ДОДАТОК А

Схема електрична підключення компонентів приладу

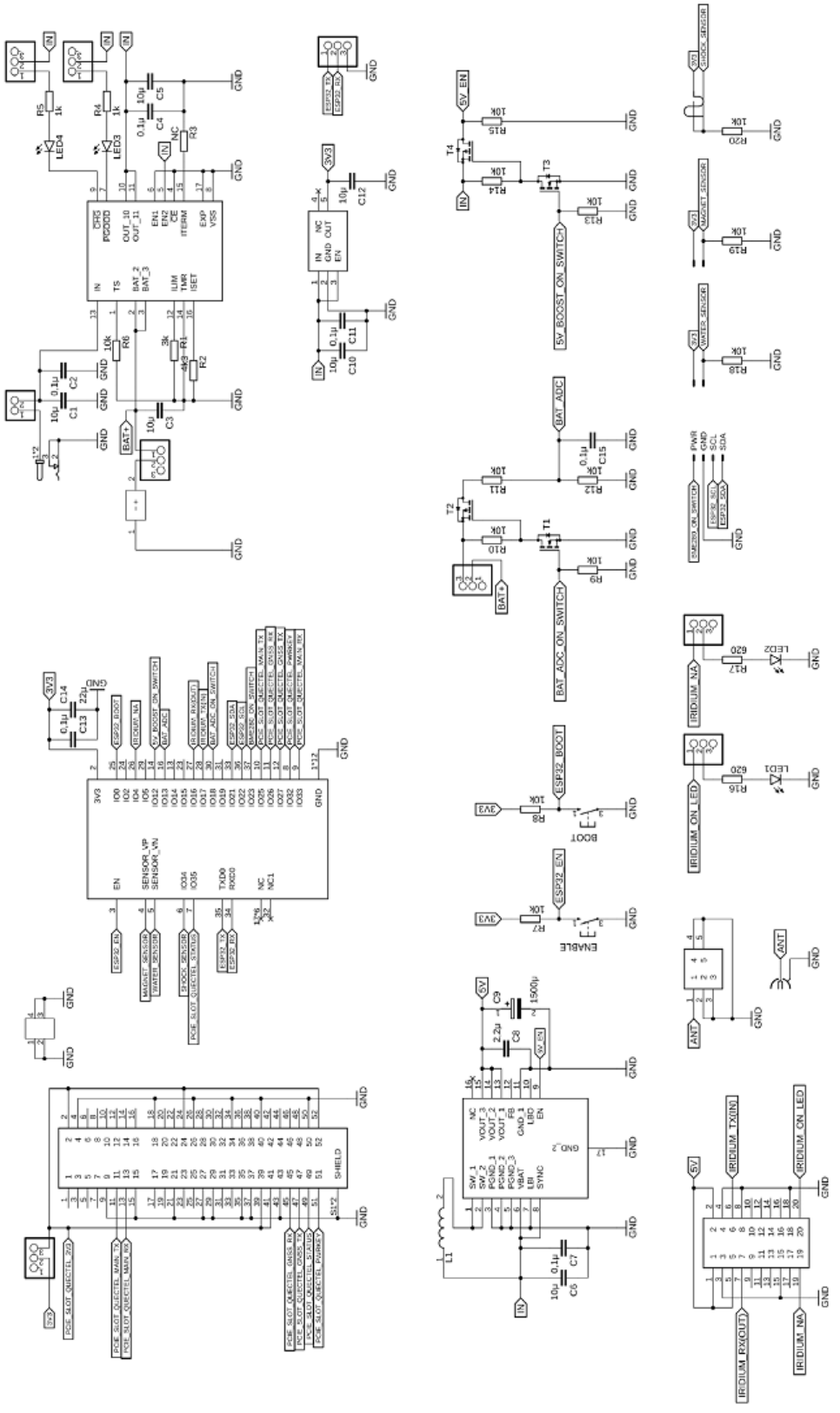

Рисунок А.1 – Схема електрична підключення компонентів приладу

ДОДАТОК Б

Код програми

```
#include <Adafruit_BME280.h>
#include <esp_adc_cal.h>
```

```
#include <TinyGPSPlus.h>
#include <TimeLib.h>

#define ADCVRef 1080 // calibration function, mV, measured by routing VRef to GPIO pin (using multimeter)
#define BUTTON_PIN_BITMASK 0x9400000000 // GPIO 34, 36 and 39

#define ShockSensor 34
#define MagnetSensor 36
#define WaterSensor 39

#define BatADCRead 13
#define SwitchPinBatteryADC 18

#define SwitchPinBME280 23

#define QuectelMainTX 25 // Quectel side Main receive
#define QuectelMainRX 33 // Quectel side Main transmit
#define SwitchPinQuectel 32

#define IridiumTX 17 // Iridium side receive
#define IridiumRX 16 // Iridium side transmit
#define IridiumNetwork 4
#define SwitchPinIridium 12

typedef struct __attribute__((packed)) {
  uint8_t version;
  uint8_t len;
  uint8_t batteryLevel;
  uint8_t flags;
  uint64_t timestamp;
  uint16_t temperature;
  uint16_t humidity;
  uint16_t pressure;
  int32_t latitude;
  int32_t longitude;
  int32_t altitude;
  uint8_t HDOP;
} Payload;

HardwareSerial quectelSerial(1);
HardwareSerial iridiumSerial(2);
Adafruit_BME280 BME280;
TinyGPSPlus GPS;
tmElements_t readableTimestamp;
time_t unixTimestamp;

// RTC variables (hold value during deepsleep mode)
RTC_DATA_ATTR Payload payload; // Final message
RTC_DATA_ATTR uint16_t timeToSleep;
RTC_DATA_ATTR uint64_t bootCount;
RTC_DATA_ATTR bool initSettings;
RTC_DATA_ATTR char quectelIMEI[16];

bool shockSensorState;
bool magnetSensorState;
bool waterSensorState;

char APN[32] = "lpwa.vodafone.iot"; // APN for Vodafone NB-IoT network
char serverIP[16] = "185.59.146.24"; // GINA server IP for NB-IoT
uint16_t port = 30070; // GINA server port for NB-IoT

char receivedDataBuffer[256];
```

```
bool sendATCommand(const char command[], const char awaitedReply[], uint32_t maxWaitTime, HardwareSerial
& refSer) {
  uint32_t startMillis;
  uint16_t index;
  char inChar;
  bool replyMatch = false;

  for (uint16_t i = 0; i < sizeof(receivedDataBuffer); i++) { // Clear received data buffer array
    receivedDataBuffer[i] = '\0';
  }

  while (refSer.available() > 0) { // Clear ESP32 RX input buffer
    refSer.read();
  }

  Serial.printf("\nCommand: %s", command);

  refSer.println(command);

  startMillis = millis();

  while (!replyMatch && millis() - startMillis < maxWaitTime) {
    while (refSer.available() > 0) {
      inChar = refSer.read();
      receivedDataBuffer[index] = inChar;
      index++;
      if (strstr(receivedDataBuffer, awaitedReply) != NULL) replyMatch = true; // Check if received reply matches
    }
  }

  Serial.print(receivedDataBuffer);

  if (replyMatch) return true;
  else return false;
}

void initConfig() {
  bool readInput = true;
  String inString = "";
  char inChar;
  char fullCommand[64];

  Serial.print("\nSet time for deepsleep [minutes] (min 10, max 65536; only numbers will be parsed): ");

