BLEデバイス製作とIPHONEからのコントロールの基本

2013/02/02

Koki Ogura CEO Laksmi-Do Corp.

Twitter: @idev_jp

Mail: idev@laksmido.com

目的：何ができるようになるの？

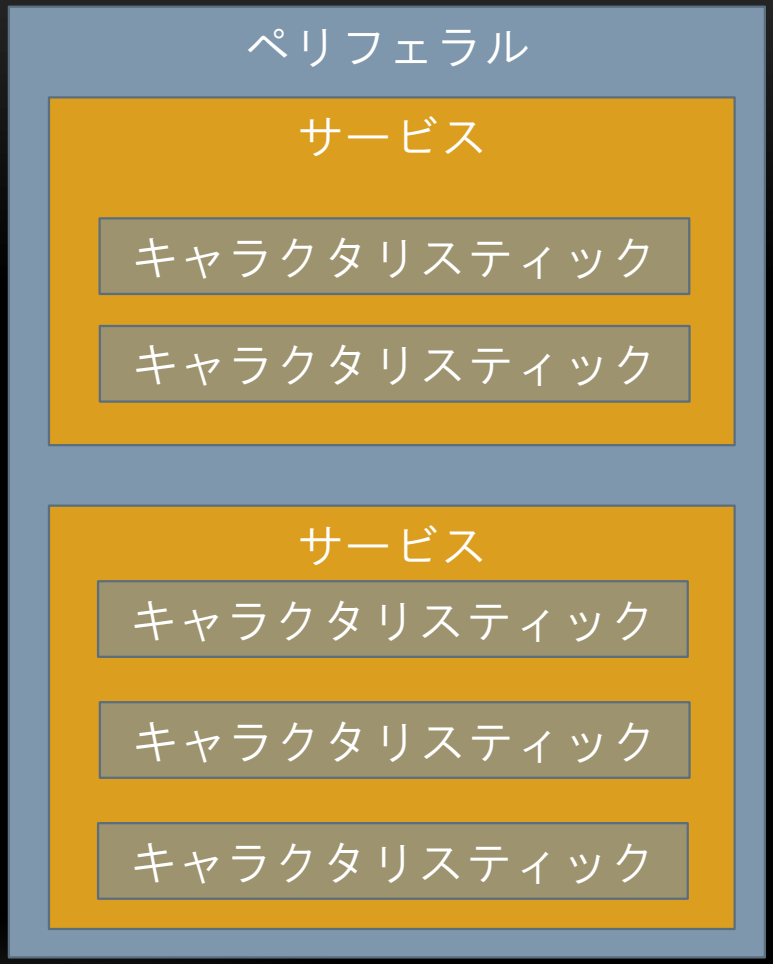
- BLEデバイスにはLEDとプッシュスイッチがついているとする
  - iPhoneからLEDの明るさを調節したり
  - スイッチが押された回数をiPhoneが知ること
- ができるようになる。

