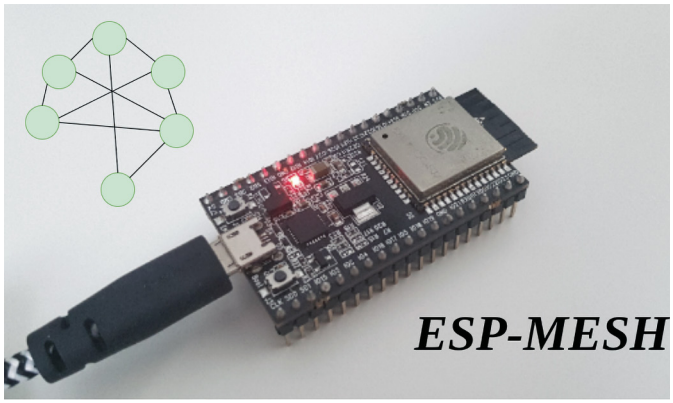
System komunikacji bezprzewodowej z użyciem modułów ESP32 oraz protokołu ESP-MESH (2)

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 poprzedniej części artykułu omówiliśmy topologie sieci bezprzewodowych oraz dokładną architekturę sieci ESP-MESH. W tej części zaczniemy od strony praktycznej.

Konfiguracja środowiska *esp-idf*

Sprawdźmy zatem w praktyce, jak realizowany jest proces automatycznej konfiguracji sieci oraz wymiany danych pomiędzy węzłami w ramach protokołu ESP-MESH. Od strony sprzętowej wykorzystamy do tego celu zestawy deweloperskie *ESP32-DEVKITC-32* [4], wyposażone w moduł *ESP-WROOM-32D* oraz konwerter USB-UART (bazujący na układzie Silabs CP2102). Całość zmontowana jest na płytce PCB z rastrem wyprowadzeń 2,54 mm (jak na **fotografii tytułowej**).

Działania rozpoczynamy od pobrania i konfiguracji środowiska programistycznego dla układów ESP32 – *esp-idf* (*Espressif IoT Development Framework*). Dla użytkowników systemu Windows przygotowano wygodny w użyciu instalator *esp-idftools-setup-2.3.exe* (zawierający m.in. kompilator, biblioteki oraz środowisko *OpenOCD*). Kompletna instrukcja dla systemów Windows (wraz z instalatorem) została udostępniona pod adresem: ***https://bit.ly/31AheLx***.

W przypadku systemu Linux i dystrybucji Debian/Ubuntu, przed procesem pobrania środowiska *esp-idf*, niezbędne jest samodzielne doinstalowanie wymaganych pakietów:

```
sudo apt-get install git wget libncurses-dev flex \
bison gperf python python-pip python-setuptools \
cmake ninja-build ccache libffi-dev libssl-dev
```

Niektóre wersje dystrybucji Debian/Ubuntu wciąż używają interpretera języka Python w wersji 2 (2.7) – środowisko *esp-idf* wymaga interpretera w wersji 3, zatem doinstalujmy brakujące pakiety:

```
sudo apt-get install python3 python3-pip \
python3-setuptools
```

Ustawmy domyślny interpreter języka:

```
sudo update-alternatives --install /usr/bin/python \
python /usr/bin/python3 10
```

Następnie, w lokalizacji *$HOME*, utwórzmy katalog *esp*, w którym umieszczone zostanie kompletne środowisko *esp-idf*:

```
cd $HOME
mkdir esp
```

W kolejnym kroku pobieramy *esp-idf* (wybierając ostatnią stabilną wersję oznaczoną numerem v4.0.1) za pomocą narzędzia *git*:

```
cd ~/esp
git clone -b v4.0.1 --recursive https://github.com/\
espressif/esp-idf.git
```

Instalacja środowiska w systemie Linux (dla systemu operacyjnego Windows przygotowano analogicznie pliki wsadowe BAT):

```
cd ~/esp/esp-idf
./install.sh
```

Nasz system jest już wyposażony w komplet niezbędnych narzędzi do poprawnej kompilacji kodu. Zgodnie ze wskazówkami wyświetlonymi przez program instalacyjny pozostaje nam jedynie ustawienie odpowiednich ścieżek dostępu do bibliotek i kompilatora. Najwygodniej zrobić to poprzez proste wywołanie dostarczonych wraz ze środowiskiem skryptów:

```
. $HOME/esp/esp-idf/export.sh
```

Możemy ustawić stały alias, wpisując:

```
alias get_idf='. $HOME/esp/esp-idf/export.sh'
```

