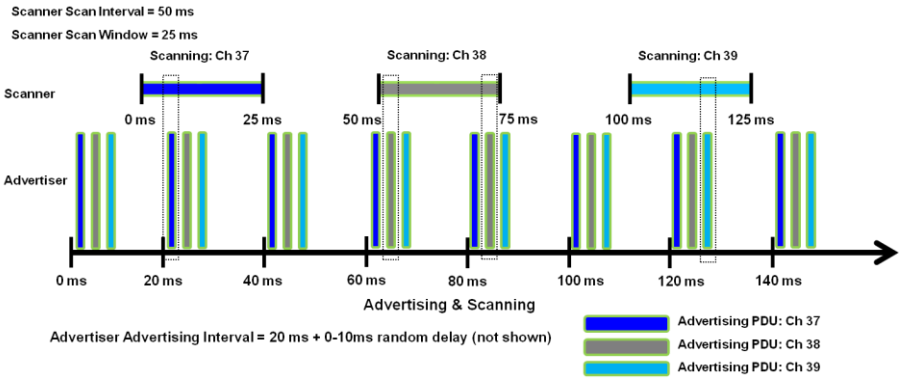
5 BLE klijent

U ovoj vežbi biće predstavljen postupak programiranja BLE klijent aplikacije. BLE klijent se, po pravilu, implementira u okviru uređaja tipa BLE centrala, za razliku od BLE servera koji radi na strani BLE periferije. Nakon što centrala otkrije prisustvo periferije (oglašavanje i skeniranje) i uspostavi konekciju sa periferijom, BLE klijent centrale se povezuje sa BLE serverom periferije. Na ovaj način, korisničkoj aplikaciji centrale je omogućeno da koristi servise periferije, odnosno pristupa karakteristikama servisa periferije radi upisa i čitanja i prima njihove notifikacije. U pripremnom delu ove vežbe, prvo će biti objašnjene procedure za oglašavanje i skeniranje, a zatim i sam postupak programiranja BLE klijenta.

5.1 Oglašavanje i skeniranje

Kao što već znamo, BLE server periodično emituje pakete za oglašavanje na kanalima za oglašavanje. Perioda slanja ovih paketa, tj. interval oglašavanja se reguliše na strani BLE servera i može se podešavati u opsegu od $20\ ms$ do $10.24\ s$. Na svaki interval oglašavanja, BLE server uključuje svoj RF predajnik i šalje uzastopno tri identična paketa za oglašavanje, po jedan na svakom od tri kanala za oglašavanje. Paketi za oglašavanje su kratki i sadrže samo elementarne podatke o BLE serveru. Slanjem paketa za oglašavanje, BLE server zapravo otkriva svoje prisustvo okolnim uređajima (BLE klijentima), koji, nakon toga mogu da iniciraju uspostavljanje konekcije.

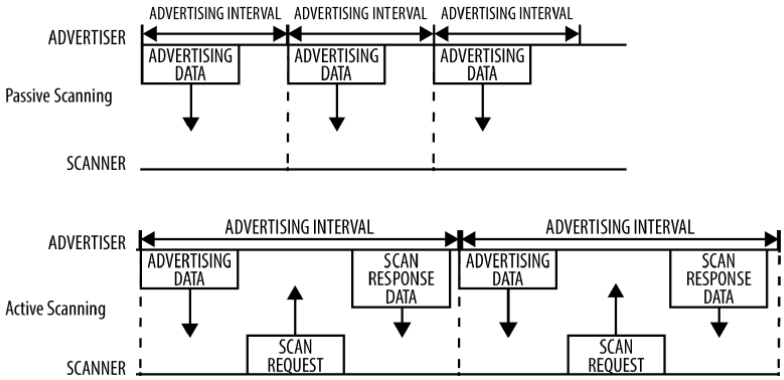
S druge strane, BLE klijent, koji je u ulozi skenera, skenira kanale za oglašavanje, prima pakete za oglašavanje i na osnovu njihovog sadržaja odlučuje da li će pokrenuti proceduru za uspostavljanje konekcije sa BLE serverom. BLE klijent ne osluškuje neprekidno kanale za oglašavanje, već se tajming rada skenera reguliše pomoću dva konfigurabilna parametra: *prozor skeniranja* (*Scan Window*), koji definiše trajanje skeniranja i *interval skeniranja* (*Scan Interval*), koji definiše koliko često se skeniranje ponavlja. U svakom sledećem prozoru skeniranja, skener osluškuje različiti kanal za oglašavanje, kao što je to ilustrovano na Sl. 1.



Sl. 1 Oglašavanje i skeniranje

BLE specifikacija pravi razliku između dva tipa skeniranja (Sl. 2):

- *Pasivno skeniranja* – BLE klijent (tj. skener), jednostavno, osluškuje kanale za oglašavanje u očekivanju paketa za oglašavanje. Pri tom, BLE server (tj. oglašivač, ili *advertiser*) nema povratnu informaciju o tome da li je i kada neki skener primio emitovan paket.
- *Aktivno skeniranje* – nakon prijema paketa za oglašavanje, BLE klijent šalje BLE serveru paket tipa *Scan Request*. Kao odgovor na prijem *Scan Request* paketa, server šalje klijentu paket tipa *Scan Response* sa dodatnim podacima o oglašavanju. U *Scan Response* paketu može da stane do 254 bajta podataka, što značajno povećava količinu podataka koja se oglašavanjem može preneti BLE klijentu.



Sl. 2 Pasivno i aktivno skeniranje

U listingu L.1 je dati jednostavan program BLE skener. Program inicira proceduru skeniranja i štampa podatke preuzete iz pristiglih paketa za oglašavanje. U liniji 19, BLE uređaj je inicijalizovan. U liniji 20, u okviru BLE uređaja, kreiran je BLE skener. U linijama 21-24, BLE skener je podešen: povezan je sa odgovarajućom *callback* funkcijom (linija 21), uključeno je aktivno skeniranje (linija 22) i podešeno je trajanje intervala i prozora skeniranja (linije 23-24). Prozor skeniranja, naravno, mora biti kraći od intervala skeniranja.

