VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Xavier Garcia Peroy

Software design for a Smart Lock System for Home Automation

School of Technology
2016

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Program in Information Technology

ABSTRACT

Author	Xavier Garcia Peroy
Title	Software design for a Smart Lock System for Home Automation
Year	2016
Language	English
Pages	39
Name of Supervisor	Smail Menani

The idea that motivated this thesis work was to design a smart lock system and access control. This system was designed to control the access to a room using portable devices such as mobile phones or tablets. Its principal objective was to create a basic, but functional prototype that could be placed in a real room, and be implemented and integrated easily without much modifications of the doors' structure, yet accomplish its purpose.

To achieve this goal the work was divided in two separate parts the hardware part and the software part. This thesis focuses on the Software part. All progresses made and final results were obtained after using a design, implementation and test method. Working closely and in constant feedback with the hardware part, the programs and code developed has been always tested with the hardware to ensure that the right results were obtained and required modifications were made if needed.

In general, the project was carried out as follows. First the documentation process was completed and then, the devices, the coding languages and tools were selected. Next, the designing process was accomplished where the basic structure of the programs was developed and created. After that, the implementation and testing process were simultaneously done to perfect the result and obtain the best solution possible.

This process was completed when the real working prototype was obtained. The device successfully connected to a self-designed smartphone app and controlled the access to the room. The prototype could be mounted on a demonstration door to show how it is supposed to work in the real application. In general, all the main goals were achieved although some rectification and improvements are still to be studied more in depth and be applied on future expansions.

CONTENTS:

LIST OF FIGURES AND TABLES:

1. INTRODUCTION

The thesis underlying idea is based on access control and physical area automation system. The physical area could be for example a laboratory, an office or a house. The developed smart lock system is meant to control and interact with a door to access a room using a smart device as interface. The principle idea is to develop an electronic system that can control the door state and modify it, using a personal electronic device such as a smartphone or a tablet.

The thesis project main goal is to try to create a functional prototype that can be installed in a real door and successfully allow the user to interact and modify the door's state using a smart device. This solution should be secure, cheap and simple, as well as versatile to allow implementation on different actuators and physical areas.

The door lock developed in this thesis is a project carried out by two students. Therefore, the reporting work has been split in two different parts: the software part of the system and the hardware part. This thesis work focuses on the software part of the smart door lock including the android application

This work is divided in several different chapters:

Chapter 2 focuses on the programming theory, the bases of the elements of the thesis and on the analysis carried out to find the components and the most convenient programming methods for the best outcome.

The following chapter, Chapter 3, describes and explains the code implemented on the door lock and explains the logic behind the developed software solution.

Chapter 4 develops the idea of the android app, explains how it works, its features and possible upgrades.

Chapter 5 explains the system tests carried out through the developing process to check the functionality of the smart lock device. Further, it describes the results observed after the testing controls.

The last chapter, Chapter 6, presents the conclusions of this thesis and points out possible extensions and improvements that could be added to the system.

2. THEORETICAL STUDY AND ANALYSIS

Before starting the design of the code and its implementation, it is important to analyse the problem to be solved and to figure out the best elements to complete the work. From the software point of view it attention needs to be paid on how these elements will communicate, transfer the desired data between them to control the system. The elements chosen must fulfil all requirements of the work, yet allowing including new features to be added to this initial system. Another important characteristic to analyse are the offered facilities apart from the program to have access to useful references and guides. The best possible elements for this project should be simple to use though powerful enough to reach the thesis purpose.

The basic structure from the software point of view of this work is shown in the following diagram:

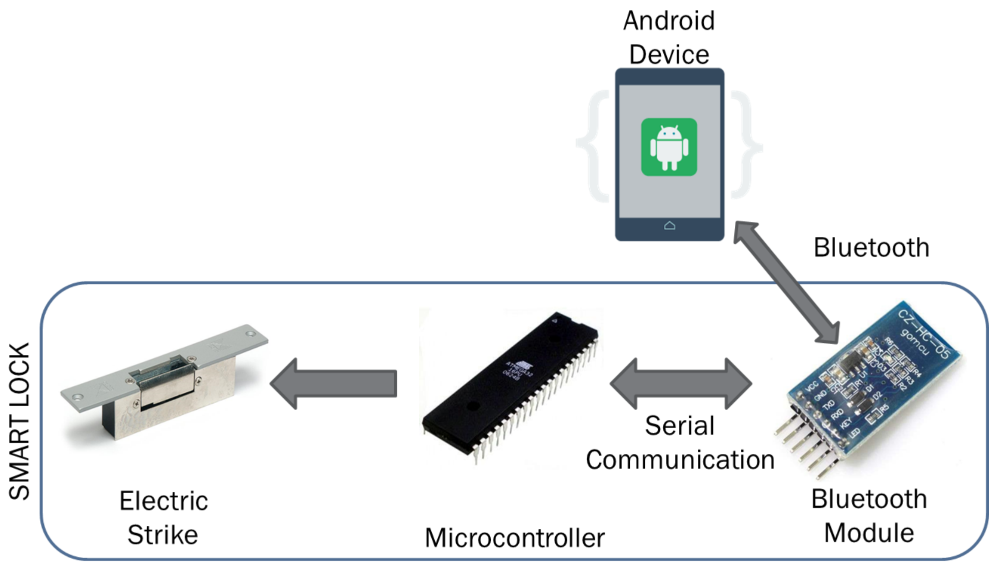


Figure 1: Structure of the system

As it is shown, two important devices control the smart lock: the microcontroller and the Bluetooth module. They have to be carefully selected to fulfil the requirements of the work. The most important aspects to while selecting are to find and to understand the communications between them as well as its simplicity and performance in an embedded compact system. The elements must be also easily accessible in the market and none expensive to create an easily repeatable system that can be implemented in the cheapest way possible.

2.1 The microcontroller Atmega32:

The microcontroller selected was the Atmega32 of the AVR family. The principal advantages of this microcontroller compared to other possibilities such as 8051, PIC or ARM families microcontrollers is that their performance is really good for a reasonable low price and that it is one of most used microcontrollers for embedded systems design and implementation. This fact generates a big community of developers, guides and references behind it, really useful for starting working with it. This family of microcontroller is completed with great free developing tools that make the software design and the code implementation really simple and smooth.

The basic characteristics of this chip are described in the introduction chapter on the microcontroller's Datasheet:

> "The ATmega32 provides the following features: 32Kbytes of In-System Programmable Flash Program memory with Read-While-Write capabilities, 1024bytes EEPROM, 2Kbyte SRAM, 32 general purpose I/O lines, 32 general purpose working registers, a JTAG interface for Boundaryscan, On-chip Debugging support and programming, three flexible Timer/Counters with compare modes, Internal and External Interrupts, a serial programmable USART, a byte oriented Two-wire Serial Interface, an 8-channel, 10-bit ADC with optional differential input stage with programmable gain (TQFP package only), a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and six software selectable power saving modes." (ATmega32(L) Complete Datasheet, page 4)

Furthermore, this microcontroller was suitable and appropriate for this project because it offers many facilities when trying to work with a Bluetooth module. As it is so widely used, a lot of manufacturers design and produce modules that are specifically meant to communicate via USART serial communication with this kind of chips and therefore, are very adaptable to this microcontroller in particular. In general the AVR microcontroller has all the aspects required in this work

and more, hence it can complete all the requirements and allow future expansions to be made without the need to change the ATmega32 or any other device.
For this thesis, the microcontroller programming will focus mainly on three modules or aspects of the device: I/O pins configuration and handling, USART serial communication and Interrupts.

- I/O pins configuration and handling:

For this application there will be at least 6 output pins required: one input pin for the interruption and two for the serial communication. The device has 44 Pins, however only 32 of them can be used as input or output pins. These pins have a highly important role in this project, as they control, for example, the mode of the Bluetooth module, the door strike state or the notification LEDs among others. These output pins are the reason how the microcontroller, more precisely how the whole smart lock system controls the peripheral devices acts and states that will give information and interact with the user, making the desired actions possible e.g.: to unlock the door.

- USART:

USART stands for Universal Synchronous Asynchronous Receiver/Transmitter. This is the default mode normally used by the AVR microcontroller. Changing the values in some of the calibration data in the EEPROM in the Atmega32 serial module can work either in USART mode or in UART mode (Universal Asynchronous Receiver/Transmitter). The main difference between both serial communication modes is that the Synchronous method has its transmitted data clocked. The clock is either recovered from the data itself or sent as an external signal. The transferred data synchronizes with system clock system, allowing the communication run under higher baud rates making all the communications securer thanks to the guaranteed timing synchronizations.
This serial communication hardware part of the microcontroller allows the AVR to transmit and receive data serially to and from other devices. Its principal characteristic is that it does not require a separate PIN for the serial clock. An agreed baud rate is pre set in both devices and used to sample the read and transmit lines.

Normally USART communications use the RS_232 logic levels and control signals to communicate with other devices, therefore using the normal AVR Pin logic levels from 0V to 5V converters should be used. In this application this conversion will not be required between the AVR microcontroller and the Bluetooth module, because both use the AVR pin logic levels and they can be directly connected.

To allow the interfaced devices to properly communicate, a suitable pre set baud rate must be found. The baud rate is the rate at which information is transferred in a communication channel. In the serial port context, "9600 baud rate" means that the serial port is capable of transferring a maximum of 9600 bits per second (presetting that every baud equals a bit).

It is important to have a known and stable system clock. The USART clock is derived from the AVR's system clock and is used to sample the Rx line and set the Tx line at precise time intervals in order to maintain the communication.

On the ATMEL's Atmega 32 datasheet provides a table of common baud rates, system clocks and error percentages, which are a useful and helpful guide in order to choose a specific pair of values.

