ESP32 + mruby/c開発のための環境構築 - WSL

Windowsの「Subsystem for Linux (WSL) 」にESP開発環境を構築

WSLは、64bit版のWindows10（またはWindows Server）上でLinuxの実行ファイルをネイティブ実行できる環境です。Windows8以前または32bit版Windows10ではWSLを利用できないため、MSYS2をお使いください。

WSLにESP-IDFをインストールすることでLinux上のESP開発環境と同等の環境を構築できます。ファームウェアのビルド速度がMSYS2より大幅に速いため（というよりもMSYS2がかなり遅いのです）、これから新たに環境構築する方はWSL環境の構築を先に試してみてください。

mrubyソースコード（拡張子「.rb」）をmrbc（mrubyコンパイラ）によって拡張子「.c」の中間バイトコードに変換し、それ（とmruby/cのランタイムプログラム）を「main.c」から参照することで動作させるのが、mruby/cアプリ開発の基本的なフローです。

ESP32のファームウェアをmruby/cで開発するためには、Espressif社が提供しているESP-IDFおよび関連ツール群をセットアップする必要があります。ESP-IDFにはESPファームウェア開発に使用可能なライブラリが含まれ、実行ファイルの作成をサポートします。

以下では、ESP-IDFや関連ツール群をセットアップした開発環境のことを「ESP開発環境」と呼び、その構築について説明します。

仮想環境、Dockerについて

例えば、Windows10 Professional上のVirtualBoxにインストールしたLinuxにESP開発環境を構築することは可能です。しかし、ホストOSにUSB接続されたESP32開発ボードをゲストOSから適切に参照できるかどうかを左右する要因のすべてが明らかではなく、期待どおりに動作しないとの報告があるようです。

したがってこのマニュアルでは、ホストOS上にESP開発環境を構築することを前提としています。

仮想環境を使用したい方も、ワークショップをスムーズに進めるためにホストOSの上の環境を先につくったうえで、ゲストOS上のESP開発環境をつくってみてください。そして、うまくできる方法やできない方法についての情報をぜひ共有してください。

また、Dockerは一般にUSBドライバに問題があるようなので、使用できないとお考えください（もし挑戦してみて使用できたら教えてください）。

USBドライバについて

今回のワークショップで使用するESP32開発キットには、「CP2102N」というシリアル-USB変換チップが使用されており、このドライバをWindowsにインストールする必要があります。もしも、以下で説明するPololu社のドライバを有効にできない場合は、WSLを諦めてMSYS2環境を選択してください。

また、お使いのOSが64bit版のWindows10以外でしたらいずれにせよMSYS2の環境を構築することになるので、USBドライバはどちらでも構いません。

2019年3月時点で、Silicon Labs社が提供するドライバではWSLとESP32が期待するように通信できません。少なくとも筆者の環境では動きませんでした。MSYS2ならば問題ありません。

【ご注意ください】下記ページからダウンロードしたドライバとWSLの組み合わせでは期待どおりには動かないと思われます。

（これはNGです）https://jp.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

代わりに、Pololu社のページからダウンロードできるドライバをインストールしてください。

https://www.pololu.com/docs/0J7/all#2

（ドライバファイルへの直リンク：https://www.pololu.com/file/0J14/pololu-cp2102-windows-121204.zip ）

ZIPファイルを解凍し、「pololu-cp2102-setup-x64.exe」をダブルクリックしてインストールしてください。

既にSilicon Labs社のドライバがインストール済みだった場合

インストール済みのドライバを完全に削除してから、Pololu社のドライバをあらためてインストールする必要があると思われます。

「デバイスのアンインストール」の操作だけではドライバファイルが削除されず、Windowsが（プラグ・アンド・プレイ機能によって）自動的に日付の新しいものを割り当ててしまいます。Silicon Labs社のドライバの日付のほうがPolulu社より新しいため、Windowsが自動的にSilicon Labs社のものを選択してしまい、ユーザの任意によってPololu社のドライバを有効にはできないのです。

一般的には、下記ページのような手順でドライバを完全に削除できますが、自己責任で実施してください。

https://www.ipentec.com/document/windows-delete-device-driver-in-driver-store

この問題を解決できない場合や、すでにインストールされているドライバがなんであるかよくわからない場合などは、WSLではなくMSYS2の環境構築をお選びください。あるいは両方を構築しておいて、ワークショップ当日に使える方を使う、というのもよいと思います。

