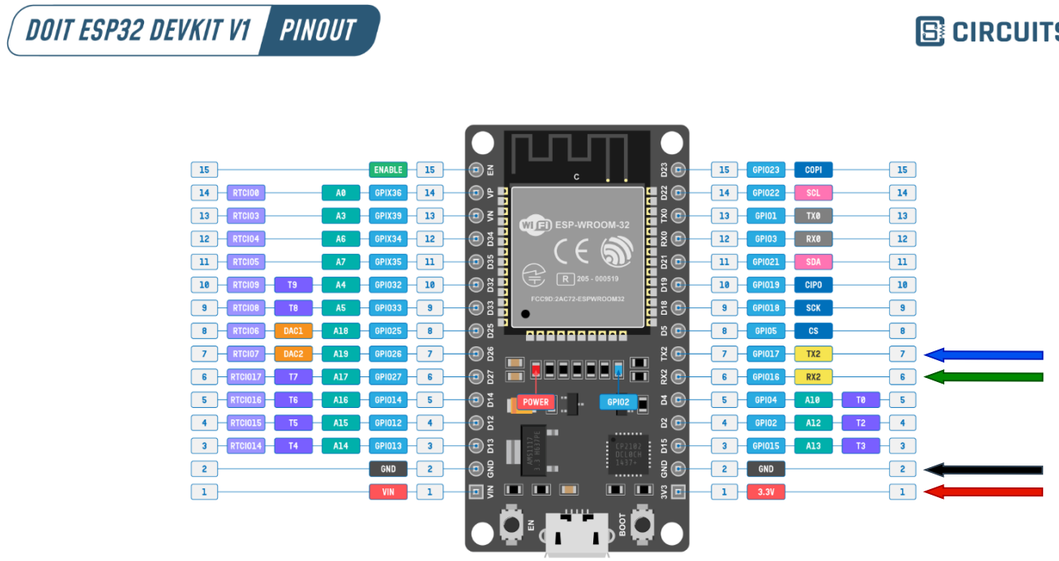
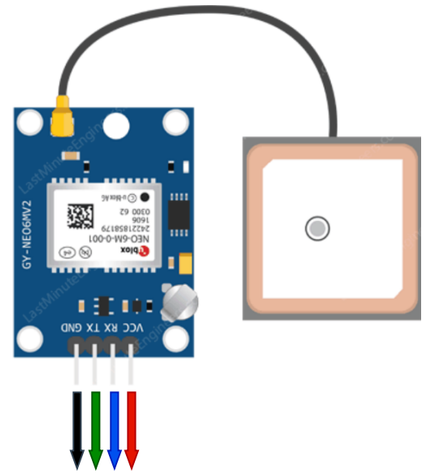
# ESP32 Pin Connection Diagram

