

Part1. BLE 編 | 2 章

BLE をとりあえず 体験する

BLE を使った iPhone とデバイスの通信を体験してみましょう。

本章ではツールキット「konashi」を使って、
まず用意されたアプリから konashi を操作してみましょう。
後半では、konashi SDK を使って、自作アプリも作ってみます。

(松村礼央)



2-1. konashiでBLEを体験する

小難しい話はさておき、とりあえずiOSデバイスからBLEデバイスを操作して、アプリによるハードウェアの制御を体験してみましょう。本章では、BLEデバイスとして**konashi**（こなし）と呼ばれるツールキットを使用して解説していきます（図2-1）。konashiは4,298円（税込み）でスイッチサイエンス^{※1}、マクニカオンラインストア^{※2}などで購入することができます。

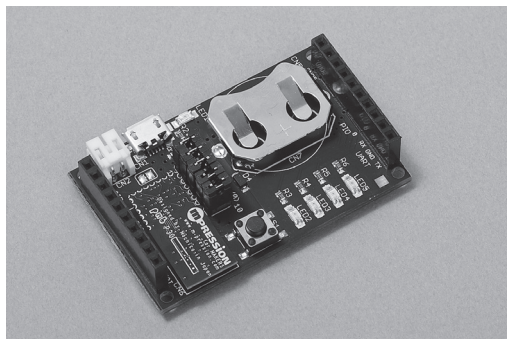


図 2-1 konashi

konashiはBLEでの通信が可能なモジュールを搭載した基板とiOS端末で利用できるSDKで構成されており、アプリ開発者がアプリをプログラミングする感覚でハードウェアを制御できるように設計されています。「操作してみよう」といっても、いきなりプログラミングからでは構えてしまいます。そこで本章ではまず「初級編」としてコードを1行も書かずにkonashiを操作できるアプリ「konashi.js」での操作を体験し、次に「中級編」としてkonashiのSDKを用いて自作アプリからBLEの操作を体験する、という構成で解説を行います。

なお、本章ではBLEを「とりあえず体験する」という点に焦点を当てています。konashiの詳細については、konashiの公式サイトを参照ください。^{※3}

※1 <https://www.switch-science.com/catalog/2102/>

※2 <http://www.macnicaonline.com/SHOP/YEWCPC002.html>

※3 <http://konashi.ux-xu.com/>

2-2. 初級編：konashiをとりあえず体験する

初級編で学ぶこと

初級編では、konashiを動かす上での環境の準備とiOS端末からの制御を体験します。ここではプログラムは1行も書きません。

- konashiをすでに触った経験がある
- 電子工作についての知識がある
- とにかく自作のアプリからBLEを体験したい

という方は、迷わず中級編、あるいは3章、4章まで読み飛ばしてください。それ以外の方は早速はじめてみましょう。

2-2-1. 用意するもの

まず、利用するiOSデバイスがkonashiに対応しているかを確認しましょう。執筆現在でのkonashiに対応したiOSデバイスは表2-1に示すとおりです。

シリーズ名	機種	使用 OS
iPhone	iPhone 4S, iPhone 5, iPhone 5S, iPhone 5C, iPhone 6, iPhone 6 Plus	iOS7.1 ~
iPad	iPad Air,iPad mini,iPad (第4世代),iPad (第3世代)	iOS7.1 ~
iPod touch	iPod touch (第5世代)	iOS7.1 ~

表 2-1 konashi に対応する iOS デバイス

次に、操作用のアプリ「konashi.js」をAppStoreからダウンロードします。キーワード「konashi」で検索すると見つかります。ダウンロードが完了したら、iOSデバイス側の準備は完了です。

konashiの準備に移ります。konashi側では「電源を供給する方法の確認」と「konashiの個体番号の確認」を行います。

konashiの電源を供給する

電源を供給する方法は「CR2032型のコイン電池」、もしくは「USB-microBケーブルによるUSB給電」のいずれかです。コイン電池はコンビニや家電量販店などで購入が可能です。購入の際は型が**CR2032であるかどうか**注意しましょう。他の型ではサイズが合わないため利用できません。USB-microBケーブルは、デジタル一眼やAndoridスマートフォンの充電やデータ転送に用いられるものです。こちらもコンビニや家電量販店で購入できます。