  while (readInput == true) {
    while (Serial.available() > 0) {
      inChar = Serial.read();
      if (isDigit(inChar)) inString += (char)inChar;
      if (inChar == 13) { // \n
        if(inString.toInt() > 9 && inString.toInt() < 65536) {
          timeToSleep = inString.toInt();
          Serial.printf("\nDeepsleep time has been set to %u minutes.\n", timeToSleep);
          readInput = false;
        }
        else {
          Serial.println("\nInput error. Restart the module and try again.");
          esp_deep_sleep_start();
        }
      }
    }
  }
```

```
  Serial.println("\nInitializaiton configuration has started.");

  payload.version = 1;
  payload.len = sizeof(payload);

  Serial.println("\nIRIDIUM");

  digitalWrite(SwitchPinIridium, HIGH);

  delay(1000);

  if (sendATCommand("AT", "OK", 350, iridiumSerial)) {
    sendATCommand("ATE0", "OK", 350, iridiumSerial); // Local echo off
    sendATCommand("AT&D0", "OK", 350, iridiumSerial); // Disable DTR signal
    sendATCommand("AT&K0", "OK", 350, iridiumSerial); // Disable flow control
    sendATCommand("AT&W0", "OK", 350, iridiumSerial); // Save active configuration to profile 0
    sendATCommand("AT&Y0", "OK", 350, iridiumSerial); // Default reset profile set to 0
  } else {
    Serial.println("\nIridium module seems not to be connected. Please check and restart the device.");
    esp_deep_sleep_start();
  }

  sendATCommand("AT*F", "OK", 10500, iridiumSerial);
  digitalWrite(SwitchPinIridium, LOW);

  Serial.println("\n----------------------------------------");

  Serial.println("\nQUECTEL");

  digitalWrite(SwitchPinQuectel, HIGH);
  delay(900);
  digitalWrite(SwitchPinQuectel, LOW);
  delay(3000);

  if (sendATCommand("AT", "OK", 350, quectelSerial)) {
    sendATCommand("ATE0", "OK", 350, quectelSerial); // Local echo off
    sendATCommand("AT&D0", "OK", 350, quectelSerial); // Disable DTR signal
    sendATCommand("AT+IFC=0,0", "OK", 350, quectelSerial); // Disable flow control
    sendATCommand("AT+GSN", "OK", 350, quectelSerial); // IMEI
    char * bufferSource = & receivedDataBuffer[2]; // Skip /r and /n in buffer
    strncat(quectelIMEI, bufferSource, 15); // Store IMEI
    sendATCommand("AT+CFUN=0", "OK", 15500, quectelSerial); // Turn off radio
    sendATCommand("AT+QCFG=\"nwscanseq\",030303,1", "OK", 350, quectelSerial); // Searching sequence of
radio access technologies prioritizes NB-IoT
    sendATCommand("AT+QCFG=\"iotopmode\",1,1", "OK", 350, quectelSerial); // Network category to be
searched under LTE RAT set to NB-IoT
    sendATCommand("AT+CFUN=1", "OK", 15500, quectelSerial); // Turn on radio
    sprintf(fullCommand, "AT+QICSGP=1,1,\"%s\",\"\",\"\",1", APN);
    sendATCommand(fullCommand, "OK", 350, quectelSerial); // APN setup
    sendATCommand("AT&W0", "OK", 350, quectelSerial); // Save active configuration to default conguration
  } else {
    Serial.println("\nQuectel module seems not to be connected. Please check and restart the device.");
