Name: John Hendrie

Student ID: 0571667

Course: ELD – 311 Microprocessors

Date: 1/25/20

Final Project Build Phase

**Objective:**

The purpose of this lab is to build a standalone prototype Arduino Uno clock with the skills learned throughout the labs of this course. This clock gadget will expand and test our abilities to control devices and manipulate them to do necessary tasks. This is a steppingstone in creating more complex devices.

**Procedure:**

The final project build phase requires the creation of a clock gadget capable of ongoing display of time, temperature, and humidity. The other capabilities included the design is Date in MM/YY, day of week on demand, setting or editing time, and setting of an alarm. This will be a standalone gadget utilizing the 9-volt battery connector and jack. Time information will be saved on the DS3231 I2C with the battery that comes with the RTC.

Modules and components required for build:

Arduino UNO board (Mega 2560)

1602 LCD Display

DS3231 I2C real time clock

Piezo buzzer

9-volt connector jack

Additional LEDs, pushbuttons, resistors, wires, necessary or completion.

**Conclusion:**

This lab showed the amount of time that needs to be put into a project to make it function properly. It is understandable why it takes years to debug and perfect software and hardware for complex machines let alone this alarm clock gadget. I Frankenstein some code and modified it to meet the requirements, but I also had difficulty with setup and function. The code did not want to work with the DHT11 then I was able to add that and subtract the temp from the DS3231 module. I had issues getting the buttons to work then I added the pull-down resistors and the buttons started to work better then not at all. This lab I have is still a work in progress, but I am proud with how far I came and took that code and made it my own. I am not complete and will continue to perfect the project until completion. This was a solid base in understanding how to create systems from software to hardware and the challenges that might be faced during the journey.

**Citation:**

Dejanra. (n.d.). dejanra/RTC\_alarm\_clock. Retrieved from <https://github.com/dejanra/RTC_alarm_clock>

I also have all the includes libraries used in the project.

**Arduino Sketch:**

/\*Arduino & DS3231 SPIDER-MAN Clock Gadget

LCD (I2C module):

SCL - 21

SDA - 20

VCC

GND

\* \*/

#include <SimpleDHT.h>

#include <LiquidCrystal\_I2C.h> // https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library

#include <JC\_Button.h>

#include <DS3232RTC.h> //http://github.com/JChristensen/DS3232RTC

#include <Streaming.h> //http://arduiniana.org/libraries/streaming/

#include <Time.h> //http://www.arduino.cc/playground/Code/Time

#include <Wire.h> //http://arduino.cc/en/Reference/Wire (included with Arduino IDE)

#include <EEPROM.h>

int pinDHT11 = 52;

SimpleDHT11 dht11(pinDHT11);

LiquidCrystal\_I2C lcd(0x27, 16, 2); // address LCD(16x2) 0x27 (or 0x3F)

const byte

BUTTON\_PIN\_UP(22),

BUTTON\_PIN\_DOWN(26),

BUTTON\_PIN\_MENU\_SELECT(30),

BUTTON\_PIN\_btnBACK(34);

#define SQW\_PIN 2 //interrupt Alarm pin

#define DEBOUNCE\_MS 20 //A debounce time of 20 milliseconds usually works well for tactile button switches.

#define REPEAT\_FIRST 1000 //ms required before repeating on long press

#define REPEAT\_INCR 200 //repeat interval for long press

#define PULLUP true //To keep things simple, we use the Arduino's internal pullup resistor.

#define INVERT true

const int count; //The number that is adjusted

int lastCount = -1; //Previous value of count (initialized to ensure it's different when the sketch starts)

unsigned long rpt = REPEAT\_FIRST; //A variable time that is used to drive the repeats for long presses

Button buttonUp(BUTTON\_PIN\_UP, PULLUP, INVERT, DEBOUNCE\_MS); //Declare the buttons

Button buttonDown(BUTTON\_PIN\_DOWN, PULLUP, INVERT, DEBOUNCE\_MS);

Button buttonSelect(BUTTON\_PIN\_MENU\_SELECT, PULLUP, INVERT, DEBOUNCE\_MS);

Button buttonbtnBACK(BUTTON\_PIN\_btnBACK, PULLUP, INVERT, DEBOUNCE\_MS);

// input actions

enum {btnMENU, btnBACK, btnUP, btnDOWN};

