MODULE: CSYM028 Μάρτιος 2024

Εργασία: CSYM028 - Σύγχρονη Αρχιτεκτονική Υπολογιστών

Κωδικός Φοιτητή: 23837254

Part 1 ( Requirement 1)

#define RGB\_GREEN\_PIN 8

#define RGB\_BLUE\_PIN 9

#define RGB\_RED\_PIN 10

#define ON\_OFF\_BUTTON\_PIN 2

#define CONTINUOUS\_MODE\_BUTTON\_PIN 5

#define TREE1\_MODE\_CHANGE\_BUTTON\_PIN 6

#define TREE2\_MODE\_CHANGE\_BUTTON\_PIN 6

#define RGB\_MODE\_CHANGE\_BUTTON\_PIN 4

#define MOTION\_SENSOR\_PIN 12

#define LIGHT\_SENSOR\_PIN A0

#define DURATION\_POTENTIOMETER\_PIN A2

#define DAYLIGHT\_THRESHOLD\_PIN A3

#define ULTRASONIC\_TRIG\_PIN 11

#define ULTRASONIC\_ECHO\_PIN 13

#define NUM\_MODES 5 // only for rgb including off

void setup() {

pinMode(RGB\_GREEN\_PIN, OUTPUT);

pinMode(RGB\_BLUE\_PIN, OUTPUT);

pinMode(RGB\_RED\_PIN, OUTPUT);

pinMode(ON\_OFF\_BUTTON\_PIN, INPUT\_PULLUP);

pinMode(CONTINUOUS\_MODE\_BUTTON\_PIN, INPUT\_PULLUP);

pinMode(TREE1\_MODE\_CHANGE\_BUTTON\_PIN, INPUT\_PULLUP);

pinMode(TREE2\_MODE\_CHANGE\_BUTTON\_PIN, INPUT\_PULLUP);

pinMode(RGB\_MODE\_CHANGE\_BUTTON\_PIN, INPUT\_PULLUP);

pinMode(MOTION\_SENSOR\_PIN, INPUT);

pinMode(LIGHT\_SENSOR\_PIN, INPUT);

pinMode(DURATION\_POTENTIOMETER\_PIN, INPUT);

pinMode(DAYLIGHT\_THRESHOLD\_PIN, INPUT);

pinMode(TREE1\_LED\_PIN, OUTPUT);

pinMode(TREE2\_LED\_PIN, OUTPUT);

pinMode(ULTRASONIC\_TRIG\_PIN, OUTPUT);

pinMode(ULTRASONIC\_ECHO\_PIN, INPUT);

// Correctly placing the attachInterrupt call within the setup function

attachInterrupt(digitalPinToInterrupt(ON\_OFF\_BUTTON\_PIN), ISR\_toggleSystemState, CHANGE);

}

oid loop() {

static unsigned long lastDebounceTimeRGB = 0;

static int lastButtonStateRGB = HIGH;

static unsigned long lastMotionTime = 0;

unsigned long currentMillis = millis();

const long DISTANCE\_THRESHOLD = 100;

long distance = readUltrasonicDistance();

int currentButtonStateRGB = digitalRead(RGB\_MODE\_CHANGE\_BUTTON\_PIN);

int currentButtonStateContinuous = digitalRead(CONTINUOUS\_MODE\_BUTTON\_PIN);

static unsigned long lastDebounceTimeContinuous = 0;

static int lastButtonStateContinuous = HIGH;

// Read the light level and determine if it's night time

int lightLevel = readPotentiometer(LIGHT\_SENSOR\_PIN);

int daylightThreshold = readPotentiometer(DAYLIGHT\_THRESHOLD\_PIN);

// Read the potentiometer and map the value to the range of 5 to 20 seconds

int duration = readPotentiometer(DURATION\_POTENTIOMETER\_PIN);

long minDuration = map(duration, 0, 1023, 5000, 20000); // Converts to milliseconds (5-20 seconds)

handleModeChange();

if (systemOn && lightLevel <= daylightThreshold) {

if (continuousMode){

controlTreeLights(TREE1\_LED\_PIN, tree1Mode);

controlTreeLights(TREE2\_LED\_PIN, tree2Mode);

changeRGBMode(rgbMode); // Change the RGB mode

} else {

// Detect motion

if (digitalRead(MOTION\_SENSOR\_PIN) == HIGH && distance <= DISTANCE\_THRESHOLD) {

if (!motionDetected) { // If it's a new motion, update the lastMotionTime

}

}

// Keep the lights on if motion was detected and we are within the min duration

if (motionDetected) {

} else {

}

}

}

} else {

// Ensure lights are off during the day

digitalWrite(TREE1\_LED\_PIN, LOW);

digitalWrite(TREE2\_LED\_PIN, LOW);

setColor(0, 0, 0);

}

}

void toggleSystemState() {

systemOn = !systemOn;

if (!systemOn) {

digitalWrite(TREE1\_LED\_PIN, LOW);

digitalWrite(TREE2\_LED\_PIN, LOW);

setColor(0, 0, 0);

}

}

// ISR for toggling system state

void ISR\_toggleSystemState() {

static unsigned long lastInterruptTime = 0;

unsigned long interruptTime = millis();

// Debounce check

if (interruptTime - lastInterruptTime > 200) {

toggleSystemState(); // Toggle the system state

lastInterruptTime = interruptTime;

}

}

void handleModeChange() {

}

void controlTreeLights(int treePin, int mode) {

}

// Utility function to set the color of the RGB LED

void setColor(int red, int green, int blue) {

analogWrite(RGB\_RED\_PIN, red);

analogWrite(RGB\_GREEN\_PIN, green);

analogWrite(RGB\_BLUE\_PIN, blue);

}

Part 1 ( Optimised Requirement 2 – Final )

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Produced By: Modern Computer Architecture

-- URL: https://mypad.northampton.ac.uk/smartsc/

-- Author: Charalampos Nikolaidis

-- Date: {February-March}/2024

-- Purpose: Make an efficient light system

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

// ap, N. (n.d.). digitalWriteFast. GitHub. Available at: https://github.com/NicksonYap/digitalWriteFast [Accessed date: 06/03/2024].

// defined(\_\_AVR\_ATmega328\_\_) is defined in the library so arduino uno R3 will work with this.

#include <digitalWriteFast.h>

/\*

-- This library optimizes/replaces Arduino I/O operations via direct port manipulation, reducing execution time and resource usage.

-- The regular digitalWrite() in Arduino Uno core (16MHz) takes about 6280nS while digitalWriteFast() port manipulation takes 125nS.

-- It enhances energy efficiency minimizing CPU active cycles and RAM footprint.

-- Requires compile time constants for pin assignments that bypasses the Arduino layer resulting in reduction in exec times.

-- its tailored for AVR microcontrollers, compatible with our atmega328 chipset.

\*/

#define RGB\_GREEN\_PIN 9

#define RGB\_BLUE\_PIN 10

#define RGB\_RED\_PIN 11

#define ON\_OFF\_BUTTON\_PIN 2

#define CONTINUOUS\_MODE\_BUTTON\_PIN 5

#define TREE1\_MODE\_CHANGE\_BUTTON\_PIN 6

#define TREE2\_MODE\_CHANGE\_BUTTON\_PIN 6

#define RGB\_MODE\_CHANGE\_BUTTON\_PIN 4

#define MOTION\_SENSOR\_PIN 12

#define LIGHT\_SENSOR\_PIN A0

#define DURATION\_POTENTIOMETER\_PIN A2

#define DAYLIGHT\_THRESHOLD\_PIN A3

#define ULTRASONIC\_TRIG\_PIN 8

#define ULTRASONIC\_ECHO\_PIN 13

#define NUM\_MODES 5 // only for rgb including off

// LEDs for trees are connected to these pins through NMOS transistors

#define TREE1\_LED\_PIN 7

#define TREE2\_LED\_PIN 3

#define UTILITY\_DELAY 50 // milliseconds ( utility )

unsigned long lastDebounceTime1 = 0; // for TREE1\_MODE\_CHANGE\_BUTTON\_PIN

unsigned long lastDebounceTime2 = 0; // for TREE2\_MODE\_CHANGE\_BUTTON\_PIN

unsigned long lastDebounceTime3 = 0; // for RGB\_MODE\_CHANGE\_BUTTON\_PIN

volatile bool systemOn = false;

bool continuousMode = false;

int tree1Mode = 0; // Example modes: 0-normal, 1-breathing, 2-random etc

int tree2Mode = 0;

bool motionDetected = false;

bool initialActivationDone = false;

int rgbMode = 0;

int colorIndex = 0; // For jump mode

int gradualStep = 0; // For gradual and smooth mode

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Initializes pins, serial communication, and interrupts for system setup.

-- Called by: Arduino runtime on start.