３パート構成

- １BLEの基本
- ２デバイス製作の基本
- ３iPhoneアプリ作成の基本

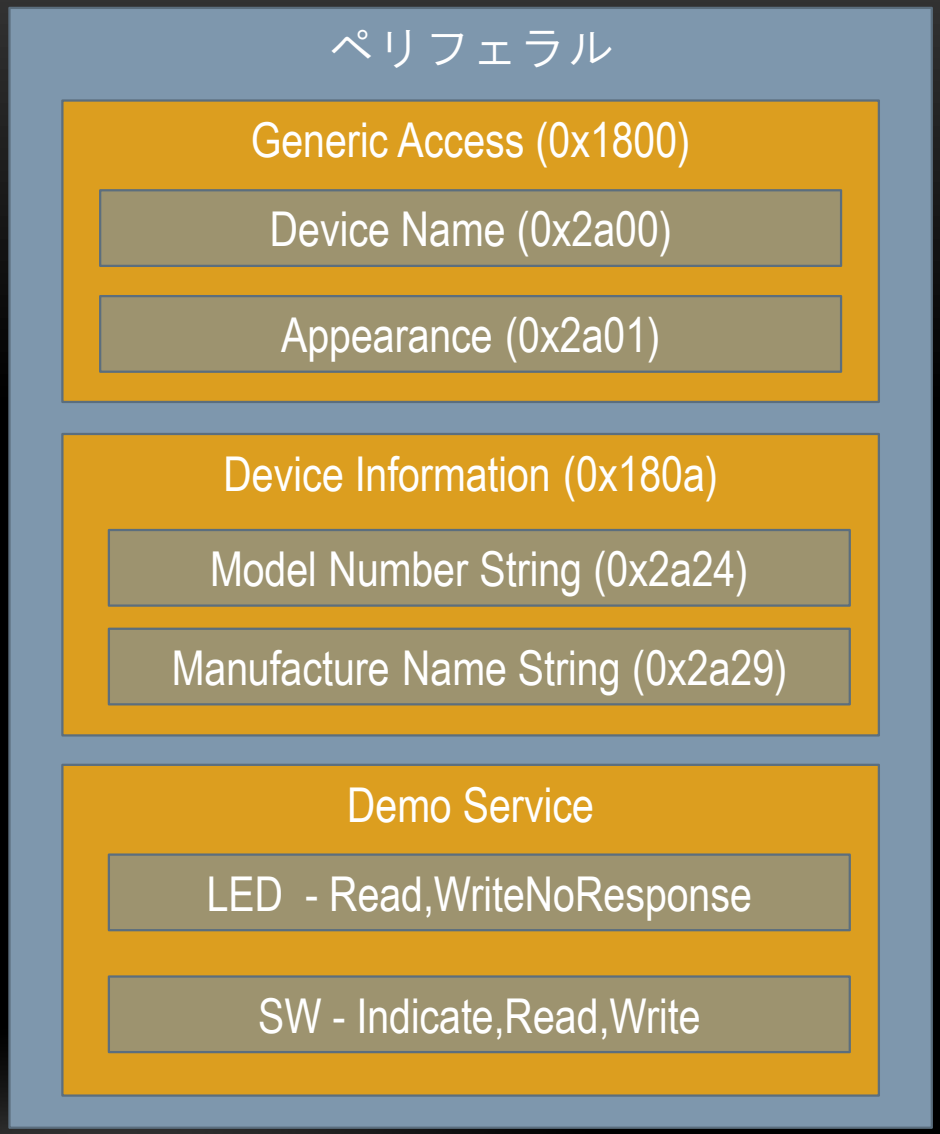
BLEの基本

- GATT(Generic Attribute Profile)を使う
- 1-1 ペリフェラル（デバイス）
- 1-2 サービス
  - uuid, description
  - advertise
- 1-3 キャラクタリスティック（値）
  - uuid, description
  - value
  - properties
    - Read, Indicate, Notify
    - Write,, WriteWithoutResponse



具体例

- General Access
  - デバイスの名前
  - 種類
- Device Information
  - モデル番号
  - 製造者
- Demo Service（独自サービス）
  - LED（独自のキャラ）
  - SW（独自のキャラ）



GATTについて

- すでに決まっているサービスやキャラクタリスティックが多くある
  - 決まっているもののUUIDは２バイトで表されている
  - そうでない自前のものは１６バイトのUUIDを独自に生成し使用する
- 以下のURLを参照
- http://developer.bluetooth.org/gatt