    esp_deep_sleep_start();
  }

  sendATCommand("AT+QPOWD", "POWERED DOWN", 10500, quectelSerial);

  Serial.println("\nInitializaiton configuration has finished.");

  initSettings = true;
}
```

```
void getSensorValues() {
  esp_sleep_wakeup_cause_t wakeup_reason;

  Serial.println("\nSensor checking has started.");

  wakeup_reason = esp_sleep_get_wakeup_cause();

  if (wakeup_reason == ESP_SLEEP_WAKEUP_EXT1) Serial.println("\nWakeup caused by external signal using
RTC_CNTL.");
  else if (wakeup_reason == ESP_SLEEP_WAKEUP_TIMER) Serial.println("\nWakeup caused by internal timer.");
  else Serial.println("\nWakeup was not caused by RTC_CNTL or internal timer.");

  if (esp_sleep_get_ext1_wakeup_status() ==
0b0000000000000000000000000000010000000000000000000000000000000000 || shockSensorState == HIGH) {
    payload.flags += 2;
    Serial.println("Shock Sensor activity (GPIO34)");
  }
  if (esp_sleep_get_ext1_wakeup_status() ==
0b0000000000000000000000000000100000000000000000000000000000000000 || magnetSensorState == HIGH) {
    payload.flags += 1;
    Serial.println("Magnet Sensor activity (GPIO36)");
  }
  if (esp_sleep_get_ext1_wakeup_status() ==
0b0000000000000000000000000001000000000000000000000000000000000000 || waterSensorState == HIGH) {
    payload.flags += 4;
    Serial.println("Water Sensor activity (GPIO39)");
  }

  Serial.println("\nSensor checking has finished.");
}

void getBatteryLevel() {
  float voltage;
  int raw;

  Serial.println("\nBattery level measurement has started.");

  adc2_config_channel_atten(ADC2_CHANNEL_4, ADC_ATTEN_11db);

  digitalWrite(SwitchPinBatteryADC, HIGH);

  delay(100);

  adc2_get_raw(ADC2_CHANNEL_4, ADC_WIDTH_BIT_12, & raw);

  digitalWrite(SwitchPinBatteryADC, LOW);

  esp_adc_cal_characteristics_t characteristics;
  esp_adc_cal_characterize(ADC_UNIT_2, ADC_ATTEN_DB_11, ADC_WIDTH_BIT_12, ADCVRef, &
characteristics);

  voltage = (esp_adc_cal_raw_to_voltage(raw, & characteristics) / 1000.0F); // mV to V
  voltage *= 2.001906577F; // Real voltage divider ratio measured using multimeter

  Serial.printf("\nBattery voltage: %.2f V", voltage);

  if (voltage >= 3.6F) {
    voltage = constrain(voltage, 3.6, 4.2);
    voltage = map(voltage * 1000, 3600, 4200, 0, 100); // 3,6-4,2 to 0-100

    Serial.printf("\nBattery voltage: %.0f %%\n", voltage);
```

```
    payload.batteryLevel = voltage * 2.55; // 0-100 to 0-255
  } else {
    Serial.println("\nBattery level is critically low. Please charge up the battery and then restart the device.");
    esp_deep_sleep_start();
  }

  Serial.println("\nBattery level measurement has finished.");
}

void getBME280Values() {
  Serial.println("\nBME280 measurement has started.");

  digitalWrite(SwitchPinBME280, HIGH);

  delay(100);

  if (BME280.begin(0x76)) {
    float temperature = BME280.readTemperature();
    float humidity = BME280.readHumidity();
    float pressure = BME280.readPressure() / 100.0F;

    digitalWrite(SwitchPinBME280, LOW);

    Serial.printf("\nTemperature = %.2f B°C", temperature);
