLA (Polylactic acid) costs 24 €/Kg (including shipping). These are reasonable prices to pay for an appliance that will be used often.

Fig. E.1 - BCN3D+ 3D printer.
3D printing has the following drawbacks:

- Very slow printing speed: Each enclosure is expected to take 5 h to print.
- Prone to errors: there are a lot of thermal and mechanical processes that can create jams, lost steps, plastic clogs, etc.
- Fragile results: Printed products are anisotropic and weak to traction on one direction due to layer adhesion.

The process followed to build an enclosure from scratch for the Buzzer Node is the following:

1) Measure accurately the dimensions of the perfboard prototype. Usually, electronics and enclosure are designed hand in hand to achieve good results. In this case, the electronics have been laid out in a square perfboard without inconvenient protruding components. Necessary measures are length, width, height of the tallest component, location of power jack, location of the antenna, etc.

2) The design is sketched quickly as to test whether or not it is esthetic and printable. Extensively long unsupported bridges tend to fall due to gravity when printed, so the enclosure ceiling has to be printed apart.

3) The approved design is prepared in Solidworks with precise measures and in two different parts (Fig. E.2). The first one is a box without the upper side. It contains interior ridges to hold the perfboard, holes for the antenna and power jack, and embossed text about the required input voltage.

4) The second designed part is the cover, which fits precisely into the box and has holes for the buzzer sound to pass through.

5) The designs are saved in .stl format. It is a surface representation of a solid body.

6) These files are loaded into Repetier Host and converted into G-code. The actual conversion is done by another program, Slic3r, which is incorporated into Repetier Host. G-code is the set of instructions required by 3D printers. To do this, Slic3r uses many settings related to the 3D printer hardware, the plastic properties, the desired physical and esthetic properties of the outcome...

7) Repetier Host then connects to the 3D printer through a Serial connection and controls and monitors the print (Fig. E.3).
Development of a domestic or industrial alarm system for low volume, highly customizable production

Fig. E.2 - Screenshot of Solidworks with the enclosure design.

Fig. E.3 - Screenshot of Repetier Host with print preview.
The result is as seen in Fig. E.4, Fig. E.5, Fig. E.6 and Fig. E.7.

Fig. E.4 - Printed Buzzer Node enclosure box.

Fig. E.5 - Detail of the Buzzer Node enclosure box. It has a hole for the power jack and embossed text with electrical parameters.

Fig. E.6 - Buzzer Node enclosure with electronics placed inside.

Fig. E.7 - Buzzer Node enclosure with cover.
Development of a domestic or industrial alarm system for low volume, highly customizable production

F. External libraries

The following libraries have been written by people unrelated to this project. They have been released online under a certain copy license, which is indicated along with each library.

This section only features public methods that the developers have made available for their libraries and which have been used in the development of the alarm. It is not in the scope of the project to study or modify their internal mechanisms, although in some cases it has been necessary.

F.1. EEPROM.h

This library is part of the Arduino core libraries. It comes installed by default.

Authors:


License: Free software; redistributable and/or modifiable under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or any later version.

Objective: Providing read/write access to the permanent memory (the EEPROM) of the ATmega family of microcontrollers.

Used in: Central Node.

Public methods used:

- byte read(int address): Returns the byte stored in given address.
- void write(int address, byte data): Stores a byte in the given address.
- void put(int address, data): Stores data of any datatype in given position.

F.2. Printf.h

Author: J. Coliz

License: Free software; you can redistributable and/or modifiable under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Objective: Simplifying debugging over Serial PC communications by allowing concatenation of strings and other datatypes.
F.3. **Wire.h**

**Author:**

- Modified (2012) by Todd Krein.

**License:** Free software; redistributable and/or modifiable under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or any later version.

**Objective:** Providing an I²C interface. It is required by the RTClib.h library.

**Used in:** Central Node.

**Public methods used:**

- void begin(): Starts the I²C interface.

---

F.4. **RTClib.h**

**Author:** Jean-Claude Wippler.

**License:** Released to the public domain.

**Objective:** Interfacing with Real Time Clock modules such as the ds1307.

**Used in:** Central Node.

**Public methods used:**

-RTC_DS1307(): Initializes a RTC_DS1307 instance.
-uint8_t begin( void): Begins communicating with the ds1307 module.
-void adjust(const DateTime& dt): Changes the internal date and time of the ds1307 module.
-DateTime now(): Fills a DateTime object with the time reported by the ds1307.
Development of a domestic or industrial alarm system for low volume, highly customizable production

Additionally, this library carries the DateTime class, which acts as a container for a specific time. Its public methods used are:

- DateTime (const char* date, const char* time): Initializes a DateTime instance.
- uint16_t year(): Returns year of the stored time.
- uint8_t month(): Returns month of the stored time.
- uint8_t day(): Returns day of the stored time.
- uint8_t hour(): Returns hour of the stored time.
- uint8_t minute(): Returns minute of the stored time.
- uint8_t second(): Returns second of the stored time.

F.5. SPI.h

This library is part of the Arduino core libraries. It comes installed by default.

Authors:
- Modified (2014) by Paul Stoffregen (Transaction API).
- Modified (2014) by Matthijs Kooijman (SPISettings AVR).
- Modified (2014) by Andrew J. Kroll (atomicity fixes).

License: Free software; redistributable and/or modifiable under the terms of either the GNU General Public License version 2 or the GNU Lesser General Public License version 2.1, both as published by the Free Software Foundation.

Objective: Providing an SPI interface for the ATmega family of microcontrollers by taking advantage of their internal hardware designed for SPI. It is required by the RF24.h library.

Used in: All nodes.

Public methods used: None.

F.6. RF24.h

Author: Original by J. Coliz. The used version is a fork of the original with improvements from many collaborators.
License: Free software; redistributable and/or modifiable under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Objective: Controlling an nRF24l01+ radio module.