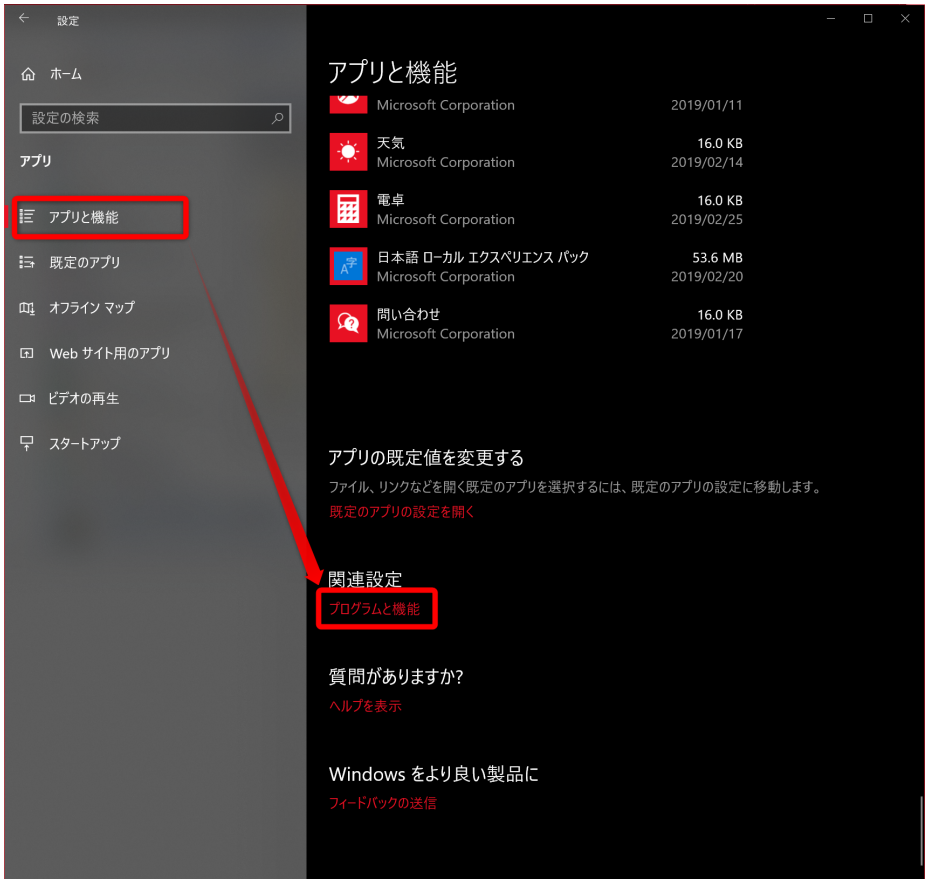
環境構築

Windows Update

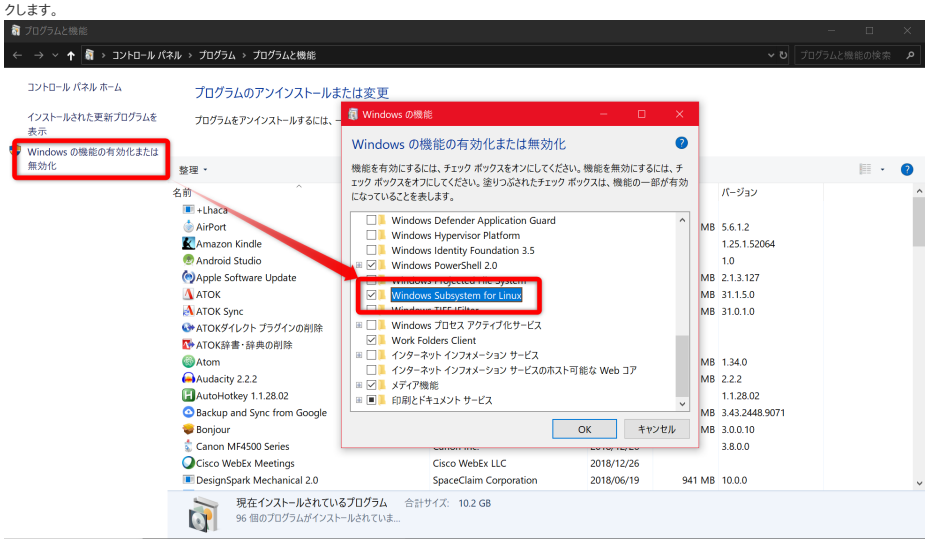
まず、Windows10をMicrosoftが提供している最新の状態に更新してください。更新されていないWindowsでは、WSLの機能のうちわれわれが必要とするものを利用できない可能性があります。