/\*

States of FSM

\*/

enum STATES

{

MAIN,

MENU\_SET\_ALARM,

MENU\_SET\_TIME,

MENU\_SET\_DATE,

SET\_HOUR,

SET\_MINUTE,

SET\_DAY,

SET\_MONTH,

SET\_YEAR,

SET\_AL\_HOUR,

SET\_AL\_MINUTE,

SET\_AL\_ON\_OFF,

ALARM\_TIME,

// Otherwise, it times out after 5 seconds, discards the changes and returns to displaying the time

};

STATES state; // Holds the current state of the system

int8\_t button;

int8\_t trigger;

int Second;

int Minute;

int Hour;

int Day;

int Month;

int Year;

int8\_t DoW;

String day\_of\_week;

unsigned char address, data;

int AL\_Hour;

int AL\_Minute;

bool alarm\_active = false;

int alarm\_out = 10;

uint32\_t blink\_interval = 500;

uint32\_t blink\_previousMillis = 0;

boolean blink\_state = false;

boolean RTC\_error = true;

boolean long\_press\_button = false;

byte bell\_symbol[8] = {

B00100,

B01110,

B01110,

B01110,

B01110,

B11111,

B00000,

B00100

};

byte menu\_RIGHT[8] = {

B10000,

B11000,

B11100,

B11110,

B11100,

B11000,

B10000,

B00000

};

byte menu\_LEFT[8] = {

B00001,

B00011,

B00111,

B01111,

B00111,

B00011,

B00001,

B00000

};

/\*

SETUP

\*/

void setup()

{

lcd.init();

lcd.backlight();

Wire.begin(); // start I2C

pinMode(11, INPUT\_PULLUP);

pinMode(alarm\_out, OUTPUT);

lcd.createChar(1, bell\_symbol);

lcd.createChar(5, menu\_RIGHT);

lcd.createChar(6, menu\_LEFT);

Wire.begin();

setSyncProvider(RTC.get); // set RTC as the Syncprovider

setSyncInterval(5); // time in sec of resync with RTC

pinMode(SQW\_PIN, INPUT\_PULLUP);

attachInterrupt(INT0, alarmIsr, FALLING);

lcd.clear();

lcd.setCursor(3, 0);

lcd.print("SPIDER-MAN");

lcd.setCursor(2,1);

lcd.print("Clock Gadget");

delay(2000);

lcd.clear();

state = MAIN;

}//End SETUP

volatile boolean alarmIsrWasCalled = false;

void alarmIsr()

{

alarmIsrWasCalled = true;

}

/\*

LOOP

\*/

void loop()

{

change\_states(); // change states of FSM

check\_inputs(); // check inputs (buttons)

check\_RTC(); // check connection RTC

if (alarmIsrWasCalled) {

if (RTC.alarm(ALARM\_1)) {

lcd.clear();

lcd.setCursor(1, 0);

lcd.print("Spiderman!!!");

lcd.setCursor(1, 1);

lcd.print("We need you!!!");

for (int i = 0; i < 25; i++) {

buttonbtnBACK.read();

if (buttonbtnBACK.wasPressed())

{

alarmIsrWasCalled = false;

break;

}

beep();

}

}

lcd.clear();

}

alarmIsrWasCalled = false;

}// End of LOOP

/\*

Finite State Machine

\*/

void change\_states()

{

//states

switch (state)

{

case MAIN:

display\_time();

display\_date();

get\_alarm(); //read stored alarm time

display\_temperature();

break;

case MENU\_SET\_ALARM:

menu\_set\_alarm();

break;

case MENU\_SET\_TIME:

menu\_set\_time();

break;

case MENU\_SET\_DATE:

menu\_set\_date();

break;

case SET\_HOUR:

display\_set\_time();

break;

case SET\_MINUTE:

display\_set\_time();

break;

case SET\_DAY:

display\_set\_date();

break;

case SET\_MONTH:

display\_set\_date();

break;

case SET\_YEAR:

display\_set\_date();

break;

case SET\_AL\_HOUR:

display\_set\_AL\_time();

break;

case SET\_AL\_MINUTE:

display\_set\_AL\_time();

break;

case SET\_AL\_ON\_OFF:

display\_set\_AL\_time();

break;

case ALARM\_TIME:

displayALARM();

delay(2000);

lcd.clear();

state = MAIN;

break;

break;

}

}

/\*

Check INPUT

\*/

void check\_inputs()

