UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Ingegneria dell'Informazione,
Ingegneria Elettrica e Matematica Applicata

Embedded System Project
Group 10

HOME SECURITY SYSTEM

Author 1 :
Michele Barbella
0622701341

Author 2:
Andrea Valitutto
0622701366

Author 3:
Salvatore Ventre
0622701343

Professor:
Vincenzo Carletti

Anno Accademico 2019/2020

Contents

List of Figures

1 Solution

To realize the embedded system project **Home Security System** to protect your home from undesired intrusions, is required to manage these following features:

- Activate/Deactivate the system.
- Two different kind of alarms: Area and Barrier.
- Activate/Deactivate both or a single alarm.
- Drive a buzzer for notifications and alarms (different sound for different alarm).
- Active alarm.
- Delayed alarms.
- User pin to activate/deactivate the system and the alarms.
- Log through the UART.

The solution was to use an **STM32F401RE** micro-controller, an electronic device integrated on a single electronic circuit, specially chosen to control and manage all the functions required for delivery. In addition, other additional components were also used, such as sensors and alarms, which allowed the proper functioning of the whole apparatus and provided a "simple" interface with the end user. The behavior of the system, managed by the micro-controller, was programmed through the development set **STM32CubeIDE** and the use of the C language.

The development was carried out also thanks to Git, a version control software that allowed to improve repository code by fixing bugs or adding functionality. During the implementation phase, tests were performed to ensure that the device responded adequately to the commands given. The initial design phase was one of the most important as it saw the choice of the micro-controller configuration.

In particular, all the parameters that constrained the connected sensors were set and the pins on the board useful for connection with the various components were set. Precisely for this reason, this phase was the one that

took the most time, and at the same time was carried out several times during the work, to verify that the assessments made were correct from an implementation point of view. Furthermore, the analysis of the requirements also led to the division of the project based on the work that had to be done on the different functions of the machine. The writing of the code occupied part of the entire development phase (design, implementation and testing) of the device management software, as well as the **debugging** or the identification and resolution of any bugs in the code itself (via debugger).

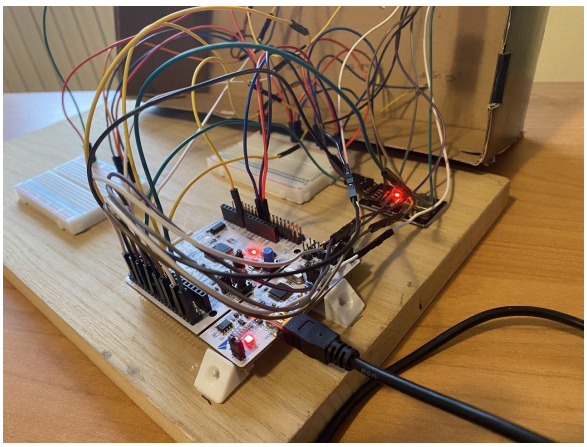

Figure 1: Structure: cable management of the device.

The final phase was used for the construction of an craft structure that recalled a prototype of the entire system. Have been developed the connections between the various sensors and the system components, and a structure has been created with household materials, such as to make simpler the test and demonstration on the functioning of the final product. Different varieties of cables were used, and two breadboards that facilitated the connections with the sensors located further away from the micro-controller, which, given the nature of the project, must be prepared so as to make easier the serial connection with a PC. Furthermore, during this phase it was preferred to use an external LED with respect to the **User LED** supplied by the card. The motivation is due both to the arrangement of the handcrafted structure built, which obstructed the view of the LED, and to the easy access to the connection of the Pin corresponding to the "user LED".

2 Design Choices

The development of the project has seen certain choices of design, which during the development of the device, have conditioned the progress. A first choice was made at the initial configuration stage, in which a **clock speed** was set equal to 42 MHz. This option placed certain constraints on the implementable parameters, such as the **prescaler** values on the different timers to perform the necessary counting operations. Later, other decisions have been made on the data structures that properly collect the data and organize the information entered by the user that interacts with the system.