WSL (Ubuntu) のインストール

「設定」→「アプリと機能」→「プログラムと機能」をクリックします。

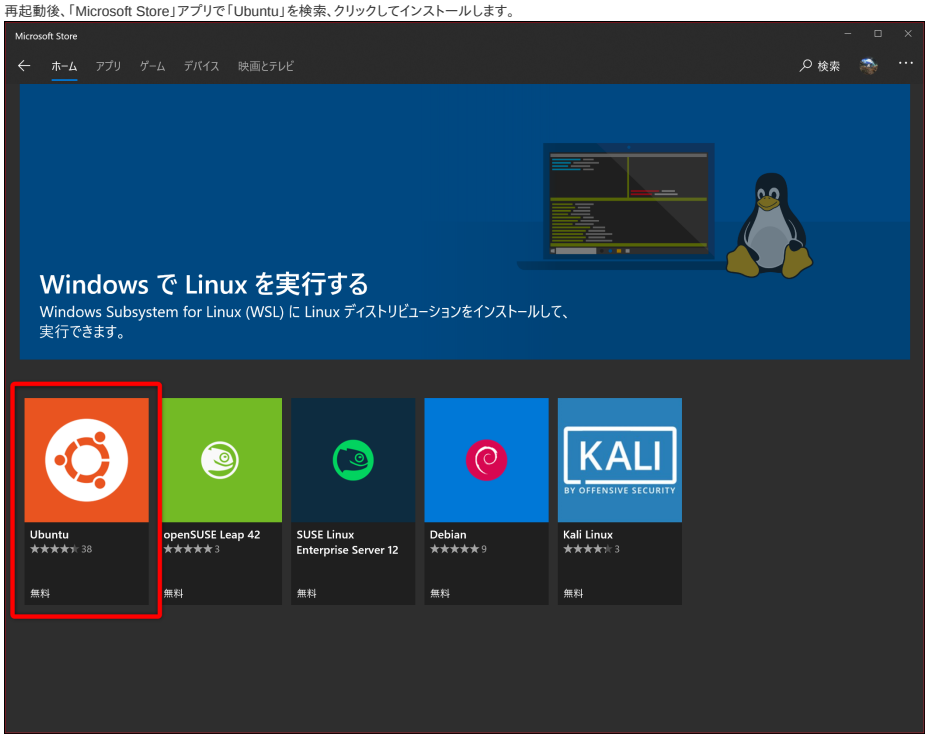


「Windowsの機能の有効かまたは無効可」をクリックし、「Windowsの機能」ダイアログ内の「Windows Subsystem for Linux」にチェックを入れ、「OK」をクリックします。

