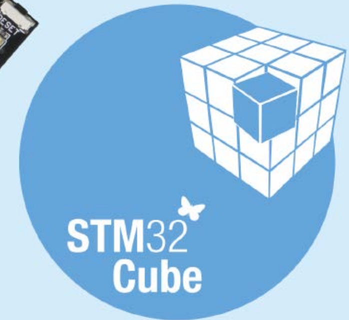
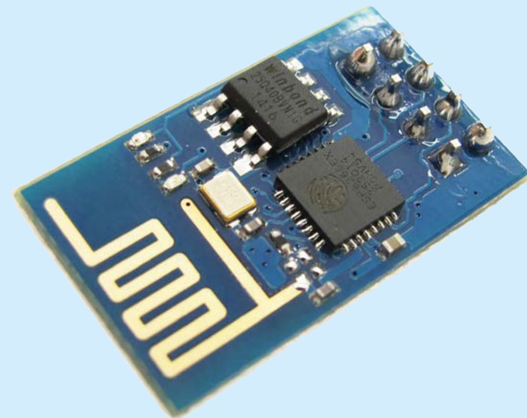


Programowanie układu STM32F4 (4)

W artykule opiszemy sposób dodania do omawianego mikrokontrolera STM32F411 obsługi sieci Wi-Fi i stosu TCP/IP. Wykorzystamy w tym celu, w roli karty sieciowej, kolejny mikrokontroler – układ ESP8266. Do komunikacji między układami użyty zostanie interfejs UART oraz polecenia AT Hayesa. Efektem naszych prac będzie bardzo prosty serwer WWW udostępniający stronę internetową, pozwalającą wybrać kolor świecenia diody RGB znajdującej się na płytce rozwojowej lub podłączonej do układu z zewnątrz.



Fotografia 1. Zdjęcie modułu ESP8266-01 (źródło: sklep internetowy kamami.pl)

Układ ESP8266 to nie tylko karta sieciowa. Jest on pełnoprawnym mikrokontrolerem, który również możemy programować. Ma on 32-bitowy rdzeń CPU Tensilica Xtensa LX106 taktowany przebiegiem o częstotliwości 80 MHz, 64 kB pamięci RAM dla kodu, 96 kB pamięci RAM dla danych oraz pamięć Flash o wielkości zależnej od wersji. Układ wspiera standardy sieci Wi-Fi 802.11 b/g/n. Występuje w kilku wersjach różniących się, oprócz pojemności pamięci Flash, także liczbą wyprowadzonych pinów i kształtem wbudowanej anteny oraz możliwością dołączenia zewnętrznej. W przykładzie posłużymy się modułem w wersji ESP8266-01 (**fotografia 1**), mającym wbudowaną antenę i wyprowadzone 8 pinów – w tym dwa piny interfejsu UART, zasilanie, masę oraz piny ogólnego przeznaczenia (GPIO).

Układ ESP operuje na napięciach 0…3,3 V, identycznie jak STM32F411 na używanej przeze mnie płytce rozwojowej KA-NUCLEO-F411CE. Domyślnie, wszystkie mikrokontrolery tej serii, mają wgrane oprogramowanie działające w roli karty sieci bezprzewodowej oraz implementacji stosu protokołów TCP/IP sterowanej przy pomocy poleceń AT Hayesa.

Dla układów z rodziny ESP8266 opracowano wiele ciekawych projektów, z czego, moim zdaniem – najbardziej ambitnym, jest programowalny nadajnik sygnału telewizyjnego – projekt „channel3”, autorstwa CNLohr (*https://goo.gl/ti7V7C*). Generuje on kolorowy obraz w standardzie NTSC i nadaje go falami radiowymi, poprzez jeden z pinów GPIO, do którego jest przyłączona antena. Generowanym obrazem są ruchome, również trójwymiarowe, wizualizacje. W trakcie emisji sygnału, mikrokontroler obsługuje także sieć Wi-Fi i pozwala na sterowanie nadawanym sygnałem przez stronę WWW.

Dlaczego więc, tak po prostu, nie zaprogramujemy tego układu? Programowanie mikrokontrolerów z serii ESP8266 nie jest tak łatwe, jak programowanie układów z rodziny STM32 z użyciem biblioteki HAL. Nie dysponujemy też tak zaawansowanym i jednocześnie łatwym w obsłudze narzędziem jak generator konfiguracji CubeMX. Cały czas pamiętać też musimy o tym, że ten sam rdzeń obsługuje zarówno nasz program, jak i wszystkie akcje związane z obsługą sieci Wi-Fi – programujemy więc zdarzenia wywoływane w odpowiednich momentach. Istnieją wprawdzie frameworki i środowiska, sprowadzające programowanie ESP8266 do programowania platformy „Arduino z Wi-Fi”, jednak znacznie ograniczają one możliwości tych układów. Następca serii ESP8266 – układ ESP32 posiada już dwa rdzenie CPU, z czego jeden może

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 97325, pass: 6yfwxr8q

być przeznaczony tylko do wykonywania naszego kodu, a drugi zajmuje się obsługą sieci Wi-Fi.

Zestaw poleceń AT Hayesa

Wspomniane już wyżej polecenia Hayesa, są równie stare jak, omówiony w poprzedniej części kursu, interfejs RS232. Od dawna służyły one i wciąż są wykorzystywane do sterowania wszelkiego rodzaju modemami – początkowo akustycznymi i podłączanymi do linii telefonicznych, gdzie polecenia przesyłane były przez interfejs COM – RS232, a obecnie do sterowania modemami GSM, UMTS, czy nawet LTE, przez interfejs USB.

Polecenia AT interpretowane przez układ ESP8266 mogą zostać użyte na cztery sposoby:

1. Dopisując po identyfikatorze polecenia znak zapytania, możemy odczytać obecnie ustawioną wartość danego parametru, np. nazwę sieci Wi-Fi, poleceniem: AT+CWJAP?.
2. Dopisując znak równości i wartość parametru, możemy ją zmienić, np. AT+CWJAP="NAZWA_SIECI","KLUCZ_SIECIOWY".
3. Dopisując znaki równości i zapytania, możemy dowiedzieć się, jakie wartości przyjmuje dany parametr, np. tryb pracy urządzenia – AT+CWMODE=?.
4. Podając sam identyfikator, możemy wykonać polecenie, np. przeskanować sieci Wi-Fi w pobliżu i wyświetlić ich nazwy – AT+CWLAP.

Jedno polecenie może występować w kilku wariantach, lecz nie musi. Zazwyczaj wykonanie polecenia kończy się zwrotem kilku linii odpowiedzi, z których ostatnia zawiera ciąg „OK", jeśli udało się wykonać polecenie lub „ERROR", w przypadku błędu. To niestety również nie jest regułą i część odpowiedzi kończy się innymi ciągami, jak na przykład „no change", w przypadku próby ponownego ustawienia takiej samej wartości dla danego parametru. Poniżej znajduje się spis popularnych poleceń, wraz z opisami ich działania.

