Programowanie STM32F4 (8)

W poprzedniej części omówiono odczyt danych z czujników analogowych z wykorzystaniem wbudowanego, w układ mikrokontrolera przetwornika A/C. Dziś na warsztat bierzemy obsługę czujników cyfrowych i interfejs I²C. Posłużymy się przykładem cyfrowego termometru i barometru BMP180 firmy Bosch.

Interfejs I²C jest kolejnym omawianym w tym kursie interfejsem komunikacyjnym, typowym dla mikrokontrolerów. Służy on głównie do odbioru danych z czujników i komunikacji z urządzeniami niewymagającymi szybkiej transmisji danych. Tym, co wyróżnia go spośród innych interfejsów jest praca w trybie magistrali. Do pojedynczej szyny danych interfejsu I²C możemy przyłączyć wiele urządzeń, a dodanie kolejnych nie powoduje konieczności dodania kolejnych linii „Slave Select" (**rysunek 1**). Połączenie I²C składa się z linii SDA, którą w obu kierunkach przesyłane są dane (od urządzenia głównego do urządzeń podrzędnych i od podrzędnych do głównego) oraz linii SCL – przebiegu zegarowego generowanego przez urządzenie nadrzędne (master), do którego muszą się dostosować wszystkie urządzenia podrzędne (slave). Inny jest też sposób, w jaki korzysta się z interfejsu. Specyfikacja protokołu I²C nie ogranicza się do dostarczenia kanału, którym przesyłane są bajty danych między urządzeniami. Zamiast przesyłać do urządzeń specyficzne dla nich komunikaty, interfejs I²C pozwala w standardowy sposób zapisywać i odczytywać dane pod określonymi adresami w pamięci urządzeń – w rejestrach lub pamięci EEPROM.

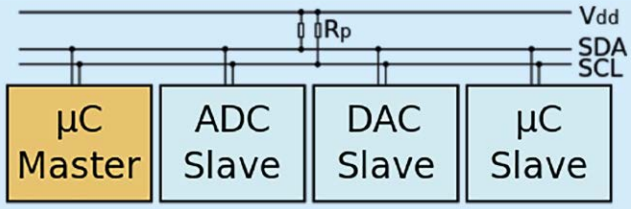
Interfejs I²C bywa również często używany do przyłączania do układu zewnętrznej pamięci EEPROM. Odczytywaną wartością może być wartość mierzona przez czujnik, a zapisywaną parametr pracy urządzenia, polecenie wykonania pomiaru, przełączenia zwrotnicy, czy blokady drzwi w tramwaju.

Interfejs I²C opracowano w latach osiemdziesiątych w firmie Philips. Początkowo był on używany głównie w sprzęcie RTV tejże firmy. Obecnie jest powszechnie stosowany w wielu układach różnych producentów. Każdy układ ma swój własny identyfikator – najczęściej 7-bitowy, choć nowsze wersje protokołu przewidują również adresację 10- i 16-bitową. Identyfikatory te są przydzielane przez firmę NXP Semiconductors. Pierwsza wersja protokołu I²C pozwalała na transmisję danych z szybkością 10 kbps (low speed) lub 100 kbps (standard speed). Obecnie standard dopuszcza również inne szybkości – fast mode: 400 kbps, fast mode plus: 1 Mbps oraz high speed mode: 3,4 Mbps.

Czujnik BMP180

Czujnik Bosch BMP180 (**rysunek 2**) umożliwia pomiar temperatury i ciśnienia atmosferycznego. Umożliwia również obliczenie wysokości nad poziomem morza, na jakiej znajduje się czujnik. Jest on dostępny w wielu sklepach z elektroniką w postaci gotowych płytek z wyprowadzonymi pinami (**rysunek 3**) wprost do przyłączenia do mikrokontrolera. Czujnik pracuje w zakresie napięcia zasilającego 1,62…3,6 V, więc może współpracować bez układów dopasowujących poziomy logiczne z płytką rozwojową Kamami KA-NUCLEO-F411.