図2-2に電源供給の方法を示します。コイン電池の場合は同図(a)、USB-microBの場合は同図(b)のように電源を供給します。

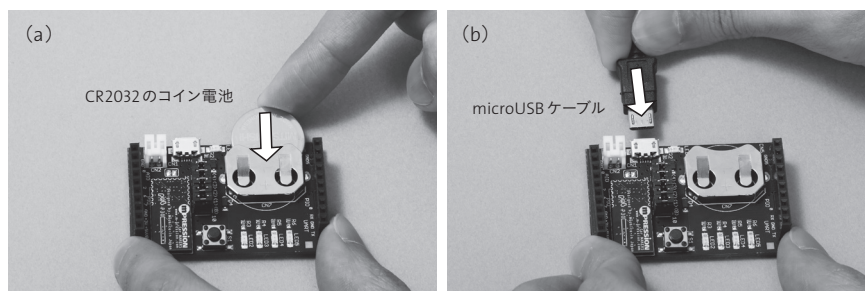


図2-2 konashiの電源供給

正常に電源が供給できていれば、基板上のLED1が緑色に点灯します。点灯ができているかを確認してください(図2-3)。

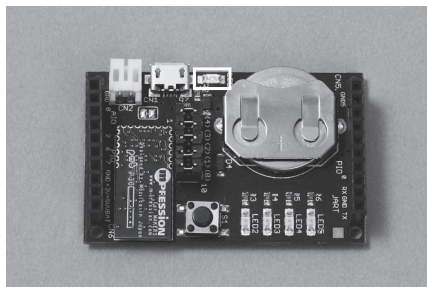


図2-3 電源が供給されると基板上のLED1が緑色に点灯する

konashiの個体番号を確認する

電源が投入できたらkonashiの個体番号を確認しましょう。konashiはBLEを利用して無線で接続するため、自分の手元にあるkonashi以外、たとえば離れた場所にあるkonashiにも簡単に接続できてしまいます。意図しないkonashiとの接続を避けるために、自分のkonashiの個体番号をあらかじめ確認しておきましょう。

個体番号の確認方法はkonashi.jsを利用して行います。konashi.jsを起動すると図2-4 (a) が立ち上がります。画面上の検索で「iosxble」で検索し、登録されている「[iOSxBLE]00-個体番号を確認する」を選択します。選択すると図2-4 (b) の画面に遷移するので、遷移後「▶」の再生ボタンを押すと、プログラムが実行されます。

プログラムを実行すると図2-4 (c) の画面に遷移します。この画面上で「Name Checking」をタップすると、konashiの個体番号を確認することができます。



図2-4 konashiの個体番号を確認する

konsahiに関連する個体番号は「konashi**-****」と表示されます。この番号がお持ちのkonashiの個体番号なので、メモをとっておきましょう。メモを取り終えたら、「Cancel」をタップして終了します。

konashi 2.0 の仕様

konashiが備える機能の一覧を表2-2に示します。konashiはデジタル入出力を5ポート、アナログ入力を3ポート備えており、外部機器との通信機能としてUART、I2C通信用のポートを各1ポート備えています。また、デジタルI/Oが出力モードの際はPWMモード（パルス幅変調信号モード）を最大3出力分利用することが可能です。さらにkonashiとスマートフォン・タブレット型端末間の受信信号強度（RSSI）を計測する機能も備わっており、これらをBLE経由で無線にて利用することができます。また、最大定格は表2-3のとおりです。

機能	搭載数	備考	最大定格値	Min.	Typ.	Max.	単位
デジタル入出力	6pin		動作温度	-30	-	85	°C
アナログ出力	3pin	リファレンス電圧(1.30V)	外部供給電圧 ^{※4}	3.2	-	12	V
UART 通信	1port	2400bps, 9600bpsをサポート	内部供給電圧 ^{※5}	1.8	-	3.6	
I2C 通信	1port	デジタルI/Oと兼用	I/O 供給電圧	1.2	-	3.6	V
PWM出力	3pin	デジタルI/Oと兼用	表 2-3 最大定格				
RSSI 計測機能							

表 2-2 konashi の機能一覧

※4 外部からの電圧供給はバッテリー端子およびUSB-microB 端子より可能です。逆流防止機能付。外部供給を行った場合、内部供給から外部供給へ自動で切り替わります。また、バッテリー端子とUSB-microBを同時に利用している場合、より電圧の高い供給源に自動で切り替わります。

※5 コイン電池により供給される電圧です。

2-2-2. konashiとiOS デバイスをBLEで接続する

電源が投入できたところでkonashiとiOSデバイスをBLEで接続してみましょう。

konashi.jsの起動画面（図2-4（a）参照）で「iosxble」を検索し、登録されている「[iOSxBLE] 01-konashi」とiOS端末をBLEで接続する」を選択します。遷移後「▶」の再生ボタンを押すとサンプルプログラムが実行され、「Find konashi」ボタンをタップすると周辺のBLEデバイスの検索が開始されます。konashiの電源が正常に投入されていればリスト上に番号が表示されま

す。接続リストに自身のkonashiの個体番号を見つけたら、その番号を選択して「完了」をタップします。タップ後、自動的に接続処理が開始されます。接続が完了するとLED2が1秒間隔で点滅します（図2-5）。これでkonashiとiOSデバイスがBLEで接続されました。

