Stacja pogodowa w oparciu o system operacyjny czasu rzeczywistego Zephyr oraz płytę prototypową Arduino Nano 33 BLE

Mateusz Matkowski (145432) Mikołaj Starzak (158958)

24 stycznia 2024

1 Wprowadzenie

Projekt związany jest z wykorzystaniem systemu Zephyr do pomiaru i prezentacji danych z czujnika temperatury i ciśnienia. Aplikacja powinna działać w oparciu o system Zephyr oraz płytę prototypową Arduino Nano 33 BLE.

2 Wykorzystane technologie

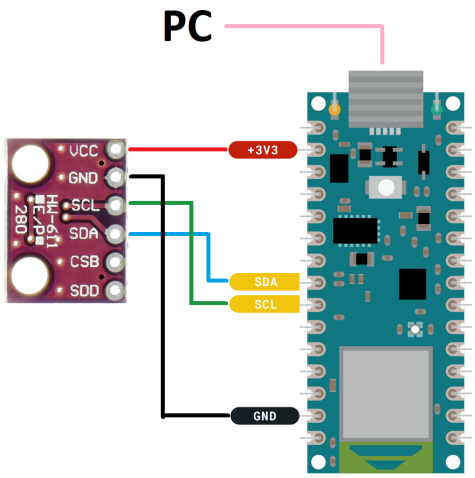
2.1 Sprzęt

- Arduino Nano 33 BLE,
- BMP280 - czujnik ciśnienia i temperatury,
- Telefon z Androidem.
- Komputer z USB.

2.2 Oprogramowanie i protokoły

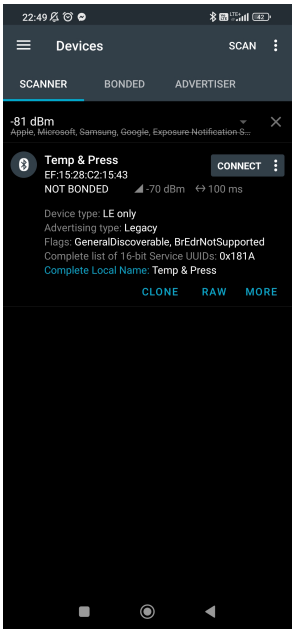
- Zephyr - system operacyjny czasu rzeczywistego,
- nRF Connect - mobilna aplikacja do zarządzania połączeniami Bluetooth na telefonach,
- I2C - protokół do komunikacji czujnika z Arduino,
- Bluetooth BLE - sieć bezprzewodowa.

3 Schemat

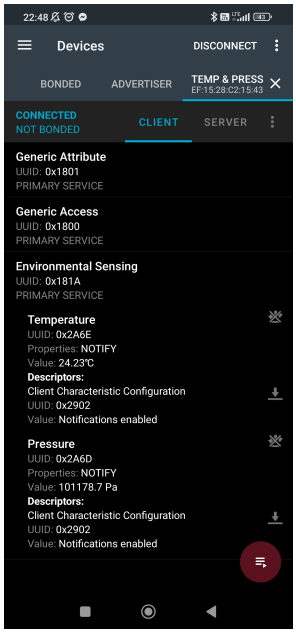


Rysunek 1: Schemat połączenia Arduino z czujnikiem BMP280 i komputerem.

4 Prezentacja działania



Rysunek 2: Reklamowany serwis "Temp & Press".



Rysunek 3: Serwis po nawiązaniu połączenia.

Po podłączeniu Arduino 33 Nano BLE do zasilania płytka zaczyna nadawać reklamę serwisu pogodowego. Używając aplikacji mobilnej nRF Connect, zyskujemy dostęp do wszystkich urządzeń używających Bluetooth w zasięgu naszego telefonu. Na Rys.2. widzimy serwis reklamowany przez nasze Arduino. Po połączeniu się z serwerem przyciskiem "Connect", dostajemy dostęp do wszystkich szczegółów serwisów. Na Rys.3., w zakładce "Environmental Sensing" znajdują się pola reprezentujące aktualną temperaturę i ciśnienie atmosferyczne. Co każde 5 sekund Arduino wysyła pomiar z czujnika na połączony telefon.