2.1 Timers

In total, we used six timers to carry out all the counting operations within the system:

- TIMER 3:
  This timer is used in **PWM** mode, that is a technique used to generate several pulse in a given period of time or at a given frequency. This allows us to manipulate the behavior of output voltage of a Pin, proportionally. The timer output is linked to Buzzer Alarm (Channel 1), causing a different sounds for each situation.

- TIMER 4:
  This timer counts the duration of the alarms, this is provided by the user when the system boots, otherwise the timer uses the duration of the default configuration as the duration.

- TIMER 5:
  This timer counts the delay of the Barrier Alarm, i.e. specifies how long the photo-resistor value is greater than the threshold value for the alarm to sound.

- TIMER 9:
  This timer counts the delay of the Area Alarm, i.e. specifies how long the signal must remain high for the alarm to sound.

- TIMER 10:
  When system boots this timer counts for 30 seconds waiting for the initial configuration. After this or if this occurs before 30 seconds, system starts and the timer starts counting ten seconds, that is the interval for system log messages.

- TIMER 11:
  This timer is used to toggle user's Pin every 1/4 second when the system is active, and at the same time when both system and Barrier Alarm are active it allows to enables the interrupt and starts ADC conversion.

2.2 Structures

To manage date and time in the project, we used a structure called "**datetime**" to set data and time, it is composed by seven fields: seconds, minutes, hours, day, date, month, year.

An other structure has been used to store the status of the whole system and its sensors. The structure used is called "**system_status**" and it has five boolean fields:

- system_active
- area_active
- barrier_active
- area_alarmed
- barrier_alarmed

We used elements of this structure to verify if the whole system and sensors are active, alarmed or not. This allows to manage required actions related to system status.

The last of structure we examine is **system_config**, it is composed by four fields:

- pin

- delay_pir
- delay_barrier
- alarm_duration

These fields can be customized during the initial configuration through the putty interface by the user or they will have a default value.

2.3 Interrupt

Taking advantage of the potential of the **NVIC**, most of the events were managed through interruptions, implementing the corresponding callback to describe the code to be executed when a particular situation occurs.

2.3.1 UART Interrupt

As for the transmission and reception of data through the UART, was chosen the transmission in DMA (Dynamic memory Access) and the reception in interruptions without the use of DMA. This choice was made on the basis of the quantity of data to be transmitted and received through the UART interface: in transmission we have a lot of data to send on the serial line: at the beginning, during the initial configuration and finally during the execution of the program.

During the initial configuration, a lot of data is sent in transmission to communicate to the user when and what data to enter to configure the system. After configuring the system, irrespective of whether it uses the default configuration or the one entered by the user, the system will send a log message every 10 seconds to communicate the status of the system. In addition, another message is transmitted on the serial port when the user enters an invalid pin or a valid or invalid sequence for activating or deactivating the system or an alarm.

As for reception, we do not have a lot of data to receive from the serial line, but only during the initial configuration if the user wants to enter a customized system configuration: pin, pir delay, barrier delay, alarm

duration, date and time. For these reasons we chosen to use interrupted reception, but without the use of DMA, while transmission with the use of DMA. For transmission the callback *void HAL_UART_TxCpltCallback (UART_HandleTypeDef * UartHandle)* has been implemented, while for the reception *void HAL_UART_RxCpltCallback (UART_HandleTypeDef * UartHandle).*

2.3.2 I²C Interrupt

In this system the I^2C protocol was used to send and receive data from the RTC (Real Time Clock), since the date and time are set only during the initial configuration and then it is not possible to change them during the execution of the program, it was chosen to use the transmission of data to RTC in blocking mode. As for data reception, since we have to read the data from the RTC every 10 seconds to print the system log message, we managed the reception of the data in non-blocking mode (Interruption mode), using DMA, so as to print the message of log only when reception is finished and all data has been completely received. When data reception is complete, the callback *void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)* has been implemented to describe the code to be executed.