Used in: All nodes.

Public methods used:

- RF24(uint8_t _cepin, uint8_t _cspin): Initializes an RF24 object. It receives a chip enable pin and an SPI chip select pin.
- bool begin(void): Starts the radio.

The following methods, which can improve the radio’s reliability, are used to edit the RF24Network library because the author did not provide adequate means to do it from the outside without creating conflicts.

- void setRetries(uint8_t delay, uint8_t count);
- void setChannel(uint8_t channel): Changes the radiofrequency channel used (0-127).
- void setPALevel(uint8_t level): Changes the amount of power consumed the radio consumes to increase reliability. There are four available power levels.
- bool setDataRate(rf24_datarate_e speed): Changes the data transmission rate between 250 Kbps, 1 Mbps and 2 Mbps.
- void setCRCLength(rf24_crlength_e length): Change the length of the cyclic redundancy check.

F.7. RF24Network.h

Author: J. Coliz.

License: Free software; redistributable and/or modifiable under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Objective: Managing a network of nRF24l01+ modules.

Used in: All nodes.

Public methods used:

- RF24Network(RF24& _radio): Initializes an RF24Network object by passing an RF24 object.
- void begin(uint8_t _channel, uint16_t _node_address): Starts the network layer.
Development of a domestic or industrial alarm system for low volume, highly customizable production

- void update(void): Makes the necessary actions so that messages are transmitted through the network.
- bool available(void): Returns true if the device’s radio module has received and is storing a new message in queue.
- void peek(RF24NetworkHeader& header): Reads information from a new message’s header without removing it from the queue.
- size_t read(RF24NetworkHeader& header, void* message, size_t maxlen): Copies the contents of a message to a variable of any size, and removes the message from the queue.
- bool write(RF24NetworkHeader& header, const void* message, size_t len): Sends a header and a message. Returns true if the message successfully reaches the destination.

Additionally, this library carries the RF24NetworkHeader class, which acts as the envelope for any message sent. Its public fields and methods are:

- uint16_t from_node
- uint16_t to_node
- unsigned char type
- RF24NetworkHeader(uint16_t _to, unsigned char _type = 0)

F.8. LiquidCrystal.h

This library is part of the Arduino core libraries. It comes installed by default.

Author: Uncredited.

License: The source code for the Arduino environment is covered by the GPL (General Public License).

Objective: Displaying characters on an LCD screen based on the Hitachi HD44780 (or a compatible) chipset. These are text-based screens. Graphics-based screens are not compatible.

Used in: Central Node.

Public methods used:
- LiquidCrystal(int rs, int rw, int enable, int d4, int d5, int d6, int d7): Initializes a LiquidCrystal object for the selected pins.
- void begin(int cols, int rows): Configures the dimensions of the screen.
- void clear(): Displays a blank space on all positions.
- setCursor(int col, int row): Moves the internal cursor to the desired position so that the next text printed will appear there.
- byte print( __ data): Displays the data on the screen, which can be char, byte, int, long, or string.
- byte write(char symbol): Displays a character. It is useful for displaying rare symbols.
- void createChar(int index, byte[8] graphics): Creates a custom character and stores it in the LCD’s RAM under the given index. Only eight custom characters can be stored in RAM.

F.9. MFRC522.h

Authors:
- Based on code by Dr. Leong.
- Rewritten (2013) by Søren Thing Andersen (Translation to English, refactored, comments, anti-collision, cascade levels).
- Extended by Tom Clement (functionality to write to sector 0 of UID changeable Mifare cards).

License: Released into the public domain.

Objective: Encapsulating methods to control an RFID reader.

Used in: Key Tray Node

Public fields and methods used:
- MFRC522(int SS_PIN, int RST_PIN): Initializes an MFRC522 object with given SPI Select pin and Reset pin.
- PCD_Init(): Initializes the chip on the reader.
- PCD_SetAntennaGain(byte mask): Changes the gain of the RF signal.
Development of a domestic or industrial alarm system for low volume, highly customizable production

- PICC_IsNewCardPresent(): Returns true if a new RFID tag is present near the reader.
- PICC_ReadCardSerial(): Reads the RFID tag’s UID and stores it in the public data field uid. Returns true if the read is successful.
- uid: Can contain information about a tag’s UID (User ID). It is a struct with: byte size; byte uidByte[10]; byte sak. The third member, sak or select acknowledgement, serves as, confirmation that the tag was successfully read.

F.10. Keypad.h

Authors:
- Mark Stanley.
- Alexander Brevig.

License: Free software; redistributable and/or modifiable under the terms of the GNU General Public License version 2.1 as published by the Free Software Foundation.

Objective: Monitoring a 3x4 (numerical) or 4x4 (alphanumerical) keypad.

Used in: Key Tray Node.

Public methods used:
- Keypad(char *userKeymap, byte *row, byte *col, byte numRows, byte numCols) Initializes a Keypad object with all the necessary data (characters on the keys, row pins, column pins, number of rows and number of columns).
- char getKey(): Returns the character on the currently pressed key, or a null character ‘\0’ if no key is pressed.

F.11. Sleep.h

This library is part of the Arduino core libraries. It comes installed by default.

Authors:
License: Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holders nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Objective: Allows the ATmega microcontroller to be put to sleep in various sleep modes. Sleeping enormously reduces battery consumption.

Used in: Window Node

Public methods used:

- void set_sleep_mode(int mode): Configures the necessary chip settings so that when it goes to sleep it only turns off the desired set of components from the chip. The more components go to sleep, the lower the energy consumption.
- void sleep_enable(): Enables sleep.
- void sleep_cpu(): Puts the chip to sleep.
- void sleep_disable(): Disables sleep.

F.12. EnableInterrupt.h

Author: Michael Anthony Schwager.

License: Apache License, Version 2.0.