- Interrupts:

Interrupts allow external events, such as inputs coming from the I/O pins or peripheral devices, or internal events (such as counter temporizations, etc.), to trigger a certain group of instructions at any required time. These events require the immediate attention of the microcontroller. If one of these interruption events occurs, the microcontroller stops its current executing instructions and executes an Interrupt Service Routine (ISR), which gathers the code required for that particular event. Once this routine is concluded the microcontrollers gets back to its previous step in the previous executed code.

2.2 HC – 05 Bluetooth module:

The other important device in this smart lock is the Bluetooth module. The module chosen is a developed and integrated module named HC – 05 Bluetooth SSP module. This module is specifically designed to work with Arduino boards or microcontrollers and it is implemented in a way to become easy to use outside the box.

It consists of a fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with a complete 2.4GHz radio transceiver and a baseband. The module incorporates a Cassira CSR Bluecore 04-External Chip Bluetooth system with a CMOS technology and an Adaptive Frequency Hopping Feature.

The most remarkable Software features are:

- Its default baud rate is set at 38400 bauds/second, with 8 data bits, one stop bit and no parity bits. It supports 9600, 19200, 38400, 57600, 115200, 230400, 460800 baud rates.
- Auto-connection to the last device on power as default setting.

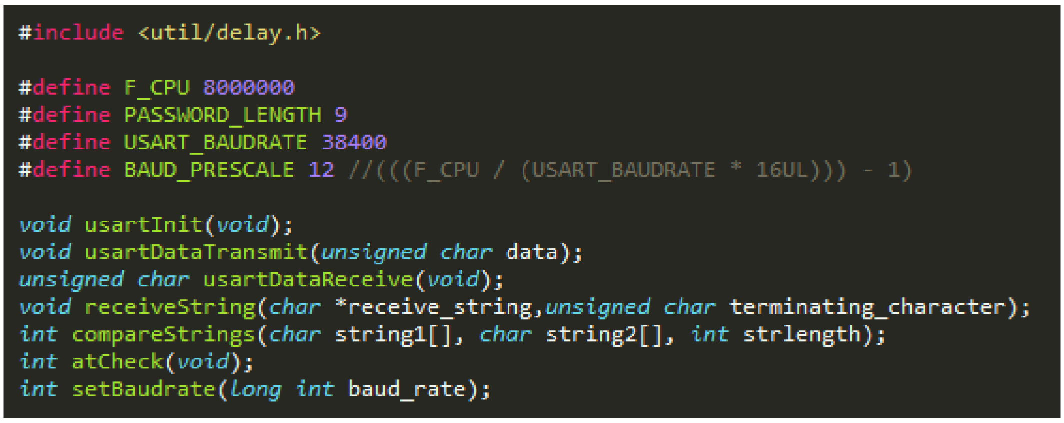
Using the Bluetooth module AT mode completes the configuration of the module. This mode allows the user to send specific formatted commands that change internal parameters of the module allowing him to set them as required. To start the module under the AT mode, the Key pin (Actually the PIO11 on the chip) must be powered before the VCC that powers up the chip is set to HIGH. Once this mode is initialized the AT commands can be send. All the AT commands must be followed by "\r\n" in order to perform a successful configuration of the module. If the command worked properly the module is going to answer with "OK\r\n" in case of f commands not ending with a question mark. For this kind of question or information in demanding commands the module will respond with the demanded information followed by "OK\r\n".

3. IMPLEMENTATION

As stated earlier, Atmega32 microcontroller from Atmel is selected to implement the embedded system of the smart lock.

There are three main parts on the program: the basic configurations and initializations, the AT configuration instructions and the smart lock mode.

All these parts use functions defined in the library called "functions.h", developed to make the programming procedure simple and to set a clear and easily understandable structure to the main program. Some of these functions in this header file are based on the Atmega32 programming method explained on the previous mentioned Atmega32's datasheet or in other resources given from the company for the USART communication. Other functions are designed to solve specific problems that are needed due to the characteristics of the problem. The rest are used in the AT configuration mode to send the appropriate commands to the BT module. All of them are important in some part, yet some of them need one another to make the solution run appropriately. In this header file are some global constant variables definitions such as the microcontroller working frequency; FCU set to 8MHz, the baud rate and others.



```
#include <util/delay.h>

#define F_CPU 8000000
#define PASSWORD_LENGTH 9
#define USART_BAUDRATE 38400
#define BAUD_PRESCALE 12 //(((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

void usartInit(void);
void usartDataTransmit(unsigned char data);
unsigned char usartDataReceive(void);
void receiveString(char *receive_string,unsigned char terminating_character);
int compareStrings(char string1[], char string2[], int strlength);
int atCheck(void);
int setBaudrate(long int baud_rate);
```

Figure 2: Functions.h example

Before explaining the code in detail a flowchart of the basic algorithm of the programming code is included to have an overview of the system operation:

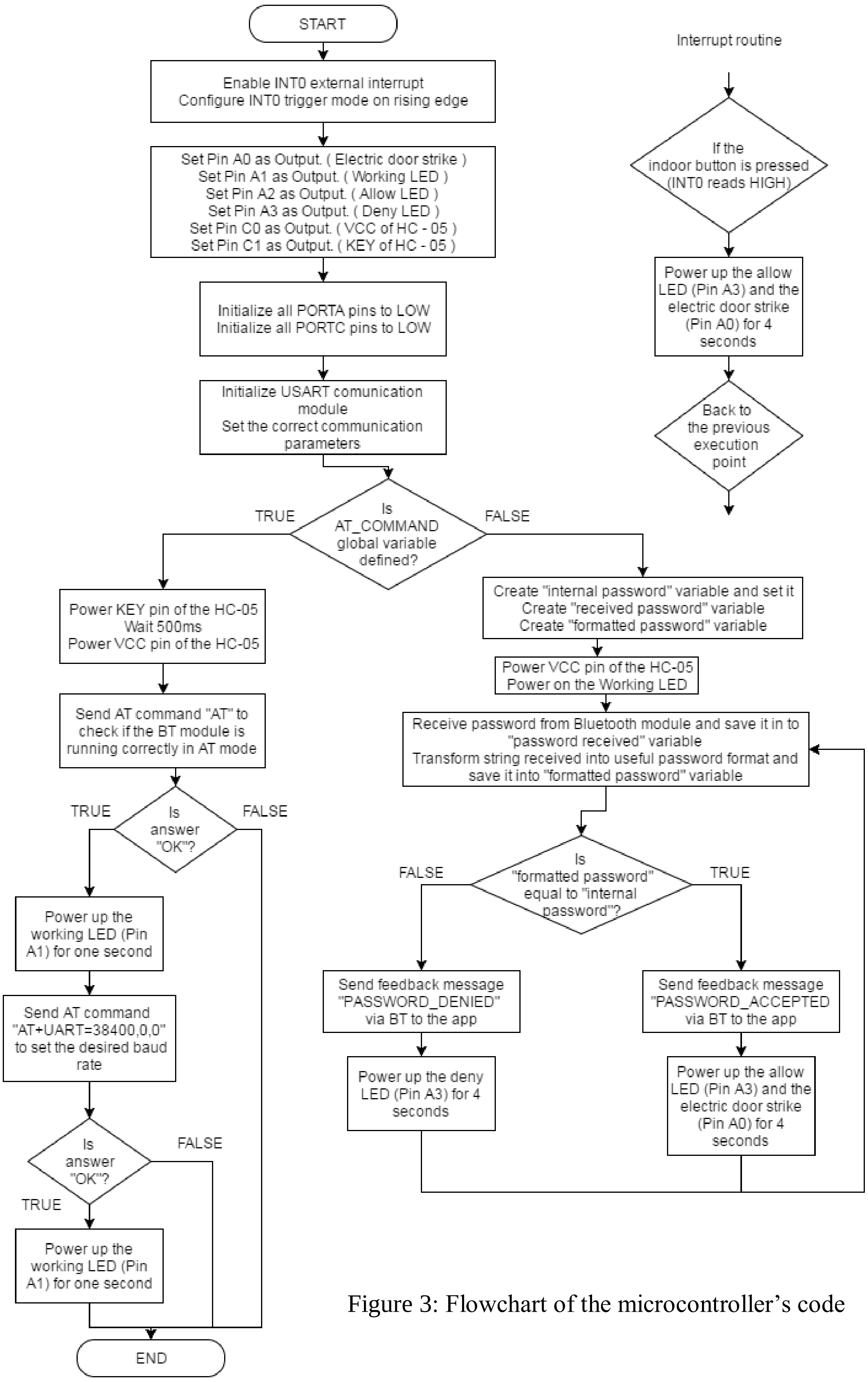

START
Enable INT0 external interrupt
Configure INT0 trigger mode on rising edge
Set Pin A0 as Output. (Electric door strike)
Set Pin A1 as Output. (Working LED)
Set Pin A2 as Output. (Allow LED)
Set Pin A3 as Output. (Deny LED)
Set Pin C0 as Output. (VCC of HC - 05)
Set Pin C1 as Output. (KEY of HC - 05)
Initialize all PORTA pins to LOW
Initialize all PORTC pins to LOW
Initialize USART comunication module
Set the correct communication parameters
Is AT_COMMAND global variable defined?
TRUE
FALSE
Power KEY pin of the HC-05
Wait 500ms
Power VCC pin of the HC-05
Send AT command "AT" to check if the BT module is running correctly in AT mode
Is answer "OK"?
TRUE
FALSE
Power up the working LED (Pin A1) for one second
Send AT command "AT+UART=38400,0,0" to set the desired baud rate
Is answer "OK""?
FALSE
TRUE
Power up the working LED (Pin A1) for one second
END
Create "internal password" variable and set it
Create "received password" variable
Create "formatted password" variable
Power VCC pin of the HC-05
Power on the Working LED
Receive password from Bluetooth module and save it in to "password received" variable
Transform string received into useful password format and save it into "formatted password" variable
Is "formatted password" equal to "internal password"?
FALSE
TRUE
Send feedback message "PASSWORD_DENIED" via BT to the app
Power up the deny LED (Pin A3) for 4 seconds
Send feedback message "PASSWORD_ACCEPTED via BT to the app
Power up the allow LED (Pin A3) and the electric door strike (Pin A0) for 4 seconds
Interrupt routine
If the indoor button is pressed (INT0 reads HIGH)
Power up the allow LED (Pin A3) and the electric door strike (Pin A0) for 4 seconds
Back to the previous execution point