2.3.3 ADC Interrupt

An analog-to-digital converter is a system that converts an analog signal, such as a light entering a photo resistor, into a digital signal. The conversion of the various channels can be performed in single, continuous, scan or discontinuous mode.

It is possible to use the ADC in Polling, in Interrupt and in DMA mode. We chose to use the ADC in interrupt mode using DMA to ensure that the conversion of these signals, during the execution of the program, takes place in non-blocking mode. Furthermore, since we do not need a continuous conversion for the operation, the conversion is done at 1/4 second intervals: the value coming from the Photo-resistor is used to understand when the laser light is obscured by the presence of an obstacle or a person.

When the laser light does not reach the Photo-resistor for a specific period, the sensor becomes alarmed and, from this moment on until the alarm sound ends, the conversion is stopped and starts again at the end of the sound. For these reasons, we chose to use a non-blocking mode of the ADC

and a non-continuous conversion. To describe the code to execute when the conversion ends, we implemented the callback *void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)*.

2.3.4 PIR Interrupt

The Pir sensor was managed in interrupt mode to ensure a non-blocking mode when particular events occur. This sensor has been managed in "GPIO_MODE_IT_RISING_FALLING" mode to ensure that the interruption is generated whenever the sensor signal becomes low or high. Based on the value of the Pir signal, are managed different situations. The entire code to be executed, when the signal becomes high or low, is described in the callback *void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)*.

2.3.5 Timer Interrupt

All timers used in this program have been managed in interrupt mode to ensure that every situation that requires the use of a timer is non-blocking for the whole system.

Each timer has different functions and when one is triggered, with the exception of timer 3, it calls the callback *void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)*, where according to the timer that is triggered, and following several controls, the code to be executed is described.

Timer 3, used in PWM mode, starts when the alarm delay timer expires, the interruption of this particular timer does not call the previous callback, but the code to be executed when this timer is triggered is described in the callback *void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)*.

3 Hardware Architecture

This section is dedicated to explaining the sensor usage belonging to 37 Elegoo Kit[1] V 2.0.

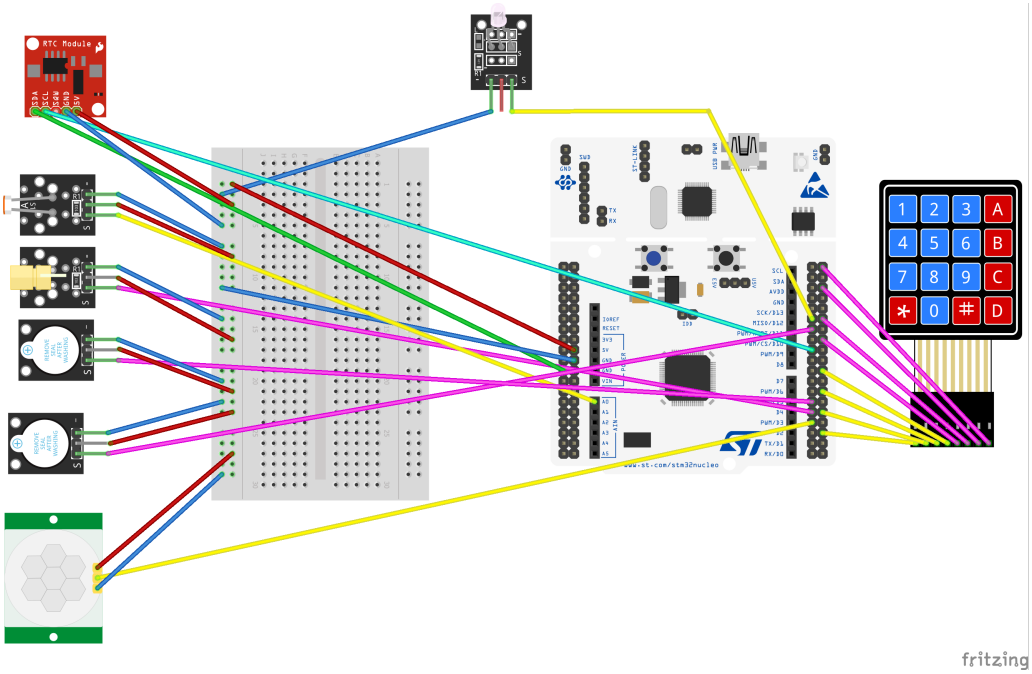

Figure 2: Hardware Architecture: in this picture you can see the hardware architecture created with Fritzing Tool