- **AT** – testuje komunikacje z układem ESP8266. Zwraca ciąg „OK", jeśli mamy połączenie, a układ jest gotowy do pracy.
- **AT+RST** – powoduje ponowne uruchomienie układu ESP8266. Ostatnim komunikatem po restarcie jest ciąg „ready".
- **AT+CWMODE=ID_TRYBU_PRACY** – wybiera tryb pracy układu: 1) urządzenie końcowe Wi-Fi, 2) punkt dostępowy Wi-Fi, 3) urządzenie końcowe + punkt dostępowy.
- **AT+CWLAP** – skanuje sieci bezprzewodowe w okolicy i podaje ich parametry – rodzaj szyfrowania, nazwę sieci, adres MAC punktu dostępowego oraz siłę sygnału. Polecenie działa tylko w trybie pracy urządzenia końcowego.
- **AT+CWJAP="NAZWA_SIECI","KLUCZ_SIECIOWY"** – łączy się z siecią bezprzewodową o podanej nazwie i kluczu sieciowym (haśle). Polecenie działa tylko w trybie pracy urządzenia końcowego.
- **AT+CWQAP** – rozłącza się z siecią bezprzewodową. Polecenie działa tylko w trybie pracy urządzenia końcowego.
- **AT+CWSAP="NAZWA_SIECI","KLUCZ_SIECIOWY",NR_KANAŁU,SZYFROWANIE** – tworzy sieć bezprzewodową, rozgłaszaną przez urządzenie, w trybie pracy punktu dostępowego (polecenie działa tylko w tym trybie). Jako szyfrowanie możemy wybrać następujące wartości: **0)** Sieć otwarta (bez szyfrowania); **2)** Szyfrowanie WPA PSK; **3)** Szyfrowanie WPA2 PSK; **4)** Szyfrowanie WPA PSK + WPA2 PSK.
- **AT+CWLIF** – wyświetla listę urządzeń przyłączonych do punktu dostępowego (polecenie działa tylko w tym trybie).
- **AT+CWDHCP=TRYB_PRACY,0/1** – włącza (1) lub wyłącza (0) klienta i/lub serwer usługi sieciowej DHCP, pozwalającej na automatyczne przydzielanie i pobieranie adresu IP oraz pozostałej konfiguracji urządzenia końcowego. Domyślnie, usługa DHCP jest włączona zarówno jako serwer w trybie punktu dostępowego (moduł przydziela konfigurację innym urządzeniom które przyłączą się do jego sieci), jak i klient w trybie pracy urządzenia końcowego (urządzenie samo pobiera sobie konfigurację).
- **AT+CIPSTA="ADRES_IP" / AT+CIPAP="ADRES_IP"** – ustawia adres IP układu ESP8266, w trybie pracy urządzenia końcowego (AT+CIPSTA) i kolejno, punktu dostępowego (AT+CIPAP).
- **AT+CIPMUX=0/1** – włącza (1) / wyłącza (0) obsługę wielu połączeń równocześnie – maksymalnie pięciu, numerowanych od 0 do 4.
- **AT+CIPSERVER=0/1,PORT** – uruchamia (1) lub zamyka (0) serwer TCP – gniazdo nasłuchujące, na podanym porcie. Polecenie działa jedynie w trybie wielopołączeniowym (AT+CIPMUX=1).
- **AT+CIPSTART=[NR_POŁĄCZENIA,]"TCP/UDP","ADRES_IP_SERWERA",PORT** – nawiązuje połączenie TCP lub UDP z serwerem o podanym adresie IP lub domenie, na wskazanym porcie. W trybie wielopołączeniowym, pierwszym parametrem jest numer połączenia (0-4), nie występuje on w trybie jednopołączeniowym.
- **AT+CIPSEND=[NR_POŁĄCZENIA,]LICZBA_BAJTÓW** – wysyła strumień bajtów o podanym rozmiarze, przez nawiązane uprzednio, poleceniem AT+CIPSERVER lub AT+CIPSTART, połączenie. W trybie wielopołączeniowym należy podać numer połączenia, w trybie jednopołączeniowym, pomijamy pierwszy parametr.
- **AT+CIPCLOSE[=NR_POŁĄCZENIA]** – zamyka nawiązane uprzednio, poleceniem AT+CIPSERVER lub AT+CIPSTART, połączenie. W trybie wielopołączeniowym należy podać jego numer.
- **AT+CIFSR** – Zwraca obecny adres IP modułu ESP8266.

Po nawiązaniu połączenia TCP lub UDP, dane odebrane od strony przeciwnej, przesyłane są przez układ ESP8266 poprzedzone ciągiem +IPD[,NR_POŁĄCZENIA],ROZMIAR_DANYCH:.

Na podstawie powyższego spisu poleceń, możemy przygotować skrypt który prześlemy do modułu ESP8266 po uruchomieniu obu mikrokontrolerów. Sekwencja poleceń, która spowoduje przyłączenie się do sieci Wi-Fi, pobranie adresu IP oraz uruchomienie serwera-gniazda TCP na porcie 80, prezentuje się następująco:

```
AT+CWMODE=1
AT+CWJAP="NAZWA_SIECI","KLUCZ_SIECIOWY"
AT+CIPMODE=1
AT+CIPSERVER=1,80
```

Jeśli chcielibyśmy utworzyć sieć Wi-Fi na kanale 11, obsługującą szyfrowania WPA i WPA2 PSK, musimy wykonać poniższe polecenia:

```
AT+CWMODE=2
AT+CWSAP="NAZWA_SIECI","KLUCZ_SIECIOWY",11,4
AT+CIPMODE=1
AT+CIPSERVER=1,80
```

Czym właściwie jest TCP, porty i jak w uproszczeniu przebiega komunikacja w Internecie?

W spisie poleceń użyłem określeń: połączenie i gniazdo TCP. Czym jednak jest to połączenie? Komputery wymieniają się w Internecie danymi w pakietach IP. Internet z natury jest siecią bezpołączeniową. Każdy pakiet może być wysłany pod dowolny adres IP bez zestawiania połączenia. Jest on przełączany indywidualnie przez routery, na trasie od źródła do celu. Z samych pakietów IP nie korzysta się jednak zbyt często.

Protokół TCP działa nad protokołem IP. Z jednej strony, udostępnia on aplikacjom połączeniowy kanał, którym przesyłane

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 97325, pass: 6yfwxr8q

są strumienie danych między dwoma urządzeniami końcowymi, z drugiej, dzieli te dane i pakuje w pakiety IP, w celu przesyłu przez sieć. Zajmuje się też retransmisją zgubionych lub uszkodzonych na trasie pakietów i szeregowaniem ich we właściwej kolejności w miejscu odbioru. Serwer TCP to gniazdo nasłuchujące – oczekuje on na połączenia przychodzące na określonym porcie (w segmentach TCP przesyłane są 16-bitowe identyfikatory portów), nawiązywane również z określonego portu po stronie aplikacji klienckiej. Protokół TCP pozwala w ten sposób na używanie wielu serwerów i nawiązywanie wielu połączeń równocześnie.