Temperatura jest mierzona z dokładnością do 0,1 stopnia w zakresie od -40…80°C, a ciśnienie z dokładnością do pojedynczego Pascala (z dopuszczalną odchyłką od 0,06 hPa do 0,02 hPa,



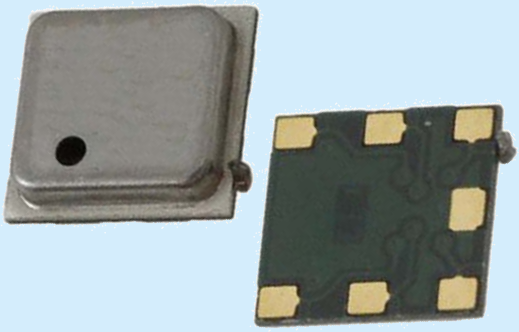
Rysunek 1. Sposób dołączania kolejnych urządzeń do magistrali (źródło: Wikipedia)

zależną od wybranego trybu pomiaru) w zakresie 300…1100 hPa, co przekłada się na pomiar wysokości n.p.m. w zakresie od -500 do +9000 metrów, z dokładnością do 0,5 lub 0,17 metra (w zależności od trybu pomiaru).

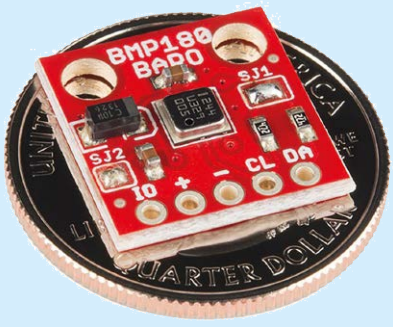
Czujnika nie musimy kalibrować, natomiast uzyskane wyniki pomiarów należy skompensować – podstawić do wzorów, wraz z wartościami kompensacji zapisanymi w pamięci EEPROM układu.

Komunikacja z czujnikiem

Czujnik BMP180 ma 7-bitowy adres/identyfikator równy „1110111". Razem z adresem w ramce I²C na pozycji 8 bitu w oktecie jest przesyłana flaga odczytu/zapisu. Zwyczajowo podając adres

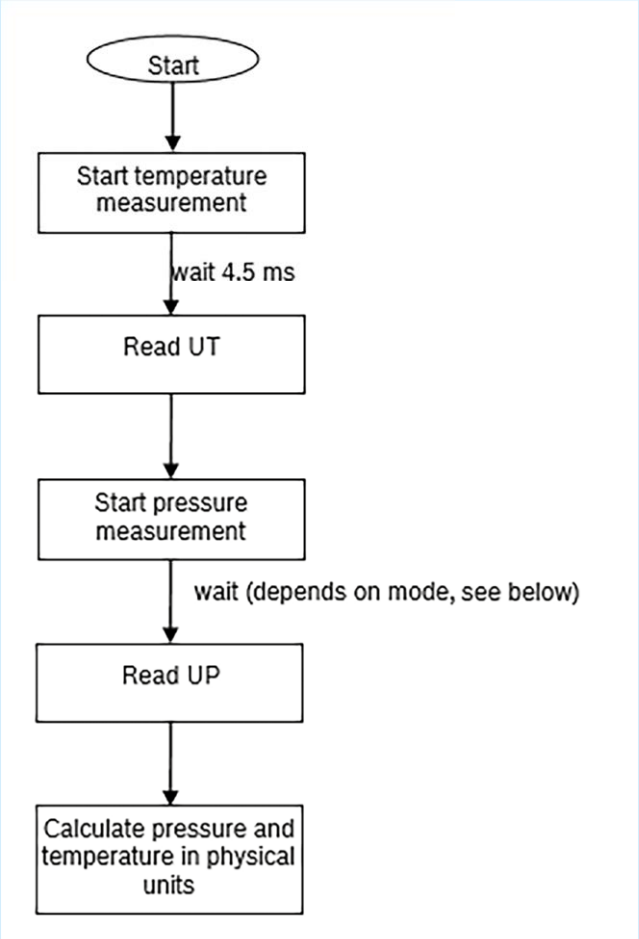


Rysunek 2. Chip czujnika (źródło: sklep internetowy Kamami.pl)



