#include <Wire.h>

#include <RTClib.h>

#include <LiquidCrystal\_I2C.h>

#include <SPI.h>

#include <SD.h>

#include <EEPROM.h>

RTC\_DS3231 rtc;

LiquidCrystal\_I2C lcd(0x27, 16, 2);

// Pin definitions

const int soilMoisturePin = A0;

const int trigPin = 9;

const int echoPin = 10;

const int pumpRelayPin = 7;

const int buzzerPin = 6;

const int chipSelect = 4;

// Rotary Encoder

const int encoderCLK = 2;

const int encoderDT = 3;

const int encoderSW = 8;

// Scheduler and EEPROM

const int maxSchedules = 3;

int scheduleHours[maxSchedules] = {6, 12, 18};

int scheduleMinutes[maxSchedules] = {0, 0, 0};

bool scheduleAutoMode[maxSchedules] = {true, true, true};

// States and constants

int currentPage = 0;

const int maxPage = 3;

bool pumpRunning = false;

bool sdAvailable = false;

bool manualMode = false;

const int soilThreshold = 600;

const int waterFull = 10;

const int waterHalf = 20;

const int waterEmpty = 30;

unsigned long lastSensorRead = 0;

const unsigned long readInterval = 5000;

unsigned long lastPumpStart = 0;

const unsigned long pumpTimeout = 10000;

// Rotary debounce

volatile int lastCLK = HIGH;

unsigned long lastEncoderTurn = 0;

const unsigned long encoderDebounce = 200;

unsigned long lastButtonPress = 0;

const unsigned long buttonDebounce = 300;

int soilMoisture = 0;

long waterDistance = 0;

void setup() {

Serial.begin(9600);

Wire.begin();

lcd.init();

lcd.backlight();

lcd.setCursor(0, 0);

lcd.print(" SINAG SYSTEM");

lcd.setCursor(0, 1);

lcd.print(" Starting up...");

delay(2000);

lcd.clear();

if (!rtc.begin()) {

lcd.print("RTC not found!");

while (1);

}

if (SD.begin(chipSelect)) {

sdAvailable = true;

}

pinMode(pumpRelayPin, OUTPUT);

pinMode(buzzerPin, OUTPUT);

digitalWrite(pumpRelayPin, LOW);

digitalWrite(buzzerPin, LOW);

pinMode(encoderCLK, INPUT);

pinMode(encoderDT, INPUT);

pinMode(encoderSW, INPUT\_PULLUP);

pinMode(trigPin, OUTPUT);

pinMode(echoPin, INPUT);

loadScheduleFromEEPROM();

}

void loop() {

unsigned long currentMillis = millis();

handleEncoder();

handleButton();

if (currentMillis - lastSensorRead >= readInterval) {

lastSensorRead = currentMillis;

soilMoisture = analogRead(soilMoisturePin);

waterDistance = readUltrasonicDistance();

DateTime now = rtc.now();

if (sdAvailable) {

File dataFile = SD.open("log.txt", FILE\_WRITE);

if (dataFile) {

dataFile.print(now.timestamp());

dataFile.print(", Soil:");

dataFile.print(soilMoisture);

dataFile.print(", WaterDist:");

dataFile.println(waterDistance);

dataFile.close();

}

}

if (!manualMode) {

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

if (scheduleAutoMode[i] && now.hour() == scheduleHours[i] && now.minute() == scheduleMinutes[i] && soilMoisture > soilThreshold && waterDistance < waterEmpty) {

digitalWrite(pumpRelayPin, HIGH);

pumpRunning = true;

lastPumpStart = currentMillis;

}

}

}

}

if (pumpRunning && millis() - lastPumpStart >= pumpTimeout) {

digitalWrite(pumpRelayPin, LOW);

pumpRunning = false;

}

displayPage();

}

void handleEncoder() {

int clkState = digitalRead(encoderCLK);

if (clkState != lastCLK && clkState == LOW) {

if (millis() - lastEncoderTurn > encoderDebounce) {

int dtState = digitalRead(encoderDT);

if (dtState != clkState) {

currentPage++;

} else {

currentPage--;

}

if (currentPage < 0) currentPage = 0;

if (currentPage > maxPage) currentPage = maxPage;

lastEncoderTurn = millis();

}

}

lastCLK = clkState;

}

void handleButton() {

if (digitalRead(encoderSW) == LOW) {

if (millis() - lastButtonPress > buttonDebounce) {

manualMode = !manualMode;

lastButtonPress = millis();

}

}

}

void displayPage() {

DateTime now = rtc.now();

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("Mode: ");

lcd.print(manualMode ? "Manual" : "Auto");

if (currentPage == 0) {

lcd.setCursor(0, 1);

lcd.print("Time: ");

lcd.print(now.hour());

lcd.print(":");

if (now.minute() < 10) lcd.print("0");

lcd.print(now.minute());

} else if (currentPage == 1) {

lcd.setCursor(0, 1);

lcd.print("Soil: ");

lcd.print(soilMoisture);

} else if (currentPage == 2) {

lcd.setCursor(0, 1);

lcd.print("Water: ");

lcd.print(waterDistance);

lcd.print("cm ");

} else if (currentPage == 3) {

lcd.setCursor(0, 1);

lcd.print("Schedules:");

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

Serial.print("["); Serial.print(i); Serial.print("] ");

Serial.print(scheduleHours[i]); Serial.print(":"); Serial.print(scheduleMinutes[i]);

Serial.print(" Mode: "); Serial.println(scheduleAutoMode[i] ? "Auto" : "Man");

}

}

}

long readUltrasonicDistance() {

digitalWrite(trigPin, LOW);

delayMicroseconds(2);

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

long duration = pulseIn(echoPin, HIGH);

long distance = duration \* 0.034 / 2;

return distance;

}

void loadScheduleFromEEPROM() {

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

scheduleHours[i] = EEPROM.read(i \* 3);

scheduleMinutes[i] = EEPROM.read(i \* 3 + 1);

scheduleAutoMode[i] = EEPROM.read(i \* 3 + 2);

}

}

void saveScheduleToEEPROM() {

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

EEPROM.write(i \* 3, scheduleHours[i]);

EEPROM.write(i \* 3 + 1, scheduleMinutes[i]);

EEPROM.write(i \* 3 + 2, scheduleAutoMode[i]);

}

}