
AdafruitESP32SPI Library Documentation

Release 1.0

ladyada

Mar 17, 2020

Contents

1	Dependencies	3
2	Installing from PyPI	5
3	Usage Example	7
4	Contributing	9
5	Documentation	11
6	Table of Contents	13
6.1	Simple test	13
6.2	Other Examples	14
6.3	adafruit_esp32spi	17
6.3.1	Implementation Notes	17
6.4	adafruit_esp32spi_socket	21
7	Indices and tables	23
	Python Module Index	25
	Index	27

CircuitPython driver library for using ESP32 as WiFi co-processor using SPI. The companion firmware is available on [GitHub](#). Please be sure to check the example code for any specific firmware version dependencies that may exist.

CHAPTER 1

Dependencies

This driver depends on:

- [Adafruit CircuitPython](#)
- [Bus Device](#)

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

CHAPTER 2

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-esp32spi
```

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

```
sudo pip3 install adafruit-circuitpython-esp32spi
```

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-esp32spi
```


CHAPTER 3

Usage Example

Check the examples folder for various demos for connecting and fetching data!

CHAPTER 4

Contributing

Contributions are welcome! Please read our [Code of Conduct](#) before contributing to help this project stay welcoming.

CHAPTER 5

Documentation

For information on building library documentation, please check out [this guide](#).

6.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/esp32spi_simpletest.py

```
1 import board
2 import busio
3 from digitalio import DigitalInOut
4 import adafruit_esp32spi.adafruit_esp32spi_socket as socket
5 from adafruit_esp32spi import adafruit_esp32spi
6 import adafruit_requests as requests
7
8 print("ESP32 SPI webclient test")
9
10 TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
11 JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.json"
12
13
14 # If you are using a board with pre-defined ESP32 Pins:
15 esp32_cs = DigitalInOut(board.ESP_CS)
16 esp32_ready = DigitalInOut(board.ESP_BUSY)
17 esp32_reset = DigitalInOut(board.ESP_RESET)
18
19 # If you have an ItsyBitsy Airlift:
20 # esp32_cs = DigitalInOut(board.D13)
21 # esp32_ready = DigitalInOut(board.D11)
22 # esp32_reset = DigitalInOut(board.D12)
23
24 # If you have an externally connected ESP32:
25 # esp32_cs = DigitalInOut(board.D9)
26 # esp32_ready = DigitalInOut(board.D10)
27 # esp32_reset = DigitalInOut(board.D5)
```

(continues on next page)

(continued from previous page)

```

28
29 spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
30 esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
31
32 requests.set_socket(socket, esp)
33
34 if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
35     print("ESP32 found and in idle mode")
36 print("Firmware vers.", esp.firmware_version)
37 print("MAC addr:", [hex(i) for i in esp.MAC_address])
38
39 for ap in esp.scan_networks():
40     print("\t%s\t\tRSSI: %d" % (str(ap["ssid"], "utf-8"), ap["rssi"]))
41
42 print("Connecting to AP...")
43 while not esp.is_connected:
44     try:
45         esp.connect_AP(b"MY_SSID_NAME", b"MY_SSID_PASSWORD")
46     except RuntimeError as e:
47         print("could not connect to AP, retrying: ", e)
48         continue
49 print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)
50 print("My IP address is", esp.pretty_ip(esp.ip_address))
51 print(
52     "IP lookup adafruit.com: %s" % esp.pretty_ip(esp.get_host_by_name("adafruit.com"))
53 )
54 print("Ping google.com: %d ms" % esp.ping("google.com"))
55
56 # esp._debug = True
57 print("Fetching text from", TEXT_URL)
58 r = requests.get(TEXT_URL)
59 print("-" * 40)
60 print(r.text)
61 print("-" * 40)
62 r.close()
63
64 print()
65 print("Fetching json from", JSON_URL)
66 r = requests.get(JSON_URL)
67 print("-" * 40)
68 print(r.json())
69 print("-" * 40)
70 r.close()
71
72 print("Done!")

