ESP32 + mruby/c開発のための環境構築 - macOS

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。

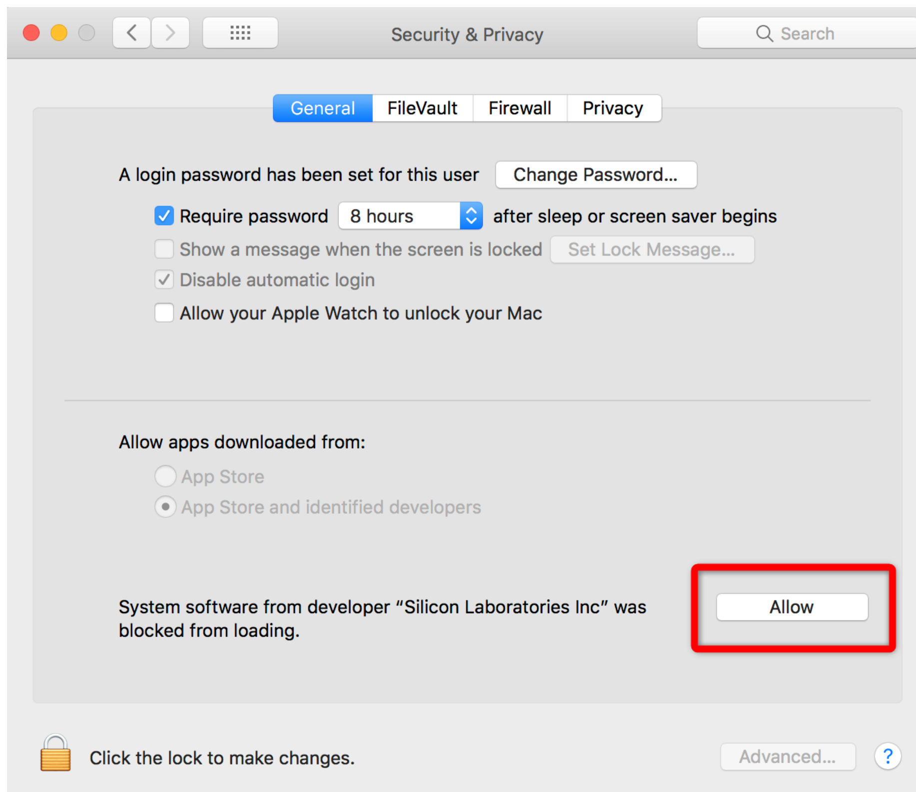
また、Dockerは一般にUSBドライバに問題があるようなので、使用できないとお考えください（もし挑戦してみて使用できたら教えてください）。

ESP開発環境

USBドライバ

「USB to UART ブリッジドライバ」をインストールします。下記ページからお使いのOSに適合するものをダウンロードして、インストールしてください。
https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

インストールの中で「機能拡張がブロックされました（System Extension Blocked）」というダイアログが表示された場合には、「システム環境設定」->「セキュリティとプライバシー」->「一般」を開き、「許可」をクリックしてください（すべての環境でブロックされるわけではないようです）。



ツールのセットアップ

ESP関連ツール群をダウンロードして解凍します。

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

pythonのバージョン管理にpyenvなどをお使いでしたら、以下のようにシステムデフォルトのpythonにバージョンを固定したほうがよいかもしれません。pyenv等を使用していない、またはpythonの利用に慣れており問題への対処が可能だと思う場合は、この設定はスキップして構いません。

```
cd $HOME/esp
echo 'system' > .python-version
```

