フルスクラッチのすすめ

自作キーボードのファームウェアを題材に

MURAOKA Taro (KoRoN, @kaoriya)

2022/12/31 発行

はじめに

本冊子は、RP2040をターゲットにして「自作キーボードのファームウェア」をフルスクラッチで書く方法を紹介し、組込みプログラムの考え方に親しんでもらおうという読物のパイロット版です。パイロット版であるが故に書きたいことを優先し、読者に要求する知識レベルの想定がトピックごとにバラバラで、一貫性に欠けることに留意してください。

RP2040というマイコンはRaspberry Pi Picoのマイコンとして知られています。そのフルスクラッチ開発では Pico SDK (https://github.com/raspberrypi/pico-sdk) を利用します。Windows で Pico SDK を使うには MSYS2 (https://www.msys2.org/) というUNIX互換レイヤを利用できます。もちろんWSLも利用可能なはずですが、筆者はもっぱらMSYS2を利用しています。

TIP	正確にはMSYS2上で動く MinGW64 (https://www.mingw-w64.org/) を使っています。

以下では Pico SDK とそれに必要なツールがインストール済みであるものとして説明していきます。

NOTE	将来、インストール方法を説明する

また実際に利用可能なキーボードファームウェアの例として YUIOP60Pi (https://github.com/koron/yuiop60pi) も参照してください。

組込みプログラムの基本

まずは最もオーソドックスな組込みプログラムの形を見てもらいましょう。以下のソースコードは本当にシンプルな何もしない、しかし有効な組込みプログラムです。

```
①
int main() {
    ②
    while (true) {
        ③
    }
}
```

① 必要なヘッダーの `include` 文の追加

② 初期化ルーチン

③ メインループ

上で示した1, 2, 及び3の部分に適切な手続きを記述すれば、この組込みプログラムはUSBキーボードや、その他の組込み機器として振舞うようになるのです。この3箇所を以下では基本部分と呼びます。

初期化ルーチン

初期化ルーチンでは、例えばUSB HIDデバイスとしての初期化手続きを行います。一般的に、RP2040自身や基板上に実装された各種ペリフェラル(部品)の初期化、利用しているライブラリの初期化、さらにファームが使うメモリの初期化など、メインループで利用するあらゆるものの初期化をこのタイミングで行います。

メインループ

メインループでは、キーボードにとって必須なタスクを順番に実行します。具体的にあげると、まずキーマトリックスをスキャンし、次にキーマトリックスの状態をキーコードへ変換しUSB HIDレポートとして送信します。

それらタスクをひたすら繰り返すことで、組込みプログラムは自作USBキーボードとして機能します。また仮に基板上にBluetoothのモジュールが実装されているならば、USBの代わりにBluetoothを初期化してBluetooth経由でHIDレポート送信するようにすれば、その組込みプログラムはBluetoothキーボードになるわけです。

キーボードとしてのタスク

以下ではキーボードとしての各々のタスクをどのように記述するのか、より細かく見ていきましょう。各タスクにはそれぞれ、前述の1.include文、2.初期化ルーチン、3.タスク実行、それぞれの基本部分に少しずつコードを追加することになります。

USB HIDデバイス

RP2040をUSB HIDデバイスとして動作させるのには、TinyUSBというライブラリを用います。この TinyUSB は Pico SDK に付属しているのでコードを書くだけで済みます。

以下に示すコードは USB HID デバイスとして必要なものを示しています。3つの基本部分に僅か1行ずつ書き足すだけで、この組込みプログラムは USB HID デバイスとなります。

NOTE	将来、USBデスクリプタを記述する必要があることを追加する

```
①
#include "tusb.h"

②
tusb_init();

③
tud_task();
```

キーマトリックススキャン

キーマトリックススキャンは、マトリクスに利用する全GPIOを入力モードかつ、プルアップかつ、出力をLowで初期化します。

またタスクにおけるスキャンする信号線の選択は当該GPIOを出力モードに切り替えることで行います。その状態で少し待ってからGPIOの全状態を読み込んだ後、当該GPIOを入力モードに戻します。あとはこれをスキャン側の全GPIOに対して行えば、キーマトリックスのスキャンは完了です。

