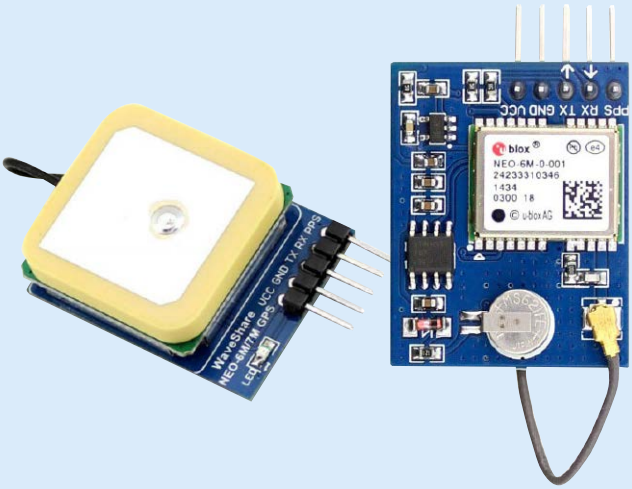
Programowanie STM32F4 (9)

W tej części cyklu poświęconego programowaniu układów STM32F4 zajmiemy się obsługą odbiornika GPS. Przedstawione zostanie działanie systemu nawigacji GPS oraz protokół NMEA-0183 – standard wymiany danych między komputerami i mikrokontrolerami a odbiornikami GPS. Wykorzystamy w tym celu tani moduł GPS – uBlox NEO-6M produkcji WaveShare, z wbudowaną anteną oraz zegarem czasu rzeczywistego. Dzięki standardowi NMEA, przedstawiony opis oraz biblioteka do komunikacji z układem GPS powinny być aktualne również dla wielu innych modułów dostępnych na rynku.

GPS jest systemem nawigacji satelitarnej zbudowanym i utrzymywanym przez Departament Obrony USA. Obejmuje on swoim zasięgiem całą kulę ziemską. Jego działanie polega na pomiarze czasu propagacji sygnału radiowego nadawanego przez satelity i odbieranego w dowolnym punkcie na Ziemi. Znając prędkość propagacji fal elektromagnetycznych, czas, w jakim sygnał został nadany oraz pozycje co najmniej czterech nadajników, można ustalić położenie odbiornika. Sygnał nadawany z satelitów GPS zawiera almanach oraz efemerydę – informacje o położeniu i torze lotu satelitów na orbicie oraz dokładny czas odmierzany przez zegary atomowe – cezowe i rubidowe umieszczone w satelitach. Satelity nadają swój sygnał w paśmie mikrofal, równocześnie na dwóch częstotliwościach nośnych – 1575,42 MHz oraz 1227,6 MHz, stosując modulację CDMA – nadając dla sygnału jedynki i zera kod rozpraszający, unikalny dla pojedynczej satelity i ortogonalny względem pozostałych kodów, aby nadajnik mógł rozróżnić sygnały nadawane przez inne satelity i jednocześnie nasłuchiwać na tylko jednej (lub dwóch) częstotliwościach.



Rysunek 1. Moduł WaveShare uBlox NEO-6M (źródło: strona WWW producenta)

Moduł GPS

Wykorzystany moduł GPS produkcji WaveShare (**rysunek 1**), bazuje na układzie odbiornika uBlox NEO-6M, ma wbudowaną antenę ceramiczną oraz pozwala na przyłączenie innej, zewnętrznej anteny, ze złączem u.FL. Ponadto moduł zawiera zegar czasu rzeczywistego, którego praca podtrzymywana jest baterią, na stałe zamontowaną na płytce. Możemy go zasilać napięciem z zakresu 2,7…5,0 V. Maksymalny pobór prądu to 80 mA. Komunikacja z układem odbywa się poprzez interfejs UART w standardzie NMEA-0183, z domyślnie ustawioną szybkością transmisji – 9600 b/s. Wartość tę możemy również zmienić na: 4800, 19200, 38400, 57600, 115200 lub 230400 b/s. Dane, w postaci serii różnych ramek NMEA, nadawane są przez moduł domyślnie co 1 sekundę, choć wartość tę można zmienić – na maksymalnie 5 ramek na sekundę.

Pojedyncza seria ramek standardu NMEA składa się z maksymalnie 480 znaków ASCII, a długość każdej ramki ograniczona jest do 82 znaków. Każda ramka przesyłana jest w osobnej linii, rozpoczyna się znakiem „$” i kończy znakiem przejścia do nowej linii („\r\n”, „\r” lub „\n”). Na początku każdej ramki znajduje się pięcioznakowy identyfikator jej typu. Te opisane w standardzie rozpoczynają się od prefiksu – dwóch znaków ASCII: „GP”, dodatkowe, dodane przez producenta układu i zawierające niestandardowe dane – zaczynają się od innych prefiksów, zależnych od producenta. Dane w ramkach przesyłane są w postaci ciągów znaków ASCII – liczby stała i zmiennoprzecinkowe, a także inne identyfikatory. Poszczególne pola danych rozdzielone są przecinkami. Na końcu każdej ramki, może, choć nie musi, znajdować się suma kontrolna. Rozpoczyna się ona znakiem gwiazdki („*”) i zawiera po sobie dwa znaki heksadecymalne.

Poniższa lista typów ramek zdefiniowana została w standardzie NMEA-0183:

- AAM – Waypoint Arrival Alarm,
- ALM – Almanac data,
- APA – Auto Pilot A sentence,
- APB – Auto Pilot B sentence,
- BOD – Bearing Origin to Destination,
- BWC – Bearing using Great Circle route,
- DTM – Datum being used,
- GGA – Fix information,
- GLL – Lat/Lon data,
- GSA – Overall Satellite data,
- GSV – Detailed Satellite data,
- MSK – Send control for a beacon receiver,
- MSS – Beacon receiver status information,
- RMA – Recommended Loran data,
- RMB – Recommended navigation data for gps,
- RMC – Recommended minimum data for gps,
- RTE – Route message,
- VTG – Vector track an Speed over the Ground,
- WCV – Waypoint closure velocity (Velocity Made Good),
- WPL – Waypoint information,
- XTC – Cross track error,
- XTE – Measured cross track error,
- ZTG – Zulu (UTC) time and time to go (to destination),
- ZDA – Date and Time.

