ESP32 + mruby/c開発のための環境構築 - MSYS2

Windowsの「MSYS2」にESP開発環境を構築

※あなたのOSが64bit版のWindows10で、かつPololu社のUSBドライバを有効化できているなら、WSLの環境構築をおすすめします。

MSYS2は、Windows上にUnixコマンドライン環境をエミュレートするための統合パッケージです。Espressif社はMSYS2に各種の必要ツールをセットアップしたオール・イン・ワンのZIPアーカイブを提供しているので、それをそのまま使用するのが最も簡単な開発環境構築です。

もしも「素のMSYS2」に依存ツールを1つずつインストールしてみたい場合は、下記ページにそのための情報があります。ただし筆者は未検証です。

https://docs.espressif.com/projects/esp-idf/en/latest/get-started/windows-setup-scratch.html

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ドライバをインストール

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。

以下のうちどちらか一方のページからお使いのOSに該当するドライバをダウンロードし、インストールしてください。もしもWSLを併用する可能性がある場合は、後者（Pololu社）のドライバをお使いください。理由はWSLの環境構築マニュアルを参照してください。

https://jp.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

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

環境構築

MSYS2（およびツール群）のダウンロードと配置

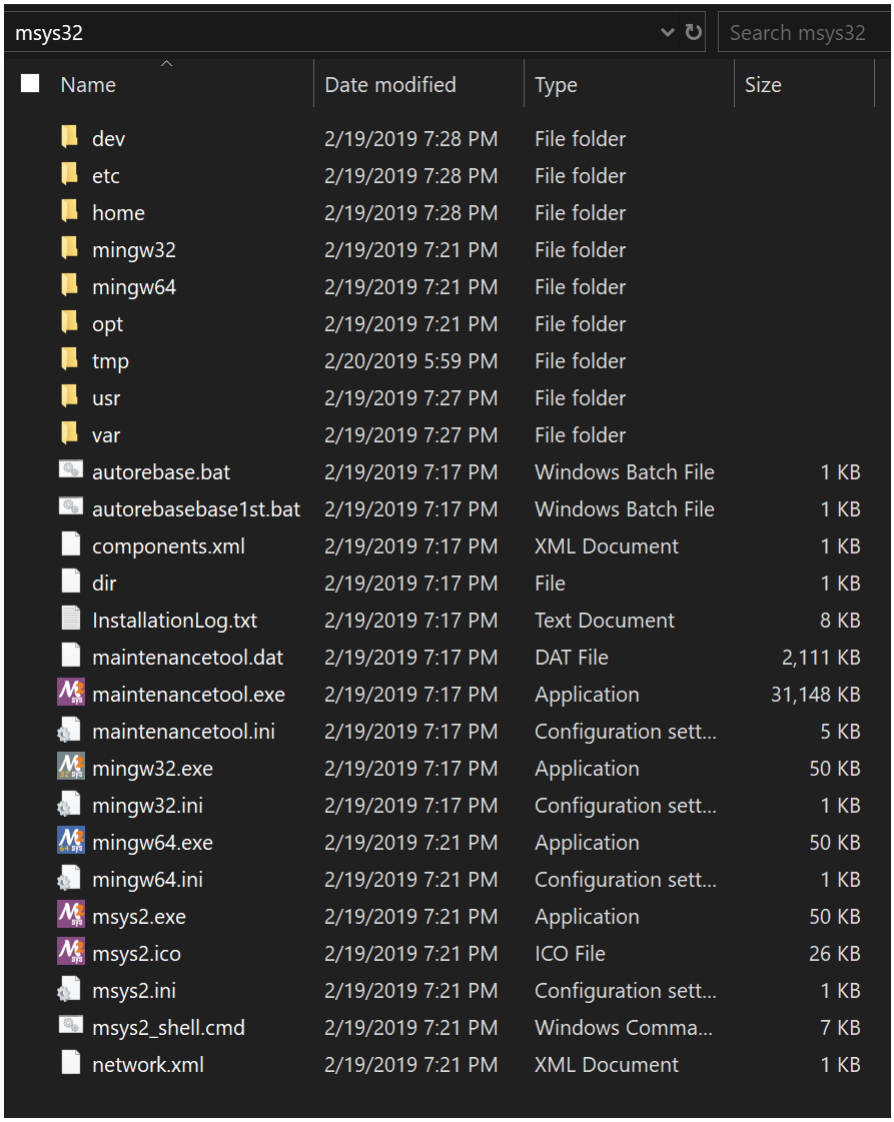
下記リンクからオール・イン・ワンのZIPアーカイブをダウンロードして、解凍してください。

https://dl.espressif.com/dl/esp32_win32_msys2_environment_and_toolchain-20181001.zip

