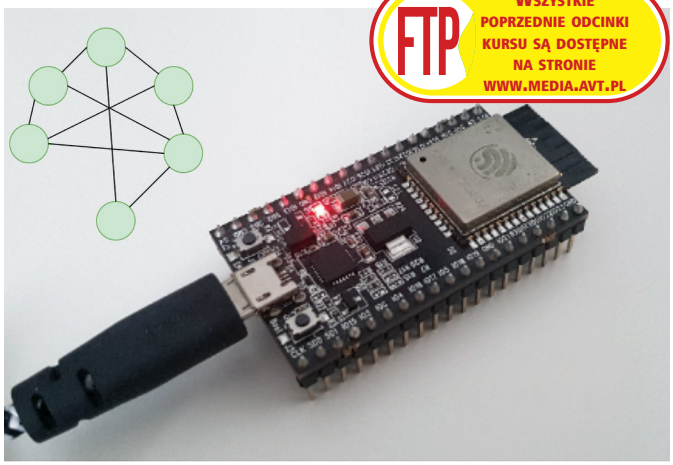
System komunikacji bezprzewodowej z użyciem modułów ESP32 oraz protokołu ESP-MESH (3)

Słowo „Mesh” – określające sieci bezprzewodowe o topologii kratowej – przylgnęło do technologii Bluetooth. Organizacja Bluetooth SIG na swojej stronie internetowej reklamuje się hasłem „Mesh networking is blue”. Choć topologia kratowa wykorzystywana jest z powodzeniem od czasów powstania pierwszych sieci komputerowych, to dopiero wprowadzenie standardu Bluetooth Mesh w połowie 2017 roku zaowocowało wzrostem zainteresowania konstruktorów urządzeń IoT tą technologią. Czy zatem przystępując do budowy sieci czujników, połączonych w łatwo konfigurowalną i skalowaną sieć Mesh, jesteśmy skazani na Bluetooth? Oczywiście, że nie!

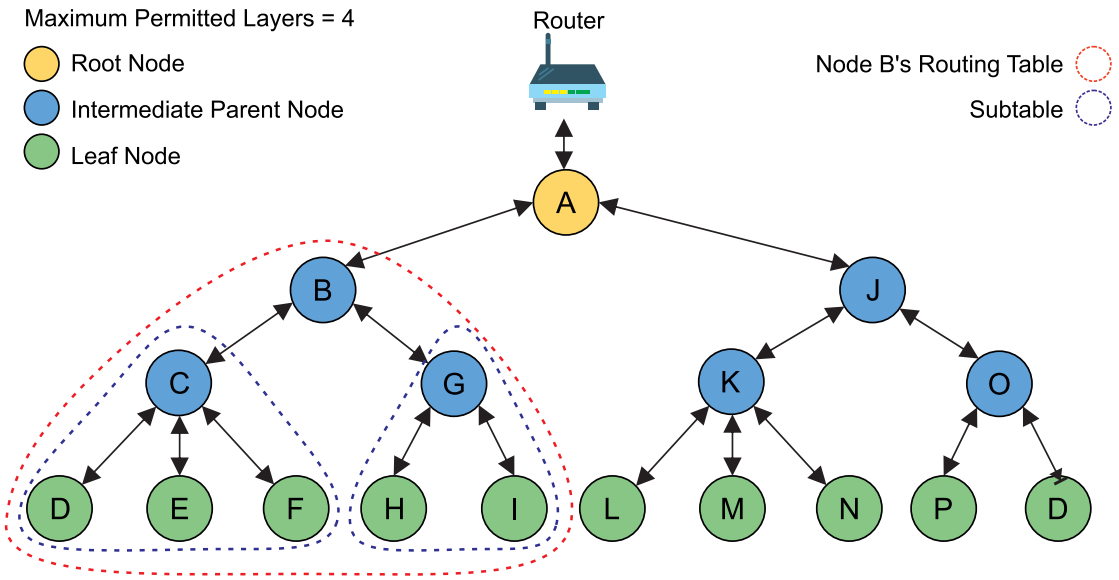


W poprzednich częściach artykułu omówione zostały zagadnienia związane z architekturą sieci, typami węzłów oraz „procesem wyborczym” węzła ROOT. Na potrzeby pierwszych testów sieci przygotowano również prostą aplikację z obsługą zdarzeń protokołu ESP-MESH. W tej części zajmiemy się rozbudowaniem kodu programu o proste mechanizmy komunikacji pomiędzy węzłami.

