ДОДАТОК А

Лістинг функції для розпізнавання підключеного до i2c шини давача

```
void scanAndDetectSensors() {
   Serial.println("Сканування шини I2C...");
   byte error, address;
   int nDevices = 0;
   // Скидаємо прапорці для всіх датчиків
   bh1750_detected = false;
   bh1750_2_detected = false;
   bme680_detected = false;

   for(address = 1; address < 127; address++ ) {
      Wire.beginTransmission(address);
      error = Wire.endTransmission();

      if (error == 0) { // Пристрій знайдено
         Serial.printf("Знайдено пристрій I2C за адресою 0x%02X", address);
         nDevices++;
         if (address == BH1750_I2C_ADDR) { // Перевірка першого BH1750
            Serial.println("  >>> BH1750 #1 Виявлено!");
            bh1750_detected = true;
         }
         else if (address == BH1750_2_I2C_ADDR) { // Перевірка другого
BH1750
             Serial.println("  >>> BH1750 #2 Виявлено!");
             bh1750_2_detected = true;
         }
         else if (address == BME680_I2C_ADDR) { // Перевірка BME680
             Serial.println("  >>> BME680 Виявлено!");
             bme680_detected = true;
         }
         else {
             Serial.println("  (Невідомий пристрій)");
         }
      } else if (error == 4) {
          // Залишаємо логування невідомої помилки, якщо потрібно
          // Serial.printf("Невідома помилка за адресою 0x%02X\n", address);
      }
   }

   if (nDevices == 0) {
```

```
        Serial.println("Пристроїв I2C не знайдено");
    } else {
        Serial.println("Сканування завершено.");
    }

    // --- Ініціалізація BH1750 #1 ---
    if (bh1750_detected) {
      if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE)) {
          Serial.println(F("BH1750 #1 успішно ініціалізовано."));
      } else {
          Serial.println(F("!!! Помилка ініціалізації BH1750 #1! Перевірте
підключення? !!!"));
          bh1750_detected = false; // Скидаємо прапорець, якщо ініціалізація не
вдалася
      }
    }

    // --- Ініціалізація BH1750 #2 ---
    if (bh1750_2_detected) {
      // Важливо: Використовуємо об'єкт lightMeter2 та іншу адресу
      if (lightMeter2.begin(BH1750::CONTINUOUS_HIGH_RES_MODE,
BH1750_2_I2C_ADDR)) {
          Serial.println(F("BH1750 #2 успішно ініціалізовано."));
      } else {
          Serial.println(F("!!! Помилка ініціалізації BH1750 #2! Перевірте
підключення/адресу? !!!"));
          bh1750_2_detected = false;
      }
    }
```

ДОДАТОК Б

Лістинг коду для виведення OSD даних

```
void displayOsdData()
{
  unsigned long currentTime = millis();
  unsigned long elapsedTime = (currentTime - startTime) / 1000;

  int minutes = elapsedTime / 60;
  int seconds = elapsedTime % 60;

  char timerBuffer[10];
  sprintf(timerBuffer, "%02d:%02d", minutes, seconds);

  osd_print(timerBuffer, strlen(timerBuffer), 13, 1);

  if (VOLT1ENABLE) readvolts(VOLT1, VOLT1XPOS, VOLT1YPOS);
  if (VOLT2ENABLE) readvolts(VOLT2, VOLT2XPOS, VOLT2YPOS);
  if (LED1ENABLE) readled(LED1, LED1XPOS, LED1YPOS);

  vcc = getvcc();

  if (vcc < 4.4)
  {
    // warning for vcc undervoltage
  }
  vcc = vcclpf.step(vcc);

  if (flag)
  {
    osd_print("!", 1, FLAGXPOS, FLAGYPOS);
    if (enableserial) Serial.print(" ! ");
  }
  else
  {
    osd_print(" ", 1, FLAGXPOS, FLAGYPOS);
    if (enableserial) Serial.print("   ");
  }

  checksystem();

  static int asd = 0;
```

```
  static int asd2 = 0;

  char name1[] = "Davach1:";
  char name2[] = "Davach2:";
  char bufffer[10];
  char bufffer2[10];
  char bufffer3[10];
  check_osd();

  if (Serial.available() > 0) {
    String inputString = Serial.readStringUntil('\n');
    inputString.trim();
    int semicolonIndex1 = inputString.indexOf(';');
    if (semicolonIndex1 != -1) {
      String davach1Str = inputString.substring(0, semicolonIndex1);
      int semicolonIndex2 = inputString.indexOf(';', semicolonIndex1 + 1);
      if (semicolonIndex2 != -1) {
        String davach2Str = inputString.substring(semicolonIndex1 + 1,
semicolonIndex2);
        String ina219Str = inputString.substring(semicolonIndex2 + 1);

        asd = davach1Str.toInt();
        asd2 = davach2Str.toInt();
        ina219_voltage = ina219Str.toFloat();

        Serial.print("Davach1: ");
        Serial.println(asd);
        Serial.print("Davach2: ");
        Serial.println(asd2);
        Serial.print("INA219 Voltage: ");
        Serial.println(ina219_voltage);
      } else {
        Serial.println("Неверный формат данных от ESP32. Ожидается
'значение1;значение2;напряжение'.");
      }
    } else {
      Serial.println("Неверный формат данных от ESP32. Ожидается
'значение1;значение2;напряжение'.");
    }
  }

  itoa(asd, bufffer, 10);
  itoa(asd2, bufffer2, 10);
  dtostrf(ina219_voltage, 5, 2, bufffer3);
```

```
  char outputBuffer[10] = "         ";
  char outputBuffer2[10] = "         ";
  char outputBuffer3[10] = "         ";
  strncpy(outputBuffer, bufffer, 9);
  osd_print(outputBuffer, 10, 5, 10);

  strncpy(outputBuffer2, bufffer2, 9);
  osd_print(outputBuffer2, 10, 5, 11);

  osd_print(name1, strlen(name1), 1, 10);
  osd_print(outputBuffer, strlen(outputBuffer), 9, 10);
  osd_print(name2, strlen(name2), 1, 11);
  osd_print(outputBuffer2, strlen(outputBuffer2), 9, 11);

  cs_off();
  SPI.end();

  led_flasher();
}

void loop()
{
  unsigned long timeloop = micros();

  displayOsdData();

  if (micros() - timeloop > LOOPTIME - 1000)
  {
    if (enableserial) Serial.print("<<<<<<<<<<<<<<<LOOP TIME");
  }
  while (micros() - timeloop < LOOPTIME)
  {
  }

  SPI.begin();

  if (enableserial) Serial.println("");
}
```