3.1 Active Buzzer Module

Pin Configuration:

1. "-": Ground
2. "+": 5V
3. "S": Signal input Pin PA6

We use this component to ring three different alarms thanks to the use of the PWM mode of Timer nr. 3.

3.2 Laser Module

Pin Configuration:

1. "S": Signal input Pin PB5
2. "-": Ground
3. "+": 5V

This component, togheter with the use of photoresistor, allows us to realize the barriern alarm.

3.3 Light Dependent Resistor Module

Pin Configuration:

1. "S": Signal output Pin, real-time output voltage IN0
2. "+": 5V
3. "-": Ground

Photoresistor is light sensitive device used to indicate the presence or absence of light, or to measure the light intensity. The output voltage is converted by the ADC in a numeric value. We have chosen a treshold value of 1000 after some experiments in different light conditions.

3.4 HC-SR501 PIR Sensor

Pin Configuration:

1. "+": 5V
2. "S": Signal output Pin PB3
3. "-": Ground

The Pin output is set with an Interrupt Mode managed by the IRQ Handler. In addition to the classic pins, there are also two potentiometers on the board to adjust a couple of parameters:
Sensitivity: This sets the maximum distance that motion can be detected. It ranges from 3 meters to approximately 7 meters.
Time Delay: This sets how long that the output will remain HIGH after detection. At minimum it is 3 seconds, at maximum it is 300 seconds or 5 minutes. The PIR sensor will detect infrared changes and if interpreted as motion, will set its output low. What is or is not interpreted as motion is largely dependent on user settings and adjustments. We decided to set both the potentiometers in counterclockwise. The device requires nearly a minute to initialize. During this period, it can and often will output false detection signals. In fact we have taken this into consideration setting.

3.5 DS-1307 Real Time Clock Module

Pin Configuration:

1. "-": Ground
2. "+": 5V
3. "SDA": Signal Pin PB7 (I2C)
4. "SCL": Signal Pin PB6 (I2C)

The DS1307 real-time clock is a low-power chip. Address and data are transferred serially through an I2C. *SCL* is used to synchronize data movement on the serial interface. *SDA* is the input/output pin for the 2-wire serial

interface. The SDA pin is open drain which requires an external pullup resistor. In our project RTC module is used to take information about year, month, date, day, hours, minutes, seconds. We find these information in the system log of the system [2].

3.6 4x4 Keypad Membrane Module

Pin Configuration:

1. "S": Signal output Pin PC4
2. "S": Signal output Pin PB13
3. "S": Signal output Pin PB15
4. "S": Signal output Pin PB2
5. "S": Signal input Pin PB12
6. "S": Signal input Pin PA12
7. "S": Signal input Pin PC5
8. "S": Signal input Pin PC8

Matrix keypads use a combination of four rows and four columns to provide button states to the host device. We have selected the first four pins (column) as push-pull output and last four pins (row) as pull-up input. In order for the micro-controller to determine which button is pressed, it first needs to pull each of the four columns (pins 1-4) either low or high one at a time, and then poll the states of the four rows (pins 5-8). Depending on the states of the columns, the micro-controller can tell which button is pressed.

3.7 Two Color LED Module

Pin Configuration:

1. "S": Signal output Pin PA5 - Yellow
2. "S": No output pin - Red
3. "-": Ground

This component is used to duplicate the user led of the board. We use it for visual feedback in case the card is covered or not visible.