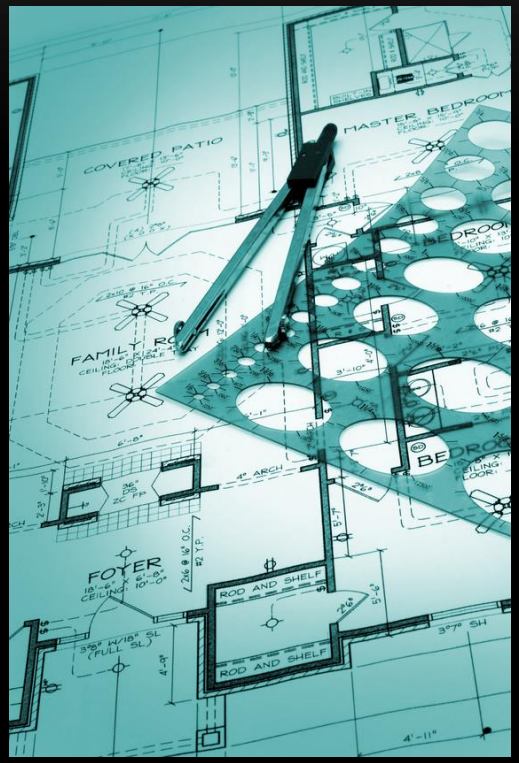
休憩

デバイス製作の基本

- BLUEGIGA社のBLE112を使う
  - http://www.bluegiga.com/bluetooth-low-energy
- BLEに必要な機能を１チップに内蔵
  - マイコン（TI社CC2540）
  - チップアンテナ（最高出力は＋３ｄＢ）
- XMLとスクリプト言語を使って動作を記述する

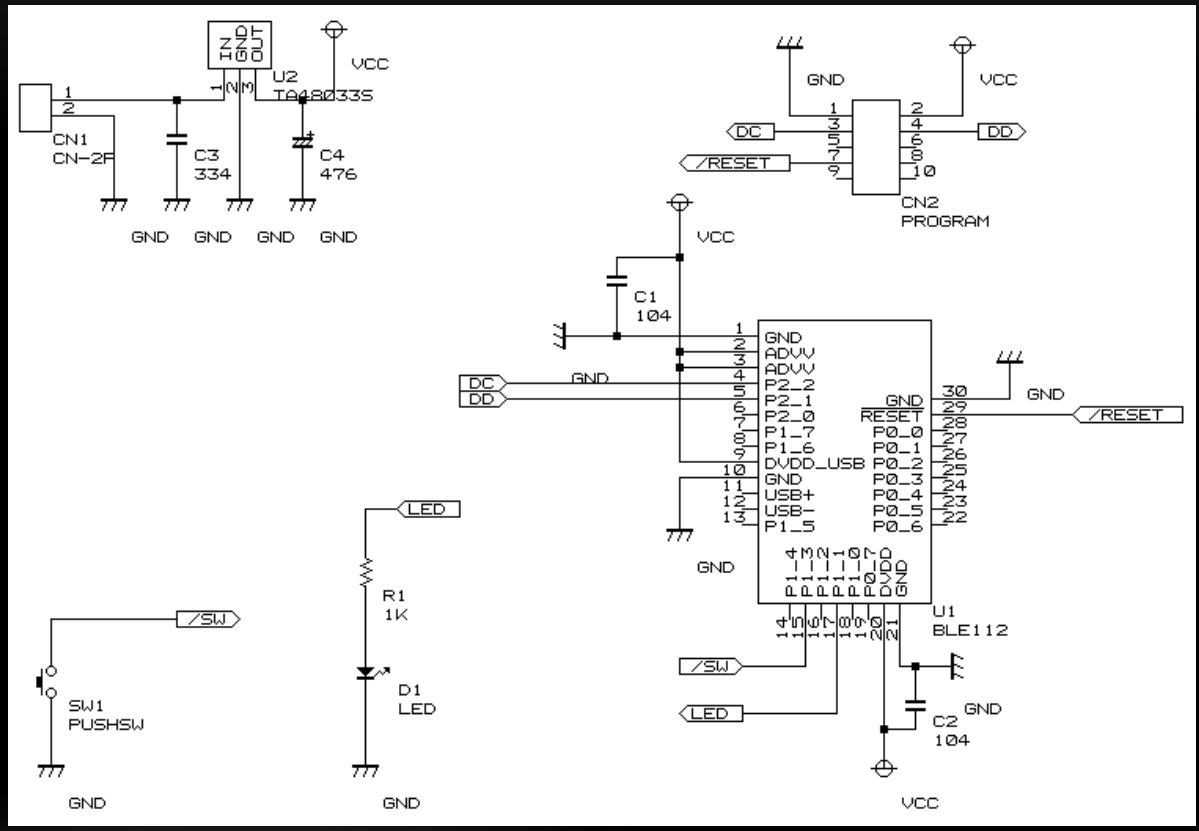
BLE112の入出力ポート

- GPIO 17
- ADC 8
- USART 2
- SPI 2
- TIMER 3
- PWM 4
- 割り当て表はBLE112 Datasheet Page 8