Ze względu na długość listingów są one dostępne na stronie artykułu: https://ep.com.pl – dla prenumeratorów, jeszcze przed ukazaniem się wydania papierowego. Równolegle do pobrania ze strony http://media.avt.pl.

ESP-MESH – tablice tras połączeń

W ramach sieci tworzonej z użyciem protokołu ESP-MESH każdy węzeł przechowuje i zarządza własną tablicą tras połączeń (*routing table*), niezbędną do prawidłowego przekierowania i dostarczenia pakietu danych do węzła docelowego. W skład tablicy tras połączeń wchodzą adresy MAC wszystkich urządzeń, podłączonych w ramach podsieci danego węzła, oraz adres węzła, który tę tablicę przechowuje. Dodatkowo tablica może zostać podzielona na podsieci, a więc tablice poszczególnych węzłów typu *child*. Przykładowy podział na tabele



Rysunek 6. Tablice tras połączeń dla sieci ESP-MESH (źródło: https://docs.espressif.com)

i podtabele został pokazany na **rysunku 6**. Węzeł B w tablicy tras połączeń zawiera węzły od B do I. Tablica ta może zostać podzielona na dwie mniejsze tabele – z węzłami od C do F oraz od G do I.

Zarządzanie tablicą tras w protokole ESP-MESH jest realizowane w sposób automatyczny. O dodaniu lub usunięciu nowego węzła, program użytkownika jest informowany poprzez zdarzenia `MESH_EVENT_ROUTING_TABLE_ADD` oraz `MESH_EVENT_ROUTING_TABLE_REMOVE`. Funkcja obsługi tych zdarzeń nie musi podejmować żadnych zadań, związanych z zarządzaniem tablicą. W większości przypadków ograniczy się wyłącznie do wyświetlenia prostych komunikatów w konsoli urządzenia, co pokazuje **listing 3**.

Użytkownik może uzyskać dostęp do informacji przechowywanych w tablicy tras poprzez wywołanie:

```
esp_err_t esp_mesh_get_routing_table (mesh_addr_t
*mac, int len, int *size);
```

Pierwszy argument funkcji stanowi wskaźnik na tablicę tras (`tabela struktur mesh_addr_t`). Drugim jest wskaźnik na zmienną typu `int`, w której zostaną umieszczone informacje o wielkości tablicy (wyrażone w bajtach). Informację o wielkości tablicy, bez danych o adresach poszczególnych węzłów, można uzyskać również z wykorzystaniem funkcji `esp_mesh_get_routing_table_size()`.

Analogiczny zestaw funkcji przygotowano również dla pobrania danych dotyczących podsieci wybranego węzła:

- `esp_mesh_get_subnet_nodes_list()`
- `esp_mesh_get_subnet_nodes_num()`

ESP-MESH – komunikacja pomiędzy węzłami

Posiadając już wiedzę o tablicach routingu oraz funkcjach związanych z dostępem do nich, przystąpmy do właściwej implementacji programu. Na potrzeby artykułu zrealizujmy najprostszy model komunikacji, w ramach której to węzeł ROOT przesyła w sposób cykliczny komunikaty „Hello World!" do wszystkich urządzeń w sieci (a więc do wszystkich urządzeń przechowywanych w jego tablicy tras połączeń).

Wprowadzanie zmian w kodzie rozpoczynamy od modyfikacji funkcji obsługi zdarzeń `mesh_event_handler()`, opisanej w pierwszej części artykułu. W tym celu, w obsłudze zdarzenia `MESH_EVENT_PARENT_CONNECTED`, informującego węzeł, że został podłączony do węzła rodzica i stał się pełnoprawnym uczestnikiem sieci, dodajemy teraz wywołanie funkcji `mesh_tx_rx_start()` (**listing 4**).