-- Modifications: --

1. Added Pins as progressed.

2. Changed to pinModeFast.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void setup() {

// setupPWMRegistryManipulation(); commented out --> explanation at the end.

pinModeFast(RGB\_GREEN\_PIN, OUTPUT);

pinModeFast(RGB\_BLUE\_PIN, OUTPUT);

pinModeFast(RGB\_RED\_PIN, OUTPUT);

pinModeFast(ON\_OFF\_BUTTON\_PIN, INPUT\_PULLUP);

pinModeFast(CONTINUOUS\_MODE\_BUTTON\_PIN, INPUT\_PULLUP);

pinModeFast(TREE1\_MODE\_CHANGE\_BUTTON\_PIN, INPUT\_PULLUP);

pinModeFast(TREE2\_MODE\_CHANGE\_BUTTON\_PIN, INPUT\_PULLUP);

pinModeFast(RGB\_MODE\_CHANGE\_BUTTON\_PIN, INPUT\_PULLUP);

pinModeFast(MOTION\_SENSOR\_PIN, INPUT);

pinModeFast(LIGHT\_SENSOR\_PIN, INPUT);

pinModeFast(DURATION\_POTENTIOMETER\_PIN, INPUT);

pinModeFast(DAYLIGHT\_THRESHOLD\_PIN, INPUT);

pinModeFast(TREE1\_LED\_PIN, OUTPUT);

pinModeFast(TREE2\_LED\_PIN, OUTPUT);

pinModeFast(ULTRASONIC\_TRIG\_PIN, OUTPUT);

pinModeFast(ULTRASONIC\_ECHO\_PIN, INPUT);

// Correctly placing the attachInterrupt call within the setup function

// attachInterrupt(digitalPinToInterrupt(ON\_OFF\_BUTTON\_PIN), ISR\_toggleSystemState, CHANGE);

EICRA |= (1 << ISC00); // Set INT0 to trigger on any logic change

EIMSK |= (1 << INT0); // Enable INT0

Serial.begin(9600);

Serial.print("Free RAM: ");

Serial.println(freeRam());

}

int freeRam () {

extern int \_\_heap\_start, \*\_\_brkval;

int v;

return (int) &v - (\_\_brkval == 0 ? (int) &\_\_heap\_start : (int) \_\_brkval);

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Manages system logic, including light and mode control based on sensor input and button states.

-- Called by: Arduino runtime in main loop.

-- Modifications: --

1. Added inputs and sensors as i progressed

2. Implement non-blocking debounce(utility) logic to improve system responsiveness and reduce delay times.

3. Integrate a light intensity control feature, allowing adjustment based on ambient light conditions or user inputs.

4. Changed to digitalWriteFast.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void loop() {

static unsigned long lastDebounceTimeRGB = 0;

static int lastButtonStateRGB = HIGH;

static unsigned long lastMotionTime = 0;

unsigned long currentMillis = millis();

const long DISTANCE\_THRESHOLD = 100;

long distance = readUltrasonicDistance();

int currentButtonStateRGB = digitalReadFast(RGB\_MODE\_CHANGE\_BUTTON\_PIN);

int currentButtonStateContinuous = digitalReadFast(CONTINUOUS\_MODE\_BUTTON\_PIN);

static unsigned long lastDebounceTimeContinuous = 0;

static int lastButtonStateContinuous = HIGH;

// Read the light level and determine if it's night time

int lightLevel = readPotentiometer(LIGHT\_SENSOR\_PIN);

int daylightThreshold = readPotentiometer(DAYLIGHT\_THRESHOLD\_PIN);

// Read the potentiometer and map the value to the range of 3 to 20 seconds

int duration = readPotentiometer(DURATION\_POTENTIOMETER\_PIN);

long minDuration = map(duration, 0, 1023, 3000, 20000); // Converts to milliseconds (3-20 seconds)

handleModeChange();

// Debounce the button press for the continuous mode toggle

if (currentMillis - lastDebounceTimeContinuous > UTILITY\_DELAY) {

if (currentButtonStateContinuous != lastButtonStateContinuous) {

lastDebounceTimeContinuous = currentMillis; // Reset debounce timer

if (currentButtonStateContinuous == LOW) { // Button pressed

continuousMode = !continuousMode; // Toggle continuous mode

}

}

}

lastButtonStateContinuous = currentButtonStateContinuous; // Update button state for next loop iteration

if (systemOn && lightLevel <= daylightThreshold) {

if (continuousMode){

controlTreeLights(TREE1\_LED\_PIN, tree1Mode);

controlTreeLights(TREE2\_LED\_PIN, tree2Mode);

changeRGBMode(rgbMode); // Change the RGB mode

} else {

// Detect motion

if (digitalReadFast(MOTION\_SENSOR\_PIN) == HIGH && distance <= DISTANCE\_THRESHOLD) {

if (!motionDetected) { // If it's a new motion, update the lastMotionTime

lastMotionTime = currentMillis;

motionDetected = true;

}

}

// Keep the lights on if motion was detected and we are within the min duration

if (motionDetected) {

if (currentMillis - lastMotionTime <= minDuration) {

// Only control lights if we're within the min duration period

controlTreeLights(TREE1\_LED\_PIN, tree1Mode);

controlTreeLights(TREE2\_LED\_PIN, tree2Mode);

changeRGBMode(rgbMode); // Change the RGB mode

} else {

// Once min duration has elapsed, turn off the lights and reset motionDetected flag

digitalWriteFast(TREE1\_LED\_PIN, LOW);

digitalWriteFast(TREE2\_LED\_PIN, LOW);

setColor(0, 0, 0);

motionDetected = false; // Reset motion detected flag for the next motion event

}

}

}

} else {

// Ensure lights are off during the day ( when light sensor is above threshold )

digitalWriteFast(TREE1\_LED\_PIN, LOW);

digitalWriteFast(TREE2\_LED\_PIN, LOW);

setColor(0, 0, 0);

}

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Changes the global state of the system, turning off all controlled outputs when deactivated.

-- Called by: ISR(INT0\_vect)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void toggleSystemState() {

systemOn = !systemOn;

if (!systemOn) {

digitalWriteFast(TREE1\_LED\_PIN, LOW);

digitalWriteFast(TREE2\_LED\_PIN, LOW);

setColor(0, 0, 0);

}

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Handles debouncing of the system state toggle switch and calls toggleSystemState when a valid state change is detected.

-- Called by: setup() as interrupt

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

// ISR for toggling system state

ISR(INT0\_vect) {

static unsigned long lastInterruptTime = 0;

unsigned long interruptTime = millis();

// Debounce check

if (interruptTime - lastInterruptTime > 200) {

toggleSystemState(); // Toggle the system state

lastInterruptTime = interruptTime;

}

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Detects button presses to cycle through different lighting modes for both tree lights and the RGB LED.

-- Called by: loop()

-- Modifications: --

1. Tried to implement 2 modes for each tree. Stayed with the single logic.

2. Changed to digitalWriteFast.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void handleModeChange() {

static int lastButtonState1 = HIGH;

static int lastButtonState2 = HIGH;

static int lastButtonState3 = HIGH;

int currentButtonState1 = digitalReadFast(TREE1\_MODE\_CHANGE\_BUTTON\_PIN);

int currentButtonState2 = digitalReadFast(TREE2\_MODE\_CHANGE\_BUTTON\_PIN);

int currentButtonState3 = digitalReadFast(RGB\_MODE\_CHANGE\_BUTTON\_PIN);

// Check for tree 1 mode change button

if (currentButtonState1 != lastButtonState1) {

if (millis() - lastDebounceTime1 > UTILITY\_DELAY) {

if (currentButtonState1 == LOW) { // Button press confirmed

tree1Mode = (tree1Mode + 1) % 5; // Cycle through the modes

lastDebounceTime1 = millis(); // Reset the debounce timer

}

}

lastButtonState1 = currentButtonState1; // Update the last button state

}

if (currentButtonState2 != lastButtonState2) {

if (millis() - lastDebounceTime2 > UTILITY\_DELAY) {

if (currentButtonState2 == LOW) { // Button press confirmed

tree2Mode = (tree2Mode + 1) % 5; // Cycle through the modes

lastDebounceTime2 = millis(); // Reset the debounce timer

}

}

lastButtonState2 = currentButtonState2; // Update the last button state

}

if (currentButtonState3 != lastButtonState3) {

if (millis() - lastDebounceTime3 > UTILITY\_DELAY) {

if (currentButtonState3 == LOW) { // Button press confirmed

rgbMode = (rgbMode + 1) % 5; // Cycle through the modes up to 5

lastDebounceTime3 = millis(); // Reset the debounce timer

}

}

lastButtonState3 = currentButtonState3; // Update the last button state

}

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Controls tree lighting effects based on the specified mode, including blinking, breathing, stroboscopic, gradual on/off, and smooth transitions.

-- Called by: loop(), when managing light behaviors based on system state and mode settings.

-- Modifications: --

Starting logic was originally completely different. I did not use switch case, i wanted to have different functions, instead kept the single function multiple modes logic.