ДОДАТОК В

Лістинг коду взаємодії мобільного застосунку та ESP32

```
package com.example.dronedatamaster



// Data class ActiveSensorsInfo
@Serializable
data class ActiveSensorsInfo(
    val detected_sensors: List<String> = emptyList(),
    val configured_sensor_type: String = "None",
    val configured_sensor_pin: Int = -1
)
// Ktor HTTP client
val client = HttpClient(CIO) {
    install(ContentNegotiation) {
        json(Json {
            ignoreUnknownKeys = true
            isLenient = true
        })
    }
    install(WebSockets) { }
    engine { requestTimeout = 10000 }
}
// Fetch DroneData (assuming DroneData is defined elsewhere)
suspend fun fetchDataFromESP32(): DroneData? {
    val url = "http://192.168.4.1/data"
    return try {
        Log.d("FetchData", "Fetching data from $url")
        val response = client.get(url)
        if (response.status == HttpStatusCode.OK) {
            response.body<DroneData>()
        } else {
            Log.e("FetchData", "Error fetching data: Status code
${response.status.value}")
            null
        }
    } catch (e: Exception) {
        Log.e("FetchData", "Error fetching data: ${e.message}")
        null
    }
```

```
}

// Send calibration request
suspend fun sendCalibrationHttpRequest(): String {
    val url = "http://192.168.4.1:80/calibrate"
    return try {
        Log.d("CalibrationHttp", "Sending POST request to $url")
        val response = client.post(url)
        val responseBody = response.body<String>()
        Log.d("CalibrationHttp", "Received response: ${response.status}, Body: $responseBody")
        if (response.status == HttpStatusCode.OK) {
            responseBody
        } else {
            "Помилка сервера: ${response.status.value}"
        }
    } catch (e: Exception) {
        Log.e("CalibrationHttp", "Error sending calibration request: ${e.message}")
        "Помилка сети: ${e.message ?: "Неизвестная ошибка"}"
    }
}
// Fetch active sensors info
suspend fun fetchActiveSensors(): Result<ActiveSensorsInfo> {
    val url = "http://192.168.4.1/active_sensors"
    return try {
        Log.d("NetworkFetch", "Запрос активных датчиков: $url")
        val response = client.get(url)
        val sensorsInfo = Json { ignoreUnknownKeys = true }.decodeFromString<ActiveSensorsInfo>(response.body<String>())
        Log.d("NetworkFetch", "Распарсенная информация: $sensorsInfo")
        Result.success(sensorsInfo)
    } catch (e: Exception) {
        Log.e("NetworkFetch", "Ошибка получения активных датчиков: ${e.message}", e)
        Result.failure(e)
    }
}
// Fetch sensor statistics (assuming SensorStat is defined elsewhere)
suspend fun fetchSensorStats(): Result<List<SensorStat>> {
    val url = "http://192.168.4.1:80/get_sensor_stats"
    return try {
        Log.d("SensorStats", "Fetching GET request from $url")
        val response = client.get(url)
        if (response.status == HttpStatusCode.OK) {
            val stats = response.body<List<SensorStat>>()
```

```
            Log.d("SensorStats", "Successfully fetched ${stats.size} stats records")
            Result.success(stats)
        } else if (response.status == HttpStatusCode.NotFound) {
            Log.w("SensorStats", "Log file not found on ESP32 (404)")
            Result.success(emptyList())
        } else {
            val errorMsg = "Ошибка сервера: ${response.status.value}"
            Log.e("SensorStats", "Error fetching stats: $errorMsg")
            Result.failure(IOException(errorMsg))
        }
    } catch (e: Exception) {
        Log.e("SensorStats", "Error fetching stats: ${e.message}", e)
        Result.failure(e)
    }
}
// Send sensor type configuration
suspend fun sendSensorTypeHttpRequest(sensorType: String): String {
    val url = "http://192.168.4.1/set_sensor_type"
    return try {
        Log.i("NetworkSend", "Отправка POST на $url с типом: $sensorType")
        val response = client.submitForm(
            url = url,
            formParameters = parametersOf("type", listOf(sensorType))
        )
        val responseBody = response.body<String>()
        Log.i("NetworkSend", "Ответ от /set_sensor_type: $responseBody")