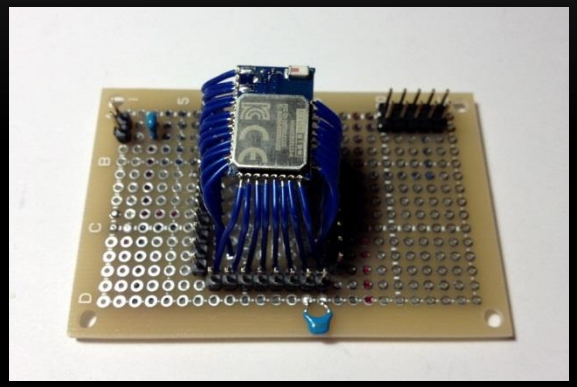


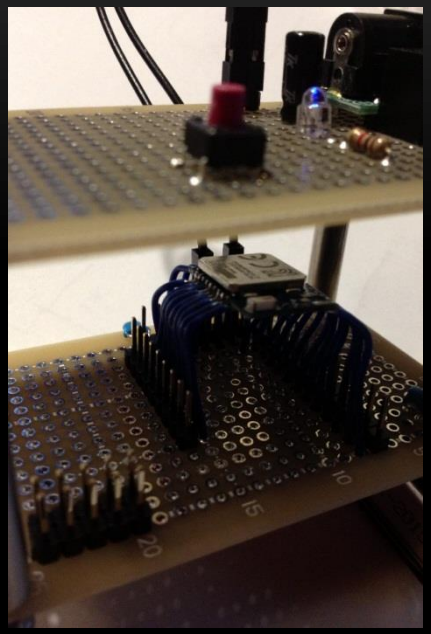
回路図

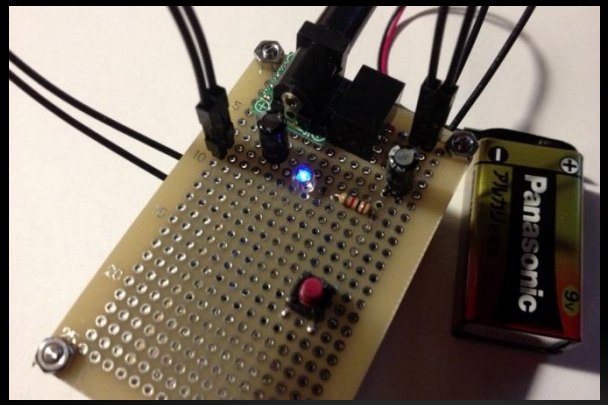
- 電源部　3.3V
- プログラム用コネクタ
- SW
- LED 3.3mA
- BLE112



製作したデバイス







BLE112のプログラム

- ハードウェア記述ファイル（hardware.xml）
- GATT記述ファイル（gatt.xml）
- スクリプト記述ファイル（DemoProg.bgs）
- プロジェクト記述ファイル（project.bgproj）

ハードウェア記述ファイル

- 使用するハードウェア機能を記述する（拡張子xml）
- <?xml version="1.0" encoding="UTF-8" ?>
- <hardware>
- <sleeposc enable="true" ppm="30" />
- <sleep enable="false" />
- <timer index="1" enabled_channels="0x3" divisor="0" mode="2" alternate="2" />
- <txpower power="15" bias="5" />
- <script enable="true" />
- </hardware>

GATT記述ファイル

- GATTを記述する（拡張子xml）
- <?xml version="1.0" encoding="UTF-8" ?>
- <configuration>
  - サービス１
  - サービス２
  - サービス３
  - …
- </configuration>