Skeniranje se startuje u funkciji *loop*(), linija 28, pozivom funkcije *start*() BLE skenera. Ova funkcija ima dva parametra: prvi definiše vreme skeniranja u sekundama (*scanTime*), dok je dugi indikacija da li se nastavlja prethodno skeniranje (*true*) ili skener započinje novo (*false*). Funkcija *start*() je blokirajuća, što znači da program ostaje u ovoj funkciji sve do isteka vremena *scanTime*. U liniji 29, pozivom funkcije *clearResults*() BLE skenera, briše se rezultat prethodnog skeniranje, odnosno, brišu se svi podaci koje je BLE skener prikupio tokom upravo završenog ciklusa skeniranja. Na kraju funkcije *loop*() je uvedeno kašnjenje od $2s$. Dakle, s obzirom na liniju 5, BLE skener je aktivan $5s$, zatim se pravi pauza od $2s$, a onda se skeniranje ponovo uključuje.

Uvek kada BLE skener primi paket za oglašavanje, izvršava se *callback* funkcija *onResult*() – linija 10. Argument ove funkcije je objekat tipa *BLEAdvertisedDevice* koji sadrži podatke ekstrahovane iz primljenog paketa za oglašavanje. Jedina aktivnost u funkciji *onResult*() je štampanje ekstrahovanih podataka preko serijskog monitora.

Napomena 1: Članove klase *BLEAdvertisedDevice* možemo videti desnim klikom mišem preko imena *BLEAdvertisedDevice* (linija 10), a zatim izborom „*Go to Definition*“ iz padajućeg menija. Otvara se nova kartica u kojoj je prikazan sadržaj heder-fajla ove klase. Videćemo brojne funkcije članice od kojih svaka omogućava pristup jednom tipu podatka uzetog iz paketa za oglašavanje.

Napomena 2: Fukcija *toString*() u naredbi *advertisedDevice.toString*().*c_str*() prikuplja sve podatke iz objekta *advertisedDevice* (sadrži podatke iz paketa za oglašavanje) i od njih pravi jedan dugačak string tipa *String*. Efekat *c_str*() je konvertovanje String-a u string koji se može štampati naredbom *printf* (*c_str*() smo videli i u nekoj od prethodnih vežbi).

L. 1. BLEScan

```
1   #include <BLEDevice.h>
2   #include <BLEUtils.h>
3   #include <BLEScan.h>
4   #include <BLEAdvertisedDevice.h>
5
6   int scanTime = 5; //U sekundama
7   BLEScan* pBLEScan;
8
9   class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
10      void onResult(BLEAdvertisedDevice advertisedDevice) {
11         Serial.printf("Advertised Device:%s\n",
12                           advertisedDevice.toString().c_str());
13      }
14  };
15
16  void setup() {
17    Serial.begin(115200);
18
19    BLEDevice::init("BLE scenner");
20    pBLEScan = BLEDevice::getScan();//Kreiranje skenera
21    pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
22    pBLEScan->setActiveScan(true);  //Aktivno skeniranje
23    pBLEScan->setInterval(1349);
24    pBLEScan->setWindow(449);       //Mora biti manje ili jednako setInterval
25  }
26
27  void loop() {
28    pBLEScan->start(scanTime, false);
29    pBLEScan->clearResults(); //brise rezultat prethodnog skeniranja
30    delay(2000);
31  }
32
```

Na Sl. 3 vidimo izlaz programa *BLEScan*. Za svaki pronađeni BLE uređaj, ispisana je jedna linija na konzoli serijskog monitora. Sadržaj paketa za oglašavanje zavisi od toga koje podatke je BLE server uvrstio u ove pakete. Na početku linije je ime uređaja. Uočavamo „ESP32 BLE Server 2“ (BLE server iz vežbe 4) i uređaj sa imenom *Thingy*. Ostali uređaji nisu uvrstili svoje ime u paket za oglašavanje, pa je za njih taj podatak izostavljen. MAC adresa BLE uređaja je obavezan podatak u svakom paketu za oglašavanje. Pojedini BLE uređaji u paket za oglašavanje uvršćuju UUID GATT servisa koji implementiraju, dok kod drugih taj podatak ne postoji. „*Manufacturer data*“ se odnosi na podatke proizvođača, odnosno nestandardne podatke, čije značenje može da protumači samo BLE klijent koji „zna“ kako da radi sa tim konkretnim BLE uređajem. „*rssi*“ nosi podatak o jačini prijemnog signala. Ovaj podatak ne potiče od BLE servera, već je dobijen od RF primopredajnika BLE skenera.

```
Advertised Device: Name: ESP32 BLE Server 2, Address: 80:65:99:a2:48:2d, serviceUUID: 030ff29f-0348-4e87-9550-ccf1cfdf615d, txPower: 9, rssi: -39
Advertised Device: Name: , Address: 17:40:1f:17:b6:b9, manufacturer data: 060001092002530456bf953ea8aa0d39528237be7d6b9f3efcb0771095, rssi: -49
Advertised Device: Name: , Address: 58:7e:61:ce:d8:fc, manufacturer data: 0d00455246326a3336414456, serviceUUID: 00001812-0000-1000-8000-00805f9b34fb, rssi: -89
Advertised Device: Name: , Address: 47:b3:72:b6:c5:5c, manufacturer data: 75001822, rssi: -77
Advertised Device: Name: Thingy, Address: fb:d8:a3:66:31:e3, manufacturer data: 5900242fa64b, serviceUUID: ef680100-9b35-4933-9b10-52ffa9740042, rssi: -50
Advertised Device: Name: , Address: 7c:c2:94:3d:1d:fa, manufacturer data: 8f030a109b0d00f81d3d94c27c81, rssi: -58
Advertised Device: Name: , Address: 59:fd:9d:6f:0c:10, serviceUUID: 0000fef3-0000-1000-8000-00805f9b34fb, rssi: -69, serviceData: J□#GSMF□2□□:
Advertised Device: Name: , Address: fb:d8:a3:66:31:e5, serviceUUID: 0000feaa-0000-1000-8000-00805f9b34fb, rssi: -50, serviceData: □□□goo.gl/pIWdir
Advertised Device: Name: , Address: 4c:24:ce:13:26:ca, manufacturer data: 8f030a10420c01c92613ce244cec, rssi: -90
```

Sl. 3 Konzola serijskog monitora tokom izvršenja programa BLEScan

5.2 ESP32-S3 BLE klijent

U ovom primeru biće objašnjen postupak programiranja BLE klijent aplikacije. Na BLE uređaju koji projektujemo biće pokrenut BLE klijent koji uspostavlja konekciju sa BLE serverom 2 iz vežbe 4. BLE Server 2

implementira jedan servis sa dve karakteristike, gde prva sadrži podatak o periodi blinkanja LED, a druga podatak o broju pritisaka na taster. Za više detalja o BLE Server 2 pogledati u vežbi 4. Preko uspostavljene BLE konekcije, BLE klijent treba da prima notifikacije o pritiscima na taster BLE servera 2 i da podatak o broju pritisaka štampa preko serijskog monitora. Takođe, BLE klijent poseduje dva tastera. Efekat pritiska na jedan od tih tastera je inkrementiranje periode blinkanje LED na strani BLE servera 2 za $100ms$, a pritiska na drugi taster dekrementiranje za $100ms$. Program BLE klijenta je dat u listingu L.2.