{

buttonUp.read(); // check state of buttons

buttonDown.read();

buttonSelect.read();

buttonbtnBACK.read();

switch (buttonSelect.wasPressed()) // check MENU/SELECT button

{

case 1:

button = btnMENU;

transition(button);

break;

}

switch (buttonUp.wasPressed()) // check UP button

{

case 1:

button = btnUP;

transition(button);

break;

}

switch (buttonUp.wasReleased()) // check long press UP button

{

// case 1:

// long\_press\_button = false;

// rpt = REPEAT\_FIRST;

// break;

}

switch (buttonUp.pressedFor(rpt))

{

case 1:

rpt += REPEAT\_INCR; //increment the long press interval

long\_press\_button = true;

button = btnUP;

transition(button);

break;

}

switch (buttonDown.wasPressed()) // check DOWN button

{

case 1:

button = btnDOWN;

transition(button);

break;

}

switch (buttonDown.wasReleased()) // check long press DOWN button

{

// case 1:

// long\_press\_button = false;

// rpt = REPEAT\_FIRST;

// break;

}

switch (buttonDown.pressedFor(rpt))

{

case 1:

rpt += REPEAT\_INCR; //increment the long press interval

long\_press\_button = true;

button = btnDOWN;

transition(button);

break;

}

switch (buttonbtnBACK.wasPressed()) // check btnBACK button

{

case 1:

button = btnBACK;

transition(button);

break;

}

}

void transition(int trigger)

{

switch (state)

{

case MAIN:

if (trigger == btnMENU)

{

lcd.clear();

state = MENU\_SET\_ALARM;

}

else if (trigger == btnBACK)

{

lcd.clear();

state = ALARM\_TIME;

}

break;

case MENU\_SET\_ALARM:

if (trigger == btnUP)

{

lcd.clear();

state = MENU\_SET\_TIME;

}

else if (trigger == btnUP)

{

lcd.clear();

state = MENU\_SET\_DATE;

}

if (trigger == btnMENU)

{

lcd.clear();

state = SET\_AL\_HOUR;

}

if (trigger == btnBACK)

{

lcd.clear();

state = MAIN;

}

break;

case MENU\_SET\_TIME:

if (trigger == btnUP)

{

lcd.clear();

state = MENU\_SET\_DATE;

}

if (trigger == btnDOWN)

{

lcd.clear();

state = MENU\_SET\_ALARM;

}

if (trigger == btnMENU)

{

lcd.clear();

state = SET\_HOUR;

}

if (trigger == btnBACK)

{

lcd.clear();

state = MAIN;

}

break;

case MENU\_SET\_DATE:

if (trigger == btnDOWN)

{

lcd.clear();

state = MENU\_SET\_TIME;

}

else if (trigger == btnDOWN)

{

lcd.clear();

state = MENU\_SET\_ALARM;

}

if (trigger == btnMENU)

{

lcd.clear();

state = SET\_DAY;

}

if (trigger == btnBACK)

{

lcd.clear();

state = MAIN;

}

break;

case SET\_HOUR:

if (trigger == btnUP)

{

Hour++;

if (Hour > 23) Hour = 0;

}

else if (trigger == btnDOWN)

{

Hour--;

if (Hour < 0) Hour = 23;

}

if (trigger == btnMENU)

{

state = SET\_MINUTE;

}

break;

case SET\_MINUTE:

if (trigger == btnUP)

{

Minute++;

if (Minute > 59) Minute = 0;

}

else if (trigger == btnDOWN)

{

Minute--;

if (Minute < 0) Minute = 59;

}

if (trigger == btnBACK)

{

state = SET\_HOUR;

}

if (trigger == btnMENU)

{

set\_time();

lcd.clear();

lcd.setCursor(2, 0);

lcd.print("Your on Time!");

delay(1000);

lcd.clear();

state = MAIN;

}

break;

case SET\_DAY:

if (trigger == btnUP)

{

Day++;

if (Day > 31) Day = 1;

}

else if (trigger == btnDOWN)

{

Day--;

if (Day < 1) Day = 31;

}

if (trigger == btnMENU)

{

state = SET\_MONTH;

}

break;

case SET\_MONTH:

if (trigger == btnUP)

{

Month++;

if (Month > 12) Month = 1;

}

else if (trigger == btnDOWN)

{

Month--;

if (Month < 1) Month = 12;

}

if (trigger == btnMENU)

{

state = SET\_YEAR;

}

if (trigger == btnBACK)

{

state = SET\_DAY;

}

break;

case SET\_YEAR:

if (trigger == btnUP)