Figure 3: Flowchart of the microcontroller's code

3.1. Basic configurations and initializations:

The code starts with some initializations of variables and registers and the modules needed in the application. The next schema shows their order and all the initialization required:

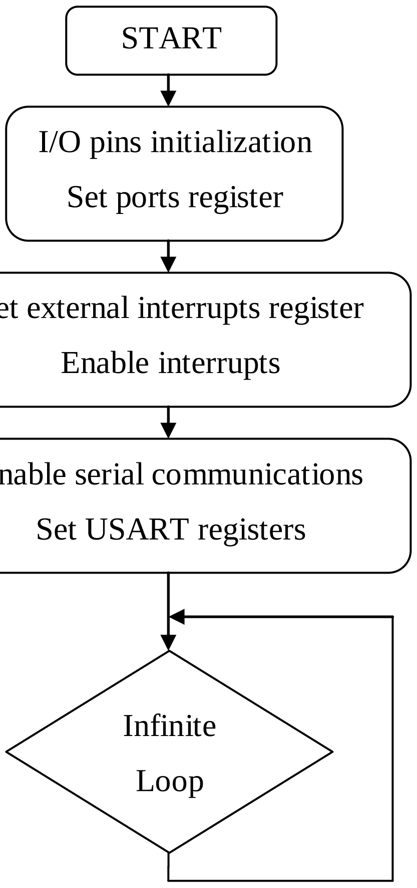


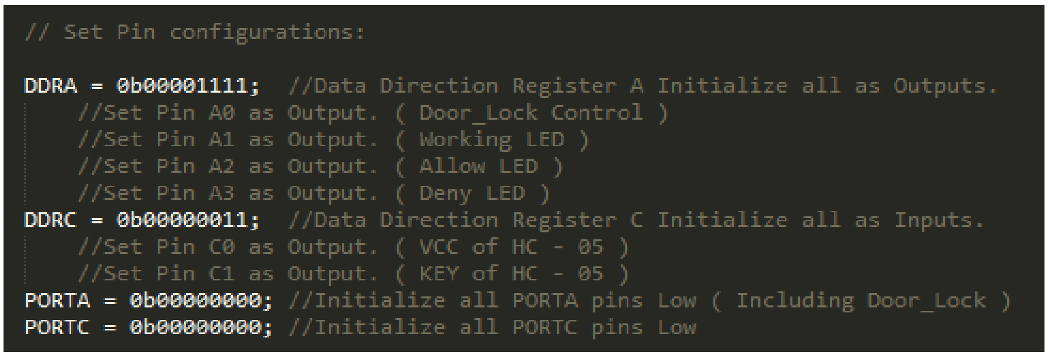
Figure 4: Basic configurations and initializations structure

The first step in this configuration and initialization process is to set the I/O pins and Ports registers. The system has six output pins and no input pins, because all the inputs are used in the USART communication or in the external interrupt for the inside button. The six outputs are divided between PORTA where the electronic switch that controls the door strike is located, pin PA0, the three notification LEDs, power LED on PA1, access permitted LED on PA2 and access denied LED on PA3, and PORTC where the power Pin of the HC – 05 module and the KEY pin are connected.

To set one of the microcontrollers pin as an input or/and output the Data Direction Register of every Port should be set. A “1” will set the pin as output while a “0” will leave it as input. Once the DDR is initialized, it would be interesting to set all outputs to a low level to have a clear starting point for all the peripherals. To do that modification the port pin configuration of both pin sets, A and C, is required. “1” will set the output pin value to high and “0” will set it low. To begin all pins will be set to low.

This method to set output pins low and high will be used throughout the code to control all the peripherals such as the door strike or the notification LEDs.

Next picture depicts how it is programmed in the solution:



```
// Set Pin configurations:

DDRA = 0b00001111;  //Data Direction Register A Initialize all as Outputs.
    //Set Pin A0 as Output. ( Door_Lock Control )
    //Set Pin A1 as Output. ( Working LED )
    //Set Pin A2 as Output. ( Allow LED )
    //Set Pin A3 as Output. ( Deny LED )
DDRC = 0b00000011;  //Data Direction Register C Initialize all as Inputs.
    //Set Pin C0 as Output. ( VCC of HC - 05 )
    //Set Pin C1 as Output. ( KEY of HC - 05 )
PORTA = 0b00000000; //Initialize all PORTA pins Low ( Including Door_Lock )
PORTC = 0b00000000; //Initialize all PORTC pins Low
```

Figure 5: Pin configurations

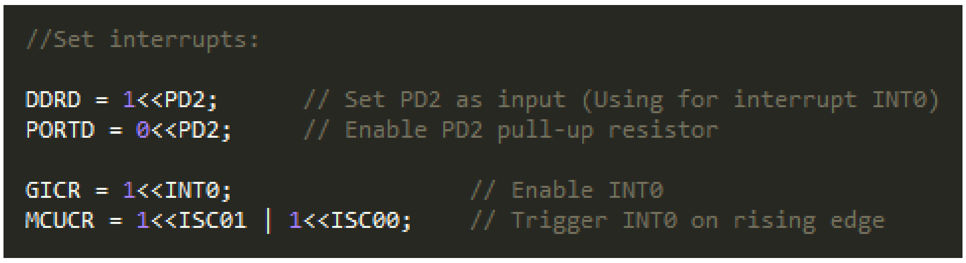
After the I/O pins have been configured, next step is to allow interrupts. An external interrupt is used to allow a button inside the room to open the door every time it is pressed. This button is connected to INT0 interrupt pin (PD2). First step is to include the AVR interrupt header, “avr/interrupt.h”, which has the required defined ports and registers and a function necessary to work with interrupts. After this and using the same procedure as before, pin PD2 must be set as an output.

Once PD2 is set, the next step is to allow external interrupts registers and fuses to be accessible and modifiable; for this the General Interrupt Control Register must be set to 1. It is important to configure this when the interrupts will trigger. In the following all the possible options are presented:

- The low level of INT0 generates an interrupt request.

- Any logical change on INT0 generates an interrupt request.
- The falling edge of INT0 generates an interrupt request.
- The rising edge of INT0 generates an interrupt request.

The hardware structure of the system sets the resting state of the INT0 pin as LOW. The best option therefore would be to set the interrupt trigger on the rising edge; this will make the door open every time the button is pushed. This selection complemented by a debouncing capacitor implemented on the hardware part will make the button reliable. For selecting this option the datasheet of the microcontroller provides the next register configuration: set ISC01 and ISC00 to “1”.



```
//Set interrupts:

DDRD = 1<<PD2;       // Set PD2 as input (Using for interrupt INT0)
PORTD = 0<<PD2;      // Enable PD2 pull-up resistor

GICR = 1<<INT0;                    // Enable INT0
MCUCR = 1<<ISC01 | 1<<ISC00;    // Trigger INT0 on rising edge
```

Figure 6: Interrupt configurations

Last configuration step for external interrupts is to call a function placed at the AVR interrupts library called “sei()”. This function is a method implemented by Atmel that enables global interrupts on the system and allows the interrupts to trigger the ISR code whenever needed.

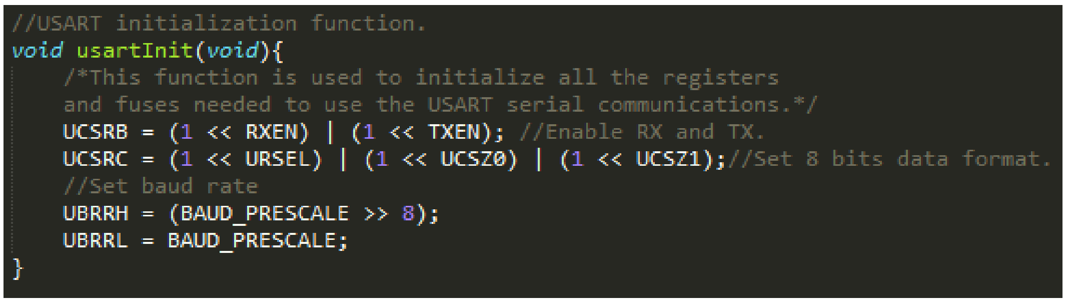
After all pins have been configured and the external interrupt for the indoor button has been initialized, the last module to be enabled is the USART communication. The first issue to settle when starting with this module is to decide the configuration parameters of the module and the serial communication. They are settled as: 38400 Baud rate, 8 data bits, 1 stop bit, and 0 parity bits. Once these values are known, the Atmega32 microcontrollers provides a highly interesting table that shows the error margin and the UBRRH/UBRRL registers value for every pair of values of baud rate and the oscillator frequency.

The combination chosen is:

- Baud rate: 38400.
- Oscillation frequency: 8MHz.
- UBRR: 12.
- Error: 0,2%.

To get the configuration chosen, some registers must be modified and configured beforehand. In order to set the 8 data bits, the registers UCSZ0 and UCSZ1 must be set to one, as well as the register USEL to allow writing all the registers. After that, the Rx and Tx communication lines must be enabled. This is achieved by setting the RXEN and TXEN register to one. By default, the communication starts with no parity and one stop bit, so there is no need to make any modifications for these features to be on. All these configurations values for the registers and internal variables are described on the internal table on the Atmega32 datasheet's USART chapter included on the references of this work.