Stroboscopic changed dozen times.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void controlTreeLights(int treePin, int mode) {

switch (mode) {

case 0: // Faster Simple blinking

digitalWriteFast(treePin, HIGH); // Turn on the light

delay(100); // On for 100 milliseconds

digitalWriteFast(treePin, LOW); // Turn off the light

delay(100); // Off for 100 milliseconds

break;

case 1: // Faster Breathing pattern

for (int i = 0; i <= 255; i += 5) { // Increase brightness more quickly

analogWrite(treePin, i);

delay(5); // Reduce delay for faster transition ( utility )

}

for (int i = 255; i >= 0; i -= 5) { // Decrease brightness more quickly

analogWrite(treePin, i);

delay(5); // Reduce delay for faster transition ( utility )

}

break;

case 2: // Faster Stroboscopic effect

for (int i = 0; i < 10; i++) { // Flash the light more times

digitalWriteFast(treePin, HIGH);

delay(10); // On for 10 milliseconds ( utility )

digitalWriteFast(treePin, LOW);

delay(10); // Off for 10 milliseconds ( utility )

}

break;

case 3: // Faster Gradual on and off

for (int i = 0; i <= 255; i += 10) { // Increase increment for quicker transition

analogWrite(treePin, i);

delay(15); // Delay for transition

}

for (int i = 255; i >= 0; i -= 10) { // Decrease increment for quicker transition

analogWrite(treePin, i);

delay(5); // Quicker off transition

}

break;

case 4: // Faster Smooth transition through colors

for (int color = 0; color <= 255; color += 5) { // Faster color change

analogWrite(treePin, color);

delay(5); // Reduced delay for smoother, faster change

}

break;

default:

digitalWriteFast(treePin, LOW); // Off by default

break;

}

}

int readPotentiometer(int pin) {

return analogRead(pin); // Reads the potentiometer value directly

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Measures the distance to an object using ultrasonic waves, by sending a pulse and calculating the time taken for the echo to return.

-- Called by: loop(), for determining if an object is within a specific range to trigger motion-based lighting.

-- Modifications: None -> Got it from here https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

long readUltrasonicDistance() {

digitalWriteFast(ULTRASONIC\_TRIG\_PIN, LOW);

delayMicroseconds(2);

digitalWriteFast(ULTRASONIC\_TRIG\_PIN, HIGH);

delayMicroseconds(10);

digitalWriteFast(ULTRASONIC\_TRIG\_PIN, LOW);

long duration = pulseIn(ULTRASONIC\_ECHO\_PIN, HIGH);

long distance = duration \* 0.034 / 2; // Speed of sound wave divided by 2 (go and return)

return distance;

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Adjusts the RGB LED's mode and executes the corresponding visual effect based on the current mode selection.

-- Called by: loop(), when managing RGB LED behaviors based on selected mode.

-- Modifications: --

Just like the tree leds, the i was following the same idea here. Starting with single function for each of the modes, ending up in a single function with switch case.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void changeRGBMode(int mode) {

// Static variables to maintain state and timing across function calls

static unsigned long previousMillis = 0;

static int colorIndex = 0; // Index for color array in jump mode

static int gradualStep = 0; // Step for gradual and smooth mode transitions

static int rgbMode = 0; // Static to maintain the mode state across calls

// Constants for timing intervals

const long intervalJump = 1000; // Interval at which to jump colors (milliseconds)

const long intervalStrobe = 250; // Interval for strobe effect (milliseconds)

const long intervalGradual = 20; // Speed of gradual change

const long intervalSmooth = 10; // Speed of smooth change

// Update the mode only if a new mode is passed in

if (mode != rgbMode) {

rgbMode = mode;

// Reset state for new mode

previousMillis = millis();

colorIndex = 0;

gradualStep = 0;

}

unsigned long currentMillis = millis(); // Get current time

// Logic to handle each mode

switch (rgbMode) {

case 1: { // Jump

if (currentMillis - previousMillis >= intervalJump) {

previousMillis = currentMillis;

int colors[][3] = {{255, 0, 0}, {0, 255, 0}, {0, 0, 255}, {255, 255, 0}, {0, 255, 255}, {255, 0, 255}, {255, 255, 255}};

setColor(colors[colorIndex][0], colors[colorIndex][1], colors[colorIndex][2]);

colorIndex = (colorIndex + 1) % 7; // Cycle through colors

}

break;

}

case 2: { // Stroboscopic

bool isOn = (currentMillis / intervalStrobe) % 2 == 0;

setColor(isOn ? 255 : 0, isOn ? 255 : 0, isOn ? 255 : 0); // Only white strobe

if (currentMillis - previousMillis >= intervalStrobe) {

previousMillis = currentMillis; // timings for flashing

}

break;

}

case 3: { // Gradual

if (currentMillis - previousMillis >= intervalGradual) {

previousMillis = currentMillis;

gradualStep = (gradualStep + 1) % 360; // Increment step for a continuous cycle

// Calculating RGB values based on sine wave for smooth transition

int r = (sin(gradualStep \* (PI / 180) \* 0.5) \* 127.5) + 127.5; // Slow down the cycle by multiplying step by 0.5

int g = (sin((gradualStep \* (PI / 180) \* 0.5) + 2 \* PI / 3) \* 127.5) + 127.5;

int b = (sin((gradualStep \* (PI / 180) \* 0.5) + 4 \* PI / 3) \* 127.5) + 127.5;

setColor(r, g, b);

}

break;

}

case 4: { // Smooth

if (currentMillis - previousMillis >= intervalSmooth) {

previousMillis = currentMillis;

gradualStep = (gradualStep + 1) % 360; // Increment and wrap the step for continuous cycles

// slower transition between colors by adjusting the sine wave frequency

float frequency = 0.5;

int r = (sin(gradualStep \* (PI / 180) \* frequency) \* 127.5) + 127.5;

int g = (sin((gradualStep \* (PI / 180) \* frequency + 2 \* PI / 3)) \* 127.5) + 127.5;

int b = (sin((gradualStep \* (PI / 180) \* frequency + 4 \* PI / 3)) \* 127.5) + 127.5;

setColor(r, g, b);

}

break;

}

default: {

setColor(0, 0, 0); // Off

break;

}

}

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Sets the RGB LED color by adjusting the intensity of its red, green, and blue components.

-- Called by: changeRGBMode(), controlTreeLights() (if modified to include RGB control), and any other function requiring direct manipulation of the RGB LED color.

-- Modifications: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void setColor(int red, int green, int blue) { // Utility function .. sets the color of the RGB LED

analogWrite(RGB\_RED\_PIN, red);

analogWrite(RGB\_GREEN\_PIN, green);

analogWrite(RGB\_BLUE\_PIN, blue);

}

/\*

------------------------------------------------------------------

Issues with: Timings / timing missteps.

analogWrite is a pre written well composed song. But with direct register manipulation i had to compose the 'music' (light effects) myself.

--------------------------------- ---------------------------------

\*/

// void setColorRegistryManipulation(int red, int green, int blue) {

// // Green on pin 9 (OC1A), Red on pin 11 (OC2A), and Blue on pin 10 (OC1B)

// OCR1A = green; // Green pin 9

// OCR2A = red; // Red pin 11

// OCR1B = blue; // Blue pin 10

// }

// void setupPWMRegistryManipulation() {

// // pins 9, 10, 11 outputs for PWM

// DDRB |= (1 << DDB1) | (1 << DDB2); // Pin 9 (PB1/OC1A), Pin 11 (PB3/OC2A) output

// DDRB |= (1 << DDB3); // Pin 10 (PB2/OC1B) output

// // Configure Timer1 for Fast PWM mode for pins 9 and 11

// TCCR1A = (1 << WGM11) | (1 << COM1A1) | (1 << COM1B1);

// TCCR1B = (1 << WGM12) | (1 << WGM13) | (1 << CS10); // No prescaler, fast PWM

// ICR1 = 0x00FF; // 8-bit resolution just as analogWrite()

// TCCR2A = (1 << WGM21) | (1 << WGM20) | (1 << COM2A1); // Fast PWM

// TCCR2B = (1 << CS20); // No prescaler

// }

**Μέρος 1: Φωτιστικά Διακόσμησης Ενεργειακά Αποδοτικά (45%)**

**VIDEO:**

[**https://nikolaidis.info/csym028/part\_1/**](https://nikolaidis.info/csym028/part_1/)

Σύνδεσμοι προς τα σχέδια στο Tinkercad (δύο λύσεις):

Fanari With Aduino Layer:

<https://www.tinkercad.com/things/8PpLsGOZMlD-lights>

Fanari Final:

<https://www.tinkercad.com/things/7HpKpcbCD2x-lights-final>

PART 2: This is only an attempt with serial manipuilations. This is only presented to show one more approach of mine for the requirement 2. Final is below this solution.  
REGISTER MANIPULATION IDEA ( not compilable )   
Thjs was an attempt to use direct port manipulation. I present it as a first approach to requirement 2. I also tried using 4 libraries that are generally used for energy efficiency.  
In this attempt I also tried implementing the lcds without the LiquidCrystal\_I2C library to provide more data on how it has impacted SRAM.

