// FINAL BLOCK 1 VALIDATION

// USER INTERFACE BLOCK

// AUTHOR: SAVANNAH TANNER

// DATE: 3/10/2024

#include <Wire.h>

#include "Adafruit\_TCS34725.h"

#include <OneWire.h>

#include <DallasTemperature.h>

#include <GxEPD.h>

#include <GxFont\_GFX.h>

#include <Fonts/FreeMonoBold9pt7b.h> // Change the font size here

#include <GxIO/GxIO\_SPI/GxIO\_SPI.h>

#include <GxIO/GxIO.h>

#include <GxDEPG0213BN/GxDEPG0213BN.h>

// Define SPI pin configurations

#define SPI\_MOSI 23

#define SPI\_MISO -1

#define SPI\_CLK 18

// Define E-ink display pin configurations

#define ELINK\_SS 5

#define ELINK\_BUSY 4

#define ELINK\_RESET 16

#define ELINK\_DC 17

#define TEMPERATURE\_BUS 19 // Digital pin where the DS18B20 is connected

#define SDA\_PIN 21

#define SCL\_PIN 22

#define BUTTON\_PIN 15

#define BUTTON\_THRESHOLD 3

#define POTENTIOMETER\_PIN 13

// Color Sensor Variables

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

// Thermometer Variables

OneWire Temperature(TEMPERATURE\_BUS);

DallasTemperature sensors(&Temperature);

bool tempUnits;

// Initialize SPI and E-ink display objects

GxIO\_Class io(SPI, /\*CS=5\*/ ELINK\_SS, /\*DC=\*/ ELINK\_DC, /\*RST=\*/ ELINK\_RESET);

GxEPD\_Class display(io, /\*RST=\*/ ELINK\_RESET, /\*BUSY=\*/ ELINK\_BUSY);

// 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;

int currentPage = 0;

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

// 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 = MAXFLOAT;

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 = MAXFLOAT;

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 = MAXFLOAT;

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 = MAXFLOAT;

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 = MAXFLOAT;

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) {

if (tempUnits == 0){

// Celsius

Serial.print("Temperature: ");

Serial.print(temperatureC);

Serial.println("�C");

}

if (tempUnits == 1){

// Fahrenheit

Serial.print("Temperature: ");

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

Serial.println("�F");

}

Serial.println();

}

else {

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

}

}

void chooseFahrenheitCelsius(){

Serial.println("Fahrenheit (F) or Celsius (C): ");

while (!Serial.available()) {

// Wait for user input

}

char choice = Serial.read();

if (choice == 'F'){

tempUnits = 1;

}

else if (choice == 'C'){

tempUnits = 0;

}

else{

Serial.println("Invalid choice. Please enter F for Fahrenheit or C for Celsius.");

chooseFahrenheitCelsius(); // Ask again if the choice is invalid

}

}

0

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

// 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]) {

// Calculate normalization factor

float maxRGB = max(max(r, g), b);

float factor = 255.0 / maxRGB;

// Normalize RGB values

rgb[0] = r \* factor;

rgb[1] = g \* factor;

rgb[2] = b \* factor;

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

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

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

Serial.println();

}

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();

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

float rgb[3];

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

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

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

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

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

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

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

}

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

// INTERRUPT FOR BUTTON PRESS

void buttonISR() {

buttonPressed = true;

}

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

// DISPLAY FUNCS

bool leftRightFunc(int currentPotValue){

bool leftRight;

int threshold = 4095 / 2;

if (currentPotValue > threshold) {

leftRight = true; // 1 represents true

} else {

leftRight = false; // 0 represents false

}

return leftRight;

}

void switchPages(bool leftRight){

if (leftRight) {

currentPage += 1;

if (currentPage >= 4){

currentPage = 0;

}

} else {

currentPage -= 1;

if (currentPage <= -1){

currentPage = 3;

}

}

Serial.println(currentPage);

displayPages(currentPage);

