AdafruitESP32SPI Library Documentation

Release 1.0

ladyada

Oct 25, 2021

Contents

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

Documentation

API documentation for this library can be found on Read the Docs.

CHAPTER 5

Contributing

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

CHAPTER 6

Documentation

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

CHAPTER 7

Table of Contents

7.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/esp32spi_simpletest.py

```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import board
import busio
from digitalio import DigitalInOut
import adafruit_requests as requests
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

print("ESP32 SPI webclient test")

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.json"


# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)
```

(continues on next page)

(continued from previous page)

```
28
29  # If you have an AirLift Shield:
30  # esp32_cs = DigitalInOut(board.D10)
31  # esp32_ready = DigitalInOut(board.D7)
32  # esp32_reset = DigitalInOut(board.D5)
33
34  # If you have an AirLift Featherwing or ItsyBitsy Airlift:
35  # esp32_cs = DigitalInOut(board.D13)
36  # esp32_ready = DigitalInOut(board.D11)
37  # esp32_reset = DigitalInOut(board.D12)
38
39  # If you have an externally connected ESP32:
40  # NOTE: You may need to change the pins to reflect your wiring
41  # esp32_cs = DigitalInOut(board.D9)
42  # esp32_ready = DigitalInOut(board.D10)
43  # esp32_reset = DigitalInOut(board.D5)
44
45  spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
46  esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
47
48  requests.set_socket(socket, esp)
49
50  if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
51      print("ESP32 found and in idle mode")
52  print("Firmware vers.", esp.firmware_version)
53  print("MAC addr:", [hex(i) for i in esp.MAC_address])
54
55  for ap in esp.scan_networks():
56      print("\t%s\t\tRSSI: %d" % (str(ap["ssid"], "utf-8"), ap["rssi"]))
57
58  print("Connecting to AP...")
59  while not esp.is_connected:
60      try:
61          esp.connect_AP(secrets["ssid"], secrets["password"])
62      except RuntimeError as e:
63          print("could not connect to AP, retrying: ", e)
64          continue
65  print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)
66  print("My IP address is", esp.pretty_ip(esp.ip_address))
67  print(
68      "IP lookup adafruit.com: %s" % esp.pretty_ip(esp.get_host_by_name("adafruit.com"))
69  )
70  print("Ping google.com: %d ms" % esp.ping("google.com"))
71
72  # esp._debug = True
73  print("Fetching text from", TEXT_URL)
74  r = requests.get(TEXT_URL)
75  print("-" * 40)
76  print(r.text)
77  print("-" * 40)
78  r.close()
79
80  print()
81  print("Fetching json from", JSON_URL)
82  r = requests.get(JSON_URL)
83  print("-" * 40)
84  print(r.json())
```

(continues on next page)

(continued from previous page)

```
print("-" * 40)
r.close()

print("Done!")
```

7.2 Other Examples

Listing 2: examples/esp32spi_cheerlights.py

```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import board
import busio
from digitalio import DigitalInOut

import neopixel
import adafruit_fancyled.adafruit_fancyled as fancy

from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

print("ESP32 SPI webclient test")

DATA_SOURCE = "https://api.thingspeak.com/channels/1417/feeds.json?results=1"
DATA_LOCATION = ["feeds", 0, "field2"]

esp32_cs = DigitalInOut(board.D9)
esp32_ready = DigitalInOut(board.D10)
esp32_reset = DigitalInOut(board.D5)
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
"""Use below for Most Boards"""
status_light = neopixel.NeoPixel(
    board.NEOPIXEL, 1, brightness=0.2
)  # Uncomment for Most Boards
"""Uncomment below for ItsyBitsy M4"""
# status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.
↪2)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

# neopixels
pixels = neopixel.NeoPixel(board.A1, 16, brightness=0.3)
pixels.fill(0)

# we'll save the value in question
```

(continues on next page)

(continued from previous page)

```
last_value = value = None

while True:
    try:
        print("Fetching json from", DATA_SOURCE)
        response = wifi.get(DATA_SOURCE)
        print(response.json())
        value = response.json()
        for key in DATA_LOCATION:
            value = value[key]
            print(value)
        response.close()
    except (ValueError, RuntimeError) as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        continue

    if not value:
        continue
    if last_value != value:
        color = int(value[1:], 16)
        red = color >> 16 & 0xFF
        green = color >> 8 & 0xFF
        blue = color & 0xFF
        gamma_corrected = fancy.gamma_adjust(fancy.CRGB(red, green, blue)).pack()

        pixels.fill(gamma_corrected)
        last_value = value
    response = None
    time.sleep(60)
```

Listing 3: examples/esp32spi_aio_post.py

```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager

print("ESP32 SPI webclient test")

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
```

(continues on next page)

(continued from previous page)

```
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
"""Use below for Most Boards"""
status_light = neopixel.NeoPixel(
    board.NEOPIXEL, 1, brightness=0.2
)  # Uncomment for Most Boards
"""Uncomment below for ItsyBitsy M4"""
# status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.
↪2)
# Uncomment below for an externally defined RGB LED
# import adafruit_rgbled
# from adafruit_esp32spi import PWMOut
# RED_LED = PWMOut.PWMOut(esp, 26)
# GREEN_LED = PWMOut.PWMOut(esp, 27)
# BLUE_LED = PWMOut.PWMOut(esp, 25)
# status_light = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

counter = 0

while True:
    try:
        print("Posting data...", end="")
        data = counter
        feed = "test"
        payload = {"value": data}
        response = wifi.post(
            "https://io.adafruit.com/api/v2/"
            + secrets["aio_username"]
            + "/feeds/"
            + feed
            + "/data",
            json=payload,
            headers={"X-AIO-KEY": secrets["aio_key"]},
        )
        print(response.json())
        response.close()
        counter = counter + 1
        print("OK")
    except (ValueError, RuntimeError) as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        continue
    response = None
    time.sleep(15)
```

7.3 adafruit_esp32spi

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

- Author(s): ladyada

7.3.1 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

disconnect()
Disconnect from the access point

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

rssi
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*, *conn_mode=0*)
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

7.4 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*, *socktype=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=0*, *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. Will only return an empty string after the configured timeout.

Parameters **bufsize** (*int*) – 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 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

R

S

U

W