ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ЩОДО ЗАБЕЗПЕЧЕННЯ САМОСТІЙНОЇ РОБОТИ СТУДЕНТІВ З ДИСЦИПЛІНИ «Робототехніка»

Частина 3

Київ, 2024

Методичні рекомендації щодо забезпечення самостійної роботи студентів з дисципліни «Робототехніка»-ч.3 для студентів усіх форм навчання спеціальності 121 «Інженерія програмного забезпечення».

Методична розробка містить опис загальної інформації, завдання щодо самостійного вивчення студентами та самоконтролю, індивідуальні завдання, складаєься з трьох частин. Призначена для методичного забезпечення самостійної роботи студентів денної форми навчання, які вивчають навчальну дисципліну «Робототехніка»

Розробники:

ГОРДІЄНКО Олександр Миколайович, кандидат технічних наук, доцент кафедри комп'ютерних інформаційних систем та технологій Інститут комп'ютерно-інформаційних технологій та дизайну ПрАТ «ВНЗ «Міжрегіональна Академія управління персоналом»

КОВАЛЬ Аліна Олександрівна, викладач кафедри комп'ютерних інформаційних систем та технологій Інститут комп'ютерно-інформаційних технологій та дизайну ПрАТ «ВНЗ «Міжрегіональна Академія управління персоналом»

Розглянуто та ухвалено на засіданні кафедри комп'ютерних інформаційних систем та технологій Інституту комп'ютерно-інформаційних технологій та дизайну ПрАТ «ВНЗ «Міжрегіональна Академія управління персоналом»

Протокол №__ від «__» _____ 2024 р.

Завідувач кафедри _____ КАВУН Сергій Віталійович

Гордієнко О.М., Коваль А.О. Методичні рекомендації щодо забезпечення самостійної роботи студентів з дисципліни «Робототехніка» .-ч.3. – К.: МАУП, 2024. – 40 с.

Зміст

Характеристики плати

Плата розробки на мікроконтроллері ESP32-S3 N16R8:

- світлодіодний модуль WS2812
- камера ST7735S
- роз'єм microSD Card
- USB-UART порт для програмування контроллера
- USB-OTG порт для підключення пристроїв
- PSRAM – 8 МБ зовнішньої оперативної пам'яті, що підключена до контактів 35, 36, 37.

Встановляння програмного забезпечення

Програмується контроллер через USB-UART перехідник CH343 (правий Type-C порт на платі).

Середовище розробки:
Arduino IDE

Бібліотека для роботи з платою:
eps32 By Espressif Systems

Плата:
4D Systems gen4-ESP32 16MB Modules (ESP32-S3R8n16)

Драйвер **CH343**:
https://www.wch-ic.com/search?t=all&q=ch343

Налаштування середовища розробки

USB CDC On Boot: **disabled**

Налаштовує USB-OTG порт як Communications Device Class (serial port). Тобто дані, що ви виводите на консоль (Serial.println()) можуть йти на USB-OTG порт замість вже під'єднаного USB-UART.

USB DFU on Boot: **disabled**

Налаштовує USB-OTG порт як Device Firmware Upgrade.

USB Firmware MSC On Boot: **disabled**

Налаштовує USB-OTG порт як Mass Storage Class (тобто флешка).

Partition Scheme: **16MB App**

App – наш код (прошивка). **OTA** пам'ять виділена під оновлення по **WIFI**. **SPIFFS** – пам'ять виділена під зберігання файлів (флешка).

Upload Mode: **UART0**

Завантаження прошивки через USB-UART порт.

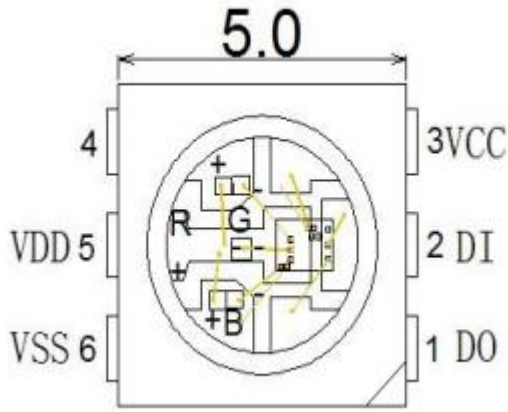
Upload Speed: **921600**

Швидкість передачі даних USB-UART під час прошивки.

USB Mode: **USB-OTG (TinyUSB)**

Розумний світлодіод

WS2812 — це світлодіодний модуль, який інтегрує в собі RGB-світлодіод і керований драйвер в одному корпусі. Кожен світлодіод може випромінювати червоне, зелене або синє світло, і їх комбінації дозволяють отримати широкий спектр кольорів. Однією з головних особливостей WS2812 є можливість керування кольорами кожного світлодіода окремо за допомогою єдиної лінії даних.
Наша плата має один вбудований світлодіодний модуль, що під'єднаний до gpio48.