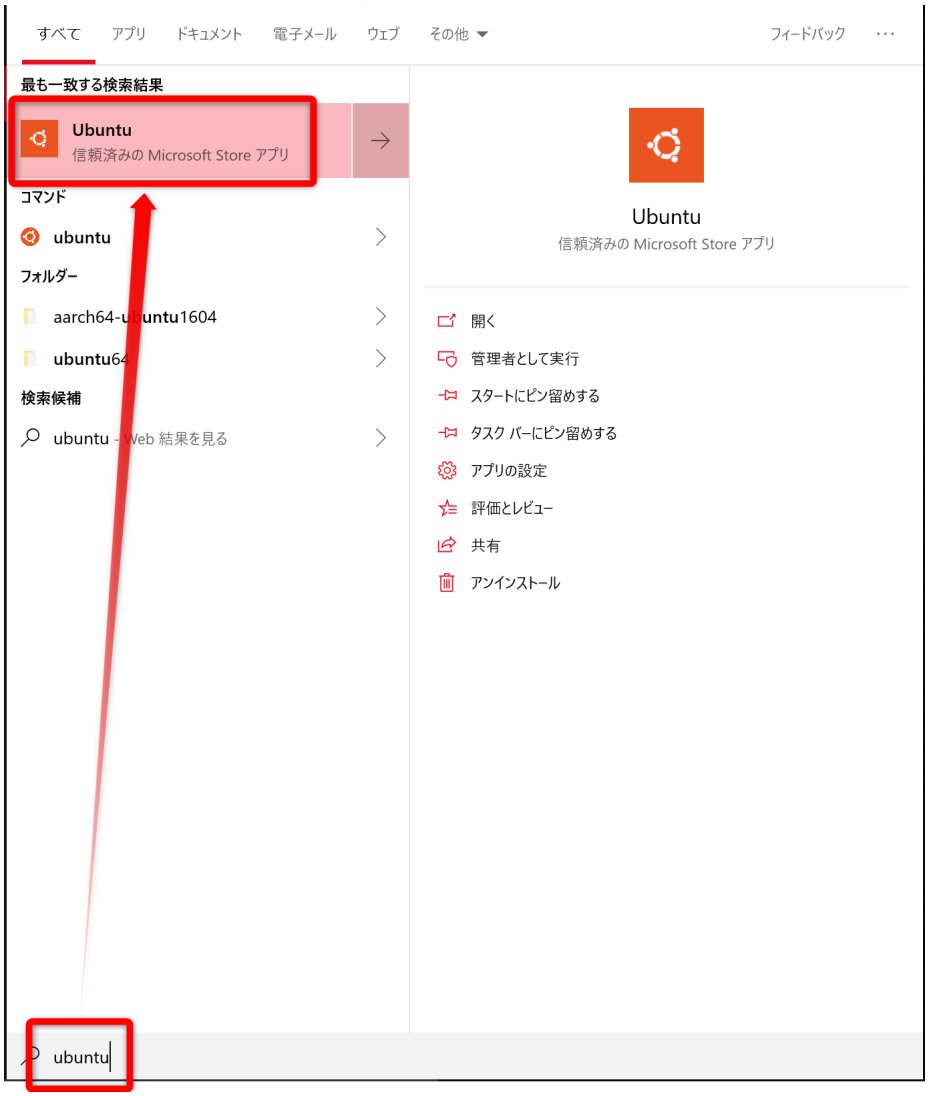


再起動を促されるので「今すぐ再起動」をクリックします。

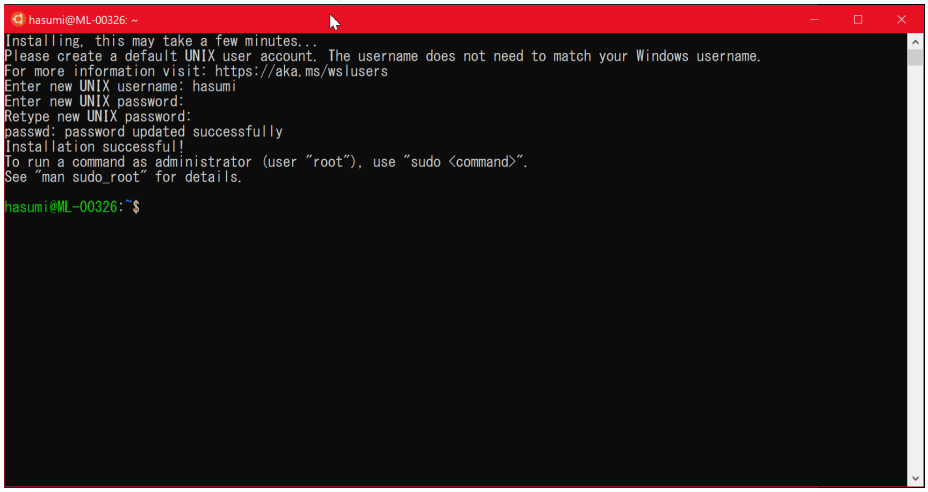
再起動後、「Microsoft Store」アプリで「Ubuntu」を検索、クリックしてインストールします。



「Ubuntu」アプリ（これがWSLのUbuntu版です）を起動します。



初回の起動時にはUnixユーザ名とパスワードの設定が必要です。



Ubuntu上での環境構築

関連パッケージをインストールします。

```
sudo apt update
sudo apt install gcc git wget make libncurses-dev flex bison \
  gperf python python-pip python-setuptools python-serial \
  python-cryptography python-future python-pyparsing
```

Espressif社が提供しているツール群をダウンロードし、解凍、配置します。

```
mkdir $HOME/esp
cd $HOME/esp
wget https://dl.espressif.com/dl/xtensa-esp32-elf-linux64-1.22.0-80-g6c4433a-5.2.0.tar.gz
tar -xzf xtensa-esp32-elf-linux64-1.22.0-80-g6c4433a-5.2.0.tar.gz
rm xtensa-esp32-elf-linux64-1.22.0-80-g6c4433a-5.2.0.tar.gz
```