GATT記述ファイル（サービス１）

```
<service uuid="1800">
    <description>Generic Access</description>
    <characteristic uuid="2a00">
        <properties read="true" const="true" />
        <value>BLE112 Demo Prog</value>
    </characteristic>
    <characteristic uuid="2a01">
        <properties read="true" const="true" />
        <value type="hex">0000</value>
    </characteristic>
</service>
```

GATT記述ファイル（サービス２）

```
<service type="primary" uuid="180A" id="manufacturer">
        <description>Device Information</description>
        <characteristic uuid="2A24">
                <properties read="true" const="true" />
                <value>LSD20130127</value>
        </characteristic>
        <characteristic uuid="2A29">
                <properties read="true" const="true" />
                <value>idev_jp</value>
        </characteristic>
</service>
```

GATT記述ファイル（サービス３）

```
<service uuid="542EED63-5CD0-4184-A840-CB4814EFD9F2" advertise="true">
        <description>Demo Service</description>
        <include id="manufacturer" />
        <characteristic uuid="100BFB9E-A739-498B-A2D0-893DB70DDD97" id="xgatt_led">
                <description>LED</description>
                <properties read="true" write_no_response="true"/>
                <value type="hex">0000</value>
        </characteristic>
        <characteristic uuid="B550FBA0-78A0-45D8-B2FE-CAF6B5CEFF44" id="xgatt_sw">
                <description>SW</description>
                <properties indicate="true" read="true" write="true"/>
                <value type="hex">0000</value>
        </characteristic>
</service>
```

スクリプト記述ファイル

- BGScript言語を使ってデバイスの動作を記述する（拡張子bgs）
- BGScript言語はBASICライクな言語
- BGScript言語はイベントドリブン
- 実際のファイルを見ながら説明します
  - LEDの明るさの変化はタイマー１チャネル１のPWMで実現する
  - SWが押されたかは割り込みを使って知る

プロジェクト記述ファイル

- プロジェクト全体のプログラム要素をxmlで記述する（拡張子bgproj）
- <?xml version="1.0" encoding="UTF-8" ?>
- <project>
- 	<gatt in="gatt.xml" />
- 	<hardware in="hardware.xml" />
- 	<script in="DemoProg.bgs" />
- 	<image out="out.hex" />
- </project>

出来上がったプログラムをアップロード

- TI社のCC DebuggerでPCとデバイスを接続する
- BLE Updateというアプリを使ってアップロード
  - CC Debuggerのポートを指定
  - 作ったプロジェクト記述ファイルを指定
  - あとはUPDATEボタンを押すだけ！
    - コンパイルやエラーチェック
    - 問題なければアップロード
    - そしてデバイスは起動してサービスをAdvertiseしはじめる

入手方法

- 最初は開発キットDKBLE112を購入するのがお勧め
  - http://www.bluegiga.com/evaluation_BLE112
  - 評価ボード
  - CC Debugger
  - BLE112 x2
  - BLED112 x1

- 日本の代理店を通すと納期が１カ月、値段も非常に高くなる（６万程度）
- Mouserで購入するのが早い（在庫あれば４日程度）安い（３万５千程度）
  - http://jp.mouser.com
  - BLE112単体では１７００円程度
  - CC Debugger単体では５千円程度

休憩

IPHONEアプリ作成の基本

- Core Bluetoothを使う
- 利用可能なBLEペリフェラルを知る
- 告知されているサービスを知る
- ペリフェラルと接続する
- ペリフェラルから切断する
- キャラクタリスティックに値を書込む
- キャラクタリスティックから値を読込む
- 汎用クラスを作ってみた
- 汎用クラスを使ったアプリ作成

CORE BLUETOOTHを使う

- Linked Frameworks and Libraries に
  - CoreBluetooth.framework
  - を追加する
- 使用するプログラムに
  - #import <CoreBluetooth/CoreBluetooth.h>
  - を追記する
- デリゲートを指定しておく
  - @interface HogeHogeClass : SuperHogeClass <CBCentralManagerDelegate>
- 使用するクラスのインスタンスを作っておく
  - @property (strong) CBCentralManager* CentralManager;
  - _CentralManager = [[CBCentralManager alloc] initWithDelegate:self queue:nil];