Objective: Enhance the default functionality of the interrupts on the microcontroller, and simplify its use. An interrupt is an event that triggers a certain function (Interrupt Service Routine). A very common use is to wake up a sleeping chip, since on certain sleeping modes it is impossible to do it any other way.

Used in: Window Node.
Development of a domestic or industrial alarm system for low volume, highly customizable production

Public methods used:

- `void enableInterrupt(uint8_t pin, void (*ISRfunction)(void), uint8_t mode)`: Enables an interrupt that triggers a user defined function when a certain event (rise, fall or any of them) happens on the selected pin.
- `void disableInterrupt(uint8_t pin)`: Disables the previously enabled interrupt.
G. Custom made libraries

These are custom libraries made for this project in particular. They serve little or no purpose for any other application.

G.1. AnalogButton.h

Objective: Monitoring the state of a pushbutton connected in an analog configuration.

Used in: Central Node.

Public methods:

- AnalogButton(int myMin, int myMax, int myDebounceTime): Initializes an AnalogButton object with a range of voltage values it can produce and the debounce time.
- void update(int analogVal): Updates internal logic. Must be called regularly.
- bool wasPressed(): Check if, on the last update, the button changed from released to pressed.
- bool wasReleased(): Check if, on the last update, the button changed from pressed to released.

G.2. AnalogKeyboard.h

Objective: Grouping together many AnalogButton objects to simplify their use. Specifically, the buttons are Select, Up, Right, Down and Left.

Used in: Central Node.

Public methods:

- AnalogKeyboard(int myPin): Initializes an AnalogKeyboard object for a single analog pin.
- void update(): Updates the state of the AnalogButtons inside.
- bool wasAnyArrowPressed(): Returns true if Up, Right, Down or Left have been pressed. Used to know when the user wants to access the list of settings to change.
- bool wasAnyButtonPressed(): Returns true if any button has been pressed. Used to know when to turn on the LCD backlight.
The AnalogButtons stored inside AnalogKeyboard are public data members and can be called individually: analogKeyboard.BTNRight.wasPressed().

G.3. Communications.h

Objective: Communicate with the other nodes following a predefined protocol. Analyze incoming messages and send outgoing messages. It encapsulates the libraries related to the nRF24l01+ radio.

Used in: Central Node.

Public methods:
- Communications(): Initializes a Communications object.
- void begin(): Prepares the necessary hardware (the nRF24l01+ module).
- void update(): Checks for incoming messages. If any is found, it will act accordingly by sending alert messages to the actuators if necessary.

G.4. Display.h

Objective: Presenting information on the LCD screen. Controlling the LCD screen backlight LED.

Used in: Central Node.

Public methods:
- Display(): Initializes a Display object.
- void Begin(): Prepares the LCD screen.
- void update(StateMachine, AnalogKeyboard, RTC_DS1307): Display the information on the screen according to the state of the internal state machine and user settings. Also updates the state of the LCD backlight LED.
- void TurnBacklightOn(): Manually turn the LCD screen backlight LED on.
- void TurnBacklightOff(): Manually turn the LCD screen backlight LED off.
G.5. Settings.h

Objective: Storing and retrieving user settings on the permanent memory (EEPROM).

Used in: Central Node.

Public methods:

- **Settings()**: Initializes a Settings object. Not really needed since the rest of the methods are static so they do not need a Settings object.
- **static void setAlarmState(bool)**: Sets the alarm to either activated or deactivated.
- **static bool isAlarmActivated()**: Retrieves the current state of the alarm.
- **static bool isPasscodeInDatabase(byte passcode[PASSCODELENGTH])**: Checks if a given passcode is already present in the database.
- **static void addNewPasscode(byte passcode[PASSCODELENGTH])**: Adds a new passcode to the database as long as it is not repeated or the database is not full.
- **static void deletePasscode(int index)**: Deletes the passcode in the designated position of the database.
- **static byte* getStoredPasscode(int index)**: Retrieves the passcode in the designated position of the database.
- **static void setBacklightMode(byte mode)**: Changes the operating mode of the LCD backlight between ‘Always on’, ‘On button press’ and ‘Always off’.
- **static byte getBacklightMode()**: Retrieves the current mode of the LCD backlight.
- **static void RestoreFactorySettings()**: Deletes all user settings and loads default values (alarm deactivated, backlight always on, and a single passcode which is 123456).

G.6. StateMachine.h

Objective: Analyzing user input and navigating through a set of different menus where the user can change options. This is the backend logic (it does not display anything on the screen).

Used in: Central Node.

Public methods:
Development of a domestic or industrial alarm system for low volume, highly customizable production

- StateMachine(): Initializes a StateMachine object.
- void update(AnalogKeyboard analogKeyboard, RTC_DS1307 clock): Updates the current state of the finite state machine based on user input. Performs necessary actions on state transitions (mostly changing settings).
- int GetState(): Returns current state of the FSM.
- int GetCursorPosition(): Returns current cursor position.

G.7. Sounds.h

Objective: Provides functions to make sounds and short melodies on a piezo to give audio feedback to the user on certain events.

Used in: Key Tray Node.

Public methods:
- Sounds(): Initializes a Sounds object. Not really needed since the rest of the methods are static so they do not need a Sounds object.
- static void KeyPressed(): Makes a short beep.
- static void RFIDdetected(): Makes a short beep.
- static void AccessGranted(): Makes a ‘happy’ three-note melody.
- static void AccessDenied(): Makes a ‘sad’ three-note melody
- static void AlarmActivated(): Makes a series of beeps to denote urgency to leave the area before any sensor is triggered.
- static void AddingNewPasscode(): Makes a distinctive melody.
- static void SentNewPasscode(): Makes a distinctive melody.
- static void Transmittion(): Makes a very short beep.
- static void CentralNodeUnresponsive(): Makes a melody that reminds of an error.

