|  |  |
| --- | --- |
| Smart alarm clock | Tudor Roxana-Ioana  *Technical University of Cluj-Napoca Computer Science Teacher : Attila Fuzes* |

Contents

[1. Introduction 2](#_Toc92966283)

[2. Components 2](#_Toc92966284)

[I. ARDUINO 2560 MEGA BOARD 2](#_Toc92966285)

[II. DS1302 Real Time Clock Module 2](#_Toc92966286)

[III. Passive buzzer module low level trigger buzzer 3](#_Toc92966287)

[IV. ESP8266 Wi-Fi Module 3](#_Toc92966288)

[V. Piezo electric sensor 4](#_Toc92966289)

[VI. Push button 4](#_Toc92966290)

[VII. Adafruit NeoPixel Digital RGBW LED Strip 4](#_Toc92966291)

[3. Design 5](#_Toc92966292)

[4. Implementation 6](#_Toc92966293)

[5. Code 7](#_Toc92966294)

# Introduction

The project represent a smat alarm clock, that the user can interact with and see different behaviours. Usually alarm clock are made with a snooze button and most of us like to hit it pretty often. In order to prevent the ariving late to school or work, because the user wants to sleep a little more, the smart alarm clock has a new integrated behaviour just for that. Having visual and auditory stimulus this will make the process of waking up a little easier.

The user sets the hour he/she wants to wake up on the website made specially for this, and the wi-fi module sends this data to the board where it will be stored in memory. Once the alarm starts ringing, the neopixels wil turn on in a radnom manner and having different colors. If the user wants to hit the snooze button the alarm clock enters in angry mode and it will start ringing faster and the leds will turn led, preventing in this way to fall back to sleep.

I wanted to do this project in order to gain more experience regarding the embbeded systems desgin, by making a trivial project that looks easy to implement but is is actuallly challenging taking in account that the operation are execute sequentially.

# Components

## ARDUINO 2560 MEGA BOARD

A close-up of a circuit board

Description automatically generated with medium confidence

The **Arduino Mega 2560** is a microcontroller board based on the [ATmega2560](http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf). It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

Arduino Mega is able to handle better my requirements, since the design used a large number of components. The ESP8266 needs pins 18,19 in order to work properly and the RTC module needs the SDA, SCL ( pins 20, 21 ) used for the I2C communication. While this pins were busy I needed one more for the interrupt, and knowing that MEGA has 4 interrupts, it was the best choice.

## DS1302 Real Time Clock Module



The DS1302 real time clock module is a cheap module with high accuracy that can be used in different projects. This RTC module provides seconds, minutes, hours, day, date, month, and year information. In this module, date is set automatically based on whether the month is 29, 30 or 31 days and also it is leap year or not. (That’s only valid until the year 2100).

This module has 5 pins: VCC (Power), GND (Ground), CLK (Clock pin), and DAT (Data pin), RST(Reset).

The Real Time Clock provides a better precision of time ( the error is minor, only 0,5 sec) and is a cheap component. It works in low power situation or in case of power failure.

## Passive buzzer module low level trigger buzzer

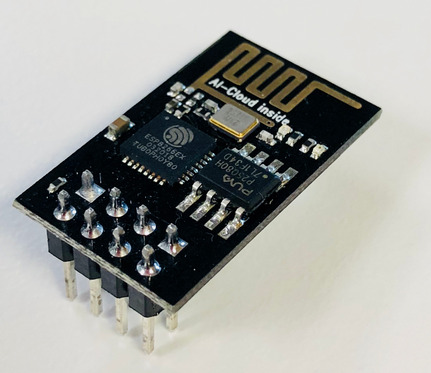


A buzzer is a device used to produce sound. There are two types of buzzer: active and passive. In the active model, buzzer starts playing sound only if connected to a power supply, but in the passive model, we need to send a pulse from the microcontroller to play sounds.  
The supply voltage of the buzzer is 3 volts, 5 volts and 12 volts.

The passive buzzer module has 3 pins: I/O(), GND(ground), VCC(Power)

The buzzer module is used in order to make the alarm clock ring. I choose an active buzzer because it was easier to work with, taking in consideration that I only want it to ring form time to time.

