<https://create.arduino.cc/editor/savvtann/f5b70057-d6bc-4e5e-b741-075201e928f0/preview>

//FINAL BLOCK 1, 2 VERIFICATION

// SENSOR CONTROL BLOCK

#include <Wire.h>

#include "Adafruit\_TCS34725.h"

#include <OneWire.h>

#include <DallasTemperature.h>

#include <TimeLib.h>

#define TEMPERATURE\_BUS 8 // Digital pin where the DS18B20 is connected

#define SDA\_PIN 21

#define SCL\_PIN 22

#define BUTTON\_PIN 7

#define POTENTIOMETER\_PIN A0

#define BUTTON\_THRESHOLD 10

// Color Sensor Variables

//Adafruit\_TCS34725 tcs = Adafruit\_TCS34725(TCS34725\_INTEGRATIONTIME\_614MS, TCS34725\_GAIN\_1X);

// Thermometer Variables

OneWire Temperature(TEMPERATURE\_BUS);

DallasTemperature sensors(&Temperature);

// Button and Potentiometer variables

int lastButtonState = HIGH; // the previous state from the input pin

int currentButtonState; // the current reading from the input pin

int lastPotValue = 0; // the previous value from the potentiometer

int currentPotValue; // the current reading from the potentiometer

volatile bool buttonPressed = false;

// Timekeeping variables

unsigned long previousMillis = 0;

unsigned long lastElapsedTimeUpdate = 0;

const long interval = 3000; // Update every 3 second

/\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// PARAMETER COMPARISON FUNCTIONS

// Function to compare RGB values to predefined PH colors

void compareColorToPH(uint16\_t r, uint16\_t g, uint16\_t b) {

// Predefined pH color values

uint16\_t phColors[][3] = {

{236, 148, 0}, // pH 6

{255, 168, 0}, // pH 6.5

{255, 150, 29}, // pH 7

{253, 105, 20}, // pH 7.5

{222, 48, 4}, // pH 8

{226, 26, 11}, // pH 8.5

{230, 4, 28} // pH 9

};

// Calculate the Euclidean distance for each pH color

float minDistance = FLT\_MAX;

int closestPHIndex = -1;

for (int i = 0; i < sizeof(phColors) / sizeof(phColors[0]); i++) {

float distance = sqrt(pow(r - phColors[i][0], 2) + pow(g - phColors[i][1], 2) + pow(b - phColors[i][2], 2));

if (distance < minDistance) {

minDistance = distance;

closestPHIndex = i;

}

}

// Display the closest pH level

float closestPH = 6.0 + closestPHIndex \* 0.5;

Serial.print("Detected pH level: ");

Serial.println(closestPH);

}

// Function to compare RGB values to predefined GH colors

void compareColorToGH(uint16\_t r, uint16\_t g, uint16\_t b) {

uint16\_t ghColors[][3] = {

{171, 187, 168}, // GH 0

{118, 165, 162}, // GH 30

{122, 146, 158}, // GH 60

{101, 111, 136}, // GH 120

{147, 130, 177} // GH 180

};

float minDistance = FLT\_MAX;

int closestGHIndex = -1;

for (int i = 0; i < sizeof(ghColors) / sizeof(ghColors[0]); i++) {

float distance = sqrt(pow(r - ghColors[i][0], 2) + pow(g - ghColors[i][1], 2) + pow(b - ghColors[i][2], 2));

if (distance < minDistance) {

minDistance = distance;

closestGHIndex = i;

}

}

float closestGH = closestGHIndex \* 30.0;

Serial.print("Detected GH value: ");

Serial.println(closestGH);

}

// Function to compare RGB values to predefined KH colors

void compareColorToKH(uint16\_t r, uint16\_t g, uint16\_t b) {

uint16\_t khColors[][3] = {

{239, 180, 66}, // KH 0

{194, 156, 60}, // KH 40

{181, 173, 112}, // KH 80

{142, 148, 102}, // KH 120

{158, 172, 139} // KH 180

};

float minDistance = FLT\_MAX;

int closestKHIndex = -1;

for (int i = 0; i < sizeof(khColors) / sizeof(khColors[0]); i++) {

float distance = sqrt(pow(r - khColors[i][0], 2) + pow(g - khColors[i][1], 2) + pow(b - khColors[i][2], 2));

if (distance < minDistance) {

minDistance = distance;

closestKHIndex = i;

}

}

float closestKH = closestKHIndex \* 40.0;

Serial.print("Detected KH value: ");