Nad protokołem TCP (i UDP) działają jeszcze inne protokoły, standaryzujące sposób przesyłania danych w sieci. Jednym z tych protokołów jest HTTP, służący do przesyłania stron internetowych oraz plików z serwerów do przeglądarek WWW. Korzysta on z protokołu TCP i zwyczajowo działa na porcie 80.

Typowe żądanie HTTP wysyłane przez przeglądarkę zaczyna się od ciągu GET / HTTP/1.1, po którym następuje kilka linii nagłówków HTTP, w których przeglądarka przedstawia siebie oraz swoje możliwości, przesyłane są także parametry połączenia. Pierwszy znak / w powyższym początku zapytania to adres pliku który chcemy pobrać – / oznacza stronę główną. Jeśli wypełnimy na stronie WWW formularz (przesyłający dane metodą GET) i wyślemy jego zawartość, wszystkie wypełnione pola zostaną dopisane po znaku pytajnika do nazwy pliku, jako kolejne zmienne – np. GET /?red=100&green=50&blue=10 HTTP/1.1 i przesłane w kolejnym żądaniu. Odpowiedź składa się z ciągu HTTP/1.1 200 OK (jeśli nie wystąpił błąd) i serii nagłówków, po której przesyłany jest plik/strona WWW. Zazwyczaj, połączenie utrzymywane jest w celu dalszej komunikacji, możemy jednak z tego zrezygnować i uprościć działanie serwera, dopisując do odpowiedzi nagłówek Connection: close i zamykając połączenie, po odesłaniu odpowiedzi.

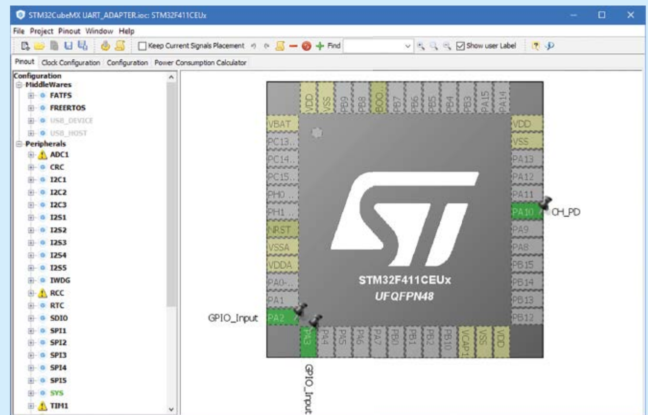
W dalszej części tego artykułu, przedstawiona została bardzo uproszczona implementacja serwera tego protokołu. Nasz program, połączy się z siecią Wi-Fi, uzyska automatycznie adres IP oraz pozostałe parametry konfiguracyjne, uruchomi serwer TCP na porcie 80 i będzie oczekiwał na połączenia przychodzące. Gdy takie zostanie nawiązane, przez przeglądarkę internetową, sprawdzony zostanie adres URI, do którego odwołuje się żądanie HTTP przesłane w połączeniu, następnie serwer odpowie na nie przesyłając stronę WWW z formularze umożliwiającym wybór koloru i zamknie połączenie. Jeśli w żądaniu przesłane zostaną również dane z formularza, program zmieni także kolor świecenia diody RGB.

Testowanie działania układu ESP8266

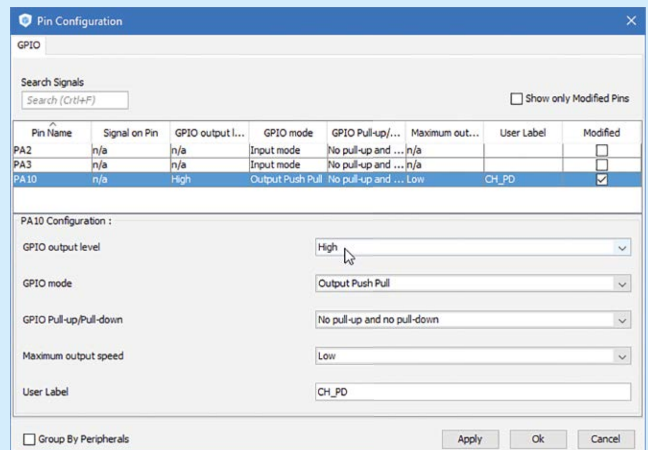
Jeśli chcemy przetestować działanie komend AT Hayesa i zobaczyć jak odpowiada na nie układ ESP8266, ale nie posiadamy adaptera USB<->UART, możemy w jego miejsce użyć płytki rozwojowej KA-NUCLEO lub dowolnej innej, z programatorem ST-LINK i wyprowadzonymi pinami RX/TX interfejsu UART, łączącego programator ST-LINK i programowany układ.

Na płytce KA-NUCLEO, dla zapewnienia kompatybilności wyprowadzeń z Arduino, są ta piny D0 (pin odbiorczy) i D1 (pin nadawczy), podłączone odpowiednio do wyprowadzeń PA3 i PA2 procesora. Za ich obsługę, po stronie układu STM32F411, odpowiada peryferial USART2.

Tworzymy w tym celu nowy projekt programu STM32CubeMX, wybieramy nasz mikrokontroler (dla przypomnienia, na płytce KA-NUCLEO, jest to układ STM32F411CEU6) i na pierwszej planszy konfiguratora – „Pinout", odnajdujemy wyżej wspomniane piny, klikamy na każdy z nich lewym przyciskiem myszy i z listy funkcji alternatywnych, wybieramy opcję „GPIO_Input", aby układ STM32 nie przeszkadzał nam w transmisji. Odszukujemy też dowolny inny pin dostępny, na naszej płytce rozwojowej, do dyspozycji użytkownika. Na płytce KA-NUCLEO, następny

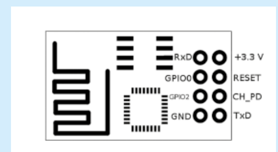


Rysunek 2. Ustawienia wyprowadzeń mikrokontrolera w programie STM32CubeMX



Rysunek 3.Ustawianie domyślnej wartości logicznej, na pinie wyjściowym „CH_PD"

pin (obok pinów RX/TX) – D2, przyłączony jest do wyprowadzenia mikrokontrolera, o oznaczeniu PA10. Ustawiamy go w tryb wyjścia – „GPIO_Output". Podłączony zostanie on do pinu „CH_PD" układu ESP8266 (**rysunek 2**).