.bash_profileファイルに環境変数を設定し、有効化します（bashではなくzshを使用しているなどの場合は適宜読み替えてください）。

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

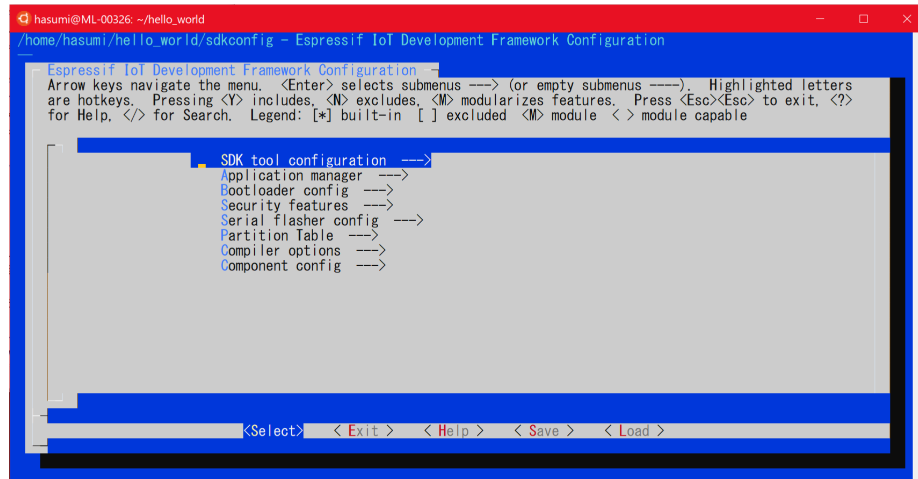
```
sudo easy_install pip
python -m pip install --user -r $IDF_PATH/requirements.txt
```

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。

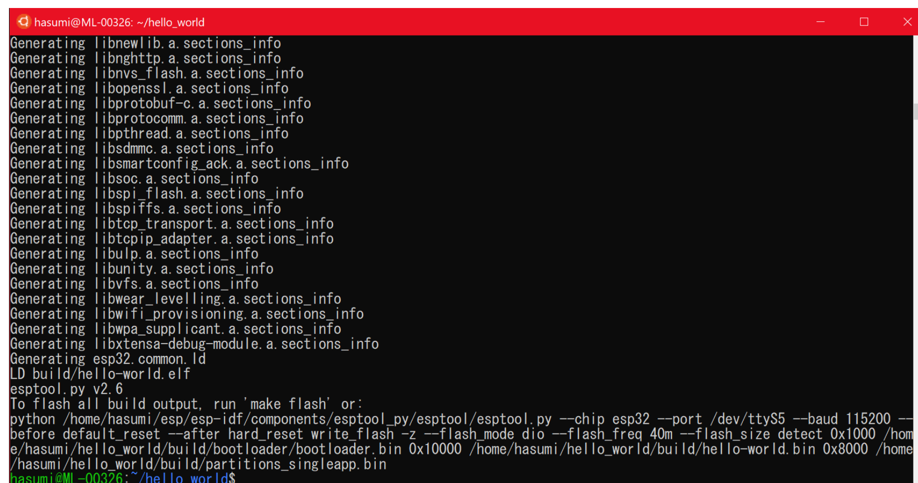
```
cp -r $IDF_PATH/examples/get-started/hello_world $HOME/esp
cd $HOME/esp/hello_world
make
```

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のインストール

homebrewが導入済みであることを想定し、手順を説明します。homebrewを使用せずにrbenvをインストールしたい場合は、WSLの環境構築マニュアルの該当部分を参考にしてください。

rbenvとruby-buildをインストールし、パスを通すなどします。

```
brew install openssl readline zlib rbenv ruby-build
echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bash_profile
echo 'if which rbenv > /dev/null; then eval "$(rbenv init -)"; fi' >> ~/.bash_profile
source $HOME/.bash_profile
```

macOSにはRubyがインストールされていますが、せっかくなので新しいバージョンをインストールしましょう。

```
rbenv install 2.6.1
```

このときに `The Ruby zlib extension was not compiled.` などのエラーが出てしまうときは、以下のようにすることでインストールできるかもしれません。

```
RUBY_CONFIGURE_OPTS="--with-zlib-dir=$(brew --prefix zlib)" rbenv install 2.6.1
```