G.8. Pitches.h

Objective: A long list of constants linking each musical note to its acoustic frequency.

Used in: Key Tray Node.

Public methods: None.
H. C++ code

The code is distributed between five sketches (the program that runs each node), custom made libraries and external libraries.

The source code developed for this project (excluding external libraries written by other developers) has a total size of 2,902 lines of code distributed as seen in Fig. H.1. This includes blank lines, comments and preprocessor directives.

![Pie chart showing the percentage of code in each node](image)

**Fig. H.1 - Percentage of total number of lines of code in each node. Image generated with the program LocMetrics.**

Due to the size of the source code, it has not been presented on paper. Instead, it can be found on the CD distributed along the Memory and Annex.
# Development of a domestic or industrial alarm system for low volume, highly customizable production

## I. Bill of materials (BOM)

### I.1. Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Measurement units</th>
<th>Quantity per pack</th>
<th>Price per pack (€)</th>
<th>Price per unit (€)</th>
<th>Quantity used</th>
<th>Total price (€)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadboard (big)</td>
<td>pcs</td>
<td>1</td>
<td>3.89</td>
<td>3.89</td>
<td>2</td>
<td>7.78</td>
<td>Connects components.</td>
</tr>
<tr>
<td>Jumper cable</td>
<td>pcs</td>
<td>30</td>
<td>2</td>
<td>0.07</td>
<td>19</td>
<td>1.27</td>
<td>Connects components.</td>
</tr>
<tr>
<td>Wire AWG 22</td>
<td>m</td>
<td>28</td>
<td>17.03</td>
<td>0.61</td>
<td>0.3</td>
<td>0.18</td>
<td>Connects components.</td>
</tr>
<tr>
<td>Real time clock ds1307</td>
<td>pcs</td>
<td>1</td>
<td>1.78</td>
<td>1.78</td>
<td>1</td>
<td>1.78</td>
<td>Keeps track of time.</td>
</tr>
<tr>
<td>nRF24I01+ (with antenna)</td>
<td>pcs</td>
<td>1</td>
<td>4.62</td>
<td>4.62</td>
<td>1</td>
<td>4.62</td>
<td>Radio module.</td>
</tr>
<tr>
<td>nRF24I01+ custom breakout board</td>
<td>pcs</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>1</td>
<td>1.00</td>
<td>Made with spare perfboard and connectors. Price is approximate.</td>
</tr>
<tr>
<td>Capacitor (22 pF)</td>
<td>pcs</td>
<td>10</td>
<td>0.3</td>
<td>0.03</td>
<td>1</td>
<td>0.03</td>
<td>Filters for the crystal.</td>
</tr>
<tr>
<td>Capacitor (4.7 μF)</td>
<td>pcs</td>
<td>120</td>
<td>5</td>
<td>0.04</td>
<td>1</td>
<td>0.04</td>
<td>Filter for the nRF24I01+.</td>
</tr>
<tr>
<td>Capacitor (220 μF)</td>
<td>pcs</td>
<td>120</td>
<td>5</td>
<td>0.04</td>
<td>1</td>
<td>0.04</td>
<td>Filter for all the board.</td>
</tr>
<tr>
<td>LCD screen, 4x20 characters</td>
<td>pcs</td>
<td>1</td>
<td>8.01</td>
<td>8.01</td>
<td>1</td>
<td>8.01</td>
<td>Displays information.</td>
</tr>
<tr>
<td>Resistor (multiple values)</td>
<td>pcs</td>
<td>1000</td>
<td>9.06</td>
<td>0.01</td>
<td>7</td>
<td>0.06</td>
<td>Used for the analog keyboard and the LCD backlight LED.</td>
</tr>
<tr>
<td>Potentiometer (10 kΩ)</td>
<td>pcs</td>
<td>10</td>
<td>2.42</td>
<td>0.24</td>
<td>1</td>
<td>0.24</td>
<td>Adjusts contrast of the LCD.</td>
</tr>
<tr>
<td>Big pushbutton</td>
<td>pcs</td>
<td>20</td>
<td>1.61</td>
<td>0.08</td>
<td>1</td>
<td>0.08</td>
<td>Select button.</td>
</tr>
<tr>
<td>Small pushbutton</td>
<td>pcs</td>
<td>50</td>
<td>1.39</td>
<td>0.03</td>
<td>4</td>
<td>0.11</td>
<td>Directional buttons.</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>39.80</strong></td>
<td><strong>Total price</strong></td>
</tr>
</tbody>
</table>

Tab. I.1 - BOM: Central Node (Alpha version).
### Movement Detector Node (Alpha version)

<table>
<thead>
<tr>
<th>Name</th>
<th>Measurement units</th>
<th>Quantity per pack</th>
<th>Price per pack (€)</th>
<th>Price per unit (€)</th>
<th>Quantity used</th>
<th>Total price (€)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Nano</td>
<td>pcs</td>
<td>1</td>
<td>6.64</td>
<td>6.64</td>
<td>1</td>
<td>6.64</td>
<td>Brain of the node.</td>
</tr>
<tr>
<td>Breadboard (big)</td>
<td>pcs</td>
<td>1</td>
<td>3.89</td>
<td>3.89</td>
<td>1</td>
<td>3.89</td>
<td>Connects components.</td>
</tr>
<tr>
<td>Jumper cable</td>
<td>pcs</td>
<td>30</td>
<td>0.07</td>
<td>0.07</td>
<td>9</td>
<td>0.60</td>
<td>Connects components.</td>
</tr>
<tr>
<td>Capacitor (10 µF)</td>
<td>pcs</td>
<td>120</td>
<td>0.04</td>
<td>0.04</td>
<td>1</td>
<td>0.04</td>
<td>Filter for the nRF24I01+.</td>
</tr>
<tr>
<td>LED (red)</td>
<td>pcs</td>
<td>40</td>
<td>0.07</td>
<td>0.07</td>
<td>1</td>
<td>0.07</td>
<td>Provides visual feedback.</td>
</tr>
<tr>
<td>nRF24I01+</td>
<td>pcs</td>
<td>4</td>
<td>3.43</td>
<td>0.86</td>
<td>1</td>
<td>0.86</td>
<td>Radio module.</td>
</tr>
<tr>
<td>nRF24I01+ custom breakout board</td>
<td>pcs</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>Made with spare perfboard and connectors. Price is approximate.</td>
</tr>
<tr>
<td>PIR sensor module</td>
<td>pcs</td>
<td>5</td>
<td>1.20</td>
<td>1.20</td>
<td>1</td>
<td>1.20</td>
<td>Detects movement.</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>14.30</strong></td>
<td></td>
</tr>
</tbody>
</table>