ESP-MESH – aplikacja użytkownika

Kompilacja programu z wykorzystaniem środowiska *esp-idf* wymaga uprzedniego przygotowania odpowiednich plików z regułami kompilacji (pliki *CMake*/*Makefile*). Aby pominąć ten etap przygotowań, wykorzystamy przygotowany przez firmę *Espressif*, gotowy szablon projektu, który pobierzemy za pomocą narzędzia *git*:

```
git clone https://github.com/espressif/esp-idf-\
template.git
```

Edycję pliku *esp-idftemplate/main/main.c* rozpoczynamy od usunięcia domyślnej zawartości funkcji `app_main()`, a następnie

dodania linii inicjalizującej stos sieciowy *LwIP*. Teoretycznie inicjalizacji stosu wymaga tylko urządzenie pełniące funkcję węzła ROOT, które jest odpowiedzialne za komunikację z bramą dostępową. Ponieważ w realizowanym przykładzie funkcja węzła ROOT nie będzie przypisana na stałe, a każdy z węzłów – w drodze głosowania – może zostać wybrany na węzeł główny, inicjalizacja jest niezbędna dla wszystkich urządzeń:

```
tcpip_adapter_init();
tcpip_adapter_dhcps_stop(TCPIP_ADAPTER_IF_AP);
tcpip_adapter_dhcpc_stop(TCPIP_ADAPTER_IF_STA);
```

W następnym kroku tworzymy domyślną pętlę zdarzeń:

```
esp_event_loop_create_default();
```

Inicjalizujemy interfejs Wi-Fi oraz wskazujemy funkcję obsługi zdarzeń dla protokołu IP – `ip_event_handler()`, która poinformuje węzeł ROOT o przydzieleniu adresu IP (zdarzenie `IP_EVENT_STA_GOT_IP`):

```
wifi_init_config_t config =
WIFI_INIT_CONFIG_DEFAULT();
esp_wifi_init(&config);
esp_event_handler_register(
  IP_EVENT,
  IP_EVENT_STA_GOT_IP,
  &ip_event_handler,
  NULL
);
esp_wifi_set_storage(WIFI_STORAGE_FLASH);
esp_wifi_start();
```

Poza ciałem funkcji `app_main()`, umieśćmy zatem również prostą funkcję obsługi zdarzeń IP – `ip_event_handler()`, która poprzez komunikat w konsoli poinformuje nas o detekcji zdarzenia `IP_EVENT_STA_GOT_IP`:

```
void ip_event_handler(
  void *arg,
  esp_event_base_t event_base,
  int32_t event_id,
  void *event_data
){
  ip_event_got_ip_t *event = (ip_event_got_ip_t *)
event_data;
  ESP_LOGI(
    TAG,
    "<IP_EVENT_STA_GOT_IP> IP: %s",
    ip4addr_ntoa(&event→ip_info.ip)
  );
}
```

Po zainicjalizowaniu stosu LwIP oraz interfejsu Wi-Fi możemy przystąpić do właściwej konfiguracji sieci. Inicjalizację sieci rozpoczynamy od wywołania `esp_mesh_init()`, które sprawdza stan sieci Wi-Fi oraz przypisuje domyślnie ustawienia. O aktualnym stanie formowania sieci oraz zmianach w jej konfiguracji użytkownik informowany jest poprzez szereg zdarzeń `mesh_event_id_t`. Do implementacji funkcji obsługi tych zdarzeń oraz ich znaczenia przejdziemy w dalszej części artykułu. Na obecnym etapie wskażmy wyłącznie, jaka funkcja będzie odpowiedzialna za ich obsługę (w omawianym przypadku będzie to funkcja `mesh_event_handler()`):

```
esp_event_handler_register(
  MESH_EVENT,
  ESP_EVENT_ANY_ID,
  &mesh_event_handler,
  NULL
);
```

Kolejnym etapem jest konfiguracja parametrów sieci, tj. ustawienie unikalnego ID sieci oraz hasła, ustawienie numeru kanału, na którym będzie realizowana komunikacja i określenie nazwy SSID, hasła oraz metod szyfrowania dla bramy dostępowej/routera. Całość konfiguracji realizowana jest poprzez strukturę `mesh_cfg_t` oraz podstruktury `mesh_addr_t`, `mesh_router_t` i `mesg_ap_cfg_t`. Edycję kodu rozpoczynamy od inicjalizacji struktury `mesh_cfg_t` wartościami domyślnymi:

```
mesh_cfg_t cfg = MESH_INIT_CONFIG_DEFAULT();
```