Nie wszystkie z wymienionych typów ramek są wysyłane przez nasz moduł, a większość tych nadawanych zawiera redundantne

– dublujące się informacje. Do odczytania podstawowych informacji potrzebujemy interpretować jedynie część z nich. Na **listingu 1** znajduje się przykładowa seria ramek nadawana przez moduł uBlox NEO-6M. Ramka GPRMC zawiera kolejno następujące wartości:

Listing 1. Ramki danych w standardzie NMEA-0183
```
$GPRMC,125523.00,A,1234.56789,N,01234.56789,E,1.395,,170417,,,A*75
$GPVTG,,T,,M,1.395,N,2.583,K,A*21
$GPGGA,125523.00,1234.56789,N,01234.56789,E,1,06,1.75,292.2,M,42.0,M,,*5B
$GPGSA,A,3,24,19,32,17,06,02,,,,,,,4.63,1.75,4.29*02
$GPGSV,4,1,14,02,10,135,25,04,,,10,06,21,096,29,10,02,277,*4F
$GPGSV,4,2,14,11,,,08,13,,,13,14,,,20,15,,,08*79
$GPGSV,4,3,14,16,,,09,17,25,045,39,19,42,058,33,24,78,158,10*48
$GPGSV,4,4,14,25,27,259,14,32,26,311,10*72
$GPGLL,1234.56789,N,01234.56789,E,125523.00,A,A*66
```

- aktualny czas UTC (12:55:23),
- pole statusu (A – Aktywny, V – Nieaktywny),
- szerokość geograficzną (2 pola – „1234.56789,N”),
- długość geograficzną (2 pola – „01234.56789,E”),
- prędkość poruszania się, w węzłach, obliczoną na podstawie zmian pozycji,
- kąt kierunku, w jakim porusza się obiekt (w stopniach),
- aktualną datę (16.04.2017),
- odchylenie magnetyczne Ziemi,
- sumę kontrolną.

Ramka GPVTG zawiera:

- ścieżkę poruszania się w stopniach, na podstawie odczytów pozycji (2 pola – "-brak odczytu-,T”),
- ścieżkę poruszania się w stopniach, na podstawie współrzędnych magnetycznych (2 pola – "-brak odczytu-,M”),
- prędkość w węzłach (2 pola – „1.395,N”),
- prędkość w kilometrach na godzinę (2 pola – „2.583,K”),
- sumę kontrolną.

Ramka GPGGA zawiera:

- aktualny czas UTC,
- szerokość i długość geograficzną,
- jakość pomiaru (0 – brak odczytu, 1 – pozycja określona na podstawie GPS),
- liczbę śledzonych satelitów,
- HDOP – dokładność pozycji w poziomie,
- wysokość w metrach nad poziomem morza,
- czas od ostatniego uaktualnienia ze wspomagającej stacji naziemnej,
- numer ID wspomagającej stacji naziemnej,
- sumę kontrolną.

Ramka GPGSA zawiera:

- flagę i sposób ustalania pozycji (2 pola) – A/M – automatyczny/manualny, 1/2/3 – brak pozycji/pozycja 2D/pozycja 3D,
- numery satelitów użytych do ustalenia pozycji (12 pól)
- DOP – dokładność ustalonej pozycji,
- HDOP – dokładność pozycji w poziomie,
- VDOP – dokładność pozycji w pionie,
- sumę kontrolną.

Ramka GPGLL zawiera:

- szerokość i długość geograficzną (4 pola),
- aktualny czas,
- status (A/V),
- sumę kontrolną.

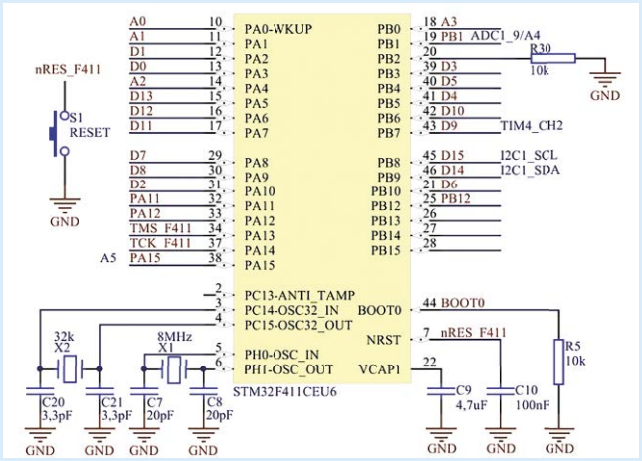
W przedstawionej sekwencji pojawiają się również ramki GPGSV. Pierwsze pole każdej z tych ramek zawiera ich liczbę w sekwencji, drugie to identyfikator kolejnej ramki (tutaj 4 ramki, z identyfikatorami od 1 do 4). Dalej w każdej ramce znajduje się liczba widocznych satelitów i sekwencje po 4 pola dotyczące poszczególnych satelitów, składające się z numeru satelity, wyniesienia satelity nad poziomem równika, w stopniach, azymutu satelity, również w stopniach, oraz poziomu odbieranego sygnału (SNR).