All these configurations are made using a function called usartInit():



```
//USART initialization function.
void usartInit(void){
    /*This function is used to initialize all the registers
    and fuses needed to use the USART serial communications.*/
    UCSRB = (1 << RXEN) | (1 << TXEN); //Enable RX and TX.
    UCSRC = (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1);//Set 8 bits data format.
    //Set baud rate
    UBRRH = (BAUD_PRESCALE >> 8);
    UBRRL = BAUD_PRESCALE;
}
```

Figure 7: USART initializations

3.2 AT configuration instructions:

This block of code is a very simple part of the program and it is only meant to be compiled and executed if, due to some hardware problems, the HC – 05 Bluetooth module must be replaced with a new one that has different pre set values. This block will just make the wireless communication device enter the AT mode, described in the theoretical discussions held in the previous chapter, and allow the microcontroller to modify some of its parameters. For now, there is only one parameter that needs to be modified: the communication baud rate. As explained before, if the microcontroller and the BT module have different baud rates the communications between them is impossible and consequently change the messages transmitted and received to senseless strings.

Therefore, the structure of this part of the code will remain as follows:

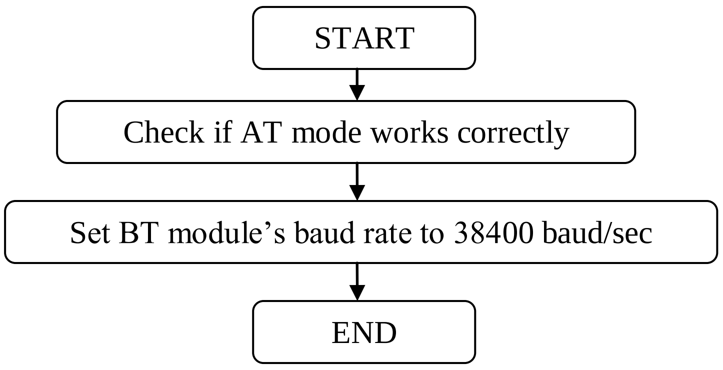
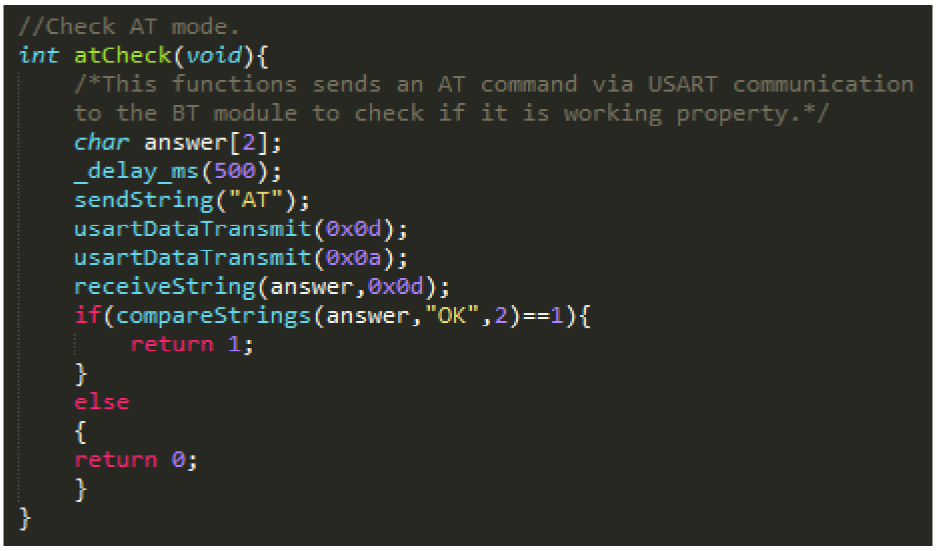

START
Check if AT mode works correctly
Set BT module's baud rate to 38400 baud/sec
END

Figure 8: AT mode structure

For both purposes a function inside the functions library has been created. Both functions have the same structure. They start sending the desired AT command in its required format, followed by "\r\n". For checking if the AT mode works correctly the command to be used is "AT". On the other hand, for setting the specific baud rate the AT command is "AT+UART=<baud_rate>,0,0". Then, the microcontroller waits for the answer of the module and if the message is: "OK", the function returns true, if not false.

In the main code this feedback is used to light up the working led for one second, allowing the user to see if the commands have been received correctly. This is done to receive feedback between the user and the device without requiring a seri-al monitor to follow the communication.

The next figure shows the function that sends the AT command to the module to check if it is working correctly:



```
//Check AT mode.
int atCheck(void){
    /*This functions sends an AT command via USART communication
    to the BT module to check if it is working property.*/
    char answer[2];
    _delay_ms(500);
    sendString("AT");
    usartDataTransmit(0x0d);
    usartDataTransmit(0x0a);
    receiveString(answer,0x0d);
    if(compareStrings(answer,"OK",2)==1){
        return 1;
    }
    else
    {
    return 0;
    }
}
```

Figure 9: AT check function

3.3 Smart Lock mode:

This is the main working mode of the application and the most important part. In this part of the code the working loop is defined and the automation of the smart lock is executed. A diagram of the working process is shown next:

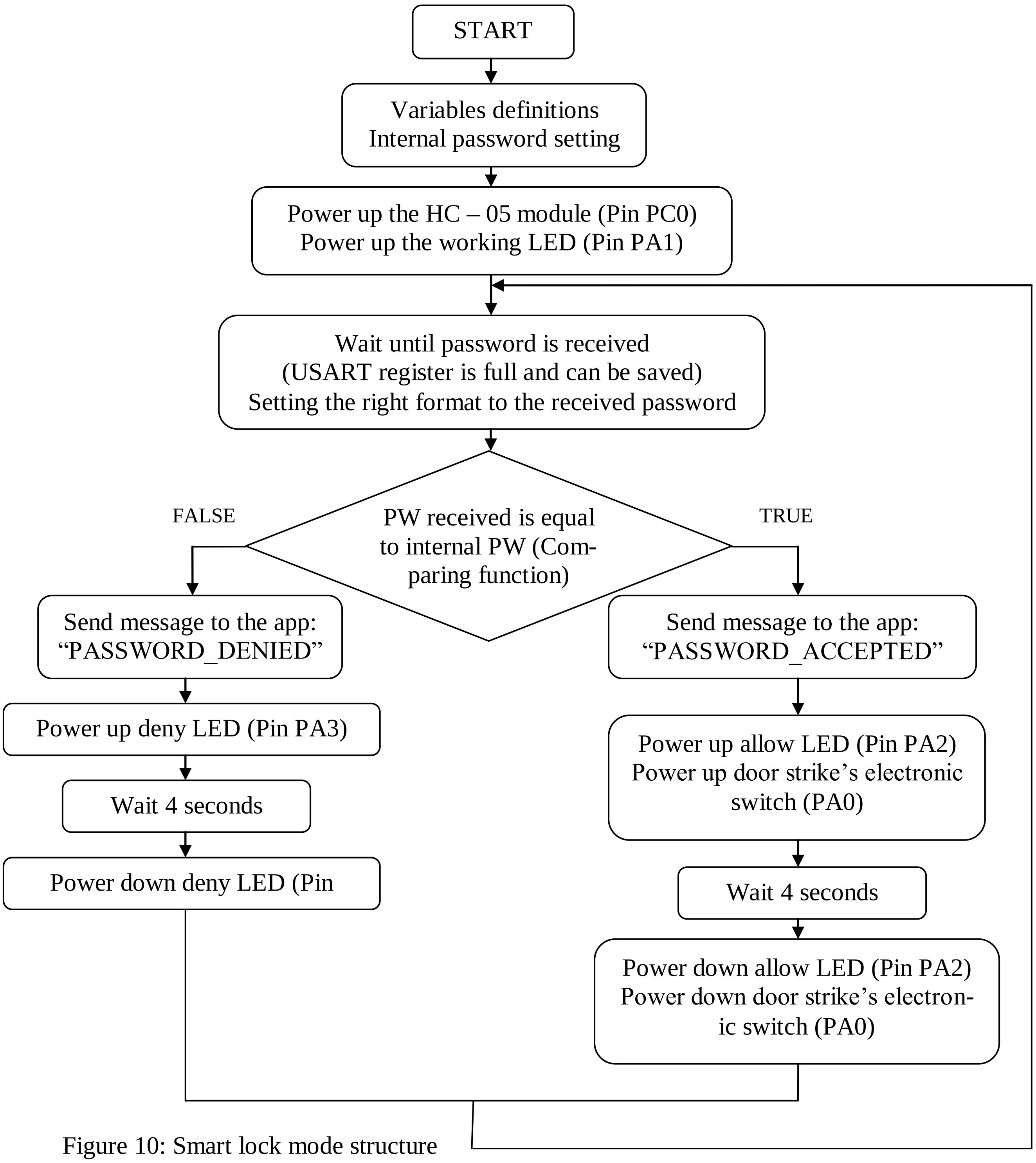

START
Variables definitions
Internal password setting
Power up the HC – 05 module (Pin PC0)
Power up the working LED (Pin PA1)
Wait until password is received
(USART register is full and can be saved)
Setting the right format to the received password
PW received is equal to internal PW (Comparing function)
FALSE
TRUE
Send message to the app: "PASSWORD_DENIED"
Power up deny LED (Pin PA3)
Wait 4 seconds
Power down deny LED (Pin
Send message to the app: "PASSWORD_ACCEPTED"
Power up allow LED (Pin PA2)
Power up door strike's electronic switch (PA0)
Wait 4 seconds
Power down allow LED (Pin PA2)
Power down door strike's electronic switch (PA0)

Figure 10: Smart lock mode structure

The Smart Lock mode starts creating helping variables to be used in the entire working mode: one string contains the internal password that will control and verify the permission on the door, and two more string variables will be used inside the loop for verifying purposes, as they will store the password received from the app to perform the authentication process.