{

Year++;

if (Year > 99) Year = 0;

}

else if (trigger == btnDOWN)

{

Year--;

if (Year < 0) Year = 99;

}

if (trigger == btnBACK)

{

state = SET\_MONTH;

}

if (trigger == btnMENU)

{

set\_date();

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("The Date is Set!");

delay(1000);

lcd.clear();

state = MAIN;

}

break;

case SET\_AL\_HOUR:

if (trigger == btnUP)

{

AL\_Hour++;

if (AL\_Hour > 23) AL\_Hour = 0;

}

else if (trigger == btnDOWN)

{

AL\_Hour--;

if (AL\_Hour < 0) AL\_Hour = 23;

}

if (trigger == btnMENU)

{

state = SET\_AL\_MINUTE;

}

break;

case SET\_AL\_MINUTE:

if (trigger == btnUP)

{

AL\_Minute++;

if (AL\_Minute > 59) AL\_Minute = 0;

}

else if (trigger == btnDOWN)

{

AL\_Minute--;

if (AL\_Minute < 0) AL\_Minute = 59;

}

if (trigger == btnMENU)

{

state = SET\_AL\_ON\_OFF;

}

if (trigger == btnBACK)

{

state = SET\_AL\_HOUR;

}

break;

case SET\_AL\_ON\_OFF:

if (trigger == btnUP)

{

alarm\_active = true;

}

else if (trigger == btnDOWN)

{

alarm\_active = false;

}

if (trigger == btnBACK)

{

state = SET\_AL\_MINUTE;

}

if (trigger == btnMENU)

{

set\_alarm();

lcd.clear();

lcd.setCursor(1, 0);

lcd.print("Your wake up!");

lcd.setCursor(1, 1);

lcd.print("is activated!");

delay(2000);

lcd.clear();

state = MAIN;

}

break;

}

}

/\*

FUNCTIONS

\*/

void beep() {

digitalWrite(alarm\_out, HIGH);

lcd.noBacklight();

delay(500);

digitalWrite(alarm\_out, LOW);

lcd.backlight();

delay(500);

}

void menu\_set\_time() {

lcd.setCursor(3, 0);

lcd.print("CLOCK MENU");

lcd.setCursor(2, 1);

lcd.print("Set the Time");

lcd.setCursor(15, 1);

lcd.write(5);

lcd.setCursor(0, 1);

lcd.write(6);

}

void menu\_set\_date() {

lcd.setCursor(3, 0);

lcd.print("CLOCK MENU");

lcd.setCursor(2, 1);

lcd.print("Set the Date");

lcd.setCursor(0, 1);

lcd.write(6);

}

void menu\_set\_alarm() {

lcd.setCursor(3, 0);

lcd.print("CLOCK MENU");

lcd.setCursor(1, 1);

lcd.print("Set the Alarm");

lcd.setCursor(15, 1);

lcd.write(5);

}

void display\_time()

{

get\_time();

lcd.setCursor(0, 0);

display\_position(Hour);

lcd.print(":");

display\_position(Minute);

lcd.print(":");

display\_position(Second);

}

void display\_date() {

get\_date();

lcd.setCursor(5, 1);

display\_position(Month);

lcd.print("/");

display\_position(Year);

DoW = weekday();

switch (DoW) {

case 1: day\_of\_week = "Sun"; break;

case 2: day\_of\_week = "Mon"; break;

case 3: day\_of\_week = "Tue"; break;

case 4: day\_of\_week = "Wed"; break;

case 5: day\_of\_week = "Thu"; break;

case 6: day\_of\_week = "Fri"; break;

case 7: day\_of\_week = "Sat"; break;

}

lcd.setCursor(0, 1);

lcd.print (day\_of\_week);

}

void display\_set\_time() {

lcd.setCursor(3, 0);

lcd.print("Set Time:");

switch (state)

{

case SET\_HOUR:

lcd.setCursor(10, 1);

lcd.print("h");

if (long\_press\_button == false) {

if (blink\_state == 0) {

lcd.setCursor(4, 1);

display\_position(Hour);

}

else {

lcd.setCursor(4, 1);

lcd.print(" ");

}

}

else {

lcd.setCursor(4, 1);

display\_position(Hour);

}

lcd.print(":");

display\_position(Minute);

break;

case SET\_MINUTE:

lcd.setCursor(10, 1);

lcd.print("m");

lcd.setCursor(4, 1);

display\_position(Hour);

lcd.print(":");

if (long\_press\_button == false) {

if (blink\_state == 0) {

lcd.setCursor(7, 1);

display\_position(Minute);