.profileファイルに環境変数を設定し、有効化します。

```
echo 'export PATH="$HOME/esp/xtensa-esp32-elf/bin:$PATH"' >> $HOME/.profile
echo 'export IDF_PATH="$HOME/esp/esp-idf"' >> $HOME/.profile
source $HOME/.profile
```

ESP-IDFを配置します。

```
cd $HOME/esp
git clone --recursive https://github.com/espressif/esp-idf.git
```

python製の関連ツールをインストールします。

```
python -m pip install --user -r $IDF_PATH/requirements.txt
```

シリアルポートの権限が必要なので、自ユーザをdialoutグループに追加します。

```
sudo usermod -a -G dialout $USER
```

プログラム作成ディレクトリについて

Ubuntu上のVimなどではなく、Windows上に別途起動するエディタでプログラムを書くつもりでしたら、WindowsとWSLのディレクトリ共有について確認しておくとよいでしょう。

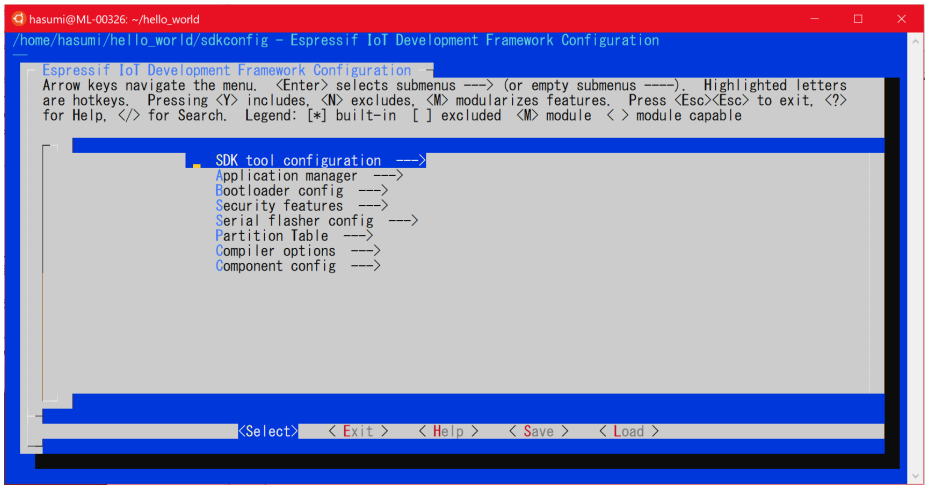
WSL上の `/mnt/c/Users/[ユーザ名]/esp` がWindows上の `c:¥Users¥[ユーザ名]/esp` に一致します。

サンプルプロジェクトをビルド

ESP-IDFに含まれているサンプルプロジェクト hello_world をコピーしてビルドしてみましょう。

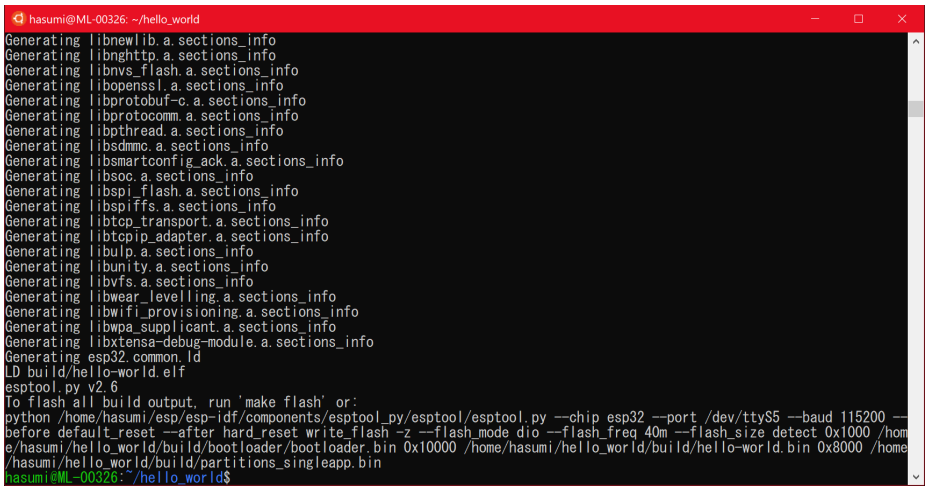
```
cp -r $IDF_PATH/examples/get-started/hello_world $HOME/esp
cd $HOME/esp/hello_world
make
```

初回の make 時には下の画像のような `make menuconfig` 相当の画面になります。この時点では設定を変更する必要がないので、エスケープキーを2回押してmenuconfigを終了してください。