LedWs2812.ino

```
#include "WS2812FX.h"

#define LEDS_COUNT  1
#define LEDS_PIN  48

WS2812FX ws2812 = WS2812FX(LEDS_COUNT, LEDS_PIN,
NEO_GRB);

void setup() {
  ws2812.begin();
  ws2812.setBrightness(10);
}
```

```
void loop() {
  setLedColor(0xff0000);
  delay(500);

  setLedColor(0x00ff00);
  delay(500);

  setLedColor(0x0000ff);
  delay(500);

  setLedColor(0xffffff);
  delay(500);
  setLedColor(0x000000);
  delay(500);
}

void setLedColor(uint32_t color){
  ws2812.setPixelColor(0, color);
  ws2812.execShow();
}
```

Задача 1.

Використайте функцію setBrightness(), щоб плавно перемикати колір (червоний, зелений, синій, білий).

Тактова кнопка та світлодіод

Вбудована тактова кнопка
Плата має дві вбудовані кнопки reset та boot. Кнопка Reset перезавантажує контроллер, а кнопка boot дозволяє вводити контролер в режим завантаження коду та користуватись нею, як звичайною кнопкою.
Коли система завантажуєть та кнопка boot натиснута - контроллер перейде в режим завантаження прошивки. Коли система завантажилась, ми можемо використовувати цю кнопку для своїх потреб. Кнопка під'єднана до gpio0 та підтягнута до +5V.

Вбудований світлодіод
Світлодіод під'єднаний до gpio2.

PushButton.ino

```
const int buttonPin = 0;
const int ledPin = 2;

bool buttonState;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  buttonState = digitalRead(buttonPin);
  digitalWrite(ledPin, buttonState);
}
```

Датчики дотику як кнопки

ESP32 підтримує функцію сенсорних датчиків (touch sensors), яка дозволяє виявляти дотик до певних пінів мікроконтролера. Це робить ESP32 дуже корисним для проєктів з інтерактивним управлінням. Однією з цікавих можливостей є touch interrupt — переривання на основі сенсорних сигналів, яке дозволяє реагувати на дотик без постійного опитування пінів.
Як працює сенсорне переривання в ESP32:

1. Сенсорні піни: 14 сенсорних пінів, які використовуються для вимірювання ємності. Якщо до піну торкаються пальцем або іншим об'єктом, ємність змінюється, і чіп це фіксує.
2. Поріг спрацьовування (threshold): Для кожного сенсорного піну можна встановити поріг чутливості. Коли сенсорний сигнал падає нижче цього порогу (що вказує на дотик), викликається переривання.
3. Переривання: Використання сенсорних переривань дозволяє ESP32 виконувати інші завдання і реагувати лише тоді, коли відбувається дотик, що економить ресурси.

TouchButton.ino

```
#include "Arduino.h"

int threshold = 1500;
bool touch1detected = false;

int T1 = 1;

void gotTouch1() {
  touch1detected = true;
}

void setup() {
  Serial.begin(115200);
  delay(1000);

  Serial.println("\n ESP32 Touch Interrupt Test\n");
  touchAttachInterrupt(T1, gotTouch1, threshold);
}
```

```
void loop() {
  if (touch1detected) {
    touch1detected = false;
    if (touchInterruptGetLastStatus(T1)) {
      Serial.println(" --- Touched");
    } else {
      Serial.println(" --- Released");
    }
  }

  delay(80);
}
```

Задача 2.

Напишіть код, який буде плавно перемикати колір світлодіодного модуля ws2812. Використайте тактову кнопку для перемикання кольорів.

Задача 3.

Використайте чотири сенсори дотику для управління кольором та яскравістю світлодіодного модуля. Дві кнопки на яскравість, дві на колір.

Переривання по таймеру

Таймери переривання на ESP32 дозволяють виконувати певний код з фіксованими інтервалами часу, незалежно від основного коду, що виконується в програмі. Це корисно для точного виконання завдань з інтервалами, таких як вимірювання часу, генерація сигналів чи опитування датчиків. ESP32 має чотири апаратні таймери, які можуть працювати незалежно. Таймери можуть бути налаштовані для створення переривань через фіксований проміжок часу, після чого вони можуть зупинитися (одноразовий режим) або повторюватися (циклічний режим).
Налаштуємо таймер на генерацію переривання кожну секунду, щоб змінювати стан світлодіода.

timer.ino

```
#define LED_PIN 2

hw_timer_t *timer = NULL;
volatile bool ledState = false;

void IRAM_ATTR onTimer() {
  ledState = !ledState;
  digitalWrite(LED_PIN, ledState);
}

void setup() {
  Serial.begin(115200);

  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, ledState);

  timer = timerBegin(1000000);
  timerAttachInterrupt(timer, &onTimer);
  timerAlarm(timer, 1000000, true, 0);
}

void loop() {}
```

Температура процесора

Вбудований температурний сенсор дозволяє вимірювати внутрішню температуру мікроконтролера, що корисно для захисту від перегріву або діагностики.