Nekoliko opštih napomena: Zadatak BLE klijenta je uspostavljanje veze sa odgovarajućim servisima na strani BLE servera. Ono što je bitno BLE klijentu je to koje servise implementira BLE server, a ne BLE server kao takav. BLE klijent koji projektujemo u ovom primeru očekuje da je na strani BLE servera implementiran servis za upravljanje LED i prebrojavanje pritisaka na taster. Ovaj servis ima svoj UUID, koji glasi: 030ff29f-0348-4e87-9550-ccf1cfdf615d (videti u vežbi 4). Dakle, na početku, BLE klijent skenira kanale za oglašavanje i prima pakete za oglašavanje od različitih BLE uređaja, čekajući da primi paket koji će sadržati očekivani UUID servisa. Kada primi takav paket, BLE klijent prekida skeniranje i uspostavlja konekciju sa BLE serverom koji je poslao taj paket. Zatim, BLE klijent pronalazi na udaljenom servisu reference na karakteristike koju su mu bitne i ako je potrebno, dozvoljava njihove notifikacije. Nakon toga, BLE klijent je u mogućnosti da pristupa karakteristikama udaljenog servisa radi upisa i čitanja i da prima notifikacije od BLE servera. Kada se konekcija raskine, BLE klijent se vraća u režim skeniranja. Dakle, kao projektanti BLE klijenta, mi moramo biti upoznati sa specifikacijom BLE servisa s kojim naš klijent treba da se poveže. To uključuje UUID servisa i UUID-ove njegovih karakteristika. Takođe, potreno je poznavati prava pristupa karakteristikama i prirodu podataka koje one sadrže.

U linijama 7, 9 i 10, deklarisana su tri UUID-a: UUID servisa i UUID-ove dve karakteristike. Ovi UUID-ovi su identičnim onim koji su korišćeni prilikom programiranja BLE servera 2 u vežbi 4. Vidimo da za predstavljanje UUID postoji poseban tip podataka – BLEUUID. U linijama 12-15 deklarisani su globalni pointeri za pristup BLE uređaju s kojim se konektujemo (*serverDevice*), skeneru (*pBLEScan*) i dvema karakteristikama udaljenog servisa, *pRemoteChar1* i *pRemoteChar2*. Nakon toga slede deklaracije promenljivih za indikaciju statusa BLE konekcije, promenljivih za rad sa tasterima, kao i još nekoliko pomoćnih promenljivih.

U funkciji *setup*(), inicijalizovan je i poreknut BLE stek (linija 112), a zatim je podešen BLE skener, na isti način kao u prethodnom primeru. Pogledajmo sada funkciju *loop*(). U ovoj funkciji, na osnovu vrednosti promenljive *connected* razdvojena su dva režima rada: režim skeniranja (*connected = false*) i režim uspostavljene konekcije (*connected = true*). U režimu skeniranja, periodično se uključuje BLE skener u trajanju *scanTime*=5*s*, sve dok promenljiva *deviceFound* ne dobije vrednost *true*, što označava da je pronađen uređaj koji implementira očekivani servis. Kao što je to objašnjeno u prethodnom primeru, uvek kada BLE skener primi paket za oglašavanje izvršava se odgovarajuća *callback* funkcija – *onResult*(), linije 32-43. U ovoj funkciji se proverava da li paket za oglašavanje uopšte sadrži UUID servisa (*haveServiceUUID*) i ako sadrži, da li taj UUID odgovara UUID-u servisa koji očekujemo – pozivom funkcije *isAdvertisingService*(*serviceUUID*). Ako je ogovarajući servis pronađen, promenljiva *deviceFound* se postavlja na *true*, zaustavlja se skeniranje (linija 37) i u promenljivoj *serverDevice* se čuva pointer (referenca) na uređaj koji je poslao paket za oglašavanje (linija 38). Vratimo se ponovo na funkciju *loop*(). Nakon što je pronađen uređaj koji sadrži traženi servis, izlazi se iz *while* petlje i poziva funkcija *connectToServer*(), koja otvara konekciju sa uređajem *serverDevice* i uspostavlja veze sa karakteristikama servisa. Ako je sve prošlo kako treba, funkcija *connectToServer*() će postaviti promenljivu *connected* na *true*, što znači da prelazimo u režim aktivne konekcije.

U režimu aktivne konekcije, linije 131-171, BLE klijent interaguje sa BLE serverom, na način koji je specifičan za konkretnu aplikaciju. Odmah nakon što je konekcija otvorena, BLE klijent čita karakteristiku za brojač pritisaka na taster, karakteristika 2, linije 132-139, i njenu vrednost štampa preko serijskog monitora. Karakteristici se pristupa preko njenog pointera (*pRemoteChar2*), koji je inicijalizovan u funkciji

connectToServer(). Prvo se proverava da li dozvoljeno čitanje vrednosti ove karakteristike, funkcija *canRead*() u liniji 133. Mi znamo da je po specifikaciji servisa ovu karakteristiku dozvoljeno čitati. Međutim prava pristupa na strani servera nisu fiksna, već serverska aplikacija može, ako to želi, da ih naknadno promeni. Ako je čitanje dozvoljeno, vrednost karakteristike se čita funkcijom *readUInt16*() u liniji 134 preko pointera na karakteristiku 2 (*pRemoteChar2*). Mi znamo da je vrednost karakteristike za brojač pritisaka na taster tipa *uint16_t*, i zato za čitanje koristimo funkciju *readUInt16*(). Osim ove, postoje i funkcije za čitanje podataka još nekoliko tipova (desni klik preko *readUInt16* i onda "*Go To Definition*"). Odmah nakon čitanje karakteristike 2, promenljiva *onConnected* se postavlja na *false* (linija 138), što znači da će se u nastavku konekcije, izvršavati „*else*" deo programa, linije 140-170.