Serial.println(closestKH);

}

// Function to compare RGB values to predefined NO2 colors

void compareColorToNO2(uint16\_t r, uint16\_t g, uint16\_t b) {

uint16\_t no2Colors[][3] = {

{239, 177, 142}, // NO2 0

{250, 182, 150}, // NO2 0.5

{232, 158, 133}, // NO2 1

{227, 127, 116}, // NO2 3

{250, 126, 114} // NO2 5

};

float minDistance = FLT\_MAX;

int closestNO2Index = -1;

for (int i = 0; i < sizeof(no2Colors) / sizeof(no2Colors[0]); i++) {

float distance = sqrt(pow(r - no2Colors[i][0], 2) + pow(g - no2Colors[i][1], 2) + pow(b - no2Colors[i][2], 2));

if (distance < minDistance) {

minDistance = distance;

closestNO2Index = i;

}

}

float closestNO2 = closestNO2Index \* 0.5;

Serial.print("Detected NO2 value: ");

Serial.println(closestNO2);

}

// Function to compare RGB values to predefined NO3 colors

void compareColorToNO3(uint16\_t r, uint16\_t g, uint16\_t b) {

uint16\_t no3Colors[][3] = {

{244, 184, 149}, // NO3 0

{254, 184, 153}, // NO3 20

{248, 165, 146}, // NO3 40

{230, 123, 103}, // NO3 80

{230, 93, 87} // NO3 160

};

float minDistance = FLT\_MAX;

int closestNO3Index = -1;

for (int i = 0; i < sizeof(no3Colors) / sizeof(no3Colors[0]); i++) {

float distance = sqrt(pow(r - no3Colors[i][0], 2) + pow(g - no3Colors[i][1], 2) + pow(b - no3Colors[i][2], 2));

if (distance < minDistance) {

minDistance = distance;

closestNO3Index = i;

}

}

float closestNO3 = closestNO3Index \* 20.0;

Serial.print("Detected NO3 value: ");

Serial.println(closestNO3);

}

\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// TEMPERATURE FUNCTIONS

void temperatureSensor() {

sensors.requestTemperatures();

float temperatureC = sensors.getTempCByIndex(0);

if (temperatureC != DEVICE\_DISCONNECTED\_C) {

// Celsius

Serial.print("Temperature: ");

Serial.print(temperatureC);

Serial.println("°C");

// Fahrenheit

Serial.print("Temperature: ");

Serial.print((temperatureC \* 9/5) + 32);

Serial.println("°F");

Serial.println();

} else {

Serial.println("Error reading temperature!");

}

}

/\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// COLOR FUNCTIONS

void printHexColor(uint16\_t r, uint16\_t g, uint16\_t b) {

unsigned long hexColor = ((unsigned long)r << 16) | ((unsigned long)g << 8) | b;

Serial.print("HEX Color: 0x");

Serial.println(hexColor, HEX);

}

void html\_rgb(uint16\_t r, uint16\_t g, uint16\_t b, uint16\_t c, float (&rgb)[3]) {

float factor = 256.0 / c;

rgb[0] = pow((r \* factor) / 255.0, 2.5) \* 255;

rgb[1] = pow((g \* factor) / 255.0, 2.5) \* 255;

rgb[2] = pow((b \* factor) / 255.0, 2.5) \* 255;

}

String html\_hex(uint16\_t r, uint16\_t g, uint16\_t b, uint16\_t c) {

float rgb[3];

html\_rgb(r, g, b, c, rgb);

return String(String((int)rgb[0], HEX) + String((int)rgb[1], HEX) + String((int)rgb[2], HEX));

}

void colorSensor() {

uint16\_t r, g, b, c, colorTemp, lux;

tcs.getRawData(&r, &g, &b, &c);

colorTemp = tcs.calculateColorTemperature(r, g, b);

lux = tcs.calculateLux(r, g, b);

Serial.print("R: "); Serial.print(r);

Serial.print(", G: "); Serial.print(g);

Serial.print(", B: "); Serial.print(b);

Serial.println();

// float rgb[3];

// html\_rgb(r, g, b, c, rgb);

// Serial.print("RGB Color: ");

// Serial.print("R: "); Serial.print((int)rgb[0]);

// Serial.print(", G: "); Serial.print((int)rgb[1]);

// Serial.print(", B: "); Serial.print((int)rgb[2]);

// Serial.println();

// String hexColor = html\_hex(r, g, b, c);

// Serial.print("HEX Color: 0x"); Serial.println(hexColor);