利用可能なBLEペリフェラルを知る

- 接続可能なBLEペリフェラルは一定時間間隔でサービスの告知をおこなっており、それを受信するには次のようにすればよい
  - If ([_CentralManager state] == CBCentralManagerStatePowerOn) {
    - [_CentralManager scanForPeripheralsWithServices:nil options:nil];
  - }
- この結果は次のデリゲートで通知される
  - - (void)centralManager:(CBCentralManager*)central didDiscoverPeripheral:(CBPeripheral*)peripheral advertisementData:(NSDictionary*)advertisementData RSSI☹NSNumber*)RSSI:
- このadvertisementDataに告知されたGATTのサービスの情報が入っている
- なお告知の受信が必要なくなれば次のようにして止める
  - [_CentralManager stopScan];

告知されているサービスを知る

- 先のadvertisementDataから告知されたサービスのuuidを取り出すには
  - NSArray* services;
  - services = [advertisementData objectForKey:@”kCBAdvDataServiceUUIDs”];
- 必要とするサービス（UUID）が次のように定義されていたとする
  - #define MY_SERVICE = @”123456-789a-bcde-f0123456789a”;
- これが告知されたサービスに含まれるかどうかの判定は
  - If ([services containsObject:[CBUUID UUIDWithString:MY_SERVICE]) {
    - // このservicesに必要とするサービスが含まれている！
  - }

ペリフェラルと接続する

- 必要としているサービスがservicesに含まれていれば、次はデリゲートでservices同時に得ることができたperipheralを使ってペリフェラルと接続する
- その方法は次のようになる
  - [_CentralManager connectPeripheral:peripheral options:nil];
- 結果はデリゲートで次の２つのうちの一つが呼ばれる
  - 接続が成功したとき
    - - (void)centralManager: (CBCentralManager*)central didConnectedPeripheral:(CBPheripheral*)peripheral;
  - 接続が失敗したとき
    - - (void)centralManager: (CBCentralManager*)central didFailToConnectedPeripheral:(CBPheripheral*)peripheral;

ペリフェラルから切断する

- 接続しているペリフェラルからは次のようにして切断する
  - [_CentralManager cancelPeripheralConnection:peripheral];
- そしてこの結果はデリゲートで次が呼ばれる
  - - (void)centralManager:(CBCentralManager*)central didDisconnectPeripheral:(CBPeripheral*)peripheral error:(NSError*)error;

ペリフェラルのサービス一覧を得る

- 告知で得られるサービスはペリフェラルの提供するサービスの一部
- 従って接続に成功したらペリフェラルの提供する全てのサービスを知る必要がある。
- まずデリゲートを受け取るクラス
  - @interface HogeClass : SuperClass <CBPeripheralDelegate>
- そして通知を得るには次のようにする
  - [peripheral setDelegate:HogeClassのインスタンス];
  - [peripheral discoverServices:nil];
- 結果はデリゲートで次が呼ばれる
  - - (void)peripheral:(CBPeripheral*)peripheral didDiscoverServices:(NSError*)error;
  - この時、peripheral.servicesにサービス一覧がすでに入っている

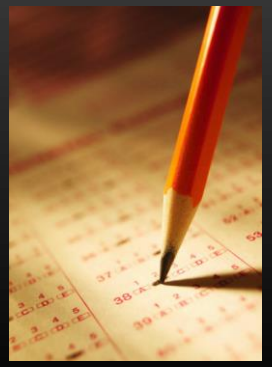
サービスのキャラクタリスティック一覧

- サービス一覧を得ることができたら、次はそれぞれのサービスのキャラクタリスティック一覧を得るようにする。
- これは先のデリゲートの関数内で次のようにする
  - _peripheral = peripheral; // _peripheralはクラスの@propertyで確保しておく
  - for (CBService* service in _peripheral.services) {
    - [_peripheral discoverCharacteristics:nil forService:service];
  - }
- 結果は今までと同様にデリゲートで呼ばれるが、この結果自体は_peripheral.services内の各serviceのservice.characteristicsに入っているのでデリゲート関数は書かなくても良い
- つまり_peripheral.servicesから各サービスservice
- service.characteristicsから各キャラクタリスティックを網羅できる