`zlib` のところが `readline` など別のライブラリかもしれません。適宜読み替えて対処してください。

最後に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。

```
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。

シリアルポートのデバイスファイル名を確認

USBポートにESP32開発ボードを接続していない状態で、 `ls -l /dev/cu.*` と打ってみてください。つぎに、ESP32開発ボードを繋いだ状態で、同じコマンドを打ってください。「USB to UART ブリッジドライバ」が正しくインストールできていれば、出力がひとつ増えるはずです。そしてそのデバイスファイルは「/dev/cu.SLAB_USBtoUART」のような名前であるはずです。

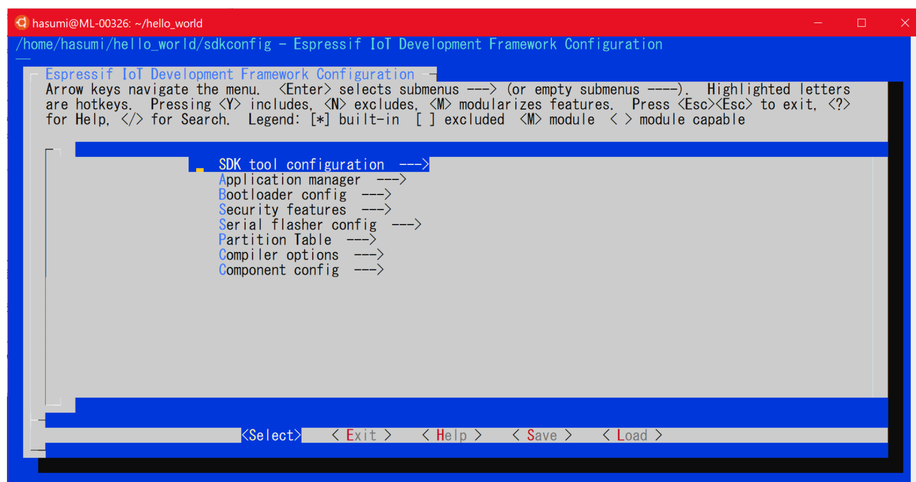
この文字列をメモしておいてください。

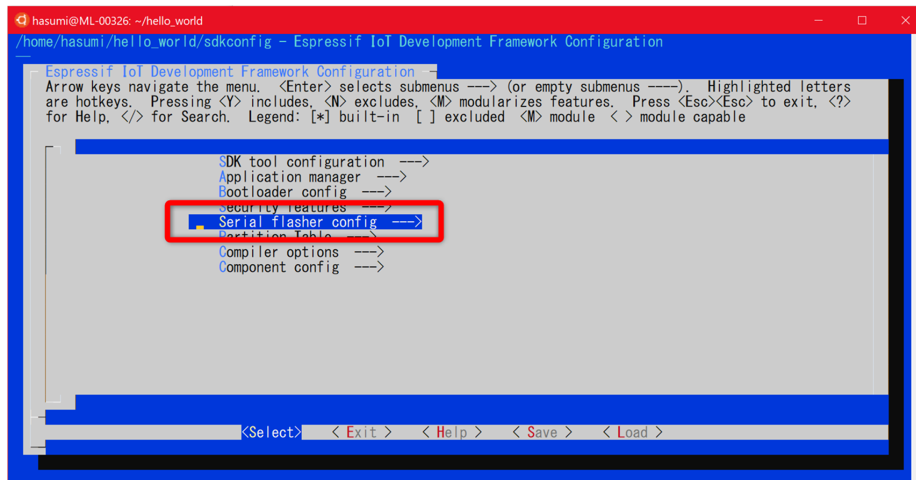
シリアルポートを設定

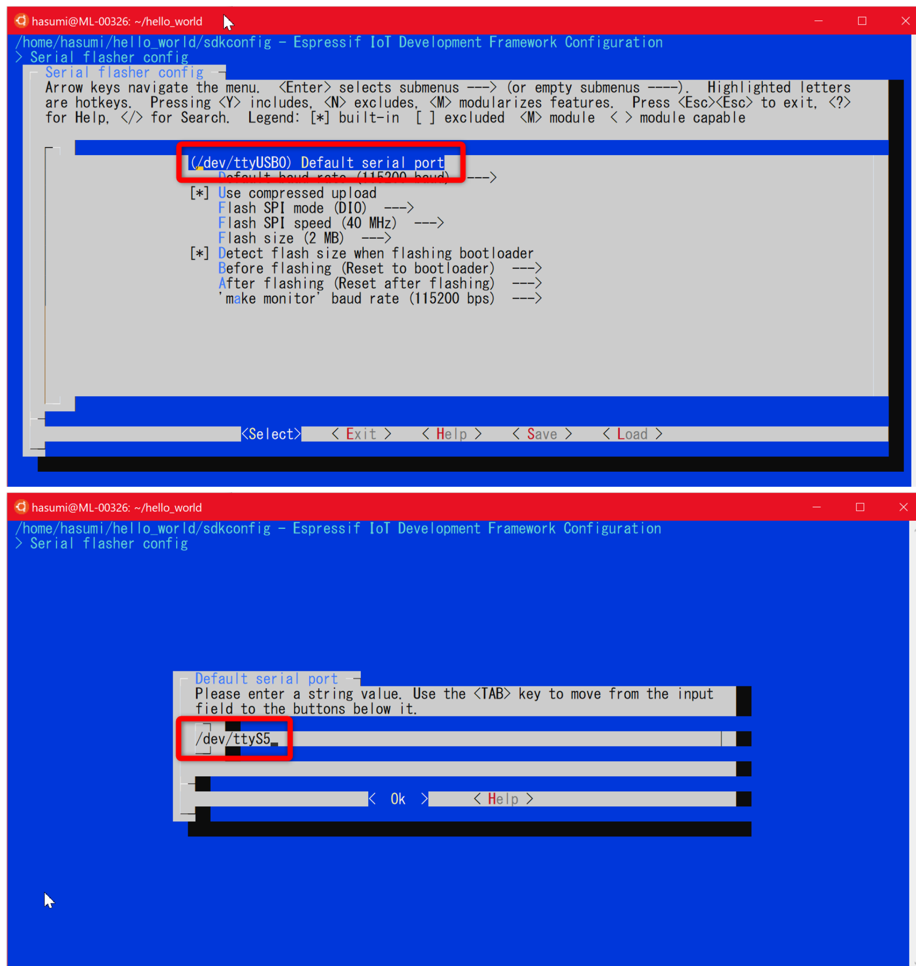
```
make menuconfig
```

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。

