15_Cloud_BLE_Relazione

Citterio Giorgio e Colombo Umberto

Lo scopo di questa attività è utilizzare ESP32 per realizzare dispositivi che utilizzano il protocollo Bluetooth Low Energy (BLE).

---

parte 1

Nella prima parte dell'attività abbiamo installato le librerie necessarie al controllo dell'ESP32 dall' IDE di Arduino (gestore schede).

Successivamente abbiamo provato il programma d'esempio seguente per la lettura e scrittura di due caratteristiche di prova e per il controllo del sensore.

sketch_server_BLE:

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>

// definizione degli UID del servizio e delle due caratteristiche
#define SERVICE_UUID         "00000000-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID1 "11111111-36e1-4688-b7f5-ea07361b26a8"
#define CHARACTERISTIC_UUID2 "22222222-36e1-4688-b7f5-ea07361b26a8"
#define CHARACTERISTIC_SENSOR "33333333-36e1-4688-b7f5-ea07361b26a8"
#define ADCPIN A0

BLEServer *pServer;
BLEService *pService;
BLECharacteristic *pCharacteristic1;
BLECharacteristic *pCharacteristic2;
BLECharacteristic *pCharacteristicSensor;


void setup()
{
  Serial.begin(115200);
  Serial.println("Starting BLE Server!");

  // attivazione del device BLE e creazione delle caratteristiche
```

```
  // la prima in R/W la seconda solo R
  BLEDevice::init("Controllo ambiente gioUmbe");
  pServer = BLEDevice::createServer();
  pService = pServer->createService(SERVICE_UUID);
  pCharacteristic1 = pService->createCharacteristic(
                      CHARACTERISTIC_UUID1,
                      BLECharacteristic::PROPERTY_READ |
                      BLECharacteristic::PROPERTY_WRITE);
  pCharacteristic2 = pService->createCharacteristic(
                      CHARACTERISTIC_UUID2,
                      BLECharacteristic::PROPERTY_READ);
  pCharacteristicSensor = pService->createCharacteristic(
                      CHARACTERISTIC_SENSOR,
                      BLECharacteristic::PROPERTY_READ);

  // impostazione dei valori iniziali delle caratteristiche
  pCharacteristic1->setValue("init");
  pCharacteristic2->setValue("1");
  pCharacteristicSensor->setValue("0");

  // attivazione del servizio e del'advertising
  pService->start();
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(true);
  pAdvertising->setMinPreferred(0x12);
  BLEDevice::startAdvertising();
}
int i = 0;
void loop()
{
  int num = analogRead(ADCPIN);
  char s[5];
  sprintf(s, "%04d", num);
  pCharacteristicSensor -> setValue(s);

  // acquisizione del valore della prima caratteristica
  std::string value = pCharacteristic1->getValue();

  // impostazione del valore della seconda caratteristica
  // (valore progressivo)
  char str[100];
  sprintf(str, "%d", i++);
  pCharacteristic2->setValue(str);

  // stampa
  Serial.print("Caratteristica sensore: ");
  Serial.print(s);
  Serial.print("    -    Caratteristica 1: ");
  Serial.print(value.c_str());
  Serial.print("    -    Caratteristica 2: ");
  Serial.println(i);

  // ripresa dell'advertising
```

```
  BLEDevice::startAdvertising();

  delay(2000);
}
```

parte 2-3

In questa parte abbiamo scaricato un’app (scanner BLE) per rilevare il nostro dispositivo e controllarne il corretto funzionamento.

parte 4

In questa parte dell’attività abbiamo realizzato un programma python utilizzando la librerie *Bleak* per leggere il valore della caratteristica del sensore.

```
import asyncio
from bleak import BleakClient
import time

ESP32_ADDRESS = "C8:C9:A3:CB:F5:66"
CHARACTERISTIC_UUID2 = "22222222-36e1-4688-b7f5-ea07361b26a8"

async def main(address,uuid):
    async with BleakClient(address) as client:
        while True:
            value = await client.read_gatt_char(uuid)
            print(value)
            time.sleep(5)