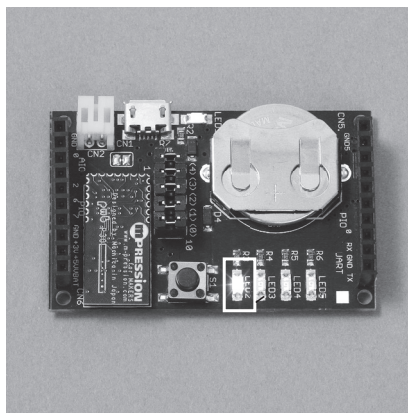


図2-5 接続が完了するとLED2が1秒間隔で点滅する

2-2-3. デジタル入出力を体験する

次のサンプルでは基板上のスイッチとLEDを制御してみましょう。先ほどと同様に「iosxble」を検索し、登録されている「[iOSxBLE] 02-デジタル入出力を体験する」を選択し、「▶」ボタンを押してプログラムを実行します。

konashiと接続後、基板上のスイッチ「SW1」を押してみましょう（図2-6）。「SW1」の押下に合わせて画面上の「ON」「OFF」の文字が変化します（図2-7）。今度は画面上に表示された「LED2」と表示されたボタンをホールドしてみましょう。ホールドに連動して基板上の「LED2」が点灯・消灯します。

このように、点灯/消灯などの0/1の値を扱うハードウェアの機能を「デジタル入出力」と呼びます。konashiには、デジタル入出力が6ポート（PIO0~PIO5）実装されており、その動作を確認しやすいようにPIO1~PIO4の4ポートがLED（LED2~LED5）に、PIO0がスイッチ（SW1）に接続されています。

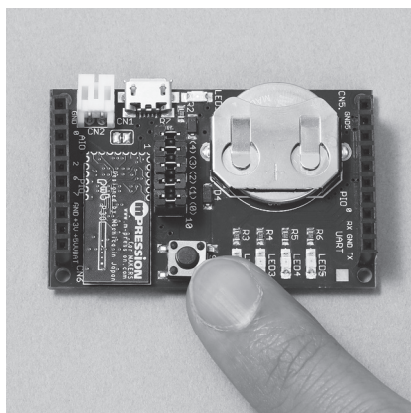


図 2-6 基板上のSW1を押すたびに画面上のOFF/ONの表示が変化

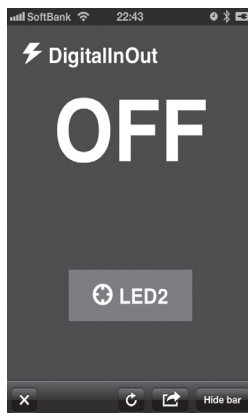


図 2-7 画面上をタップするとLED2が点灯、タップを離すとLED2が消灯

2-2-4. 受信信号強度（RSSI）を計測する

3つ目のサンプルではBLE通信の受信信号強度（RSSI）を計測してみます。このサンプルではkonashiが受信する信号の強度を測定することができます。RSSIを利用することで、iOSデバイスとBLEデバイスのおよその距離を測位したり、いわゆる忘れ物防止タグのようなものを簡単に実装したりすることができます。

それでは実行してみましょう。「iosxble」を検索し、登録されている「[iOSxBLE] 03-受信信号強度（RSSI）を計測する」を選択し、「▶」ボタンを押してプログラムを実行します。接続が完了すると図2-8のような画面が現れます。

画面の赤い部分がバーとしてRSSIの値と連動しており、iOSデバイスがkonashiに近づくほど表示上の数値が小さくなります。実際にiOSデバイスを近づけたり、遠くへ離したりして数値がどのように変化するか試してみましょう。

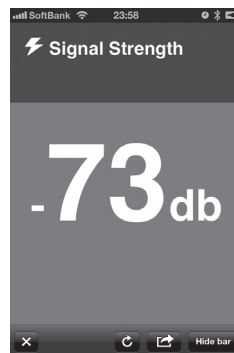


図 2-8 RSSI が1秒おきにアップデートされ、値の表示とバーの描画が更新

2-2-5. Web サービスと連動させる

最後のサンプルでは、Webサービスと連携するアプリからBLEを利用してハードウェアを連動させてみます。このサンプルでは、Webサービス「SoundCloud」と連携するアプリと konashi の基板上のスイッチ「SW1」を連動させることで、スイッチ「SW1」から音楽の再生/一時停止/曲スキップを行います。

SoundCloudとは、音楽や音声ファイルを共有する Web サービスです。YouTubeの音楽版というイメージで捉えるとわかりやすいかもしれません。多くのプロミュージシャンが自身の作品のリミックスなどを積極的に公開しており、ユーザー登録するだけで、それらの楽曲を簡単に試聴することができます。筆者もこのサービスをよく利用しており、本サンプルはこのSoundCloudをハンズフリーで利用できるように、と考えて作成したものです。