任意の場所に解凍して構いませんが、一般的にはWindowsがインストールされているCドライブ内のどこかがよいと思われます。解凍にはかなり時間がかかります。解凍してからフォルダを移動するのにも時間がかかるので、最初から配置したい場所に解凍しましょう。

解凍・配置時に注意すべきことがあります。その過程でなんらかのアンチウィルスの仕組みが一部のファイルを自動削除してしまうかもしれません。これが発生すると、正しく使用できなくなります。

解凍すると現れる `msys32` ディレクトリ内が下の画像のようになっていれば、概ね問題ないと思われます。



筆者の環境では当初、etcディレクトリなどが削除されてしまったので、仮想のLinux上で解凍して対処しました。

しかしその後、本稿執筆のために再現させようとしても再現しなかったため、本当はなにが原因だったのかわからないし回避方法も不明です。

MSYS2の起動と設定

`msys32` ディレクトリ内の `mingw32.exe` を起動してください。`mingw64.exe` ではダメです。これはあなたのWindowsが64bit版であるか32bit版であるかにかかわりません。ESP-IDFがmingw64に対応していません。

以降、MSYS2を起動するときは毎回 `mingw32.exe` を使用してください。

.bash_profileファイルに環境変数を設定し、有効化します。

```
echo 'export IDF_PATH="$HOME/esp/esp-idf"' >> $HOME/.bash_profile
source $HOME/.bash_profile
```

ESP-IDFを配置します。macOSやWSLでインストールが必要だった関連ツール群はすでにインストールされているため、簡単です。

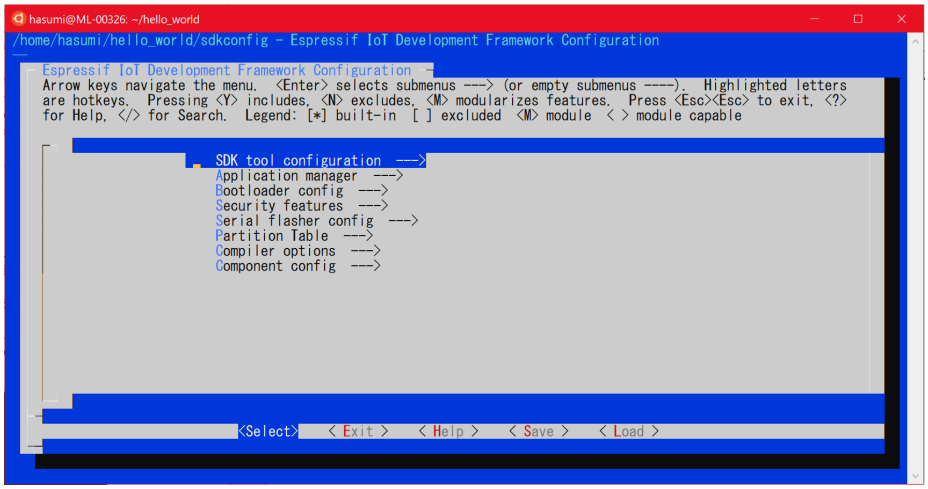
```
mkdir $HOME/esp && cd $HOME/esp
git clone --recursive https://github.com/espressif/esp-idf.git
```