TempSensor.ino

```
void setup() {
  Serial.begin(115200);
}

void loop() {
  float temp_celsius = temperatureRead();

  Serial.print("Temp onBoard ");
  Serial.print(temp_celsius);
  Serial.println("°C");

  delay(1000);
}
```

Задача 4.

Використайте таймер, щоб перемикати колір світлодіодного модуля ws2812 кожні три секунди.

Задача 5.

Використайте таймер, щоб отримувати температуру щосекунди.

Зовнішня пам’ять

ESP32-S3 підтримує роботу з SD-картами, що дозволяє використовувати їх як зовнішнє сховище для зберігання даних, журналів або медіафайлів. SD-карти можна підключити через SPI або SDMMC інтерфейси.
SDMMC — це інтерфейс для підключення SD-карт до мікроконтролерів через спеціалізований контролер. На відміну від SPI режиму, який використовує стандартні пін-контролери для обміну даними, SDMMC дозволяє здійснювати більш швидкий і надійний обмін даними з SD-картою завдяки підтримці більш високих швидкостей передачі та додаткових ліній передачі.
На платі встановлено microSd роз’єм, що підключено до 38, 39, 40 пінів. З використанням трьох пінів контроллер працює в однобітному режимі.

SdCard.ino

```
#include "FS.h"
#include "SD_MMC.h"

int clk = 39;
int cmd = 38;
int d0 = 40;

void setup() {
  Serial.begin(115200);

  if(! SD_MMC.setPins(clk, cmd, d0)){
      Serial.println("Pin change failed!");
      return;
  }

  if (!SD_MMC.begin("/sdcard", true)) {
    Serial.println("Card Mount Failed");
    return;
  }

  uint8_t cardType = SD_MMC.cardType();
  Serial.print("SD_MMC Card Type: ");
  switch(cardType){
```

```
    case CARD_NONE:
      Serial.println("No SD_MMC card attached");
      return;
    case CARD_MMC:
      Serial.println("MMC");
      break;
    case CARD_SD:
      Serial.println("SDSC");
      break;
    case CARD_SDHC:
      Serial.println("SDHC");
      break;
    default:
      Serial.println("UNKNOWN");
  }
  Serial.printf("SD_MMC Card Size: %lluMB\n", SD_MMC.cardSize() /
(1024 * 1024));
  Serial.printf("Total space: %lluMB\n", SD_MMC.totalBytes() / (1024
* 1024));
  Serial.printf("Used space: %lluMB\n", SD_MMC.usedBytes() / (1024
* 1024));
  Serial.println("Listing directory");
  File root = SD_MMC.open("/");
  if (!root || !root.isDirectory()) {
    Serial.println("Failed to open directory");
    return;
  }
  File file = root.openNextFile();
  while (file) {
    if (file.isDirectory()) {
      Serial.printf(" DIR: %s\n", file.name());
    } else {
      Serial.printf("FILE: %s\tSIZE: %u\n", file.name(), file.size());
    }
    file = root.openNextFile();
  }
}
```

USB клавіатура

Запрограмуємо контроллер для роботи у режимі клавіатури. Плата має два Type-C USB роз'єми. Правий, для програмування. Лівий – для комунікації через USB інтерфейс.
Підключаємо лівий роз'єм плати до комп'ютера. Комп'ютер бачить нашу плату як клавіатуру. Якщо натиснути кнопку boot на платі, то плата починає друкувати текст. Щоб побачити, що друкується, відкрийте, наприклад, блокнот та встановіть курсор для друку. Натисніть кнопку boot.

KeyboardMessage.ino

```
#include "USB.h"
#include "USBHIDKeyboard.h"
USBHIDKeyboard Keyboard;

const int buttonPin = 0;
int previousButtonState = HIGH;
int counter = 0;

void setup() {
  pinMode(buttonPin, INPUT_PULLUP);
  Keyboard.begin();
  USB.begin();
}

void loop() {
  int buttonState = digitalRead(buttonPin);
  if ((buttonState != previousButtonState)&& (buttonState == LOW)) {
    counter++;
    Keyboard.print("You pressed the button ");
    Keyboard.print(counter);
    Keyboard.println(" times.");
  }
  previousButtonState = buttonState;
}
```

USB миша

Запрограмуємо контроллер для роботи у режимі миші. Підключаємо лівий порт до комп'ютера. Натискаючи кнопку boot, курсор миші буде переміщатися на 1 піксель по горизонталі та вертикалі кожні 5 мілі секунд.