また、ターミナル（ウインドウ）のサイズが小さすぎると「menuconfig画面をつくれない」という意味のエラーがでます。サイズを大きくして再度 `make` してください。

設定ファイルが自動で生成され（これによって次回の `make` コマンドでは設定画面が表示されなくなります。明示的に表示するためのコマンドが `make menuconfig` です）、プロジェクトのビルドが始まるはずです。下の画像のような出力で終了すれば正常です。



正常終了しなかった場合は、これまでの手順のどこかを抜かしたか、入力ミスなどで正しく手順を踏めていなくてエラーメッセージに気づかず進んでしまったことが考えられます。

Rubyについて

mrubyのビルドにはCRuby（最も一般的なRuby実装）が必要です。

Rubyのインストールには複数の方法がありますが、複数のRubyをシステム内に共存させるためのツール「rbenv」をインストールすることを推奨します。

ワークショップの後半に時間があれば筆者作のmruby/c用便利ツール `mrubyc-utils` を使う予定があり、rbenvの環境のほうがスムーズに使用できます。

Rubyをインストール

rbenvをインストールします。

```
cd $HOME
git clone https://github.com/rbenv/rbenv.git $HOME/.rbenv
```

パスを通すなどします。

```
echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> $HOME/.profile
echo 'eval "$(rbenv init -)"' >> $HOME/.profile
source .profile
```

ruby-buildをインストールします。

```
mkdir -p "$(rbenv root)"/plugins
git clone https://github.com/rbenv/ruby-build.git "$(rbenv root)"/plugins/ruby-build
```

WSLにはシステムデフォルトのRubyがありません。mrubyのビルドにはCRubyが必要なので、まずはCRubyをインストールします。非常に時間がかかりますので気長に実行してください。

```
sudo apt-get install -y libssl-dev libreadline-dev zlib1g-dev
rbenv install 2.6.1
```

たったいまインストールしたCRubyをグローバルデフォルトに設定します。

```
rbenv global 2.6.1
ruby --version
```

上のコマンドで、`ruby 2.6.1p33 (2019-01-30 revision 66950) [x86_64-linux]` のように出力されればOKです。

mrubyをインストールします。現状、mruby-2.xはmruby/cには使えないので、1.4.1をインストールしてください。

```
rbenv install mruby-1.4.1
```

つづきはワークショップで！

お疲れ様でした！　これにて環境構築は終了です。ワークショップ当日お目にかかれることを楽しみにしています！

プログラムを書くためのテキストエディタの準備もお忘れなく。

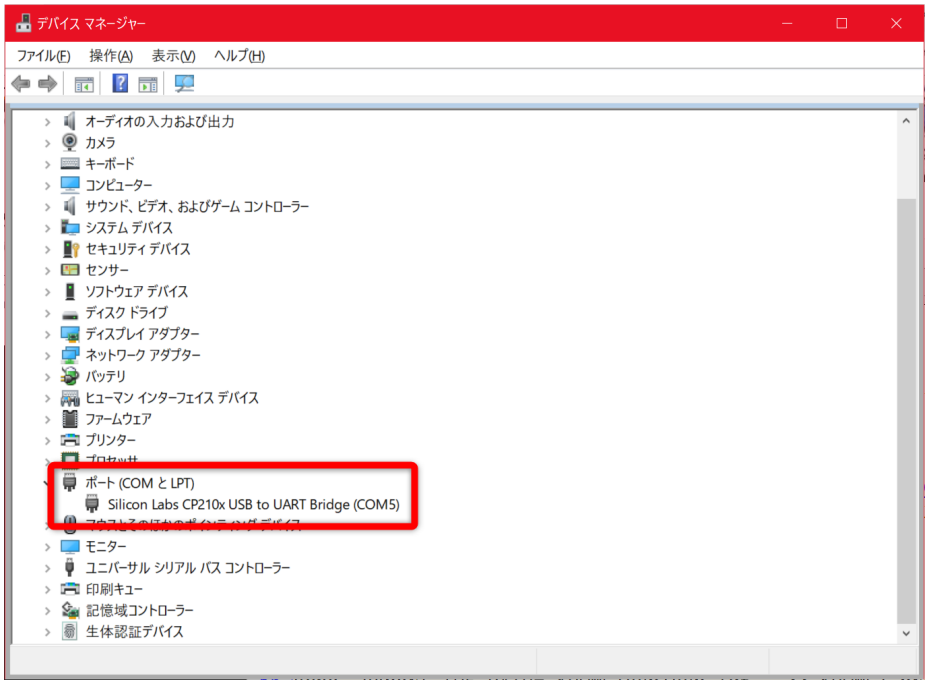
※以下の手順はワークショップ当日に行うものです。

USBポートの動作確認

COMポート番号を確認

「デバイスマネージャー」アプリを開き、その状態のままUSBケーブルのマイクロコネクタ側をESP32開発ボードに、タイプAコネクタをWindowsパソコンに接続します。