    Serial.printf("\nHumidity = %.2f %%", humidity);
    Serial.printf("\nPressure = %.2f hPa\n", pressure);

    payload.temperature = map(temperature * 256, -32768, 32768, 0, 65536); // -128-128 to 0-65536
    payload.humidity = humidity * 655.36; // 0-100 to 0-65536
    payload.pressure = pressure * 10; // 0-6553,6 to 0-65536
  } else {
    Serial.println("\nCould not find valid BME280 sensor. Check wiring.");

    digitalWrite(SwitchPinBME280, LOW);
  }

  Serial.println("\nBME280 measurement has finished.");
}

void getPositon() {
  uint32_t startMillis;

  Serial.println("\nTrying to obtain GPS data.");

  digitalWrite(SwitchPinQuectel, HIGH);
  delay(900);
  digitalWrite(SwitchPinQuectel, LOW);
  delay(3000);

  delay(3000);

  sendATCommand("AT+QGPS=1", "OK", 350, quectelSerial);

  startMillis = millis();

  while (millis() - startMillis < 60000) {
    if (sendATCommand("AT+QGPSLOC=0", "OK", 350, quectelSerial)) { // Checking if position is locked
      sendATCommand("AT+QGPSGNMEA=\"RMC\"", "OK", 350, quectelSerial); // NMEA RMC sentence

      for (uint16_t i = 0; i < sizeof(receivedDataBuffer); i++) {
        GPS.encode(receivedDataBuffer[i]); // Sentence encoding
```

```
    }

    sendATCommand("AT+QGPSGNMEA=\"GGA\"", "OK", 350, quectelSerial); // NMEA GGA sentence

    for (uint16_t i = 0; i < sizeof(receivedDataBuffer); i++) {
      GPS.encode(receivedDataBuffer[i]); // Sentence encoding
    }

    sendATCommand("AT+QGPSEND", "OK", 350, quectelSerial);

    readableTimestamp.Second = GPS.time.second();
    readableTimestamp.Hour = GPS.time.hour();
    readableTimestamp.Minute = GPS.time.minute();
    readableTimestamp.Day = GPS.date.day();
    readableTimestamp.Month = GPS.date.month();
    readableTimestamp.Year = GPS.date.year() - 1970; // Years since 1970

    unixTimestamp = makeTime(readableTimestamp);

    payload.timestamp = unixTimestamp;
    payload.latitude = GPS.location.lat() * 1000000;
    payload.longitude = GPS.location.lng() * 1000000;
    payload.altitude = GPS.altitude.meters() * 10000;
    payload.HDOP = round(GPS.hdop.hdop()); // Round to whole number

    Serial.printf("\nUNIX timestamp: %u", unixTimestamp);
    Serial.printf("\nLatitude: %f", GPS.location.lat());
    Serial.printf("\nLongitude: %f", GPS.location.lng());
    Serial.printf("\nAltitude: %.2f", GPS.altitude.meters());
    Serial.printf("\nHDOP: %.0f\n", round(GPS.hdop.hdop()));

    Serial.println("\nGPS data has been obtained.");

    return;
   }

   delay(2000);
  }

  sendATCommand("AT+QGPSEND", "OK", 350, quectelSerial);

  Serial.println("\nNo valid GPS data has been obtained.");

  payload.flags += 8;
}

bool sendDataQuectel() {
  uint32_t startMillis;

  Serial.println("\nTrying to send data using Quectel.");

  Serial.println("\nSearching for NB-IoT network.");

  startMillis = millis();

  while (millis() - startMillis < 60000) {
    if (sendATCommand("AT+CEREG?", "+CEREG: 0,5", 350, quectelSerial) || sendATCommand("AT+CEREG?",