Zadaniem funkcji `mesh_tx_rx_start()`, jest utworzenie dwóch nowych zadań: `esp_mesh_tx_start` oraz `esp_mesh_rx_start`, które będą odpowiedzialne za nadawanie i odbiór pakietów w sieci. Ponieważ środowisko *esp-idf* bazuje na systemie *FreeRTOS*, zadanie to może zostać zrealizowane za pomocą standardowych dla tego systemu wywołań `xTaskCreate()` (**listing 5**).

W przyjętych na potrzeby artykułu założeniach, za nadawanie danych odpowiada wyłącznie węzeł ROOT, tak więc implementację zadania `esp_mesh_tx_task()`, rozpoczynamy od sprawdzenia funkcji, jaką dany węzeł pełni w sieci ESP-MESH (**listing 6**). Ponieważ funkcja przypisana do węzła może w sposób dynamiczny ulec zmianie, zadanie to nie jest usuwane dla węzłów niebędących węzłem ROOT:

Kolejnym zadaniem węzła ROOT jest pobranie danych do tablicy tras połączeń (**listing 7**) oraz przygotowanie wiadomości (tekstu „Hello World!"), która będzie przesłana do wszystkich węzłów w tablicy. Dane wysyłane i odbierane za pomocą funkcji `esp_mesh_send()` oraz `esp_mesh_recv()` muszą zostać „opakowane" w strukturę typu `mesh_data_t`:

```
struct mesh_data_t {
        uint8_t *data,
        uint16_t size,
        mesh_proto_t proto,
        mesh_tos_t tos
}
```

Pole *data* wskazuje na dane do wysłania, natomiast pole `size` określa ich rozmiar. Protokół transmisji danych może zostać zdefiniowany poprzez wyliczeniowy typ danych `mesh_proto_t`:

- `MESH_PROTO_BIN` (wartość domyślna)
- `MESH_PROTO_HTTP`
- `MESH_PROTO_JSON`
- `MESH_PROTO_MQTT`

Ostatnie z pól struktury `mesh_data_t` określa zachowanie sieci w przypadku wystąpienia potrzeby retransmisji pakietu. Aktualnie protokół ESP-MESH wspiera wyłącznie opcje retransmisji na poziomie punkt-punkt (`MESH_TOS_P2P`) lub pozwala na jej całkowite wyłączenie (`MESH_TOS_DEF`). Znając znaczenie poszczególnych pól struktury `mesh_data_t`, możemy do funkcji wysyłania komunikatów dopisać kolejne linie kodu, związane z przygotowaniem wiadomości „Hello World!":

```
mesh_data_t msg;
msg.data = (uint8_t*)"Hello World!";
msg.size = sizeof("Hello World!");
msg.proto = MESH_PROTO_BIN;
msg.tos = MESH_TOS_P2P;
```

Ostatnim elementem zadania `esp_mesh_tx_task()` jest wywołanie funkcji `esp_mesh_send()` (odpowiedzialnej za wysłanie pakietu w ramach sieci) do wszystkich węzłów znajdujących się w tablicy routingu. Ogólną postać funkcji `esp_mesh_send()` przedstawiono poniżej:

```
esp_err_t esp_mesh_send (const mesh_addr_t *to,
        const mesh_data_t *data,
```

```
        int flag,
        const mesh_opt_topt[],
        int opt_count
);
```

Pierwsze dwa argumenty funkcji to adres węzła docelowego (lub wartość `NULL`, jeżeli pakiet jest adresowany do węzła ROOT) oraz dane do wysłania, w postaci struktury `mesh_data_t`. Kolejny argument stanowią flagi wiadomości, które opisują kierunek transferu danych, w tym dwie najważniejsze: wartość 0, w przypadku wysłania wiadomości do węzła ROOT, oraz wartość `MESH_DATA_P2P`, jeżeli dane kierowane są do wewnętrznego węzła w sieci. Dwa ostatnie argumenty to opcje dodatkowe, związane z routowaniem pakietów do grup węzłów oraz na poziomie bramy dostępowej a węzłami wewnętrznymi (w omawianym przypadku opcje nie są wykorzystywane). Pełny kod zadania `esp_mesh_tx_task()` pokazuje **listing 8**.