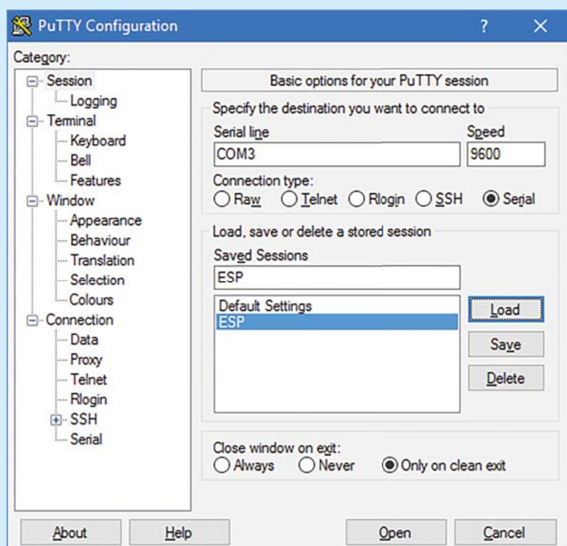
Rysunek 4. Wyprowadzenia modułu ESP8266 (źródło: sklep internetowy kamami.pl)

Ponieważ pin „CH_PD", przez cały czas pracy z układem, powinien być w stanie wysokim (3,3 V), ustawiamy mu ten stan jako domyślny – przechodzimy do planszy „Configuration", z zakładki „System" wybieramy pozycję GPIO, odszukujemy pin „CH_PD", zaznaczamy go i z listy „GPIO Output Level" wybieramy opcję „High" (**rysunek 3**).

Za pomocą kabli połączeniowych, łączymy płytkę KA-NUCLEO i układ ESP8266. Piny RX/TX interfejsu UART obu układów podłączamy na przemian, aby pin odbiorczy jednego z nich był połączony z pinem nadawczym drugiego. Pin ustawiony w tryb wyjścia (D2), łączymy z pinem „CH_PD" układu ESP. Nie zapominamy także o podłączeniu zasilania (3,3 V) i masy (**rysunek 4**).

Następnie generujemy nowy projekt, w sposób opisany w poprzednich częściach oraz importujemy go w środowisku System Workbench for STM32, kompilujemy źródła, wgrywamy program na mikrokontroler i uruchamiamy go. Teraz możemy już uruchomić, znany z poprzedniej części, program PuTTY i podobnie jak zostało to wcześniej opisane, wybrać w nim typ połączenia szeregowego – w polu „Connection Type" wybieramy opcję „Serial", wybrać port szeregowy, który system operacyjny przydzielił programatorowi ST-LINK – np. „COM3" oraz ustawić szybkość

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 97325, pass: 6yfwxr8q



Rysunek 5. Ustawianie parametrów połączenia szeregowego, w programie PuTTY

połączenia (**rysunek 5**). Układy ESP z nowszym oprogramowaniem pracują domyślnie z szybkością transmisji 115200, ze starszym – 9600. Test komunikacji z modułem ESP8266 pokazano na **rysunku 6**.

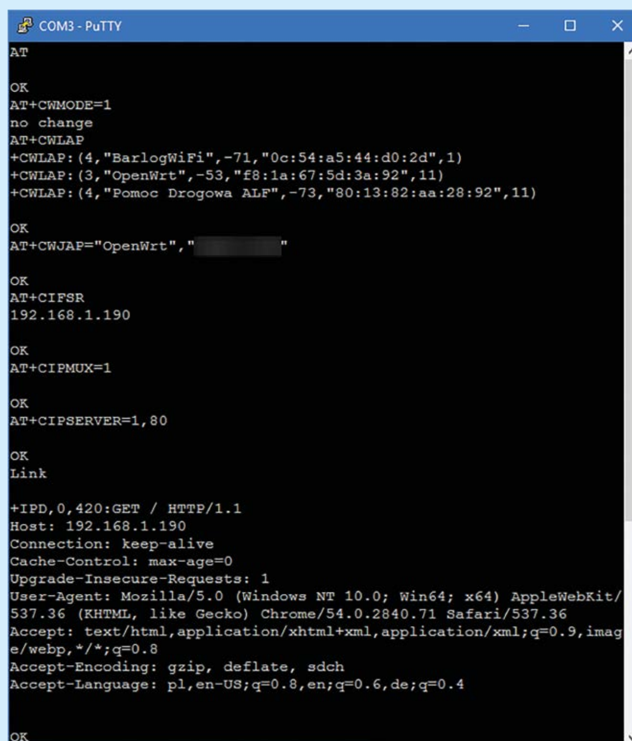
Układ ESP8266, podobnie jak większość innych urządzeń z którymi komunikujemy się poleceniami AT Hayesa, interpretuje tylko jeden format oznaczeń końca linii/polecenia. Są to niedrukowalne znaki ASCII – CR (powrót karetki/kursora na początek linii) oraz LF (przejście do nowej linii), co przypomina działanie maszyny do pisania. Niestety, nie jest to format wykorzystywany przez Windowsa. Aby więc układ ESP8266 prawidłowo odebrał od nas polecenie, musimy zakończyć je tymi dwoma znakami. Wykonujemy to, kombinacjami klawiszy Ctrl+M (CR) oraz Ctrl+J (LF).

Serwer HTTP i ustawianie koloru diody RGB przez stronę WWW

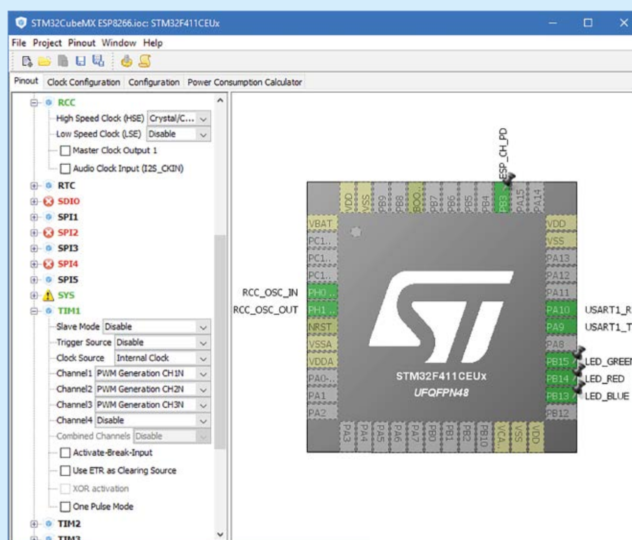
Przejdźmy teraz do właściwego projektu. Ponownie uruchamiamy generator konfiguracji STM32CubeMX, wybieramy mikrokontroler i przechodzimy do konfiguracji pinów (plansza „Pinout"). Tym razem nie skorzystamy już z tych samych wyprowadzeń interfejsu UART jak poprzednio, ponieważ chcemy umożliwić komunikację między mikrokontrolerem STM32 i układem ESP8266, a nie ESP8266 i programatorem ST-LINK. Na płytce rozwojowej KA-NUCLEO, możemy w tym celu skorzystać z interfejsu USART1 – jego wyprowadzenia nie są podłączone nigdzie indziej, poza samym mikrokontrolerem. Pinem nadawczym będzie pin D8, przyłączony do wyprowadzenia procesora o oznaczeniu PA9, a odbiorczym D2 – wyprowadzenie PA10. Ponieważ pin D2 zajęty został przez interfejs UART, musimy także przenieść pin połączony z wyprowadzeniem „CH_PD" układu ESP8266 – na płytce KA-NUCELO, skorzystamy teraz z pinu D3 – wyprowadzenie PB3. Podobnie jak poprzednio, ustawiamy na tym pinie stan domyślny wysoki (3,3 V).

