Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Urządzenia peryferyjne

Akwizycja sygnałów

Wysyłanie i odbieranie danych kanałem BLE

Prowadzący:

dr inż. Jan Nikodem

Grupa:

Piątek 13:15 TP

Autorzy:

Daniel Glazer, 252743

Paweł Helisz, 252779

Wrocław 10.12.2021

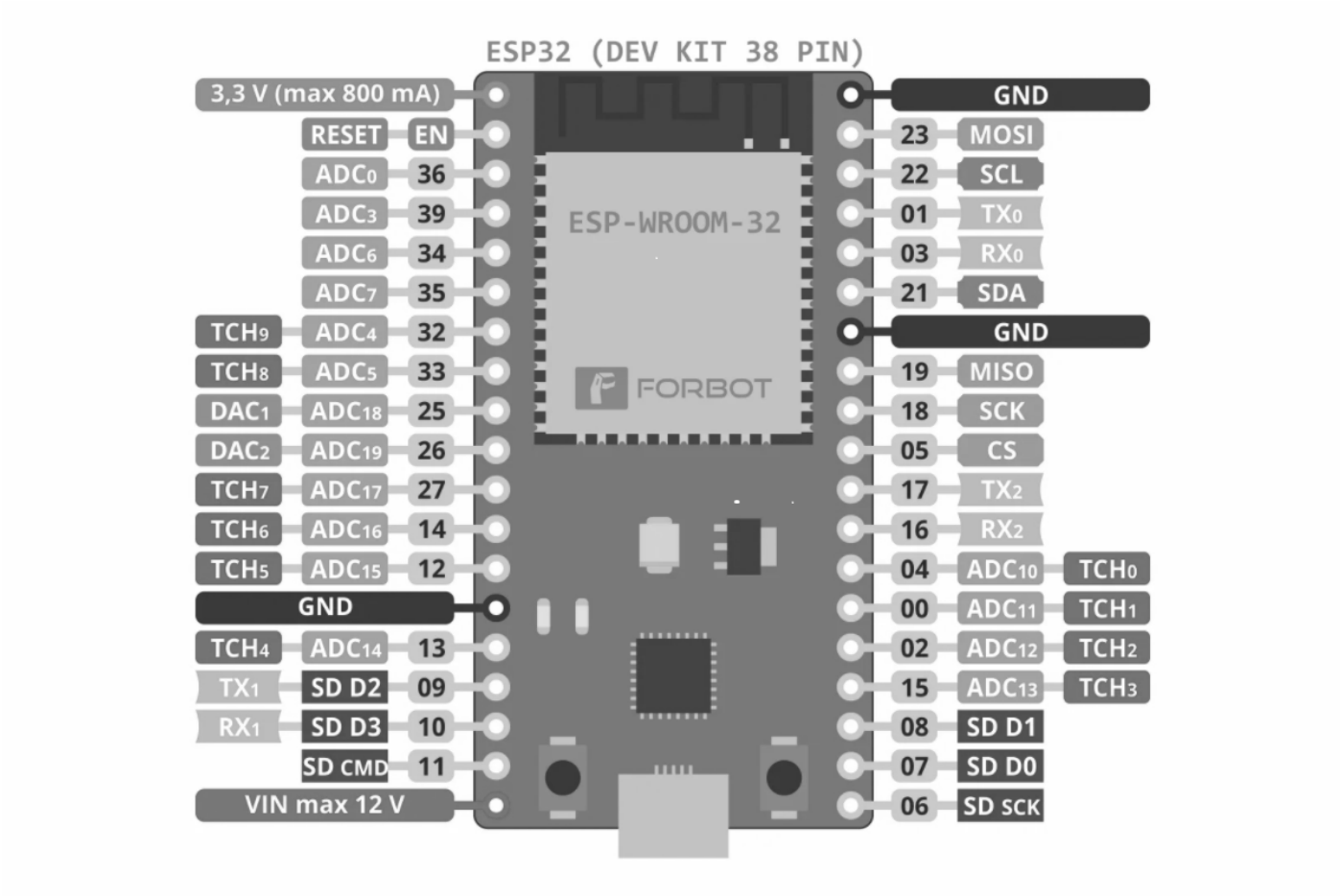
1 Cel ćwiczenia

Celem ćwiczenia była kontynuacja pracy z układem ESP32 DevKit. Należało w tym celu wykorzystać oprogramowanie kanału komunikacyjnego, które zostało wykonane na poprzednim laboratorium i rozszerzyć je o obsługę czujnika DHT11. Czujnik miał pobierać dane (temperatura, wilgotność), a następnie wysyłać je wcześniej utworzonym kanałem BLE do smartfona. Odebrane dane należało przedstawić na wykresie.

2 ESP32

ESP32 jest układem typu SoC (ang. System-on-a-chip) produkowanym przez chińską firmę Espressif Systems. Układ ten integruje w sobie niezbędne elementy do komunikacji przez WiFi (802.11 b/g/n) oraz przez Bluetooth (klasyczny 4.2 i BLE), dzięki czemu idealnie nadaje się do budowy energooszczędnych urządzeń Internetu Rzeczy (IoT).

2.1 Opis wyprowadzeń modułu ESP32 DevKit:



2.2 Specyfikacja płytki ESP32 DevKit

1. Komunikacja WiFi:
   (a) standard 802.11 b/g/n 2,4 GHz
   (b) prędkość transmisji do 150 Mb/s
   (c) zabezpieczenia WiFi: WEP, WPA/WPA2, PSK/Enterprise, AES / SHA2 / Elliptical Curve Cryptography / RSA-4096
2. Komunikacja Bluetooth:
   (a) BLE
3. Zasilanie:
   (a) napięcie pracy: 2,3 – 3,6 V
   (b) napięcie zasilania: 4,8 – 12 V
   (c) maksymalny pobór prądu: 800 mA
4. Aktualizacja oprogramowania:
   (a) UART
   (b) OTA
5. CPU:
   (a) Dual Core Tensilica LX6 240 MHz
   (b) obudowa: QFN48-pin (6 mm × 6 mm)
   (c) interfejsy: UART/SDIO/SPI/I2C/I2S,
   (d) dostępne 30 GPIO,
   (e) 12 kanałowy ADC,
   (f) 2 kanałowy DAC.
