

第1章

STM32向けを流用して、
ラズベリー・パイ Pico と RZ マイコンで試す

開発の準備…環境構築と ビルド&拡張の手順

関本 健太郎



写真1 ラズベリー・パイ Pico を使って MicroPython の拡張にトライ

第1章ではデバッグできるようになるところまで

● 速度や使い勝手が完璧ではない MicroPython

MicroPython を使うと、素早くプログラムを作成し実行できる反面、特定のハードウェア・リソースにアクセスできなかったり、MicroPython の中間言語によるオーバーヘッドで実行速度が期待に及ばなかったりすることがあります。

このような場合に、MicroPython のモジュールの一部を、アセンブラや C、C++ で記述することで、アクセスできるハードウェア・リソースを追加したり、処理速度を改善したりできます。

そこで本稿では、ラズベリー・パイ Pico (以降、Pico) を例に、MicroPython の拡張モジュールの仕組みを解説し、拡張モジュールの作成方法について、実装例を示しながら説明します(写真1)。

さらに、第5章では RZ マイコン(ルネサス エレクトロニクス)を搭載したマイコン・ボード GR-MANGO 向けの MicroPython に、CPU レジスタ・アクセス・モジュール、LCD およびカメラ処理クラスの実装方法を紹介し、実装したソースコードは本書ウェブ・ページから提供します。

MicroPython の内部アーキテクチャ

MicroPython の作者の YouTube⁽¹⁾ によると、STM32 向けの MicroPython の内部アーキテクチャは、図1 のようになります。ちなみにこの動画にはアーキテクチャの説明だけでなく、MicroPython を高速化するヒントがたくさん説明されています。

● 幾つかのブロックに分けて考える

MicroPython の内部アーキテクチャは、大きく分けると次の通りです。

- マイコンのブート機能
- MicroPython のコンパイラ機能
- バイト・コードを実行する MicroPython の仮想マシン
- ネイティブ・コードを直接実行する Just-In-Time 機能(一部のマイコンでのみサポートされている)
- MicroPython のランタイム・ライブラリ機能
- 外部バインディング・モジュール

● 第9部でトライすること

本稿では、図1の右側に示すモジュール追加による機能拡張をどのように行うのかを、幾つかの例を示しながら説明していきます。ちなみに、本稿の範囲外ですが、マイコン依存部は

- マイコンのブート機能
- ランタイム・ライブラリ中の MCU 周辺機能

です。従って MicroPython を特定のマイコンに移植する際は、大きく分けてこの2つの部分を実装する作業になります。

特定のマイコンで MicroPython の仮想マシン機能を動作させるだけなら、マイコンのブート部分を実装するだけなので、半日もあれば実装できるでしょう。例えば RP2040 マイコン向けの rp2 ポート(特定のマイコン向けに移植された MicroPython をポートと呼ぶ)では、MicroPython は、Pico 用の SDK をベースに、ベアメタル、つまり OS のない環境で動作しています。

■ダウンロード・データについて■

データは当時の MicroPython のビルド全体を zip ファイルにしたので、ビルド、実行はそのまま動作するはずですが、ただし、最新の MicroPython とはファイル構成は一部変更されたり、API が変更されたりしていますので、現状のものとは詳細部分で異なっています。