Tworzymy projekt

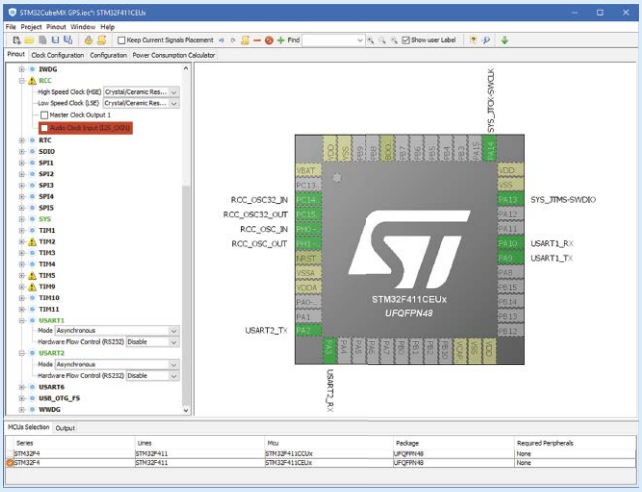
Spróbujemy teraz utworzyć projekt odbierający od modułu GPS dane dotyczące:

- aktualnej daty i godziny,
- współrzędnych geograficznych,
- wysokości nad poziomem morza,
- prędkości poruszania się, w węzłach oraz kilometrach na godzinę,
- liczby satelitów, z których odbierane są dane,
- oraz dokładności z jaką ustalone zostało położenie.

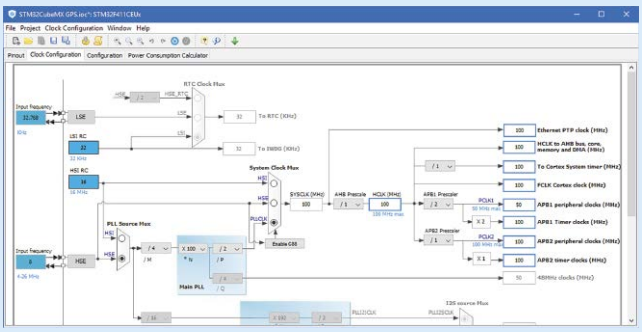
W przykładowym projekcie utworzymy uniwersalną bibliotekę współpracującą z modułami GPS różnych firm, odbierzemy za jej



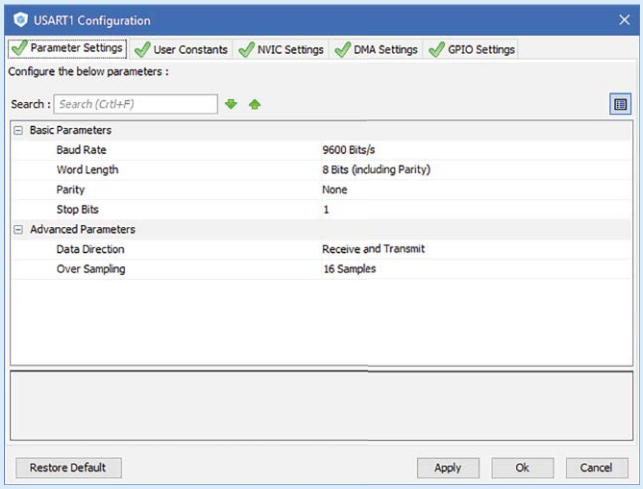
Rysunek 2. Schemat ideowy przyłączenia wyprowadzeń procesora do pinów kompatybilnych z Arduino, na płytce KA-NUCLEO-F411 (źródło: dokumentacja płytki KA-NUCLEO-F411)



Rysunek 3. Konfiguracja wyprowadzeń w programie STM32CubeMX



Rysunek 4. Konfiguracja sygnału zegarowego w programie STM32CubeMX



Rysunek 5. Konfiguracja interfejsów UART w programie STM-32CubeMX

pomocą ww. dane, a następnie wyślemy je poprzez drugi interfejs UART (przyłączony do wbudowanego programatora) do komputera, w czytelnej dla człowieka formie. Biblioteka ta może zostać wykorzystana w urządzeniu logującym swoje pozycje na karcie pamięci lub w pamięci EEPROM czy Flash, trackerze GPS, przesyłającym informacje o położeniu dane poprzez radio lub sieć komórkową na serwer, czy też w roli zegara czasu rzeczywistego, zsynchronizowanego z zegarem atomowym znajdującym się na orbicie.

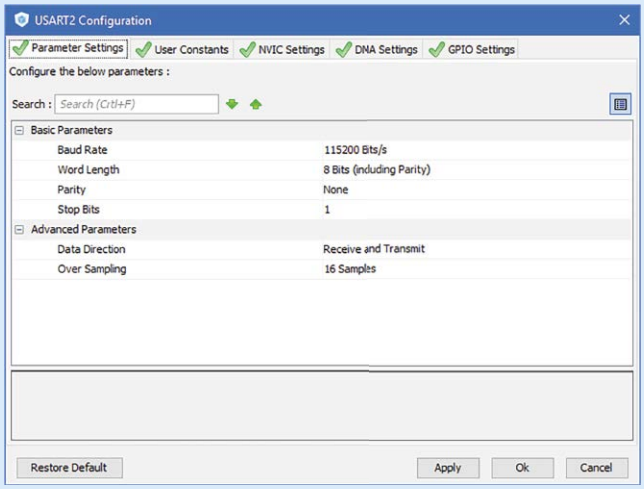
1. Uruchamiamy program STM32CubeMX i tworzymy w nim nowy projekt. W kreatorze wyboru mikrokontrolera wybieramy posiadany przez nas układ. Dla przypomnienia – na używanej podczas tworzenia kursu płytce rozwojowej Kamami KA-NUCLEO-F411 znajduje się układ STM32F411CEU6.
2. Na pierwszej planszy generatora konfiguracji STM32CubeMX definiujemy interfejsy i piny, z których będzie korzystał nasz program. Jeśli do posiadanego przez nas układu podłączony jest zewnętrzny oscylator kwarcowy, tak jak na płytce Kamami KA-NUCLEO-F411, z listy po lewej stronie okna rozwijamy zakładkę RCC i z pola „High Speed Clock (HSE)" wybieramy pozycję „Crystal/Ceramic Resonator".