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include <stdio.h>

void serialBegin() {

uint16\_t baud\_rate = 103; // UBRR value for 9600 baud with 16MHz clock

UBRR0H = (baud\_rate >> 8);

UBRR0L = baud\_rate;

UCSR0B = (1 << RXEN0) | (1 << TXEN0); // Enable receiver and transmitter

UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); // Set frame: 8data, 1 stop bit

}

enum TrafficLightState {

RED,

RED\_YELLOW,

GREEN,

YELLOW

};

/\*

\* Due to a known issue with older Arduino IDE versions (in 1.7.x series from Arduino.org),

\* the compiler incorrectly places autogenerated function prototypes before custom type definitions,

\* causing build errors when enums are used as function arguments. This issue is discussed in detail

\* on the Arduino Stack Exchange (Majenko, 2016). The workaround involves manually declaring

\* function prototypes and custom types (enums, in this case) before their first use in the code.

\* This approach ensures compatibility across different versions of the Arduino IDE, including

\* Tinkercad’s Arduino simulation environment, which may use an older build system similar to Arduino 1.7.x.

\* <https://arduino>.stackexchange.com/questions/28133/cant-use-enum-as-function-argument

\*/

void setTrafficLightState(TrafficLightState state);

void updateTrafficLightState(unsigned long currentMillis);

void checkMotionSensor();

void setPedestrianRGB(bool isRed);

void updateAudibleVisualCues(unsigned long currentMillis);

void updateLCDStatus(unsigned long currentMillis);

String getStateText(TrafficLightState state);

unsigned long getRemainingTime(unsigned long currentMillis);

void handlePedestrianButtonPress();

void processPedestrianRequest(unsigned long currentMillis);

volatile bool pedestrianRequest = false;

volatile unsigned long buttonPressTimestamp = 0;

bool motionDetected = false;

unsigned long lastDisplayedTime = 0;

bool isStaticTextInitialized = false;

TrafficLightState trafficLightState = GREEN;

unsigned long stateChangeTimestamp = 0;

unsigned long previousMillis = 0; // Stores the last time the display was updated

const long interval = 300; // Interval at which to refresh the display (300ms)

int position = 0; // Current position of the scrolling text

bool scrollingForward = true; // Direction of scroll

LiquidCrystal\_I2C lcd\_0x20(LCD\_ADDRESS\_0x20, 16, 2);

LiquidCrystal\_I2C lcd\_0x26(LCD\_ADDRESS\_0x26, 16, 2);

void setup() {

// Set BUTTON\_PIN and MOTION\_SENSOR\_PIN as input, enable pull-up for BUTTON\_PIN

DDRD &= ~((1 << DDD2) | (1 << DDD3));

PORTD |= (1 << PORTD2);

// Set LED and BUZZER pins as output

DDRD |= (1 << DDD4) | (1 << DDD5) | (1 << DDD6) | (1 << DDD7); // RED\_LIGHT\_CAR\_PIN to RED\_LIGHT\_PED\_PIN

DDRB |= (1 << DDB1) | (1 << DDB2) | (1 << DDB3); // BUZZER\_PIN, RGB\_RED\_PIN, RGB\_GREEN\_PIN

// External Interrupt Setup for BUTTON\_PIN (INT0)

EICRA |= (1 << ISC01); // Set INT0 to trigger on falling edge

EIMSK |= (1 << INT0); // Enable INT0

// Initialize Serial Communication

serialBegin();

// LCD and backlight initialization code would remain similar, utilizing the LiquidCrystal\_I2C library

// Set initial traffic light state

setTrafficLightState(GREEN);

// Enable global interrupts

sei();

}

// ISR for BUTTON\_PIN press

ISR(INT0\_vect) {

// Handle pedestrian button press logic

}

// Placeholder for the setTrafficLightState function to be adjusted for direct register manipulation

void setTrafficLightState(/\* TrafficLightState state \*/) {

// Implementation would directly manipulate PORTx registers

}

int freeRam () {

extern int \_\_heap\_start, \*\_\_brkval;

int v;

return (int) &v – (\_\_brkval == 0 ? (int) &\_\_heap\_start : (int) \_\_brkval);

}

void loop() {

unsigned long currentMillis = millis();

checkMotionSensor();

if (pedestrianRequest) {

processPedestrianRequest(currentMillis);

}

updateTrafficLightState(currentMillis);

updateAudibleVisualCues(currentMillis);

updateLCDStatus(currentMillis);

Serial.print(“Free RAM: “);

Serial.println(freeRam());

}

void handlePedestrianButtonPress() {

pedestrianRequest = true;

buttonPressTimestamp = millis();

}

void processPedestrianRequest(unsigned long currentMillis) {

if (trafficLightState == GREEN && pedestrianRequest) {

if ((!motionDetected) || (motionDetected && currentMillis – buttonPressTimestamp >= MOTION\_DETECTED\_DELAY)) {

setTrafficLightState(YELLOW);

buttonPressTimestamp = currentMillis;

pedestrianRequest = false; // Reset request to prevent re-entry

}

}

}

void setTrafficLightState(TrafficLightState state) {

trafficLightState = state;

stateChangeTimestamp = millis();

digitalWrite(RED\_LIGHT\_CAR\_PIN, state == RED || state == RED\_YELLOW ? HIGH : LOW);

digitalWrite(YELLOW\_LIGHT\_CAR\_PIN, state == YELLOW || state == RED\_YELLOW ? HIGH : LOW);

digitalWrite(GREEN\_LIGHT\_CAR\_PIN, state == GREEN ? HIGH : LOW);

// Manage pedestrian light directly within state transitions if needed

if(state == RED) {

delay(500); // Delay for 0.5 seconds

setPedestrianRGB(false); // Turn pedestrian light green

} else {

setPedestrianRGB(true); // Otherwise, keep pedestrian light red

}

// Specific handling when transitioning to GREEN to clear the timer display

if (state == GREEN) {

lcd\_0x20.setCursor(6, 1); // Adjust cursor to the position of the timer

lcd\_0x20.print(“ “); // Clear the timer display area

}

}

void updateTrafficLightState(unsigned long currentMillis) {

// Extended logic to transition from YELLOW to RED for vehicles and then to make pedestrian light GREEN

switch (trafficLightState) {

case RED:

updateCarDisplay(false);

// After sufficient RED time, transition to RED\_YELLOW before going to GREEN for cars (and RED for pedestrians)

if (currentMillis – stateChangeTimestamp >= RED\_TO\_GREEN\_DELAY) {

setTrafficLightState(RED\_YELLOW);

}

break;

case RED\_YELLOW:

// Transition to GREEN for cars, which implies RED for pedestrians

if (currentMillis – stateChangeTimestamp >= YELLOW\_TO\_RED\_DELAY) {

setTrafficLightState(GREEN);

}

break;

case GREEN:

updateCarDisplay(true);

break;

case YELLOW:

// Transition from YELLOW to RED for cars, which will then trigger GREEN for pedestrians

if (currentMillis – stateChangeTimestamp >= YELLOW\_TO\_RED\_DELAY) {

setTrafficLightState(RED);

}

break;

}

}

void checkMotionSensor() {

motionDetected = digitalRead(MOTION\_SENSOR\_PIN) == HIGH;

}

void setPedestrianRGB(bool isRed) {

digitalWrite(RGB\_RED\_PIN, isRed ? HIGH : LOW);

digitalWrite(RGB\_GREEN\_PIN, !isRed ? HIGH : LOW);

}

void updateAudibleVisualCues(unsigned long currentMillis) {

if (trafficLightState == RED || trafficLightState == RED\_YELLOW) {

tone(BUZZER\_PIN, 1000);

} else {

noTone(BUZZER\_PIN);

}

}

void updateLCDStatus(unsigned long currentMillis) {

String stateText = getStateText(trafficLightState);

unsigned long remainingTime = getRemainingTime(currentMillis) / 1000; // Convert to seconds

if (!isStaticTextInitialized) {

lcd\_0x20.clear();

lcd\_0x20.print(“State: “ + stateText);

isStaticTextInitialized = true;

}

// Only update the timer for states other than GREEN without a pedestrian request

if (trafficLightState != GREEN || pedestrianRequest) {

if (lastDisplayedTime != remainingTime) {

lcd\_0x20.setCursor(6, 1); // Adjust based on your layout

if (remainingTime < 10) { // Padding for single digit numbers

lcd\_0x20.print(“0”); // Padding for consistency

}

lcd\_0x20.print(String(remainingTime + 1) + “s “); // Update time

lastDisplayedTime = remainingTime;