では、実行してみましょう。「iosxble」を検索し、「[iOSxBLE] 04-Webサービスと連動させる」を選択し、「▶」ボタンを押してプログラムを実行します。まず、これまでのサンプルと同様に [Find konashi] で konashi を接続します。次に画面上に表示される「Load Music」をタップし、あらかじめ設定されたキーワードを用いて SoundCloud からトラックの情報をロードします。本サンプルでは、検索のキーワードとして「Dead Mou5e」というアーティストを用いています。通信状況によってロードに要する時間は異なりますが、準備が整い次第、再生が開始されます（図2-9）。再生中にスイッチ「SW1」を短く押すと音楽が停止します。逆に音楽が停止中に短く押すと音楽が再生します。また、音楽の再生・停止状態に関わらず長押しすると次の曲にスキップを行います。

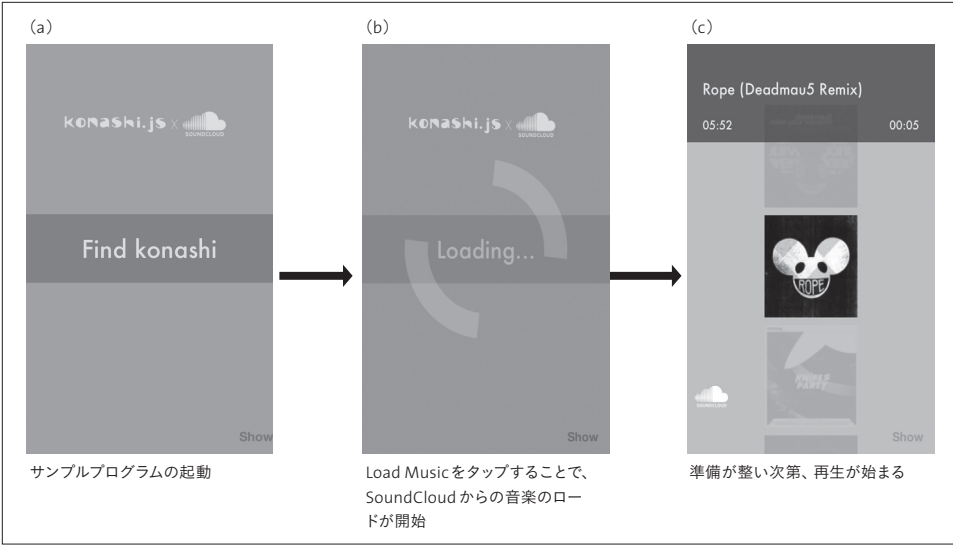


図2-9 konashiとWebサービス「SoundCloud」を連動させるサンプルプログラム

2-3. 中級編：konashiをiOS-SDKを利用して制御する

中級編で学ぶこと

中級編では、konashiのiOS-SDK (konashi-ios-sdk) を利用して制御を体験します。iOS-SDKはCore Bluetoothフレームワークをラップしたライブラリで、Core Bluetoothを意識せずに、iOSからkonashiを制御することができます。中級編では、まずはCore Bluetoothまでは踏み込まずに、iOSプログラミングによるBLEを活用したサンプルを体験します。

サンプルは次の3つです。

- 2-3-1. 周辺のkonashiを検索する
- 2-3-2. 接続完了時にLEDを点灯させてみる
- 2-3-3. 受信信号強度 (RSSI) を計測する

それでは、さっそくiOS-SDKを利用してkonashiを制御してみましょう。

2-3-1. 周辺のkonashiを検索する

初級編でのサンプルからわかるように、konashiを制御するにはkonashiを見つけなければ始まりません。というわけで、まずは周辺のkonashiをiOS-SDKから検索してみましょう。

◎ ここで学ぶこと

- [Konashi initialize] (konashiクラスのインスタンスの生成/初期化)
- [Konashi find] (konashiの検索/初期化)

実装手順

1: konashi-ios-sdkの入手

gitから最新版のkonashi-ios-sdkをcloneしておきます。

```
$ git clone https://github.com/YUKAI/konashi-ios-sdk.git
```

2: ライブラリの追加とインポート

konashi-ios-sdkの本体はkonashi-ios-sdk/Konashi/Konashiにあるので、こちらをimportします。

3: konashiを初期化する

konashiを利用するためにinitializeメソッドからインスタンスを生成し、初期化を行います。以下のように、アプリ全体がインスタンス化された直後に呼ばれるviewDidLoadなどの処理中に記述しましょう。

```
[Konashi initialize];
```

メソッドの戻り値は

- 初期化した場合：KonashiResultSuccess
- すでに初期化されていた場合：KonashiResultFailure

となります。

4: 周辺のkonashiを検索する

次に、iOS端末周辺のkonashiを検索します。konashiの検索はfindメソッドを用います。メソッドはinitializeの直後に記述します。

1

2

3

4

5

6

7

8

9

10

11

12

```
[Konashi find];
```

メソッドが正常に実行されれば、リストに列挙される形で周辺の konashi のリストが自動的に表示されます。列挙された konashi を選択し、タップすると自動的に接続を行います。