}

else {

lcd.setCursor(7, 1);

lcd.print(" ");

}

}

else {

lcd.setCursor(7, 1);

display\_position(Minute);

}

break;

}

unsigned long blink\_currentMillis = millis();

if (blink\_currentMillis - blink\_previousMillis > blink\_interval) {

blink\_previousMillis = blink\_currentMillis;

if (blink\_state == 0) {

blink\_state = 1;

} else {

blink\_state = 0;

}

}

}

void display\_set\_date() {

lcd.setCursor(3, 0);

lcd.print("Set Date:");

switch (state)

{

case SET\_DAY:

if (long\_press\_button == false) {

if (blink\_state == 0) {

lcd.setCursor(4, 1);

display\_position(Day);

}

else {

lcd.setCursor(4, 1);

lcd.print(" ");

}

}

else {

lcd.setCursor(4, 1);

display\_position(Day);

}

lcd.print("/");

display\_position(Month);

lcd.print("/");

display\_position(Year);

break;

case SET\_MONTH:

lcd.setCursor(4, 1);

display\_position(Day);

lcd.print("/");

if (long\_press\_button == false) {

if (blink\_state == 0) {

lcd.setCursor(7, 1);

display\_position(Month);

}

else {

lcd.setCursor(7, 1);

lcd.print(" ");

}

} else {

lcd.setCursor(7, 1);

display\_position(Month);

}

lcd.print("/");

display\_position(Year);

break;

case SET\_YEAR:

lcd.setCursor(4, 1);

display\_position(Day);

lcd.print("/");

display\_position(Month);

lcd.print("/");

if (long\_press\_button == false) {

if (blink\_state == 0) {

lcd.setCursor(10, 1);

display\_position(Year);

}

else {

lcd.setCursor(10, 1);

lcd.print(" ");

}

}

else {

lcd.setCursor(10, 1);

display\_position(Year);

}

break;

}

unsigned long blink\_currentMillis = millis();

if (blink\_currentMillis - blink\_previousMillis > blink\_interval) {

blink\_previousMillis = blink\_currentMillis;

if (blink\_state == 0) {

blink\_state = 1;

} else {

blink\_state = 0;

}

}

}

void display\_set\_AL\_time() {

lcd.setCursor(1, 0);

lcd.print("Set Alarm Time:");

if (alarm\_active == true) {

lcd.setCursor(12, 1);

lcd.print("ON");

} else {

lcd.setCursor(12, 1);

lcd.print("OFF");

}

switch (state)

{

case SET\_AL\_HOUR:

lcd.setCursor(10, 1);

lcd.print("H");

if (long\_press\_button == false) {

if (blink\_state == 0) {

lcd.setCursor(4, 1);

display\_position(AL\_Hour);

}

else {

lcd.setCursor(4, 1);

lcd.print(" ");

}

}

else {

lcd.setCursor(4, 1);

display\_position(AL\_Hour);

}

lcd.print(":");

display\_position(AL\_Minute);

break;

case SET\_AL\_MINUTE:

lcd.setCursor(10, 1);

lcd.print("M");

lcd.setCursor(4, 1);

display\_position(AL\_Hour);

lcd.print(":");

if (long\_press\_button == false) {

if (blink\_state == 0) {

lcd.setCursor(7, 1);

display\_position(AL\_Minute);

}

else {

lcd.setCursor(7, 1);

lcd.print(" ");

}

}

else {

lcd.setCursor(7, 1);

display\_position(AL\_Minute);

}

break;

case SET\_AL\_ON\_OFF:

lcd.setCursor(7, 1);

display\_position(AL\_Minute);

if (alarm\_active == true) {

if (blink\_state == 0) {

lcd.setCursor(14, 1);

lcd.print(" ");

lcd.setCursor(12, 1);

lcd.print("ON");

}

else {

lcd.setCursor(12, 1);

lcd.print(" ");

}

}

else {

if (blink\_state == 0) {

lcd.setCursor(12, 1);

lcd.print("OFF");

}

else {

lcd.setCursor(12, 1);

lcd.print(" ");

}

}

break;

}

unsigned long blink\_currentMillis = millis();

if (blink\_currentMillis - blink\_previousMillis > blink\_interval) {

blink\_previousMillis = blink\_currentMillis;

if (blink\_state == 0) {

blink\_state = 1;