Tab. 1.2 - BOM: Movement Detector Node (Alpha version).

### Buzzer Node (Alpha version)

<table>
<thead>
<tr>
<th>Name</th>
<th>Measurement units</th>
<th>Quantity per pack</th>
<th>Price per pack (€)</th>
<th>Price per unit (€)</th>
<th>Quantity used</th>
<th>Total price (€)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Nano</td>
<td>pcs</td>
<td>1</td>
<td>6.64</td>
<td>6.64</td>
<td>1</td>
<td>6.64</td>
<td>Brain of the node.</td>
</tr>
<tr>
<td>Breadboard (medium)</td>
<td>pcs</td>
<td>1</td>
<td>2.92</td>
<td>2.92</td>
<td>1</td>
<td>2.92</td>
<td>Connects components.</td>
</tr>
<tr>
<td>Jumper cable</td>
<td>pcs</td>
<td>30</td>
<td>0.07</td>
<td>0.07</td>
<td>8</td>
<td>0.53</td>
<td>Connects components.</td>
</tr>
<tr>
<td>Capacitor (4.7 µF)</td>
<td>pcs</td>
<td>120</td>
<td>0.04</td>
<td>0.04</td>
<td>1</td>
<td>0.04</td>
<td>Filter for the nRF24I01+ (transmitting only).</td>
</tr>
<tr>
<td>Buzzer</td>
<td>pcs</td>
<td>10</td>
<td>0.30</td>
<td>0.30</td>
<td>1</td>
<td>0.30</td>
<td>Makes noise when alarm is triggered.</td>
</tr>
<tr>
<td>nRF24I01+</td>
<td>pcs</td>
<td>4</td>
<td>3.43</td>
<td>0.86</td>
<td>1</td>
<td>0.86</td>
<td>Radio module.</td>
</tr>
<tr>
<td>nRF24I01+ custom breakout board</td>
<td>pcs</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>Made with spare perfboard and connectors. Price is approximate.</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>12.29</strong></td>
<td></td>
</tr>
</tbody>
</table>

Tab. 1.3 - BOM: Buzzer Node (Alpha version).
### Key Tray Node (Alpha version)

<table>
<thead>
<tr>
<th>Name</th>
<th>Measurement units</th>
<th>Quantity per pack</th>
<th>Price per pack (€)</th>
<th>Price per unit (€)</th>
<th>Quantity used</th>
<th>Total price (€)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadboard (big)</td>
<td>pcs</td>
<td>1</td>
<td>3.89</td>
<td>3.89</td>
<td>1</td>
<td>3.89</td>
<td>Connects components.</td>
</tr>
<tr>
<td>Jumper cable</td>
<td>pcs</td>
<td>30</td>
<td>0.07</td>
<td>23</td>
<td>1.53</td>
<td>Connects components.</td>
<td></td>
</tr>
<tr>
<td>Wire AWG 22</td>
<td>m</td>
<td>28</td>
<td>0.61</td>
<td>0.30</td>
<td></td>
<td></td>
<td>Connects components.</td>
</tr>
<tr>
<td>Voltage regulator 3.3 V(TS111733)</td>
<td>pcs</td>
<td>5</td>
<td>0.75</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>Provides 3.3 V to the nRF24I01+ and the RFID-RC522, since the regulator on the Arduino board cannot provide the necessary current.</td>
</tr>
<tr>
<td>Resistor (multiple values)</td>
<td>pcs</td>
<td>1000</td>
<td>0.01</td>
<td>3</td>
<td>0.03</td>
<td></td>
<td>Pullups for the SPI Select lines and protection for the LEDs.</td>
</tr>
<tr>
<td>Piezo</td>
<td>pcs</td>
<td>10</td>
<td>0.15</td>
<td>1</td>
<td>0.15</td>
<td></td>
<td>Provides user with sound feedback.</td>
</tr>
<tr>
<td>LED (multiple colors)</td>
<td>pcs</td>
<td>40</td>
<td>0.07</td>
<td>2</td>
<td>0.14</td>
<td></td>
<td>Provides user with visual feedback.</td>
</tr>
<tr>
<td>nRF24I01+</td>
<td>pcs</td>
<td>4</td>
<td>0.86</td>
<td>1</td>
<td>0.86</td>
<td></td>
<td>Radio module.</td>
</tr>
<tr>
<td>nRF24I01+ custom breakout board</td>
<td>pcs</td>
<td>1</td>
<td>1.00</td>
<td>1</td>
<td>1.00</td>
<td></td>
<td>Made with spare perfboard and connectors. Price is approximate.</td>
</tr>
<tr>
<td>Capacitor (10 µF)</td>
<td>pcs</td>
<td>120</td>
<td>0.04</td>
<td>1</td>
<td>0.04</td>
<td></td>
<td>Filter for the nRF24I01+.</td>
</tr>
<tr>
<td>RFID-RC522 reader</td>
<td>pcs</td>
<td>1</td>
<td>5.20</td>
<td>5.20</td>
<td>1</td>
<td>5.20</td>
<td>Reads RFID tags.</td>
</tr>
<tr>
<td>4x3 numerical keypad</td>
<td>pcs</td>
<td>1</td>
<td>4.54</td>
<td>4.54</td>
<td>1</td>
<td>4.54</td>
<td>Lets the user input numerical passcodes.</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>32.98</strong></td>
<td></td>
</tr>
</tbody>
</table>