MouseControll.ino

```
#include "USB.h"
#include "USBHIDMouse.h"
USBHIDMouse Mouse;

const int buttonPin = 0;
const int x = 1;
const int y = 1;

void setup() {
  pinMode(buttonPin, INPUT_PULLUP);
  Mouse.begin();
  USB.begin();
}

void loop() {
  int buttonState = digitalRead(buttonPin);

  if (buttonState == LOW){
    Mouse.move(x, y);
  }

  delay(5);
}
```

USB HID Vendor

USB HID Vendor – це пристрій, що використовує USB для зв'язку з комп'ютером і працює за протоколом HID (Human Interface Device). Цей протокол зазвичай використовується для пристроїв введення, як-от клавіатури, миші, джойстики. Однак "Vendor" означає, що цей пристрій створений спеціально для певного застосування. Це може бути спеціалізоване обладнання, яке взаємодіє з комп'ютером, але не належить до типових HID-пристроїв. Наприклад, це може бути спеціальний контролер, розроблений для певної задачі в промисловості або іграх, який використовує протокол HID для простого підключення без встановлення складних драйверів.

HIDVendor.ino

```
#include "USB.h"
#include "USBHIDVendor.h"
USBHIDVendor Vendor;

const int buttonPin = 0;
int previousButtonState = HIGH;

void setup() {
  pinMode(buttonPin, INPUT_PULLUP);
  Vendor.begin();
  USB.PID(0x0008);
  USB.VID(0x0004);
  USB.productName("MyName");
  USB.manufacturerName("I_AM");
  USB.serialNumber("555-000-555");
  USB.begin();
}

void loop() {
  int buttonState = digitalRead(buttonPin);
  if ((buttonState != previousButtonState) && (buttonState == LOW)) {
    Vendor.println("Hello World!");
  }
  previousButtonState = buttonState;
}
```

Дані можна приймати написавши програму у C або Python з використанням бібліотеки HID.

Задача 6.

Створіть новий файл на карті пам’яті. Записуйте дані температурного сенсора у файл.

Задача 7.

Створіть файл на карті пам’яті microSD та наповніть його довільним текстом, наприклад, кодом програми. Зчитайте файл за допомогою ESP32 та передайте на комп’ютер за допомогою USBHIDKeyboard.

Завдання 8.

Використайте чотири датчики дотику для керування курсором миші.

Задача 9.

Напишіть програму, яка буде отримувати команди з комп’ютера для керування яскравістю та кольором модульного світлодіода ws2812.

Wifi Blink

Створимо власну WIFI мережу та веб сервера для керування світлодіодом на платі. Підключимось до платичи через телефон використовуючи назву точки та пароль, що запрограмовані в коді, як ssid та password. На серевері створимо головну сторінку, яка міститиме перемикач світлодіода. Щоб перейти на головну сторінку нашого веб сервера налаштуємо mDNS, щоб ip-адреса сервера відповідала домену esp.local або використовуємо ip-адресу веб сервера.

WebBlink.ino

```
#include <WiFi.h>
#include <WebServer.h>
#include <ESPmDNS.h>

// Ваше ім'я Wi-Fi мережі
const char* ssid = "esp1";
// Пароль Wi-Fi
const char* password = "123456789";

#define LED_PIN 2

WebServer server(80);

void welcome() {
  // Головна сторінка з кнопками для керування
  // LED і відображенням поточного стану
  String html = "<!DOCTYPE html><html><head>"
                "<meta charset='UTF-8'>"
"<meta name='viewport' content='width=device-width, initial-
scale=1'><title>LED Control </title></head><body><h1>Керування
LED</h1>";

  // Додаємо інформацію про поточний стан LED
  if (digitalRead(LED_PIN) == HIGH) {
    html += "<p>LED зараз: Увімкнений</p>";
  } else {
    html += "<p>LED зараз: Вимкнений</p>";
  }
```

```
  // Кнопки для керування LED
  html +=
"<form action='/light_up' method='POST'>"
"<button type='submit'>Light Up</button>"
"</form><br>"
"<form action='/light_down' method='POST'>"
"<button type='submit'>Light Down</button>"
"</form></body></html>";

  server.send(200, "text/html", html);
}

void lightUp() {
  digitalWrite(LED_PIN, HIGH);
  server.sendHeader("Location", "/", true);
  server.send(302, "text/plain", "Lighting Up LED! Redirecting...");
}

void lightDown() {
  digitalWrite(LED_PIN, LOW);
  server.sendHeader("Location", "/", true);
  server.send(302, "text/plain", "Light Down LED! Redirecting...");
}

void setup() {
  pinMode(LED_PIN, OUTPUT);

  Serial.begin(115200);

  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, password);

  // Ініціалізація mDNS
  if (!MDNS.begin("esp")) { //esp.local
    Serial.println("Не вдалося налаштувати mDNS!");
    while (1) {
      delay(1000);
    }
  }
```

```
  server.on("/", HTTP_GET, welcome);
  server.on("/light_up", HTTP_POST, lightUp);
  server.on("/light_down", HTTP_POST, lightDown);

  server.begin();
  Serial.println("HTTP сервер запущений!");

  Serial.print("http://esp.local on ");
  Serial.println(WiFi.softAPIP());
}

void loop() {
  server.handleClient();
}
```

Задача 10.

Змініть код так, щоб ESP32 підключалася до існуючої WIFI мережі.