## ESP8266 Wi-Fi Module



The ESP8266 is a low-cost [Wi-Fi](https://en.wikipedia.org/wiki/Wi-Fi) microchip, with built-in [TCP/IP networking software](https://en.wikipedia.org/wiki/TCP/IP_stack), and [microcontroller](https://en.wikipedia.org/wiki/Microcontroller) capability. This small module allows microcontrollers to connect to a Wi-Fi network and make simple TCP/IP connections using [Hayes](https://en.wikipedia.org/wiki/Hayes_command_set)-style commands.

Using the Wi-Fi module the user is able to remotely control the time it sets the alarm clock according to IoT concepts. The price of ESP8266 was low and the communication done using it works just fine for my project

## Piezo electric sensor



A piezoelectric sensor is a device that uses the [piezoelectric effect](https://en.wikipedia.org/wiki/Piezoelectric_effect) to measure changes in [pressure](https://en.wikipedia.org/wiki/Pressure), [acceleration](https://en.wikipedia.org/wiki/Acceleration), [temperature](https://en.wikipedia.org/wiki/Temperature), [strain](https://en.wikipedia.org/wiki/Strain_(materials_science)), or [force](https://en.wikipedia.org/wiki/Force) by converting them to an [electrical charge](https://en.wikipedia.org/wiki/Electrical_charge).

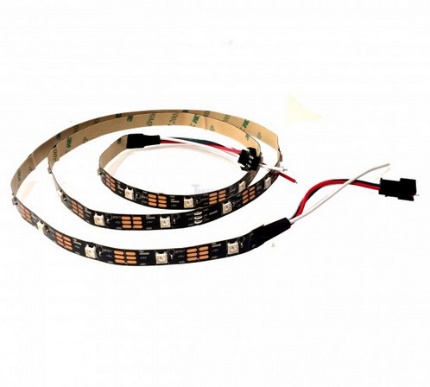
I choose it in order to detect the pressure the user will hit on the snooze button. If it’s bigger than a reference threshold, then the angry clock behavior starts.

## O imagine care conține interior Descriere generată automatPush button

A push button switch is a small, sealed mechanism that completes an electric circuit when you press on it. When it's on, a small metal spring inside makes contact with two wires, allowing electricity to flow. When it's off, the spring retracts, contact is interrupted, and current won't flow.

The push button is used for the interrupt behavior. When the button is pushed the alarm turn off. It’s a cheap component and it does it jobs properly.

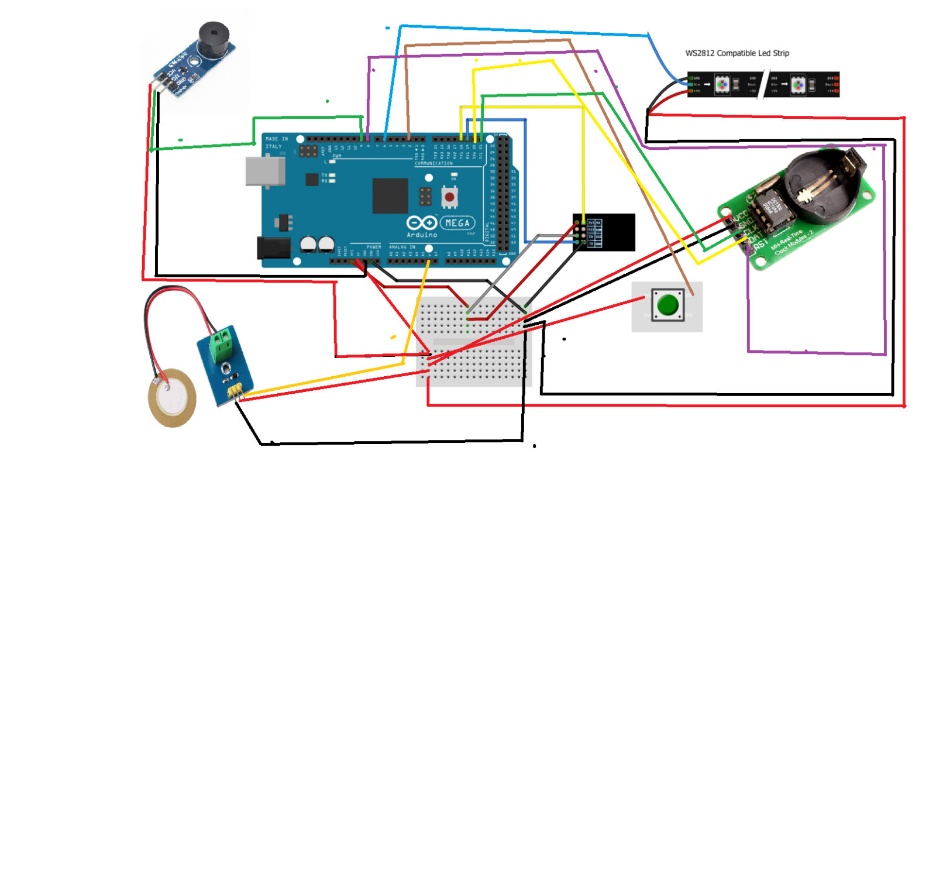
## Adafruit NeoPixel Digital RGBW LED Strip