L. 2. BLEClient

```
1    #include "BLEDevice.h"
2
3    const int BTN_UP_PIN = 5;  //pin tastera za inkrement
4    const int BTN_DOWN_PIN = 6;//pin tastera za dekrement
5
6    //UUID servis s kojim zelimo da se povezemo
7    BLEUUID serviceUUID("030ff29f-0348-4e87-9550-ccf1cfdf615d");
8    //UUID-ovi karakteritika za koje smo zainteresovani
9    BLEUUID char1UUID("db79d281-1563-4454-9daf-5351ef61d3d7");
10   BLEUUID char2UUID("e8942db4-98ac-4b9c-8bc9-3c6b8843ed42");
11
12   BLEAdvertisedDevice* serverDevice; //pointer na BLE uredjaj za konektovanje
13   BLEScan* pBLEScan; //pointer na BLE skener
14   BLERemoteCharacteristic* pRemoteChar1; //pointer na karakt. za LED period
15   BLERemoteCharacteristic* pRemoteChar2; //pointer na karakteristiku za taster
16
17   bool connected = false;   //indikacija aktivne konekcije
18   bool onConnected = false; //indikacija upravo otvorene konekcije
19   bool deviceFound = false; //indikacija da je pronadjen BLE Server 2
20
21   int scanTime = 5; //vreme skeniranja (u sekundama)
22   uint16_t remoteLEDPeriod = 100; //perioda blinkanja LED na BLE Server 2
23   int btnUpState = LOW;       //stanje tastera za inkrement periode blinkanja
24   int btnDownState = LOW;
25   int lastBtnUpState = LOW;   //stanje tastera za dekrement periode blinkanja
26   int lastBtnDownState = LOW;
27   bool remoteLEDPeriodChanged = false; //indikacija promene periode blinkanja
28   uint16_t remoteBtnCntr = 0; //vrednost brojaca pritisaka na taster
29
30   //callback funkcija BLE skenera
31   class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
32     void onResult(BLEAdvertisedDevice advertisedDevice) {
33       Serial.print("BLE Advertised Device found: ");
34       Serial.println(advertisedDevice.toString().c_str());
35       if(advertisedDevice.haveServiceUUID()&&
36                     advertisedDevice.isAdvertisingService(serviceUUID)) {
37         pBLEScan->stop();
38         serverDevice = new BLEAdvertisedDevice(advertisedDevice);
39         deviceFound = true;
40       }
41     }
42   };
43
44   //callback funkcije BLE klijenta
45   class MyClientCallback : public BLEClientCallbacks {
46     void onConnect(BLEClient* pclient) {  }
47     void onDisconnect(BLEClient* pclient) {
48       connected = false;
```

```
49      Serial.println("Disconnected");
50    }
51  };
52
53  //callback funkcija notifikacije karakteristike 2
54  void notifyCallback(BLERemoteCharacteristic* pBLERemoteCharacteristic,
55                      uint8_t* pData,size_t length, bool isNotify) {
56      remoteBtnCntr = (pData[1] << 8) + pData[0];
57      Serial.print("Button counter: ");
58      Serial.println(remoteBtnCntr);
59  }
60
61  //povezivanje sa BLE serverom
62  void connectToServer(){
63      connected = false;
64      Serial.print("Povezivanje sa: ");
65      Serial.println(serverDevice->getAddress().toString().c_str());
66      BLEClient* pClient = BLEDevice::createClient();
67      pClient->setClientCallbacks(new MyClientCallback());
68      //Povezivanje sa BLE Serverom.
69      pClient->connect(serverDevice);
70      pClient->setMTU(517); //podesavanje maksimalne duzine paketa(inace je 23)
71
72      //Pribavljanje reference na servis
73      BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
74      if (pRemoteService == nullptr) {
75        Serial.print("Servis nije pronadjen UUID: ");
76        Serial.println(serviceUUID.toString().c_str());
77        pClient->disconnect();
78        return;
79      }
80      Serial.println("Servis je pronadjen");
81      //Pribavljanje reference na karakteristiku 1 (LED period).
82      pRemoteChar1 = pRemoteService->getCharacteristic(char1UUID);
83      if (pRemoteChar1 == nullptr) {
84        Serial.print("Karakteristika 1 nije pronadjena UUID: ");
85        Serial.println(char1UUID.toString().c_str());
86        pClient->disconnect();
87        return;
88      }
89      Serial.println("Karakteristika 1 je pronadjena");
90      //Pribavljanje reference na karakteristiku 2 (brojac pritisaka na taster)
91      pRemoteChar2 = pRemoteService->getCharacteristic(char2UUID);
92      if (pRemoteChar2 == nullptr) {
93        Serial.print("Karakteristika 2 nije pronadjena UUID: ");
94        Serial.println(char2UUID.toString().c_str());
95        pClient->disconnect();
96        return;
97      }
98      Serial.println("Karakteristika 2 je pronadjena");
99      //Dozvola notifikacija karakteristike 2
100     if(pRemoteChar2->canNotify())
101       pRemoteChar2->registerForNotify(notifyCallback);
102
103     connected = true;
104     onConnected = true;
105     return;
106 }
107
108 void setup() {
109  pinMode(BTN_UP_PIN, INPUT_PULLDOWN);
110  pinMode(BTN_DOWN_PIN, INPUT_PULLDOWN);
111  Serial.begin(115200);
```

```
 BLEDevice::init("ESP32 BLE Client");
 //podesavanje BLE skenera
 pBLEScan = BLEDevice::getScan();
 pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
 pBLEScan->setInterval(1349);
 pBLEScan->setWindow(449);
 pBLEScan->setActiveScan(true);
}

void loop() {
  if(!connected){
    deviceFound = false;
    while(!deviceFound){
      delay(1000);
      Serial.println("Scanning...");
      pBLEScan->start(scanTime, false);
      pBLEScan->clearResults();
    }
    connectToServer();
  } else{ //konekcija je aktivna
    if(onConnected){ //konekcija je upravo otvorena
      if(pRemoteChar2->canRead()) {
        remoteBtnCntr = pRemoteChar2->readUInt16();
        Serial.print("Button counter: ");
        Serial.println(remoteBtnCntr);
      }
      onConnected = false;
      remoteLEDPeriod = 0;
    } else{ //konekcija u toku
      btnUpState = digitalRead(BTN_UP_PIN);
      btnDownState = digitalRead(BTN_DOWN_PIN);
      if(btnUpState == HIGH && lastBtnUpState == LOW){
        if(remoteLEDPeriod < 2000){
          remoteLEDPeriod = remoteLEDPeriod + 100;
          remoteLEDPeriodChanged = true;
        }
      }
      lastBtnUpState = btnUpState;
      if(btnDownState == HIGH && lastBtnDownState == LOW){
        if(remoteLEDPeriod > 0){
          remoteLEDPeriod = remoteLEDPeriod - 100;
          remoteLEDPeriodChanged = true;
        }
      }
      lastBtnDownState = btnDownState;

      if(remoteLEDPeriodChanged){
        if(pRemoteChar1->canWrite()){
          uint8_t ledPeriodChar[2];
          ledPeriodChar[0] = remoteLEDPeriod & 0xFF;
          ledPeriodChar[1] = (remoteLEDPeriod >> 8) & 0xFF;
          pRemoteChar1->writeValue(ledPeriodChar,2,false);
        }
        Serial.print("Remote period: ");
        Serial.println(remoteLEDPeriod);
        remoteLEDPeriodChanged = false;
      }
    }
  }
  delay(1000);
}
```