```

6.2 Other Examples

Listing 2: examples/esp32spi_cheerlights.py

```

1 import time
2 import board
3 import busio
4 from digitalio import DigitalInOut

```

(continues on next page)

(continued from previous page)

```

5
6 from adafruit_esp32spi import adafruit_esp32spi
7 from adafruit_esp32spi import adafruit_esp32spi_wifimanager
8
9 import neopixel
10 import adafruit_fancyled.adafruit_fancyled as fancy
11
12 # Get wifi details and more from a secrets.py file
13 try:
14     from secrets import secrets
15 except ImportError:
16     print("WiFi secrets are kept in secrets.py, please add them there!")
17     raise
18
19 print("ESP32 SPI webclient test")
20
21 DATA_SOURCE = "https://api.thingspeak.com/channels/1417/feeds.json?results=1"
22 DATA_LOCATION = ["feeds", 0, "field2"]
23
24 esp32_cs = DigitalInOut(board.D9)
25 esp32_ready = DigitalInOut(board.D10)
26 esp32_reset = DigitalInOut(board.D5)
27 spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
28 esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
29 """Use below for Most Boards"""
30 status_light = neopixel.NeoPixel(
31     board.NEOPIXEL, 1, brightness=0.2
32 ) # Uncomment for Most Boards
33 """Uncomment below for ItsyBitsy M4"""
34 # status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.
35 ↪2)
36 wifi = adafruit_esp32spi_wifimanager.ESPSPiWiFiManager(esp, secrets, status_light)
37
38 # neopixels
39 pixels = neopixel.NeoPixel(board.A1, 16, brightness=0.3)
40 pixels.fill(0)
41
42 # we'll save the value in question
43 last_value = value = None
44
45 while True:
46     try:
47         print("Fetching json from", DATA_SOURCE)
48         response = wifi.get(DATA_SOURCE)
49         print(response.json())
50         value = response.json()
51         for key in DATA_LOCATION:
52             value = value[key]
53             print(value)
54         response.close()
55     except (ValueError, RuntimeError) as e:
56         print("Failed to get data, retrying\n", e)
57         wifi.reset()
58         continue
59
60     if not value:
61         continue

```

(continues on next page)

(continued from previous page)

```

61     if last_value != value:
62         color = int(value[1:], 16)
63         red = color >> 16 & 0xFF
64         green = color >> 8 & 0xFF
65         blue = color & 0xFF
66         gamma_corrected = fancy.gamma_adjust(fancy.CRGB(red, green, blue)).pack()
67
68         pixels.fill(gamma_corrected)
69         last_value = value
70     response = None
71     time.sleep(60)

```

Listing 3: examples/esp32spi_aino_post.py

```

1  import time
2  import board
3  import busio
4  from digitalio import DigitalInOut
5  import neopixel
6  from adafruit_esp32spi import adafruit_esp32spi
7  from adafruit_esp32spi import adafruit_esp32spi_wifimanager
8
9  print("ESP32 SPI webclient test")
10
11  # Get wifi details and more from a secrets.py file
12  try:
13      from secrets import secrets
14  except ImportError:
15      print("WiFi secrets are kept in secrets.py, please add them there!")
16      raise
17
18  # If you are using a board with pre-defined ESP32 Pins:
19  esp32_cs = DigitalInOut(board.ESP_CS)
20  esp32_ready = DigitalInOut(board.ESP_BUSY)
21  esp32_reset = DigitalInOut(board.ESP_RESET)
22
23  # If you have an externally connected ESP32:
24  # esp32_cs = DigitalInOut(board.D9)
25  # esp32_ready = DigitalInOut(board.D10)
26  # esp32_reset = DigitalInOut(board.D5)
27
28  spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
29  esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
30  """Use below for Most Boards"""
31  status_light = neopixel.NeoPixel(
32      board.NEOPIXEL, 1, brightness=0.2
33  ) # Uncomment for Most Boards
34  """Uncomment below for ItsyBitsy M4"""
35  # status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.
36  ↪ 2)
37  # Uncomment below for an externally defined RGB LED
38  # import adafruit_rgbled
39  # from adafruit_esp32spi import PWMOut
40  # RED_LED = PWMOut.PWMOut(esp, 26)
41  # GREEN_LED = PWMOut.PWMOut(esp, 27)
42  # BLUE_LED = PWMOut.PWMOut(esp, 25)

```

(continues on next page)