}

} else {

// If the state is GREEN and no pedestrian request, clear the timer display area

if (lastDisplayedTime != 0) {

lcd\_0x20.setCursor(6, 1); // Adjust cursor to timer position

lcd\_0x20.print(“ “); // Clear the timer display area

lastDisplayedTime = 0; // Reset last displayed time

}

}

}

String getStateText(TrafficLightState state) {

switch (state) {

case RED: return “Red”;

case RED\_YELLOW: return “Red+Yellow”;

case GREEN: return “Green”;

case YELLOW: return “Yellow”;

default: return “”;

}

}

unsigned long getRemainingTime(unsigned long currentMillis) {

switch (trafficLightState) {

case RED:

case GREEN:

return PEDESTRIAN\_CROSSING\_TIME – (currentMillis – stateChangeTimestamp);

case RED\_YELLOW:

case YELLOW:

return YELLOW\_TO\_RED\_DELAY – (currentMillis – stateChangeTimestamp);

default:

return 0;

}

}

void updateCarDisplay(bool isGreen) {

unsigned long currentMillis = millis();

if (currentMillis – previousMillis >= interval) {

previousMillis = currentMillis;

// Define messages as char arrays

const char greenMessage[] = “Commercial Green Car”;

const char redMessage[] = “Commercial Red Car”;

const char\* message = isGreen ? greenMessage : redMessage;

int messageLength = strlen(message); // Use strlen for char arrays

int displayWidth = 16;

int scrollRange = max(0, messageLength – displayWidth);

lcd\_0x26.clear();

char displayBuffer[17]; // Plus one for null terminator

memset(displayBuffer, ‘ ‘, sizeof(displayBuffer)); // Fill with spaces

displayBuffer[16] = ‘\0’; // Ensure null-termination

// Calculate start index based on current position and direction

int startIndex = scrollingForward ? position : scrollRange – position;

startIndex = max(0, min(startIndex, scrollRange));

// Copy relevant part of message to displayBuffer

for (int I = 0; I < displayWidth && (startIndex + i) < messageLength; i++) {

displayBuffer[i] = message[startIndex + i];

}

// Print the displayBuffer

lcd\_0x26.print(displayBuffer);

// Update position and change direction if needed

if (scrollingForward && position >= scrollRange) {

scrollingForward = false; // Change direction to scroll back

} else if (!scrollingForward && position <= 0) {

scrollingForward = true; // Change direction to scroll forward

}

// Update position for the next iteration

position += scrollingForward ? 1 : -1;

}

}

// #include <avr/pgmspace.h>

// Global scope for PROGMEM strings

// const char greenCarMsg[] PROGMEM = “Commercial Green Car”;

// const char redCarMsg[] PROGMEM = “Commercial Red Car”;

// void displayStaticEfficientFromProgmem() {

// char buffer[25]; // Buffer for copying the message from PROGMEM

// strcpy\_P(buffer, (isGreen ? greenCarMsg : redCarMsg)); // You need to pass isGreen to this function or decide which message to use before calling

// buffer[24] = ‘\0’; // Explicitly null-terminate

// lcd\_0x26.clear();

// lcd\_0x26.print(buffer); // Display the static message

// }

// void updateCarDisplay(bool isGreen) {

// unsigned long currentMillis = millis();

// if (currentMillis – previousMillis >= interval) {

// previousMillis = currentMillis;

// displayStaticTestFromProgmem(); // Adjust this function to either accept isGreen or decide the message before calling

// }

// }

PART 2: Requirement 1

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

#define BUTTON\_PIN 2

#define MOTION\_SENSOR\_PIN 3

#define RED\_LIGHT\_CAR\_PIN 4

#define YELLOW\_LIGHT\_CAR\_PIN 5

#define GREEN\_LIGHT\_CAR\_PIN 6

#define RED\_LIGHT\_PED\_PIN 7

#define BUZZER\_PIN 9

#define RGB\_RED\_PIN 10

#define RGB\_GREEN\_PIN 11

#define LCD\_ADDRESS\_0x20 0x20

#define LCD\_ADDRESS\_0x26 0x26

#define YELLOW\_TO\_RED\_DELAY 2500

#define RED\_TO\_GREEN\_DELAY 3000

#define PEDESTRIAN\_CROSSING\_TIME 8000

#define SOUND\_SPEEDUP\_TIME 5000

#define MOTION\_DETECTED\_DELAY 5000

enum TrafficLightState {

RED,

RED\_YELLOW,

GREEN,

YELLOW

};

/\*

\* Due to a known issue with older Arduino IDE versions (in 1.7.x series from Arduino.org),

\* the compiler incorrectly places autogenerated function prototypes before custom type definitions,

\* causing build errors when enums are used as function arguments. This issue is discussed in detail

\* on the Arduino Stack Exchange (Majenko, 2016). The workaround involves manually declaring

\* function prototypes and custom types (enums, in this case) before their first use in the code.

\* This approach ensures compatibility across different versions of the Arduino IDE, including

\* Tinkercad's Arduino simulation environment, which may use an older build system similar to Arduino 1.7.x.

\* https://arduino.stackexchange.com/questions/28133/cant-use-enum-as-function-argument

\*/

void setTrafficLightState(TrafficLightState state);

void updateTrafficLightState(unsigned long currentMillis);

void checkMotionSensor();

void setPedestrianRGB(bool isRed);

void updateAudibleVisualCues(unsigned long currentMillis);

void updateLCDStatus(unsigned long currentMillis);

String getStateText(TrafficLightState state);

unsigned long getRemainingTime(unsigned long currentMillis);

void handlePedestrianButtonPress();

void processPedestrianRequest(unsigned long currentMillis);

volatile bool pedestrianRequest = false;

volatile unsigned long buttonPressTimestamp = 0;

bool motionDetected = false;

unsigned long lastDisplayedTime = 0;

bool isStaticTextInitialized = false;

TrafficLightState trafficLightState = GREEN;

unsigned long stateChangeTimestamp = 0;

unsigned long previousMillis = 0; // Stores the last time the display was updated

const long interval = 300; // Interval at which to refresh the display (300ms)

int position = 0; // Current position of the scrolling text

bool scrollingForward = true; // Direction of scroll

LiquidCrystal\_I2C lcd\_0x20(LCD\_ADDRESS\_0x20, 16, 2);

LiquidCrystal\_I2C lcd\_0x26(LCD\_ADDRESS\_0x26, 16, 2);

void setup() {

pinMode(BUTTON\_PIN, INPUT\_PULLUP);

pinMode(MOTION\_SENSOR\_PIN, INPUT);

pinMode(RED\_LIGHT\_CAR\_PIN, OUTPUT);

pinMode(YELLOW\_LIGHT\_CAR\_PIN, OUTPUT);

pinMode(GREEN\_LIGHT\_CAR\_PIN, OUTPUT);

pinMode(RED\_LIGHT\_PED\_PIN, OUTPUT);

pinMode(BUZZER\_PIN, OUTPUT);

pinMode(RGB\_RED\_PIN, OUTPUT);

pinMode(RGB\_GREEN\_PIN, OUTPUT);

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

lcd\_0x20.init();

lcd\_0x20.backlight();

lcd\_0x26.init();

lcd\_0x26.backlight();

setTrafficLightState(GREEN);

Serial.begin(9600);

Serial.print("Free RAM: "); // in loop -4 bytes of ram

Serial.println(freeRam());

}

int freeRam () {

extern int \_\_heap\_start, \*\_\_brkval;

int v;

return (int) &v - (\_\_brkval == 0 ? (int) &\_\_heap\_start : (int) \_\_brkval);

}

void loop() {

unsigned long currentMillis = millis();

checkMotionSensor();

if (pedestrianRequest) {

processPedestrianRequest(currentMillis);

}

updateTrafficLightState(currentMillis);

updateAudibleVisualCues(currentMillis);

updateLCDStatus(currentMillis);

}

void handlePedestrianButtonPress() {

pedestrianRequest = true;

buttonPressTimestamp = millis();

}

void processPedestrianRequest(unsigned long currentMillis) {

if (trafficLightState == GREEN && pedestrianRequest) {

if ((!motionDetected) || (motionDetected && currentMillis - buttonPressTimestamp >= MOTION\_DETECTED\_DELAY)) {

setTrafficLightState(YELLOW);

buttonPressTimestamp = currentMillis;

pedestrianRequest = false; // Reset request to prevent re-entry

}

}

}

void setTrafficLightState(TrafficLightState state) {

trafficLightState = state;

stateChangeTimestamp = millis();

digitalWrite(RED\_LIGHT\_CAR\_PIN, state == RED || state == RED\_YELLOW ? HIGH : LOW);

digitalWrite(YELLOW\_LIGHT\_CAR\_PIN, state == YELLOW || state == RED\_YELLOW ? HIGH : LOW);

digitalWrite(GREEN\_LIGHT\_CAR\_PIN, state == GREEN ? HIGH : LOW);