Tokom trajanja konekcije, nadgledaju se dva tastera i u slučaju pritiska, ažurira se perioda blinkanja „serverske“ LED (promenljiva *remoteLEDPeriod*), tj. period blinkanja se povećava ili smanjuje za $100ms$, a zatim se nova vrednost upisuje u karakteristiku 1. Za upis vrednosti karakteristike, na raspolaganju je samo funkcija *writeValue*() koja očekuje da joj se vrednost za upis prenese u obliku niza bajtova. Zbog toga je, pre samog upisa, prvo deklarisan niz bajtova dužine 2 bajta (*ledPeriodChar*[2]), a zatim je sadržaj promenljive *remoteLEDPeriod,* koja je tipa *uint16_t*, prepisan u dva bajta niza *ledPeriodChar*. Argumenti funkcije *writeValue*() su: referenca na niz bajtova, njegova dužina i indikacija da li BLE klijent treba da očekuje potvrdu upisa od strane BLE servera (*false* znači da se potvrda ne očekuje).

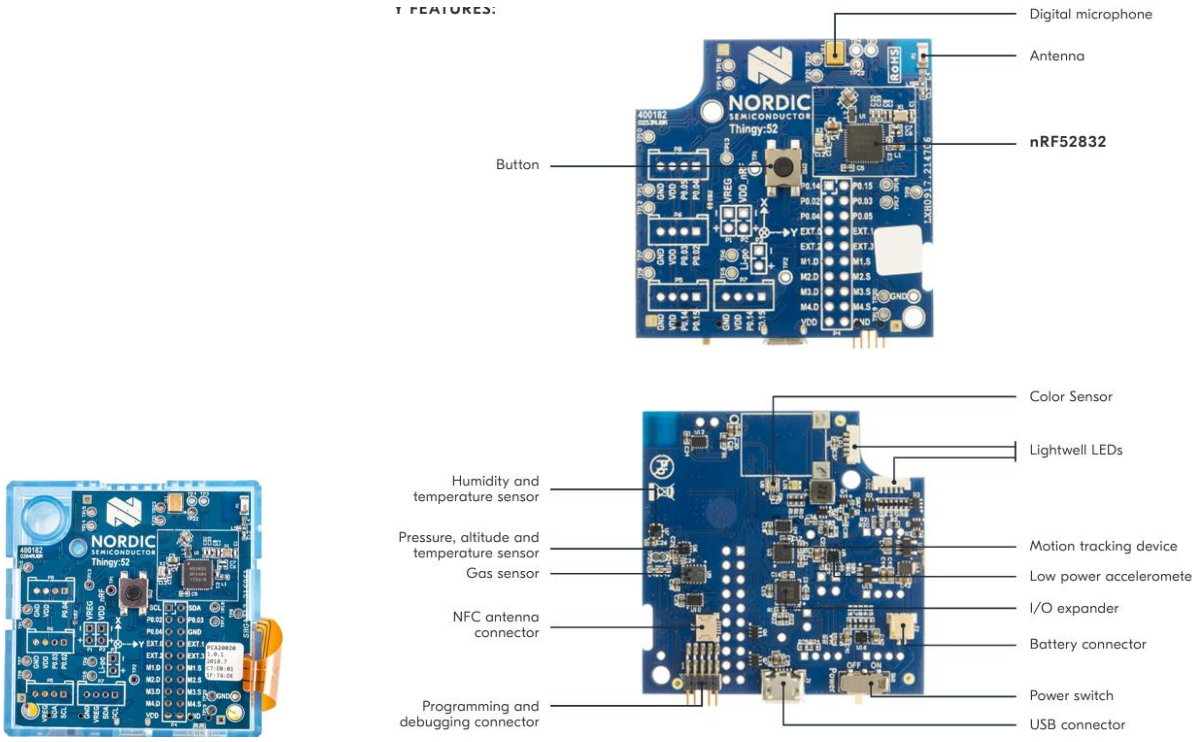
Razmotrimo sada funkciju *connectToServer*() koja se poziva u liniji 130, nakon što je otkriven uređaj s kojim želimo da se povežemo. U ovoj funkciji, prvo se kreira instanca BLE klijenta i povezuje sa odgovarajućim *callback* funkcijama (linije 66 i 67). *Callback* funkcije *onConnect*() i *onDesconnect*() (linije 45 -51) služe za dojavu promene statusa BLE konekcije (uspostavljena/raskinuta konekcija). Naredbom *pClient->connect*(*serverDevice*) u liniji 69, nalaže se BLE klijentu da uspostavi konekciju sa BLE serverom na uređaju *serviceDevice* (uređaj koji je upravo otkriven skeniranjem). Odmah nakon upostavljene konekcije menja se parametar BLE konekcije koji definiše maksimalnu dužinu paketa (linija 70). Podrazumevana dužina je samo 21 bajt; sa novim podešavanjem, paketi koji će se razmenjivati uspostavljenom konekcijom mogu biti maksimalne dužine 517 bajta. U kôdu funkcije *connectToServer*() koji sledi u nastavku uspostavlja se veza se servisom i karkateristikama udaljenog BLE servera. Prvo, u liniji 73, naredbom *pRemoteService = pClient->getService(serviceUUID)* pribavlja se refernca na servis prema UUID-u servisa. U pozadini, BLE klijent prosleđuje ovaj zahtev BLE serveru i od njega dobija *handle* na traženi servis. Ukoliko pointer *pRemoteService* ostane prazan, tj. jednak *nullptr* to znači da traženi servis ne postoji na strani BLE server. U tom slučaju, nama srvhe da nastavljamo dalje i BLE konekcija se raskida (linija 77). Na sličan način, ali sada preko pointera na servis, *pRemoteService*, dolazimo do referenci za dve karakteristike (linije 82 – 97). Nepostojanje tražene karateristike na BLE serveru, takođe, dovodi do raskida konekcije. Dodatno, za karakteristiku 2, dozvoljavaju se notifikacije uz povezivanje sa odgovarajućom *callback* funkcijom, naredba *pRemoteChar2->registerForNotify(notifyCallback)* u liniji 101. Efekat je taj da će nakon prijema notifikacije karakteristike 2 biti pozvana funkcija *notifyCallback*() u kojoj će dostavljena vrednost brojača pritiska na taser biti prebačena u odgovarajuću lokalnu promenljivu (linije 54-59). Vrednost karakteristie je dostupna u obliku niza bajtova, *uint8_t* pData*, i zato je neophodna konverzija u tip *uint16_t*. Na kraju funkcije *connectToServer*(), postavljaju se indikacije da je konekcija otvorena (*connected=true*) i da je konekcija upravo otvorena (*onConnecte=true*).