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。

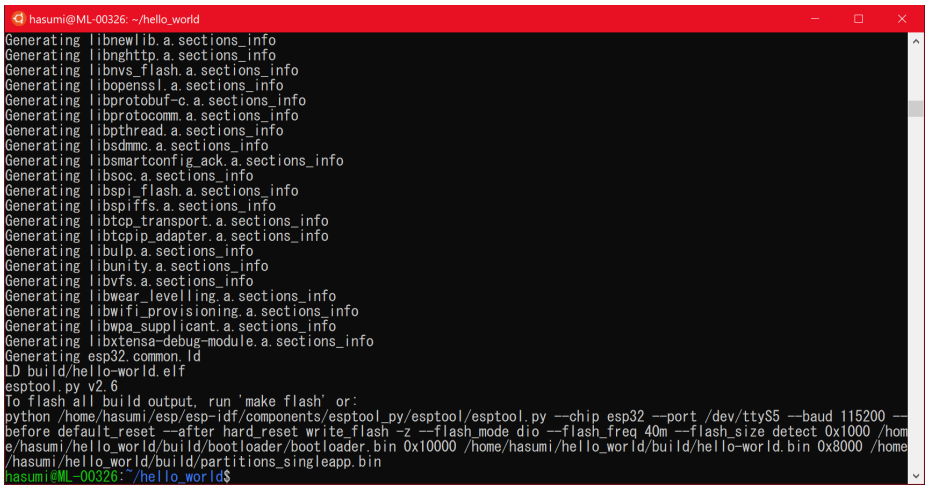
```
cp -r $IDF_PATH/examples/get-started/hello_world $HOME/esp
cd $HOME/esp/hello_world
make
```

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インストール（Rubyinstaller2を使用）

※MSYS2にrbenvをインストールするのも可能なようですが、簡単ではなさそうです。挑戦してみたい方はネットで調べてみてください。

最初にCRubyをインストールします。mrubyのビルドにはCRubyが必要なためです。

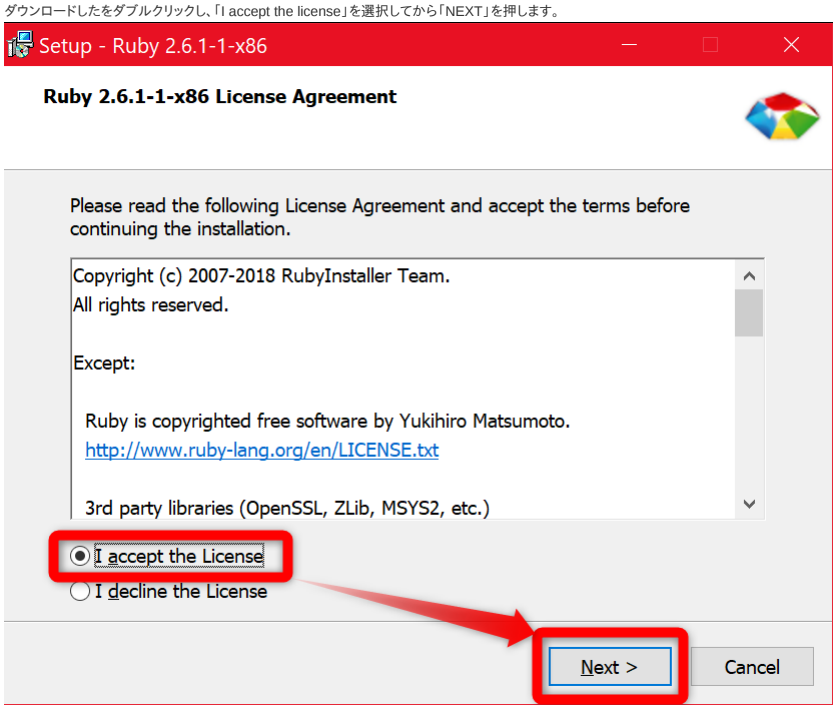
MSYS2上ではなく、WindowsのGUI上で行ってください。WindowsでRubyを使用するための専用インストーラをダウンロードします。

https://github.com/oneclick/rubyinstaller2/releases/download/RubyInstaller-2.6.1-1/rubyinstaller-2.6.1-1-x86.exe