Tab. I.4 - BOM: Key Tray Node (Alpha version).
<table>
<thead>
<tr>
<th>Name</th>
<th>Measurement units</th>
<th>Quantity per pack</th>
<th>Price per pack (€)</th>
<th>Price per unit (€)</th>
<th>Quantity used</th>
<th>Total price (€)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATmega328p</td>
<td>pcs</td>
<td>10</td>
<td>20.98</td>
<td>2.10</td>
<td>1</td>
<td>2.10</td>
<td>Brain of the node. Must have the Arduino Bootloader preinstalled, or an external programmer will be required</td>
</tr>
<tr>
<td>Breadboard (small)</td>
<td>pcs</td>
<td>5</td>
<td>4.66</td>
<td>0.93</td>
<td>2</td>
<td>1.86</td>
<td>Connects components.</td>
</tr>
<tr>
<td>Crystal (16 MHz)</td>
<td>pcs</td>
<td>20</td>
<td>1.36</td>
<td>0.07</td>
<td>1</td>
<td>0.07</td>
<td>Sets the ATmega328p's clock.</td>
</tr>
<tr>
<td>Capacitor (22 pF)</td>
<td>pcs</td>
<td>10</td>
<td>0.3</td>
<td>0.03</td>
<td>2</td>
<td>0.06</td>
<td>Filters for the crystal.</td>
</tr>
<tr>
<td>Capacitor (4.7 μF)</td>
<td>pcs</td>
<td>120</td>
<td>5</td>
<td>0.04</td>
<td>1</td>
<td>0.04</td>
<td>Filter for the nRF24l01+.</td>
</tr>
<tr>
<td>Capacitor (220 μF)</td>
<td>pcs</td>
<td>120</td>
<td>5</td>
<td>0.04</td>
<td>1</td>
<td>0.04</td>
<td>Filter for all the board.</td>
</tr>
<tr>
<td>Switch with lever</td>
<td>pcs</td>
<td>1</td>
<td>1.84</td>
<td>1.84</td>
<td>1</td>
<td>1.84</td>
<td>Detects when a door or window has been opened.</td>
</tr>
<tr>
<td>Resistor (multiple values)</td>
<td>pcs</td>
<td>1000</td>
<td>9.06</td>
<td>0.01</td>
<td>3</td>
<td>0.03</td>
<td>Protect the LED, pull-up the switch and pull-up the nRF24l01+ SPI Select line</td>
</tr>
<tr>
<td>Battery holder</td>
<td>pcs</td>
<td>20</td>
<td>1.62</td>
<td>0.08</td>
<td>1</td>
<td>0.08</td>
<td>Accommodates 3 AAA batteries.</td>
</tr>
<tr>
<td>Diode (1N4007)</td>
<td>pcs</td>
<td>50</td>
<td>1.95</td>
<td>0.04</td>
<td>1</td>
<td>0.04</td>
<td>Protects against inverse current.</td>
</tr>
<tr>
<td>Jumper cable</td>
<td>pcs</td>
<td>30</td>
<td>2</td>
<td>0.07</td>
<td>2</td>
<td>0.13</td>
<td>Connects components.</td>
</tr>
<tr>
<td>Wire AWG 22</td>
<td>m</td>
<td>28</td>
<td>17.03</td>
<td>0.61</td>
<td>0.15</td>
<td>0.09</td>
<td>Connects components.</td>
</tr>
<tr>
<td>nRF24l01+</td>
<td>pcs</td>
<td>4</td>
<td>3.43</td>
<td>0.86</td>
<td>1</td>
<td>0.86</td>
<td>Radio module.</td>
</tr>
<tr>
<td>nRF24l01+ custom breakout board</td>
<td>pcs</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>1</td>
<td>1.00</td>
<td>Made with spare perfboard and connectors. Price is approximate.</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>8.24</strong></td>
<td></td>
</tr>
</tbody>
</table>