// Manage pedestrian light directly within state transitions if needed

if(state == RED) {

delay(500); // Delay for 0.5 seconds

setPedestrianRGB(false); // Turn pedestrian light green

} else {

setPedestrianRGB(true); // Otherwise, keep pedestrian light red

}

// Specific handling when transitioning to GREEN to clear the timer display

if (state == GREEN) {

lcd\_0x20.setCursor(6, 1); // Adjust cursor to the position of the timer

lcd\_0x20.print(" "); // Clear the timer display area

}

}

void updateTrafficLightState(unsigned long currentMillis) {

// Extended logic to transition from YELLOW to RED for vehicles and then to make pedestrian light GREEN

switch (trafficLightState) {

case RED:

updateCarDisplay(false);

// After sufficient RED time, transition to RED\_YELLOW before going to GREEN for cars (and RED for pedestrians)

if (currentMillis - stateChangeTimestamp >= RED\_TO\_GREEN\_DELAY) {

setTrafficLightState(RED\_YELLOW);

}

break;

case RED\_YELLOW:

// Transition to GREEN for cars, which implies RED for pedestrians

if (currentMillis - stateChangeTimestamp >= YELLOW\_TO\_RED\_DELAY) {

setTrafficLightState(GREEN);

}

break;

case GREEN:

updateCarDisplay(true);

break;

case YELLOW:

// Transition from YELLOW to RED for cars, which will then trigger GREEN for pedestrians

if (currentMillis - stateChangeTimestamp >= YELLOW\_TO\_RED\_DELAY) {

setTrafficLightState(RED);

}

break;

}

}

void checkMotionSensor() {

motionDetected = digitalRead(MOTION\_SENSOR\_PIN) == HIGH;

}

void setPedestrianRGB(bool isRed) {

digitalWrite(RGB\_RED\_PIN, isRed ? HIGH : LOW);

digitalWrite(RGB\_GREEN\_PIN, !isRed ? HIGH : LOW);

}

void updateAudibleVisualCues(unsigned long currentMillis) {

if (trafficLightState == RED || trafficLightState == RED\_YELLOW) {

tone(BUZZER\_PIN, 1000);

} else {

noTone(BUZZER\_PIN);

}

}

void updateLCDStatus(unsigned long currentMillis) {

String stateText = getStateText(trafficLightState);

unsigned long remainingTime = getRemainingTime(currentMillis) / 1000; // Convert to seconds

if (!isStaticTextInitialized) {

lcd\_0x20.clear();

lcd\_0x20.print("State: " + stateText);

isStaticTextInitialized = true;

}

// Only update the timer for states other than GREEN without a pedestrian request

if (trafficLightState != GREEN || pedestrianRequest) {

if (lastDisplayedTime != remainingTime) {

lcd\_0x20.setCursor(6, 1); // Adjust based on your layout

if (remainingTime < 10) { // Padding for single digit numbers

lcd\_0x20.print("0"); // Padding for consistency

}

lcd\_0x20.print(String(remainingTime + 1) + "s "); // Update time

lastDisplayedTime = remainingTime;

}

} else {

// If the state is GREEN and no pedestrian request, clear the timer display area

if (lastDisplayedTime != 0) {

lcd\_0x20.setCursor(6, 1); // Adjust cursor to timer position

lcd\_0x20.print(" "); // Clear the timer display area

lastDisplayedTime = 0; // Reset last displayed time

}

}

}

String getStateText(TrafficLightState state) {

switch (state) {

case RED: return "Red";

case RED\_YELLOW: return "Red+Yellow";

case GREEN: return "Green";

case YELLOW: return "Yellow";

default: return "";

}

}

unsigned long getRemainingTime(unsigned long currentMillis) {

switch (trafficLightState) {

case RED:

case GREEN:

return PEDESTRIAN\_CROSSING\_TIME - (currentMillis - stateChangeTimestamp);

case RED\_YELLOW:

case YELLOW:

return YELLOW\_TO\_RED\_DELAY - (currentMillis - stateChangeTimestamp);

default:

return 0;

}

}

void updateCarDisplay(bool isGreen) {

unsigned long currentMillis = millis();

if (currentMillis - previousMillis >= interval) {

previousMillis = currentMillis;

// Define messages as char arrays

const char greenMessage[] = "Commercial Green Car";

const char redMessage[] = "Commercial Red Car";

const char\* message = isGreen ? greenMessage : redMessage;

int messageLength = strlen(message); // Use strlen for char arrays

int displayWidth = 16;

int scrollRange = max(0, messageLength - displayWidth);

lcd\_0x26.clear();

char displayBuffer[17]; // Plus one for null terminator

memset(displayBuffer, ' ', sizeof(displayBuffer)); // Fill with spaces

displayBuffer[16] = '\0'; // Ensure null-termination

// Calculate start index based on current position and direction

int startIndex = scrollingForward ? position : scrollRange - position;

startIndex = max(0, min(startIndex, scrollRange));

// Copy relevant part of message to displayBuffer

for (int i = 0; i < displayWidth && (startIndex + i) < messageLength; i++) {

displayBuffer[i] = message[startIndex + i];

}

// Print the displayBuffer

lcd\_0x26.print(displayBuffer);

// Update position and change direction if needed

if (scrollingForward && position >= scrollRange) {

scrollingForward = false; // Change direction to scroll back

} else if (!scrollingForward && position <= 0) {

scrollingForward = true; // Change direction to scroll forward

}

// Update position for the next iteration

position += scrollingForward ? 1 : -1;

}

}

Part 2 ( Fanari ): Requirement 2 ( Optimized Code – Final )

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Produced By: Modern Computer Architecture

-- URL: https://mypad.northampton.ac.uk/smartsc/

-- Author: Charalampos Nikolaidis

-- Date: {February-March}/2024

-- Purpose: Make a simple pedestrial crossroad utilizing traffic lights. ( rbg / led )

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

/\*

--

General optimisation principles:

1. String literals can consume lots of storage.

2. Optimize variables. Remove unused variables and use long variables ( 4 bytes ) only if necessary.

3. Declare variables as locally as possible.

4. Avoid recursions.

5. Avoid including large libraries.

6. Store constant data in storage instead of memory with PROGMEM.

7. Skip the bootloader.

--

SRAM (Static Random-Access Memory) is used for storing variables and data structures while an Arduino program is running.

\*/

// ap, N. (n.d.). digitalWriteFast. GitHub. Available at: https://github.com/NicksonYap/digitalWriteFast [Accessed date: 06/03/2024].

// defined(\_\_AVR\_ATmega328\_\_) is defined in the library so arduino uno R3 will work with this.

#include <digitalWriteFast.h>

/\*

-- This library optimizes/replaces Arduino I/O operations via direct port manipulation, reducing execution time and resource usage.

-- The regular digitalWrite() in Arduino Uno core (16MHz) takes about 6280nS while digitalWriteFast() port manipulation takes 125nS.

-- It enhances energy efficiency minimizing CPU active cycles and RAM footprint.

-- Requires compile time constants for pin assignments that bypasses the Arduino layer resulting in reduction in exec times.

-- Its tailored for AVR microcontrollers, compatible with our atmega328 chipset.

\*/

#include <LiquidCrystal\_I2C.h>

// Defined pins this way so the library digitalWriteFast can utilize them properly as documented.

#define BUTTON\_PIN 2

#define MOTION\_SENSOR\_PIN 3

#define RED\_LIGHT\_CAR\_PIN 4

#define YELLOW\_LIGHT\_CAR\_PIN 5

#define GREEN\_LIGHT\_CAR\_PIN 6

#define RED\_LIGHT\_PED\_PIN 7

#define BUZZER\_PIN 9

#define RGB\_RED\_PIN 10

#define RGB\_GREEN\_PIN 11

#define CROSSWALK\_LEDS 12

#define LCD\_ADDRESS\_0x20 0x20

#define LCD\_ADDRESS\_0x26 0x26

#define YELLOW\_TO\_RED\_DELAY 2500

#define RED\_TO\_GREEN\_DELAY 3000

#define PEDESTRIAN\_CROSSING\_TIME 8000

#define SPEEDUP\_TIME 5000

#define MOTION\_DETECTED\_DELAY 5000

enum TrafficLightState {

RED,

RED\_YELLOW,

GREEN,

YELLOW

};

/\*

\* Due to a known issue with older Arduino IDE versions (in 1.7.x series from Arduino.org),

\* the compiler incorrectly places autogenerated function prototypes before custom type definitions,

\* causing build errors when enums are used as function arguments. This issue is discussed in detail

\* on the Arduino Stack Exchange (Majenko, 2016). The workaround involves manually declaring

\* function prototypes and custom types (enums, in this case) before their first use in the code.

\* This approach ensures compatibility across different versions of the Arduino IDE, including

\* Tinkercad's Arduino simulation environment, which may use an older build system similar to Arduino 1.7.x.