RGB LEDs make a variety of colors based on this model. To control each RGB LED, you need three digital pins of a microcontroller (or the development boards like Arduino). NeoPixel is the Adafruit brand for addressable LEDs.

The Neopixels are used for the visual effect they provide. You pay more for a longer strip ( the number of addressable pixels are taken intro account) , but for this project a smaller strip works just fine.

# Design



# Implementation

The code implementation was fun an challenging to do. Mostly the part that simulates simultaneity, where the buzzer start ringing and te neopixels are turn on and off following a pattern. In order to do that I had to define some classes, which had variables like pin, ledcolor, but most importantly TimeOn and TimeOff. This time dictates the number of miliseconds the buzzer or one pixel is on, and the other the time the pixel will turn off. And to create simultaneity if I set the buzzer to start at 100 miliseconds and the neopixel at 150 miliseconds, the time between them will be so small that it will create a sense on concurency.

For the wifi and clock part is I have made a function that will compute the current time calling the RTC function and then interpreting the input received by the serial communication that happens between the wifi module and the board.

The RTC module uses a I2C communication protocol. This is a bus interface connection incorporated into the devices. It uses only two bidirectional open-darin lines for communication. SDA and SCL. Bith of these lines are pulled high.

SDA = transfer of data

SCL = clock signal

It operates in 2 modes: slave and master . Each data send to from RTC through SDA pin is synchronized by a high to the low pulse of each clock on the SCL line.

The data line cannot change when the clock line is high, it can only change when the clock line is low.

The fact that the RTC module has a battery it will to give accurate time and data continuously.

The interrupt behaviour is given by the pushbutton, that is connected to pin 2 on the board. The pin 2 on Mega board has INT0, that can be used in order to attach an interrupt and bind it to a function or using it with the ISR register. Once the interrupt is attached, everytime the user presses the button the system will interrupt and the function attached will be executed before continuing.

# Code

int sensorPin = A6;

int buzzer = 9;

int ledoutput = 0;

int ThresHOLD = 400;

int button = 2;

#include <Adafruit\_NeoPixel.h>

#ifdef \_\_AVR\_\_

#include <avr/power.h> // Required for 16 MHz Adafruit Trinket

#endif

//libraries needed for the real time clock

#include <TimeLib.h>

#include <TimerOne.h>

#include <NTPClient.h>

#include <WiFi.h>

#include <WiFiClient.h>

#include <WiFiServer.h>

#include <WiFiUdp.h>

#include <ThreeWire.h>

#include <RtcDS1302.h>

#include <ArduinoHttpClient.h>

#define DEBUG true

#define PIN 6

#define NUMPIXELS 30

ThreeWire myWire(20, 21, 8); // IO, SCLK, CE

RtcDS1302<ThreeWire> Rtc(myWire);

Adafruit\_NeoPixel pixels(NUMPIXELS, PIN, NEO\_GRB + NEO\_KHZ800);

int array\_brow\_right\_1[4] = { 0 , 14 , 23};

int array\_brow\_right\_2[4] = { 1 , 15 , 24};

int array\_brow\_left\_1[4] = { 6 , 10 , 25 };

int array\_brow\_left\_2[4] = { 7 , 11 , 26};

volatile int alarm\_on\_int=0;