Układ STM32F411CEU6 ma 3 interfejsy USART (USART1, USART2 i USART3), które możemy uruchomić na wyprowadzeniach procesora o identyfikatorach: peryferial USART1 – PA10 (pin RX)/PA9 (pin TX), PB7/PB6 lub PB3/PA15, USART2 – PA3/PA2 oraz USART6 – PA12/PA11. W omawianym przykładzie, na potrzeby komunikacji z modułem GPS, wybrany został peryferial USART1, na pinach PA10/PA9, odpowiadających wyprowadzeniom kompatybilnym z Arduino – D2/D8. Do tych wyprowadzeń przyłączamy nasz moduł GPS – do pinu D2 (odbiorczego po stronie mikrokontrolera) przyłączamy pin nadawczy (TX) modułu, do pinu nadawczego, ze strony mikrokontrolera (TX), możemy opcjonalnie podłączyć pin odbiorczy odbiornika GPS (w przedstawionym przykładzie nie potrzebujemy transmitować do modułu żadnych danych). Nie możemy też oczywiście zapomnieć o zasileniu modułu GPS – pin VCC (zasilanie układu) przyłączamy do pinu 3V3 płytki KA-NUCLEO-F411, a GND (masę) do pinu GND po stronie płytki. Schemat ideowy połączeń pokazano na **rysunku 2**.

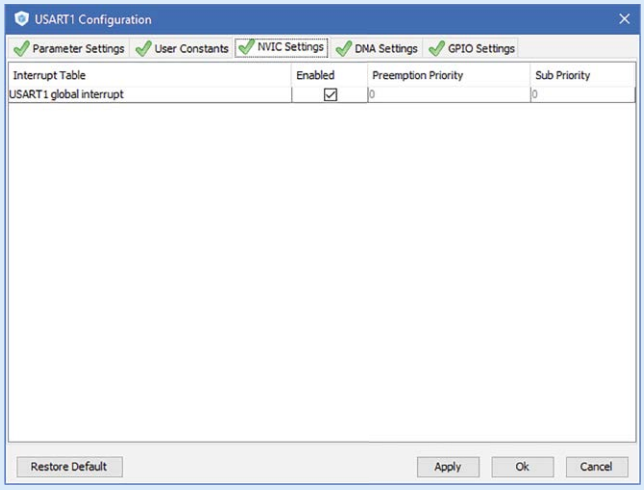
Po podłączeniu modułu GPS włączamy wybrany powyżej interfejs UART. W tym celu, z listy po lewej stronie, rozwijamy pozycję „USART2" i z pola Mode wybieramy pozycję „Asynchonous". Dokładny opis konfiguracji i działania interfejsu UART zawarty został w czwartej części tego kursu.

W projekcie skorzystamy także z interfejsu UART, przyłączonego na płytce KA-NUCLEO (poprzez wyprowadzenia układu – PA2 (TX) i PA3 (RX)) do programatora, który umożliwi nam przekazania odczytów do komputera. Interfejs ten uruchamiany w sposób identyczny jak opisany powyżej. Konfigurację wyprowadzeń w programie STM32CubeMX pokazano na **rysunku 3**.

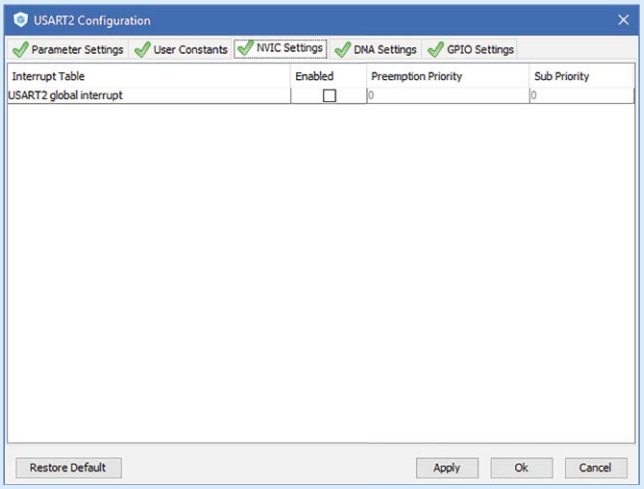
3. Po skonfigurowaniu wyprowadzeń przechodzimy do zakładki „Clock Configuration" i w identyczny sposób, jak w poprzednich częściach, konfigurujemy sygnał taktujący rozchodzący się po układzie. Jeśli do układu mikrokontrolera podłączony jest zewnętrzny oscylator kwarcowy, z pola „PLL Source MUX" wybieramy pozycję „HSE" i w polu „Input frequency" wpisujemy częstotliwość (w MHz) sygnału generowanego przez oscylator (na płytce Kamami KA-Nucleo jest to wartość 8 MHz). Dalej, w polu „System Clock MUX", wybieramy pozycję „PLLCLK". Następnie, w polu „HCLK (MHz)", wpisujemy częstotliwość taktowania całego układu po przejściu przez pętlę PLL. Wpisujemy tutaj maksymalną dozwoloną wartość – 100 MHz. Opis znaczenia tych parametrów zawarty został w pierwszej części kursu (**rysunek 4**).
4. Teraz możemy już przejść do trzeciej zakładki – „Configuration" i ustawić parametry pracy obu wykorzystywanych peryferiali. W tym celu klikamy kolejno na przyciski USART1 i po zakończeniu konfiguracji pierwszego peryferialu – USART2. W przypadku interfejsu przyłączonego



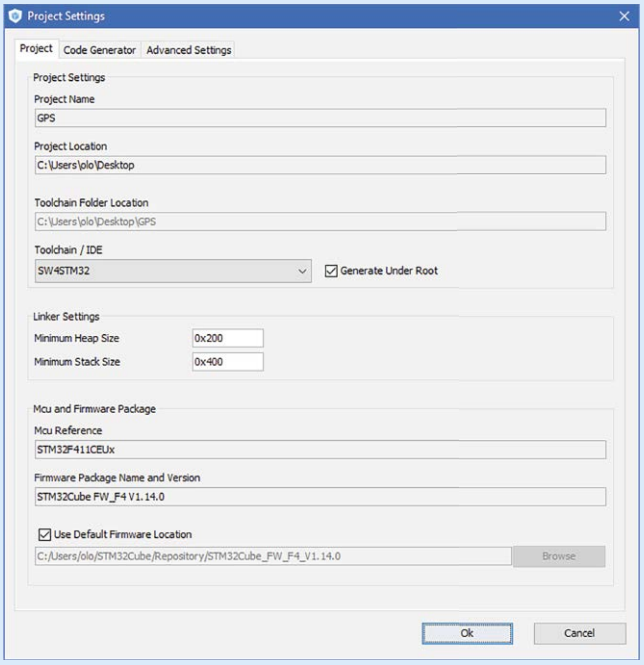
Rysunek 6. Konfiguracja interfejsów UART w programie STM-32CubeMX