3.8 IOC

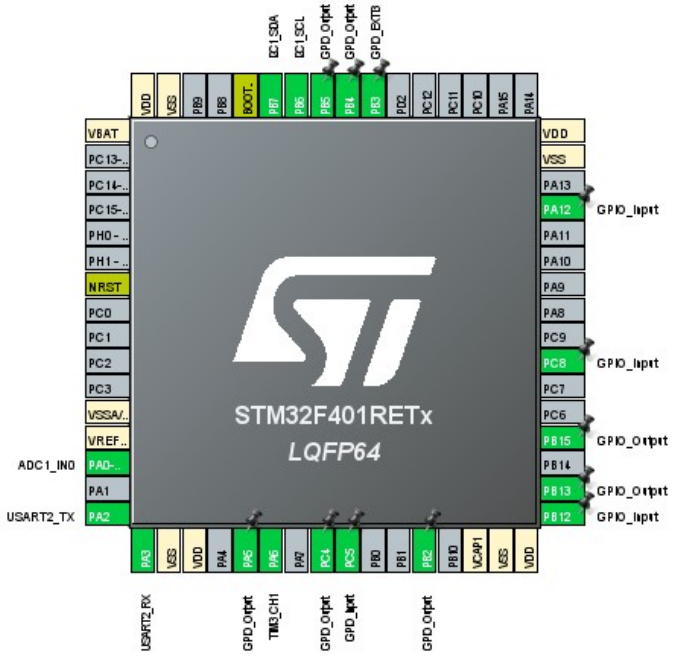

Figure 3: IOC: in this picture you can see the configuration of all pins we used.

4 Software Architecture

The software architecture is the basic organization of a system, expressed by its components, from the relationships between them and with the environment, and the principles that they guide the project and evolution. The architecture of a software system was defined in the first design phase, the architectural one. The primary purpose was to break down the system into subsystems: the realization of more distinct components is less complex than the realization of a system as a monolith.

IOC file: First, an initial configuration has been created through the service that the IDE makes available to us, that is, to graphically choose the interfaces and pins to be used, acting directly on the card. This configuration requires a lot of attention, because the IDE allows you to automatically create the code to initialize all the peripherals, so if during the development of the program you have to change configuration, and you want to do it via graphics, a new code will be generated and all the files automatically generated by the IDE such as "**main.c**" or "**tim.c**" will be overwritten, so if code was written in these files, it will be lost.

4.1 Source Package

The heart of the software is described in the **source package**, here we find a series of files where each file describes a different functionality of the system. A series of files in this package were automatically generated by the IDE at the time of configuration, all the files produced after configuration are described below.

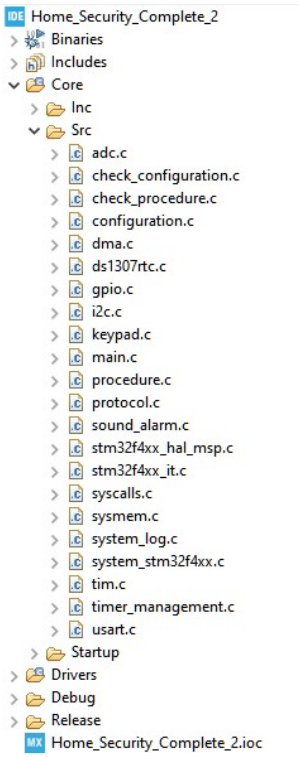

Figure 4: Source files: in this picture you can see the structure of source package and relative files.

- "**configuration.c**": the configuration to be used by the system is managed in this file. Here the interaction with the user begins, through the putty interface. First of all, a message is printed to prepare the user to interact with the system. Then you are asked to enter the pin to be used, the alarm delay, the duration, the date and the time. If no configuration is provided by the user, the system will use a default configuration. All these data are stored in a structure defined in "**configuration.h**".
- "**check_configuration.c**": this file carries out all the checks on the data entered during the initial configuration provided by the user. The pin must be 4 digits, the delays of alarms are max 30 seconds, the duration of the alarms is max 60 seconds. Finally, it is checked if date and time are valid.

