ESP32+FluidNC+Raspberry Pi 4B+CNCjs セットアップ草案

草案者: Wood Burned

2025 年 8 月 8 日

1　ESP32 ボードの準備とファーム書き込み

CNC 用モーションコントローラとして ESP32 を用いる。ファームウェアは FluidNC を採用する。初期導入は PC 上の Google Chrome から FluidNC 公式のオンラインインストーラを用いる。Web Serial API を使うため Chrome 前提とする。

手順の概観は次の通り。

1. ESP32 を USB で PC に接続する（給電は PC 側から行う）。
2. Google Chrome で FluidNC Web Installer を開き、Connect で該当シリアルポートを選択する。
3. Install から ESP32 向けビルド（Wi-Fi 有効版）を選択して書き込む。[1]
4. 書き込み完了後、**インストーラの「Config」（または「Files」）欄**から `config.yaml` を新規作成する。基本パラメータはここで対話的に調整する（手入力で YAML を配布せず、インストーラ側で編集・保存する）。

注意点として、各軸の STEP/DIR/EN のピン割付は、ESP32 で**出力に使ってよい GPIO** を選ぶ。ブートストラップ用のストラッピングピン（GPIO0, 2, 5, 12, 15）やフラッシュ接続ピン（GPIO6〜11）、入力専用ピン（GPIO34〜39）は避ける。UART の TX/RX（GPIO1/3）も原則避ける。割付は後述の設定方針と合わせて決定する。

Listing 1: Chrome での書き込みと config 作成の流れ（要点）

```
1) Chromeでインストーラを開く -> Connect -> ポート選択
2) Install -> "ESP32 / Wi-Fi" を選んで書き込み
3) 再起動後、インストーラの "Config" 欄で config.yaml を新規作成・保存
- 軸やピン、移動量・速度・加速度などはこの画面から編集
4) 以降の編集・保存も同画面から行う（実機確認->再調整）
```

[1] 運用では Wi-Fi は使わない。導入や更新時のみ使う方針とする。

2　FluidNC の基本設定（Config 欄で行う方針）

本書では、**基本パラメータを配布せず**、FluidNC Web Installer の **Config 欄から対話的に調整**する。理由は、機械固有の要素（スクリューピッチ、マイクロステップ、剛性、負荷、ドライバ仕様）が初期値を大きく左右するためである。次の考え方で進める。

設定の流れ（対話的チューニング）

1. 軸とモータの論理構成を定義する（X/Y/Z、スレーブ軸の有無など）。
2. 各軸の**ピン割付**を決める。前節の**使用可能 GPIO の制約**を満たす組合せを選ぶ。
3. **steps_per_mm** は実測で校正する。10,mm, 50,mm の移動指令とダイヤルゲージ等での測長を繰り返し、誤差を詰める。
4. **max_rate** と **acceleration** は安全側から立ち上げ、失歩・共振・発熱を観察しつつ段階的に上げる。
5. 方向反転やリミット/プローブ極性は、実機の挙動に合わせて修正する。

補足: Config 欄はプレーンテキストでの保存を前提とする。エディタでのリッチテキスト貼付は避ける。

3　Raspberry Pi 4B への CNCjs 導入（デスクトップ版 OS ＋ 公式 Dockerfile）

CNCjs は Web ベースの G-code 送受信/UI であり、Raspberry Pi 4B で常用する。OS は **Raspberry Pi OS（64-bit, Desktop 版）**を用いる。導入時のみ Wi-Fi を有効化し、**運用は有線（USB/有線 LAN）**とする。「電プチ」（突然の電源断）対策として Overlay File System を有効化する運用を推奨する。

OS インストール（Raspberry Pi Imager）

1. PC で Raspberry Pi Imager を起動し、OS は **Raspberry Pi OS (64-bit) with desktop** を選ぶ。
2. Advanced Settings（歯車）で、ホスト名／ロケール、**SSH 有効化**、**Wi-Fi 設定（導入時のみ使用）**を事前指定して書き込む。

3. 初回起動後、必要なパッケージを更新する。

Listing 2: 初期パッケージ導入（APT）

```
sudo apt update
sudo apt -y full-upgrade
導入に使うツール群

sudo apt -y install git curl build-essential docker.io
導入後の運用は有線、Wi-Fiは無効化（必要に応じて）

sudo rfkill block wifi
```