}

void displayPages(int currentPage){

int xValues[] = {0, 1, 2, 3, 4, 5};

int yValues[] = {10, 20, 15, 30, 25, 35};

int numPoints = sizeof(xValues) / sizeof(xValues[0]);

int myIntList[] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

int listLength = sizeof(myIntList) / sizeof(myIntList[0]);

switch (currentPage){

case 0:

displayText("Page 0: Example Text");

break;

case 1:

displayText("Page 1: Example Nums: \n 1 1 2 ");

break;

case 2:

displayIntList(myIntList, listLength);

break;

case 3:

displayGraph(xValues, yValues, numPoints);

break;

default:

displayText("default");

break;

}

return;

}

void displayText(const char\* text) {

Serial.println("displaytext");

display.fillScreen(GxEPD\_WHITE); // Clear the screen

display.setTextColor(GxEPD\_BLACK);

display.setFont(&FreeMonoBold9pt7b);

display.setCursor(20, 50);

display.print(text);

display.update();

delay(1000);

}

void displayIntList(int list[], int length) {

Serial.println("Displaying Integer List");

display.fillScreen(GxEPD\_WHITE); // Clear the screen

display.setTextColor(GxEPD\_BLACK);

display.setFont(&FreeMonoBold9pt7b);

// Set initial cursor position

int x = 20;

int y = 50;

// Display each element of the list

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

display.setCursor(x, y);

display.print(list[i]);

// Move the cursor to the next position

y += 20; // You can adjust the vertical spacing based on your preference

// Check if the next position goes beyond the display height

if (y > display.height() - 20) {

// If it does, reset y and move to the next column

y = 50;

x += 60; // You can adjust the horizontal spacing based on your preference

}

}

display.update();

delay(1000);

}

void displayGraph(int xValues[], int yValues[], int numPoints) {

Serial.println("Displaying Graph");

display.fillScreen(GxEPD\_WHITE); // Clear the screen

display.setTextColor(GxEPD\_BLACK);

display.setFont(&FreeMonoBold9pt7b);

// Define graph properties (currently hard coded)

int xStart = 20; // Starting X-coordinate of the graph

int yStart = 20; // Starting Y-coordinate of the graph

int graphWidth = 200; // Width of the graph

int graphHeight = 100; // Height of the graph

// Calculate the X and Y scale factors

float xScale = graphWidth / (float)(numPoints - 1);

float yScale = graphHeight / (float)(\*std::max\_element(yValues, yValues + numPoints) - \*std::min\_element(yValues, yValues + numPoints));

// Draw the X and Y axes

display.drawLine(xStart, yStart + graphHeight, xStart + graphWidth, yStart + graphHeight, GxEPD\_BLACK); // X-axis

display.drawLine(xStart, yStart, xStart, yStart + graphHeight, GxEPD\_BLACK); // Y-axis

// Draw the graph points and connecting lines

for (int i = 0; i < numPoints - 1; i++) {

int x1 = xStart + i \* xScale;

int y1 = yStart + graphHeight - ((yValues[i] - \*std::min\_element(yValues, yValues + numPoints)) \* yScale);

int x2 = xStart + (i + 1) \* xScale;

int y2 = yStart + graphHeight - ((yValues[i + 1] - \*std::min\_element(yValues, yValues + numPoints)) \* yScale);

// Draw connecting lines

display.drawLine(x1, y1, x2, y2, GxEPD\_BLACK);

// Draw circles at each data point

display.fillCircle(x1, y1, 2, GxEPD\_BLACK);

display.fillCircle(x2, y2, 2, GxEPD\_BLACK);

}

display.update();

delay(1000);

}

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

// GENERAL FUNCTIONS

void setup() {

// set up serial monitor

Serial.begin(9600);

Serial.print("hey girl hey");

// set up color sensor

tcs.begin();

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

if (tcs.begin()) {

Serial.println("Found sensor");

} else {

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

while (1);

}

// set up temperature sensor

sensors.begin();

// set up button

pinMode(BUTTON\_PIN, INPUT\_PULLUP);

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

// set up screen

display.init();

display.setRotation(1);

// decide on units of temperature

chooseFahrenheitCelsius();

// run first round of tests

block2();

}

void loop() {

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

// 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;

bool leftRight = leftRightFunc(currentPotValue);

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

// BUTTON STUFF

currentButtonState = digitalRead(BUTTON\_PIN);

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

Serial.println("Button pressed");

switchPages(leftRight);

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

}

lastButtonState = currentButtonState;