(continued from previous page)

```

42 # status_light = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)
43 wifi = adafruit_esp32spi_wifimanager.ESP32SPI_WiFiManager(esp, secrets, status_light)
44
45 counter = 0
46
47 while True:
48     try:
49         print("Posting data...", end="")
50         data = counter
51         feed = "test"
52         payload = {"value": data}
53         response = wifi.post(
54             "https://io.adafruit.com/api/v2/"
55             + secrets["aio_username"]
56             + "/feeds/"
57             + feed
58             + "/data",
59             json=payload,
60             headers={"X-AIO-KEY": secrets["aio_key"]},
61         )
62         print(response.json())
63         response.close()
64         counter = counter + 1
65         print("OK")
66     except (ValueError, RuntimeError) as e:
67         print("Failed to get data, retrying\n", e)
68         wifi.reset()
69         continue
70     response = None
71     time.sleep(15)

```

6.3 adafruit_esp32spi

CircuitPython driver library for using ESP32 as WiFi co-processor using SPI

- Author(s): ladyada

6.3.1 Implementation Notes

Hardware:

Software and Dependencies:

- Adafruit CircuitPython firmware for the supported boards: <https://github.com/adafruit/circuitpython/releases>
- Adafruit's Bus Device library: https://github.com/adafruit/Adafruit_CircuitPython_BusDevice

```

class adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol (spi,          cs_pin,
                                                         ready_pin,    reset_pin,
                                                         gpio0_pin=None, *,
                                                         debug=False)

```

A class that will talk to an ESP32 module programmed with special firmware that lets it act as a fast an efficient WiFi co-processor

MAC_address

A bytearray containing the MAC address of the ESP32

MAC_address_actual

A bytearray containing the actual MAC address of the ESP32

ap_listening

Returns if the ESP32 is in access point mode and is listening for connections

bssid

The MAC-formatted service set ID of the access point we're connected to

connect (*secrets*)

Connect to an access point using a secrets dictionary that contains a 'ssid' and 'password' entry

connect_AP (*ssid, password, timeout_s=10*)

Connect to an access point with given name and password. Will wait until specified timeout seconds and return on success or raise an exception on failure.

Parameters

- **ssid** – the SSID to connect to
- **passphrase** – the password of the access point
- **timeout_s** – number of seconds until we time out and fail to create AP

create_AP (*ssid, password, channel=1, timeout=10*)

Create an access point with the given name, password, and channel. Will wait until specified timeout seconds and return on success or raise an exception on failure.

Parameters

- **ssid** (*str*) – the SSID of the created Access Point. Must be less than 32 chars.
- **password** (*str*) – the password of the created Access Point. Must be 8-63 chars.
- **channel** (*int*) – channel of created Access Point (1 - 14).
- **timeout** (*int*) – number of seconds until we time out and fail to create AP

firmware_version

A string of the firmware version on the ESP32

get_host_by_name (*hostname*)

Convert a hostname to a packed 4-byte IP address. Returns a 4 bytearray

get_scan_networks ()

The results of the latest SSID scan. Returns a list of dictionaries with 'ssid', 'rssi', 'encryption', bssid, and channel entries, one for each AP found

get_socket ()

Request a socket from the ESP32, will allocate and return a number that can then be passed to the other socket commands

get_time ()

The current unix timestamp

ip_address

Our local IP address

is_connected

Whether the ESP32 is connected to an access point

network_data

A dictionary containing current connection details such as the 'ip_addr', 'netmask' and 'gateway'

ping (*dest, ttl=250*)

Ping a destination IP address or hostname, with a max time-to-live (ttl). Returns a millisecond timing value

pretty_ip (*ip*)

Converts a bytearray IP address to a dotted-quad string for printing

reset ()

Hard reset the ESP32 using the reset pin

rsssi

The receiving signal strength indicator for the access point we're connected to

scan_networks ()

Scan for visible access points, returns a list of access point details. Returns a list of dictionaries with 'ssid', 'rssi' and 'encryption' entries, one for each AP found

server_state (*socket_num*)

Get the state of the ESP32's internal reference server socket number

set_analog_read (*pin, atten=3*)

Get the analog input value of pin. Returns an int between 0 and 65536.

Parameters

- **pin** (*int*) – ESP32 GPIO pin to read from.
- **atten** (*int*) – attenuation constant

set_analog_write (*pin, analog_value*)

Set the analog output value of pin, using PWM.

Parameters

- **pin** (*int*) – ESP32 GPIO pin to write to.
- **value** (*float*) – 0=off 1.0=full on

set_certificate (*client_certificate*)

Sets client certificate. Must be called BEFORE a network connection is established. :param str client_certificate: User-provided .PEM certificate up to 1300 bytes.

set_digital_read (*pin*)

Get the digital input value of pin. Returns the boolean value of the pin.