電プチ対策（Overlay File System）

Raspberry Pi OS は `raspi-config` から **Overlay File System** を有効化できる。運用中は rootfs を実質 Read-Only 化し、変更は RAM 上に乗る。設定変更や更新時は一時的に無効化して再起動する運用にする。

Listing 3: OverlayFS の有効化（対話/非対話の例）

```
対話（raspi-config）

sudo raspi-config # Performance Options -> Overlay File System -> Enable
非対話（新しめのraspi-configで有効）

sudo raspi-config nonint enable_overlayfs
sudo reboot
```

CNCjs の導入（公式 Dockerfile を用いる）

公式リポジトリの Dockerfile を用いて **Docker イメージをローカルビルド**し、コンテナとして常駐運用する。公式 Dockerfile は `dist/cncjs` の成果物を前提とするため、**先に Node/Yarn でビルドする**。

Listing 4: Node/Yarn でフロントをビルド（Pi 上で可）

1) CNCjs のビルド（dist 生成）

```
Node 18系を前提（bookworm既定のnodejsで可。必要ならNodeSource等を使用）

sudo apt -y install nodejs npm
sudo npm -g install yarn

git clone https://github.com/cncjs/cncjs.git
cd cncjs
yarn install
yarn run build # dist/cncjs が生成される
```

Listing 5: Docker ビルドと起動

2) 公式 Dockerfile でイメージ化

```
リポジトリ直下にある公式Dockerfileを使用

docker build -t cncjs:local -f Dockerfile .
デフォルトExposeは 8000。ホスト8080に割り当てる例。
/dev/ttyUSB* (or /dev/ttyACM*) をコンテナへ渡す

docker run -d --name cncjs
--restart=unless-stopped
--device /dev/ttyUSB0
-p 8080:8000 cncjs:local
```

3) 起動・アクセス・運用方針

- ブラウザから `http://<raspi-ip>:8080/` にアクセスし、CNCjs UI を操作する。ブラウザは Google Chrome を推奨する。
- ESP32 は USB 直結し、CNCjs のポート選択で `/dev/ttyUSB*` または `/dev/ttyACM*` を選ぶ。
- 運用は有線前提、Wi-Fi は基本無効化（導入・更新時のみ有効）。

Listing 6: systemd で自動起動（コンテナを常駐させる例）

```
/etc/systemd/system/cncjs.service

[Unit]
Description=CNCjs container
After=network-online.target docker.service
Wants=docker.service

[Service]
Restart=always
ExecStart=/usr/bin/docker start -a cncjs
ExecStop=/usr/bin/docker stop cncjs

[Install]
WantedBy=multi-user.target
有効化

sudo systemctl daemon-reload
sudo systemctl enable --now cncjs
```

4 この草案の要点（実施順）

1. Google Chrome で FluidNC Web Installer を使い、ESP32 へ書き込み後、**インストーラの Config 欄**から `config.yaml` を作って編集する（配布 YAML は用いない）。
2. 各軸のピン割付は ESP32 の GPIO 制約を満たす組合せにする（ストラッピング・フラッシュ・入力専用ピンは避ける）。
3. Raspberry Pi OS は**デスクトップ版**を Raspberry Pi Imager で導入。導入時のみ Wi-Fi 有効、以後は無効化。電プチ対策で OverlayFS を有効化。

4. CNCjs は**公式 Dockerfile** でローカルビルドし、コンテナ運用する（デフォルト Expose は 8000、例では 8080 へ割当）。

参考文献

[1] FluidNC Web Installer（公式）.

[2] FluidNC Config file Overview（公式 Wiki）.

[3] FluidNC Motion Setup（公式 Wiki）.

[4] Espressif ESP-IDF: GPIO & RTC GPIO（公式）.

[5] Espressif ESP32 Series Datasheet（公式）.

[6] Raspberry Pi Documentation: Wi-Fi/Imager Advanced Settings（公式）.

[7] Raspberry Pi Whitepaper: Making a more resilient file system（公式）.

[8] CNCjs Installation / Raspberry Pi Setup Guide（公式）.

[9] CNCjs 公式 Dockerfile（master）.

[10] Docker Hub: cncjs/cncjs（公式イメージ）.