6. Konwerter USB-TTL (UART)
7. Raster wyprowadzeń: 2,54 mm
8. Wymiary modułu: 55 × 28 mm

3 Początkowy program Arduino

Początkowy program Arduino miał za zadanie wysyłać dwie liczby zmiennoprzecinkowe, które po podłączeniu modułu DHT11 miały być odczytanymi wartościami temperatury i wilgotności. W celu wysłania danych należało połączyć je w jeden ciąg znaków.

3.1 Struktura kodu

Pierwszym etapem było zawarcie bibliotek odpowiedzialnych za obsługę ciągów znakowych oraz *Bluetooth Low Energy* w układzie ESP32.

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

Następnie zadeklarowano UUID (ang. *Universally Unique Identifier*) dla serwera BLE, oraz dla typu połączenia (*indication*).

```
#define serviceID  "25AE1441-05D3-4C5B-8281-93D4E07420CF"
#define indicateID "25AE1444-05D3-4C5B-8281-93D4E07420CF"
```

Kolejnym krokiem było skonfigurowanie serwera BLE oraz typu połączenia.

```
 // Ustawienie nazwy
BLEDevice::init("Helisz_GLazer");

// Stworzenie serwera BLE
g_pServer = BLEDevice::createServer();
g_pServer->setCallbacks(new MyServerCallbacks());

// Utworzenie uslugi BLE
BLEService* pService = g_pServer->createService(serviceID);

// Scharakteryzowanie polaczenia jako indication
uint32_t propertyFlags = BLECharacteristic::PROPERTY_INDICATE;
BLECharacteristic* pCharIndicate =
        pService->createCharacteristic(indicateID, propertyFlags);

// Utworzenie descryptora
pCharIndicate->addDescriptor(new BLE2902());
pCharIndicate->setValue("");
g_pCharIndicate = pCharIndicate;

// Uruchomienie uslugi BLE
pService->start();
```

```
// Skonfigurowanie rozglaszania
BLEAdvertising* pAdvertising = BLEDevice::getAdvertising();
pAdvertising->addServiceUUID(serviceID);
pAdvertising->setScanResponse(true);

// Uruchomienie rozglaszania
 BLEDevice::startAdvertising();
```