Rysunek 3. Gotowa płytka z układem czujnika BMP180 (źródło: sklep internetowy Kamami.pl)

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 33372, pass: 6nwd5fk4



Rysunek 4. Algorytm programu obsługi układu Bosch BMP180

urządzenia w postaci liczby heksadecymalnej, podaje się jego „adres odbiorczy" i „nadawczy", które dla tego czujnika wynoszą odpowiednio: 0xEF (odczyt) i 0xEE (zapis). Czujnik może przesyłać dane z prędkością do 3,4 Mbps.

Układ BMP180 ma wiele rejestrów do zapisu i odczytu. W 16-bitowych rejestrach o identyfikatorach AC1 (adres 0xAA), AC2 (0xAC), AC3 (0xAE), AC4 (0xB0), AC5 (0xB2), AC6 (0xB4), B1 (0xB6), B2 (0xB8), MB (0xBA), MC (0xBC) oraz MD (0xBE) są umieszczone fabrycznie współczynniki kompensacji używane do przekształcenia odczytów wbudowanego układu A/C w rzeczywiste wartości w stopniach Celsjusza lub Paskalach. Po wykonaniu pomiaru w komórkach pamięci o adresach 0xF6, 0xF7 i 0xF8 są przechowywane nieskompensowane wartości odczytu temperatury (UT) lub ciśnienia atmosferycznego (UP).

8-bitowy rejestr pod adresem 0xF4 przyjmuje polecenia do wykonania – start pomiaru temperatury lub ciśnienia z ustaloną dokładnością.

Liczby zapisane w rejestrach AC1, AC2, AC3, B1, B2, MB, MC, MD to wartości ze znakiem (signed short/int16_t). Liczby z rejestrów AC4, AC5, AC6 to wartości bez znaku (unsigned/uint16_t).

Algorytm wykonywania pomiarów, zamieszczony w karcie katalogowej czujnika przewiduje następujące akcje:

- Pobranie zawartości rejestrów AC1, AC2, AC3, AC4, AC5, AC6, B1, B2, MB, MC, MD i przechowanie ich w zmiennych.
- Zapisanie do rejestru 0xF4, wartości 0x2E, powodujące rozpoczęcie pomiaru temperatury.
- Odczekanie czasu 4,5 milisekundy.
- Pobranie i przechowanie w 16-bitowej zmiennej UT zawartości pamięci spod adresów 0xF6 oraz 0xF7.
- Zapisanie do rejestru 0xF4 polecenia wykonania pomiaru ciśnienia, z wybraną dokładnością – 0x34 dla najmniejszej dokładności, 0x74 dla standardowej, 0xB4 dla dużej lub 0xF4 dla największej.
- Odczekanie czasu 4,5 ms, 7,5 ms, 13,5 ms lub 25,5 ms (w zależności od wybranej dokładności pomiaru).
- Pobranie zawartości 24-bitów pamięci począwszy od adresu 0xF6 (0xF7, 0xF8) do zmiennej UP.
- Obliczenie, zgodnie ze wzorami podanymi w fragmencie karty katalogowej zamieszczonym obok, skompensowanej wartości temperatury i ciśnienia.
- Wynikiem obliczeń będzie temperatura w 0,1°C oraz ciśnienie w Paskalach.
- Nie jest możliwe wykonanie pomiaru tylko ciśnienia atmosferycznego, polecenia należy wykonywać w ściśle określonej, przedstawionej powyżej, kolejności.
- Obliczenie wysokości n.p.m. możliwe jest poprzez podstawienie zmierzonej wartości ciśnienia do wzoru

$$wysokość = 44330 \cdot \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5,255}}\right)$$

gdzie:
p to zmierzona wartość ciśnienia,
p_0 wartość ciśnienia na poziomie morza (np. 1013,25 hPa).