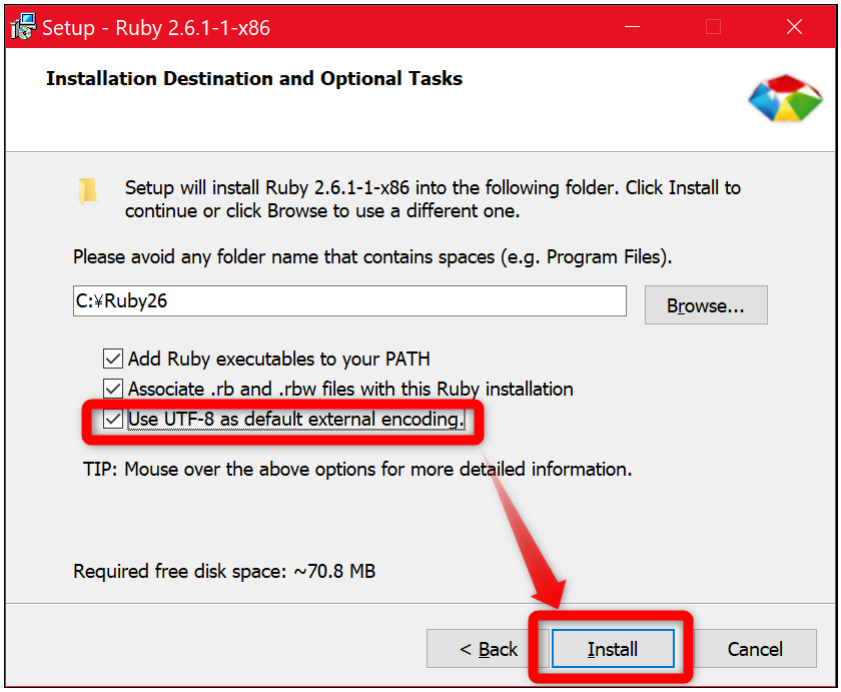
32bit版のRubyインストーラをダウンロードします。MSYS2でESP-IDFを使用するために `mingw32.exe` をPOSIXエミュレータとして使用するので、これにあわせています。

ホストOSが64bit版のWindows8などである場合は、64bit用のRubyインストーラ（rubyinstaller-2.6.1-1-x64.exe）でもよいかもしれませんが、未検証です。一般的に、64bit版のWindowsでは32bit版の実行ファイルが動作します。逆は動作しません。

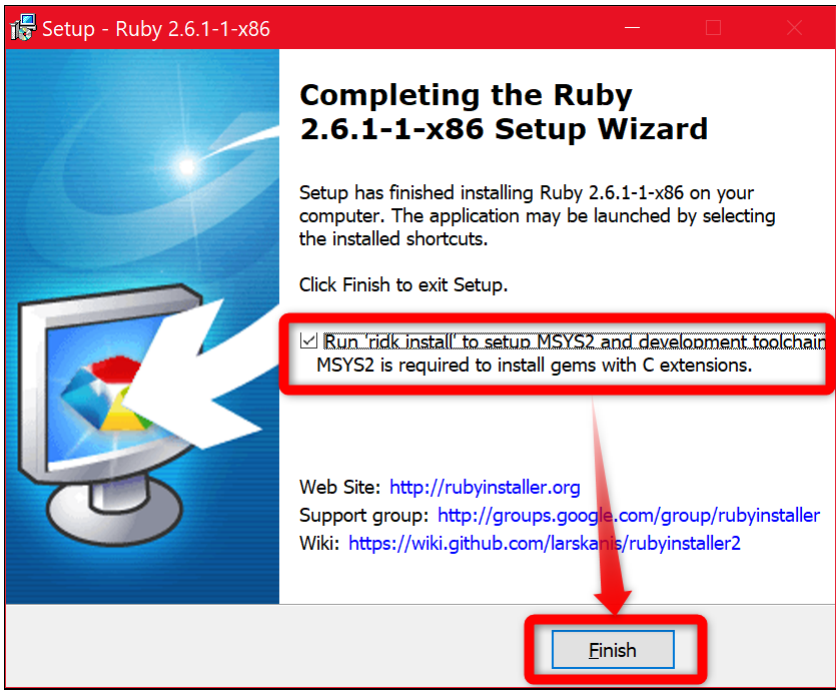
ダウンロードしたをダブルクリックし、「I accept the license」を選択してから「NEXT」を押します。