After these variables are defined, the microcontroller starts powering up the working LED to let the user know that the Smart Lock mode is running. At the same time, it powers up the Bluetooth module to enter the normal functioning mode and allow communication via UART with the Atmega32.

Then, the infinite loop starts and as a result the application, too. The first function to be executed is the function designed to receive strings via the USART communication. This function uses an interrupt or register condition that stalls the Rx (USART reading line) register until RXC is complete to copy its content to a string. It continues doing this until the receiving buffer is no longer full and the entire message is received. The characters are received one by one and stored by using pointers into an array to contain the full message received.

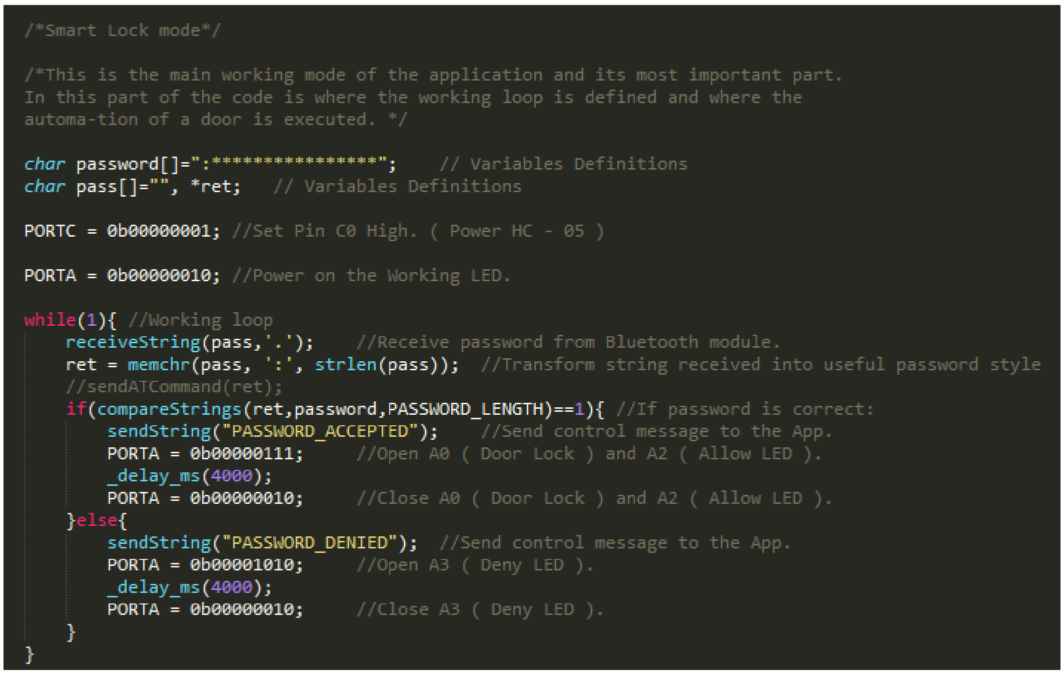
Then, once the full message received from the BT module is stored in the memory of the microcontroller, some functions from the predefined "string.h" library are used to ensure a suitable format of the message in order to be compared with the internal password of the system. This step is very important, as the received information format, using the UART functions, can change depending on the executing time and on the communicating devices. This extra information must be excluded from the received message to ensure a proper message format. For example, there could be "\r\n" sent after some messages. Thanks to this function the password information always has to start with the symbol ":" and finish with ".". The information between this two signs, ":" included but "." not, will be the one used in the verification process. This ensures that the message will always have the same kind of format and is recognizable and comparable for the microcontroller. If this step is lacked the communication can sometimes become messy and makes the system act erratically and randomly.

Once the message is received and well formatted, the comparison with the internal password is performed. Using a string comparing function the program decides whether the message received is correct and therefore, the door must be opened or the access denied. This comparison is carried out by a simple character-to-character comparison function that compares the internal password and the formatted received message.

If the comparison results are correct, the A0 output where the door strike is placed and the A3 output where the allow LED is connected are going to be powered to HIGH for 4 seconds. This time can be modified depending on the door and the characteristics of the room and users. Then, the microcontroller will send via the USART communication, more precisely via the Bluetooth module, the message "PASSWORD_ACCEPTED" to the app. This message is useful to monitor whether the door has received the message. While the door strike is powered the door can be pushed and the user can enter. After this time the door will be locked again, but in case it was opened the same door strike will allow it to close without needing it to be powered, just due to the physical construction of the strike. If the comparison, on the other hand, results to be incorrect the door strike will remain locked and the red LED will be set as HIGH for 4 seconds. Then the microcontroller will send via the USART communication the message "PASSWORD_DENIED" to the app, with the same purpose as before.

After this process the cycle goes back to the password receiving point to wait for a new password entry and to restart the entire smart lock mode again.

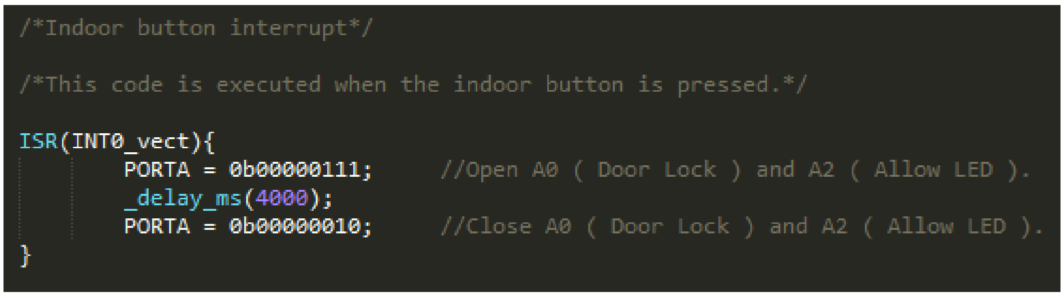
The next image shows how all the different functions are called and how this part of the code is structured. The password has been hidden for security purposes, however the following figure should still easily describe all the steps just mentioned before and give insights on how the process is driven inside the eternal loop as well as how all the different functions implemented in the solution's library are ordered to perform the objective of this smart lock system.



```c
/*Smart Lock mode*/

/*This is the main working mode of the application and its most important part.
In this part of the code is where the working loop is defined and where the
automa-tion of a door is executed. */

char password[]=":****************";    // Variables Definitions
char pass[]="", *ret;   // Variables Definitions

PORTC = 0b00000001; //Set Pin C0 High. ( Power HC - 05 )

PORTA = 0b00000010; //Power on the Working LED.

while(1){ //Working loop
    receiveString(pass,'.');    //Receive password from Bluetooth module.
    ret = memchr(pass, ':', strlen(pass));  //Transform string received into useful password style
    //sendATCommand(ret);
    if(compareStrings(ret,password,PASSWORD_LENGTH)==1){ //If password is correct:
        sendString("PASSWORD_ACCEPTED");    //Send control message to the App.
        PORTA = 0b00000111;     //Open A0 ( Door Lock ) and A2 ( Allow LED ).
        _delay_ms(4000);
        PORTA = 0b00000010;     //Close A0 ( Door Lock ) and A2 ( Allow LED ).
    }else{
        sendString("PASSWORD_DENIED");  //Send control message to the App.
        PORTA = 0b00001010;     //Open A3 ( Deny LED ).
        _delay_ms(4000);
        PORTA = 0b00000010;     //Close A3 ( Deny LED ).
    }
}
```

Figure 11: Smart lock mode implementation

Aside from this loop, the external interrupt allowed in the configuration part is always active and hence, can be triggered at any point during both running modes. The INT0, PD2 pin, is always connected to the normally LOW press button. This button can be pressed at any time and the interrupt ISR code will be triggered and starts. This code is similar to the code executing the control loop to verify the password received; the only exception is the missing feedback message to the app. This means whenever the indoor button is pressed the door is unlocked for 4 seconds. This is a simple but convenient way of opening the door from inside without the need to use a smart device.



```c
/*Indoor button interrupt*/

/*This code is executed when the indoor button is pressed.*/

ISR(INT0_vect){
        PORTA = 0b00000111;     //Open A0 ( Door Lock ) and A2 ( Allow LED ).
        _delay_ms(4000);
        PORTA = 0b00000010;     //Close A0 ( Door Lock ) and A2 ( Allow LED ).
}
```

Figure 12: ISR code

4. APP DEVELOPMENT

This fifth chapter focuses on the Android control app that will interface the door lock automation system and the ideas involved around it.

The developed app will be the users' interface with the door and is crucial in the project. The app must be capable of verification of the users' identity by using a password authentication process and to connect the smartphone or tablet to the Bluetooth module to send the correct password to the microcontroller and to receive feedback back from the smart lock. These four functionalities are the basic parts of the app to let the user communicate with the lock system. Further, this application should be able to have some kind of database where the users and passwords of the system are stored and consulted if required. Another important functionality of the system is the constant feedback with the user. The app must include a text notify or label that constantly informs the user of the latest function or step performed and allow him to see if something in the process went wrong, for example if the Bluetooth module has not been reached and the connection was cancelled. This information is useful in the developing process, too; because it gives important insights of the internal process.

The app must be simple and operate quickly so the opening process does not require too much time. The number of interactions with the user should be reduced to a minimum. Less information demands leads to a simpler and more efficient app that will allow the user to open the door in the fastest and smoothest way possible. For these goals, the graphical interface has only few buttons and elements and is highly user friendly.