5.3 Thingy:52

Thingy:52 je multi-senzorska razvojna platforma, koja objedinjuje BLE SoC iz familije nRF52 firme *Nordic Semiconductors* sa podrškom za BLE verzije 5.3 i mnoštvo senzora i aktuatora (Sl. 4). Ovaj uređaj sadrži senzor pokreta, akcelerometar, senzore temperature, vlažnosti i vazdušnog pritiska, senzor kvaliteta vazduha, senzor boje, mali zvučnik, mikrofon, višebojnu LED i taster. Thingy:52 se može koristiti za brzi razvoj raznorodnih BLE uređaja, naravno uz podršku odgovarajućeg softverskog razvojnog okruženja (koje nije Arduino). Iako je prevashodno namenjen razvoju novih aplikacija, Thingy:52 dolazi kao potpuno operativan BLE uređaj sa implementiranim BLE serverom i servisima za pristup i upravljanje svim podržanim senzorima/aktuatorima. U ovoj vežbi, mi nećemo programirati Thingy:52, već ćemo ga koristiti kao BLE server.

Više podataka o Thingy:52 se može naći na linku:

https://www.nordicsemi.com/Products/Development-hardware/Nordic-Thingy-52



(a) (b)

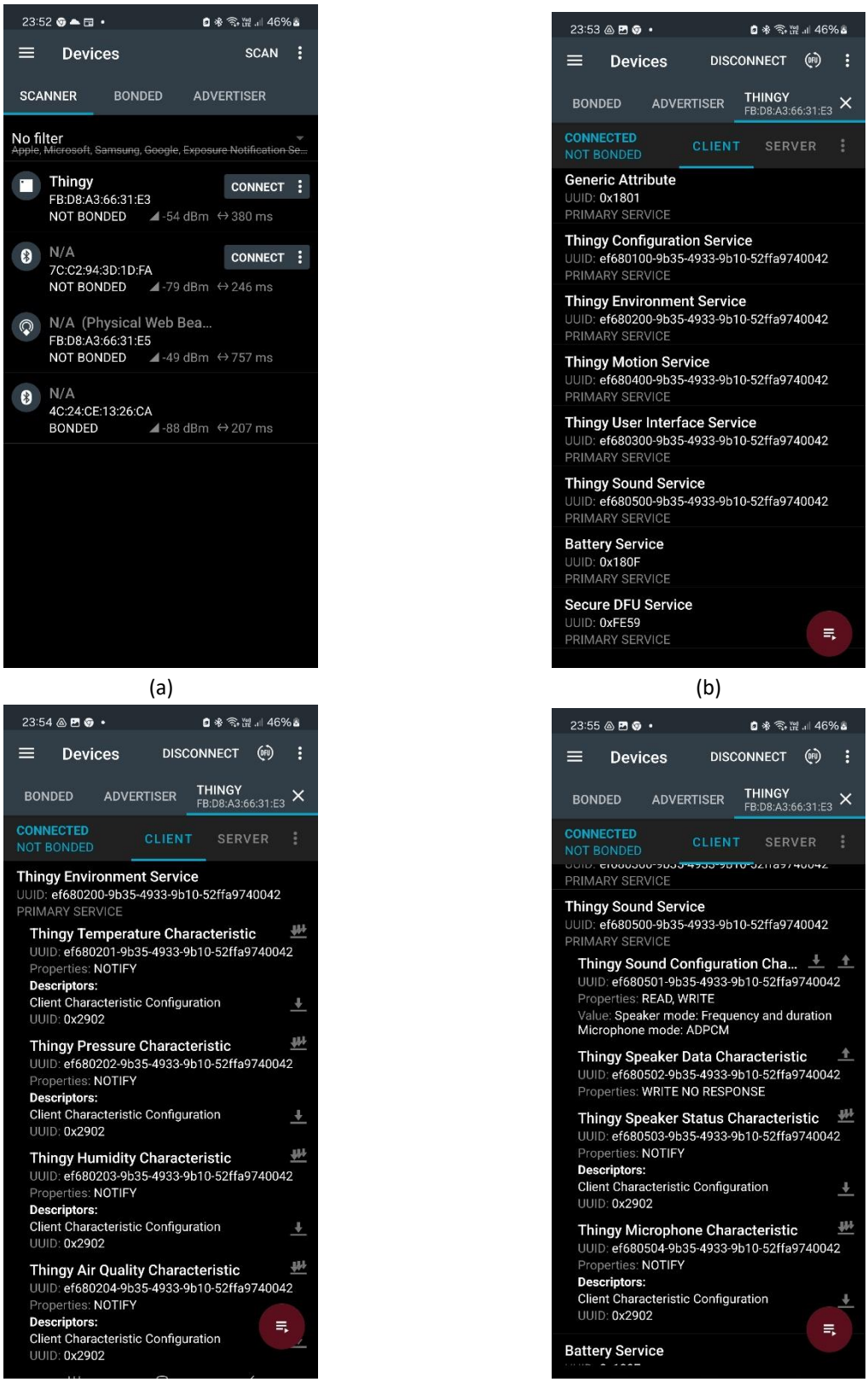
Sl. 4 Thingy:52: (a) Thingy u kutiji; (b) štampana ploča gledano sa gornje i donje strane

Thingy:52 se uključuje kliznim prekidačem koji se nalazi s bočne strane. LED koja treperi u plavoj boji označava da je Thingy u režimu oglašavanja i može se otkriti aplikacijom *nRFConnect* (Sl. 5(a)). Nakon konektovanja, LED nastavlja da treperi, ili sada u zelenoj boji. U *nRFConnect* aplikaciji možemo videti spisak servisa koji implementira ovaj uređaj (Sl. 5(b)). Postoji ukupno 7 servisa, od kojih se svaki odnosi na jedan specifičan aspekt rada ove razvojne platforme. *Thingy Configuration Service* omogućava podešavanje opštih parametara BLE uređaja i konekcije. *Thingy Environment Service* omogućava pristup senzorima temperature, vlažnosti vazduha, vazdušnog pritiska i kvaliteta vazduha. Posredstvom *Thingy Motion Service* pristupa se senzoru pokreta i akceleromentru. Upravljanje LED i pristup tasteru je omogućen putem servisa *Thingy User Interface Service*. *Thingy Sound Service* pruža pristup zvučniku i mikrofonu. *Battery Service* je standardni servis za praćenje nivoa napunjenosti baterije. Konačno, servis *Secure DFU Service* pruža podršku za ažuriranje firmvera.