メソッドの返り値は、

- konashi の探索が開始された場合：KonashiResultSuccess
- すでに konashi との接続が完了していた場合：KonashiResultFailure

となります。

試してみる

サンプル	KonashiFind
------	-------------

konashi の電源を入れ、上記の手順で実装したアプリを実行してみましょう。起動後、自動的に find メソッドが実行され、手元の konashi が **konashi**-*** という形で検索されたら、該当の番号をタップします。すると無事、接続が完了します。

2-3-2. 接続完了時にLEDを点灯させてみる

先のサンプルでは、接続はできますが見た目に何の変化もないため、接続したかどうかが一見してわかりません。そこで、konashi への接続が完了後、基板上に接続されている LED を点灯させて、接続を確認してみましょう。

◎ ここで学ぶこと

- [Konashi addObserver: (id)notificationObserver selector:(SEL)notificationSelector name:(NSString*)notificationName] (イベントのオブザーバの生成)
- [Konashi pinMode:(int)pin mode:(int)mode] (konashi のデジタル入出力ピンの初期化/設定)
- [Konashi digitalWrite:(int)pin value:(int)value] (konashi デジタル出力)

1: konashiの接続の完了通知を受け取る

接続が完了したことをkonashiから通知として受け取ることができます。konashiではこれをイベント呼びます。

konashiはiOSデバイスと無線で接続されています。そのため、「接続が完了した」などの状態を知るためには必ずiOSデバイスとの通信を要します。このとき、たとえば状態が明らかになるまでkonashiのスレッドが待機したとしたらどうなるでしょうか？ 答えは簡単で、スレッドがロックしてしまいます。これを回避するために、konashiではイベントという形でiOSアプリに通知を送ることで、非同期に状態を知ることができるように設計されています。

konashiを使うにあたっての基本的なイベントの遷移は図2-10のようになります。図から、konashiの接続の完了を知るにはKonashiEventReadyToUseNotificationイベントをキャッチすればよいことがわかります。

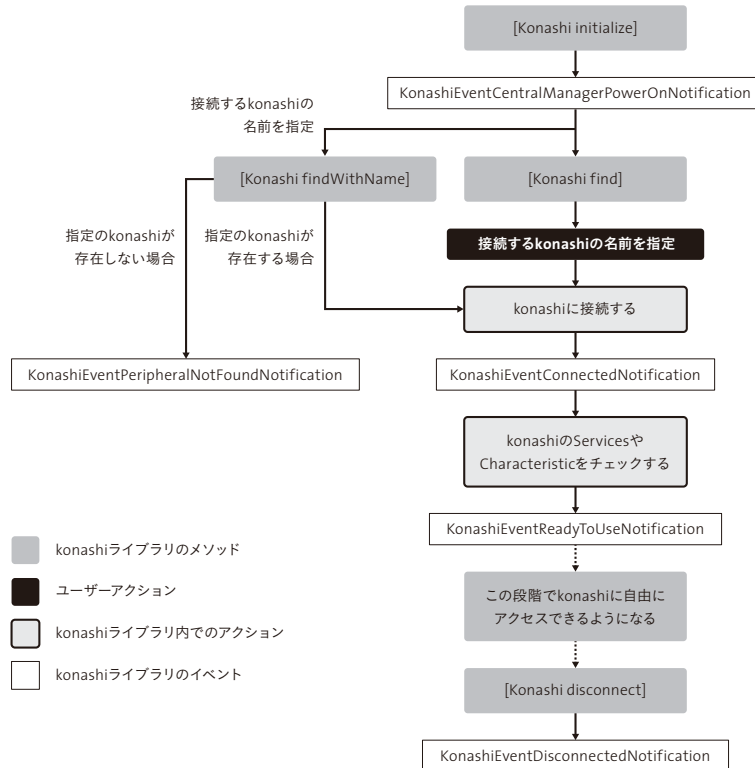


図 2-10 konashiの基本的なイベントの遷移

konashi のイベントのキャッチは addObserver メソッドを利用します。

```
[Konashi addObserver: (id)notificationObserver  
                 selector: (SEL)notificationSelector  
                 name: (NSString*)notificationName];
```

第1引数である notificationObserver にはオブザーバを指定します。通常は、konashi 自身なので self を指定します。第2引数である notificationSelector にはイベント発火時に呼び出されるメソッドを指定します。第3引数である notificationName にはキャッチしたいイベント名を記述します。

KonashiEventReadyToUseNotification イベントをキャッチするようなオブザーバを実装すると、次のようになります。イベント発火時に呼び出されるメソッドは、とりあえず ready として定義しています。

```
[Konashi addObserver:self selector:@selector(ready) name: KonashiEventReadyToUse  
Notification];
```