Tab. 1.5 - BOM: Window Node (Alpha version).
<table>
<thead>
<tr>
<th>Name</th>
<th>Measurement units</th>
<th>Quantity per pack</th>
<th>Price per pack (€)</th>
<th>Price per unit (€)</th>
<th>Quantity used</th>
<th>Total price (€)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfboard</td>
<td>pcs</td>
<td>1</td>
<td>6.00</td>
<td>0.5</td>
<td>3.00</td>
<td></td>
<td>Board to solder all components.</td>
</tr>
<tr>
<td>Chip socket (28 pins)</td>
<td>pcs</td>
<td>10</td>
<td>0.15</td>
<td>1</td>
<td>0.15</td>
<td></td>
<td>Protects the chip against accidental burning when soldering.</td>
</tr>
<tr>
<td>ATmega328p</td>
<td>pcs</td>
<td>10</td>
<td>2.10</td>
<td>1</td>
<td>2.10</td>
<td></td>
<td>Microcontroller.</td>
</tr>
<tr>
<td>Female power jack (2.1 mm)</td>
<td>pcs</td>
<td>10</td>
<td>0.10</td>
<td>1</td>
<td>0.10</td>
<td></td>
<td>Provides connection for an external power source.</td>
</tr>
<tr>
<td>Voltage regulator 3.3 V (TS111733)</td>
<td>pcs</td>
<td>5</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td></td>
<td>Reduces input voltage to 3.3 V in order to prevent damage to the nRF24I01+ board.</td>
</tr>
<tr>
<td>Diode (1N4007)</td>
<td>pcs</td>
<td>50</td>
<td>0.04</td>
<td>1</td>
<td>0.04</td>
<td></td>
<td>Protects against inverse current.</td>
</tr>
<tr>
<td>Crystal (16 MHz)</td>
<td>pcs</td>
<td>20</td>
<td>0.07</td>
<td>1</td>
<td>0.07</td>
<td></td>
<td>Sets the ATmega328p's clock.</td>
</tr>
<tr>
<td>Capacitor (22 pF)</td>
<td>pcs</td>
<td>10</td>
<td>0.03</td>
<td>2</td>
<td>0.06</td>
<td></td>
<td>Filters for the crystal.</td>
</tr>
<tr>
<td>Capacitor (4.7 µF)</td>
<td>pcs</td>
<td>120</td>
<td>0.04</td>
<td>1</td>
<td>0.04</td>
<td></td>
<td>Filter for the nRF24I01+.</td>
</tr>
<tr>
<td>Capacitor (220 µF)</td>
<td>pcs</td>
<td>120</td>
<td>0.04</td>
<td>1</td>
<td>0.04</td>
<td></td>
<td>Filter for all the board.</td>
</tr>
<tr>
<td>Transistor (pN2222A)</td>
<td>pcs</td>
<td>20</td>
<td>0.08</td>
<td>1</td>
<td>0.08</td>
<td></td>
<td>Drives the buzzer.</td>
</tr>
<tr>
<td>nRF24I01+</td>
<td>pcs</td>
<td>4</td>
<td>0.86</td>
<td>1</td>
<td>0.86</td>
<td></td>
<td>Radio module. Needs Dupont connectors to prevent damage.</td>
</tr>
<tr>
<td>Wire AWG 22</td>
<td>m</td>
<td>28</td>
<td>0.61</td>
<td>0.00</td>
<td></td>
<td></td>
<td>Connects components.</td>
</tr>
<tr>
<td>Led</td>
<td>pcs</td>
<td>40</td>
<td>0.07</td>
<td>1</td>
<td>0.07</td>
<td></td>
<td>Provides feedback to debug node without the buzzer. Is detachable.</td>
</tr>
<tr>
<td>Resistor</td>
<td>pcs</td>
<td>1000</td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
<td></td>
<td>Protects LED.</td>
</tr>
<tr>
<td>Female Dupont connector</td>
<td>pcs</td>
<td>10</td>
<td>0.14</td>
<td>0.5</td>
<td>0.07</td>
<td></td>
<td>Used to connect the LED and the nRF24I01+.</td>
</tr>
<tr>
<td>Buzzer</td>
<td>pcs</td>
<td>1</td>
<td>3.24</td>
<td>1</td>
<td>3.24</td>
<td></td>
<td>Makes noise when alarm is triggered.</td>
</tr>
<tr>
<td>PLA filament (3 mm, green)</td>
<td>g</td>
<td>1000</td>
<td>0.02</td>
<td>80</td>
<td>1.83</td>
<td></td>
<td>Used to 3D print the enclosure.</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>12.51</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### I.2. Tools

#### Tools (electronics prototyping)

<table>
<thead>
<tr>
<th>Name</th>
<th>Measurement</th>
<th>Quantity per pack</th>
<th>Price per pack (€)</th>
<th>Price per unit (€)</th>
<th>Quantity used</th>
<th>Total price (€)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power source with variable Vo</td>
<td>pcs</td>
<td>1</td>
<td>8.00</td>
<td>8.00</td>
<td>1</td>
<td>8.00</td>
<td>Used to temporarily power on prototypes.</td>
</tr>
<tr>
<td>Tweezers</td>
<td>pcs</td>
<td>1</td>
<td>0.80</td>
<td>0.80</td>
<td>1</td>
<td>0.80</td>
<td>Used to work with small components.</td>
</tr>
<tr>
<td>Cable cutter</td>
<td>pcs</td>
<td>1</td>
<td>1.20</td>
<td>1.20</td>
<td>1</td>
<td>1.20</td>
<td>Used to cut and peel wire to adequate length.</td>
</tr>
<tr>
<td>Multimeter</td>
<td>pcs</td>
<td>1</td>
<td>15.00</td>
<td>15.00</td>
<td>1</td>
<td>15.00</td>
<td>Used to debug circuits.</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>25.00</strong></td>
<td></td>
</tr>
</tbody>
</table>


#### Tools (embedded programming)

<table>
<thead>
<tr>
<th>Name</th>
<th>Measurement</th>
<th>Quantity per pack</th>
<th>Price per pack (€)</th>
<th>Price per unit (€)</th>
<th>Quantity used</th>
<th>Total price (€)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC + periferics</td>
<td>pcs</td>
<td>1</td>
<td>800.00</td>
<td>800.00</td>
<td>1</td>
<td>800.00</td>
<td>Generic price. Programming can be done on a much lower range PC but CAD needs more processing power.</td>
</tr>
<tr>
<td>USB cable type B</td>
<td>pcs</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>2</td>
<td>0.00</td>
<td>Included with Arduino MEGA.</td>
</tr>
<tr>
<td>USB cable type Mini</td>
<td>pcs</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>2</td>
<td>0.00</td>
<td>Included with Arduino Nano.</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>800.00</strong></td>
<td></td>
</tr>
</tbody>
</table>

Tab. I.8 - BOM: Tools for embedded programming.
## Development of a domestic or industrial alarm system for low volume, highly customizable production

