Adafruitminiesptool Library Documentation

Release 1.0

ladyada

Mar 17, 2020

Contents

ROM loader for ESP chips, works with ESP8266 or ESP32. This is a 'no-stub' loader, so you can't read MD5 or firmware back on ESP8266.

See this document for protocol we're implementing: https://github.com/espressif/esptool/wiki/Serial-Protocol

See this for the 'original' code we're miniaturizing: https://github.com/espressif/esptool/blob/master/esptool.py

There's a very basic Arduino ROM loader here for ESP32: https://github.com/arduino-libraries/WiFiNINA/tree/master/examples/Tools/FirmwareUpdater

CHAPTER 1

Dependencies

This driver depends on:

- Adafruit CircuitPython

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the Adafruit library and driver bundle.

1.1 Installing from PyPI

On supported GNU/Linux systems like the Raspberry Pi, you can install the driver locally from PyPI. To install for current user:

```
pip3 install adafruit-circuitpython-miniesptool
```

To install system-wide (this may be required in some cases):

```
sudo pip3 install adafruit-circuitpython-miniesptool
```

To install in a virtual environment in your current project:

```
mkdir project-name && cd project-name
python3 -m venv .env
source .env/bin/activate
pip3 install adafruit-circuitpython-miniesptool
```

CHAPTER 2

Usage Example

Check the examples folder for demo sketches to upload firmware to ESP8266 and ESP32

CHAPTER 3

Contributing

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER 4

Documentation

For information on building library documentation, please check out this guide.

CHAPTER 5

Table of Contents

5.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/miniesptool_esp8266program.py

```
import time
import board
import busio
from digitalio import DigitalInOut
import adafruit_miniesptool

print("ESP8266 mini prog")

uart = busio.UART(board.TX, board.RX, baudrate=115200, timeout=1)
resetpin = DigitalInOut(board.D5)
gpio0pin = DigitalInOut(board.D6)
# On ESP8266 we will 'sync' to the baudrate in initialization
esptool = adafruit_miniesptool.miniesptool(
    uart, gpio0pin, resetpin, flashsize=1024 * 1024, baudrate=256000
)

esptool.debug = False
esptool.sync()

print("Synced")
print(esptool.chip_name)
print("MAC ADDR: ", [hex(i) for i in esptool.mac_addr])
esptool.flash_file("esp8266/AT_firmware_1.6.2.0.bin", 0x0)
# 0x3FC000 esp_init_data_default_v05.bin
esptool.flash_file("esp8266/esp_init_data_default_v05.bin", 0x3FC000)
# 0x3FE000 blank.bin
esptool.flash_file("esp8266/blank.bin", 0x3FE000)
```

(continues on next page)

(continued from previous page)

```
esptool.reset()
time.sleep(0.5)
```

5.2 adafruit_miniesptool

ROM loader for ESP chips, works with ESP8266 or ESP32. This is a 'no-stub' loader, so you can't read MD5 or firmware back on ESP8266.

See this document for protocol we're implementing: https://github.com/espressif/esptool/wiki/Serial-Protocol

See this for the 'original' code we're miniaturizing: https://github.com/espressif/esptool/blob/master/esptool.py

There's a very basic Arduino ROM loader here for ESP32: https://github.com/arduino-libraries/WiFiNINA/tree/master/examples/Tools/FirmwareUpdater

- Author(s): ladyada

5.2.1 Implementation Notes

Hardware:

Software and Dependencies:

- Adafruit CircuitPython firmware for the supported boards: https://github.com/adafruit/circuitpython/releases

class adafruit_miniesptool.**miniesptool**(*uart*, *gpio0_pin*, *reset_pin*, ***, *flashsize*, *baudrate=115200*)

A miniature version of esptool, a programming command line tool for ESP8266 and ESP32 chips. This version is minimized to work on CircuitPython boards, so you can burn ESP firmware direct from the CPy disk drive. Handy when you have an ESP module wired to a board and need to upload new AT firmware. Its slow! Expect a few minutes when programming 1 MB flash.

baudrate

The baudrate of the UART connection. On ESP8266 we cannot change this once we've started syncing. On ESP32 we must start at 115200 and then manually change to higher speeds if desired

check_command(*opcode*, *buffer*, *checksum=0*, *timeout=0.1*)

Send a command packet, check that the command succeeded and return a tuple with the value and data. See the ESP Serial Protocol for more details on what value/data are

static checksum(*data*, *state=239*)

Calculate checksum of a blob, as it is defined by the ROM

chip_name

The specific name of the chip, e.g. ESP8266EX, to the best of our ability to determine without a stub bootloader.

chip_type

ESP32 or ESP8266 based on which chip type we're talking to

debug

Print out all sent/received UART data plus some debugging output

flash_begin(***, *size=0*, *offset=0*)

Prepare for flashing by attaching SPI chip and erasing the number of blocks requred.

flash_block(*data*, *seq*, *timeout=0.1*)

Send one block of data to program into SPI Flash memory

flash_file(*filename*, *offset=0*, *md5=None*)
Program a full, uncompressed binary file into SPI Flash at a given offset. If an ESP32 and md5 string is passed in, will also verify memory. ESP8266 does not have checksum memory verification in ROM

get_erase_size(*offset*, *size*)
Calculate an erase size given a specific size in bytes. Provides a workaround for the bootloader erase bug on ESP8266.

get_response(*opcode*, *timeout=0.1*)
Read response data and decodes the slip packet, then parses out the value/data and returns as a tuple of (value, data) where each is a list of bytes

mac_addr
The MAC address burned into the OTP memory of the ESP chip

md5(*offset*, *size*)
On ESP32 we can ask the ROM bootloader to calculate an MD5 on the SPI flash memory, from a location over a size in bytes. Returns a string with the MD5 in lowercase

read_register(*reg*)
Read a register within the ESP chip RAM, returns a 4-element list

reset(*program_mode=False*)
Perform a hard-reset into ROM bootloader using gpio0 and reset

send_command(*opcode*, *buffer*)
Send a slip-encoded, checksummed command over the UART, does not check response

static slip_encode(*buffer*)
Take a bytearray buffer and return back a new bytearray where 0xdb is replaced with 0xdb 0xdd and 0xc0 is replaced with 0xdb 0xdc

sync()
Put into ROM bootload mode & attempt to synchronize with the ESP ROM bootloader, we will retry a few times

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index