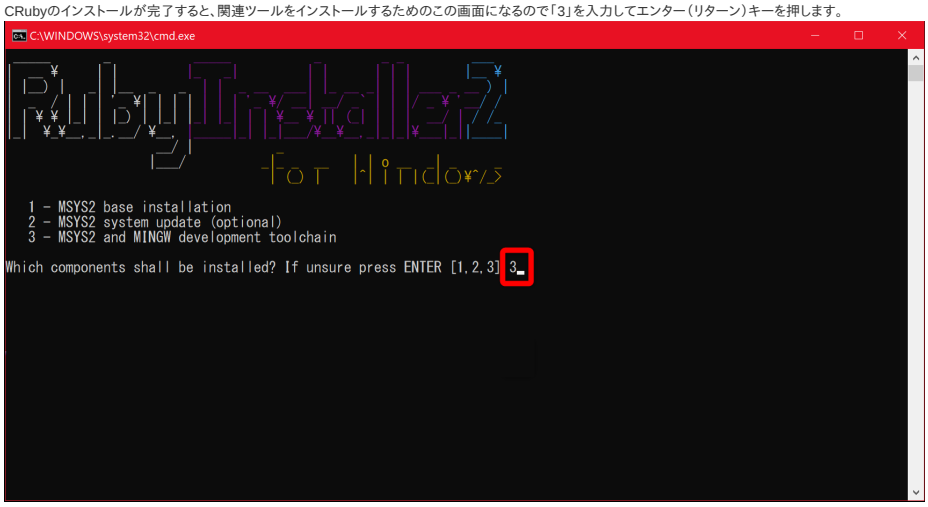
「Use UTF-8 as default external encoding.」のチェックが外れているでしょうから、チェックして、「Install」を押します（エンコーディングは今回のワークショップには関係ありませんが、他の用途に使用するときにこのほうがよさそうです）。



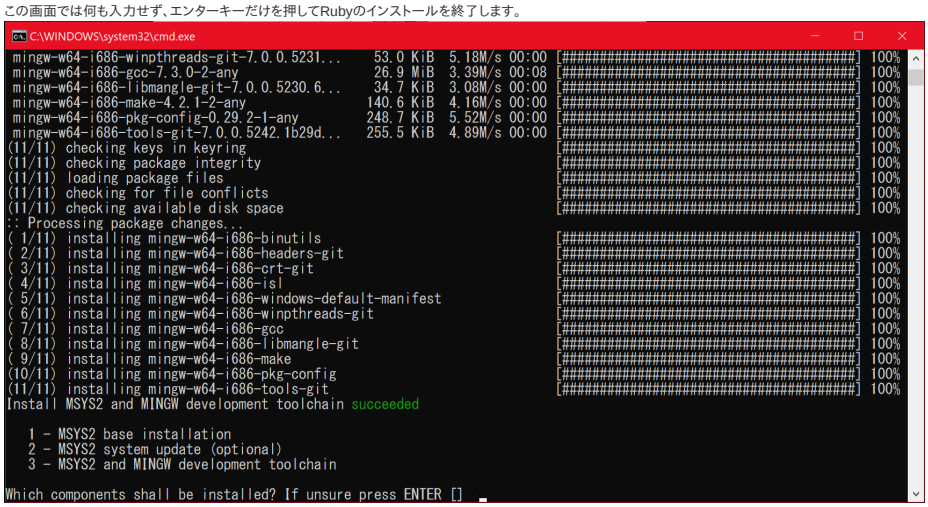
「Run 'ridk...'」にチェックが入っていることを確認して、「Finish」を押します。



CRubyのインストールが完了すると、関連ツールをインストールするためのこの画面になるので「3」を入力してエンター（リターン）キーを押します。



この画面では何も入力せず、エンターキーだけを押してRubyのインストールを終了します。



MSYS2のコマンドラインにパスを通します。

```
cd $HOME
echo 'export PATH="/c/Ruby26-x64/bin:$PATH"' >> $HOME/.bash_profile
source $HOME/.bash_profile
```

確認します。

```
ruby --version
```

上のコマンドで `ruby 2.6.1p33 (2019-01-30 revision 66950) [i386-mingw32]` が出力されればOKです。

つぎにmrubyをインストールします。mruby2.xはまだmruby/cと統合されていないので、1.4.1を使用します。

```
cd $HOME
wget https://github.com/mruby/mruby/archive/1.4.1.zip
unzip 1.4.1.zip
cd mruby-1.4.1
ruby minirake
```

パスを通します。

```
echo 'export PATH="$HOME/mruby-1.4.1/build/host/bin:$PATH"' >> $HOME/.bash_profile
source $HOME/.bash_profile
```