### Tools (soldering)

<table>
<thead>
<tr>
<th>Name</th>
<th>Measurement units</th>
<th>Quantity per pack</th>
<th>Price per pack (€)</th>
<th>Price per unit (€)</th>
<th>Quantity used</th>
<th>Total price (€)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pliers</td>
<td>pcs</td>
<td>1</td>
<td>5</td>
<td>5.00</td>
<td>1</td>
<td>5.00</td>
<td>Used to cut perfboard to size.</td>
</tr>
<tr>
<td>Solder wire (50 g)</td>
<td>pcs</td>
<td>1</td>
<td>3.33</td>
<td>3.33</td>
<td>1</td>
<td>3.33</td>
<td>Used to solder components to perfboard. Due to its low price per board and difficulty to measure, it is considered a tool and not an expendable.</td>
</tr>
<tr>
<td>Solder removing pump</td>
<td>pcs</td>
<td>1</td>
<td>8.4</td>
<td>8.40</td>
<td>1</td>
<td>8.40</td>
<td>Used to correct wrong connections.</td>
</tr>
<tr>
<td>Soldering iron HAKKO FX-888</td>
<td>pcs</td>
<td>1</td>
<td>101.5</td>
<td>101.46</td>
<td>1</td>
<td>101.46</td>
<td>Used to solder components. It has variable temperature.</td>
</tr>
<tr>
<td>Solder paste (50 g)</td>
<td>pcs</td>
<td>1</td>
<td>4</td>
<td>4.00</td>
<td>1</td>
<td>4.00</td>
<td>Used to prevent solder oxidation.</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>129.18</strong></td>
<td></td>
</tr>
</tbody>
</table>


### Tools (3D printing)

<table>
<thead>
<tr>
<th>Name</th>
<th>Measurement units</th>
<th>Quantity per pack</th>
<th>Price per pack (€)</th>
<th>Price per unit (€)</th>
<th>Quantity used</th>
<th>Total price (€)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D printer (BCN3D+)</td>
<td>pcs</td>
<td>1</td>
<td>896</td>
<td>896.00</td>
<td>1</td>
<td>896.00</td>
<td>Bought disassembled.</td>
</tr>
<tr>
<td>Hairspray (500 ml)</td>
<td>pcs</td>
<td>1</td>
<td>4</td>
<td>4.00</td>
<td>1</td>
<td>4.00</td>
<td>Used to improve adherence to the base.</td>
</tr>
<tr>
<td>Glass cleaning blade</td>
<td>pcs</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>1</td>
<td>1.00</td>
<td>Used to remove dry hairspray from the base.</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>901.00</strong></td>
<td></td>
</tr>
</tbody>
</table>

## I.3. Software

### Software (Embedded development)

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Conditions</th>
<th>Price (€)</th>
<th>Free alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino IDE</td>
<td>Development of code, embedded programming, debugging.</td>
<td></td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Notepad++</td>
<td>Development of code.</td>
<td></td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

*Tab. I.11 - BOM: Embedded development software.*

### Software (Enclosure manufacturing)

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Conditions</th>
<th>Price (€)</th>
<th>Free alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solidworks</td>
<td>Design of enclosures.</td>
<td>Subscription (1 user, 1 year)</td>
<td>1,187</td>
<td>SketchUp</td>
</tr>
<tr>
<td>Slic3r</td>
<td>Turning CAD designs into Gcode (instructions for the 3D printer)</td>
<td></td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Repetier Host</td>
<td>PC control and monitoring of the 3D printer.</td>
<td></td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

*Tab. I.12 - BOM: Enclosure manufacturing software.*

### Software (Documentation)

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Conditions</th>
<th>Price (€)</th>
<th>Free alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Office 360</td>
<td>Memory and annex edition.</td>
<td>Subscription (1 user, 1 year)</td>
<td>60</td>
<td>OpenOffice</td>
</tr>
<tr>
<td>Mendeley Desktop</td>
<td>References management.</td>
<td></td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Autocad 2014</td>
<td>Preparation of diagrams and schematics.</td>
<td>Subscription (1 user, 1 year)</td>
<td>1,541</td>
<td>FreeCAD</td>
</tr>
<tr>
<td>yEd Graph Editor</td>
<td>State machine diagrams.</td>
<td></td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>KiCad</td>
<td>Schematics edition.</td>
<td></td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>LocMetrics</td>
<td>Source code analytics.</td>
<td></td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

*Tab. I.13 - BOM: Documentation software.*
J. Schematics

The following pages are electrical schematics for the five nodes in Alpha version plus the Buzzer Node in Beta version.
Serial
SPI
RST
D0,RX
D1,TX
D2
D3,PWM
D4
Vcc
GND
XTAL1
XTAL2
Vcc
D5,PWM
AREF
D6,PWM
GND
D7
A0
D8
A1
PWM,D9
A2
PWM,D10
A3
PWM,D11,MOSI
A4
MISO,D12
PWM,SCK,D13
ATmega328p
C
22p
16 MHz
CRYSTAL
3.3_V
CE
CSN
GND
IRQ
MISO
MOSI
SCK
NRF24L01+
C
10u
1N4007
R
6.8k
VCC
3xAAA_Battery
Switch (Long lever)
LED
1
2
Optimal Vcc: 3.3 V

ing the document as if you were reading it naturally.
K. CAD blueprints

The following pages are plots for the Buzzer Node’s Beta version 3D printed enclosure. They have been designed keeping in mind the limitations of 3D printers in regard to tolerances and unsupported material.
Development of a domestic or business security system for low volume, highly customizable production.

Eduard Tiron
20/05/2015

Buzzer Node Enclosure Box

Author:
Eduard Tiron
Date:
20/05/2015

Buzzer Node Enclosure Box

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

Sheet:
1/1

Size:
A4

Scale:
1:1

Material:
Green PLA

Weight:
55 g
Development of a domestic or business security system for low volume, highly customizable production

Author: Eduard Tiron
Date: 20/05/2015

Buzzer Node Enclosure Cover

Sheet 1/1

Weight: 25 g

Material: Green PLA

Scale 2:1

Project: Development of a domestic or business security system for low volume, highly customizable production
Development of a domestic or industrial alarm system for low volume, highly customizable production