Ustawiamy także wyprowadzenia procesora, do których na płytce rozwojowej KA-NUCLEO dołączono rezonator kwarcowy. W tym celu, na liście po lewej stronie odszukujemy pozycje „RCC" i po jej rozwinięciu, z pola „High Speed Clock (HSE)" wybieramy opcję „Crystal/Ceramic Resonator".

Odszukujemy też wyprowadzenia, do których jest przyłączona dioda RGB i ustawiamy im funkcje alternatywne (klikamy na nie LPM) „TIM1_CHxN", czyli wyjścia pierwszego licznika. Na płytce KA-NUCLEO są to piny PB13, PB14 i PB15, odpowiedzialne odpowiednio za kolory: niebieski, czerwony i zielony. Proponuję przy



Rysunek 6. Komunikacja z modułem ESP8266



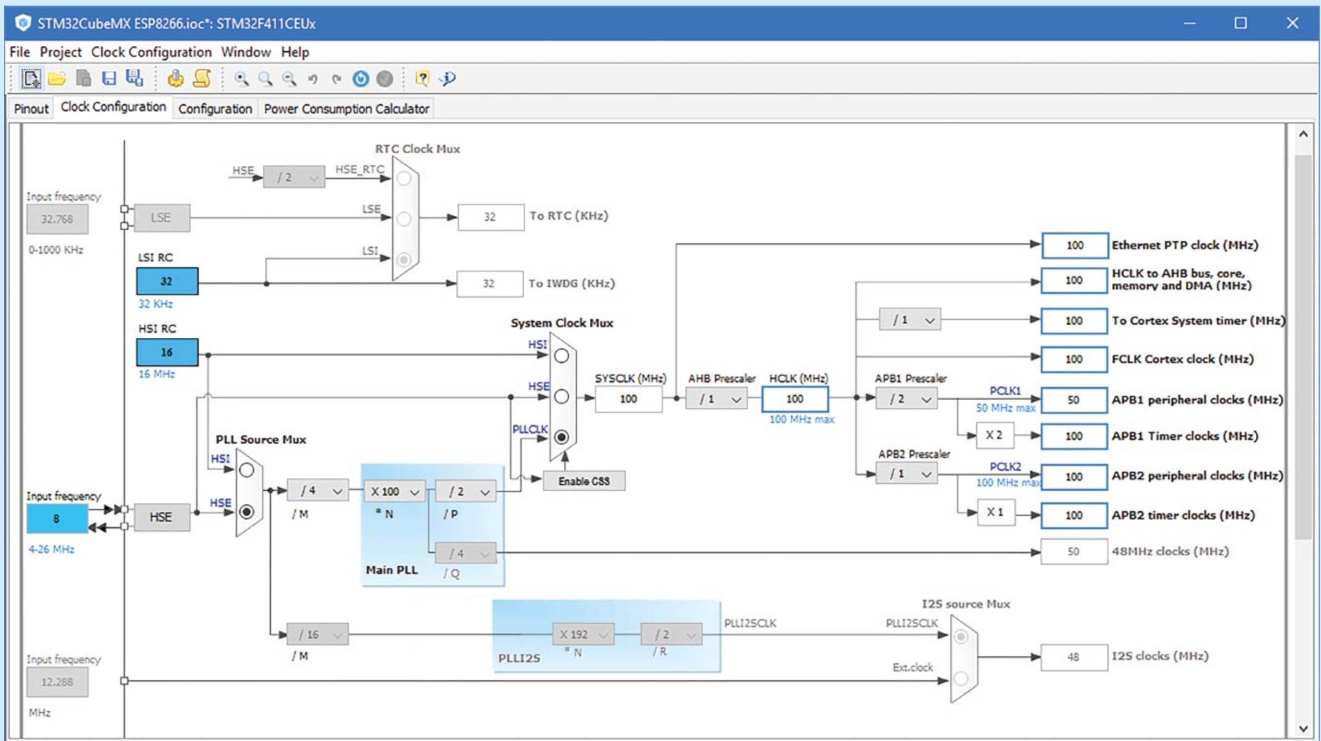
Rysunek 7. Konfiguracja wyprowadzeń układu STM32, w programie STM32CubeMX

okazji nadać im nazwy (PPM) „LED_BLUE", „LED_RED" i „LED_GREEN" – będą one wykorzystywane dalej w kodzie.

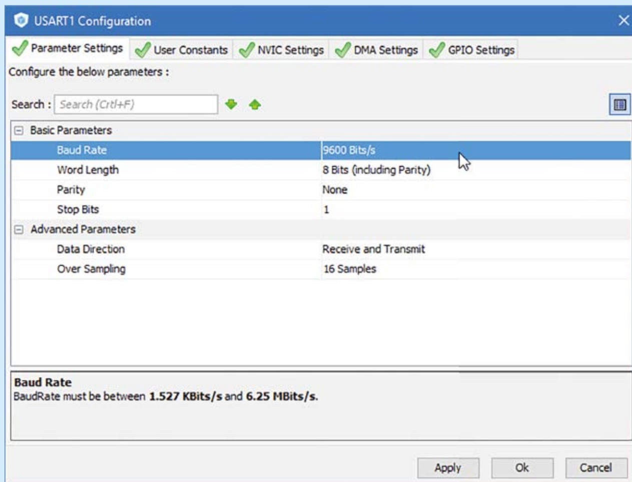
Musimy teraz jeszcze skonfigurować kanały licznika – na liście po lewej stronie odszukujemy pozycję „TIM1" i z pól „ChannelX", wybieramy opcje „PWM Generation CHxN", gdzie X/x to numer od 1 do 3 (numery kanałów). Ustawiamy również sygnał taktujący wchodzący na licznik – pole „Clock Source", na wartość „Internal Clock" (**rysunek 7**). Konfiguracja liczników i generatora sygnału PWM została dokładniej opisana w drugiej części niniejszego kursu.