Algorytm postępowania przy odczycie danych z czujnika BMP180 zamieszczono na stronie 15 karty katalogowej dostępnej w materiałach dodatkowych do tego artykułu.

Tworzymy projekt

Utworzymy teraz projekt realizujący opisaną powyżej procedurę odbioru danych z czujników i przesyłający te dane do komputera przez interfejs UART, programator ST-LINK i kabel USB.

Uruchamiamy program STM32CubeMX i tworzymy w nim nowy projekt. W kreatorze wyboru mikrokontrolera wybieramy posiadany przez nas układ. Dla przypomnienia – na płytce Kamami KA-NUCLEO-F411 używanej w tym kursie zamontowano układ STM32F411CEU6.

Na pierwszej planszy generatora konfiguracji STM32CubeMX definiujemy interfejsy i wyprowadzenia, z których będziemy korzystali w programie. Jeśli do posiadanego przez nas układu jest dołączony zewnętrzny oscylator kwarcowy (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".

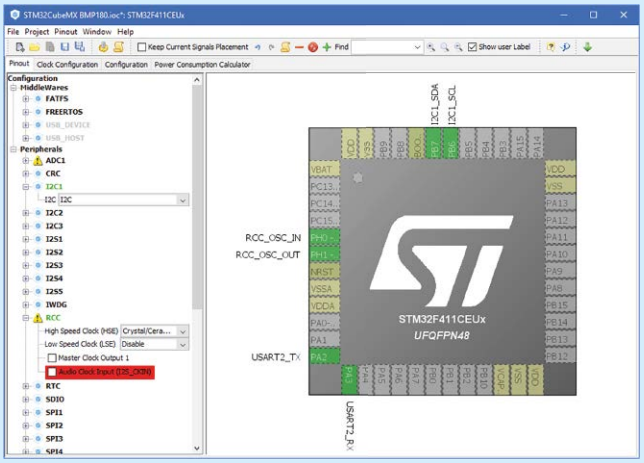
Układ STM32F411CEU6 ma trzy interfejsy I²C, które możemy uruchomić na wyprowadzeniach procesora:

1. Moduł I2C1 – PB7 (SDA) i PB6 (SCL) lub PB9 (SDA) i PB8 (SCL).
2. Moduł I2C2 – PB3 (SDA) i PB10 (SCL) lub PB9 (SDA) i PB10 (SCL).
3. Moduł I2C3 – PB4 (SDA) i PA8 (SCL) lub PB8 (SDA) i PA8 (SCL).

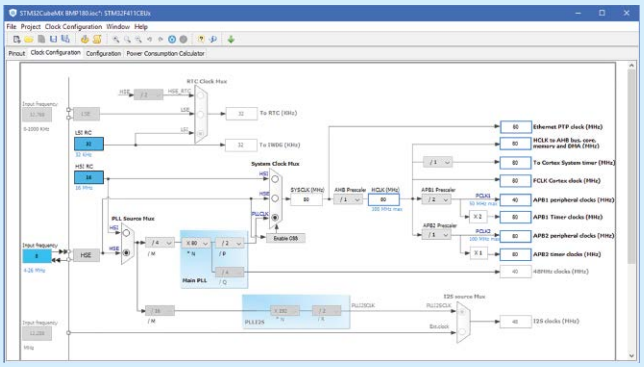
W omawianym przykładzie wybrano moduł I2C1 oraz piny PB7/PB6 odpowiadające wyprowadzeniom płytki o oznaczeniach D9 i D10. Do tych wyprowadzeń dołączamy czujnik BMP180. Nie możemy też zapomnieć o rezystorach podciągających 4,7 kΩ (rys. 1), a także o połączeniu mas układów oraz zasileniu czujnika napięciem 3,3 V.