Następnie ustawiamy unikalny numer identyfikacyjny sieci (wspólny dla wszystkich węzłów w ramach jednej sieci):

```
static const uint8_t MESH_ID[6] = {0x77, 0x77, 0x77,
0x77, 0x77, 0x77};
memcpy((uint8_t *) &cfg.mesh_id, MESH_ID, 6);
```

W kolejnym kroku konfigurujemy numer kanału, na którym będzie realizowana komunikacja (kanały 0…14), oraz ustawiamy SSID i hasło dla bramy dostępowej/routera:

```
#define MESH_CHANNEL          7
#define MESH_ROUTER_SSID      "network-ssid"
#define MESH_ROUTER_PASSWD    "network-password"

cfg.channel = MESH_CHANNEL;
cfg.router.ssid_len = strlen(MESH_ROUTER_SSID);
memcpy((uint8_t *) &cfg.router.ssid, MESH_ROUTER_
SSID, cfg.router.ssid_len);
memcpy((uint8_t *) &cfg.router.password, MESH_
ROUTER_PASSWD, strlen(MESH_ROUTER_PASSWD));
```

Następnie konfigurujemy hasło oraz szyfrowanie dla sieci Mesh oraz określamy maksymalną liczbę połączeń w ramach *softAP*:

```
#define MESH_AP_AUTHMODE      WIFI_AUTH_WPA2_PSK
#define MESH_AP_PASSWD        "Mesh AP Password"
#define MESH_AP_CONNECTIONS   6

esp_mesh_set_ap_authmode(MESH_AP_AUTHMODE);
cfg.mesh_ap.max_connection = MESH_AP_CONNECTIONS;
memcpy((uint8_t *) &cfg.mesh_ap.password, MESH_AP_
PASSWD, strlen(MESH_AP_PASSWD));
```

Ostatnim etapem konfiguracji jest ograniczenie liczby warstw tworzonej sieci oraz określenie wartości „progu wyborczego” dla procesu automatycznego wyboru węzła ROOT (wartości od 0.0 do 1.0):

```
#define MESH_MAX_LAYER              6

esp_mesh_set_max_layer(MESH_MAX_LAYER));
esp_mesh_set_vote_percentage(1));
```

Tak przygotowaną strukturę `mesh_cfg_t` przekazujemy do funkcji `esp_mesh_set_config()`, co pozwala nam ostatecznie uruchomić sieć za pomocą wywołania `esp_mesh_start()`:

```
esp_mesh_set_config(&cfg);
esp_mesh_start();
```

Pełny kod funkcji `app_main()`, wraz z dodatkowymi makrami `ESP_ERROR_CHECK()`, umożliwiającymi śledzenie poprawności wykonania poszczególnych etapów inicjalizacji i konfiguracji sieci, pokazano na **listingu 1** (dostępny w materiałach dodatkowych).

Do przeprowadzenia pierwszej, poprawnej kompilacji programu niezbędne jest zdefiniowanie wskazanej uprzednio funkcji obsługi zdarzeń sieci Mesh – `mesh_event_handler()`. Wybrane zdarzenia, zgłaszane przez sieć, zostały przedstawione w **tabeli 1**.

Bazując na danych z tabeli 1, zaimplementujmy funkcję obsługi zdarzeń – `mesh_event_handler()`, która na obecnym etapie realizacji projektu swoją funkcjonalność ograniczy niemal wyłącznie do logowania zdarzeń w konsoli, z wykorzystaniem wywołań `ESP_LOGI()`. Pełny kod funkcji `mesh_event_handler()` został pokazany na **listingu 2** (dostępny w materiałach dodatkowych).

Kompilacja i zaprogramowanie modułu

Dzięki wyposażeniu zestawów ESP32-DEVKITC-32 w konwerter USB-UART oraz układ autoresetu (układ sterowania wyprowadzeniami *EN* oraz *BOOT* za pomocą linii *RTS* i *DTR*), kompilacja i wgranie programu zostaje ograniczone do dwóch komend:

```
idf.py build
idf.py -p <identyfikator_
urządzenia> flash (np. idf.py -p /
dev/ttyUSB0 flash)
```

Do monitorowania pracy poszczególnych węzłów możemy wykorzystać polecenie `idf.py` *monitor* lub dowolny inny emulator terminala, np. `picocom`:

```
idf.py -p /dev/ttyUSB0 monitor
picocom /dev/ttyUSB1 -b 115200
```