なお、addObserver メソッドの実装箇所はどこでもかまいません。ここで viewDidLoad 中の initialize 後に記述します。

2: デジタル入出力ピンの初期化 / 設定

初級編で学んだように、konashi ではデジタル入出力ピンが8つ搭載されています。デジタル入出力は入力にも出力にもできるため、どちらで利用するかをまず宣言する必要があります。アプリ「KonashiFind」では「LEDを光らせる = 出力として使う」ということにするため、PIO1を出力に設定します。

設定には pinMode メソッドを利用します。メソッドの第1引数 pin には PIO の番号を、第2引数 mode には、

- 入力の場合 : KonashiPinModeInput
- 出力の場合 : KonashiPinModeOutput

を代入します。ここではPIO1を出力としてLED2を駆動したいのでpinにKonashiDigitalIO1、modeにKonashiPinModeOutputを渡します。

```
[Konashi pinMode: KonashiDigitalIO1 mode: KonashiPinModeOutput];
```

3: デジタル出力でLEDを点灯させる

PIO1を出力に設定できたら満を持して、LED2を点灯させましょう。LED2を点灯させるにはPIO1をHIGHに出力します。PIOの出力の設定はdigitalWriteメソッドを利用します。

```
[Konashi digitalWrite:(int)pin value:(int)value];
```

メソッドの第1引数pinにはPIOの番号を、第2引数valueにはKonashiLevelHighもしくはKonashiLevelLowのいずれかを代入します。ここでは、PIO1からKonashiLevelHighを出力したいので、次のように記述します。

```
[Konashi digitalWrite:KonashiDigitalIO1 value:KonashiLevelHigh];
```

4: 接続完了時の挙動を実装する

先のオブザーバの実装で、イベントが発火した際、readyメソッドが呼ばれるように定義しました。ここでPIOの初期化/設定を行い、LED2を点灯するように実装すれば、接続完了をLED2の点灯で知ることができます。readyの実装例を次に示します。

```
- (void)ready
{
    // PIOの初期化/設定を行います
    [Konashi pinMode: KonashiDigitalIO1 mode: KonashiPinModeInput];

    // PIO1 をHIGHにしてLEDを駆動
    [Konashi digitalWrite:KonashiDigitalIO1 value:KonashiLevelHigh]; // PIO1をHIGHに
}
```

試してみる

サンプル	KonashiPioDrive
------	-----------------

konashiの電源を入れて、アプリを実行してみましょう。誤りなく実装できていれば、konashiへの接続直後、LED2が点灯します。

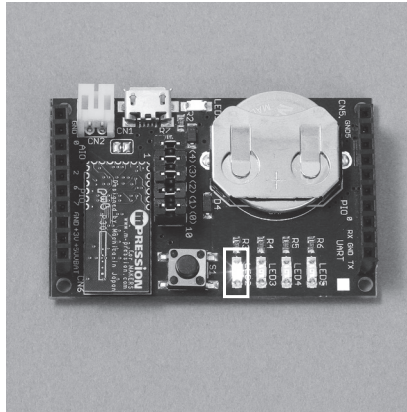


図 2-11 接続するとLED2が点灯する

2-3-3. 受信信号強度（RSSI）を計測する

接続まで確認できたので、本章の最後としてkonashiから受信した信号の強度(RSSI:Receive Signal Strength Indicator)を取得してみましょう。RSSIを取得することで「konashiに対してiOSデバイスがおおよその領域に存在するのか」、「konashiとiOSデバイスが離れたかどうか」などを、ある程度の精度で定量的に計測することができます。

◎ ここで学ぶこと

- [Konashi signalStrengthReadRequest] (RSSI取得のリクエストを送信する)
- KonashiEventSignalStrengthDidUpdateNotification (RSSIの値が更新された際に発火するイベント)
- [Konashi signalStrengthRead] (更新されたRSSIの値を取得する)

実装手順

1：一定の周期でRSSI取得のリクエストを送信する

RSSIを取得するためにRSSIの取得リクエストを送信する必要があります。書き込みの場合とは異なり、読み込みの場合、konashiではkonashiに対して一度、読み込みのリクエストを送信する必要があります。

RSSIの取得リクエストの送信は以下のメソッドを利用します。

```
[Konashi signalStrengthReadRequest];
```

定期的にRSSIを取得するためには、NSTimerでタイマを作成し、一定時間おきにsignalStrengthReadRequestメソッドを実行すれば問題ありません。たとえば、signalStrengthReadRequestメソッドを実行する関数をonRSSITimerと定義し、NSTimerで周期実行するためには、次のように記述します。タイマの作成は前節の「接続完了時にLEDを点灯させてみる」で利用したready関数に記述し、konashiとの接続が確立されたタイミングで実行するようにコーディングしておきます。

```
- (void)ready
{
    NSLog(@"READY");

    // 電波強度タイマ
    NSTimer *tm = [NSTimer
        scheduledTimerWithTimeInterval:0.1
        target:self
        selector:@selector(onRSSITimer:)
        userInfo:nil
        repeats:YES
    ];

    [tm fire];
}

- (void) onRSSITimer:(NSTimer*)timer
{
    [Konashi signalStrengthReadRequest];
}
```