} else {

blink\_state = 0;

}

}

}

void displayALARM() {

lcd.setCursor(2, 0);

lcd.print("<< ALARM >>");

lcd.setCursor(3, 1);

display\_position(AL\_Hour);

lcd.print(":");

lcd.setCursor(6, 1);

display\_position(AL\_Minute);

lcd.setCursor(9, 1);

lcd.print("H");

if (alarm\_active == true) {

lcd.setCursor(13, 1);

lcd.print(" ");

lcd.setCursor(11, 1);

lcd.print("ON");

}

else {

lcd.setCursor(12, 1);

lcd.print("OFF");

}

}

void get\_time() {

Wire.beginTransmission(0x68);

Wire.write(0); //set register to zero

Wire.endTransmission();

Wire.requestFrom(0x68, 3);// 3 bytes (sec, min, hour)

Second = bcdToDec(Wire.read() & 0x7f);

Minute = bcdToDec(Wire.read());

Hour = bcdToDec(Wire.read() & 0x3f);

}

void get\_date() {

Wire.beginTransmission(0x68);

Wire.write(4);//set register to 3 (day)

Wire.endTransmission();

Wire.requestFrom(0x68, 3); // 3 bytes (day, month, year) - DOW get from Time.h library

Day = bcdToDec(Wire.read());

Month = bcdToDec(Wire.read());

Year = bcdToDec(Wire.read());

}

void set\_time() {

Wire.beginTransmission(0x68);

Wire.write(0x00);

Second = 0;

Wire.write(decToBcd(Second));

Wire.write(decToBcd(Minute));

Wire.write(decToBcd(Hour));

Wire.write(0x00);

Wire.endTransmission();

}

void set\_date() {

Wire.beginTransmission(0x68);

Wire.write(4);

//Wire.write(decToBcd(DoW));

Wire.write(decToBcd(Day));

Wire.write(decToBcd(Month));

Wire.write(decToBcd(Year));

Wire.endTransmission();

}

void set\_alarm() { //save alarm time to EEPROM

EEPROM.write(0, AL\_Hour);

EEPROM.write(1, AL\_Minute);

EEPROM.write(2, alarm\_active);

RTC.setAlarm(ALM1\_MATCH\_HOURS, 0, AL\_Minute, AL\_Hour, 0); //sec,min,hour RTC.alarm(ALARM\_1); //ensure RTC interrupt flag is cleared

if (alarm\_active == true) {

RTC.alarmInterrupt(ALARM\_1, true);

}

else {

RTC.alarmInterrupt(ALARM\_1, false);

}

}

void get\_alarm() { //read alarm time from EEPROM

AL\_Hour = EEPROM.read(0);

if (AL\_Hour > 23) AL\_Hour = 0;

AL\_Minute = EEPROM.read(1);

if (AL\_Minute > 59) AL\_Minute = 0;

alarm\_active = EEPROM.read(2);

if (alarm\_active == true) {

lcd.setCursor(9, 0);

lcd.write(1);

}

else {

lcd.setCursor(5, 1);

lcd.print(" ");

}

}

void display\_temperature() {

int temp\_C = RTC.temperature() / 4;

lcd.setCursor(12, 0);

if (temp\_C < 10) {

lcd.print('0');

}

// lcd.print(temp\_C);

// lcd.print((char)223);

// lcd.print("C");

// read without samples.

byte temperature = 0;

byte humidity = 0;

int err = SimpleDHTErrSuccess;

if ((err = dht11.read(&temperature, &humidity, NULL)) != SimpleDHTErrSuccess) {

Serial.print("Read DHT11 failed, err="); Serial.println(err);delay(1000);

return;

}

lcd.setCursor(12,0);

lcd.print((int)temperature);

lcd.print((char)223);

lcd.print("C");

lcd.setCursor(12,1);

lcd.print((int)humidity);

lcd.print(" %");

delay(1000);

}

// check communication with RTC

void check\_RTC()

{

if (timeStatus() != timeSet)

RTC\_error = true; // RTC could not be read

else

RTC\_error = false; // RTC could be read

}

byte decToBcd(byte val) {

return ( (val / 10 \* 16) + (val % 10) );

}

byte bcdToDec(byte val) {

return ( (val / 16 \* 10) + (val % 16) );

}

void display\_position(int digits) {

if (digits < 10)

lcd.print("0");

lcd.print(digits);

}