asyncio.run(main(ESP32_ADDRESS,CHARACTERISTIC_UUID2))
```

parte 5

In quest’ultima parte dell’attività abbiamo realizzato il programma per l’ESP32 per il controllo completo dell’ambiente tramite BLE, direzione e velocità del motore e lettura dei dati del sensore.

sketch_server_BLE_sensore_motore:

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
```

```
// definizione degli UID del servizio e delle due caratteristiche
#define SERVICE_UUID         "177c5272-c6c8-4ecd-a2e4-6e6bb7651ce1"
#define CHARACTERISTIC_UUID1 "776be67f-9fcb-40b7-87cb-458e23f15638"
#define CHARACTERISTIC_SENSOR "a89adab3-fb71-488d-940f-dea867890df9"
#define CHARACTERISTIC_MOTORDIRECTION "a395aaa4-12fd-484f-b2e7-b4941e0e001a"
#define CHARACTERISTIC_MOTORSPEED "8f0a2fa8-cad4-435d-907b-38f44f4fe5d3"
#define ADCPIN A0 //SP
#define AVANTI_PIN 14 //GPIO14
#define INDIETRO_PIN 27 //GPIO27
#define VELOCITA_PIN 33 //GPIO33

BLEServer *pServer;
BLEService *pService;
BLECharacteristic *pCharacteristic1;
BLECharacteristic *pCharacteristicSensor;
BLECharacteristic *pCharacteristicMotorDirection;
BLECharacteristic *pCharacteristicMotorSpeed;


void setup()
{
  Serial.begin(115200);
  Serial.println("Starting BLE Server!");
  pinMode(AVANTI_PIN, OUTPUT);
  pinMode(INDIETRO_PIN, OUTPUT);

  // attivazione del device BLE e creazione delle caratteristiche
  // la prima in R/W la seconda solo R
  BLEDevice::init("Controllo ambiente gioUmbe");
  pServer = BLEDevice::createServer();
  pService = pServer->createService(SERVICE_UUID);
  pCharacteristic1 = pService->createCharacteristic(
                      CHARACTERISTIC_UUID1,
                      BLECharacteristic::PROPERTY_READ |
                      BLECharacteristic::PROPERTY_WRITE);
  pCharacteristicSensor = pService->createCharacteristic(
                      CHARACTERISTIC_SENSOR,
                      BLECharacteristic::PROPERTY_READ);
  pCharacteristicMotorDirection = pService->createCharacteristic(
                      CHARACTERISTIC_MOTORDIRECTION,
                      BLECharacteristic::PROPERTY_WRITE);
  pCharacteristicMotorSpeed = pService->createCharacteristic(
                      CHARACTERISTIC_MOTORSPEED,
                      BLECharacteristic::PROPERTY_WRITE);

  // impostazione dei valori iniziali delle caratteristiche
  pCharacteristic1->setValue("init");
  pCharacteristicSensor->setValue("0");
  pCharacteristicMotorDirection->setValue("A");
  pCharacteristicMotorSpeed->setValue("0");

  // attivazione del servizio e del'advertising
  pService->start();
```

```
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(true);
  pAdvertising->setMinPreferred(0x12);
  BLEDevice::startAdvertising();
}

void loop()
{
  // acquisizione del valore della prima caratteristica
  std::string value = pCharacteristic1->getValue();

  //lettura valori sensore e salvataggio
  int num = analogRead(ADCPIN);
  char s[5];
  sprintf(s, "%04d", num);
  pCharacteristicSensor -> setValue(s);

  //controllo del motore
  std::string velo = pCharacteristicMotorSpeed->getValue();
  std::string dir = pCharacteristicMotorDirection->getValue();
  const char* direzione = dir.c_str();
  const char* vel = velo.c_str();
  int velocita = atoi(vel);
  if (strcmp("A", direzione) == 0)
  {
    digitalWrite(INDIETRO_PIN, LOW);
    digitalWrite(AVANTI_PIN, HIGH);
    analogWrite(VELOCITA_PIN, velocita);
  }
  if (strcmp("I", direzione) == 0)
  {
    digitalWrite(INDIETRO_PIN, HIGH);
    digitalWrite(AVANTI_PIN, LOW);
    analogWrite(VELOCITA_PIN, velocita);
  }

  // stampa
  Serial.print("Caratteristica 1: ");
  Serial.print(value.c_str());
  Serial.print("    -    Caratteristica sensore: ");
  Serial.print(s);
  Serial.println();
  Serial.print("Caratteristica motore direzione: ");
  Serial.print(direzione);
  Serial.print("    -    Caratteristica motore velocità: ");
  Serial.print(velocita);
  Serial.println();

  // ripresa dell'advertising
  BLEDevice::startAdvertising();

  delay(2000);
}
```