Od svih podržanih servisa, nama će biti od interesa *Thingy Environment Service* i *Thingy Sound Service*. Na Sl. 5(c) i (d) su prikazane karakteristike ova dva servisa. Vidimo da sve karakteristike *Thingy Environment Service* podržavaju notifikacije, ali ne i čitanje/upis. Nakon što dozvolimo notifikacije, moći ćemo da vidimo i izmerene vrednosti temperature, vazdušnog pritiska, relativne vlažnosti i kvaliteta vazduha. Karakteristike servisa *Thingy Sound Service* omogućavaju podešavanje režima rada zvučnika i mikrofona.

Detaljna specifikacija Thingy:52 servisa je dostupna na linku:

https://nordicsemiconductor.github.io/Nordic-Thingy52-FW/documentation/firmware_architecture.html#fw_arch_ble_services



Sl. 5 Pregled servisa i karakteristika Thingy:52 pomoću aplikacije *nRFConnect*

Na Sl. 6 je prikazan isečak iz tabele koja specificira karakteristike servisa *Thingy Environment Service*. Bazni UUID ovog servisa je EF68xxxx-9B35-4933-9B10-52FFA9740042. Kada se xxxx zameni sa 0200, dobija se UUID samog servisa. UUID-ovi karakteristika se dobijaju kada se xxxx zameni odgovarajućom vrednošću iz kolone UUID. U tabeli takođe vidimo i format vrednosti svake karakteristike. Na primer, podatak o temperaturi se

predstavlja sa dva bajta, prvi je celobrojni, a drugi razlomljeni deo vrednosti temperature. Dakle, ako je temperatura 23.5^{0}C , tada će u prvom bajtu biti vrednost 23, a u drugom 5. Slično važi i za karakteristiku za vazdušni pritisak, s tim što se celobrojni deo predstavlja 32-bitnim celim brojem. Podatak o vlažnosti vazduha se čuva u samo jednom bajtu.

Name	UUID	Type	Data	Description
Base UUID	EF68xxxx-9B35-4933-9B10-52FFA9740042			
Weather station service	0200			
Temperature characteristic	0201	Notify	2 bytes	Temperature in Celsius • int8_t - integer • uint8_t - decimal
Pressure characteristic	0202	Notify	5 bytes	Pressure in hPa • int32_t - integer • uint8_t - decimal
Humidity characteristic	0203	Notify	1 bytes	Relative humidity in % • uint8_t - RH

Sl. 6 Isečak iz specifikacije servisa *Thingy Environment Service* (u tabeli je greškom ime ovog servisa napisano kao *Weather station service*)

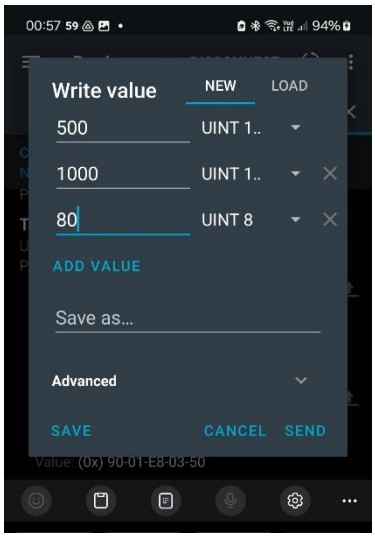
Name	UUID	Type	Data	Description
Base UUID	EF68xxxx-9B35-4933-9B10-52FFA9740042			
Thingy sound service	0500			

Config characteristic	0501	Write/Read	2 bytes	Sound configuration
				• uint8_t - Speaker mode:
				  ◦ 0x01 - Frequency and duration.
				  ◦ 0x02 - 8-bit PCM
				  ◦ 0x03 - Sample
				• uint8_t - Microphone mode
				  ◦ 0x01 - ADPCM
				  ◦ 0x02 - SPL

Speaker data characteristic	0502	Write without resp	Max 273 bytes	In frequency mode:
				• uint16_t - Frequency in Hz (0 - stop forever).
				• uint16_t - Duration in ms (0 = forever).
				• uint8_t - Volume in % (0-100).
				In PCM mode:
				• uint8_t [20 - 273] 8-bit PCM samples
				In sample mode:
				• uint8_t - sample ID (0x00 - 0x08)
				  ◦ 0 Collect_Point_00.wav
				  ◦ 1 Collect_Point_01.wav
				  ◦ 2 Explosion_02.wav
				  ◦ 3 Explosion_04.wav
				  ◦ 4 Hit_00.wav
				  ◦ 5 Pickup_01.wav

Sl. 7 Isečak iz specifikacije servisa *Thingy Sound Service*

UUID servisa *Thingy Sound Service* je EF680500-9B35-4933-9B10-52FFA9740042 (Sl. 7). Interesantne su prve dve karakteristike. Karakteristika *Config characteristic* definiše režima rada zvučnika i mikrofona. Na primer, režim rada zvučnika 0x01 (što je i podrazumevana vrednost), znači da se zvučnikom upravlja tako što se zadaje frekvencija i trajanje tona koji on treba da generiše. Karakteristika *Speaker data characteristic* se koristi za dostavljanje radnih parametara zvučniku: frekvencija tona u Hz, trajanje tona u ms i jačina zvuka u %. Na primer, ako preko aplikacije *nRFConnect*, za ovu karakteristiku unesemo vrednosti kao na Sl. 8, Thingy će se oglasiti kratkotrajnim tonom. Za podatke tipa *unit16_t* treba birati „*UINT 16 (Little Endian)*“.



