
ESP8266 USB Software Driver

I wanted to have USB on the ESP8266, and a while ago I saw on the ESP32 flier, it read: “Rich Peripherals” ... “Sorry, no USB!” I thought to myself, that is ridiculous. Of course there’s USB.

So, it was born. This is a software USB stack running on the ESP8266/ESP8285. It requires only one external resistor, between D- and 3.3V.

Limitations

- You cannot use SDKs newer than 1.5.X. (As of dec-02-2016, 1.5.4 is the latest SDK that has been verified to be compatible.)
- All ESP SDK 2.0 SDKs are incompatible.
- By default, the chip expects D- on GPIO 4 and D+ on GPIO 5 - but any GPIO pair may be used. It is important to note that both D- and D+ MUST be adjacent.
- This project only operates with low-speed USB (1.5 MBit/s) ideal for making peripherals, not for fake network devices and usb-serial bridges.

That said - you can still write “control” messages that communicate with the ESP8266.

Control messages are a great way to encapsulate your data since they handle all the framing and describing what function you wish to pass data for.

Resources

- Getting started guide
- Forum for discussion of development

Pre-built binaries

See bottom of this thread: <https://github.com/cnlohr/espusb/issues/40#issuecomment-377770936>

Examples

This project contains an example that simulates a usb keyboard and mouse. It also provides a web interface to actually remote control these virtual devices.

How to use

For a complete guide that includes information on how to install the toolchain or set up a docker container, go here: [Getting started guide](#) If you already have an environment with the toolchain, then here is what you wanna do:

Software

- Install libusb and recursively clone this repo:

```
1 sudo apt-get install libusb-1.0-0-dev
2 git clone --recursive https://github.com/cnlohr/espusb
3 cd espusb
```

- `user.cfg` contains some settings that you might have to change to get it working.
- Building/flashing the main binary

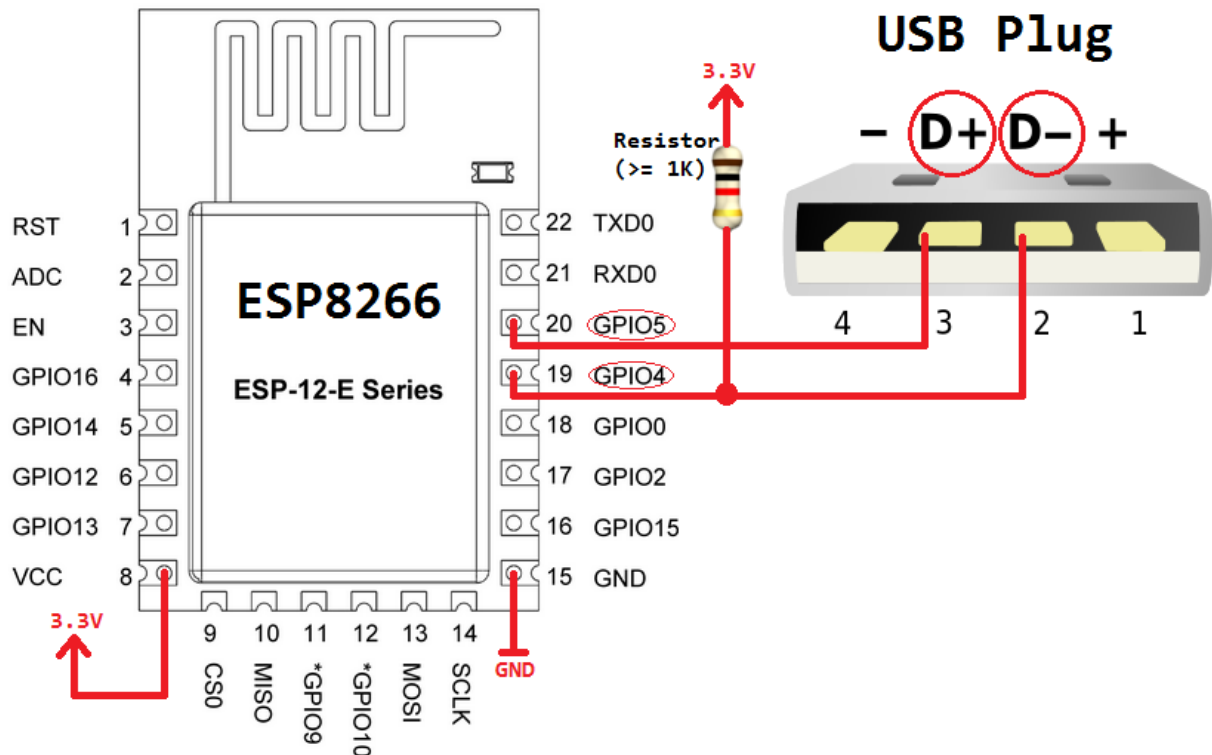
```
1 make ESP_ROOT=~/.esp8266/esp-open-sdk/ burn
```

- Building/flashing the web interface (for the example mentioned above)

```
1 make ESP_ROOT=~/.esp8266/esp-open-sdk/ burnweb
```

For more advanced building/flashing methods just run `make` without any parameters.