Na sam koniec należało przekonwertować liczby zmiennoprzecinkowe na ciąg znakowy i rozdzielić je separatorem.

```
char buffY[10];
char buffX[10];
dtostrf(x, 2, 2, buffY);
dtostrf(y, 2, 2, buffX);

std::string wartosci = "";
wartosci += buffX;
wartosci += "/";
wartosci += buffY;
g_pCharIndicate->setValue(wartosci);
g_pCharIndicate->indicate();
```

4 Aplikacja na telefon

Do utworzenia aplikacji wykorzystano gotowe moduły (`https://github.com/alexanderlavrushko/BLEProof-collection`), które należało dostosować do założeń przedstawionych w instruckji laboratorium.

4.1 Kod i działanie

4.1.1 GraphView

Aplikacja miała wyświetlić odebrane dane na wykresie. W tym celu została wykorzystana biblioteka *GraphView* (`https://github.com/jjoe64/GraphView`). Umożliwia ona utworzenie wykresu wykorzystującego więcej niż jedną serię danych, a także skalowanie.

Aby wykorzystać bibliotekę *GraphView*, należało dodać zależność do pliku *build.gradle* w zakładce *dependencies*:

```
implementation 'com.jjoe64:graphview:4.2.2'
```

Następnym krokiem było dodanie do pliku *activity_main* komponentu wykresu:

```
<com.jjoe64.graphview.GraphView
    android:layout_width="match_parent"
    android:layout_height="200dip"
    android:id="@+id/graph" />
```

Skonfigurowanie i dodanie serii danych do wykresu:

```
    // konfiguracja wykresu
val graph = findViewById<View>(R.id.graph) as GraphView
    graph.viewport.isScalable = true
    graph.viewport.isScrollable = true
    graph.viewport.setScalableY(true)
    graph.viewport.setScrollableY(true)

    // utworzenie serii danych
    series = LineGraphSeries<DataPoint>()
    series1 = LineGraphSeries<DataPoint>()

    // dodanie serii danych do wykresu
    graph.addSeries(series)
    graph.addSeries(series1)
```

4.1.2 Szukanie i łączenie się aplikacji z układem ESP32

Aplikacja miała za zadanie łączyć się z układem ESP32. W tym celu należało ustawić UUID (taki sam jak na ESP32):

```
private const val SERVICE_UUID = "25AE1441-05D3-4C5B-8281-93D4E07420CF"
private const val CHAR_FOR_INDICATE_UUID = "25AE1444-05D3-4C5B-8281-93D4E07420CF"
```

Kolejnym krokiem było uruchomienie skanera wykrywającego zaprogramowany układ po naciśnięciu przełącznika:

```
switchConnect.setOnCheckedChangeListener { _, isChecked ->
    when (isChecked) {
        true -> {
            val filter = IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED)
            registerReceiver(bleOnOffListener, filter) }
         false -> {
           unregisterReceiver(bleOnOffListener) }
    }
    bleRestartLifecycle() }
```

Następnie uruchomiano skaner z filtrem UUID:

```
// Ustawienie filtra
val serviceFilter = scanFilter.serviceUuid?.uuid.toString()
isScanning = true
lifecycleState = BLELifecycleState.Scanning

// Uruchomienie skanera
bleScanner.startScan(mutableListOf(scanFilter), scanSettings, scanCallback)
```