Ostatnim i niezbędnym etapem do przeprowadzenia poprawnej kompilacji kodu jest implementacja zadania `esp_mesh_rx_task()`. Ponieważ tym razem nie stawiamy ograniczeń na funkcję, jaką pełni węzeł (węzeł ROOT realizuje również odczyt danych), konstrukcja zadania będzie wyłącznie obudowanym wywołaniem funkcji `esp_mesh_recv()`, co pokazuje **listing 9**.

Po kompilacji i wgraniu oprogramowania za pomocą poleceń:

```
idf.py build
idf.py p <identyfikator_urządzenia> flash (np. idf.
py p /dev/ttyUSB0 flash)
```

Sprawdźmy następnie poprawność komunikacji w ramach tworzonej sieci. Zaczynamy od włączenia zasilania pierwszego z węzłów, któremu z powodu braku innych urządzeń automatycznie zostaje przypisana rola węzła ROOT:

```
ESP-MESH: <MESH_EVENT_PARENT_CONNECTED>layer:0>1,
parent:53:66:50:76:b2:b0<ROOT>, ID:77:77:77:77:77:77
ESP-MESH: <MESH_EVENT_TODS_REACHABLE>state:0
ESP-MESH: <MESH_EVENT_ROOT_ADDRESS>root
address:24:0a:c4:03:ac:85
ESP-MESH: <IP_EVENT_STA_GOT_IP> IP: 192.168.0.17
```

Zgodnie z implementacją, detekcja zdarzenia `MESH_EVENT_PARENT_CONNECTED` powiązana jest z uruchomieniem zadań `esp_mesh_tx_task()` oraz `esp_mesh_rx_task()`:

```
ESP-MESH: [TXTASK][1/1] Sending 'Hello World'
message to child node (24:0a:c4:03:ac:84)
ESP-MESH: [RXTASK] Received 'Hello World!' message
from node (24:0a:c4:03:ac:84)
```

Ponieważ w tablicy tras połączeń węzła przechowywany jest również jego własny adres, urządzenie ROOT cyklicznie wysyła wiadomość „Hello World!" do samego siebie. Dołączenie kolejnych dwóch węzłów w sieci automatycznie zwiększa rozmiar tablicy routingu po stronie węzła ROOT i zwiększa liczbę iteracji pętli `for()`:

```
ESP-MESH: [TXTASK][1/1] Sending 'Hello World'
message to child node (24:0a:c4:03:ac:84)
ESP-MESH: [RXTASK] Received 'Hello World!' message
from node (24:0a:c4:03:ac:84)
...
ESP-MESH: <MESH_EVENT_ROUTING_TABLE_ADD>add 1, new:2
ESP-MESH: <MESH_EVENT_CHILD_CONNECTED>aid:1,
bc:dd:c2:c1:d0:d4
ESP-MESH: [TXTASK][1/2] Sending 'Hello World'
message to child node (24:0a:c4:03:ac:84)
ESP-MESH: [RXTASK] Received 'Hello World!' message
from node (24:0a:c4:03:ac:84)
ESP-MESH: [TXTASK][2/2] Sending 'Hello World'
message to child node (bc:dd:c2:c1:d0:d4)
...
ESP-MESH: <MESH_EVENT_ROUTING_TABLE_ADD>add 1,
new:3
ESP-MESH: <MESH_EVENT_CHILD_CONNECTED>aid:2,
bc:dd:c2:c2:ca:60
ESP-MESH: [TXTASK][1/3] Sending 'Hello World'
message to child node (24:0a:c4:03:ac:84)
ESP-MESH: [RXTASK] Received 'Hello World!' message
from node (24:0a:c4:03:ac:84)
ESP-MESH: [TXTASK][2/3] Sending 'Hello World'
message to child node (bc:dd:c2:c1:d0:d4)
ESP-MESH: [TXTASK][3/3] Sending 'Hello World'
message to child node (bc:dd:c2:c2:ca:60)
```

Pokazana powyżej przykładowa realizacja stanowi jedynie niewielki fragment możliwości dostarczanych przez protokół ESP-MESH. Bardziej szczegółowe opisy interfejsu API oraz funkcjonalności protokołu zostały udostępnione przez firmę Espressif pod następującymi adresami:

- https://bit.ly/339ocYG
- https://bit.ly/3n2jVxR

Łukasz Skalski
contact@lukasz-skalski.com