Po wybraniu modułu I²C oraz wyprowadzeń procesora, rozwijamy na liście po lewej stronie okna programu STM32CubeMX zakładkę odpowiadającą wybranemu modułowi i w polu „I2C" wybieramy pozycję „I2C" – standardowy tryb pracy.

W projekcie skorzystamy także z interfejsu UART, a dokładniej modułu UART2 przyłączonego na płytce KA-NUCLEO poprzez wyprowadzenia układu PA2 (TX) i PA3 (RX) do programatora, który umożliwi nam przekazania odczytów do komputera. Dokładny



Rysunek 5. Konfiguracja wyprowadzeń w programie STM32CubeMX



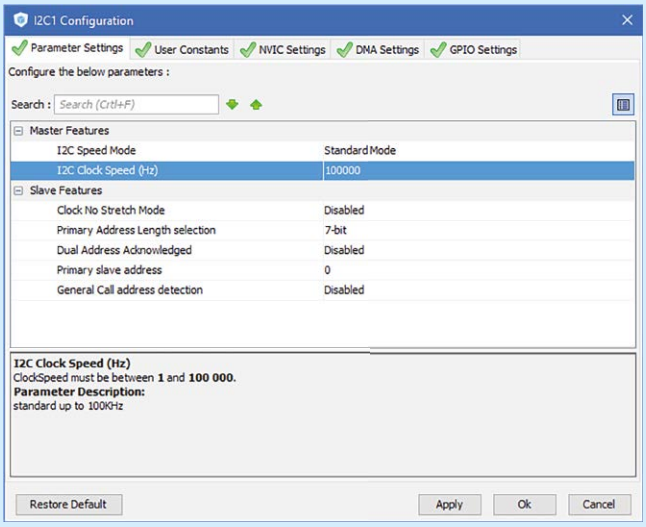
Rysunek 6. Zakładka „Clock Configuration" w programie STM32CubeMX

opis konfiguracji i działania interfejsu UART zawarto w czwartej części kursu. Aby uruchomić interfejs UART, z listy w lewej części okna CubeMX rozwijamy zakładkę „UART2" i w polu „Mode" wybieramy opcję „Asynchonous" (**rysunek 5**).

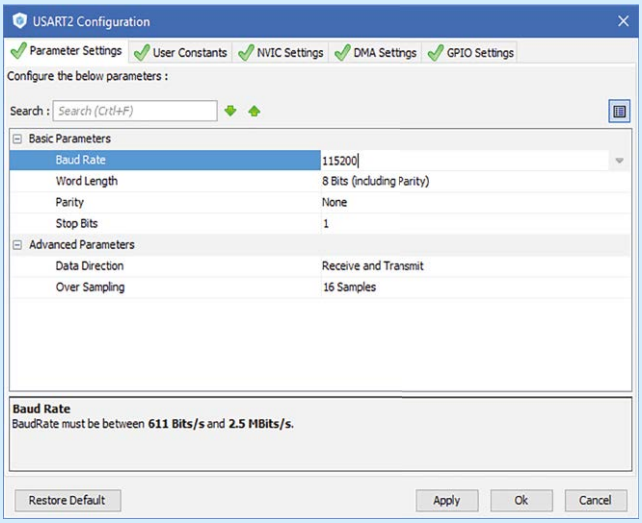
Po skonfigurowaniu wyprowadzeń, przechodzimy do zakładki „Clock Configuration" (**rysunek 6**) i w identyczny sposób, jak w poprzednich częściach, konfigurujemy sygnał taktujący rozchodzący się po układzie. Jeśli do układu mikrokontrolera jest dołączony zewnętrzny oscylator kwarcowy, z pola „PLL Source MUX" wybieramy pozycję „HSE" i w polu „Input frequency" wpisujemy częstotliwość (w MHz) przebiegu generowanego przez oscylator (na płytce Kamami KA-Nucleo jest to 8 MHz). Dalej, w polu „System Clock MUX" wybieramy pozycję „PLLCLK". Następnie, w polu „HCLK (MHz)" wpisujemy żądaną częstotliwość taktowania całego układu po przejściu przez pętlę PLL. Zazwyczaj wybieraliśmy w tym polu maksymalną dozwoloną częstotliwość - 100 MHz. Przy korzystaniu z modułu I²C nie jest to jednak możliwe, dlatego proponuję wpisać tam nieco niższą częstotliwość np. 80 MHz.