Parameters **pin** (*int*) – ESP32 GPIO pin to read from.

set_digital_write (*pin, value*)

Set the digital output value of pin.

Parameters

- **pin** (*int*) – ESP32 GPIO pin to write to.
- **value** (*bool*) – Value for the pin.

set_esp_debug (*enabled*)

Enable/disable debug mode on the ESP32. Debug messages will be written to the ESP32's UART.

set_pin_mode (*pin, mode*)

Set the io mode for a GPIO pin.

Parameters

- **pin** (*int*) – ESP32 GPIO pin to set.
- **value** – direction for pin, digitalio.Direction or integer (0=input, 1=output).

set_private_key (*private_key*)

Sets private key. Must be called BEFORE a network connection is established. :param str private_key: User-provided .PEM file up to 1700 bytes.

socket_available (*socket_num*)

Determine how many bytes are waiting to be read on the socket

socket_close (*socket_num*)

Close a socket using the ESP32's internal reference number

socket_connect (*socket_num, dest, port, conn_mode=0*)

Open and verify we connected a socket to a destination IP address or hostname using the ESP32's internal reference number. By default we use 'conn_mode' TCP_MODE but can also use UDP_MODE or TLS_MODE (dest must be hostname for TLS_MODE!)

socket_connected (*socket_num*)

Test if a socket is connected to the destination, returns boolean true/false

socket_open (*socket_num, dest, port, conn_mode=0*)

Open a socket to a destination IP address or hostname using the ESP32's internal reference number. By default we use 'conn_mode' TCP_MODE but can also use UDP_MODE or TLS_MODE (dest must be hostname for TLS_MODE!)

socket_read (*socket_num, size*)

Read up to 'size' bytes from the socket number. Returns a bytearray

socket_status (*socket_num*)

Get the socket connection status, can be SOCKET_CLOSED, SOCKET_LISTEN, SOCKET_SYN_SENT, SOCKET_SYN_RCVD, SOCKET_ESTABLISHED, SOCKET_FIN_WAIT_1, SOCKET_FIN_WAIT_2, SOCKET_CLOSE_WAIT, SOCKET_CLOSING, SOCKET_LAST_ACK, or SOCKET_TIME_WAIT

socket_write (*socket_num, buffer*)

Write the bytearray buffer to a socket

ssid

The name of the access point we're connected to

start_scan_networks ()

Begin a scan of visible access points. Follow up with a call to 'get_scan_networks' for response

start_server (*port, socket_num, conn_mode=0, ip=None*)

Opens a server on the specified port, using the ESP32's internal reference number

status

The status of the ESP32 WiFi core. Can be WL_NO_SHIELD or WL_NO_MODULE (not found), WL_IDLE_STATUS, WL_NO_SSID_AVAIL, WL_SCAN_COMPLETED, WL_CONNECTED, WL_CONNECT_FAILED, WL_CONNECTION_LOST, WL_DISCONNECTED, WL_AP_LISTENING, WL_AP_CONNECTED, WL_AP_FAILED

unpretty_ip (*ip*)

Converts a dotted-quad string to a bytearray IP address

wifi_set_entenable ()

Enables WPA2 Enterprise mode

wifi_set_entidentity (*ident*)

Sets the WPA2 Enterprise anonymous identity

wifi_set_entpassword (*password*)
Sets the desired WPA2 Enterprise password

wifi_set_entusername (*username*)
Sets the desired WPA2 Enterprise username

wifi_set_network (*ssid*)
Tells the ESP32 to set the access point to the given ssid

wifi_set_passphrase (*ssid, passphrase*)
Sets the desired access point ssid and passphrase

6.4 adafruit_esp32spi_socket

A socket compatible interface thru the ESP SPI command set

- Author(s): ladyada

`adafruit_esp32spi.adafruit_esp32spi_socket.getaddrinfo` (*host, port, family=0, sock-
type=0, proto=0, flags=0*)

Given a hostname and a port name, return a 'socket.getaddrinfo' compatible list of tuples. Honestly, we ignore anything but host & port

`adafruit_esp32spi.adafruit_esp32spi_socket.set_interface` (*iface*)
Helper to set the global internet interface

class `adafruit_esp32spi.adafruit_esp32spi_socket.socket` (*family=2, type=1,
proto=0, fileno=None,
socknum=None*)

A simplified implementation of the Python 'socket' class, for connecting through an interface to a remote device

available ()