Serial.print(", C: "); Serial.print(c);

Serial.print(", ColorTemp: "); Serial.print(colorTemp);

Serial.print(", Lux: "); Serial.print(lux);

Serial.println();

compareColorToPH(r, g, b); // call the PH function

compareColorToGH(r, g, b); // Call the GH function

compareColorToKH(r, g, b); // Call the KH function

compareColorToNO2(r, g, b); // Call the NO2 function

compareColorToNO3(r, g, b); // Call the NO3 function

}

\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// INTERRUPT FOR BUTTON PRESS

void buttonISR() {

buttonPressed = true;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// TIME/DATE FUNCTIONS

void printElapsedTime() {

unsigned long currentMillis = millis();

Serial.print("Elapsed Time: ");

Serial.print((currentMillis - userStartTime) / 1000); // Convert milliseconds to seconds

Serial.println(" seconds");

}

void setUserDateTime() {

Serial.println("Please enter the current time (HH:MM:SS): ");

while (!Serial.available()) {

// Wait for user input

}

int hour = Serial.parseInt();

while (Serial.read() != ':') {

// Wait for the separator

}

int minute = Serial.parseInt();

while (Serial.read() != ':') {

// Wait for the separator

}

int second = Serial.parseInt();

Serial.println("Please enter the current date (YYYY-MM-DD): ");

while (!Serial.available()) {

// Wait for user input

}

int year = Serial.parseInt();

while (Serial.read() != '-') {

// Wait for the separator

}

int month = Serial.parseInt();

while (Serial.read() != '-') {

// Wait for the separator

}

int day = Serial.parseInt();

// Calculate the start time in milliseconds

userStartTime = millis() - (second \* 1000 + minute \* 60 \* 1000 + hour \* 60 \* 60 \* 1000 +

day \* 24 \* 60 \* 60 \* 1000 + month \* 30 \* 24 \* 60 \* 60 \* 1000 +

year \* 365 \* 24 \* 60 \* 60 \* 1000);

Serial.println("User input time and date set successfully!");

}

void calculateElapsedTime() {

unsigned long currentMillis = millis();

unsigned long elapsedMillis = currentMillis - userStartTime;

// Convert milliseconds to seconds, minutes, and hours

unsigned long seconds = elapsedMillis / 1000;

unsigned long minutes = seconds / 60;

unsigned long hours = minutes / 60;

Serial.print("Elapsed Time: ");

Serial.print(hours);

Serial.print(" hours, ");

Serial.print(minutes % 60);

Serial.print(" minutes, ");

Serial.print(seconds % 60);

Serial.println(" seconds");

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// GENERAL FUNCTIONS

void setup() {

// tcs.begin();

// Wire.begin(SDA\_PIN, SCL\_PIN);

// Wire.setClock(400000); // set datarate to 400kbits/sec

sensors.begin();

Serial.begin(9600);

pinMode(BUTTON\_PIN, INPUT\_PULLUP);

attachInterrupt(digitalPinToInterrupt(BUTTON\_PIN), buttonISR, FALLING);

setUserDateTime();

// if (tcs.begin()) {

// Serial.println("Found sensor");

// } else {

// Serial.println("No TCS34725 found ... check your connections");

// while (1);

// }

}

void loop() {

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// TEMPERATURE STUFF

//temperatureSensor();

//delay(5000);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// TIME STUFF

// Check if it's time to update elapsed time

unsigned long currentMillis = millis();

if (currentMillis - lastElapsedTimeUpdate >= interval) {

calculateElapsedTime(); // Update elapsed time

lastElapsedTimeUpdate = currentMillis; // Save the last update time

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// BUTTON STUFF

currentButtonState = digitalRead(BUTTON\_PIN);

if (lastButtonState == LOW && currentButtonState == HIGH) {

Serial.println("Button pressed");

// Add code here to handle button press (e.g., switch pages)

}

lastButtonState = currentButtonState;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// POTENTIOMETER STUFF

// Read the potentiometer value

currentPotValue = analogRead(POTENTIOMETER\_PIN);

// Check if the potentiometer value has changed significantly

if (abs(currentPotValue - lastPotValue) > BUTTON\_THRESHOLD) {

Serial.print("Potentiometer value: ");

Serial.println(currentPotValue);

// Add code here to handle potentiometer value change (e.g., adjust brightness)

}

lastPotValue = currentPotValue;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// COLOR SENSOR STUFF

// colorSensor();

// delay(5000);

}