Wifi WS2812

Створимо власну Wifi мережу та веб сервер, який буде працювати за адресою http://esp.local та керуватимемо світлодіодним модулем. Головна сторінка містить повзунки яскравості та кольору світлодіода.

WebWS2812.ino

```
#include <Adafruit_NeoPixel.h>
#include <WiFi.h>
#include <WebServer.h>
#include <ESPmDNS.h>

#define LED_PIN 48 // Пін для підключення WS2812
#define NUM_LEDS 1 // Кількість світлодіодів у
                   // стрічці (на платі 1)

Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUM_LEDS, LED_PIN,
NEO_GRB + NEO_KHZ800);

int brightness = 255; // Яскравість (0 - 255)
uint32_t currentColor = strip.Color(255, 0, 0);  // Початковий колір
(червоний)

// HTTP сервер
WebServer server(80);

// Функція для оновлення кольору і яскравості
void updateLED() {
  strip.setBrightness(brightness);
  strip.setPixelColor(0, currentColor); // Встановлення кольору
  strip.show();  // Оновлення LED
}

void handleRoot() {
  // Головна сторінка з кнопками для керування
  // LED і відображенням поточного стану
  char hexColor[8];
  sprintf(hexColor, "#%06X", currentColor);
```

```
  String html = "<!DOCTYPE html>"
"<html><head><title>LED Control</title>"
"<meta charset='UTF-8'>"
"<meta name='viewport' "
"content='width=device-width, initial-scale=1'>"
"</head><body>"
"<h1>Керування кольором та яскравістю</h1>"
"<p>Поточна яскравість: " + String(brightness) + "</p>"
// Форма для вибору кольору
"<form action='/set_color' method='POST'>"
"Обрати колір: <input type='color' "
"name='color' value='" + String(hexColor) + "'><br>"
"<button type='submit'>Set Color</button>"
"</form><br>"
// Форма для зміни яскравості
"<form action='/set_brightness' method='POST'>"
"Brightness: <input type='range' "
"name='brightness' min='0' max='255' value='" + String(brightness) +
"' onchange='this.nextElementSibling.value="
"this.value'><br>"
"<output>" + String(brightness) + "</output><br>"
"<button type='submit'>Set Brightness</button>"
"</form>"
"</body></html>";

  server.send(200, "text/html", html);
}

// Обробник для зміни кольору
void handleSetColor() {
  if (server.hasArg("color")) {
    String colorStr = server.arg("color");
    if (colorStr.charAt(0) == '#') {
      // Видалення символу '#'
      colorStr.remove(0, 1);
    }
    currentColor = strtol(colorStr.c_str(), nullptr, 16);
  }

  updateLED();  // Оновити колір LED
  server.sendHeader("Location", "/", true);
```

```
  server.send(302, "text/plain", "Redirecting...");
}

// Обробник для зміни яскравості
void handleSetBrightness() {
  if (server.hasArg("brightness")) {
    brightness = server.arg("brightness").toInt();
  }

  updateLED();  // Оновити яскравість LED
  server.sendHeader("Location", "/", true);
  server.send(302, "text/plain", "Redirecting...");
}

void setup() {
  Serial.begin(115200);

  // Налаштування SoftAP
  WiFi.softAP("EspWeb", "123456789");
  Serial.println("SoftAP запущений!");

  // Ініціалізація mDNS
  if (!MDNS.begin("esp")) { //esp.local
    Serial.println("Не вдалося налаштувати mDNS!");
    while (1) {
      delay(1000);
    }
  }

  strip.begin(); // Ініціалізація стрічки WS2812
  strip.show(); // Очищення всіх пікселів
  updateLED(); // початкове налаштування

  // Налаштування маршрутів
  // Головна сторінка
  server.on("/", HTTP_GET, handleRoot);
  // Маршрут для зміни кольору
  server.on("/set_color", HTTP_POST, handleSetColor);
  // Маршрут для зміни яскравості
  server.on("/set_brightness", HTTP_POST, handleSetBrightness);
```

```
  server.begin();
  Serial.println("HTTP сервер запущений на http://esp.local");
}

void loop() {
  server.handleClient();  // Обробка запитів
}
```

Задача 11.

Додайте на головну сторінку виведення температури процесора ESP32.

Дисплей ST7735S

Підключимо дисплей до плати. Пін камери VCC до +5В плати. Пін камери LED до +3.3В плати. GND до GND. Інші піни підключаємо, як запрограмовано в коді.