"+CEREG: 0,1", 350, quectelSerial)) {

      Serial.println("\nConnected to NB-IoT network.");

      delay(1000);
```

```
    sendATCommand("AT+QIACT=1", "OK", 150000, quectelSerial); // Activate PDP (Packet Data Protocol) context

    delay(1000);

    char fullCommand[64];

    sprintf(fullCommand, "AT+QIOPEN=1,0,\"TCP\",\"%s\",%u,0,1", serverIP, port);

    if (sendATCommand(fullCommand, "+QIOPEN: 0,0", 150000, quectelSerial)) { // Open TCP session with the destination server
      sprintf(fullCommand, "AT+QISEND=0,%u", payload.len + (uint8_t) strlen(quectelIMEI) + 1); // +1 = ;

      sendATCommand(fullCommand, ">", 350, quectelSerial); // Send TCP data

      quectelSerial.write(quectelIMEI);
      quectelSerial.write(";");

      const uint8_t * payloadInt = (uint8_t * ) & payload;

      for (uint8_t i = 0; i < payload.len; i++) {
        quectelSerial.write(payloadInt[i]);
      }

      delay(350);

      char inChar;

      while (quectelSerial.available() > 0) {
        inChar = quectelSerial.read();
        Serial.print(inChar);
      }

      delay(1000);

      sprintf(fullCommand, "+QISEND: %u,%u,0", payload.len + (uint8_t) strlen(quectelIMEI) + 1, payload.len + (uint8_t) strlen(quectelIMEI) + 1);

      if (!sendATCommand("AT+QISEND=0,0", fullCommand, 90000, quectelSerial)) break; // Check for ACK from the server

      delay(1000);

      sendATCommand("AT+QICLOSE=0", "OK", 10500, quectelSerial); // Close TCP session

      sendATCommand("AT+QPOWD", "POWERED DOWN", 10500, quectelSerial);

      Serial.println("\nData has been successfully sent using Quectel.");

      return true;
    }
    break;
  }
  delay(2000);
}

sendATCommand("AT+QICLOSE=0", "OK", 10500, quectelSerial); // Close TCP session

sendATCommand("AT+QPOWD", "POWERED DOWN", 10500, quectelSerial);

Serial.println("\nQuectel was not able to send the data.");
```

```
  return false;
}

void sendDataIridium() {
  uint16_t crc;
  char inChar;
  uint8_t sendAttempt = 1;
  uint32_t startMillis;

  Serial.println("\nTrying to send data using Iridium.");

  digitalWrite(SwitchPinIridium, HIGH);

  delay(1000);

  char fullCommand[16];
  char command[] = "AT+SBDWB=";

  sprintf(fullCommand, "%s%u", command, payload.len);

  sendATCommand(fullCommand, "READY", 350, iridiumSerial); // Send data to Iridium outgoing buffer

  const uint8_t * payloadInt = (uint8_t * ) & payload;

  for (uint8_t i = 0; i < payload.len; i++) {
    iridiumSerial.write(payloadInt[i]);
    crc += (uint16_t) payloadInt[i];
  }

  iridiumSerial.write(crc >> 8);
  iridiumSerial.write(crc & 0xFF);

  delay(300);

  while (iridiumSerial.available() > 0) {
    inChar = iridiumSerial.read();
    Serial.print(inChar);
  }

  while (sendAttempt < 4) {
    Serial.printf("\nSending attempt: %u/3", sendAttempt);

    Serial.println("\nSearching for Iridium network.");

    startMillis = millis();

    while (millis() - startMillis < 60000) {
      Serial.print(".");

      if (digitalRead(IridiumNetwork) == HIGH) { // CSQ >= 1