```
hasumi@ML-00326: ~/hello_world
I (160) esp_image: segment 5: paddr=0x000325fc vaddr=0x40086b70 size=0x01410 (  5136) load
0x40086b70: vTaskPlaceOnEventListRestricted at /home/hasumi/esp/esp-idf/components/freertos/tasks.c:4590

I (167) boot: Loaded app from partition at offset 0x10000
I (168) boot: Disabling RNG early entropy source...
I (169) cpu_start: Pro cpu up.
I (173) cpu_start: Application information:
I (178) cpu_start: Project name:     hello-world
I (183) cpu_start: App version:      1
I (188) cpu_start: Compile time:     18:52:10
I (193) cpu_start: Compile date:     Feb 27 2019
I (198) cpu_start: ESP-IDF:          v3.3-beta1-453-g140b6e389
I (205) cpu_start: Starting app cpu, entry point is 0x40080df0
0x40080df0: call_start_cpu1 at /home/hasumi/esp/esp-idf/components/esp32/cpu_start.c:265

I (196) cpu_start: App cpu up.
I (215) heap_init: Initializing. RAM available for dynamic allocation:
I (222) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (228) heap_init: At 3FFB2ED8 len 0002D128 (180 KiB): DRAM
I (234) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (241) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (247) heap_init: At 40087F80 len 00018080 (96 KiB): IRAM
I (253) cpu_start: Pro cpu start user code
I (271) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
Hello world!
This is ESP32 chip with 2 CPU cores, WiFi/BT/BLE, silicon revision 1, 4MB external flash
Restarting in 10 seconds...
```