        responseBody
    } catch (e: Exception) {
        Log.e("NetworkSend", "Помилка відправки типу давача: ${e.message}", e)
        "Ошибка сети: ${e.message ?: "Невідома помилка"}"
    }
}
// Main Activity
class MainActivity : ComponentActivity() {
    @OptIn(ExperimentalMaterial3Api::class)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            val systemTheme = isSystemInDarkTheme()
            var isDarkTheme by rememberSaveable { mutableStateOf(systemTheme) }

            DroneDataMasterTheme(
                darkTheme = isDarkTheme
            ) {
```

```
            val navController = rememberNavController()
            Scaffold(
                modifier = Modifier.fillMaxSize(),
                topBar = {
                    TopAppBar(
                        title = { Text("Drone Data Master") },
                        actions = {
                            IconButton(onClick = { isDarkTheme = !isDarkTheme }) {
                                Icon(
                                    imageVector = if (isDarkTheme) {
                                        Icons.Outlined.LightMode
                                    } else {
                                        Icons.Outlined.DarkMode
                                    },
                                    contentDescription = "Переключить тему"
                                )
                            }
                        }
                    )
                },
                bottomBar = { BottomNavigationBar(navController = navController) }
            ) { innerPadding ->
                NavHost(
                    navController = navController,
                    startDestination = Screen.FlightController.route,
                    modifier = Modifier.padding(innerPadding)
                ) {
                    composable(Screen.Sensors.route) {
                        SensorScreen(modifier = Modifier.fillMaxSize())
                    }
                    composable(Screen.FlightController.route) {
                        FlightControllerScreen(modifier = Modifier.fillMaxSize())
                    }
                }
            }
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        client.close()
        Log.d("MainActivity", "Ktor client closed.")
    }
}
```

```
// Screens for navigation
sealed class Screen(val route: String, val title: String) {
    object Sensors : Screen("sensors", "ESP32 Sensors")
    object FlightController : Screen("flight_controller", "STM32 FC")
}

// Bottom navigation bar component
@Composable
fun BottomNavigationBar(navController: NavController) {
    val items = listOf(
        Screen.FlightController,
        Screen.Sensors
    )
    NavigationBar {
        val navBackStackEntry by navController.currentBackStackEntryAsState()
        val currentRoute = navBackStackEntry?.destination?.route
        items.forEach { screen ->
            NavigationBarItem(
                icon = { Text(text = screen.title.take(3).uppercase()) },
                label = { Text(screen.title) },
                selected = currentRoute == screen.route,
                onClick = {
                    if (currentRoute != screen.route) {
                        navController.navigate(screen.route) {
                            popUpTo(navController.graph.startDestinationId) { saveState =
true }
                            launchSingleTop = true
                            restoreState = true
                        }
                    }
                }
            )
        }
    }
}
```

ДОДАТОК Г

Демонстраційний матеріал

№ докумен-та	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮИК 290415.024 ПЗ	Пояснювальна записка	А4, 85 с.
		Додаткові матеріали	
2		Демонстраційний матеріал у	А4, 14 с.
		вигляді презентації	

Змін.	Арк.	Номер докум.	Підпис	Дата	ГЮИК 290415.024 ВД			
Розроб.		Коваль А.Р			Розроблення літаючого макету дрону на базі ARM мікроконтролера для збору та обробки даних у реальному часі	Літера	Аркуш	Аркушів
Перевір.		Євсєєв В.В.				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.		Демська Н.П						
Затв.		Невлюдов І.Ш.						