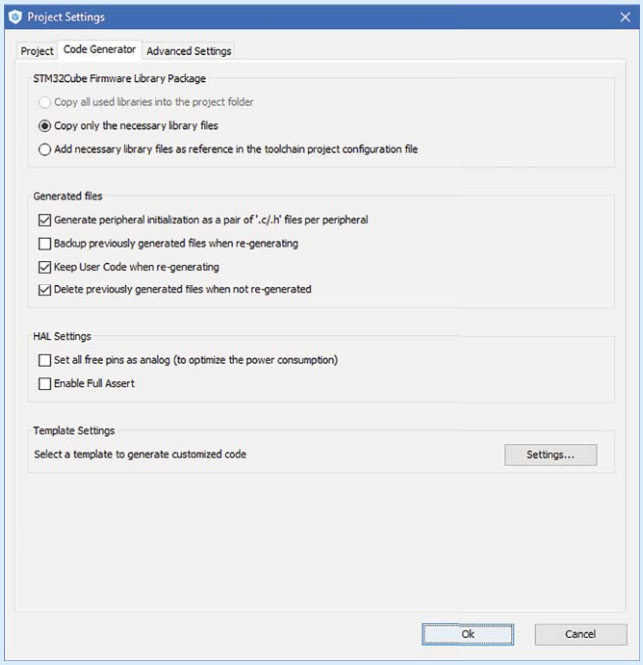
Rysunek 7. Włączanie obsługi przerwania w opcjach USART1 w programie STM32CubeMX



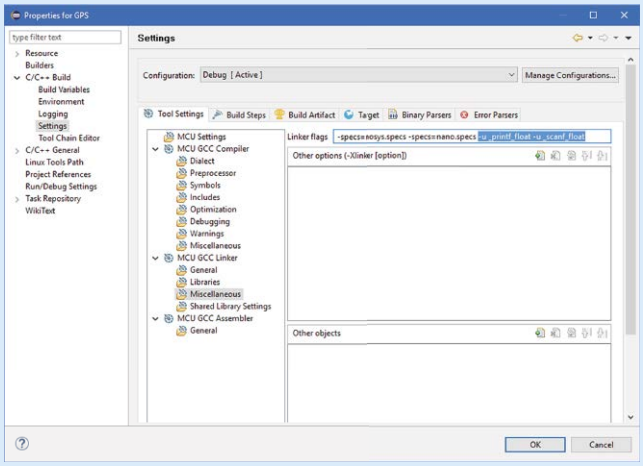
Rysunek 8. Wyłączanie obsługi przerwania w opcjach USART2 w programie STM32CubeMX



Rysunek 9. Eksport projektu z programu STM32CubeMX do programu System Workbench for STM32 – zakładka *Project*



Rysunek 10. Eksport projektu z programu STM32CubeMX do programu System Workbench for STM32 – zakładka *Code Generator*



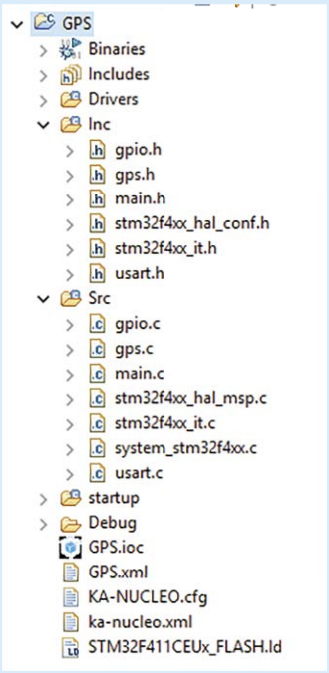
Rysunek 11. Zmiana parametrów wywołania linkera w opcjach projektu programu System Workbench for STM32

do modułu GSM, w pole Baud Rate wpisujemy wartość 9600 Bits/s, pozostałych wartości nie zmieniamy. Dla interfejsu przyłączonego do komputera szybkość transmisji powinna wynosić 115200 Bits/s. Konfiguracja obu interfejsów została przedstawiona na zrzutach ekranu na **rysunkach 5** i **6**.

5. Dla interfejsu przyłączonego do odbiornika GPS potrzebujemy jeszcze włączyć przerwanie wywoływane w momencie otrzymania od GPS nowych danych. W tym celu ponownie przechodzimy do konfiguracji interfejsu USART1, tym razem jednak, z paska na górze okna konfiguracyjnego, wybieramy zakładkę NVIC Settings i zaznaczamy tam jedyną opcję – „USART1 global interrupt" (**rysunek 8**, **rysunek 9**).
6. Nie pozostaje nam już nic innego, jak wygenerować projekt i zaimportować go w środowisku IDE. Będąc jeszcze w programie STM32CubeMX, klikamy ikonę zębatki znajdującą się na pasku narzędziowym. W nowym oknie wybieramy nazwę projektu (pole „Project Name"), ścieżkę dostępu do miejsca, w którym ma on zostać zapisany („Project Location"), z pola „Toolchain/IDE" wybieramy używane przez nas środowisko – „SW4STM32", w zakładce „Code Generation" zaznaczamy opcję „Generate peripheral initialization as a pair of ‚.c/.h' files per pepipheral" i klikamy przycisk „OK" (**rysunek 10**, **rysunek 11**).
7. Uruchamiamy program System Workbench for STM32, zamykamy planszę powitalną, w ramce „Project Explorer" klikamy prawym przyciskiem myszy i z menu kontekstowego wybieramy kolejno: „Import" –> „Existing Projects into Workspace", podajemy ścieżkę dostępu, wybieramy nowo utworzony projekt i zatwierdzamy import przyciskiem „Finish".
8. Aby możliwe było korzystanie z wartości zmiennoprzecinkowych, w funkcjach printf(), sprintf(), scanf() oraz sscanf(), konieczne jest dodanie parametrów „-u _printf_float" oraz „-u _scanf_float" do linii polecenia linkera. Robimy to, klikając prawym przyciskiem myszy na nazwę nowego projektu, z menu kontekstowego wybierając pozycję „Properties" oraz nawigując do „C/C++ Build" –> „Settings" –> „Miscellaneous" i dopisując do pola „Linker flags" wartość: „ -u _printf_float