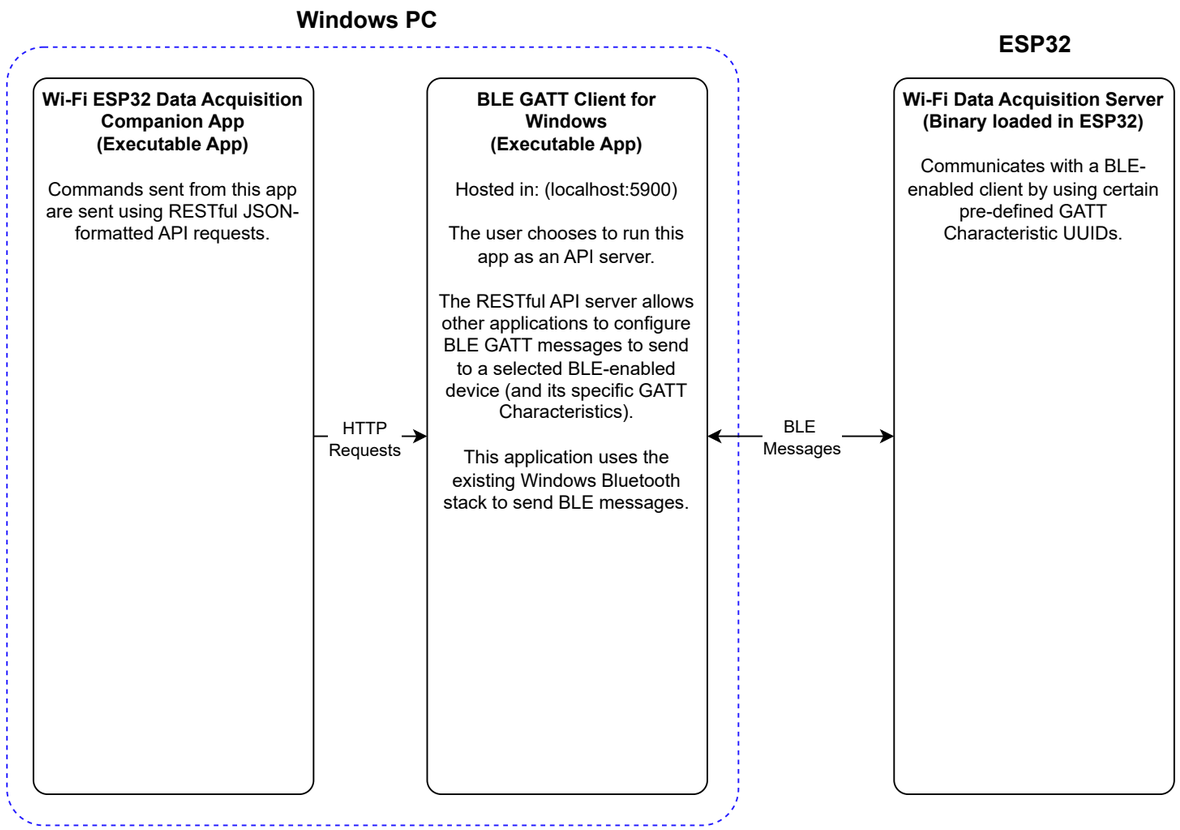
**Note:** Not all ESP32 board have the same pinout. The GPIO pins selected for UART have been carefully selected by referring to the pinout of my specific board mode and seeing if it maps out with the following ESP-IDF mapping for GPIO pins:
`{ESP-IDF Version}/esp-idf/components/soc/{compile target device}/include/soc/gpio_sig_map.h`

Refer to the information in **main/gps_library/gps_library.h** for more information.