2：RSSIの取得完了イベントを取得する

リクエスト送信後、RSSIの取得の完了はイベントとして受け取ることができます。RSSIの取得の完了イベント `KonashiEventSignalStrengthDidUpdateNotification` で定義されており、`addObserver` メソッドを利用して取得することができます。今回はイベント発火時に呼び出す関数を `updateRSSI` として定義しています。

```
[Konashi addObserver:self
 selector:@selector(updateRSSI)
 name:KonashiEventSignalStrengthDidUpdateNotification];
```

3：RSSIの取得時の挙動を実装する

`updateRSSI`の実装例を次に示します。更新したRSSI値の取得は `signalStrengthRead` メソッドを利用します。実行するとログ中に取得されたRSSIの値が記録されます。RSSIの値は `konashi` とiOSデバイス間の距離が近ければ-40dBほど、遠ければ-90dBほどになります。

```
- (void) updateRSSI
{
    NSLog(@"READ_STRENGTH: %d", [Konashi signalStrengthRead]);
}
```

試してみる

サンプル	KonashiReadRSSI
------	-----------------

`konashi`の電源を入れてアプリ「KonashiReadRSSI」を実行してみましょう。誤りなく実装できていれば、`konashi`への接続直後、ログ中に次のような値が現れるはずです。値が取得できたら、`konashi`に対してiOSデバイスをさまざまな方向に向けてみて値がどのように変化するかを確認してみましょう。

```
> READ_STRENGTH: -40  
> READ_STRENGTH: -45  
> READ_STRENGTH: -52
```

iPhoneをkonashiに近づけるとRSSIの値が上昇し、遠ざけると減少します。値の増減が確認できたら、次はkonashiを手で覆ってみて、値がどのように変化するかを確認してみましょう。水分を多く含む人体でkonashiを遮蔽することで、電波が届きにくなり、RSSIの値も減少します。

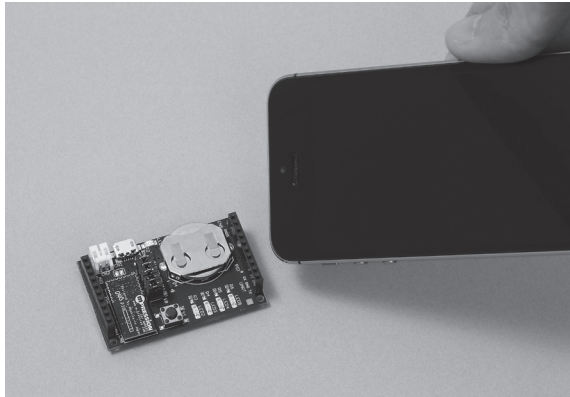


図 2-12 konashiに近づけるとRSSIの値が変化する

この結果からわかるように、見通しの良い環境であればRSSIの値からある程度の距離は推定することができますが、人が密集している場所などではRSSIの値と距離は必ずしも比例するわけではないことがわかります。ウェアラブルデバイスやビーコンなどでRSSIから距離計測を行う場合はこの点に注意する必要があります。

本章では、BLEを活用したツールキットkonashiを用いて、iOSデバイスとBLEデバイスとの連携を体験しました。BLEではサービスを実現するためには、ある程度ハードウェアを扱える、つまり本章で体験した電源の供給などに関する知識が最低限、必要となります。konashiについては4章でも引き続き利用していくので、不明な点があれば本章を適宜、読み返すと良いでしょう。

次の章では、いよいよBLEの仕様について説明していきたいと思います。

フィジカルコンピューティング・ツールキット「konashi」とその開発の背景

konashiは、スマートフォンとハードウェアを連携するシステムを簡単にプロトタイプできるツールを実現する、ということを目指し筆者 松村を筆頭に、たけいひでゆき氏（現 beatrobo Inc.）、田所祐一氏（現 東京工業大学）、菊谷侑平氏（現 東京工業大学）、竹元翔太氏（現 iXs Research Corp.）の手によって開発したツールキットです。^{*6}

そもそもの着想は2010年に、筆者 松村とたけい氏が趣味で開発した1台の小型ロボット「monaka」にさかのぼります。スマートフォン黎明期の当時、monakaはスマートフォンと連携する携帯型の小型ロボットとして開発しました。^{*7} その際、開発していたスマートフォン側のアプリケーションとハードウェアの連携を行うクラシックBTを利用したBluetoothによるロボット制御ボードがkonashiの原型となっています。

この3年後、筆者 松村とたけいひでゆき氏が当時在籍していたユカイ工学にてBLEモジュールをベースに、iOSとハードウェア連携のボードとして再設計したことでkonashiが生まれました。