        Serial.println("\nConnected to Iridium network.");
        if (sendATCommand("AT+SBDI", "+SBDI: 1", 60000, iridiumSerial)) { // SBD session

          sendATCommand("AT*F", "OK", 10500, iridiumSerial);
          digitalWrite(SwitchPinIridium, LOW);

          Serial.println("\nData has been successfully sent using Iridium.");
          return;
        } else {
          Serial.println("\nError occured when sending data.");
          break;
```

```
      }
     }
     delay(1000);
    }
    sendAttempt++;
  }

  sendATCommand("AT*F", "OK", 10500, iridiumSerial);
  digitalWrite(SwitchPinIridium, LOW);

  Serial.println("\nIridium was not able to send the data.");
}

void setup() {
  bool dataSent;

  pinMode(ShockSensor, INPUT);
  shockSensorState = digitalRead(ShockSensor);
  pinMode(MagnetSensor, INPUT);
  magnetSensorState = digitalRead(MagnetSensor);
  pinMode(WaterSensor, INPUT);
  waterSensorState = digitalRead(WaterSensor);
  pinMode(SwitchPinBatteryADC, OUTPUT);
  pinMode(SwitchPinBME280, OUTPUT);
  pinMode(SwitchPinIridium, OUTPUT);
  pinMode(IridiumNetwork, INPUT_PULLDOWN);
  pinMode(SwitchPinQuectel, OUTPUT);

  payload.batteryLevel = 0;
  payload.flags = 0;
  payload.timestamp = 0;
  payload.temperature = 0;
  payload.humidity = 0;
  payload.pressure = 0;
  payload.latitude = 0;
  payload.longitude = 0;
  payload.altitude = 0;
  payload.HDOP = 0;

  Serial.begin(115200);
  quectelSerial.begin(115200, SERIAL_8N1, QuectelMainRX, QuectelMainTX);
  iridiumSerial.begin(19200, SERIAL_8N1, IridiumRX, IridiumTX);

  if (!initSettings) initConfig();

  Serial.printf("\nBoot sequence number: %u\n", bootCount);

  getSensorValues();
  getBatteryLevel();
  getBME280Values();
  getPositon();

  dataSent = sendDataQuectel();
  if (!dataSent) sendDataIridium();

  if (shockSensorState == LOW && magnetSensorState == LOW && waterSensorState == LOW) {
    esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK, ESP_EXT1_WAKEUP_ANY_HIGH); // Sensor
trigger wakeup
  } else {
    esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK, ESP_EXT1_WAKEUP_ALL_LOW);
  }
  esp_sleep_enable_timer_wakeup(timeToSleep * 60 * 1000000);
```

```
  bootCount++;

  Serial.println("\nDevice is going to deepsleep mode.");
  esp_deep_sleep_start();
}

void loop() {} // not used
```

ДОДАТОК В

Демонстраційний матеріал

Формат	Позиція	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість	Примітки
			Текстові документи		
А4	1	ГЮІК 411710.06 ПЗ	Пояснювальна	99	с.
			записка		ф. А4
			Інші документи		
	2		Додаток А	1	с.
			Схема електрична		ф. А4
			підключення		
			компонентів приладу		
	3		Додаток Б	10	с.
			Код програми		ф. А4
	4		Додаток В	15	у форматі
			Демонстраційні		*.ppt
			матеріали		15 слайдів
			ГЮІК 411710.06 ВД		

Змін	Арк.	№ докум.	Підпис	Дата	Оптимізація логістичних процесів з використанням технологій Інтернету речей	Літе	Аркуш	Аркуш
Розробив		Півень Є. І.				Н	1	1
Перевірив		Чала О. О.						
Норм. контрол		Стародубцев Н.Г.				Кафедра КІТАР ХНУРЕ		
Затв.		Невлюдов І.Ш.						