\* https://arduino.stackexchange.com/questions/28133/cant-use-enum-as-function-argument

\*/

void setTrafficLightState(TrafficLightState state);

void updateTrafficLightState(unsigned long currentMillis);

void checkMotionSensor();

void setPedestrianRGB(bool isRed);

void updateAudibleVisualCues(unsigned long currentMillis);

void updateLCDStatus(unsigned long currentMillis);

String getStateText(TrafficLightState state);

unsigned long getRemainingTime(unsigned long currentMillis);

void handlePedestrianButtonPress();

void processPedestrianRequest(unsigned long currentMillis);

// Global usefull vars i will be using later.

volatile bool pedestrianRequest = false;

volatile unsigned long buttonPressTimestamp = 0;

bool motionDetected = false;

unsigned long lastDisplayedTime = 0;

bool isStaticTextInitialized = false;

TrafficLightState trafficLightState = GREEN;

unsigned long stateChangeTimestamp = 0;

unsigned long previousMillis = 0; // Stores the last time the display was updated

const long interval = 300; // Interval at which to refresh the display (300ms)

int position = 0; // Current position of the scrolling text

bool scrollingForward = true; // Direction of scroll

LiquidCrystal\_I2C lcd\_0x20(LCD\_ADDRESS\_0x20, 16, 2);

LiquidCrystal\_I2C lcd\_0x26(LCD\_ADDRESS\_0x26, 16, 2);

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Initializes pins, serial communication, and interrupts for system setup.

-- Called by: Arduino runtime on start.

-- Modifications: --

1. Added Pins as progressed.

2. Multiple changes to lcd.

3. Changed to pinModeFast.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void setup() {

pinModeFast(BUTTON\_PIN, INPUT\_PULLUP);

pinModeFast(MOTION\_SENSOR\_PIN, INPUT);

pinModeFast(RED\_LIGHT\_CAR\_PIN, OUTPUT);

pinModeFast(YELLOW\_LIGHT\_CAR\_PIN, OUTPUT);

pinModeFast(GREEN\_LIGHT\_CAR\_PIN, OUTPUT);

pinModeFast(RED\_LIGHT\_PED\_PIN, OUTPUT);

pinModeFast(CROSSWALK\_LEDS, OUTPUT);

pinModeFast(BUZZER\_PIN, OUTPUT);

pinModeFast(RGB\_RED\_PIN, OUTPUT);

pinModeFast(RGB\_GREEN\_PIN, OUTPUT);

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

lcd\_0x20.init();

lcd\_0x20.backlight();

lcd\_0x26.init();

lcd\_0x26.backlight();

setTrafficLightState(GREEN);

Serial.begin(9600);

Serial.print("Free RAM: "); // in loop -4 bytes of ram

Serial.println(freeRam());

}

// func to print the free ram

int freeRam () {

extern int \_\_heap\_start, \*\_\_brkval;

int v;

return (int) &v - (\_\_brkval == 0 ? (int) &\_\_heap\_start : (int) \_\_brkval);

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Executes main control loop, handling sensor checks and state updates.

-- Called by: Arduino runtime.

-- Modifications: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void loop() {

unsigned long currentMillis = millis();

checkMotionSensor();

if (pedestrianRequest) {

processPedestrianRequest(currentMillis);

}

updateTrafficLightState(currentMillis);

updateAudibleVisualCues(currentMillis);

updateLCDStatus(currentMillis);

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Initiates pedestrian crossing request on button press.

-- Called by: Button press interrupt.

-- Modifications: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void handlePedestrianButtonPress() {

pedestrianRequest = true;

buttonPressTimestamp = millis();

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Processes pedestrian button requests within traffic light logic.

-- Called by: loop().

-- Modifications: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void processPedestrianRequest(unsigned long currentMillis) {

if (trafficLightState == GREEN && pedestrianRequest) {

if ((!motionDetected) || (motionDetected && currentMillis - buttonPressTimestamp >= MOTION\_DETECTED\_DELAY)) {

setTrafficLightState(YELLOW);

buttonPressTimestamp = currentMillis;

pedestrianRequest = false; // Reset request to prevent re-entry

}

}

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Sets the state of traffic lights and manages related device states.

-- Called by: processPedestrianRequest(), updateTrafficLightState().

-- Modifications:

1. Added logic to handle car light here instead of updateTrafficLightState();

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void setTrafficLightState(TrafficLightState state) {

trafficLightState = state;

stateChangeTimestamp = millis();

digitalWriteFast(RED\_LIGHT\_CAR\_PIN, state == RED || state == RED\_YELLOW ? HIGH : LOW);

digitalWriteFast(YELLOW\_LIGHT\_CAR\_PIN, state == YELLOW || state == RED\_YELLOW ? HIGH : LOW);

digitalWriteFast(GREEN\_LIGHT\_CAR\_PIN, state == GREEN ? HIGH : LOW);

// Manage pedestrian light directly within state transitions if needed

if(state == RED) {

delay(500); // Wait for 0.5 seconds

setPedestrianRGB(false); // Turn pedestrian light green

} else {

setPedestrianRGB(true); // Otherwise, keep pedestrian light red

}

// Specific handling when transitioning to GREEN to clear the timer display

if (state == GREEN) {

lcd\_0x20.setCursor(6, 1);

lcd\_0x20.print(" "); // Clear the timer display area

}

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Updates traffic light state based on timing and current state.

-- Called by: loop().

-- Modifications:

1. Added seperate red yellow logic instead of handling it withing case red.

2. Added simple yellow logic to use it from green to red.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void updateTrafficLightState(unsigned long currentMillis) {

// Extended logic to transition from YELLOW to RED for vehicles and then to make pedestrian light GREEN

switch (trafficLightState) {

case RED:

updateCarDisplay(false);

// After sufficient RED time, transition to RED\_YELLOW before going to GREEN for cars (and RED for pedestrians)

if (currentMillis - stateChangeTimestamp >= RED\_TO\_GREEN\_DELAY) {

setTrafficLightState(RED\_YELLOW);

}

break;

case RED\_YELLOW:

// Transition to GREEN for cars, which makes RED for pedestrians

if (currentMillis - stateChangeTimestamp >= YELLOW\_TO\_RED\_DELAY) {

setTrafficLightState(GREEN);

}

break;

case GREEN:

updateCarDisplay(true);

break;

case YELLOW:

// Transition from YELLOW to RED for cars, which will then trigger GREEN for pedestrians

if (currentMillis - stateChangeTimestamp >= YELLOW\_TO\_RED\_DELAY) {

setTrafficLightState(RED);

}

break;

}

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Checks motion sensor state and updates detection status.

-- Called by: loop().

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void checkMotionSensor() {

motionDetected = digitalReadFast(MOTION\_SENSOR\_PIN) == HIGH;

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Controls pedestrian RGB LED based on the provided state.

-- Called by: setTrafficLightState().

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void setPedestrianRGB(bool isRed) {

digitalWriteFast(RGB\_RED\_PIN, isRed ? HIGH : LOW);

digitalWriteFast(RGB\_GREEN\_PIN, !isRed ? HIGH : LOW);

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Manages auditory and visual cues based on traffic light state and timing.

-- Called by: loop().

-- Modifications:

1. Added functionality for SPEEDUP\_TIME

2. Added cross road lights

3. Fixes issue where cross road lights did not turn on for PEDESTRIAN\_CROSSING\_TIME

TODO: make crossroad lights flicker at speedUpElapsed.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void updateAudibleVisualCues(unsigned long currentMillis) {

if (trafficLightState == RED || trafficLightState == RED\_YELLOW) {

unsigned long elapsedTimeSinceStateChange = currentMillis - stateChangeTimestamp;

// Check if we are within the SPEEDUP\_TIME of the pedestrian crossing period ending

if (PEDESTRIAN\_CROSSING\_TIME - elapsedTimeSinceStateChange <= SPEEDUP\_TIME) {

// Calculate the proportion of the SPEEDUP\_TIME that has elapsed to scale the tone frequency

unsigned long speedUpElapsed = SPEEDUP\_TIME - (PEDESTRIAN\_CROSSING\_TIME - elapsedTimeSinceStateChange);

// Scale the frequency based on the elapsed time during the speed up period

int frequency = 1000 + (speedUpElapsed \* 1000 / SPEEDUP\_TIME); // Increase frequency over time

tone(BUZZER\_PIN, frequency);

// Flicker CROSSWALK\_LEDS during speed up time

if ((currentMillis / 250) % 2) { // control flicker speed

digitalWriteFast(CROSSWALK\_LEDS, HIGH);

} else {

digitalWriteFast(CROSSWALK\_LEDS, LOW);

}

} else {

tone(BUZZER\_PIN, 1000); // Default tone frequency before SPEEDUP\_TIME begins

digitalWriteFast(CROSSWALK\_LEDS, HIGH); // Keep CROSSWALK\_LEDS on before the SPEEDUP\_TIME starts

}

} else {

noTone(BUZZER\_PIN); // Turn off the tone in other states

digitalWriteFast(CROSSWALK\_LEDS, LOW); // Ensure CROSSWALK\_LEDS are off when not in RED or RED\_YELLOW state

}

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Updates LCD displays with current traffic light state and timing information.

-- Called by: loop().