- "**keypad.c**": in this file the user actions on the keypad are managed, special functions are implemented which allow to read precisely what the user presses on the keypad, accurately managing the button "bouncing" function. In addition, is implemented a function that completely returns the 4-digit pin. A buzzer sound is associated with each key pressed to communicate to the user that the character has been typed correctly.

- "**procedure.c**": The sequence entered by the user to activate/deactivate the system or sensors is managed in this file. The procedure starts when the user enters the "#" character on the keypad, then enters the pin, and finally the symbols, to activate/deactivate the system or the alarms and the system status, are updated in a defined structure of the "**procedure.h**" file. Based on the sequence entered by the user, very accurate checks are made to fully comply with the project specifications.

- "**check_procedure.c**": In this file are managed all the controls that regulate the sequence entered by the user. In particular, it is checked if the pin is the same chosen during the initial configuration. Also, if the sequence is valid or not. For all of these checks, there is a printed message that informs the user.

- "**timer_management.c**": the timer functions are managed in this file, two callbacks are implemented:

  - *void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)* describes everything that must be performed when a particular timer expires (except Timer 3);
  - *void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)* which is called when Timer 3 starts, in which the functionality of emitting different sounds is implemented with respect to the alarm that is triggered using the timer in PWM mode.

- "**system_log.c**": System log messages are managed in this file, two functions are implemented. The first takes the date and time from the structure defined in the "**ds1307rtc.h**" file and the date message to be printed on *Putty* is formatted. In the second function, checks are made on the state of the system and, based on it, a message is formatted to

be printed on *Putty*, concatenating it with the message of the previous function.

- "**protocol.c**": in this file the system behavior is managed when the PIR sensor signal becomes high or low. This sensor has been configured in interrupt mode and in *RISIG_FALLING* mode, so when an interruption is generated, the code to be executed is described in the callback *void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)*. Furthermore, since the ADC has been managed in interruption mode, another callback is also implemented within this file which is called when the analog-to-digital conversion of the photo-resistor ends. The callback is *void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)* and it is very important to understand when the virtual barrier is crossed by an attacker.

- "**sound_alarm.c**": different sounds for different alarms that are triggered, are managed in this file. Two functions are implemented, the first checks are performed to understand which sensor is alarmed and the second function sets the *Prescaler* and period parameters of Timer 3, managed in PWM mode, in order to have different duty cycles to emit different sounds in based on the alarm and then the alarm duration timer starts.

- "**ds1307rtc.c**": this is the driver to be able to use RTC (Real Time Clock). In this file, there is an init function to initialize the device and understand when it is ready to exchange data, registers are defined to understand where to write or read data from the RTC. Two functions are implemented for converting data from **bcd** to decimal and vice versa and two other functions to set the date and time in the RTC and to read the data from the RTC. As the data to be read are frequent, every 10 seconds, the reception of the data was managed in an interruption mode, using DMA, and then the callback *void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)* was implemented to describe the code to be executed when the data reception is completed.

4.2 Header Package

The header files are often used to define the physical layout of program data, pieces of procedural code and/or forward declarations while promoting encapsulation and the reuse of code.

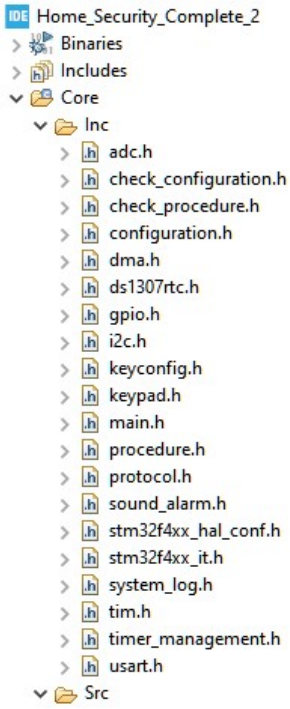

Figure 5: Header files: in this picture you can see the structure of header package and relative files.