For a better and clearer view of the applications' main functioning processes, a general flowchart of the applications functioning way is included. In the following scheme it is possible to observe the steps and verifications the app must complete to open the door.

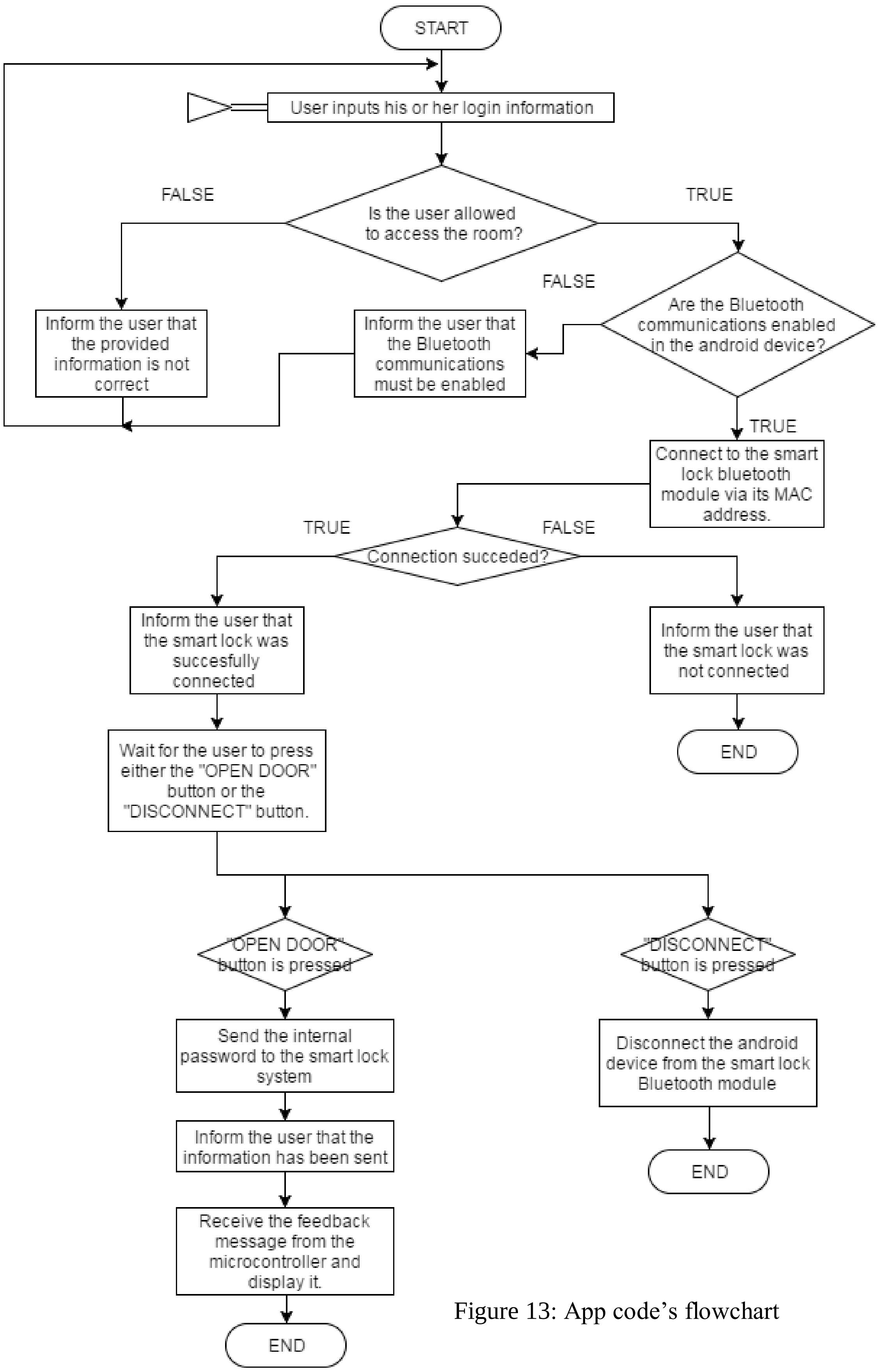

START
User inputs his or her login information
Is the user allowed to access the room?
FALSE
TRUE
Inform the user that the provided information is not correct
Are the Bluetooth communications enabled in the android device?
FALSE
Inform the user that the Bluetooth communications must be enabled
TRUE
Connect to the smart lock bluetooth module via its MAC address.
Connection succeded?
TRUE
FALSE
Inform the user that the smart lock was succesfully connected
Inform the user that the smart lock was not connected
END
Wait for the user to press either the "OPEN DOOR" button or the "DISCONNECT" button.
"OPEN DOOR" button is pressed
"DISCONNECT" button is pressed
Send the internal password to the smart lock system
Disconnect the android device from the smart lock Bluetooth module
END
Inform the user that the information has been sent
Receive the feedback message from the microcontroller and display it.
END

Figure 13: App code's flowchart

Before starting with the app itself it needs to be said that the developed app in this work is neither the definitive nor the complete option. This app was developed after the prototype device was obtained to have a better simulation of the whole system and to get a demonstration door working. Unfortunately, due to the lack of time and knowledge only a first simple solution was designed. The app was developed by using a tool designed by MIT University that introduces programmers to the android app designing world with simple but powerful graphical tool. As described on the tool's webpage:

> "MIT App Inventor is an innovative beginner's introduction to programming and app creation that transforms the complex language of text-based coding into visual, drag-and-drop building blocks. The simple graphical interface grants even an inexperienced novice the ability to create a basic, fully functional app within an hour or less." (MIT App Inventor Web Page)

Through the simplicity of the app and its user friendliness the user is able to familiarize with it quickly. Therefore it is not surprising that nearly 3 million users all around the globe such as educators, governments, product designers, researchers and all kind of entrepreneurs and app builders use it.

This programming tool is divided in two different parts. The first one is the graphical designer part where all the elements that the user will see and interact with are placed and settled. In this part the application allows the developer to choose between wide ranges of already predefined objects and elements that cover almost every need during the app development. These objects vary from a button to a Bluetooth client and can be configured and modified to satisfy all required needs.

The second part is the programming part where a prospering and intuitive graphical programming environment is provided. There, the functionalities and actions of the app are defined by using a simple and versatile block configuration that allows the user to have a clear and intuitive view of the developed code structure. Depending on the included elements in the graphical environment part some functions and methods will be provided while others are not therefore, only some actions will be performable.

When programming with this interface proper names to all the elements of the app should be given. This will make the programming task much easier and comprehensible, and moreover allow other programmers to understand the developed code easier and faster.

An example of these blocks is presented in the next figure:

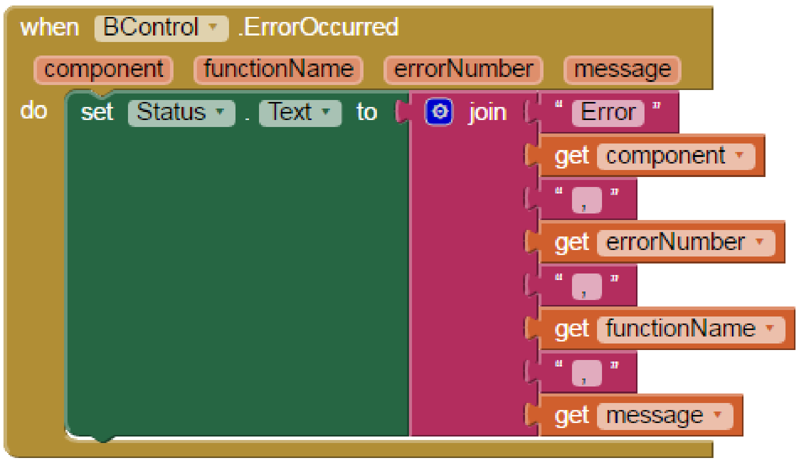


Figure 14: App Inventor programming block

This tool offers a QR code from which the application can be downloaded onto any Android mobile phone. It also gives the possibility to build and download the .apk installer. Having this installer, any installation protocol is possible such as creation of an webpage charged with the file allowing the user to enter the page and obtain the app or a QR code can be created and placed next to the door to let users scan and download the installer, etc. Having this simple .apk installer opens a lot of possibilities to the download procedure that can be implemented in the smart lock system.

Thanks to this tool the first approach to the app was obtained. This "Door lock" basic app could successfully connect to the Bluetooth module of the device, send the password to it and receive the device's feedback messages. In addition, it could correctly interact with the Bluetooth module and change its current state as the user desired.

Once this app was loaded some improvements were made to start get more security features such as the decisions regarding who can use the door lock and other features.

The app has two screen/interfaces: the login screen and the Bluetooth control interface, which allow the user to open the door.