Sl. 8 Upis vrednosti karakteristike *Speaker data characteristic* servisa *Thingy Sound Service*. U poljima za tip podataka prve dve vrednosti izabrano je *„UINT 16 (Little Endian)"*

5.4 Zadatak

Zadatak 5.1 (25): Cilj projekta je realizacija BLE uređaja na bazi ESP32-S3 platforme u ulozi BLE klijenta. BLE uređaj treba da preuzima podatke o temperaturi, vazdušnom pritisku i relativnoj vlažnosti od Thingy:52 senzora i štampa ih preko serijskog terminala.

Neposredno nakon uspostavljanja BLE konekcije sa Thingy:52

- BLE klijent šalje komandu Thingy:52 za generisanje kratkotrajnog tona kao potvrdu uspešne konekcije.

Tokom trajanja BLE konekcije, uređaj treba da:

- Prima notifikacije od Thingy:52 koje sadrže podatke o temperaturi, vazdušnom pritisku i relativnoj vlažnosti.
- Štampa svaki primljeni podatak na serijskom monitoru.

Napomena 1: Program treba organizovati po ugledu na premer „ESP32-S3 BLE klijent": periodično skeniranje, konektovanje, povratak na skeniranje nakon raskida konekcije. U odnosu „ESP32-S3 BLE klijent", program koji treba da napišete će biti obimniji, jer treba da ostvari vezu sa dva servisa i ukupno četiri karakteristike. Takođe, tri karakteristike podržavaju notifikacije, i za svaku od njih je potrebno predvideti posebnu *callback* funkciju za notifikaciju.

Napomena 2: U paketima za oglašavanje koje emituje Thingy:52 nisu uvršteni UUID-ovi svih podržanih servisa, već samo UUID servisa *Thingy Configuration Service*: EF680100-9B35-4933-9B10-52FFA9740042

```
Advertised Device: Name: , Address: 47:b3:72:b6:c5:5c, manufacturer data: 75001822, rssi: -77
Advertised Device: Name: Thingy, Address: fb:d8:a3:66:31:e3, manufacturer data: 5900242fa64b, serviceUUID: ef680100-9b35-4933-9b10-52ffa9740042,
Advertised Device: Name: , Address: 7c:c2:94:3d:1d:fa, manufacturer data: 8f030a109b0d00f81d3d94c27c81, rssi: -58
```

To znači da kao kriterijum za otkrivanje uređaja u *callback* funkciji BLE skenera treba koristiti ovaj UUID.

Napomena 3: Podaci koji pristižu notifikacijama dolaze kao niz bajtova, zato je potrebno obratiti pažnju na specifikaciju njihovog formata (Sl. 6 i Sl. 7). Na primer, podatak o temperaturi sadrži dva bajta: prvi predstavlja celobrojni, a drugi razlomljeni deo vrednosti temperature. Podatak o pritisku sadrži čak 5 bajta:

prva 4 su celobrojni, peti razlomljeni deo. Prva četiri bajta se tumače kao 32-bitni ceo broj u *little endian* formatu. Na primer, ako u notifikaciji karakteristike za pritisak dobijemo niz bajtova: 0xE0, 0x03, 0x00, 0x00, 0x49, zapravo smo dobili vrednost $992.73\ Pa$ (Paskala). (0xE0, 0x03) je niža, a (0x00, 0x00) viša 16-bitna reč celobrojnog dela. U okviru svake 16-bitne reči, na početku je viši, a onda niži bajt.

Napomena 4: Radi generisanja tona nakon uspostavljene konekcije nije potrebno podešavati režim rada zvučnika upisom u karakteristiku *Config characteristic* – oslonićemo se na podrazumevano podešavanje. Potrebno je samo izvršiti upis vrednosti u karakteristiku *Speaker data characteristic*, koja je dužine 5 bajta: dva *uint16_t* i jedan *uint8_t*. Međutim, to ne znači da treba tri puta pozivati funkciju *writeValue*(), za svaki pojedinačni podatak po jednom, već je potrebno tri podatka, na odgovarajući način, spakovati u niz od 5 bajt i funkciju *writeValue*() pozvati samo jedanput. Na primer, za frekvenciju $500Hz$, trajanje $1000ms$ i jačinu zvuka 80%, sadržaj niza bajtova bi bio: „F401E80350“. Prve 4 hex cifre predstavljaju 16-bitnu vrednost 500 u *little endian* formatu – prvo niži, pa viši bajt. Slično, „E803“ je 1000 (*little endian*), dok je „50“, naravno vrednost 80_{10}. Niz bajtova se može deklarisati npr. ovako:

```
uint8_t soundArray[5] = {0xF4,0x01,0xE8,0x03,0x50};
```

Ili se može popunjavati programskim putem, bajt po bajt.

Napomena 5: Prilikom provere prava upisa karakteristike *Speaker data characteristic*, umesto funkcije *canWrite*(), koja je korišćena u primeru „ESP32-S3 BLE klijent“, potrebno je koristiti funkciju *canWriteNoResponse*(). To je zato što ova karakteristika dopušta isključivo upis bez potvrde upisa (videti u tabeli na Sl. 7).

Zadatak 5.2 (30): Unaprediti BLE uređaj iz zadatka 1 tako da osim štampanja preko serijskog monitora, primljene podatke prikazuje na 4-cifarskom displeju. Sledi dopunjena specifikacija:

Specifikacija uređaja:

- Uređaj treba da sadrži dve LED diode i 4-cifarski displej.

Neposredno nakon uspostavljanja BLE konekcije sa Thingy:52

- BLE klijent šalje komandu Thingy:52 za generisanje kratkotrajnog tona kao potvrdu uspešne konekcije.

Tokom trajanja BLE konekcije, uređaj treba da:

- Prima notifikacije od Thingy:52 koje sadrže podatke o temperaturi, vazdušnom pritisku i relativnoj vlažnosti.
- Štampa svaki primljeni podatak na serijskom monitoru.
- Čuva primljene podatke u odgovarajućim lokalnim promenljivama.

Prikazivanje podataka na displeju:

- Na displeju se naizmenično prikazuju vrednosti temperature, vazdušnog pritiska i relativne vlažnosti, svaka u trajanju od 2 sekunde.
- Status LED dioda menja se u skladu sa prikazom na displeju:
    - Zelena LED svetli dok je prikazana temperatura.
    - Crvena LED svetli dok je prikazan vazdušni pritisak.
    - Obe LED svetle dok je prikazana relativna vlažnost.
    - Zbog ograničenja broja cifara na displeju, prikazuje se samo celobrojni deo podatka o vazdušnom pritisku.

Nakon raskida BLE konekcije:

- LED diode i displej se gase.

- ♦ ♦ ♦ -