「USB to UART ブリッジドライバ」がインストール済みなので、画像のように「ポート（COMとLPT）」内に「Silicon Labs CP210x USB to UART Bridge (COM5)」のような項目が現れるはずです。名称は環境によって異なる可能性があります。



最後の「(COM5)」の「5」の数字が、みなさんの環境では異なる可能性があります。WSL上のシリアルポート `/dev/ttyS5` がWindows側のCOM5ポートにマッピングされます。終端の数字が一致します。

たとえば、あなたのWindowsのデバイスマネージャー上で「COM4」となっていた場合は、`/dev/ttyS4` があなたの使用するべきシリアルポートです。この数字を覚えておいてください。

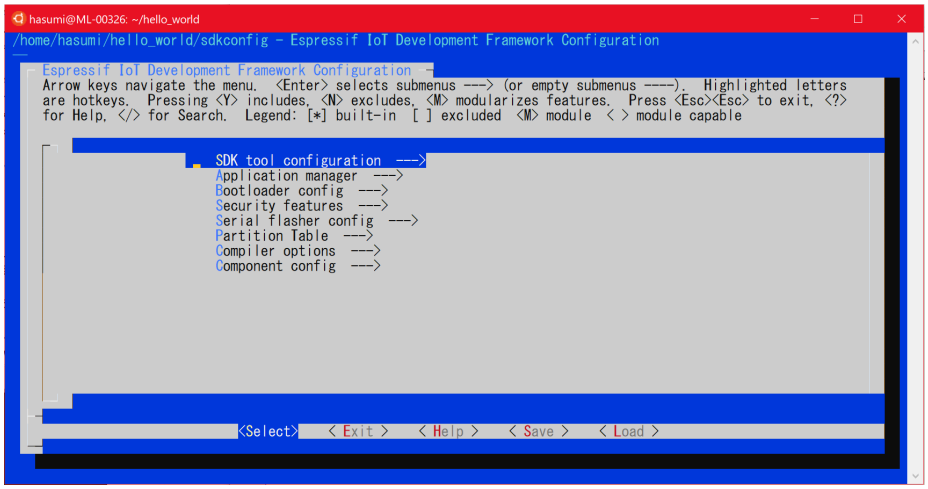
シリアルポートを設定

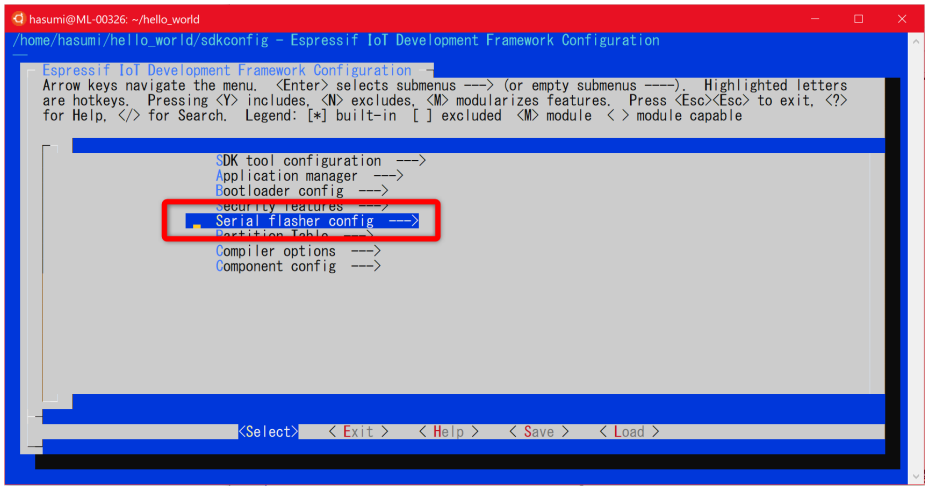
```
make menuconfig
```

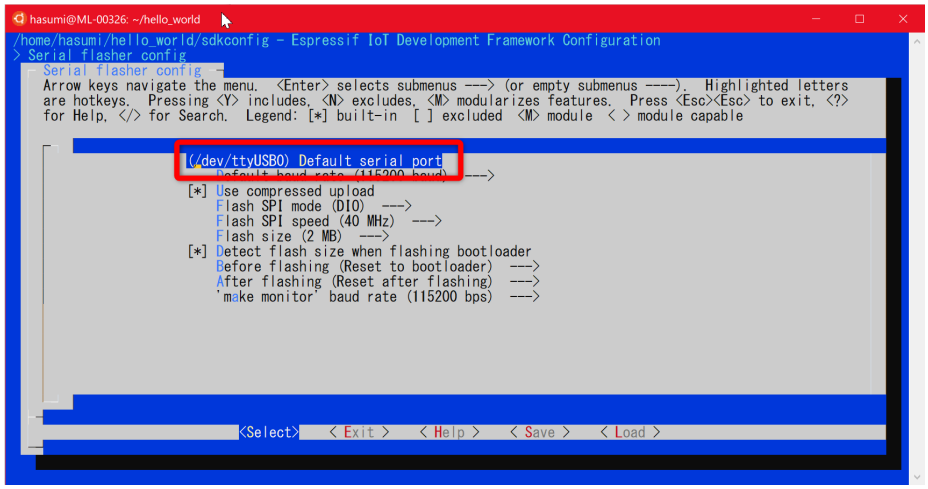
上記コマンドで設定画面を起動し、カーソルキーとエンターキーで「Serial flasher config」→「(/dev/ttyUSB0) Default serial port」と選択し、ポートを「※（下で説明します）」に変更してエンターキーで確定し、何度かエスケープキーを押すと保存するか確認されるので「」を選択してください。