Po udanym znalezieniu modułu ESP32 nawiązywane jest połączenie:

```
override fun onConnectionStateChange(gatt: BluetoothGatt, status: Int, newState: Int) {

            val deviceAddress = gatt.device.address

            if (status == BluetoothGatt.GATT_SUCCESS) {
                if (newState == BluetoothProfile.STATE_CONNECTED) {
                    appendLog("Connected to $deviceAddress")

                    Handler(Looper.getMainLooper()).post {
                        lifecycleState = BLELifecycleState.ConnectedDiscovering
                        gatt.discoverServices()
                    }
                }
            }
        }
```

4.1.3 Konwertowanie otrzymanych danych i umieszczenie ich na wykresie

Odebrany przez aplikację ciąg znaków należało rozdzielić, a następnie dokonać konwersji na typ zmiennoprzecinkowy.

```
textViewIndicateValue.text = strValue
var firstNumber = ""
var secondNumber = ""
var firstPart = true

for (i in strValue) {
    if (i == '/') {
        firstPart = false
        continue}
```

```
        if (firstPart)
            firstNumber += i
        else
            secondNumber += i }

    // konwertowanie danych na liczby
    temp = firstNumber.toDouble()
    humidity = secondNumber.toDouble()
```

Ostatnim krokiem było dodanie wartości temperatury i wilgotności do serii danych:

```
series?.appendData(DataPoint(lastX, temp), true, 100)
series1?.appendData(DataPoint(lastX++, humidity), true, 100)
```

5 Moduł DHT11

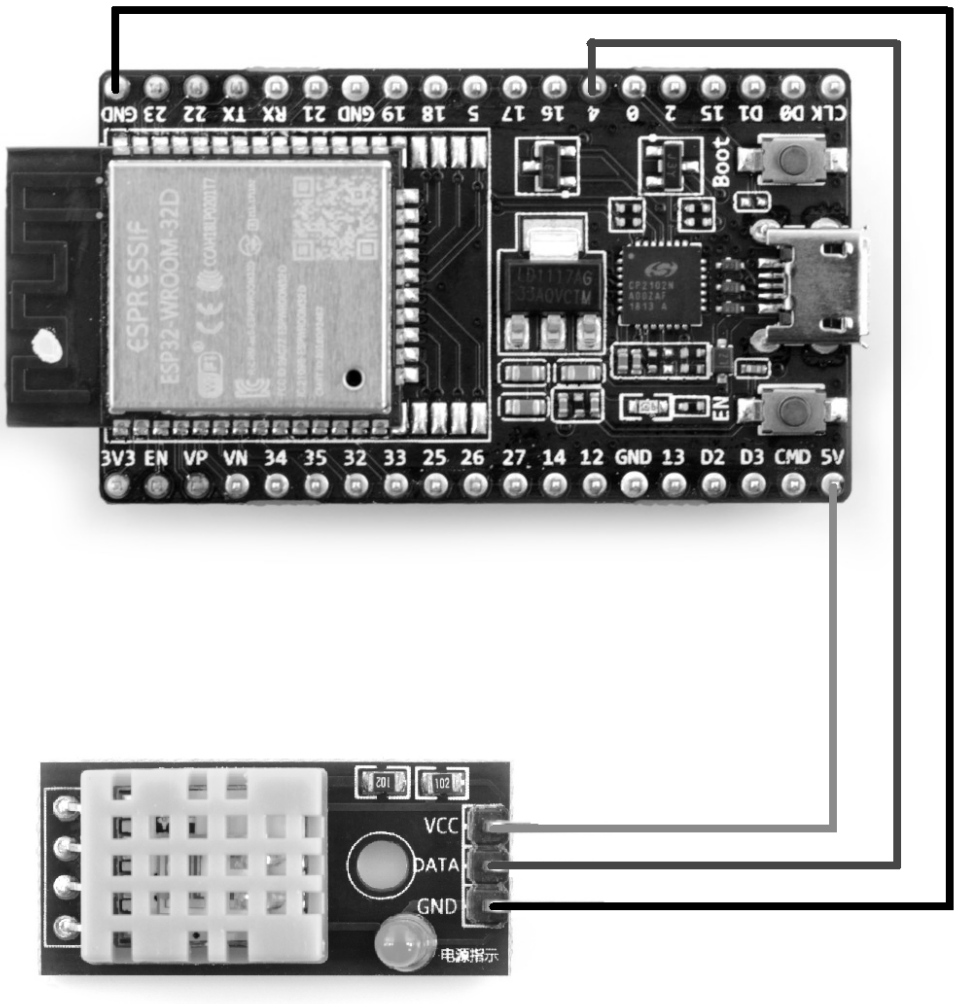
Moduł DHT11 zawiera czujnik umożliwiający pomiar temperatury i wilgotności powietrza. Na płytce znajdują się również niezbędne do poprawnego działania układu: rezystor oraz kondensator filtrujący napięcie zasilania. Czujnik można obsługiwać przy pomocy mikrokontrolera lub dowolnego zestawu uruchomieniowego np. Arduino. Wyprowadzeniami są złącza goldpin (raster 2,54 mm), umożliwiające podłączenie czujnika za pomocą przewodów lub bezpośrednie wpięcie w płytkę stykową.