#define countof(a) (sizeof(a) / sizeof(a[0]))

char hours[2], minutes[2];

char Time[] = "TIME:00:00:00";

char Date[] = "DATE:00/00/2000";

byte last\_second, second\_, minute\_, hour\_, day\_, month\_;

int year\_;

//In order to be able to make the buzzer and the Neopixel run concurenttly I choose to structure them in classes and

//define parameters like onTime and offTime in their class. They are important to kep in mind because they establish

// the time of running and change their state accordingly. I choose to work with milis() in order to be able to create the

//effect o simulataneity. For example if the buzzer start after 100 millis and the neopixels after 150 millis , then the difference

//in time would be so little for the human ear and eye, that it woul create the illusion that they run in the same time.

class Neopixel

{

//class variables

int pixel\_nr;

//first pixel to turn on

int pixel\_nr2;

//second pixel to turn on

int Ledcolor;

//set the color of the pixel

int val;

//value used for the second update function in order to create different colors of the pixels

long onTime;

// time the Neopixels are on

long offTime;

// time the Neopixels are off

//maintina the current state

unsigned long previousState;

public:

Neopixel (int pix, int pix2, int color , long on, long off) //constructor of the Neopixel

{

pixel\_nr = pix;

pixel\_nr2 = pix2;

Ledcolor = color;

val = 50;

onTime = on;

offTime = off;

previousState = 0;

// set previous state to 0

}

void Update() //function for the angry neopixel behaviour. The leds turn red and form a little brow

{

unsigned long currentState = millis();

//the millis function will establish the current state

//if leds are turned on, and the onTime has passed, now it's time to change their state

if ((Ledcolor == 255 ) && (currentState - previousState >= onTime))

{

Ledcolor = 0;

//turn off led

previousState = currentState;

//previousStae is now currentState

pixels.setPixelColor(pixel\_nr, pixels.Color(Ledcolor, 0, 0));

pixels.setPixelColor((pixel\_nr + 1), pixels.Color(Ledcolor, 0, 0));

pixels.setPixelColor( pixel\_nr2, pixels.Color(Ledcolor, 0, 0));

pixels.setPixelColor(( pixel\_nr2 + 1), pixels.Color(Ledcolor, 0, 0));

pixels.show();

}

//if leds are turned off, and the offTime has passed, now it's time to change their state

if ((Ledcolor == 0 ) && (currentState - previousState >= offTime))

{

Ledcolor = 255;

//turn on led

previousState = currentState;

pixels.setPixelColor(pixel\_nr, pixels.Color(Ledcolor, 0, 0));

pixels.setPixelColor((pixel\_nr + 1), pixels.Color(Ledcolor, 0, 0));

pixels.setPixelColor( pixel\_nr2, pixels.Color(Ledcolor, 0, 0));

pixels.setPixelColor(( pixel\_nr2 + 1), pixels.Color(Ledcolor, 0, 0));

pixels.show();

} }

//function for the normal neopixel behaviour. The leds turn in different colors and turn on in a pseudo random manner

void Update2()

{

unsigned long currentState = millis();

//the millis function will establish the current state

if ((Ledcolor > 0) && (currentState - previousState >= onTime)) //if leds are turned on, and the onTime has passed, now it's time to change their state

{

Ledcolor = 0;

previousState = currentState;

pixels.setPixelColor((pixel\_nr), pixels.Color(0, 0, 0));

//turn off pixel number 1

pixels.setPixelColor( pixel\_nr2, pixels.Color(Ledcolor, 0, 0));

//turn off pixel number 2

pixels.show();

}

// }

if ((Ledcolor == 0 ) && (currentState - previousState >= offTime))

//if leds are turned off, and the offTime has passed, now it's time to change their state

{

// for ( int i = 0;i < NUMPIXELS - 1; i++)

// {

//Ledcolor = 64;

//set value to 64 in order to have a quarter of their full brightness in order to make it more pleasing to the human eye

pixels.setPixelColor((pixel\_nr ), pixels.Color( val , val - 20, Ledcolor - val));

