特集2

第3章 ESP32-C3用プログラムのコーディング/コンパイル/書き込みに使う各ツールのインストール

ステップ1…開発環境の構築

中林 智之

この章ではESP32-C3用のプログラムをRustで開発する環境を準備します.

Rustのクロス・ビルド用ツールチェーンと,ESP32シリーズ固有のツールをインストールします.ESP32-C3の命令セットがRustでサポートされているRISC-V命令セットであるおかげで,Rustツールチェーンを Rustの標準的な作法で構築できます.

本章で出てくるURL,コマンドは次のサポート・ページにも掲載しています.

```
https://interface.cqpub.co.jp/2305rust2/
```

ツール①:Rustのクロス・ビルド用ツールチェーン

LinuxとmacOSでは次のコマンドでRustのクロス・ビルド用ツールチェーンがインストールできます.

```
$ curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Windowsの場合はrustupのウェブ・ページの手順に従ってインストールしてください.

```
https://rustup.rs/
```

2023年2月現在,ESP32シリーズのstd環境をビルドするためには追加でnightlyツールチェーンが必須となっています.nightlyツールチェーンとは,Rustの実験的な機能も使うことができるツールチェーンです.通常は,仕様が確定したstableと呼ばれる安定版のツールチェーンを使いますが,今回のようにRustの仕様として安定していない機能が必要な場合にはnightlyツールチェーンを使います.

今回,nightlyツールチェーンが必要な理由を解説しておきます.この後追加するコンパイル・ターゲットのriscv32imc-esp-espidfは,Rustツールチェーン上でTier3という扱いです[注1].Tier3のターゲットは,rustupでビルド済みのstdが配布されていないため,ローカルでビルドする必要があります.このためにはunstableな(安定化されていない)cargo featureを使います.unstableなcargo featureはnightlyツールチェーンでしか使えないため,nightlyツールチェーンのインストールが必須となっています.

逆に言うと,stdをソースコードからビルドする以外ではnightlyの機能は必要ありません.事実上,今回のコードは全てstableなRustコンパイラでビルド可能なものとなっています.

次のコマンドで,nightlyツールチェーンとstdのソースコードをインストールします.

```
$ rustup toolchain install nightly-2022-10-01
$ rustup component add rust-src --toolchain nightly-2022-10-01-x86_64-unknown-linux-gnu
```

● RISC-Vターゲット

RISC-VはRustがデフォルトでサポートする命令セットであるため,セットアップはコンパイル・ターゲットを追加するだけです[注2].

```
$ rustup target add riscv32imc-esp-espidf
```

● ビルド・ツール

stdを使ってビルドするときに,リンク・オプションのカスタマイズが必要です.そのためにldproxyをインストールします.

```
$ cargo install ldproxy
```

ツール②:Rustバインディング生成用ツールチェーン

ESP-IDFはC言語で書かれているため,C言語とRustとを相互にやりとりできるようにする工程がビ

注1:Rustでのコンパイル・ターゲットは,どの程度自動テストされているかによってTier1,Tier2,Tier3のいずれかに分類されます.Tier1は自動ビルドおよび自動テストが実施されており動作が保証されています.Tier2は自動ビルドは実施されています.Tier3では自動ビルドも自動テストも実施されておらず,公式なバイナリ配布はありません.

注2:今回は使用しませんが,no_stdターゲットの場合はriscv32imc-unknown-none-elfターゲットを追加します.