確認します。

```
mrbc --veriosn
```

上のコマンドで `mruby 1.4.1 (2018-4-27)` と出力されればOKです。

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。

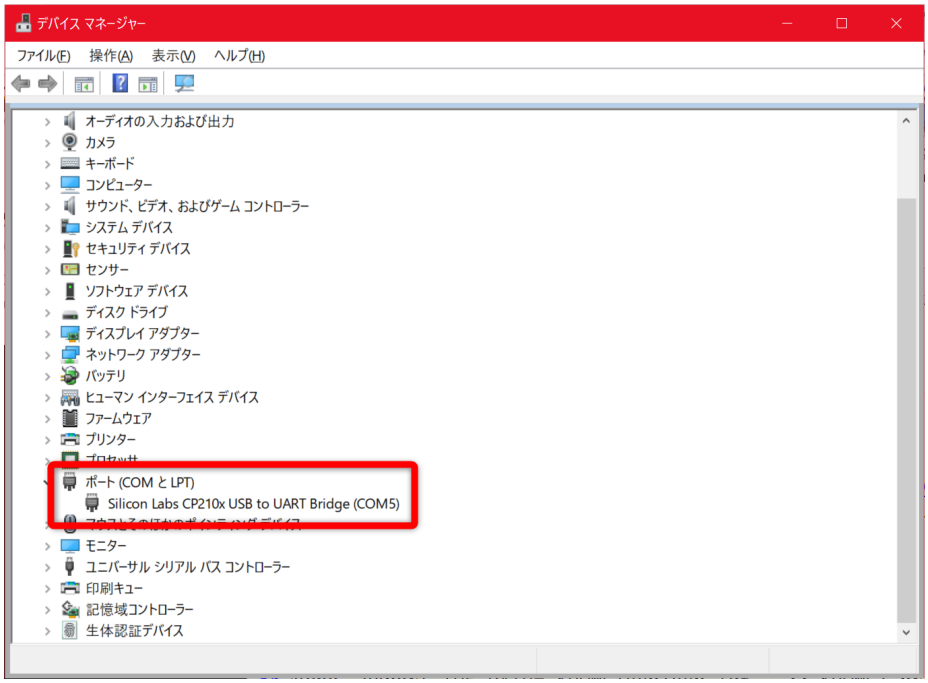
※以下の手順はワークショップ当日に行うものです。

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。

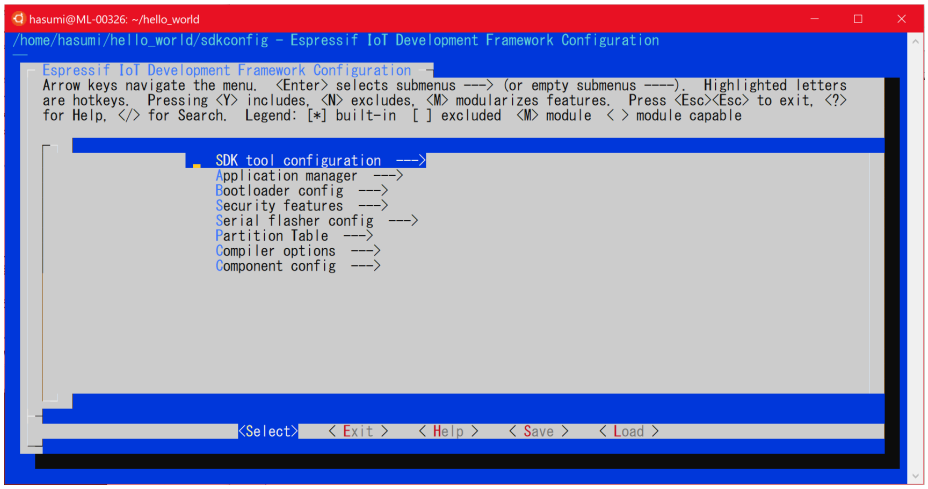
たとえば、あなたのWindowsのデバイスマネージャー上で「COM4」となっていた場合は、 `/dev/ttyS4` があなたの使用するべきシリアルポートです。この数字を覚えておいてください。