図 2-13 Androidスマートフォン連携型小型ロボット「monaka」

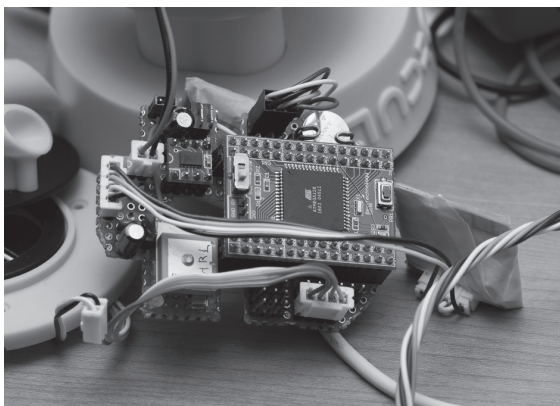


図 2-14 konashiの原型となった monaka のボード

筆者と菊谷氏によってBLEを活用したiOS連携のボードとして再設計するのと併行し、konashiはスマートフォンを中心としたシステムのためのツールとして活用できるようObjective-C（iOS-SDK）やJavaScript（konashi.js^{*8}）用の開発環境の実装が、たけい氏、田所氏を中心になされました。このとき、我々が参考ににしたのがフィジカルコンピューティングと呼ばれる概念です。

いわゆる一般的なコンピュータの入出力はマウス、キーボードなどの入力とディスプレイなどの出力で構成されています。我々はマウスやキーボードを介して処理した情報をデータとして入力し、コンピュータで処理した情報をディスプレイに出力することで、コンピュータとインタラクションを行い、日々デジタルコンテンツを体験し、消費し、そして創造しています。しかし、このデジタルコンテンツはそれら入出力系によって大きく制約を受けているのではないかと。つまり、これら既存の入出力系に加え、我々の身の回りの物理的な世界と入出力系をより多様に接続することができれば、人とコンピュータとの新しいインタラクションが生まれ、デジタルコンテンツに新しい広がりが見られるのではないかと……。このような考えをもとにして、ニューヨーク大学のTom Igoe氏、Dan O'Sullivan氏によって提案された概念が「フィジカルコンピューティング」です。

このフィジカルコンピューティングの提案の背景には「物理的な世界とデジタルコンテンツを結びつける」という考えがみえて、2000年頃にMITの石井裕氏が提唱したタンジブルビットとの関連をみてとることができます。このような思想的背景からうかがえるように、フィジカルコンピューティングにおいては、コンピュータに関してハードウェア、ソフトウェアの知識、デジタルコンテンツやコンピュータと人とのインタラクションに関する哲学的な思想を、横断的に扱うことが必要であることがわかります。実際、Tom Igoe氏、Dan O'Sullivan氏の教育プログラムにおいても、ソフトウェア、ハードウェアの基礎から学ぶ内容となっていました。ここにフィジカルコンピューティングの困難さがあります。しかしこの困難さが、その後、フィジカルコンピューティングのためのツールキット登場のキッカケとなります。

小林茂氏（現 情報科学芸術大学院大学（IAMAS）教授）が中心となって開発したフィジカル・コンピューティングのためのツールキット「Gainer」^{*9}と、イタリアのデザインの専門学校IDI Ivrea (Interaction Design Institute Ivrea)において、Massimo Banzi氏が中心となって開発したツールキット「Arduino」^{*10}は、同じ2005年に発表されました。2007年に登場した「Arduino Fio」は、IAMAS 小林茂氏が2007年「未踏ソフトウェア創造事業」においてSparkFun Electronicsと協力して開発したArduinoの正式な互換機種です。フィジカルコンピューティングのためのフレームワーク「Funnel I/O」に対応しているほか、XBeeと呼ばれる無線通信規格への対応、電源の充電機能などを備えており、ArduinoやGAINERにおいて課題として残っていた点を克服しています。

GAINER、Arduino、Arduino Fio、これらのツールキットが2000年代後半にMIT

メディアラボNeil Gershenfeld氏が提唱したパーソナルファブリケーションの運動の一翼を担い、Chris Anderson氏の提唱するメイカームーブメントへとつながります。

それらのムーブメントが盛り上がりを見せるのと並行し、冒頭でも述べたように2010年代以降、小規模なパソコン並の処理性能を備えたスマートフォン・タブレットが台頭することとなります。これに伴いスマートフォン・タブレットと連携するツールキットの要求が高まってきました。これにmonakaをベースとして応え、設計したものがkonashiなのです。

※6 konashi 2.0以降の開発はユカイ工学株式会社によります。

※7 Android-Robo Controlled by Android Mobile Phone <http://youtu.be/N1Kpc-qgtbQ>

※8 konashi.jsのjsdo.it連携機能については株式会社カヤックの協力によります。

※9 <http://gainer.cc/>

※10 <http://www.arduino.cc/>