必要なキャラクタリスティックを得る

- サービスのuuidをs_uuid、キャラのuuidをc_uuidとした時、キャラのクラスCBCharacteristicへの参照は次のようにして得ることができる
- for (CBService* service in _peripheral.services) {
  - if ([service.UUID isEqual:[CBUUID UUIDWithString:s_uuid]]) {
    - for (CBCharacteristic* characteristic in service.characteristics) {
      - if ([characteristic.UUID isEqual:[CBUUID UUIDWithString:c_uuid]]) {
        - // 見つかった！
      - }
    - }
  - }
- }

キャラクタリスティックに値を書込む



- GATTでWriteを指定しているキャラクタリスティックへの書込みは
  - - (void)write:(CBCharacteristic*)characteristic buffer:(void*)buf length:(int)len
  - {
    - NSData* data = [NSData dataWithBytes:buf length:len];
    - [_peripheral writeValue:data forCharacteristic:characteristic type:CBCharacteristicWriteWithResponse];
  - }
- GATTでWriteWithoutResponseを指定しているキャラクタリスティックなら最後のパラメタをCBCharacteristicWriteWithoutResponseとする

キャラクタリスティックから値を読込む

- 読込む時は「読込む為の依頼」をし、結果はデリゲートの関数で得ることとなる。読込みの依頼は
  - [_peripheral readValueForCharacteristic:characteristic];
- 結果は次のデリゲート関数に通知される
  - (void)peripheral:(CBPeripheral*)peripheral didUpdateValueForCharacteristic:(CBCharacteristic*)characteristic error:(NSError*)error;
- このキャラクタリスティックの値は
  - NSData* data = characteristic.value;
  - const void* buf = [data bytes];
  - NSInteger len = [data length];
- で得ることができる

INDICATEやNOTIFYの依頼

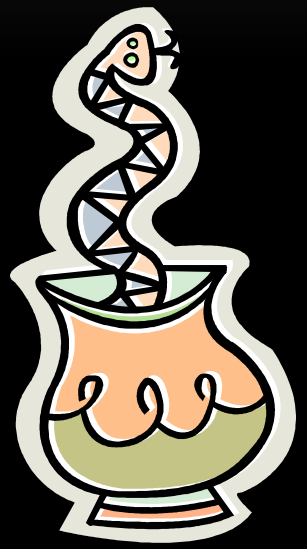
- 接続するだけではIndicateやNotifyは実行されない。
- このプロパティを持っているキャラクタリスティックに対して通知の要請をするには次のようにする
  - [_peripheral setNotifyValue:TRUE forCharacteristic:characteristic];
  - これ以降は読込み依頼に対するのと同じデリゲートで値の更新があった時に通知されるようになる

汎用クラスを作ってみた

- CoreBluetoothは結果がデリゲートで通知される場合が多く非同期処理が必要になる。そこでこれら（の多く）を隠蔽した汎用クラスを作ってみた
  - BLEBaseClass.h
  - BLEBaseClass.m

汎用クラスの使い方１

- ヘッダーのインポート
  - #import “BLEBaseClass.h”
- デリゲートの設定
  - @interface HogeClass () <BLEDeviceClassDelegate>
- プロパティの追加
  - @propterty (strong) BLEBaseClass* BaseClass;
  - @propterty (readwrite) BLEDeviceClass* Device;
- 初期化とサービスのスキャン開始
  - _BaseClass = [[BLEBaseClass alloc] init];
  - [_BaseClass scanDevices:nil];