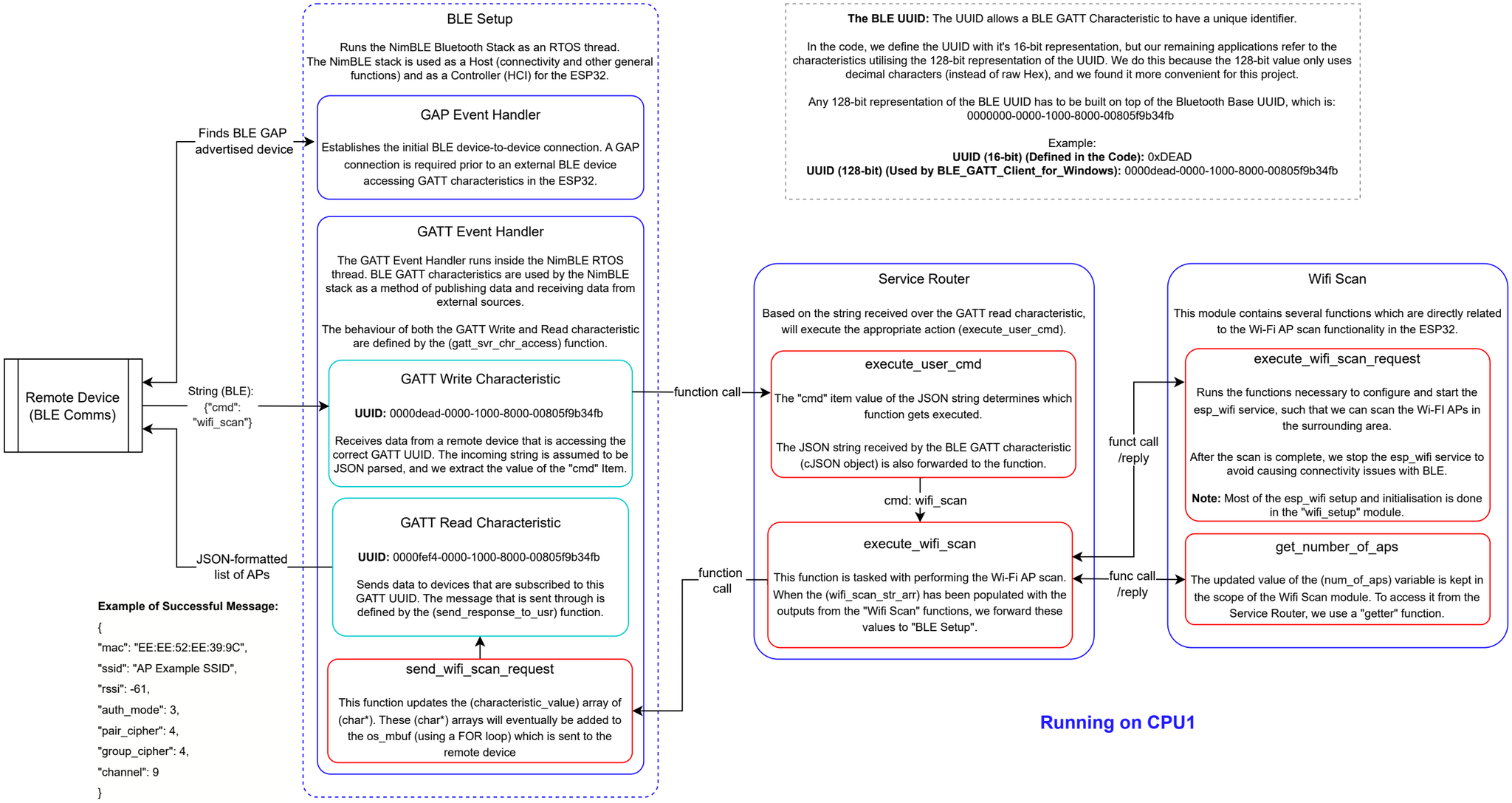


# Project Applications Coordination



# ESP32 (wifi_scan) Function




**The BLE UUID:** The UUID allows a BLE GATT Characteristic to have a unique identifier.

In the code, we define the UUID with it's 16-bit representation, but our remaining applications refer to the characteristics utilising the 128-bit representation of the UUID. We do this because the 128-bit value only uses decimal characters (instead of raw Hex), and we found it more convenient for this project.