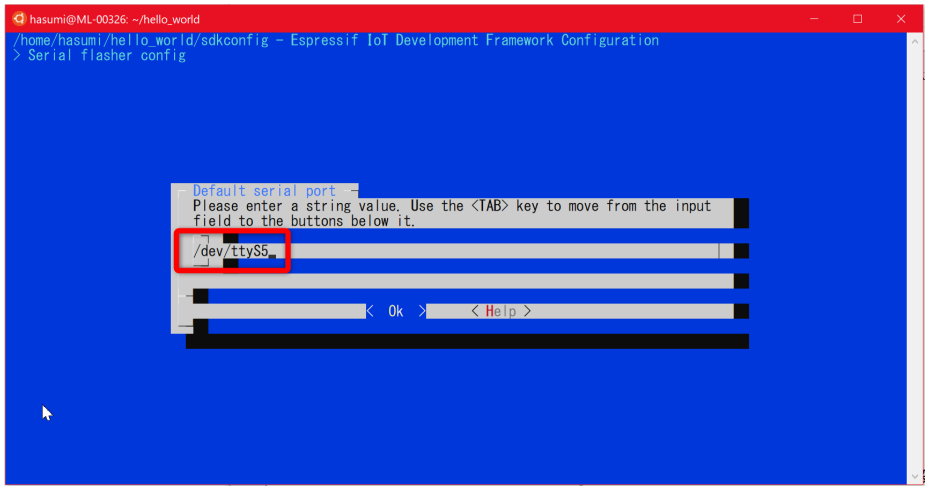
- （※）ポート名について：
    - macOSの場合：先ほどメモをとった「/dev/cu.SLAB_USBtoUART」のような文字列

- Windows（WSL）の場合：「/dev/ttyS5」（最後の数字を先ほど確認したCOM番号と同じものに変更してください）
- Windows（MSYS2）の場合：「COM5」（先ほど確認したCOM名と同じ。先頭にスラッシュ「/」は不要です）









サンプルプロジェクトを書き込み、実行

このコマンドでプロジェクトがビルドされます。

```
make
```

このコマンドでプロジェクトが書き込まれます。makeコマンドの一般的な動作と同様、プログラムファイルの更新日時から計算される依存関係上必要な場合は、ビルドが先に実行されます。

```
make flash
```

このコマンドでESP32がリブートしてファームウェアが先頭から実行され、実行中のデバッグ情報などが標準出力に書き出されます。

```
make monitor
```

上の3つのコマンドは以下のように一度に実行できます。

```
make flash monitor
```

make monitorの出力に「Hello world!」の文字が出ていれば成功です！　サンプルプログラムがESP32の上で動いています！

このコンソールモニタは、`ctrl + ]` で終了できます。