TftTest.ino

```
#include <Adafruit_GFX.h>
#include <Adafruit_ST7735.h>
#include <SPI.h>

#define TFT_SCLK   42    // SCK
#define TFT_MOSI   41    // SDA
#define TFT_DC     40    // A0
#define TFT_RST    39    // RESET
#define TFT_CS     38    // CS

Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC,
TFT_MOSI, TFT_SCLK, TFT_RST);

void setup(void) {
  tft.initR(INITR_BLACKTAB);
  tft.fillScreen(ST77XX_BLACK);

  tft.println("Hello World!");
  delay(1000);

  // play
  tft.fillScreen(ST77XX_BLACK);
  tft.fillRoundRect(25, 10, 78, 60, 8, ST77XX_WHITE);
  tft.fillTriangle(42, 20, 42, 60, 90, 40, ST77XX_RED);
  delay(500);
  // pause
  tft.fillRoundRect(25, 90, 78, 60, 8, ST77XX_WHITE);
  tft.fillRoundRect(39, 98, 20, 45, 5, ST77XX_GREEN);
  tft.fillRoundRect(69, 98, 20, 45, 5, ST77XX_GREEN);
  delay(500);
  // play color
  tft.fillTriangle(42, 20, 42, 60, 90, 40, ST77XX_BLUE);
  delay(50);
```

```
  // pause color
  tft.fillRoundRect(39, 98, 20, 45, 5, ST77XX_RED);
  tft.fillRoundRect(69, 98, 20, 45, 5, ST77XX_RED);
  // play color
  tft.fillTriangle(42, 20, 42, 60, 90, 40, ST77XX_GREEN);
}

void loop() {
  tft.invertDisplay(true);
  delay(500);
  tft.invertDisplay(false);
  delay(500);
}
```

Задача 12.

Створіть власну WIFI мережу та виведіть SSID, пароль та ip-адресу сервера на дисплей.

Фотокамера

Виведемо фото з камери на дисплей натискаючи кнопку boot.
Зображення із камери отримуємо у стисненому форматі JPEG.
Конвертуємо у формат BMP RGB 565.

Camera2Display.ino

```
#include <esp_camera.h>
#include <Adafruit_GFX.h>
#include <Adafruit_ST7735.h>
#include <TJpg_Decoder.h>

const int buttonPin = 0;
volatile bool buttonPressed = false;

void IRAM_ATTR handleButtonPress() {
  buttonPressed = true;
}

#define TFT_SCLK   42    // SCK
#define TFT_MOSI   41    // SDA
#define TFT_DC     40    // A0
#define TFT_RST    39    // RESET
#define TFT_CS     38    // CS

Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC,
TFT_MOSI, TFT_SCLK, TFT_RST);

#define XCLK_GPIO_NUM  15
#define SIOD_GPIO_NUM  4
#define SIOC_GPIO_NUM  5

#define Y2_GPIO_NUM 11
#define Y3_GPIO_NUM 9
#define Y4_GPIO_NUM 8
#define Y5_GPIO_NUM 10
#define Y6_GPIO_NUM 12
#define Y7_GPIO_NUM 18
#define Y8_GPIO_NUM 17
#define Y9_GPIO_NUM 16
```

```
#define VSYNC_GPIO_NUM 6
#define HREF_GPIO_NUM  7
#define PCLK_GPIO_NUM  13

static camera_config_t camera_config = {
    .pin_pwdn     = -1,
    .pin_reset    = -1,
    .pin_xclk     = XCLK_GPIO_NUM,
    .pin_sccb_sda = SIOD_GPIO_NUM,
    .pin_sccb_scl = SIOC_GPIO_NUM,

    .pin_d7 = Y9_GPIO_NUM,
    .pin_d6 = Y8_GPIO_NUM,
    .pin_d5 = Y7_GPIO_NUM,
    .pin_d4 = Y6_GPIO_NUM,
    .pin_d3 = Y5_GPIO_NUM,
    .pin_d2 = Y4_GPIO_NUM,
    .pin_d1 = Y3_GPIO_NUM,
    .pin_d0 = Y2_GPIO_NUM,

    .pin_vsync = VSYNC_GPIO_NUM,
    .pin_href  = HREF_GPIO_NUM,
    .pin_pclk  = PCLK_GPIO_NUM,

    .xclk_freq_hz = 20000000,
    .ledc_timer   = LEDC_TIMER_0,
    .ledc_channel = LEDC_CHANNEL_0,

    .pixel_format = PIXFORMAT_JPEG,
    .frame_size = FRAMESIZE_SXGA,

    .jpeg_quality = 12,
    .fb_count = 1,
    .fb_location = CAMERA_FB_IN_PSRAM,
    .grab_mode = CAMERA_GRAB_LATEST
};

bool tft_output(int16_t x, int16_t y,
uint16_t w, uint16_t h, uint16_t* bitmap)
{
```

```
  if ( y >= tft.height() ) return 0;
  tft.drawRGBBitmap(x, y, bitmap, w, h);
  return 1;
}
void setup() {
  attachInterrupt(
     digitalPinToInterrupt(buttonPin),
     handleButtonPress, FALLING);

  Serial.begin(115200);
  esp_err_t err = esp_camera_init(&camera_config);
  if (err != ESP_OK) {
    Serial.println("Camera Init Failed");
  }
  sensor_t *s = esp_camera_sensor_get();
  s->set_hmirror(s, 1);

  tft.initR(INITR_BLACKTAB);
  tft.setRotation(1);

  TJpgDec.setJpgScale(8);
  TJpgDec.setCallback(tft_output);
}
void take_photo(){
    camera_fb_t * fb = NULL;
    fb = esp_camera_fb_get();
    if (!fb) {
        Serial.println("Camera capture failed");
        return;
    }

    TJpgDec.drawJpg(0, 0, fb->buf, fb->len);
    esp_camera_fb_return(fb);
}
void loop() {
  if (buttonPressed) {
    take_photo();
    buttonPressed = false;
  }
  delay(50);
}
```

Онлайн відеоспостереження

Налаштуємо плату, щоб підключатися до існуючої мережі WIFI та веб сервер, що буде показувати відео з камери у браузері. В консолі буде надруковано ip-адресу веб сервера.