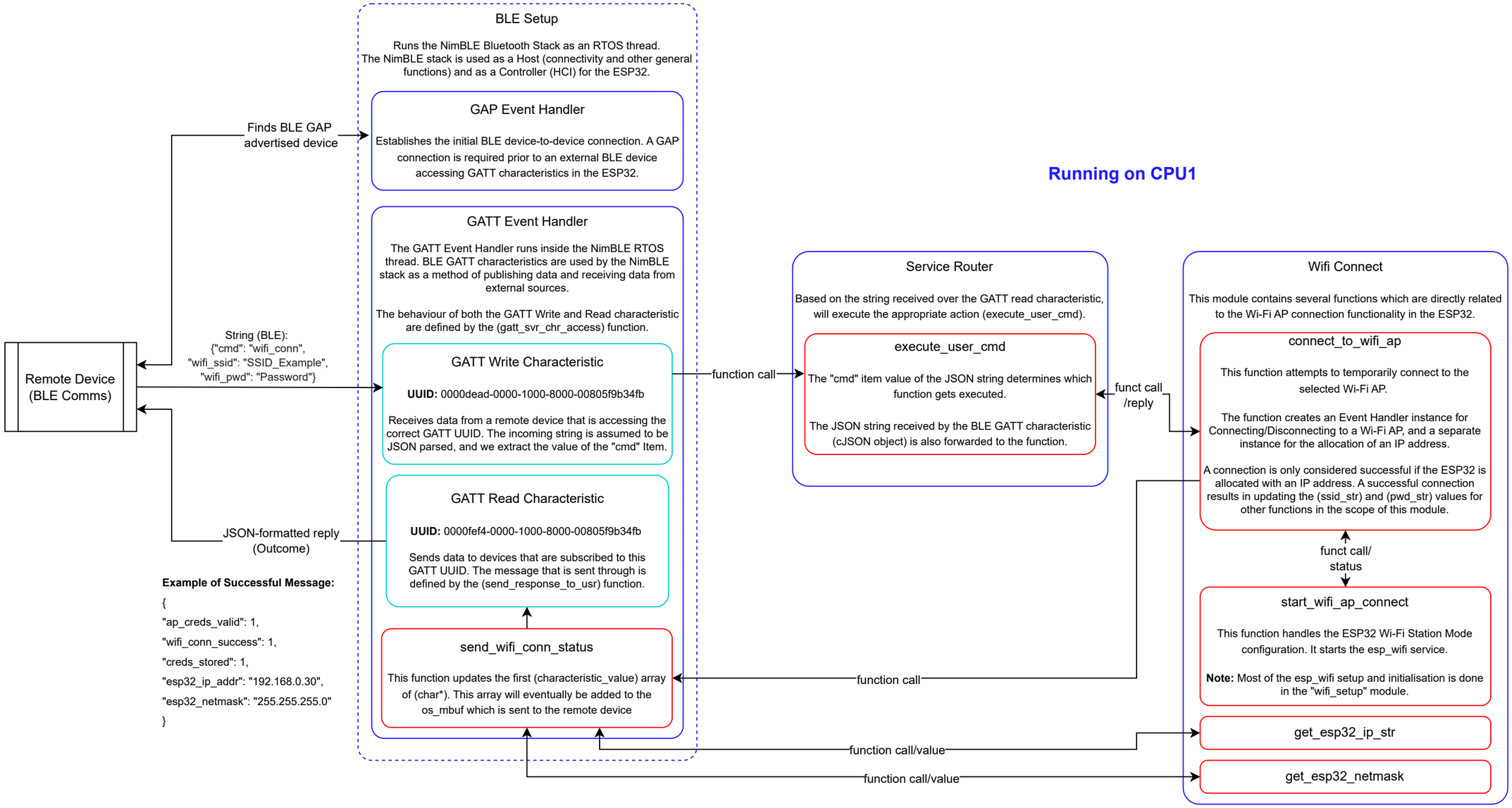
Any 128-bit representation of the BLE UUID has to be built on top of the Bluetooth Base UUID, which is: 0000000-0000-1000-8000-00805f9b34fb

Example:
**UUID (16-bit) (Defined in the Code):** 0xDEAD
**UUID (128-bit) (Used by BLE_GATT_Client_for_Windows):** 0000dead-0000-1000-8000-00805f9b34fb