5 Fragmenty kodu

```
57  static const struct bt_data ad[] = {
58      BT_DATA_BYTES(BT_DATA_FLAGS, (BT_LE_AD_GENERAL | BT_LE_AD_NO_BREDR)),
59      BT_DATA_BYTES(BT_DATA_UUID16_ALL, BT_UUID_16_ENCODE(BT_UUID_ESS_VAL))
60  };
```

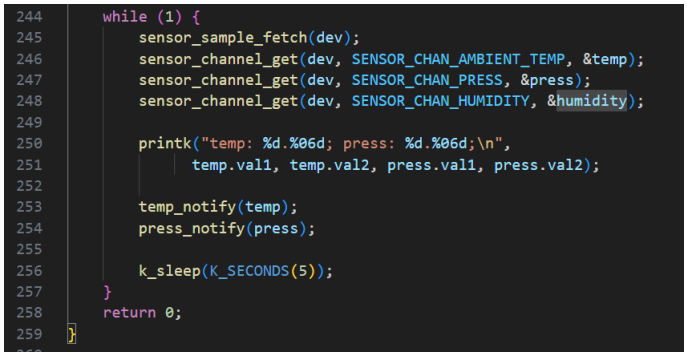
Rysunek 4: Definicja reklamy dla serwisu.

```
114 BT_GATT_SERVICE_DEFINE(ess_srv,
115     BT_GATT_PRIMARY_SERVICE(BT_UUID_ESS),
116     BT_GATT_CHARACTERISTIC(BT_UUID_TEMPERATURE, BT_GATT_CHRC_NOTIFY, NULL, NULL, NULL, NULL),
117     BT_GATT_CCC(tempmc_ccc_cfg_changed, TEMPS_GATT_PERM_DEFAULT),
118     BT_GATT_CHARACTERISTIC(BT_UUID_PRESSURE, BT_GATT_CHRC_NOTIFY, NULL, NULL, NULL, NULL),
119     BT_GATT_CCC(pressmc_ccc_cfg_changed, PRESSS_GATT_PERM_DEFAULT),
120 );
```

Rysunek 5: Definicja serwisu.

```
223 int main(void){
224     int err;
225 
226     err = bt_enable(NULL);
227     if (err) {
228         printk("Bluetooth init failed (err %d)\n", err);
229         return 0;
230     }
231 
232     bt_ready();
233     bt_conn_auth_cb_register(&auth_cb_display);
234 
235     bt_temp_cb_register(&temp_cb);
236     bt_press_cb_register(&press_cb);
237 
238     const struct device *dev = get_bme280_device();
239     if (dev == NULL) {
240         return 0;
241     }
242     struct sensor_value temp, press, humidity;
243 
```

Rysunek 6: Inicjalizacja Bluetooth BLE i czujnika BMP280.



```
244     while (1) {
245         sensor_sample_fetch(dev);
246         sensor_channel_get(dev, SENSOR_CHAN_AMBIENT_TEMP, &temp);
247         sensor_channel_get(dev, SENSOR_CHAN_PRESS, &press);
248         sensor_channel_get(dev, SENSOR_CHAN_HUMIDITY, &humidity);
249 
250         printk("temp: %d.%06d; press: %d.%06d;\n",
251                temp.val1, temp.val2, press.val1, press.val2);
252 
253         temp_notify(temp);
254         press_notify(press);
255 
256         k_sleep(K_SECONDS(5));
257     }
258     return 0;
259 }
```

Rysunek 7: Główna pętla.

6 Kompilacja i flash'owanie

6.1 Budowanie projektu

west build -p always -b arduino_nano_33_ble {ścieżka do projektu}
-DDTC_OVERLAY_FILE="arduino_i2c.overlay"

6.2 Flash'owanie Arduino

west flash –bossac={ścieżka do bossac.exe} –bossac-port="COM3"