//turn on pixel number 1 - choose different computation for val in order to create different colors

pixels.setPixelColor( pixel\_nr2, pixels.Color( val , val / 2, Ledcolor / 2));

//turn on pixel number 2 -choose different computation for val in order to create different colors

val += 20;

if (val > 64)

val = 20;

previousState = currentState;

//previous state the current state in order to compute it the next time milis function is called

pixels.show();

// }

if (Ledcolor < 0 )

Ledcolor = 0;

}

} //end Update2 function

void Reset()

//reset function in order to turn all the leds off once their program is over

{ Ledcolor = 0;

pixels.setPixelColor(pixel\_nr, pixels.Color(Ledcolor, 0, 0));

pixels.setPixelColor((pixel\_nr + 1), pixels.Color(Ledcolor, 0, 0));

pixels.setPixelColor( pixel\_nr2, pixels.Color(Ledcolor, 0, 0));

pixels.setPixelColor(( pixel\_nr2 + 1), pixels.Color(Ledcolor, 0, 0));

previousState = 0;

pixels.show();

}

};

class Buzzer

//class definition for Buzzer

{

//class variables

int buzzerPin;

//pin of buzzer

long onTime;

// on time

long offTime;

// off time

//maintaint the current state

int buzzerState;

//variables to keep in mind the current and previous state

unsigned long previousState;

public:

Buzzer (int pin, int on , int off)

//constructor for the Buzzer

{

buzzerPin = pin;

pinMode (buzzerPin, OUTPUT);

onTime = on;

offTime = off;

buzzerState = HIGH;

//set it to HIGH, because this buzzer is low triggered and i don't want it to start ringing

previousState = 0;

}

void Update()

//update function that implements the main behaviour of the buzzer

{

unsigned long currentState = millis();

//put in the current state the result given by the millis function.

// if the buzzer is on ( is low triggeref) and it's on time has passed , we change it's state and turn it off.

if ((buzzerState == LOW ) && (currentState - previousState >= onTime))

{

buzzerState = HIGH;

previousState = currentState;

digitalWrite(buzzerPin, buzzerState);

}

// if the buzzer is off ( is low triggeref) and it's off time has passed , we change it's state and turn it off.

else if ((buzzerState == HIGH) && (currentState - previousState >= offTime))

{

buzzerState = LOW;

previousState = currentState;

digitalWrite(buzzerPin, buzzerState);

}

}

void Reset()

//reset function for the Buzzer

{

buzzerState = HIGH;

previousState = 0;

digitalWrite(buzzerPin, buzzerState);

}

};

Neopixel pixel\_1(0, 6, 0, 150, 300);

//first three object are used for the angy clock behaviour

Neopixel pixel\_2(14, 10, 0, 160, 400);

Neopixel pixel\_3(23, 25, 0, 170, 500);

Neopixel pixel\_4(0, 28, 63, 300, 300);

//the next 9 objects are used for the normal clock behaviour

Neopixel pixel\_5(1, 27, 60, 350, 400);

Neopixel pixel\_6(2, 26, 59, 400, 500);

Neopixel pixel\_7(3, 25, 58, 450, 550);

Neopixel pixel\_8(5, 24, 52, 500, 600);

Neopixel pixel\_9(5, 24, 48, 500, 600);

Neopixel pixel\_10(6, 23, 44, 500, 600);

Neopixel pixel\_11(7, 22, 40, 550, 700);

Neopixel pixel\_12(8, 21, 39, 600, 800);

Buzzer buzz(9, 50, 100);

//buzzer object

//Buzzer buzz2(9, 300, 500);

//Buzzer\_Song buzz2(9,100,200,500);

void setup() {

// put your setup code here, to run once:

// Serial.begin (9600);

pinMode(buzzer, OUTPUT);

//setup pinmode and asssign value

digitalWrite(buzzer, HIGH);

Serial.begin(115200);

//set baoud rate to the serial communication between the board and the wifi module

Serial1.begin(115200);

Serial.print("compiled: ");

//print the time of compilation given by the Real time clock respecting the format

Serial.print(\_\_DATE\_\_);