上述の説明をおおよそのコードで示すと以下の通りとなります。

```
①
#include "hardware/gpio.h"

②
// 以下をマトリクスに使用する全GPIOについて繰り返す
uint8_t gpio;
gpio_init(gpio);
gpio_set_dir(gpio, GPIO_IN);
gpio_pull_up(gpio);
gpio_put(gpio, false);

③
// 以下をスキャンする全GPIOについて繰り返す
uint8_t gpio;
gpio_set_dir(gpio, GPIO_OUT);
busy_wait_us_32(1); // GPIOの状態が反映されるのを少し待つ
uint32_t status = gpio_get_all(); // 全GPIOを一度に取得する
gpio_set_dir(gpio, GPIO_IN);
```

USB HIDレポートへの変換と送信

USB HID レポートへの変換と送信は、キーマトリックススキャンで得た `status` を USB HID のキーコードに変換し、レポートとして送信します。

以下に示すコードは同時押しに対応してない、極めて単純に、押されたキーに応じてアルファベットの A, B, C を送信するだけのものです。`status` の押下したキーに対応するビットは `0` になり、押してないキーに対応するビットは 1 になります。それらを検出して適切なキーコードに変換して送信しています。

```
①
#include "tusb.h"
#include "usb_descriptors.h"

③
uint8_t modifiers = 0;
uint8_t codes[6] = {0};
if ((status & (1 << 10)) == 0) {
    codes[0] = 0x04; // 'A'
} else if ((status & (1 << 11)) == 0) {
    codes[0] = 0x05; // 'B'
} else if ((status & (1 << 12)) == 0) {
    codes[0] = 0x06; // 'C'
}
if (tud_hid_n_ready(ITF_NUM_HID)) {
    tud_hid_n_keyboard_report(ITF_NUM_HID,
        REPORT_ID_KEYBOARD, modifiers, codes);
}
```

実用上は同じレポートを送らないためのコード等を、これらに追加します。

今後の発展: その他のタスク

USBキーボードとしては、以上のコードを組み合わせれば、最低限の機能を持ったものとして成立します。しかし実用上はもう少し機能がある方が良いかもしれません。例えばレイヤー機能であったり、RGB LED (Neo Pixel) の制御機能だったりです。

レイヤー機能

レイヤー機能は、ファームウェア内部に現在有効なレイヤという情報を持ち、その状態に応じて、キーマトリックスのあるビットが `0` だった時に送信するキーコードを変えることで、実現できます。また有効なレイヤーを変更するための特殊キーを取り扱えるようにする必要もあるでしょう。

RGB LEDの制御

RGB LEDの制御は pico-exampls (https://github.com/raspberrypi/pico-examples) の中に制御用のサンプルがあります。それを利用すればRGB LEDに対して制御データ≒光り方のデータを簡単に送信できます。あとは3のタスクとして、光り方のデータを更新し、RGB LEDへと送出するだけです。

おわりに

本冊子は、フルスクラッチで自作キーボードファームウェアを書く、ということについてどの程度の興味を持ってもらえるかを測るために、パイロット版として執筆しました。興味を持ってもらった方で質問などあれば、Twitter や GitHub でお尋ねください。

https://twitter.com/kaoriya
https://github.com/koron/an-encouragement-of-full-scratching

以下には、参考になりそうなリンクを示しておきます。

フルスクラッチの作例

https://github.com/koron/yuiop60pi

上記作例の回路図

https://github.com/koron/yuiop/tree/main/yuiop60pi

Pico SDK

https://github.com/raspberrypi/pico-sdk

Pico Examples

https://github.com/raspberrypi/pico-examples

*QMK (*有名なキーボードファームウェア*)*

https://github.com/qmk/qmk_firmware/