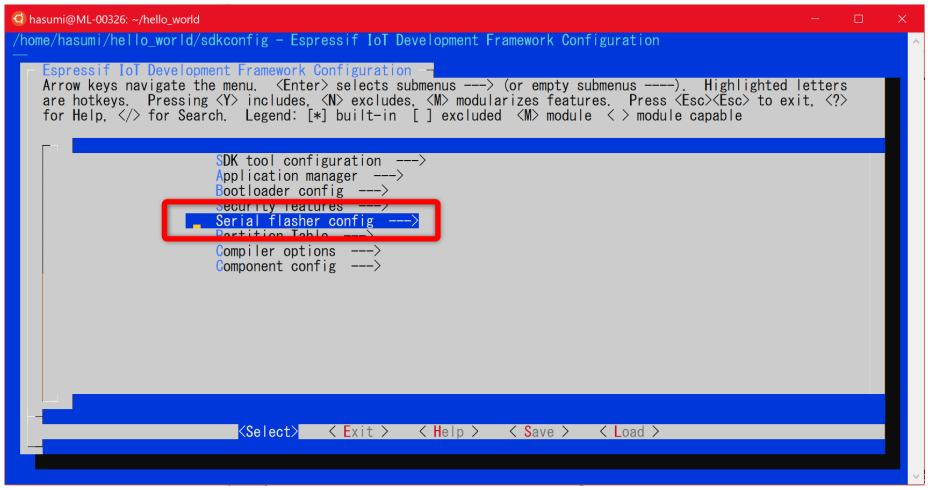
シリアルポートを設定

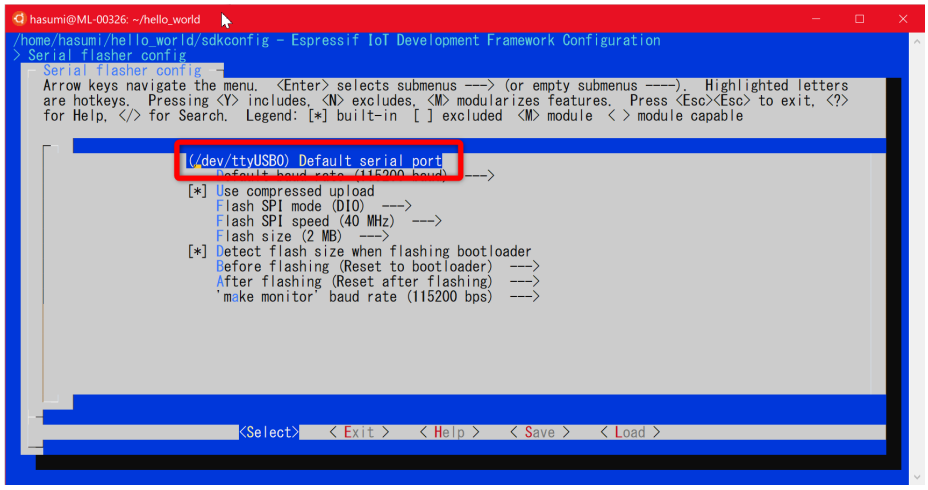
```
make menuconfig
```

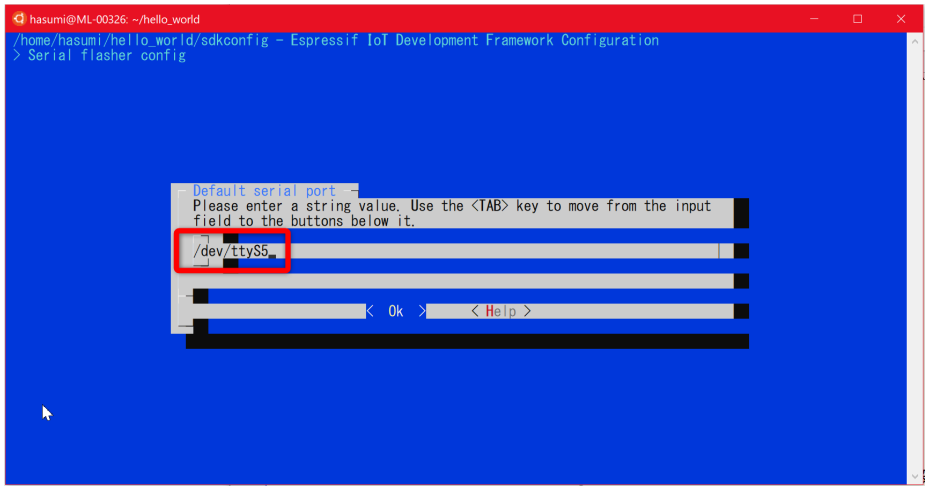
上記コマンドで設定画面を起動し、カーソルキーとエンターキーで「Serial flasher config」→「(/dev/ttyUSB0) Default serial port」と選択し、ポートを「※（下で説明します）」に変更してエンターキーで確定し、何度かエスケープキーを押すと保存するか確認されるので「」を選択してください。

- （※）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。