Serial.println(\_\_TIME\_\_);

Rtc.Begin();

RtcDateTime compiled = RtcDateTime(\_\_DATE\_\_, \_\_TIME\_\_);

if (!Rtc.IsDateTimeValid())

//verification done for the RTC and some updates in order to synchronize with the current time

{

// Common Causes:

// 1) first time you ran and the device wasn't running yet

// 2) the battery on the device is low or even missing

Serial.println("RTC lost confidence in the DateTime!");

Rtc.SetDateTime(compiled);

}

if (Rtc.GetIsWriteProtected())

{

Serial.println("RTC was write protected, enabling writing now");

Rtc.SetIsWriteProtected(false);

}

if (!Rtc.GetIsRunning())

{

Serial.println("RTC was not actively running, starting now");

Rtc.SetIsRunning(true);

}

RtcDateTime now = Rtc.GetDateTime();

if (now < compiled)

{

Serial.println("RTC is older than compile time! (Updating DateTime)");

Rtc.SetDateTime(compiled);

}

else if (now > compiled)

{

Serial.println("RTC is newer than compile time. (this is expected)");

}

else if (now == compiled)

{

Serial.println("RTC is the same as compile time! (not expected but all is fine)");

}

pinMode(2 ,INPUT);

//setup pinMode for the button used for interrupt

digitalWrite(2, HIGH);

// set it to high, beacuse the pushbutton has a pulldown resistor

attachInterrupt(digitalPinToInterrupt(2),buttonInterrupt,RISING);

//attach an interrupt to pin2 where the button is connected. The pin 2 is INT0.

pixels.begin();

//start the neopixels

sendData("AT+RST\r\n", 2000, false); // reset module

//setup command for the wifi module

sendData("AT+CWMODE=3\r\n", 1000, false);

// configure as

//access point

sendData("AT+CWLAP\r\n", 7000, false);

sendData("AT+CIFSR\r\n", 1000, DEBUG);

// get ip address

sendData("AT+CWSAP?\r\n", 2000, DEBUG);

// get SSID info

sendData("AT+CIPMUX=1\r\n", 1000, false);

// configure for multiple connections

sendData("AT+CIPSERVER=1,80\r\n", 1000, false);

// turn on

}

int angry\_clock = 0, cnt = 0;

//variables to eep track of the change of the behaviour

unsigned long previousState = 0;

int offTime = 10000;

int onTime = 100;

volatile int alarm\_on = 0;

void loop() {

// put your main code here, to run repeatedly:

int value = analogRead(sensorPin);

unsigned long currentState = millis();

if ( alarm\_on == 1) {

//if alarm is on call the update function for the buzzer adn the 9 objects created previously. If

//the alarm enters in the angry clock mode only te buzzer and the three object are called

buzz2.Update();

if (angry\_clock == 1 )

{

pixel\_1.Update();

pixel\_2.Update();

pixel\_3.Update();

buzz.Update();

}

else {

pixel\_4.Update2();

pixel\_5.Update2();

pixel\_6.Update2();

pixel\_7.Update2();

pixel\_8.Update2();

pixel\_9.Update2();

pixel\_10.Update2();

pixel\_11.Update2();

pixel\_12.Update2();

}

if (cnt < 8000 )

//count that keeps the alarm clock stuck in angry clock mode for a while

{cnt ++;

// Serial.println (cnt);

}

else

{ cnt = 0;

angry\_clock = 0;

pixel\_1.Reset();

pixel\_2.Reset();

pixel\_3.Reset();

buzz.Reset();

}

if (value >= ThresHOLD)

//piezoelectric sensor is used to see the pressure the user applies on the sensor in order to hit snooze. If is larger than the

{

//reference thresholf, then the alram enters in the angry clock behaviour

// digitalWrite(LED\_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)

Serial.println("KNOCK!");

angry\_clock = 1;

}

pixels.clear();

//reset neopixels

}

else { buzz.Reset(); //buzz2.Reset();

//reset all objects after alarm if off

pixel\_4.Reset();

pixel\_5.Reset();

pixel\_6.Reset();

pixel\_7.Reset();

pixel\_8.Reset();

pixel\_9.Reset();

pixel\_10.Reset();

pixel\_11.Reset();

pixel\_12.Reset();

}

wifi\_clock\_function();

}

void wifi\_clock\_function()

//function used for the wifi connection and interpret the input given from the website.

