```
/*
*ESP32に接続したスイッチの状態読み取りとLEDの制御をBLE経由で行う。
*スイッチ情報はタイマー割り込みを使って1秒毎に情報を更新する。
*スイッチにはReadとNotifyの属性を与える。
*PC側にはWebbluetoothAPIを用いたhtmlを用意する。
*/
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

bool FLAG = 0;                                  //タイマー割り込みフラグ。割り込みが発生したら"1"を立てる

hw_timer_t * timer = NULL;

//割込みサービスルーチン
void IRAM_ATTR Check() {                        //IRAM_ATTRを付ける事により、この関数が内部RAMに配置される。
                                                //(Flashに配置されるとディレイが生じて動作に問題が起きる可能性が有るらしい)
  FLAG = 1;                                     //割り込みが有ったらフラグを立てる
}

BLEServer* pServer = NULL;
BLECharacteristic* pCharacteristic_SW = NULL;
BLECharacteristic* pCharacteristic_LED = NULL;

bool deviceConnected = false;
bool oldDeviceConnected = false;
uint8_t value = 0;                              //SW情報格納用(+LEDキャラクタリスティックス初期化用)

// See the following for generating UUIDs:
// https://www.uuidgenerator.net/

#define SERVICE_UUID            "77920fa2-1586-4de5-adc0-4eebe0861350"  //UUID Generatorで生成したサービスUUIDと
#define CHARACTERISTIC_SW_UUID  "8f47052e-907e-4824-9300-368fa08fadaa"  //SW情報のキャラクタリスティックスUUID
#define CHARACTERISTIC_LED_UUID "17b65d0f-c42f-4cf5-9159-8fc71fb52b96"  //LED情報のキャラクタリスティックスUUID

class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
    }
    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
};

class MyCallbacks: public BLECharacteristicCallbacks {                  //LED制御データを受信した時の処理を記述
    void onWrite(BLECharacteristic *pCharacteristic_LED) {
      std::string value = pCharacteristic_LED->getValue();

      if (value.length() > 0) {
        Serial.println("*********");
        Serial.printf("New Value: %d¥n",value[0]);
        Serial.println("*********");
        if ((value[0] & 0x02) != 0) {
          digitalWrite(32, HIGH);
        }
        else if ((value[0] & 0x02) == 0) {
          digitalWrite(32, LOW);
        }
        if ((value[0] & 0x01) != 0) {
          digitalWrite(33, HIGH);
        }
        else if ((value[0] & 0x01) == 0) {
          digitalWrite(33, LOW);
        }
      }
    }
};


void setup() {
  Serial.begin(115200);                 //シリアルポートを115.2kbpsで有効化

  //入力ピン割り当て(プルアップ抵抗付き)
  pinMode(25, INPUT_PULLUP);
  pinMode(26, INPUT_PULLUP);

  //出力ピン割り当て
  pinMode(32, OUTPUT);
  pinMode(33, OUTPUT);

  //２つの出力ピンをオフ (Highでオン、Lowでオフ)
  digitalWrite(32, LOW);
  digitalWrite(33, LOW);

  timer = timerBegin(0, 80, true);              //timer=1us  タイマー0を80分周、カウントアップモードにする。
```

```
                                          //源振が80MHzなので、80分周してクロックを1MHz(=1uS)にする。
  timerAttachInterrupt(timer, &Check, true);      //タイマー0のハンドラで、割り込み時呼び出される関数を指定し、割り込みタイプをエッジに指定
  timerAlarmWrite(timer, 1000000, true);          //タイマー0が100万回(1秒)経ったら割り込み、オートリロード(true)する
  timerAlarmEnable(timer);                        //タイマースタート

  // Create the BLE Device
  BLEDevice::init("ESP32_IO");

  // Create the BLE Server
  pServer = BLEDevice::createServer();
  pServer -> setCallbacks(new MyServerCallbacks());

  // Create the BLE Service
  BLEService *pService = pServer -> createService(SERVICE_UUID);

  // Create a BLE Characteristic
  pCharacteristic_SW = pService -> createCharacteristic(
                       CHARACTERISTIC_SW_UUID,
                       BLECharacteristic::PROPERTY_READ   |
                       BLECharacteristic::PROPERTY_NOTIFY
                     );

  // Create a BLE Descriptor
  pCharacteristic_SW -> addDescriptor(new BLE2902());

 // Create a BLE Characteristic
  pCharacteristic_LED = pService -> createCharacteristic(
                       CHARACTERISTIC_LED_UUID,
                       BLECharacteristic::PROPERTY_READ   |
                       BLECharacteristic::PROPERTY_WRITE
                     );

  pCharacteristic_LED -> setCallbacks(new MyCallbacks());
  pCharacteristic_LED -> setValue(&value, 1);                  //LEDキャラクタリスティックスの初期値設定(valueは"0"で初期化されている)

  // Start the service
  pService -> start();

  // Start advertising
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising -> addServiceUUID(SERVICE_UUID);
  pAdvertising -> setScanResponse(false);
  pAdvertising -> setMinPreferred(0x0);  // set value to 0x00 to not advertise this parameter
  BLEDevice::startAdvertising();
  Serial.println("Waiting a client connection to notify...");
}

void loop() {
    // notify changed value
    if (FLAG == 1) {                      //SW読み込みタイミング？(SWの読み込みはBLEの接続有無に関わらず行っている
      if (digitalRead(25) == LOW) {       //SW状態読み取り及びvalueに反映
        value = 0x02;
      }
      else {
        value = 0x00;
      }
      if (digitalRead(26) == LOW) {
        value = value | 0x01;
      }
      else {
        value = value & 0xFE;
      }

      if (deviceConnected) {                //valueを送信(BLE接続中なら)
          pCharacteristic_SW -> setValue(&value, 1);
          pCharacteristic_SW -> notify();
      }
      FLAG = 0;                             //SW読み込みフラグクリア
    }
    // disconnecting
    if (!deviceConnected && oldDeviceConnected) {
        delay(500); // give the bluetooth stack the chance to get things ready
        pServer->startAdvertising(); // restart advertising
        Serial.println("start advertising");
        oldDeviceConnected = deviceConnected;
    }
    // connecting
    if (deviceConnected && !oldDeviceConnected) {
        // do stuff here on connecting
        oldDeviceConnected = deviceConnected;
    }
}
```