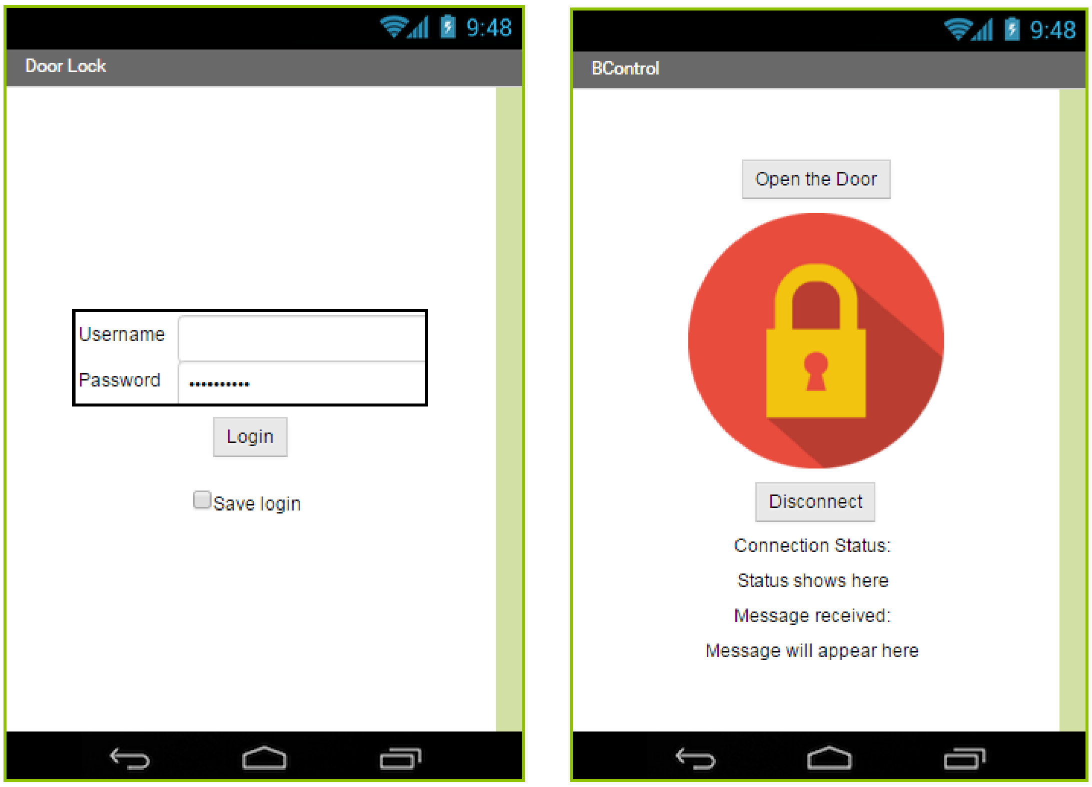


Figure 15: Android Application's Screen 1 and Screen 2

The first one to be loaded after opening the application is the login screen. This screen has a simple username and password login system that allows the user to enter his or her credentials to verify if he or she has permission to access the room where the door lock is placed. For this first approach the app has only access to an internal database with only few registered users. The database can be configured when developing the app and users can be added. This gives some difficulties to administrate all users in the system and to add new users to the database, because a new app must be created each time the system administrator wants to add a new user or simply wants to change some information of an existing one. The best solution to the login interface problem would be to create a complete internet server database with the students' numbers and password to administrate there which students, teachers or lab workers are allowed to enter the room. This database could be managed by the system administrator allowing him to have an extensive control of who has the right to access any room at any time, including possible future expansion like allowing him to have an access register to know who was in each room at which time and for how long.

Focusing more on the login screen design, one notices, it is formed by several elements: two text boxes and labels where the user will set his account name and password, as well as one button that allows the user's data validation and a check box that saves the users' information for more convenient usage. Also, even if it is not on screen a tiny database is included to save the users information to let the save login button work.

The basic idea of this screen is that the user inputs his or her username and password and it clicks the login button to validate them. On the one hand, if these two inputs are included in the authorized database of the system the user is allowed to jump to the next screen. On the other hand, if the username and password are not recognized the next screen will not pop up and an error message will appear under the login button. Another feature is the save login check box. If it is activated when the user clicks the login button, the app will remember this information and saves it in the text box for the next time the user opens the app.

The second and more complex screen is in charge of the Bluetooth communication with the smart lock to have control over the current state.

The BT control screen is formed by few different elements distinguished in two buttons: One to open the door and another to disconnect the Bluetooth connection. A picture to make everything simply more user friendly and four text tags to display information about the connection status. There are also three non-visible elements, a notify, a Bluetooth client and an internal clock.

The working way is quite more complex than the login screen. When this screen starts the first thing occurring is the app checking if the Bluetooth module of the smartphone or tablet is turned on. If it is not a notify pops up with a message advising the user to turn the Bluetooth on in order to continue with the door control process. After that, the user has to go back to the screen and restart again, this time with the Bluetooth in the device activated. If the Bluetooth module is on, the BT client of the application automatically connects to the MAC address of the HC – 05 module of the smart lock. If this connection is done correctly the status label is changed to “Door lock BT connected” and if not to “Door lock BT not connected”, to advice the user to reload again the login screen to start over. Normally, if all the steps are correctly followed and the Door lock is close enough to the mobile system, the Door lock will automatically connect to the BT module.

This automatic connection only to the MAC address of the Bluetooth device is done for security purposes. If someone wants to know the message that is transmitted from the app to the system he or she will not be able to connect to the app or any other device that can read and show the password. This is a first step to block anybody except the project developers to know the internal password designed for the communication between the app and the smart lock. More security could be added by encrypt communication to improve the communication connection restrictions. A good idea is to encrypt the password in the microcontroller and in the app code designed to further restrict who has access to the password.

Once the app and the module are connected the “Open Door” button can be used. When clicked, the button makes the app send an internal defined password to the Atmega32 microcontroller internal software to the smart lock. The smart lock, as explained before, will verify this password and will act consequently to the message received.

Then, the app awaits the feedback to inform if the door has been successfully opened or not. For now, the app only shows the received message in the information labels to inform the user that the smart lock successfully received the password. The idea for further extensions of the code is to use the received information to monitor the processes of the Smart Lock as previously mentioned. If the message is received the smart lock system has unlocked the door and access is available. It is a good way to let the app and the control system know that the password has been sent after the "Open Door" button was actually pressed and arrived at the smart lock to modify its state.

The basic idea behind the first app is to check if the user has the right to send the password to the door, and once this authentication has turned out to be true all the steps necessaries to send the password and unlock the door strike start. For now, the system is closed and secures, because the only communication channel for the app is done via Bluetooth with the HC – 05 module. As soon as the internet database is included in the system, the app need to include more security restrictions in the log in process to avoid external non authorized users to enter the database and change the access rights of the users in the system.

The developed app has a high performance and is efficient enough for its main purpose. The app needs around seven to nine seconds from when it is first started till the door is finally opened, in case the credentials of the user have already been saved in the system. The user only needs to press two buttons, the login button and the open door button to open the door. The most consuming part of the process is the direct connection of the mobile device with the Bluetooth module, however this part cannot be avoided as it gives the system a very important boost in its security control and performance. In general, this first app approach achieves the overall goals set. It is simple, yet fast and clear to the user.

After all these explanations it would be interesting to analyse how the app impacts the system and if the android system requires any specifications to run it. The app needs little memory to be installed in the system, only 1,30Mb. It has no need for any specific permission to be run or access to any data source or Internet connection. The app showed to not recognisably affect the energy consuming rate or noticeable slow down the system performance of the device even though it was always running in the background.

In general and as already commented before, the app is really simple and fits all android systems. The only real restriction to the correct functioning is the necessity of a Bluetooth module in the android device. Using a Google test it was obtained that the app could be installed on 100% of phones that have an actually working Bluetooth module and exactly 100% of all Android mobile phones present in the Google App Store, giving the whole system a wide range of controlling devices and possibilities to interact with the door.

5. SYSTEM TEST

This chapter shows all the requirements needed to test the system and to have a functional prototype from the software point of view. It also explains the results obtained when the system was tested possible and changes that could fit the system.

The testing process was made in two steps. First a breadboard test was carried out to see if the code and all the hardware structure worked fine together. After the PCB prototype was designed and checked, it was implemented into a demonstration door.

First and only step from the software side of the project was to program the microcontroller. As it was already mentioned before, Atmel Studio 7 was used to build the code into the Atmega32 and an AVRISP mkII programmer was used to allow the communication between the PC and the microcontroller. To make this procedure easier, an AVR Universal Board was used to interface the Atmega32 in the first testing attempts. Using the ISP port of this board it was simple and quick to charge programs into the microcontroller memory without needing complex connections or external systems.

There are two main concerns to ensure a good programming performance.

First one is the selection of an appropriate ISP clock frequency. The AVRISP mkII supports ISP frequencies from 51 Hz up to 8.0 MHz. To ensure a good communication when using the ISP programming interface, the ISP clock frequency must not exceed the limit target device frequency supported. For this microcontroller the internal clock frequency is always ¼. (The maximum ISP clock frequency depends on the device system clock, internal clock division, etc.)

The second concern is the configuration of the system clock and clock options. The datasheet of the microcontroller is included in the chapter “System Clock and Clock Options” where a lot of tables and information cover all possible clocks, oscillators and crystals that the microcontroller could use. It is interesting to know the meaning of all the registers and fuses but, while using the Atmel Studio, it will not be necessary to change these values one by one. The program offers a Clock setting option in the device-programming menu where the system clock can be specified without needing to enter all fuses in detail.

This project, as shown in the hardware structure part, uses an 8MHz external crystal oscillator. In order to configure this kind of crystal clock the “Ext. Crystal/Resonator High freq: Start-up time: 1K CK + 0 ms” option must be selected in the device programming menu clock fuses options.

After these configurations were established, tests were carried out with different components and different options to get the best performance.