-u _scanf_float" (**rysunek 12**).

9. Modyfikujemy kod źródłowy pliku „Src/main.c" zgodnie z **listingiem 2** oraz tworzymy dwa nowe pliki – „gps.h", w podfolderze „Inc" oraz „gps.c", w „Src", wypełniając je zawartością **listingów 3** i **4**. Aby to zrobić, w panelu Project Explorer, znajdującym się z lewej strony głównego okna środowiska, klikamy prawym przyciskiem myszy na nazwę podfolderu („Inc" i „Src"), a następnie, z menu kontekstowego wybieramy kolejno: „New", „File", podajemy jego nazwę i zatwierdzamy, klikając „Finish" (**rysunek 13**).



Rysunek 12. Drzewo plików i katalogów po dodaniu plików nowej biblioteki w programie System Workbench for STM32

10. Po dokonaniu modyfikacji zapisujemy zmiany w plikach, kompilujemy, wgrywamy i uruchamiamy program na mikrokontrolerze – klikamy ikony młotka i robaka, znajdujące się na pasku narzędziowym. Gdy program zostanie już uruchomiony, włączamy program PuTTY, wybieramy w jego ustawianiach typ połączenia – „Serial", jego szybkość – 115200 kbps, nazwę portu szeregowego – tę możemy sprawdzić w Menadżerze Urządzeń bądź dmesg-u i klikamy przycisk OK (**rysunek 14**, **rysunek 15**).

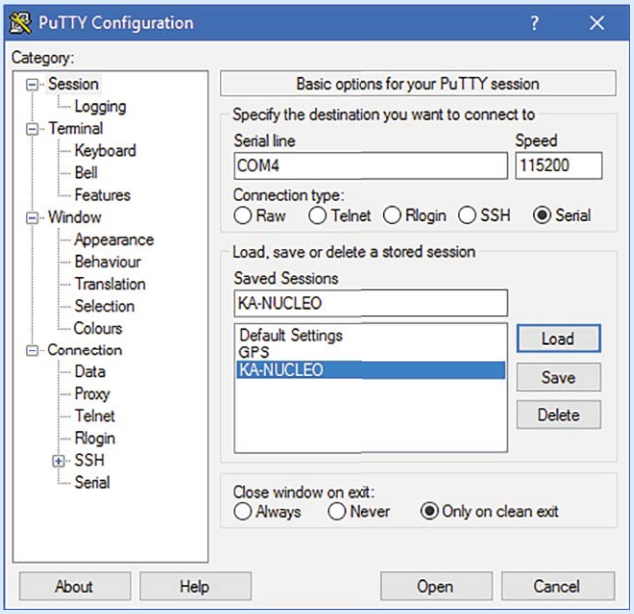
Do pliku main.c dodaliśmy zmienną „volatile uint8_t recv_char" oraz funkcję „void HAL_UART_RxCpltCallback(UART_HandleTypeDef * uart)", a na samym początku sekcji „USER CODE 2" wywołujemy funkcję „HAL_UART_Receive_IT(&huart1, &recv_char, 1)". Powoduje to włączenie obsługi przerwania interfejsu UART. Po odebraniu pojedynczego znaku od modułu GPS znak ten trafia do zmiennej „recv_char", a następnie wywoływana jest funkcja obsługi przerwania – „HAL_UART_RxCpltCallback()". W funkcji tej wywołujemy funkcję