5.1 Parametry:

1. Napięcie zasilania: 3 V do 5,5 V
2. Średni pobór prądu: 0,2 mA
3. Temperatura
   (a) Zakres pomiarowy: 0 - 50 °C
   (b) Dokładność: 1 °C
   (c) Czas odpowiedzi: 6 - 15 s (typowo 10 s)
4. Wilgotność:
   (a) Zakres pomiarowy: 20 - 90 %RH
   (b) Rozdzielczość: 8-bitów (±1 % RH)
   (c) Dokładność ±4 RH (przy 25 °C)
   (d) Zakres pomiarowy: 6 - 30 s

RH - Wilgotność względna wyrażana w procentach. Jest to stosunek rzeczywistej wilgoci w powietrzu do maksymalnej jej ilości, którą może utrzymać powietrze w danej temperaturze.

5.2 Schemat podłączenia:



1. VCC (+) - napięcie zasilania z zakresu: 3 V do 5,5 V
2. GND (-) - masa układu
3. DATA - wyjście cyfrowe podłączane do wejścia mikrokontrolera (pin 4)

5.3 Konfiguracja *Arduino IDE* pod moduł DHT11

W celu konfiguracji modułu DHT11 należało w programie Arduino IDE przejść do zakładki Sketch > Include Library > Manage Libraries, aby pobrać biblioteki: *DHT sensor library by Adafruit* oraz *Adafruit Unified Sensor*

5.4 Program Arduino wraz z modułem DHT11

Aby umożliwić obsługę modułu DHT11 w programie, należało dołączyć bibliotekę DHT.h.

```
#include "DHT.h"
#define DHTTYPE DHT11
```

Następnie należało zdefiniować port, do którego został podłączony czujnik na płytce ESP32:

```
uint8_t DHTPin = 4;
```

Kolejnym krokiem było zainicjowanie sensora DHT11:

```
DHT dht(DHTPin, DHTTYPE);
```

Po udanym zainicjowaniu należało uruchomić czujnik i przypisać zadeklarowanemu wyżej pinowi tryb pracy:

```
dht.begin();
pinMode(DHTPin, INPUT);
```