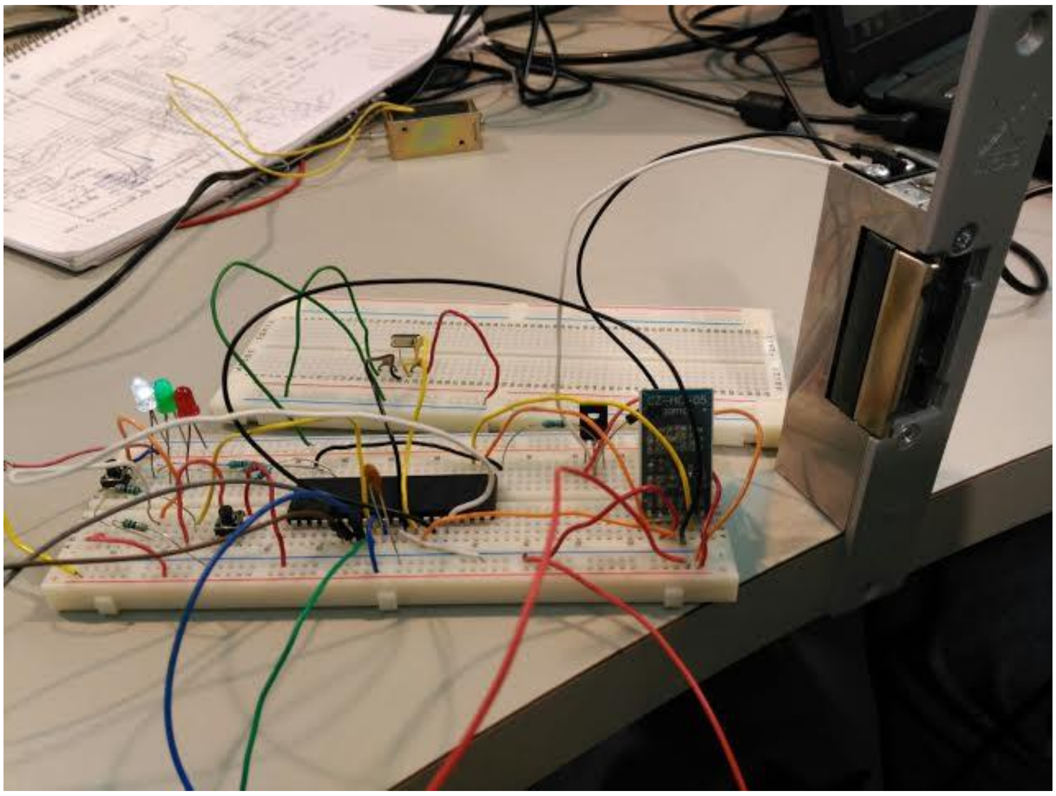


Figure 16: Breadboard test

The only problem found in the testing attempts was the unusually and randomly USART communication which when not done correctly, the device does not recognize the door password. The reason for this problem is probably a margin of error in the communication due to the values of frequency and baud rate selected. This little margin can on really few occasions lead to miscommunication between the Bluetooth module and the MCU and change the password. Modifying some internal parameters and avoiding too complicated passwords will reduce the error possibility.

The first test was successful and the system worked as it was designed too.

After the first breadboard test was carried out, an integrated design was implemented to have a single board system that could be easily installed in the door.

The Smart lock can be programmed and reprogrammed easily due to the implemented ISP port. This port facilitates the software development for the system and supports several trials and demos in an easy and practical way only using the AVRISP mkII programmer and the Atmel Studio.

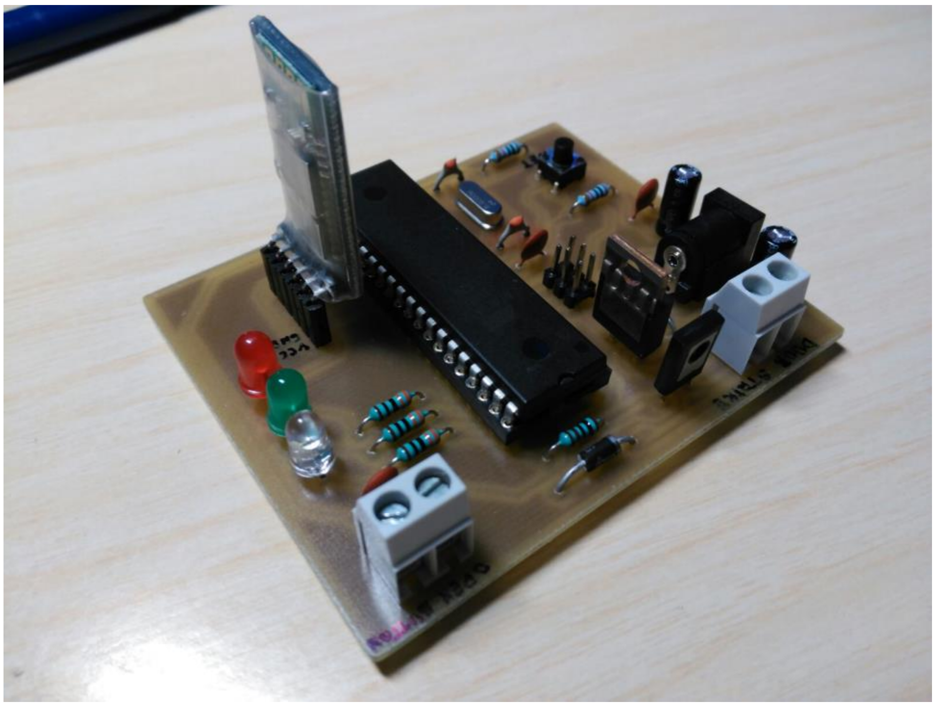


Figure 17: PCB protoype

The PCB and other technical details are explained in more detail in the hardware part of this work. After it was finally designed and operative, the software solution was charged to the AVR microcontroller and tested.

When tested the PCB successfully accomplished the purpose of this project. The only thing differing from the breadboard test was the feedback message “PASSWORD_ACCEPTED” or “PASSWORD_DENIED” was not received by the app. This could be a problem should be solved when further extensions are developed.

After some research and trying to find the reason of this malfunction a highly probable reason was pointed out for the mismatch between the two tests. The only difference between the prototype and the breadboard system is the system clock. When using the universal board the microcontroller is provided with a compact full oscillator that works as system clock. In the prototype a crystal resonator is used. This difference could have made the clock system in the prototype unstable and lead to failure of feedback messages, since the use of the USART serial communication got more complex. Tests were made to see if the prototype would work with an oscillator and showed that when removing the system microcontroller and replacing it with the AVR universal board the message was received correctly. As a result, next prototypes and implementations of the system should include a full oscillator to solve the problem and have feedback messages in the app to be able to use the information in future expansions.

6. CONCLUSIONS

The Smart lock system that is developed in this thesis is successfully achieved.

The main objective of the thesis is to design and manufacture an operational prototype that can be in real conditions. The full operation of the developed prototype requires only a power socket or an external battery of 12 volts. This prototype is developed with the aim to keep it as simple as possible and to make it repeatable and expandable for other interested students and projects for further development. Another important goal is to use as few components as possible to keep the system producing process as cheap as possible.

The process of developing the work was completed as stated in the requirement of the project. After the research and the planning part, the whole system is designed and structured allowing both sides HW and SW of the project to start implementing the proper solutions and resources for every element of the work.

In particular the software solutions designed in this part of the project is successfully implemented on an android mobile phone and can manage and remotely control the smart lock system via Bluetooth communication channel. The onboard microcontroller and the Bluetooth module are set up to communicate successfully. The transmission of information between the user interface app and the smart lock device is carried out correctly, allowing the authentication process to run in the desired way and so enabling the smart device to act consequently through the actuator of the system, the door strike. Also, on the other part of the software solution, the basic android app could successfully connect the HC – 05 BT module and the android device and send the proper information to open the door in a secure way, taking care of the user permissions and access rights.

After all the work, as commented in the previous chapter, one small uncompleted functionality is still present in the prototype and needs to be taken into account in the next development and improvements of the project. The feedback messages sent from the microcontroller to the app should be improved and modified as explained before to have more control over the process.

The main idea behind the design of this prototype is to facilitate a base and a starting point to all the students in VAMK that want to keep on improving and expanding the work as much as possible and as best as possible, to get a really complete and functional system.

The main concerns in further implementations would be to improve security, for example by encrypting the internal password of the system or by improving the apps' log in screen, and creating a database with users that will allow a real implementation of the door access control and set bases for the creation of a monitoring system. Also including other functionalities to the board would be an interesting idea to try to include more verification possibilities and to improve the system in order to automatize the authentication steps.

LIST OF REFERENCES

Printed books:

Barnett Cox, Barnett O'Cull (2004). Embedded C Programming and the Atmel AVR. Clifton Park. Delmar Cengage Learning.

Williams, Elliot (2013). Make: AVR Programming: Get under the hood of the AVR microcontroller family. USA. Maker Media.

Trevennor, Alan (2012). Practical AVR Microcontrollers: Games, Gadgets, and Home Automation with the Microcontroller Used in the Arduino. USA. T+A.

Electronic publications:

Phung, S.L (2008-2015). Getting Started with C Programming for the ATMEL AVR Microcontrollers. Version 2.0.

Camera, Dean (2015). AVR Programming Methods. Source: www.fourwalledcubicle.com.

Camera, Dean (2015). Using the USART in AVR-GCC. Source: www.fourwalledcubicle.com.

Camera, Dean (2015). Interrupt Driven USART in AVR-GCC. Source: www.fourwalledcubicle.com.

Datasheets:

ATmega32/L Datasheet (2011). Atmel Corporation. USA. www.atmel.com.

AVR069: AVRISP mkII Communication Protocol (2006). Atmel Corporation. USA. www.atmel.com.

Internet URL sources:

AVR C Programming:

- http://www.nxp.com/files/training/doc/dwf/AMF_ENT_T0001.pdf

AVR External Interrupts C Programming:

- http://www.avr-tutorials.com/interrupts/avr-external-interrupt-c-programming

USART guides:

- http://maxembedded.com/2013/09/the-usart-of-the-avr/

- http://extremeelectronics.co.in/avr-tutorials/using-the-usart-of-avr-microcontrollers/

HC – 05 modules guides and explanations:

- http://www.tec.reutlingen-university.de/uploads/media/DatenblattHC-05_BT-Modul.pdf

- http://www.robotshop.com/media/files/pdf/rb-ite-12-bluetooth_hc05.pdf

- http://www.martyncurrey.com/arduino-with-hc-05-bluetooth-module-at-mode/