Listing 2. Kod źródłowy pliku main.c

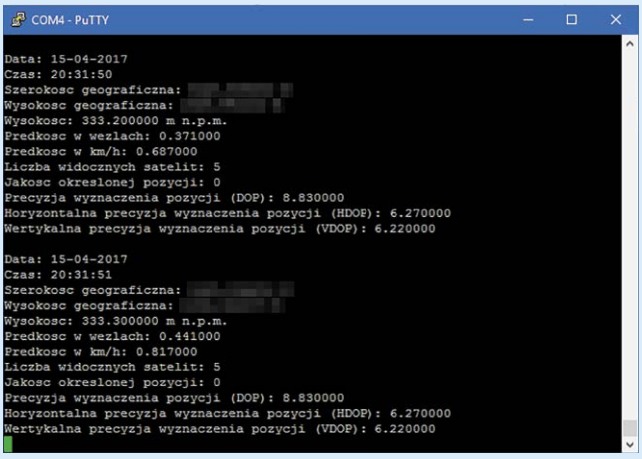
```
/* USER CODE BEGIN Includes */
#include „gps.h"
#include „string.h"
/* USER CODE END Includes */

/* USER CODE BEGIN 0 */
volatile struct gps_state gps_handle;
volatile uint8_t recv_char;

void HAL_UART_RxCpltCallback(UART_HandleTypeDef * uart) {
    if (uart == &huart1) {
        gps_recv_char(&gps_handle, recv_char);
        HAL_UART_Receive_IT(&huart1, &recv_char, 1);
    }
}
/* USER CODE END 0 */

int main(void)
{
    ...
    /* USER CODE BEGIN 2 */
    gps_handle = gps_init(&huart1);
    HAL_UART_Receive_IT(&huart1, &recv_char, 1);
    char output_buffer[100];
    for (uint8_t i = 0; i < 100; i++) output_buffer[i] = ,\0';
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1) {
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
        sprintf(output_buffer, „\r\n");
        HAL_UART_Transmit(&huart2, output_buffer, strlen(output_buffer), 100);
        sprintf(output_buffer, „Data: %02d-%02d-20%02d\r\n", gps_handle.date_day, gps_handle.date_mounth, gps_handle.date_year);
        HAL_UART_Transmit(&huart2, output_buffer, strlen(output_buffer), 100);
        sprintf(output_buffer, „Czas: %02d:%02d:%02d\r\n", gps_handle.time_hour, gps_handle.time_min, gps_handle.time_sec);
        HAL_UART_Transmit(&huart2, output_buffer, strlen(output_buffer), 100);
        sprintf(output_buffer, „Szerokosc geograficzna: %f %c\r\n", gps_handle.latitude, gps_handle.latitude_direction);
        HAL_UART_Transmit(&huart2, output_buffer, strlen(output_buffer), 100);
        sprintf(output_buffer, „Dlugosc geograficzna: %f %c\r\n", gps_handle.longitude, gps_handle.longitude_direction);
        HAL_UART_Transmit(&huart2, output_buffer, strlen(output_buffer), 100);
        sprintf(output_buffer, „Wysokosc: %f m n.p.m.\r\n", gps_handle.altitude);
        HAL_UART_Transmit(&huart2, output_buffer, strlen(output_buffer), 100);
        sprintf(output_buffer, „Predkosc w wezlach: %f\r\n", gps_handle.speed_knots);
        HAL_UART_Transmit(&huart2, output_buffer, strlen(output_buffer), 100);
        sprintf(output_buffer, „Predkosc w km/h: %f\r\n", gps_handle.speed_kilometers);
        HAL_UART_Transmit(&huart2, output_buffer, strlen(output_buffer), 100);
        sprintf(output_buffer, „Liczba widocznych satelit: %d\r\n", gps_handle.satelites_number);
        HAL_UART_Transmit(&huart2, output_buffer, strlen(output_buffer), 100);
        sprintf(output_buffer, „Jakosc określonej pozycji: %d\r\n", gps_handle.quality);
        HAL_UART_Transmit(&huart2, output_buffer, strlen(output_buffer), 100);
        sprintf(output_buffer, „Precyzja wyznaczenia pozycji (DOP): %f\r\n", gps_handle.dop);
        HAL_UART_Transmit(&huart2, output_buffer, strlen(output_buffer), 100);
        sprintf(output_buffer, „Horyzontalna precyzja wyznaczenia pozycji (HDOP): %f\r\n", gps_handle.hdop);
        HAL_UART_Transmit(&huart2, output_buffer, strlen(output_buffer), 100);
        sprintf(output_buffer, „Wertykalna precyzja wyznaczenia pozycji (VDOP): %f\r\n", gps_handle.vdop);
        HAL_UART_Transmit(&huart2, output_buffer, strlen(output_buffer), 100);
        HAL_Delay(1000);
    }
    /* USER CODE END 3 */
}
```

Listing 3. Kod źródłowy pliku gps.h

```
#ifndef gps_header
#define gps_header

#include „stm32f4xx_hal.h"
#include „usart.h"
#include „stdlib.h"
#include „string.h"

struct gps_state
{
    UART_HandleTypeDef * uart;
    uint8_t line_buffer[100];
    uint8_t writer_position;
    uint8_t reader_position;
    uint8_t field_buffer[30];
    uint8_t field_position;
    uint8_t date_day;
    uint8_t date_mounth;
    uint8_t date_year;
    uint8_t time_hour;
    uint8_t time_min;
    uint8_t time_sec;
    double latitude;
    char latitude_direction;
    double longitude;
    char longitude_direction;
    double altitude;
    double speed_knots;
    double speed_kilometers;
    uint8_t satelites_number;
    uint8_t quality;
    double dop;
    double hdop;
    double vdop;
};

struct gps_state gps_init(UART_HandleTypeDef * _uart);
void gps_recv_char(struct gps_state * state, uint8_t recv_char);
void gps_read_field(struct gps_state * state);
void gps_process_line(struct gps_state * state);
void gps_process_gprmc(struct gps_state * state);
void gps_process_gpvtg(struct gps_state * state);
void gps_process_gpgga(struct gps_state * state);
void gps_process_gpgsa(struct gps_state * state);
#endif
```



Rysunek 13. Konfiguracja programu PuTTY



Rysunek 14. Okno programu PuTTY z odebraną transmisją

„gps_recv_char(&gps_handle, recv_char);", zapisującą kolejne znaki do bufora biblioteki „gps.h/gps.c".

Zmienna „volatile struct gps_state gps_handle" przechowuje bufory i liczniki znaków, a także aktualne dane odebrane z GPS. Implementuje ona strukturę danych „struct gps_state" z pliku „gps.h". Jest tworzona i wypełniana zerowymi wartościami przez funkcję „gps_init()" z „gps.c", wywoływaną na początku sekcji „USER CODE 2".

Funkcja „gps_recv_char()" z biblioteki „gps.h/gps.c" oczekuje na odebranie znaku „$". Następnie, wszystkie kolejne znaki, aż do odebrania znaku nowej linii – „\n" lub „\r", zapisuje na kolejnych pozycjach bufora „line_buffer" i inkrementuje licznik „writer_position". Po odebraniu znaku „\n" lub „\r" jest wywoływana funkcja „gps_process_line()", która korzystając z funkcji pomocniczej „gps_read_field()" odczytującej pola do znaku przecinka lub końca ciągu („\0"), odczytuje typ ramki danych i wywołuje odpowiednią funkcję ją przetwarzającą – „gps_process_gprmc()", „gps_process_gpvtg()", „gps_process_gpgga()" lub „gps_process_gpgsa()". Funkcje te odczytują kolejne pola, parsują ich zawartość za pomocą funkcji „scanf()" i zapisują ją do zmiennych w strukturze „gps_state".