Teraz możemy już przejść do zakładki „Configuration" i wykonać ustawienia obu wykorzystywanych modułów – I2C1 oraz UART2. Aby przejść do konfiguracji interfejsu I²C klikamy przycisk „I2C1" znajdujący się w polu „Connectivity". Konfiguracja dzieli się na dwie sekcje – „Master Features" oraz „Slave Features". Tym razem, interesuje nas jedynie sekcja „Master Features" zawierająca parametry pracy urządzenia nadrzędnego, którym jest mikrokontroler. Układy STM32 mogą również pracować w trybie urządzenia podrzędnego, jako sterownik czujnika lub innego urządzenia. Istnieje też możliwość pracy w trybie MultiMaster, gdzie jest wiele urządzeń nadrzędnych.

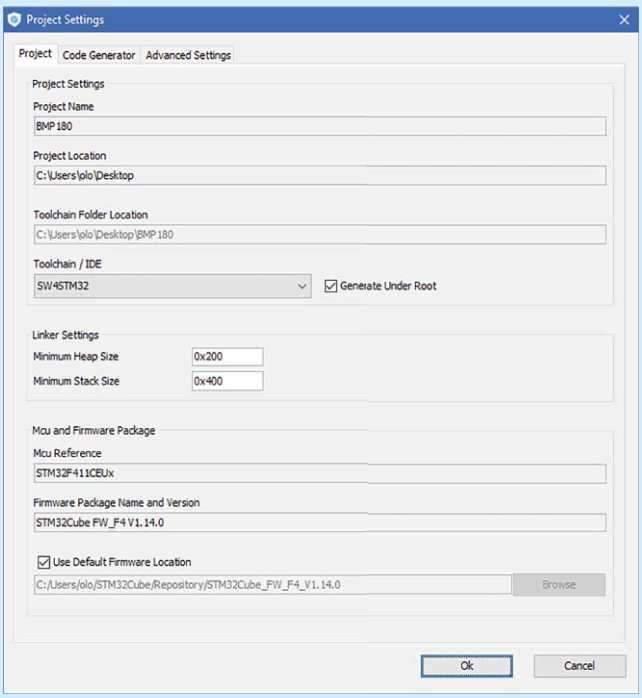
Pola „I2C Clock Mode" oraz „I2C Clock Speed (Hz)" określają szybkość transmisji. Przy wyborze trybu „Standard Mode"