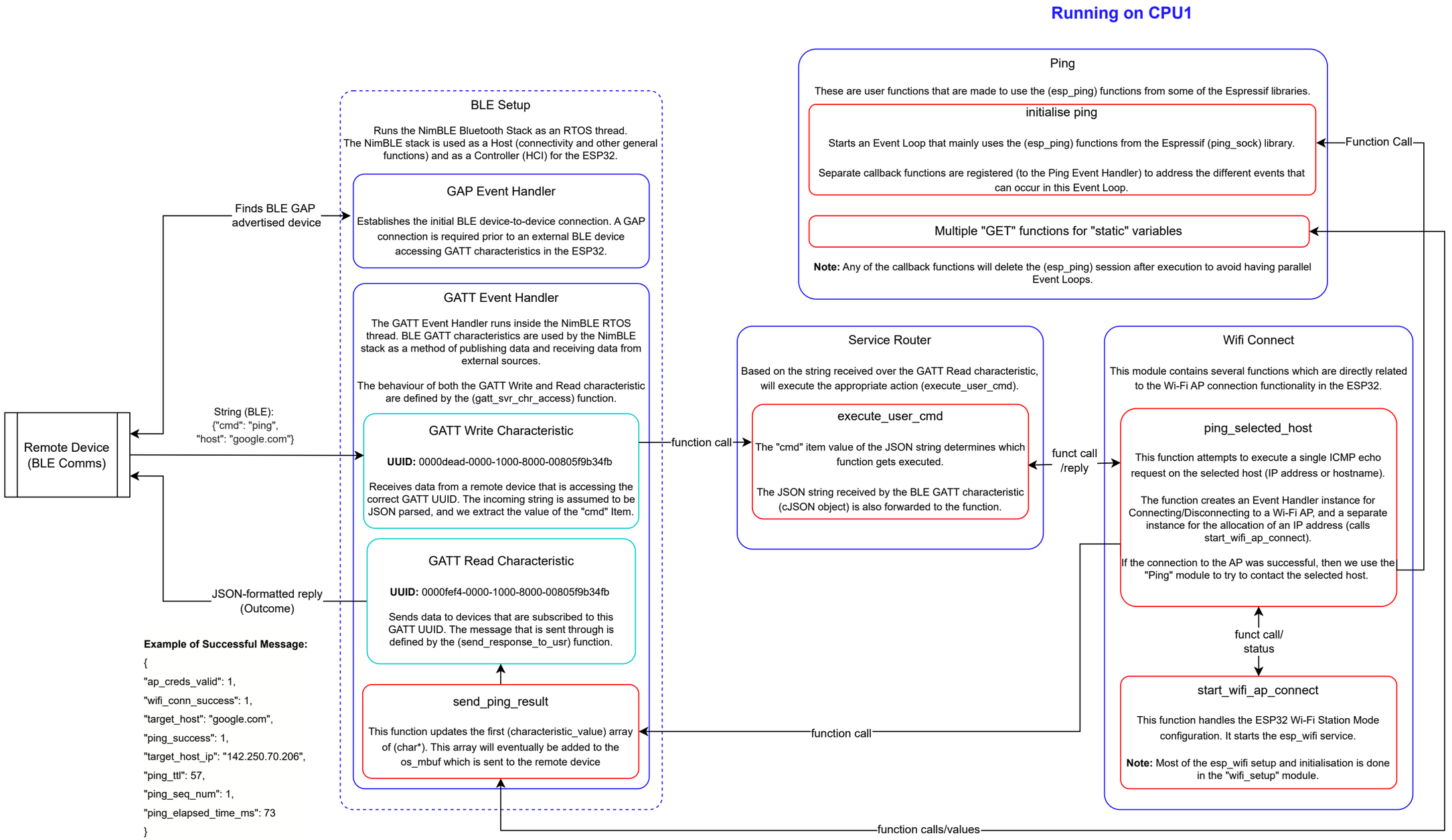
**Example of Successful Message:**


```
{
"mac": "EE:EE:52:EE:39:9C",
"ssid": "AP Example SSID",
"rssi": -61,
"auth_mode": 3,
"pair_cipher": 4,
"group_cipher": 4,
"channel": 9
}
```