Returns how many bytes of data are available to be read (up to the MAX_PACKET length)

close ()

Close the socket, after reading whatever remains

connect (*address, conntype=None*)

Connect the socket to the 'address' (which can be 32bit packed IP or a hostname string). 'conntype' is an extra that may indicate SSL or not, depending on the underlying interface

connected ()

Whether or not we are connected to the socket

read (*size=0*)

Read up to 'size' bytes from the socket, this may be buffered internally! If 'size' isnt specified, return everything in the buffer. NOTE: This method is deprecated and will be removed.

readline ()

Attempt to return as many bytes as we can up to but not including ' '

recv (*bufsize=0*)

Reads some bytes from the connected remote address. :param int bufsize: maximum number of bytes to receive

send (*data*)

Send some data to the socket

settimeout (*value*)

Set the read timeout for sockets, if value is 0 it will block

socknum

The socket number

write (*data*)

Sends data to the socket. NOTE: This method is deprecated and will be removed.

CHAPTER 7

Indices and tables

- `genindex`
- `modindex`
- `search`

a

adafruit_esp32spi.adafruit_esp32spi, 17
adafruit_esp32spi.adafruit_esp32spi_socket,
21

A

adafruit_esp32spi.adafruit_esp32spi (module), 17

adafruit_esp32spi.adafruit_esp32spi_socket (module), 21

ap_listening (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol attribute), 18

available () (adafruit_esp32spi.adafruit_esp32spi_socket.socket method), 21

get_scan_networks () (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol method), 18

get_socket () (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol method), 18

get_time () (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol method), 18

getaddrinfo () (in module adafruit_esp32spi.adafruit_esp32spi_socket), 21

B

bssid (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol attribute), 18

C

close () (adafruit_esp32spi.adafruit_esp32spi_socket.socket method), 21

connect () (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol method), 18

connect () (adafruit_esp32spi.adafruit_esp32spi_socket.socket method), 21

connect_AP () (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol method), 18

connected () (adafruit_esp32spi.adafruit_esp32spi_socket.socket method), 21

create_AP () (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol method), 18

ip_address (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol attribute), 18

is_connected (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol attribute), 18

MAC_address (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol attribute), 17

MAC_address_actual (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol attribute), 18

E

ESP_SPIcontrol (class in adafruit_esp32spi.adafruit_esp32spi), 17

F

firmware_version (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol attribute), 18

G

get_host_by_name () (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol method), 18

N

network_data (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol attribute), 18

P

ping () (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol method), 19

pretty_ip () (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol method), 19

R

read () (adafruit_esp32spi.adafruit_esp32spi_socket.socket method), 21

readline () (adafruit_esp32spi.adafruit_esp32spi_socket.socket method), 21

recv() (adafruit_esp32spi.adafruit_esp32spi_socket.socket, method), 21

reset() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 19

rsssi (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, attribute), 19

S

scan_networks() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 19

send() (adafruit_esp32spi.adafruit_esp32spi_socket.socket, method), 21

server_state() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 19

set_analog_read() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 19

set_analog_write() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 19

set_certificate() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 19

set_digital_read() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 19

set_digital_write() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 19

set_esp_debug() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 19

set_interface() (in module adafruit_esp32spi.adafruit_esp32spi_socket), 21

set_pin_mode() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 19

set_private_key() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 20

settimeout() (adafruit_esp32spi.adafruit_esp32spi_socket.socket, method), 21

socket (class in adafruit_esp32spi.adafruit_esp32spi_socket), 21

socket_available() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 20

socket_close() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 20

socket_connect() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 20

socket_connected() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 20

socket_open() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 20

socket_read() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 20

socket_status() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 20

socket_write() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 20

ESP_SPIcontrol (adafruit_esp32spi.adafruit_esp32spi_socket.socket, attribute), 21

ssid (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, attribute), 20

start_server() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 20

status (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, attribute), 20

U

pretty_ip() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 20

W

wifi_set_entenable() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 20

wifi_set_entidentity() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 20

wifi_set_entpassword() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 20

wifi_set_entusername() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 21

wifi_set_network() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 21

wifi_set_passphrase() (adafruit_esp32spi.adafruit_esp32spi.ESP_SPIcontrol, method), 21

write() (adafruit_esp32spi.adafruit_esp32spi_socket.socket, method), 22