Rysunek 7. Konfiguracja interfejsu I²C w programie STM32CubeMX

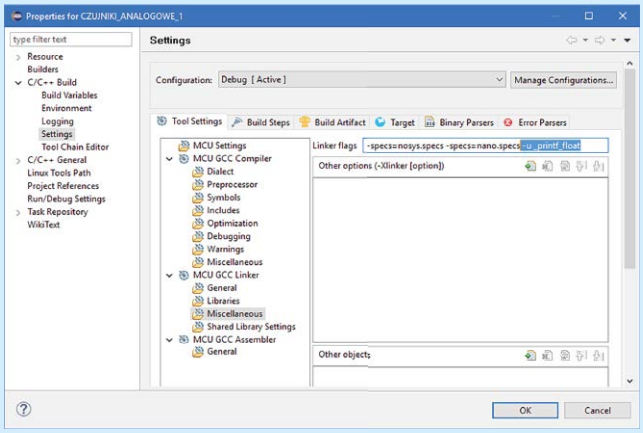


Rysunek 8. Konfiguracja interfejsu UART w programie STM32CubeMX

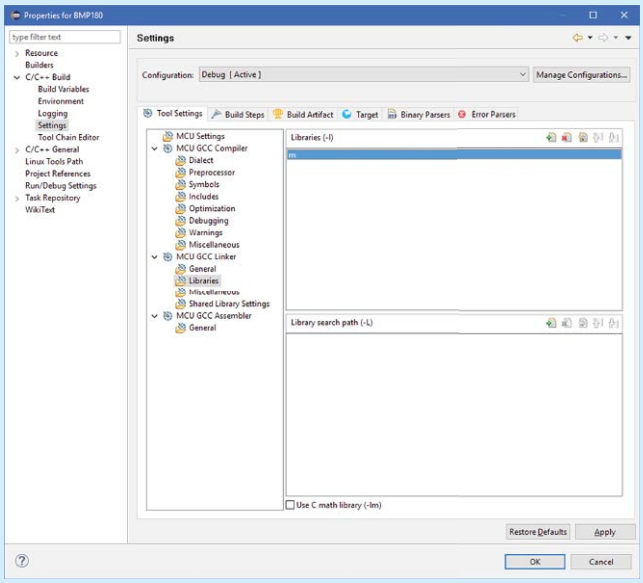


Rysunek 9. Konfiguracja projektu w STM32CubeMX

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **33372**, pass: **6nwd5fk4**



Rysunek 10. Zmiana parametrów wywołania linkera w środowisku IDE System Workbench for STM32



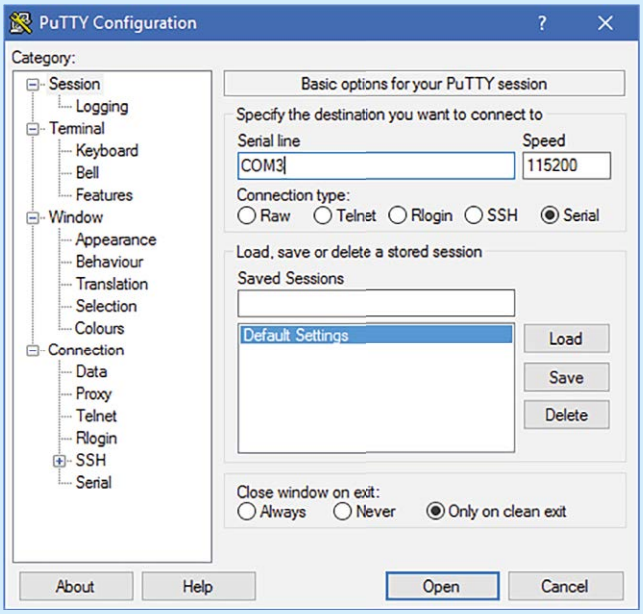
Rysunek 11. Dodawanie biblioteki *math* w ustawieniach linkera

maksymalna częstotliwość taktowania, którą możemy ustawić to 100 kHz. Po wyborze opcji „Fast mode" w polu „I2C Clock Speed (Hz)" możemy wybrać maksymalną częstotliwość równą 400 kHz. Dla trybu pracy urządzenia podrzędnego (sekcja „Slave Features"), najważniejszymi opcjami są „Primary address length selection" oraz „Primary slave address". Określamy w nich: długość adresu oraz sam adres urządzenia. Dla adresu 7-bitowego należy podać liczbę dziesiętną z zakresu od 0 do 127. Na podstawie tej wartości poprzez jej pomnożenie przez 2 lub pomnożenie przez 2 i dodanie 1, generowane są adres nadawczy i odbiorczy (**rysunek 7**).

Dla interfejsu UART zadowalają nas domyślne parametry połączenia – „Baud Rate: 115200", „Word Length: 8 bits", brak bitu parzystości (none) i 1 bit stopu (**rysunek 8**).

Nie pozostaje nam już nic innego, jak wygenerowanie projektu i zaimportowanie go w środowisku IDE. Będąc jeszcze w programie STM32CubeMX klikamy w 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 w przycisk „OK" (**rysunek 9**).