# ESP32 (wifi_conn) Function



# ESP32 (ping) Function




**Example of Successful Message:**


```
{
"ap_creds_valid": 1,
"wifi_conn_success": 1,
"target_host": "google.com",
"ping_success": 1,
"target_host_ip": "142.250.70.206",
"ping_ttl": 57,
"ping_seq_num": 1,
"ping_elapsed_time_ms": 73
}
```

# ESP32 (gps_location) Function

## Running on CPU1

### BLE Setup

Runs the NimBLE Bluetooth Stack as an RTOS thread.
The NimBLE stack is used as a Host (connectivity and other general functions) and as a Controller (HCI) for the ESP32.

#### GAP Event Handler

Establishes the initial BLE device-to-device connection. A GAP connection is required prior to an external BLE device accessing GATT characteristics in the ESP32.

#### GATT Event Handler

The GATT Event Handler runs inside the NimBLE RTOS thread. BLE GATT characteristics are used by the NimBLE stack as a method of publishing data and receiving data from external sources.

The behaviour of both the GATT Write and Read characteristic are defined by the (gatt_svr_chr_access) function.

##### GATT Write Characteristic

**UUID:** 0000dead-0000-1000-8000-00805f9b34fb

Receives data from a remote device that is accessing the correct GATT UUID. The incoming string is assumed to be JSON parsed, and we extract the value of the "cmd" Item.

##### GATT Read Characteristic

**UUID:** 0000fef4-0000-1000-8000-00805f9b34fb

Sends data to devices that are subscribed to this GATT UUID. The message that is sent through is defined by the (send_response_to_usr) function.

#### send_gps_location

This function updates the first (characteristic_value) array of (char*). This array will eventually be added to the os_mbuf which is sent to the remote device.

#### Service Router

Based on the string received over the GATT read characteristic, will execute the appropriate action (execute_user_cmd).

##### execute_user_cmd

The "cmd" item value of the JSON string determines which function gets executed.

The JSON string received by the BLE GATT characteristic (cJSON object) is also forwarded to the function.

Remote Device (BLE Comms)

String (BLE): {"cmd": "gps_location"}

Finds BLE GAP advertised device

JSON-formatted reply (outcome)

function call

cmd: gps_location

**Example of Successful Message:**

```
{
 "gps_position":
 "35925.00,,,,,0,00,99.99,,,,,,*6C\r\n
$GPGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*30\r\n
$GPGSV,1,1,01,07,,,28*75\r\n
$GPGLL,,,,,235925.00,V,N*40\r\n
$GPRMC,235926.00,V,,,,,,,070624,,,N*73\r\n
$GPVTG,,,,,,,,,N*30\r\n
$GPGGA,235926.00,,,,,0,00,99.99,,,,,,*6F\r\n
$GPGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*30\r\n
$GPGSV,1,1,01,07,,,27*7A\r\n
$GPGLL,,,,,235926.00,V,N*43\r\n
$GPRMC,235927.00,V,,,,,,
}
```

## Running on CPU0

### GPS Library

#### GPS Thread

**Note:** The GPS location outputted from the NEO-6M module tends to be unreliable while its calibrating its location from the satellites. The module will start flashing a red LED once it has synchronised with the available GPS satellites (this process can take upwards of an hour).

##### gps_task

This task is started in CPU0 prior to the configuration of the BLE GATT Server. The task periodically reads the UART Tx stream from the NEO-6M module. The resulting GPS data stream is captured in a static (char) array that can be accessed from other threads (and cores).

**Note:** The NEO-6M outputs a data stream that includes several GPS Data Exchange formats such as (GPGGA, GPVTG, GPGLL, GPGSV, etc). This data is not parsed in any way before being sent out as a reply through BLE GATT.

#### get_latest_gps_data

The updated value of the (latest_gps_data) variable is kept in the scope of the GPS Library module. To access it from the scope of the BLE Setup, we use a "getter" function.

function call/value