Przechodzimy do zakładki „Clock Configuration" i podobnie jak w pierwszej części kursu, konfigurujemy sygnał taktujący. W polu „PLL Source Mux" wybieramy opcję „HSE", w polu „System Clock Mux", wartość „PLLCLK". Ustawiamy także częstotliwość naszego rezonatora kwarcowego – „Input frequency", na płytce KA-NUCLEO jest to wartość 8 MHz, oraz pożądaną częstotliwość

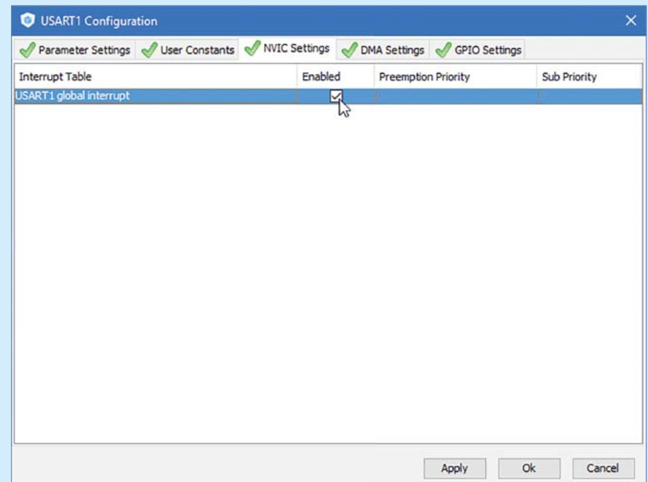
Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 97325, pass: 6yfwxr8q



Rysunek 8. Konfiguracja pętli PLL



Rysunek 9. Ustawianie szybkości połączenia szeregowego



Rysunek 10. Włączanie obsługi przerwania interfejsu UART

taktowania układu – „HCLK (MHz)", na wartość 100 MHz (dla wykorzystywanej przeze mnie płytki rozwojowej jest to wartość maksymalna). Konfigurację generatora PLL pokazano na **rysunku 8**.

Pozostaje nam jeszcze ustawić szybkość pracy interfejsu UART oraz parametry pracy licznika – generatora sygnału PWM, zgodnie z opisem z części 2 i 3. Przechodzimy do planszy „Configuration" i z pola „Connectivity" wybieramy pozycję „USART1". W nowootwartym oknie, w zakładce „Parameter Settings", w polu „Baud Rate" wpisujemy szybkość pracy interfejsu, z jaką nawiązana ma zostać komunikacja z układem ESP8266 – dla starszych wersji firmware będzie to wartość 9600, dla nowszych – 115200 (**rysunek 9**). W zakładce „NVIC Settings", włączamy jedyne przerwanie na liście (**rysunek 10**).

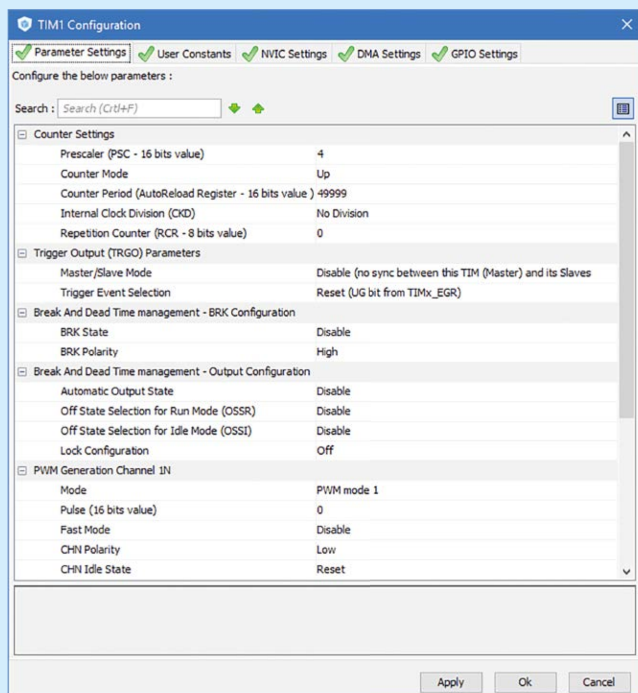
Następnie, po zapisaniu ustawień, z planszy „Configuration", głównego okna programu, wybieramy pozycję „Control" -> „TIM1" i ustawiamy parametry generowanego sygnału PWM – dzielnik „częstotliwości" wejściowej – „Prescaler", na 4, wartość do której zlicza licznik – „Counter Period", na 49999 oraz odwracamy wyjście każdego z kanałów „PWM Generation Channel xN": „CHN Polarity" – „Low" (**rysunek 11**). Dokładne znaczenie tych wartości oraz sposób obliczania pożądanej częstotliwości sygnału PWM opisany został w części drugiej tego kursu.

Teraz możemy już wygenerować projekt i zaimportować go w środowisku IDE System Workbench for STM32. Po zaimportowaniu projektu, potrzebujemy jeszcze ustawić go tak, aby na etapie linkowania/łączenia kodu, dodawana była do niego biblioteka math. Klikamy prawym przyciskiem myszy na nazwę projektu w panelu po lewej stronie, z menu kontekstowego wybieramy opcję „Properties", dalej rozwijamy opcje: „C/C++ Build", „Settings", „Tool Settings", „MCU GCC Linker", „Libraries", odznaczamy opcję „Use C math library (-lm)" i dodajemy tą bibliotekę – „m", ręcznie, do listy „Libraries" (**rysunek 12**).

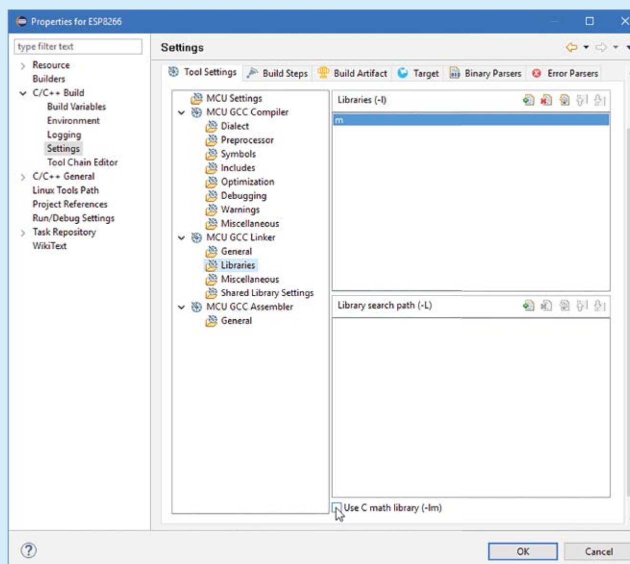
Otwieramy plik „main.c" i do oznaczonych w listingu sekcji USER CODE, dopisujemy znajdujący się w nich kod (**listing 1**). Następnie kompilujemy, wgrywamy i uruchamiamy na mikrokontrolerze program.

Po uruchomieniu, program wywołuje funkcję **esp_setup()**, która przesyła do układu ESP skrypt konfigurujący, za każdym razem

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 97325, pass: 6yfwxr8q



Rysunek 11. Ustawienia licznika – generatora sygnału PWM



Rysunek 12. Dodawanie biblioteki math do projektu w środowisku System Workbench for STM32

czekając na odpowiedź potwierdzającą wykonanie danej komendy („OK") lub informacje o błędzie („ERROR"). Polecenie przesyłane są przy pomocy funkcji **esp_send_cmd()**, ta z kolei wykorzystuje funkcje **uart_write_line()** i **uart_read_line()**. Po prawidłowym skonfigurowaniu chipu ESP8266, dioda LED RGB zapala się na zielono i uruchamiana jest obsługa przerwań, w przypadku niepowodzenia, dioda zapala się na czerwono.