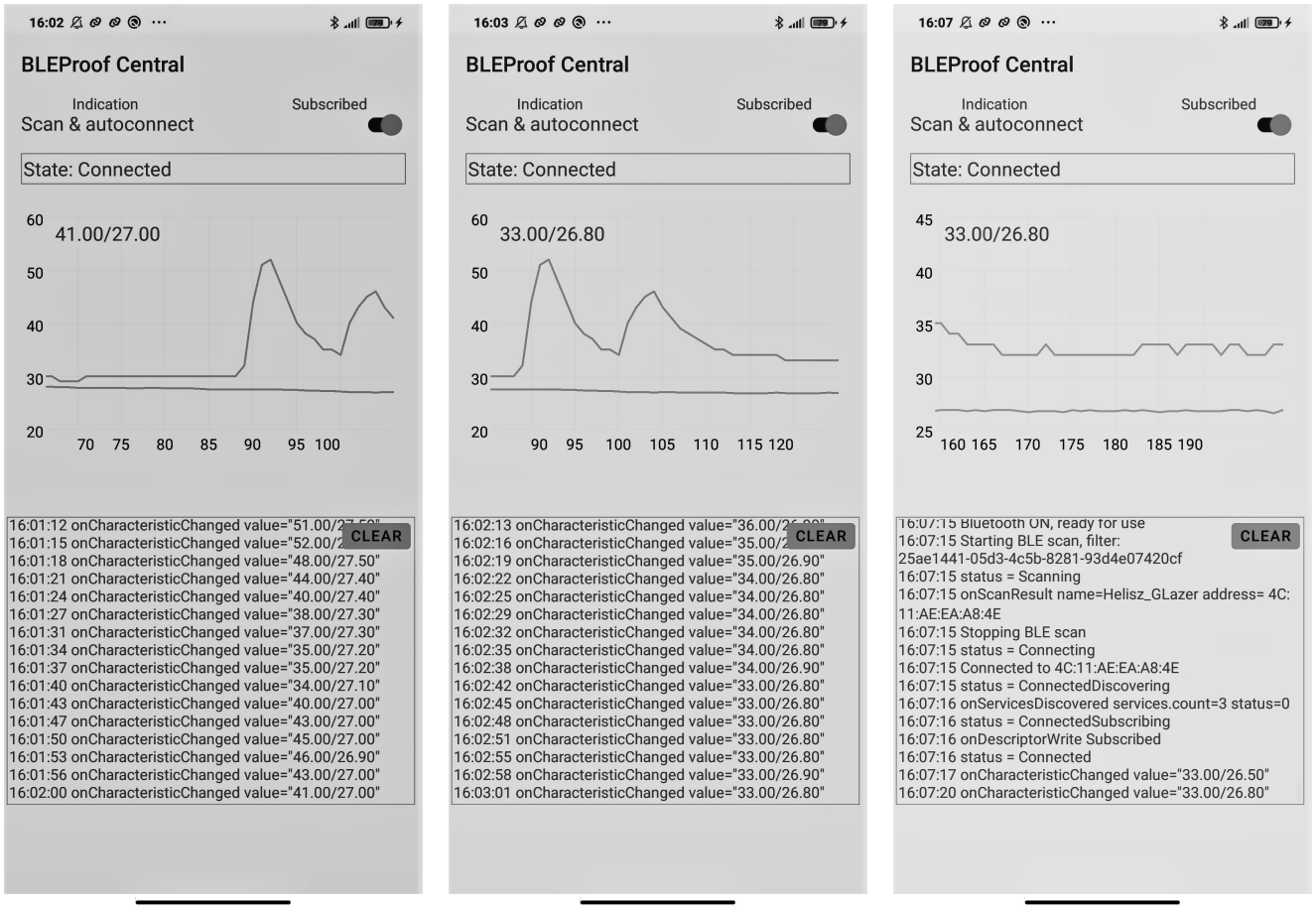
W ostatnim kroku należało dodać funkcje odpowiedzialne za odbieranie danych z czujnika i konwertowanie ich na ciąg znaków:

```
Temperature = dht.readTemperature();
Humidity = dht.readHumidity();
dtostrf(Temperature, 2, 2, buffY);
dtostrf(Humidity, 2, 2, buffX);
```

6 Obsługa aplikacji

Aby przesłać dane z czujnika do aplikacji na telefonie, należy na początku podłączyć według schematu czujnik DHT11 do płytki ESP32. Następnie należy zainstalować i uruchomić aplikację na telefonie. W kolejnym kroku wgrywamy program Arduino na ESP32. W aplikacji uruchamiamy skanowanie za pomocą przełącznika. Jeśli połączenie z ESP32 zostanie nawiązane, w polu tekstowym będzie widniał napis *State: Connected* i otrzymane dane będą umieszczone na wykresie.

7 Ostateczne wyniki



Rysunek 1: Rezultaty uzyskane po połączeniu aplikacji z układem