Використаємо приклад:
Examples>ESP32>Camera>CameraWebServer

Обираємо #define CAMERA_MODEL_ESP32S3_EYE

Вказуємо назву WIFI та пароль. Тільки 2.4GHz
const char *ssid = "**********";
const char *password = "**********";

Задача 13.

Створіть власну WIFI мережу та веб сервер. На головній сторінці розмістіть кнопку. Кнопка має робити фото та виводити на дисплей.

Задача 14.

Користуючись прикладом Camera2Display.ino створіть маршрут "/image.jpg". Створіть обробник, який зробить фото та відправить клієнту. Відправляючи зображення, змініть тип контенту з "text/html" на "image/jpeg"

Синхронізація часу з інтернетом

Підключимось до існуючої мережі WIFI. Синхронізуємо годинник контроллера з сервером часу pool.ntp.org використовуючи бібліотеку time.h та виведемо час на дисплей. Строка EET-2EEST,M3.5.0/3,M10.5.0/4 задає правила зміни часових поясів та переходу на літній і зимовий час. EET-2EEST — це часовий пояс. M3.5.0/3 — це правило переходу на літній час. M10.5.0/4 — це правило повернення до стандартного часу.

WifiHttpTime2Tft.ino

```
#include <Adafruit_GFX.h>
#include <Adafruit_ST7735.h>
#include <SPI.h>

#include <WiFi.h>
#include <time.h>

const char* ssid     = "***";
const char* password = "******";

// Timezone for Europe/Kyiv (EET + DST)
const char* timeZone =
            "EET-2EEST,M3.5.0/3,M10.5.0/4";
const char* ntpServer = "pool.ntp.org";

#define TFT_SCLK   42    // SCK
#define TFT_MOSI   41    // SDA
#define TFT_DC     40    // A0
#define TFT_RST    39    // RESET
#define TFT_CS     38    // CS

Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC,
TFT_MOSI, TFT_SCLK, TFT_RST);

void setup(void) {
  tft.initR(INITR_BLACKTAB);
  tft.fillScreen(ST77XX_BLACK);

  // Connect to Wi-Fi
```

```
  tft.print("Connecting to ");
  tft.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    tft.print(".");
  }
  tft.println("");
  tft.println("WiFi connected.");

  configTzTime(timeZone, ntpServer);
  tft.println("Time synchronized with NTP server.");

  // Wait for time to be set
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    tft.println("Failed to obtain time");
    return;
  }

  tft.setTextSize(2);
  tft.setTextColor(ST77XX_WHITE, ST77XX_BLACK);
  tft.fillScreen(ST77XX_BLACK);
}
void loop() {
  tft.setCursor(0, 0);
  // Get the current time
  struct tm timeinfo;
  if (getLocalTime(&timeinfo)) {
    tft.println(&timeinfo, "%A");
    tft.println(&timeinfo, "%B %d");
    tft.println(&timeinfo, "%Y");
    tft.println();
    tft.println(&timeinfo, "%H:%M:%S");
  } else {
    tft.println("Failed to obtain time");
  }
  delay(200);
}
```

Виділення паям’яті для файлів

ESP32-S3 підтримує файлову систему SPIFFS (SPI Flash File System), яка дозволяє зберігати файли безпосередньо у флеш-пам’яті мікроконтролера. Це зручно для зберігання налаштувань, вебсторінок, зображень або інших статичних ресурсів.
Оберемо в налаштуваннях **Partition Scheme** – Small або Medium App.

spiffs.ino

```
#include "FS.h"
#include "SPIFFS.h"

void setup() {
  Serial.begin(115200);
  if (!SPIFFS.begin(true)) {
    Serial.println("SPIFFS Mount Failed.");
  }

  uint64_t total =
          SPIFFS.totalBytes() / (1024 * 1024);
  uint64_t used =
          SPIFFS.usedBytes() / (1024 * 1024);

  Serial.printf("Total space: %lluMB\n", total);
  Serial.printf("Used space: %lluMB\n", used);

  File file =
      SPIFFS.open("/example.txt", FILE_APPEND);
  if (!file) {
    Serial.println("Помилка відкриття файлу для запису");
    return;
  }
  file.println("Привіт, SPIFFS!");
  file.close();
  Serial.println("Дані записані у файл");

  // Читання даних з файлу
  file = SPIFFS.open("/example.txt", FILE_READ);
  if (!file) {
```

```
    Serial.println("Помилка відкриття файлу для читання");
    return;
  }
  Serial.println("Дані з файлу:");
  while (file.available()) {
    Serial.write(file.read());
  }
  file.close();
}

void loop() {}
```

Задача 15.