In this package there are all the files with extensions **.h** which contains C function prototypes and macro definitions to be shared between several source files. There are two types of header files: the files that the programmers write and the files that come with compiler. In some of them, we defined also the structures to properly collect the data and organize the information:

- "configuration.h" : **system_config_s**
- "ds1307rtc.h" : **date_time_s**

- "keypad.h" : structure of the Keypad **keypad_t**, composed by Column, Row and LastKey
- "procedure.h" : **system_status_s**
- "sound_alarm.h" : definition of type "**TState**": AREA_ALARMED, BARRIER_ALARMED, AREA_BARRIER_ALARMED.

5 Software Interfaces

In this project several software interfaces have been used to carry out a data or signal exchange between the various modules and to ensure that the user is able to communicate and interact with the system.

5.1 Putty

Output interface
The use of Putty is fundamental for the interaction between the system and the user. This software is a terminal emulator and has been used for message management and communication. In particular, Putty is set in "serial" mode on the COM port that represents the connection with the micro-controller. The reception and sending of messages with the micro-controller is done with an appropriate speed of communication, chosen during the design phase. This value represents the "baud rate", that is the number of symbols that are transmitted on the communication channel in a certain time interval. During the implementation phase it was chosen as 9600 to have a low error probability.

5.2 User Led

Output interface
The micro-controller is equipped with a green LED (User Led), which indicates the global state of the system. The situations shown are three:

- The led is off, during the phase boot of the system.
- After the boot, if the system is not active, the led is on.
- While the system is active, the user led is blinked periodically.

5.3 KeyPad Module

Input interface
This 16 button keypad provides a useful human interface component for the project. In fact, during the execution of the software, it is required to enter commands, pins and codes that can be sent to the system through this interface.

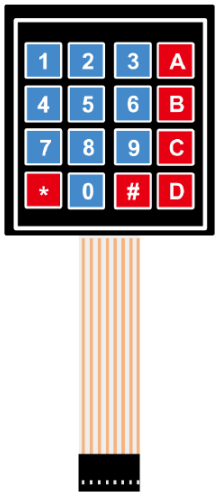

Figure 6: Keypad: in this picture you can see the keypad membrane module used for the communication between user and system.

5.4 Buzzer

Output interface
For a better user experience, has been added a Buzzer that emits a sound when a key is pressed on the Keypad module. This interface allows the user, to understand when the command is actually inserted. And also, after the boot, using the Buzzer, the system confirms the various operations through a short sound.

6 Software Protocols

To allow proper communication between the various components of our system, have been used appropriate software protocols. A protocol can be defined as a set of modes or rules of interaction that two or more entities, connected to each other, must comply with in order to operate.
In our case, given the need for a Serial communication between PC and micro-controller, we used the UART/USART protocol to insert an initial system configuration, and I^2C to perform a serial transfer of information between the micro-controller and the RTC component. Finally, for the correct use of the sensors, the ADC analog-to-digital signal conversion protocol was used.

6.1 UART

UART is an information transmission mode in which receiver and transmitter are not synchronized during communication. This allows transmission and reception at your own pace: the transmitter cannot be at the same speed as the data receiver. Within our project, this protocol regulates the exchange of information between the PC and the micro-controller, during the first configuration phase, where the system is initialized with appropriate data entered by the user. In particular, through the Putty software interface, a serial communication channel is established with the micro-controller. The speed between the two entities has been set to 9600 bits per second, which is the symbol velocity that affects the rate of transmission on a data channel (Baud Rate).

Figure 7: Uart: in this picture you can see a scheme of UART communication.

6.2 I²C

For a proper functioning of the system, it is necessary to use the RTC (Real Time Clock) to obtain information about date and time. The setting of this information takes place during the initial phase, through the Putty interface that interacts with the user. Then the UART protocol transmits the information to the micro-controller and finally the micro-controller transfers it to the RTC with I^2C protocol.
The I^2C protocol requires two serial communication lines:

- SDA (**S**erial **DA**ta) for data.
- SCL (**S**erial **CL**ock) for the clock (the presence of this signal indicates that the I^2C is a synchronous bus.