{

RtcDateTime dt = Rtc.GetDateTime();

//get current time

if (dt.IsValid())

{

Time[12] = dt.Second() % 10 + 48;

Time[11] = dt.Second() / 10 + 48;

Time[9] = dt.Minute() % 10 + 48;

Time[8] = dt.Minute() / 10 + 48;

Time[6] = dt.Hour() % 10 + 48;

Time[5] = dt.Hour() / 10 + 48;

}

//put it in proper order in a string in order to be able to compare it with the webiste input

if ( hours[0] == Time[5] && hours[1] == Time[6] && minutes[0] == Time[8] && minutes[1] == Time[9]) //if they are the same , ring the alarm

{ if( alarm\_on\_int == 0 )

//a control variable is used in order to check if an interrupt was done while ringing the alarm.

alarm\_on = 1;

//If no ring till the minute passes. If the interrupt was called, then turn off the alarm, even though

else alarm\_on = 0;

//the minute and hours of the input are the same with the one's given by the RTC clock

}

else { alarm\_on\_int = 0;}

if (Serial1.available()) {

//wifi interpretation

// read HTTP request

char c = Serial1.read();

if (Serial1.find("+IPD,")) {

delay(500);

int connectionId = Serial1.read() - 48; // read()

String header = "HTTP/1.1 200 OK\r\nContent-type:text/html\r\nConnection: close\r\n"; //used a header in roder to be abel to run it on iphone.Because apple phones have a rigouros security polices.

String webclient = "\r\n<!DOCTYPE html>\r\n<html>\r\n";

webclient += "<p>\r\n When you set your alarm. Respect the format (ex. 03:00) </p>\r\n"; //the user should respect the format if he wants teh data to be interpreted correctly

webclient += "<form action=\"/get\"> Alarm: <input type=\"text\" name=\"alarm\"><input type=\"submit\" value=\"Submit\">\r\n</form>";

if ( hours[0] == Time[5] && hours[1] == Time[6] && minutes[0] == Time[8] && minutes[1] == Time[9]) //teh website shows the alarm is ringing

{// Serial.println("alarm");

webclient += "<p>\r\n Alarm is ringing \r\n</p>\r\n ";

}

String cipSend = "AT+CIPSEND=";

cipSend += connectionId;

cipSend += ",";

cipSend += webclient.length() + header.length();

cipSend += "\r\n";

sendData(cipSend, 100, DEBUG);

sendData(header, 150, DEBUG);

sendData(webclient, 400, DEBUG);

String closeCommand = "AT+CIPCLOSE=";

closeCommand += connectionId; // append connection id

closeCommand += "\r\n";

sendData(closeCommand, 300, DEBUG);

}

} }

String sendData(String command, const int timeout, boolean debug)

//sendData function used for the interpretation of serial communcation between board and wifi module.

{

String response = "";

Serial1.print(command); // send command to the esp8266

long int time = millis();

while ((time + timeout) > millis()) {

while (Serial1.available()) {

char c = Serial1.read(); // read next char

response += c;

}

}

//check the hhtp response to see if the user has submited an input and parse it. After parse sootre it in some global variables

if (response.indexOf("GET /get?alarm=") >= 0)

{ int index = response.indexOf("GET /get?alarm=");

//in order to be able to compare it with the RTC values each time the loops executes

//Serial.println("AICI");

Serial.println(response[ index + 15]);

Serial.println(response[ index + 16]);

Serial.println(response[ index + 20]);

Serial.println(response[ index + 21]);

hours[0] = response[index + 15];

hours[1] = response[index + 16];

minutes[0] = response[ index + 20];

minutes[1] = response[index + 21];

String s = hours;

s += minutes;

}

if (debug) {

Serial.print(response);

}

return response;

}

//intrerrupt function that changes the state to turn off alarm and changes the control variable

void buttonInterrupt()

{

alarm\_on\_int = 1;

alarm\_on = 0;

}