W przerwaniu, obsługiwanym przez funkcję **HAL_UART_RxCpltCallback()**, wywoływaną po odebraniu każdego znaku, oczekujemy na odebranie od modułu ESP8266 ciągu +IPD,, oznaczającego odebranie danych, w uprzednio odebranym połączeniu

Listing 1. Fragment programu odpowiedzialny za współpracę z modułem Wi-Fi ESP8266

```
[...]

/* USER CODE BEGIN Includes */
#include „stdlib.h”
#include „string.h”
/* USER CODE END Includes */

/* USER CODE BEGIN PV */
UART_HandleTypeDef * esp_uart = &huart1;
volatile uint8_t esp_recv_char;
volatile uint8_t esp_char_counter = 0;
char esp_pattern[] = „+IPD,”;

volatile uint8_t esp_recv_flag = 0;
volatile char esp_recv_mux;
volatile char esp_recv_buffer[1024];
volatile uint16_t esp_recv_len;

char webpage[483], error[139];
/* USER CODE END PV */

[...]

/* USER CODE BEGIN 0 */
// Funkcja obliczająca korekcję gamma i ustawiająca jasność diody
void set_led_brightness(TIM_HandleTypeDef * timer, uint32_t channel, uint8_t brightness) {
    int32_t value = powf((double) brightness / 255.0, 2.2) * 49999;
    __HAL_TIM_SET_COMPARE(timer, channel, value);
}

// Funkcja ustawiająca kolor świecenia diody RGB
void set_color(uint8_t red, uint8_t green, uint8_t blue) {
    set_led_brightness(&htim1, TIM_CHANNEL_2, red);
    set_led_brightness(&htim1, TIM_CHANNEL_3, green);
    set_led_brightness(&htim1, TIM_CHANNEL_1, blue);
}

// Funkcja wysyłająca podany ciąg znaków przez interfejs UART
void uart_write_line(UART_HandleTypeDef * handler, char * text) {
    HAL_UART_Transmit(handler, text, strlen(text), 1000);
    HAL_UART_Transmit(handler, „\r\n”, 2, 100);
}

// Funkcja odbierająca linię tekstu przez interfejs UART
void uart_read_line(UART_HandleTypeDef * handler, char * buffer, uint16_t buffer_size) {
    HAL_StatusTypeDef status;
    char current_char;
    uint16_t char_counter = 0;
    while (char_counter < buffer_size - 1) {
        status = HAL_UART_Receive(handler, &current_char, 1, 1);
        if (status == HAL_OK) {
            if (current_char == ‚\r’ || current_char == ‚\n’)
                if (char_counter == 0) continue;
                else break;
                 (buffer + char_counter++) = current_char;
        }
    }
```


Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 97325, pass: 6yfwxr8q

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 97325, pass: 6yfwxr8q

Listing 1. cd.

```
    *(buffer + char_counter) = ‚\0’;
}

// Funkcja odczytująca pojedynczy znak odebrany przez UART
char uart_read_char(UART_HandleTypeDef * handler) {
    char buffer = ‚\0’;
    HAL_UART_Receive(handler, &buffer, 1, 1000);
    return buffer;
}

// Funkcja wysyłająca polecenie do modułu ESP8266
// i oczekująca na jego potwierdzenie
uint8_t esp_send_cmd(UART_HandleTypeDef * uart, char * command) {
    char response[30];
    response[0] = ‚\0’;
    uart_write_line(uart, command);
    __HAL_UART_FLUSH_DRREGISTER(&huart1);
    while (strcmp(response, „OK”) != 0
        && strcmp(response, „no change”) != 0
        && strcmp(response, „ERROR”) != 0)
        uart_read_line(uart, response, 30);
    if (strcmp(response, „ERROR”) == 0) return 0;
    else return 1;
}

// Funkcja wysyłająca dane przez nawiązane połączenie TCP
// i zamykająca to połączenie
void esp_send_data_and_close(UART_HandleTypeDef * uart, char mux_id, char * content) {
    char cmd[17];
    sprintf(cmd, „AT+CIPSEND=%c,%d”, mux_id, strlen(content));
    uart_write_line(uart, cmd);
    HAL_Delay(20);
    HAL_UART_Transmit(uart, content, strlen(content), 5000);
    HAL_Delay(100);
    sprintf(cmd, „AT+CIPCLOSE=%c”, esp_recv_mux);
    uart_write_line(esp_uart, cmd);
}

// Funkcja uruchamiająca obsługę przerwań
void esp_start_int_recv(UART_HandleTypeDef * uart) {
    __HAL_UART_FLUSH_DRREGISTER(uart);
    HAL_UART_Receive_IT(uart, &esp_recv_char, 1);
}

// Funkcja obsługująca przerwanie, wywoływana w momencie odebrania
// przez interfejs UART pojedynczego bajtu danych
void HAL_UART_RxCpltCallback(UART_HandleTypeDef * uart) {
    if (esp_recv_char == esp_pattern[esp_char_counter]) {
        esp_char_counter++;
        if (esp_char_counter == 5) {
        // Jeśli odbierzemy ciąg znaków „+IPD,”:
        // Odczytujemy numer połączenia do zmiennej esp_recv_mux
            esp_recv_mux = uart_read_char(uart);
            uart_read_char(uart);
       // Odczytujemy długość odebranych dancyh do esp_recv_len
            char length_str[5];
            char current_char = 0;
            uint8_t char_counter = 0;
            do {
                current_char = uart_read_char(uart);
                length_str[char_counter++] = current_char;
            } while (current_char != ‚:’);
            length_str[char_counter] = ‚\0’;
            uint16_t esp_recv_len = atoi(&length_str);
            // Odbieramy dane do bufora esp_recv_buffer
            HAL_UART_Receive(uart, esp_recv_buffer, esp_recv_len, 1000);
            esp_recv_flag = 1;
            return;
        }
    } else esp_char_counter = 0;
    // Ponowne uruchomienie przerwania
    HAL_UART_Receive_IT(uart, &esp_recv_char, 1);
}

// Funkcja przesyłająca do modułu ESP8266 polecenia konfigurujące
uint8_t esp_setup() {
    HAL_Delay(500); // Oczekujemy na uruchomienie modułu
    if (!esp_send_cmd(esp_uart, „AT+CWMODE=1”)) return 0;
    if (!esp_send_cmd(esp_uart, „AT+CWJAP=\”NAZWA_SIECI\”,\”KLUCZ_SIECIOWY\””)) return 0;
    if (!esp_send_cmd(esp_uart, „AT+CIPMUX=1”)) return 0;
    if (!esp_send_cmd(esp_uart, „AT+CIPSERVER=1,80”)) return 0;
    return 1;
}

// Funkcja wywoływana w momencie otrzymania danych przez połączenie TCP
void handle_request() {
    // Odczytujemy pierwsze 6 znaków odebranego żądania HTTP
    char request_begining[7];
    for (uint8_t i = 0; i < 6; i++)
        request_begining[i] = esp_recv_buffer[i];
    request_begining[6] = ‚\0’;
    // Jeśli przesłane zostały parametry - początek żądania:
    // „GET /?red=XXX&green=XXX&blue=XXX HTTP/1.1”
    if (strcmp(request_begining, „GET /?”) == 0) {
        // Odczytujemy pierwsze 3 liczby, jakie pojawią się w adresie
        // - są to wartości poszczególnych kolorów składowych (RGB)
        int nums[3] = { 0, 0, 0 };
        int num_counter = 0;
        uint8_t last_char_was_digit = 0;
        for (int i = 6; i < 41; i++)
            if (esp_recv_buffer[i] >= ‚0’ && esp_recv_buffer[i] <= ‚9’) {
                last_char_was_digit = 1;
                nums[num_counter] *= 10;
                nums[num_counter] += esp_recv_buffer[i] - ‚0’;
            } else if (last_char_was_digit == 1) {
                last_char_was_digit = 0;
                num_counter++;
                if (num_counter == 4) break;
            }
        // Ustawiamy kolor na diodzie RGB
```