Analizując komunikaty wyświetlane poprzez interfejs szeregowy, sprawdźmy poprawność formowania sieci. Rozpoczynamy od podłączenia pierwszego z węzłów. Z szeregu komunikatów, dotyczących m.in. konfiguracji interfejsu Wi-Fi, wybierzmy te związane z protokołem ESP-MESH. Pierwsze z komunikatów dotyczą uruchomienia sieci z wybranym numerem identyfikacyjnym:

```
ESP-MESH:
<MESH_EVENT_MESH_STARTED>ID:77:77:77:77:77:77
ESP-MESH: Mesh starts successfully: root not fixed
```

Następnie, po jednogłośnym „procesie wyborczym" (dostępny jest tylko jeden węzeł w systemie), urządzenie ROOT przystępuje do próby połączenia z wybranym punktem dostępowym/routerem. Poprawne połączenie spowoduje utworzenie pierwszej warstwy sieci – warstwę 0 stanowi punkt dostępowy o adresie 53:66:50:76:b2:b0; oraz warstwę drugą – węzeł ROOT o adresie 24:0a:c4:03:ac:85, któremu został przydzielony adres IP 192.168.0.17:

```
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
```

Włączenie zasilania drugiego z węzłów (o adresie sprzętowym bc:dd:c2:c1:d0:d4) skutkuje dołączeniem do węzła ROOT nowego węzła *child i* dodaniem nowego urządzenia do tablicy routingu. Po stronie urządzenia ROOT zostaną wówczas wyświetlone następujące komunikaty:

```
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
```

Natomiast w konsoli szeregowej dołączonego węzła znajdziemy:

```
ESP-MESH: <MESH_EVENT_PARENT_CONNECTED>layer:0-->2,
parent:24:0a:c4:03:ac:85<layer2>, 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
```

Tabela 1. Wybrane zdarzenia mesh_event_id_t

Zdarzenie mesh_event_id_t	Opis
MESH_EVENT_STARTED	sieć Mesh została uruchomiona
MESH_EVENT_STOPPED	sieć Mesh została zatrzymana
MESH_EVENT_CHANNEL_SWITCH	zmiana numeru kanału do komunikacji
MESH_EVENT_CHILD_CONNECTED	podłączenie węzła child do softAP
MESH_EVENT_CHILD_DISCONNECTED	odłączenie węzła child od softAP
MESH_EVENT_ROUTING_TABLE_ADD	dodanie nowego węzła do tablicy routingu
MESH_EVENT_ROUTING_TABLE_REMOVE	usunięcie węzła z tablicy routingu
MESH_EVENT_LAYER_CHANGE	zmiana numeru warstwy sieci
MESH_EVENT_VOTE_STARTED	rozpoczęcie procesu głosowania
MESH_EVENT_VOTE_STOPPED	zakończenie procesu głosowania
MESH_EVENT_ROOT_ADDRESS	adres węzła ROOT
MESH_EVENT_NETWORK_STATE	aktualny stan sieci

Tym samym została poprawnie uformowana druga warstwa sieci. Rodzicem nowego węzła jest węzeł ROOT o adresie 24:0a:c4:03:ac:85. W ostatnim kroku wykonamy test awarii węzła. Po wyłączeniu zasilania urządzenia ROOT jedyny węzeł drugiej warstwy sieci dokona detekcji zdarzenia i podejmie próbę ponownego połączenia z węzłem rodzicem. Każda nieudana próba zostaje zakończona komunikatem:

```
ESP-MESH: <MESH_EVENT_PARENT_DISCONNECTED>reason:2
```

Przy utrzymującym się stanie braku połączenia następuje automatyczne przełączenie węzła ROOT:

```
ESP-MESH: <MESH_EVENT_PARENT_CONNECTED>layer:2-->1,
parent:54:67:51:77:b3: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:bc:dd:c2:c1:d0:d5
ESP-MESH: <IP_EVENT_STA_GOT_IP> IP: 192.168.0.38
```

W kolejnej części

Następnym etapem będzie rozbudowanie kodu programu o komunikację na poziomie węzeł-węzeł oraz węzeł-root.

Łukasz Skalski
contact@lukasz-skalski.com

Przypisy:
[4] kamami.pl, 22.07.2020 ***https://bit.ly/2D2n1Qr***

Bibliografia:
- ***https://bit.ly/3jO9Zqe***
- ***https://bit.ly/2XbOy8P***
- ***https://bit.ly/33boD54***