As mentioned above, the I^2C is a bus with a clock (SCL) and a data line (SDA) and 7 possible bits of addressing. A bus has two types of nodes:

- master node - the device that emits the clock signal
- slave node - the node that synchronizes on the clock signal without being able to control it.

This way you can read and adjust RTC.

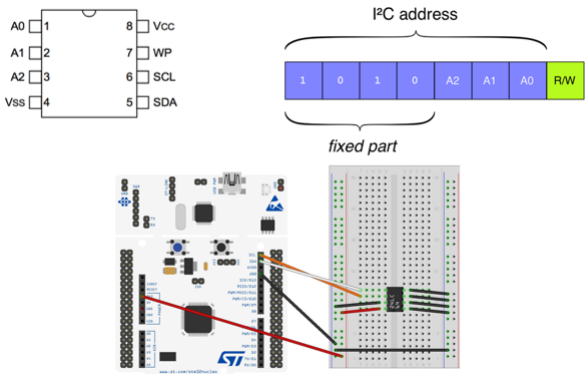

Figure 8: I^2C: in this picture you can see a scheme of communication between RTC and MCU.

6.3 ADC

Analog to digital converter allows the micro-controller to accept an analog value like a sensor output and convert the signal into the digital domain. The analog signal can vary from 0 volts to a maximum voltage level that is called "reference voltage". In order to realize a digital signal "SET" (1) or "RESET" (0), was chosen a threshold value after several experiments. If the analog signal is more than this value, the output is 1, otherwise the output is 0.

ADC is implemented as a 12-bit Successive Approximation Register ADC It can have a variable number of multiplexed input channels:

- External Sources
- Internal Sources

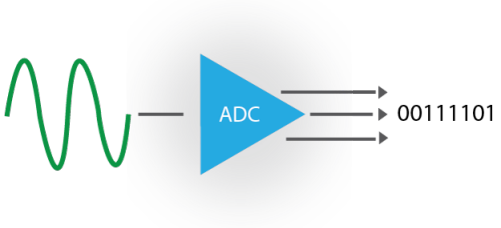

Figure 9: ADC: in this picture you can see a simple scheme of ADC converter.

A/D conversion of the various channels can be performed in single, continuous, scan or discontinuous mode:

- Single Conversion Mode
  - Single Channel
  - Multi Channel
- Continuous Conversion Mode
  - Single Channel
  - Multi Channel
- Injected Conversion Mode
- Dual Modes

7 Conclusions

During the system design phase, implementation choices were made which were the result of our knowledge learned during the study on Embedded Systems. The final system, respects the specific requests and has been tested first, taking individually the components that compose it, and subsequently, in its final stage. The user who uses the system is provided with the feedback necessary to understand its operation and to keep track of the status of the device in real time through a "System Log", which, through the Putty interface, shows the condition of the sensors and alarms at regular time intervals.

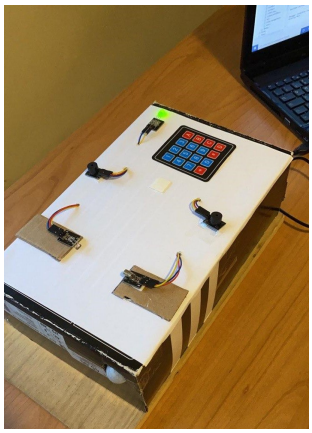

Figure 10: Final Device: in this picture you can see a craft model of the system.

The initial configuration occupies an important role for the operation of the entire device, since during this phase all the data necessary for setting the system parameters configuration. In addition, a User Pin (4-digit code) is also required, which the user must enter whenever a new command is given to the system. If the configuration is not provided within 30 seconds, the system will start using the basic configuration. Except for transmitting the configuration at boot time, each command sent to the system must be started using the user pin. If the pin sent does not match the one provided in the configuration, the command is rejected. In conclusion, the "Home Security System" project was brought to an end, following the implementation choices justified largely by the specifications required by the deliverable.

References

[1] ELEGOO. 37 sensor kit tutorial for uno v2.0.

[2] DALLAS Semiconductor. Ds1307 serial real time clock module.