第2章 ハードウェアに極力依存しないデバイス・ドライバを実現

embedded-halでデバイス・ドライバを抽象化して実装する

高野 祐輝

リスト1 Cargo.tomlの修正

```
[dependencies]
embedded-hal = "=1.0.0-alpha.9"
rppal = { version = "0.14", features = ["hal"] }
bitflags = "1.3"
```

ここでは特集1第1部第3章でRustとラズベリー・パイ4を使って実装した環境モニタを，embedded-halを用いてより抽象的にデバイス・ドライバを実装する方法を解説します．

embedded-halとはRustのライブラリの1つであり，組み込みシステム向けのハードウェア抽象化を行うものです．halは，Hardware Abstraction Layerの略です．ラズベリー・パイ4で利用するのみならば，特集1第1部第3章のようにライブラリrppalを利用するだけで可能です．embedded-halを用いることで，ハードウェアに極力依存しないデバイス・ドライバを記述可能になります．

別の言い方をすると，基板のベンダはembedded-halでI²CやSPIを抽象化すればよく，ペリフェラル・デバイスのベンダはembedded-halで抽象化されたインターフェースに対して実装すればよくなります．

embedded-halを用いた抽象化

● embedded-halのメリット

rppalを用いると容易にI²Cなどのデバイスを利用できます．しかし，特集1第1部第3章で説明したようなrppalを使ったプログラムはラズベリー・パイ4に特化してしまっており，ラズベリー・パイ4以外の他のボードへの流用が難しいという欠点があります．

そこで，ここではembedded-halというライブラリを用いて，I²Cなどのデバイス・ドライバを抽象的に記述します．

embedded-halは組み込み向けのトレイトの集合と言えるようなライブラリで，現在バージョン1リリースに向けて開発が進められています．2022年2月12日の最新バージョンは，1.0.0-alpha.9です．

● トレイトはRustの抽象化機能

トレイトとは，Rustの抽象化機能の1つであり，JavaのInterface，C++の純粋仮想関数，Haskellの型クラスに相当します．

例えば，「哺乳類」というトレイトと，「猫」という型を定義したとします．このとき，「猫」に「哺乳類」というトレイトを実装することで，「猫は哺乳類である」と宣言できます．

つまり，トレイトは「XはYである」といった，述語の一種であると考えることができます．

embedded-halによる抽象化では，「猫」と具体的な何かに対して考えるのではなく，「哺乳類」と抽象化して考えることに似ています．具体例は説明していくので，ここではあまり深く考えず読み流す程度で結構です．

ここでは，rpihalというクレートを作成し，そこに実装していきます．

rpihalの作成は以下のようにします．

```
$ cargo new rpihal
```

ファイル構成は，特集1第1部第3章で作成したrpidevと同じなので割愛します．

embedded-halを利用するために，Cargo.tomlを**リスト1**のように修正します．

今回は，開発版のembedded-halを利用するため，バージョンに=1.0.0-alpha.9と指定します．また，rppalが提供するembedded-halを用いた抽象化プログラムを利用するため，featuresにhalを追加します．

● embedded-halのI2cトレイト

embedded-halでは，I²Cデバイス用のトレイトはembedded_hal::i2cに定義されています．特に重要なのは，ErrorTypeとI2cトレイトです．

ErrorTypeトレイトは**リスト2**のように定義されています．type Error: Errorは，左側のtype Errorで，「ErrorTypeトレイトの関連型であるError型」を，右側の：Errorで，「この型はErrorトレイトを実装していなければならない」を意味しています．Errorトレイトは，同じくembedded_hal::i2c