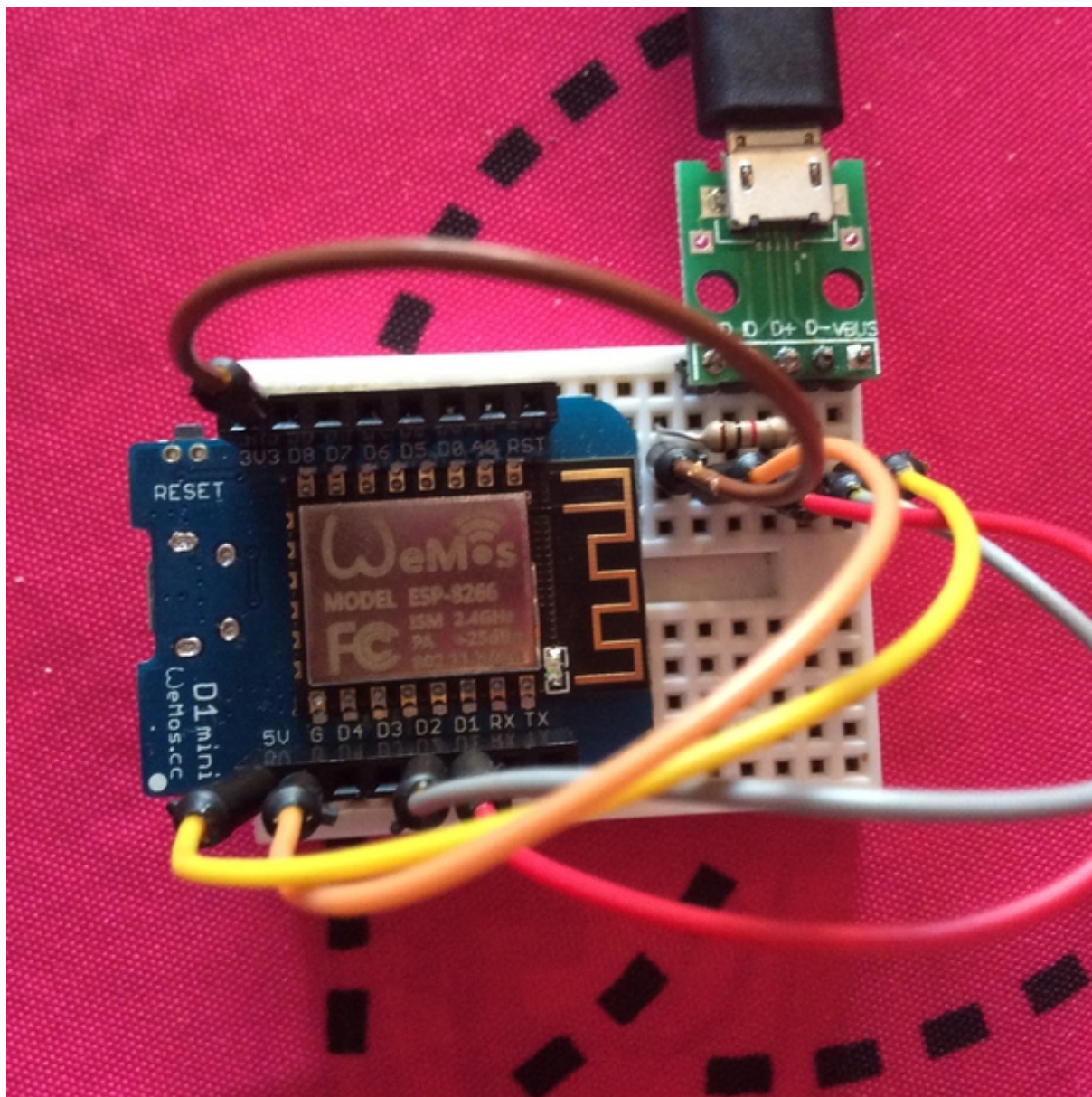
Hardware



Wemos D1 Mini

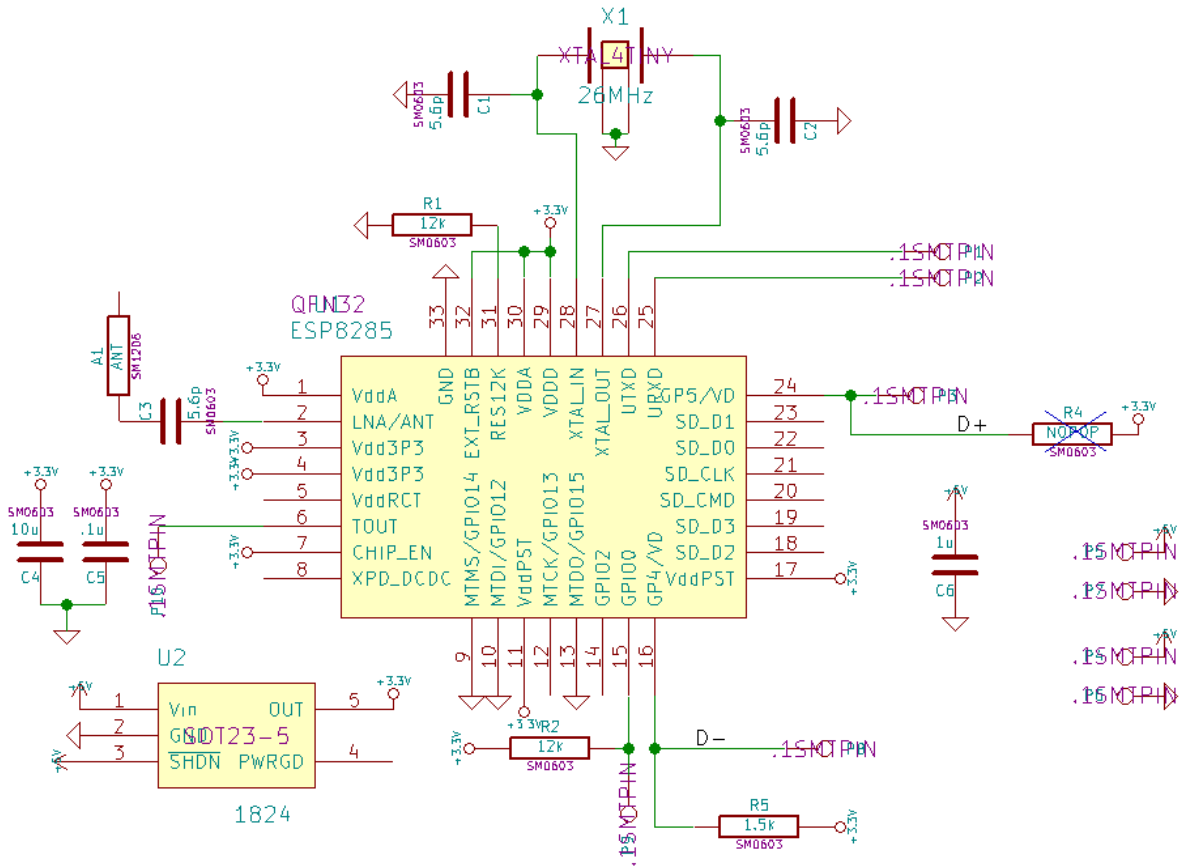
The Wemos D1 mini is a breadboard friendly ESP board.

You would need to flash the espusb firmware through the USB-serial connection beforehand, and then wire the GND, 5V, 3.3V, D1, D2 pins to the USB following the wiring instructions. Here is a picture of a setup with one breadboard:



Advanced information

Hardware



NOTE: GPIO12/14 do not need to be connected to ground. GPIO15 may be connected via a pull-down resistor. For stability, a pull-up on GPIO2 is recommended.

Also, checkout the hardware/ folder. It has kicad designs for my tiny board. Though they are for the ESP8285, the same pin configuration may be used to run on the ESP8266.

Memory Usage

This is typical memory usage for a two-endpoint (in addition to EP0) device.

In summary:

Total SRAM: 232 bytes + Descriptors (~317 bytes)

Total Flash/IRAM (Only iram, tables and usb_init can be stuck in slow flash): 1422 bytes

More details:

SRAM:

```
1 0000007c D usb_ramtable
2 0000006c B usb_internal_state
```

IRAM:

You must write: `usb_handle_custom_control` - the demos here hook it up to the command engine which allows self flashing on 1MB+ parts.

C functions:

```
1 0000002a T usb_pid_handle_ack
2 0000014f T usb_pid_handle_data
3 00000022 T usb_pid_handle_out
4 000000e9 T usb_pid_handle_in
5 00000002 T usb_pid_handle_sof
6 00000039 T usb_pid_handle_setup
```

From/To (in ASM):

```
1 40100f20 t usb_asm_start
2 401011ef t usb_asm_end
3 Total length of ASM: 2cf
```