Aleksander Kurczyk

Listing 4. Kod źródłowy pliku gps.c

```
#include „gps.h"

struct gps_state gps_init(UART_HandleTypeDef * uart)
{
    struct gps_state state;
    state.uart = uart;
    for(uint8_t i=0; i<100; i++) state.line_buffer[i] = ,\0';
    state.writer_position = 0;
    state.reader_position = 0;
    for(uint8_t i=0; i<30; i++) state.field_buffer[i] = ,\0';
    state.field_position = 0;
    state.date_day = 0;
    state.date_mounth = 0;
    state.date_year = 0;
    state.time_hour = 0;
    state.time_min = 0;
    state.time_sec = 0;
    state.latitude = 0.0;
    state.latitude_direction = ,?';
    state.longitude = 0.0;
    state.longitude_direction = ,?';
    state.altitude = 0.0;
    state.speed_knots = 0.0;
    state.speed_kilometers = 0.0;
    state.satelites_number = 0;
    state.quality = ,?';
    state.dop = 0.0;
    state.hdop = 0.0;
    state.vdop = 0.0;
    return state;
}

void gps_recv_char(struct gps_state * state, uint8_t recv_char)
{
    if (state->writer_position == 0 && recv_char == ,$')
    {
        state->writer_position++;
    } else if (state->writer_position >= 1 && state->writer_position < 99)
    {
        if (recv_char == ,\r' || recv_char == ,\n')
        {
            state->line_buffer[state->writer_position - 1] = ,\0';
            state->writer_position = 0;
            gps_process_line(state);
        } else
        {
            state->line_buffer[state->writer_position - 1] = recv_char;
            state->writer_position++;
        }
    } else
    {
        state->writer_position = 0;
    }
}

void gps_read_field(struct gps_state * state)
```

Listing 4. cd.

```
{
    state->field_position = 0;
    while(state->line_buffer[state->reader_position] != ,,' && state->line_buffer[state->reader_position] != ,\0'
          && state->field_position < 29)
    {
        state->field_buffer[state->field_position] = state->line_buffer[state->reader_position];
        state->reader_position++;
        state->field_position++;
          }
    state->field_buffer[state->field_position] = ,\0';
    state->reader_position++;
}

void gps_process_line(struct gps_state * state)
{
    state->reader_position = 0;
    gps_read_field(state);
    if(strcmp(state->field_buffer, „GPRMC") == 0) gps_process_gprmc(state);
    else if(strcmp(state->field_buffer, „GPVTG") == 0) gps_process_gpvtg(state);
    else if(s trcmp(state->field_buffer, „GPGGA") == 0) gps_process_gpgga(state);
    else if(strcmp(state->field_buffer, „GPGSA") == 0) gps_process_gpgsa(state);
}

void gps_process_gprmc(struct gps_state * state)
{
    //$GPRMC,212846.00,A,5025.81511,N,01639.92090,E,0.196,,140417,,,A*73
    gps_read_field(state);
    if(strlen(state->field_buffer) > 0)
    {
        uint32_t tmp;
        sscanf(state->field_buffer, „%d", &tmp);
        state->time_sec = tmp % 100;
        state->time_min = (tmp / 100) % 100;
        state->time_hour = (tmp / 10000) % 100;
    }
    gps_read_field(state);
    gps_read_field(state);
    if(strlen(state->field_buffer) > 0) sscanf(state->field_buffer, „%lf", &(state->latitude));
    gps_read_field(state);
    if(strlen(state->field_buffer) > 0) sscanf(state->field_buffer, „%c", &(state->latitude_direction));
    gps_read_field(state);
    if(strlen(state->field_buffer) > 0) sscanf(state->field_buffer, „%lf", &(state->longitude));
    gps_read_field(state);
    if(strlen(state->field_buffer) > 0) sscanf(state->field_buffer, „%c", &(state->longitude_direction));
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    if(strlen(state->field_buffer) > 0) {
        uint32_t tmp;
        sscanf(state->field_buffer, „%d", &tmp);
        state->date_year = tmp % 100;
        state->date_mounth = (tmp / 100) % 100;
        state->date_day = (tmp / 10000) % 100;
    }
}

void gps_process_gpvtg(struct gps_state * state)
{
    //$GPVTG,,T,,M,0.196,N,0.363,K,A*2B
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    if(strlen(state->field_buffer) > 0) sscanf(state->field_buffer, „%lf", &(state->speed_knots));
    gps_read_field(state);
    gps_read_field(state);
    if(strlen(state->field_buffer) > 0) sscanf(state->field_buffer, „%lf", &(state->speed_kilometers));
}

void gps_process_gpgga(struct gps_state * state)
{
    //$GPGGA,212846.00,5025.81511,N,01639.92090,E,1,04,4.72,281.1,M,42.0,M,,*5F
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    if(strlen(state->field_buffer) > 0) sscanf(state->field_buffer, „%d", &(state->quality));
    gps_read_field(state);
    if(strlen(state->field_buffer) > 0) sscanf(state->field_buffer, „%d", &(state->satelites_number));
    gps_read_field(state);
    gps_read_field(state);
    if(strlen(state->field_buffer) > 0) sscanf(state->field_buffer, „%lf", &(state->altitude));
}

void gps_process_gpgsa(struct gps_state * state)
{
    //$GPGSA,A,3,10,18,21,15,,,,,,,,,6.79,4.72,4.89*01
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    gps_read_field(state);
    if(strlen(state->field_buffer) > 0) sscanf(state->field_buffer, „%lf", &(state->dop));
    gps_read_field(state);
    if(strlen(state->field_buffer) > 0) sscanf(state->field_buffer, „%lf", &(state->hdop));
    gps_read_field(state);
    if(strlen(state->field_buffer) > 0) sscanf(state->field_buffer, „%lf", &(state->vdop));
}
```

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 47858, pass: 9seghusa