macOS Setup Tool Chain for Apple Silicon

Diego Hernandez Gomez / Piguaso

macOS Setup Tool Chain for Apple Silicon

This guide details the steps to set up the development toolchain for the Raspberry Pi Pico on macOS with Apple Silicon processors.

Step 1: Install Homebrew

Homebrew is a very useful package manager for installing software on macOS. If you don't have Homebrew installed, you can install it by running the following command in your terminal:

```
/bin/bash -c "$(curl␣-fsSL␣https://raw.githubusercontent.com/Homebrew/install/
    HEAD/install.sh)"
```

Step 2: Download and Install the ARM Toolchain

Next, download and install the ARM GNU Toolchain compatible with Apple Silicon from the following link:

- ARM GNU Toolchain download page

Download the specific package for Apple Silicon computers:

- https://developer.arm.com/-/media/Files/downloads/gnu/13.3.rel1/binrel/arm-gnu-toolchain-13.3.rel1-darwin-arm64-arm-none-eabi.pkg

Open the `.pkg` file and follow the installer's instructions to complete the installation.

Verify the Installation

After installation, verify that the toolchain is correctly installed in the `/Applications` folder. Run the following command in the terminal to check the location:

```
ls /Applications/ArmGNUToolchain/13.3.rel1/arm-none-eabi
```

Step 3: Set Up the Environment

With Homebrew installed, run the following commands in the terminal to install `git` and `python3`.

```
# Install git
brew install git

# Install python3
brew install python3
```

Step 4: Create a Working Directory and Install the SDK and Examples

1. Create a directory for the SDK

```
mkdir pico
cd pico
```

2. Clone the SDK from GitHub

```
git clone -b master https://github.com/raspberrypi/pico-sdk.git
```

3. Initialize SDK submodules

```
cd pico-sdk
git submodule update --init
cd ..
```

4. Clone the examples

```
git clone -b master https://github.com/raspberrypi/pico-examples.git
```

Step 5: Set Environment Variables

Configure the necessary environment variables for the SDK and the toolchain.

```
export PICO_SDK_PATH=../../pico-sdk
export PICO_TOOLCHAIN_PATH=/Applications/ArmGNUToolchain/13.3.rel1/arm-none-eabi
```

Step 6: Create a Sample Project

1. Create a directory for the project

```
mkdir test
cd test
```

2. Copy the necessary configuration file

```
cp ../pico-sdk/external/pico_sdk_import.cmake .
```

3. Create the C source file (`test.c`)

```
nano test.c
```

This sample code in `test.c` turns the LED on and off at defined intervals. You can adjust the LED's on and off time by changing the values in `LED ON` and `LED OFF`.

```
#include "pico/stdlib.h"
#include "pico/cyw43_arch.h"

int main() {
    stdio_init_all();
    if (cyw43_arch_init()) {
        printf("Wi-Fi init failed");
        return -1;
    }
    while (true) {
        cyw43_arch_gpio_put(CYW43_WL_GPIO_LED_PIN, 1); // LED ON
        sleep_ms(250); // LED ON time
        cyw43_arch_gpio_put(CYW43_WL_GPIO_LED_PIN, 0); // LED OFF
        sleep_ms(250); // LED OFF time
    }
}
```

4. Create the `CMakeLists.txt` file

```
nano CMakeLists.txt
```

Insert the following content in `CMakeLists.txt`:

```
cmake_minimum_required(VERSION 3.13)
include(pico_sdk_import.cmake)
project(test_project C CXX ASM)
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)
pico_sdk_init()
add_executable(test test.c)
pico_add_extra_outputs(test)
target_link_libraries(test pico_cyw43_arch_none pico_stdlib)
```

Step 7: Build the Project

1. Create the build directory

```
mkdir build
cd build
```

2. Configure the build for the `pico_w` board

```
cmake -DPICO_BOARD=pico_w ..
```

3. Compile the project

```
make
```

Step 8: Finalization

If the build was successful, a `.uf2` file will be generated in the `build` directory. You can load this file onto your Raspberry Pi Pico W by holding down the `BOOTSEL` button while connecting it to your computer.