汎用クラスの使い方２

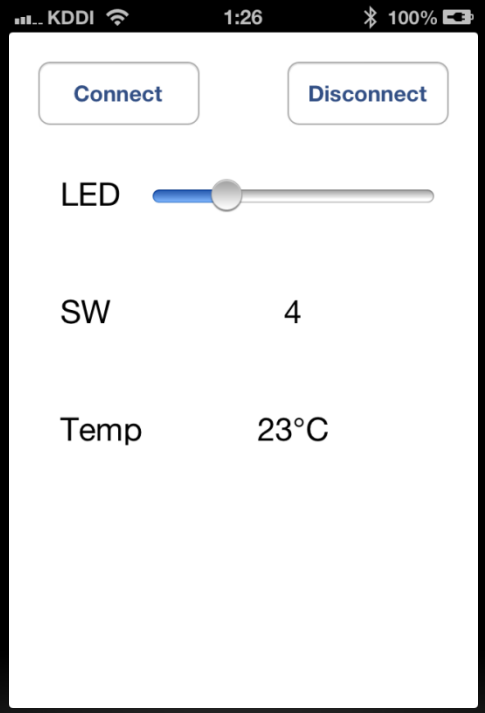
- 接続、切断
  - _Device = [_BaseClass connectService:SERVICE_UUID];
  - [_BaseClass disconnectService:SERVICE_UUID];
- デリゲートの指定（値を読込んだ時の通知）
  - _Device.delegate = self;
- キャラクタリスティッククラスへの参照
  - CBCharacteristic* c = [_Device getCharacteristic:SERVICE_UUID characteristic:CHARACTERISTIC_UUID];
- 書込み
  - NSData* data = [NSData dataWithBytes:&a length:2];
  - [_Device writeWithResponse:c value:data];
  - [_Device writeWithoutResponse:c value:data];

汎用クラスの使い方３

- 読込み依頼、通知依頼
  - [_Device readRequest:c];
  - [_Device notifyRequest:c];
- 通知を受けた時のデリゲート
  - - (void)didUpdateValueForCharacteristic:(BLEDeviceClass*)device Characteristic:(CBCharacteristic*)characteristic;
- 値の取出し
  - NSData* data = characteristic.value;
  - const void* buf = [data bytes];
  - NSInteger len = [data length];

汎用クラスを使ったアプリ作成

- ソースファイルで説明



休憩

BLE指南

- 常に連続してキャラクタスティックを変更する場合はプロパティとしてwriteではなくwrite_no_response=“true”として不必要な通信を抑制する。
- 公開するサービス記述にはadvertise=“true“としてiPhoneに知らせるようにする。
- 自前のサービスやキャラクタスティックのUUIDは必ず生成した128bitのものを使用する。
- キャラクタリスティックの変数長は１6進数で記述したものが反映される。例えば<value type=“hex”>0000000000</value>なら５バイトの変数。
- また、const=“true”を付けていない変数は自分でプログラムから初期化する必要がある。
- 変数長は２０バイトまでが一回の通信パケットで送れるのでパフォーマンスが良い。

BLE112指南

- P1_0はUSB Pull Upでシステムが使用する。（USB使用時）
- P1_7はDC-DCコントロールでシステムが使う。
- ADCの14番はIC内の温度センサとつながっている。
- USB(cdc.xml)を記述しているとTIMER1を使ったPWMが機能しない。
- PWMはTIMER1のALTERNATE=“2”のポートでないとできない。
- GPIOの入力に（チャタリング防止等で）キャパシタを付けると発振するのか動作がおかしくなる。特に割り込みを記述している時。
- P0_0, P0_1を入力にしているとソフトタイマーの割り込み時（もしくはAD変換要求時か？）に不必要な電圧降下を起こす。
- 入力ポートについては外部にプルアップ・プルダウンは付けない方が良い。

BGSCRIPT指南

- イベント内で長い処理が必要ならタイムアウトにならないようにconfig.xml内に<script_timeout value="0" />と記入しておく。
- デバイスがスリープするのを防ぐためにはhardware.xml内に<sleep enable="false" />と記述する。

CORE BLUETOOTH指南

- iPhoneからデバイスを切断しても、接続開始時から約一分間はシステムから切断しない。（デバイス側からみると繋がったままとなる）
- GATTを変更した場合はiPhoneの設定からBluetoothを一旦切ってから入れなおさないとデバイスGATTが反映されない。デバッグ時にこれがはまる。

THANKS

- Downlod
  - http://www.laksmido.com/bledemo/bledemo.zip