Додайте ще два часові пояси:
Australia/Melbourne "EST-10EST,M10.1.0,M4.1.0/3"
America/Los_Angeles "PST8PDT,M3.2.0,M11.1.0"
Натискаючи тактову кнопку, перемикайте часові пояси та виводьте назву країни.

Задача 16.

Зкопіюйте текстовий файл з microSD у spiffs. Виведіть вміст файлу у консоль.

Використовуємо PSRAM

Pseudo Static RAM — це тип пам'яті, який дозволяє розширити обсяг доступної оперативної пам'яті для програм. Зазвичай використовується для зберігання даних, які не потребують високошвидкісного доступу, таких як великі масиви, буфери для зображень, аудіо дані, тощо. Розмір PSRAM у esp32-s3 n16r8 - 8 МБ. Це значно більше, ніж кількість пам'яті, доступна в інтегрованій SRAM (520 КБ), що дозволяє розробникам працювати з великими обсягами даних.

psram.ino

```
void setup() {
  Serial.begin(115200);

  Serial.printf("Розмір SRAM: %zu байтів\n\n", ESP.getHeapSize());
  Serial.printf("Доступно SRAM: %zu байтів\n\n",
ESP.getFreeHeap());
  Serial.printf("Розмір PSRAM: %zu байтів\n\n", ESP.getPsramSize());
  Serial.printf("Доступно PSRAM: %zu bytes\n", ESP.getFreePsram());
  Serial.println("Створюємо масив розміром 4МБ у PSRAM");
  uint32_t* data = (uint32_t *) ps_malloc(1024 * 1024 *
sizeof(uint32_t));
  Serial.printf("Доступно PSRAM: %zu байтів\n",
ESP.getFreePsram());
  free(data);
}
void loop() {}
```

BLE Server

BLE (Bluetooth Low Energy) сервер — це пристрій, який транслює свої послуги (сервіси) та характеристики (characteristics) через BLE з'єднання, дозволяючи іншим BLE-сумісним пристроям (клієнтам) підключатись до нього та взаємодіяти з його даними. BLE сервери найчастіше використовуються в пристроях, які надають певну інформацію або приймають команди — це можуть бути датчики, медичні пристрої, смарт-годинники тощо.

ble_echo.ino

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

// Налаштування сервісу і характеристики
#define SERVICE_UUID "8D53DC1D-1DB7-4CD3-868B-8A527460AA84"
#define CHARACTERISTIC_UUID "DA2E7828-FBCE-4E01-AE9E-261174997C48"

BLECharacteristic *pCharacteristic;
bool deviceConnected = false;
String receivedValue = "";

// Клас, що обробляє події сервера
class MyServerCallbacks : public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
        deviceConnected = true;
        Serial.println("Клієнт під'єднаний");
    }

    void onDisconnect(BLEServer* pServer) {
        deviceConnected = false;
        Serial.println("Клієнт від'єднаний");
    }
};

// Клас, що обробляє події характеристики
```

```
class MyCallbacks : public BLECharacteristicCallbacks {
 void onWrite(BLECharacteristic *pCharacteristic) {
  receivedValue = pCharacteristic->getValue();
  if (receivedValue.length() > 0) {
    Serial.print("Отримано: ");
    Serial.println(receivedValue.c_str());
    // Відправка ехо-відповіді
    pCharacteristic->setValue(receivedValue);
    pCharacteristic->notify();
   }
 }
};

void setup() {
 Serial.begin(115200);

 // Ініціалізація BLE
 BLEDevice::init("BLE Echo Server");
 BLEServer *pServer = BLEDevice::createServer();
 pServer->setCallbacks(new MyServerCallbacks());

 // Створення сервісу
 BLEService *pService =
         pServer->createService(SERVICE_UUID);

   // Створення характеристики
 pCharacteristic =
   pService->createCharacteristic(
       CHARACTERISTIC_UUID,
       BLECharacteristic::PROPERTY_READ |
       BLECharacteristic::PROPERTY_WRITE |
       BLECharacteristic::PROPERTY_NOTIFY
      );
 pCharacteristic->
            setCallbacks(new MyCallbacks());
 pCharacteristic->addDescriptor(new BLE2902());

 // Запуск сервісу
 pService->start();
 pServer->getAdvertising()->start();
 Serial.println("BLE Echo Server запущено.");
```

```
}

void loop() {
  // Можна додати іншу логіку або перевірки
  delay(1000);
}
```

Задача 17.

Створіть WIFI веб сервер. На головній сторінці має бути форма надсилання файлу. Сервер має приймати файл та зберігати у spiffs.

Задача 18.

Отримайте значення температури процесора відправивши команду temp на BLE-сервер.