-- Modifications:

1. Fixed the refresh rate issue

2. Fixes issue where time when red -> green showed the maximum integer number.

3. Fixed issue where after the 2. fix numbers showed as 00s when in green, instead of clear display.

4. Changes to make it more energy efficient and not refresh when not needed. ( example: green for cars )

\*\* 5. Failed attempts to make it print const char instead of string.\*\*

\*\* 6. Failed attempts to utilize PROGMEM. Maybe wrong buffer? \*\* --> see bottom

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void updateLCDStatus(unsigned long currentMillis) {

String stateText = getStateText(trafficLightState);

unsigned long remainingTime = getRemainingTime(currentMillis) / 1000; // Convert to seconds

if (!isStaticTextInitialized) {

lcd\_0x20.clear();

lcd\_0x20.print("State: " + stateText);

isStaticTextInitialized = true;

}

// Only update the timer for states other than GREEN without a pedestrian request

if (trafficLightState != GREEN || pedestrianRequest) {

if (lastDisplayedTime != remainingTime) {

lcd\_0x20.setCursor(6, 1);

if (remainingTime < 10) { // Padding for single digit numbers

lcd\_0x20.print("0"); // Padding

}

lcd\_0x20.print(String(remainingTime + 1) + "s "); // Update time

lastDisplayedTime = remainingTime;

}

} else {

// If the state is GREEN and no pedestrian request, clear the timer display area

if (lastDisplayedTime != 0) {

lcd\_0x20.setCursor(6, 1);

lcd\_0x20.print(" "); // Clear the timer display area

lastDisplayedTime = 0; // Reset last displayed time

}

}

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Returns textual representation of the current traffic light state.

-- Called by: updateLCDStatus().

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

String getStateText(TrafficLightState state) {

switch (state) {

case RED: return "Red";

case RED\_YELLOW: return "Red+Yellow";

case GREEN: return "Green";

case YELLOW: return "Yellow";

default: return "";

}

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Calculates remaining time for the current traffic light phase.

-- Called by: updateLCDStatus().

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

unsigned long getRemainingTime(unsigned long currentMillis) {

switch (trafficLightState) {

case RED:

case GREEN:

return PEDESTRIAN\_CROSSING\_TIME - (currentMillis - stateChangeTimestamp);

case RED\_YELLOW:

case YELLOW:

return YELLOW\_TO\_RED\_DELAY - (currentMillis - stateChangeTimestamp);

default:

return 0;

}

}

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: February-March/2024

-- Purpose: Updates LCD display for car state with scrolling text.

-- Called by: updateTrafficLightState().

-- Modifications:

1. Added interval refresh rate.

2. Added 2 commercial sentences, 1 for green and 1 for red car light state.

3. Added a horizontal scroll animation.

4. Fixed - added buffer to not stcroll endlessly.

\*\* 5. Failed attempt to make characters and animations using bytes instead of simple words \*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

void updateCarDisplay(bool isGreen) {

unsigned long currentMillis = millis();

if (currentMillis - previousMillis >= interval) {

previousMillis = currentMillis;

// Define messages as char arrays

const char greenMessage[] = "Commercial Green Car";

const char redMessage[] = "Commercial Red Car";

const char\* message = isGreen ? greenMessage : redMessage;

int messageLength = strlen(message); // Use strlen for char arrays

int displayWidth = 16;

int scrollRange = max(0, messageLength - displayWidth);

lcd\_0x26.clear();

char displayBuffer[17]; // Plus one for null terminator

memset(displayBuffer, ' ', sizeof(displayBuffer)); // Fill with spaces

displayBuffer[16] = '\0'; // Ensure null-termination

// Calculate start index based on current position and direction

int startIndex = scrollingForward ? position : scrollRange - position;

startIndex = max(0, min(startIndex, scrollRange));

// Copy relevant part of message to displayBuffer

for (int i = 0; i < displayWidth && (startIndex + i) < messageLength; i++) {

displayBuffer[i] = message[startIndex + i];

}

// Print the displayBuffer

lcd\_0x26.print(displayBuffer);

// Update position and change direction if needed

if (scrollingForward && position >= scrollRange) {

scrollingForward = false; // Change direction to scroll back

} else if (!scrollingForward && position <= 0) {

scrollingForward = true; // Change direction to scroll forward

}

// Update position for the next iteration

position += scrollingForward ? 1 : -1;

}

}

/\* ------------------------------------------------

This is a way to make lcd more efficient.

It is incomplete because i could not retrieve the const char from program memory.

My final thoughts on what could it went wrong, is the buffer. Looping through it crashed the lcds.

------------------------------------------------ \*/

// #include <avr/pgmspace.h>

// Global scope for PROGMEM strings

// const char greenCarMsg[] PROGMEM = "Commercial Green Car";

// const char redCarMsg[] PROGMEM = "Commercial Red Car";

// void displayStaticEfficientFromProgmem() {

// char buffer[25]; // Buffer for copying the message from PROGMEM

// strcpy\_P(buffer, (isGreen ? greenCarMsg : redCarMsg));

// buffer[24] = '\0'; // Explicitly null-terminate

// lcd\_0x26.clear();

// lcd\_0x26.print(buffer); // Display the static message

// }

// void updateCarDisplay(bool isGreen) {

// unsigned long currentMillis = millis();

// if (currentMillis - previousMillis >= interval) {

// previousMillis = currentMillis;

// displayStaticTestFromProgmem();

// }

// }

**Μέρος 2: Hello World (Φανάρι Κυκλοφορίας) (45%)**

VIDEO:

<https://nikolaidis.info/csym028/part_2/>

Σύνδεσμοι προς τα σχέδια στο Tinkercad (δύο λύσεις):

Requirement 1:

<https://www.tinkercad.com/things/lPZFlrjV19c-fanari>

Requirement 2: ( Final + Optimized )

<https://www.tinkercad.com/things/bfDTuaUNUOD-final-fanari>

Μέρος 3 (10%)

Για κάθε μια από τις παρακάτω δραστηριότητες, υποβάλλετε την πιο δημιουργική και προχωρημένη εκδοχή σας\*:

|  |
| --- |
| Interrupts: Instead of digitalPinToInterrupt calling a function on button press ( pin 2 ), I invoked the ISR. Inside setup I enabled the internal pull up on the pin 2 input. To attach the interrupt directly via Register Manipulation, I needed to use External Interrupt Control Register and External Interrupt Mask Register for our chip. This setup trigger the interrupt on the edge but you can adjust ISC00 and ISC01 in EICRA for different trigger conditions. In my example I gained efficiency and memory usage bypassing the Arduino layer. |
| Registers | DDR | Port | PIN:  In all of my examples, in both parts I found an external library that utilizes register manipulations bypassing the Arduino layer and overwriting current Arduino functions.  This library is also documented and acknowledged by Arduino referenced here: <https://www.arduino.cc/reference/en/libraries/digitalwritefast/>  The regular digitalWrite() in Arduino Uno core (16MHz) takes about 6280nS while digitalWriteFast() port manipulation takes 125nS.  The library approach could be slightly better to direct port manipulation due to similar efficiency results and better code maintainability and readability.  This library bypasses the Arduino layers and directly manipulates the ports defined.  #denife is a requirement for the library. |
| Activity Light, Switch, Motion Sensor, Pot, LDR, Ultrasonic:  In my examples above, using millis(), I focused on reducing the unnecessary ‘reading’ of the sensors. Each sensor should be read only when needed.  In updateCarDisplay (eg)  The String class dynamically holds memory, which can lead to fragmentation or increased memory usage.  Prefering to use character arrays for static text like char can optimize the code. I have more examples commented out at the end.  I also optimized the amount of times the processor reads each input/output. Having if statements to prevent each sensor to be read unnecessarily. |
| LDR | Analog Input/Output: We can optimize analog signals with smoother LDR readings, reading the signals less times, adjusting ADCSRA before the reads or with register manipulations which proved to be quite a challenge. In my case I reduced the reading times of these ports as much as possible.  I also tried using PROGMEM storing data in flash memory, freeing up SRAM. Char also uses less space than String reducing heap fragmentation.  Direct manipulation of the microcontroller registers allows tighter control of the hardware, resulting in faster response times and potentially lower power consumption, as it eliminates the overhead introduced by the Arduino layer which is quite abstract.  I have such examples in the end of the final/optimized solutions.  I also optimized the amount of times the processor reads each input/output. Having if statements to prevent each sensor to be read unnecessarily. |
| Sound | Piezo: We can incorporate fewer things for these sensors such as: use of faster pin toggling for sound, generating sound only when needed, utilizing sleep modes for silence periods |

\*\* NOTE FOR TINKERCAD\*\*

However, as noted in my video, while Tinkercad provides a great option for prototyping and learning, it may not fully simulate the nuances of register manipulation due to the abstraction of higher level Arduino functions.

Testing on a physical ESP32 microcontroller and on a raspberry PI 4, I noticed differences in performance gains and behavior in both direct port manipulation and digitalWriteFast, both being faster than the simulator.