Listing 1. cd.

```
        set_color(nums[0], nums[1], nums[2]);
        // Zwracamy stronę WWW z formularzem wyboru kolorów
        esp_send_data_and_close(esp_uart, esp_recv_mux, webpage);
    // Jeśli żądanie dotyczy strony głównej - «GET / HTTP/1.1»
    } else if (strcmp(request_begining, „GET / „) == 0)
        esp_send_data_and_close(esp_uart, esp_recv_mux, webpage);
    // W innym przypadku - zwracamy komunikat o błędzie (404 Not Found)
    else esp_send_data_and_close(esp_uart, esp_recv_mux, error);
    // Resetujemy flagę obsługi danych i wznawiamy odbiór w przerwaniach
    esp_recv_flag = 0;
    HAL_UART_Receive_IT(esp_uart, &esp_recv_char, 1);
}
/* USER CODE END 0 */

int main(void) {
     [...]
    /* USER CODE BEGIN 2 */
    HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_1);
    HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_2);
    HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_3);
    strcpy(webpage, „HTTP/1.1 200 OK\r\n”);
    strcat(webpage, „Content-Type: text/html\r\n”);
    strcat(webpage, „Content-Lenght: 398\r\n”);
    strcat(webpage, „Connection: close\r\n\r\n”);
    strcat(webpage, „<!DOCTYPE html>\r\n<html>\r\n<head>\r\n”);
    strcat(webpage, „<title>Dioda RGB</title>\r\n</head>\r\n”);
    strcat(webpage, „<body>\r\n<form method=\”get\”>\r\n”);
    strcat(webpage, „<p><b>Czerwony (0-255):</b> <input type=\”text\””);
    strcat(webpage, „ name=\”red\” value=\”0\” /></p>\r\n”);
    strcat(webpage, „<p><b>Zielony (0-255):</b> <input type=\”text\””);
    strcat(webpage, „ name=\”green\” value=\”0\” /></p>\r\n”);
    strcat(webpage, „<p><b>Niebieski (0-255):</b> <input type=\”text\””);
    strcat(webpage, „ name=\”blue\” value=\”0\” /></p>\r\n”);
    strcat(webpage, „<input type=\”submit\” value=\”Ustaw kolor\” />\r\n”);
    strcat(webpage, „</form>\r\n</body>\r\n</html>”);
    strcpy(error, „HTTP/1.1 404 Not Found\r\n”);
    strcat(error, „Content-Type: text/html\r\n”);
    strcat(error, „Content-Lenght: 48\r\n”);
    strcat(error, „Connection: close\r\n\r\n”);
    strcat(error, „<html><body><h1>404 Not Found</h1></body></html>”);
    if (esp_setup()) {
        set_color(0, 60, 0);
        esp_start_int_recv(esp_uart);
    } else set_color(60, 0, 0);
    /* USER CODE END 2 */
    /* USER CODE BEGIN WHILE */
    while (1) {
        /* USER CODE END WHILE */
        /* USER CODE BEGIN 3 */
        // Obsługujemy dane przychodzące przez połączenie TCP,
        // jeśli w przerwaniu ustawiono flagę esp_recv_flag
        if (esp_recv_flag == 1) handle_request();
    }
    /* USER CODE END 3 */
}
```

TCP. Po ciągu +IDP, przesyłany jest numer połączenia (od 0 do 4), długość odebranego ciągu danych oraz właściwe dane. Informacje te są odczytywane i zapisywane do zmiennych **esp_recv_mux** (numer połączenia), **esp_recv_buffer** (odebrany ciąg), **esp_recv_len** (długość bufora), następnie ustawiana jest flaga **esp_recv_flag**.

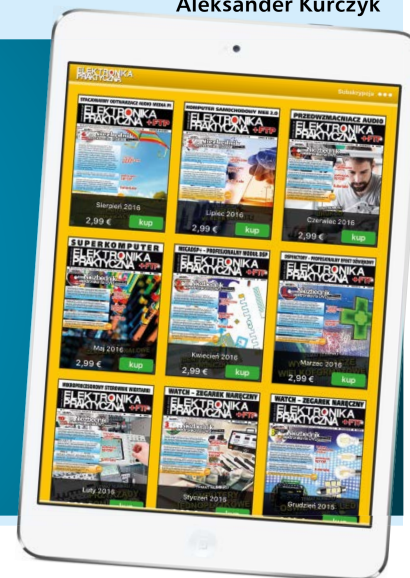
W pętli głównej programu, po ustawieniu w przerwaniu flagi, wywoływana jest funkcja **handle_request()** obsługująca przychodzące żądanie HTTP. Na podstawie początku żądania, podejmowana jest decyzja, jaką stronę WWW zwrócić, oraz czy zmienić kolor ustawiony na diodzie RGB. Ten odczytywany jest z adresu URL jako trzy następujące po sobie liczby 8-bitowe zawierająca informacje o jasności kolejnych trzech kolorów składowych – czerwonego, zielonego i niebieskiego. Dalej, przy pomocy funkcji **set_color()** i **set_led_brightness()** kolor ten jest ustawiany.

Przy pomocy modułu ESP możemy w podobny sposób przesyłać informacje odebrane z czujników – temperatury, czy wilgotności, do różnych serwisów internetowych lub własnego serwera, przez API HTTP – interfejsu bazującego na protokole HTTP, zaprojektowany z myślą o komunikacji między programami, lub dowolny inny protokół. Możemy też wykorzystać układ ESP do sterowania poruszającym się pojazdem, czy pobierać i drukować najnowsze tweety lub tytułu artykułów do przeczytania, odbieranych z RSS. Wiele serwisów i usług internetowych, udostępnia API, pozwalające wykonać te czynności, w bardzo łatwy sposób.

W kolejnej części kursu, napiszemy program sterujący adresowalnymi paskami diod LED RGB, bazujących na chipie WS2812B. Kod przedstawionych powyżej projektów, jest dostępny na serwerze FTP.

Aleksander Kurczyk

ELEKTRONIKA PRAKTYCZNA
na tabletach z systemami iOS i Android



Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **97325**, pass: **6yfwxr8q**