Uruchamiamy program System Workbench for STM32, zamykamy planszę powitalną, w ramce „Project Explorer" klikamy prawym przyciskiem myszy i z menu kontekstowego wybieramy



Rysunek 12. Ustawianie parametrów połączenia UART w programie PuTTY

„Import" ➝ „Existing Projects into Workspace". Podajemy ścieżkę dostępu, wybieramy nowo utworzony projekt i zatwierdzamy import przyciskiem „Finish".

Aby było możliwe korzystanie z wartości zmiennoprzecinkowych, w funkcji *sprintf()* jest konieczne dodanie parametru „-u _printf_float" do linii polecenia linkera. Robimy to klikając prawym przyciskiem myszy na nazwę nowego projektu. Z menu kontekstowego wybieramy pozycję „Properties" oraz nawigując do „C/C++ Build" ➝ „Settings" ➝ „Miscellaneous" i dopisując do pola „Linker flags" wartość: „ -u _printf_float" (**rysunek 10**).

Z powodu błędu w oprogramowaniu System Workbench, możemy mieć problem z kompilacją kodu zawierającego odwołania do funkcji z biblioteki *math*. Aby temu zaradzić, należy wejść w ustawienia projektu (klikamy prawym przyciskiem na jego nazwę i z menu wybieramy polecenie „Properties"). Następnie otwieramy kartę „C/C++ Build" ➝ „Settings" i w kolejnym panelu, wewnątrz nowej karty, rozwijamy „MCU GCC Linker" ➝ „Libraries", przewijamy kartę w dół, odznaczamy opcję „Use C math library" oraz dodajemy (przycisk z plusem), w polu „Libraries", bibliotekę o nazwie „m". Dalej klikamy przycisk „Apply" oraz „OK" (**rysunek 11**).

Modyfikujemy kod źródłowy plik „Src/main.c" zgodnie z **listingiem 1** oraz tworzymy dwa nowe pliki: „bmp180.h" (**listing 2**), w podfolderze „Inc" oraz „bmp180.c" w „Src" (**listing 3**). 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 w „Finish".

Po modyfikacjach zapisujemy zmiany w plikach, kompilujemy, wgrywamy i uruchamiamy program na mikrokontrolerze – klikamy w ikony młotka i robaka znajdujące się na pasku narzędziowym. Gdy program zostanie już uruchomiony, włączamy program PuTTY (opisany dokładniej w czwartej części serii), wybieramy w jego ustawianiach typ połączenia „Serial" i prędkość transmisji „115200 kbps" (**rysunek 12**). Nazwę portu szeregowego możemy sprawdzić w Menedżerze Urządzeń bądź dmesg-u i klikamy w przycisk „OK".

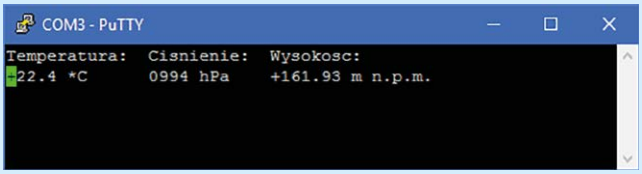
Plik nagłówkowy („bmp180.h") definiuje funkcje i struktury danych biblioteki. Jest dołączany wszędzie tam, gdzie istnieje potrzeba skorzystania z biblioteki. Towarzyszący mu plik „.c"

Listing 1. Plik programu głównego *main.c*

```
/* USER CODE BEGIN Includes */
#include „bmp180.h”
/* USER CODE END Includes */

int main(void)
{
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_I2C1_Init();
    MX_USART2_UART_Init();
    /* USER CODE BEGIN 2 */
    bmp_state bmp = bmp_init(&hi2c1);
    bmp_read_compensation_data(&bmp);
    double temperature, pressure, altitude;

    char output[50];
    sprintf(output, „Temperatura:  Cisnienie:  Wysokosc:
\r\n”);
    HAL_UART_Transmit(&huart2, output, strlen(output), 100);
    /* USER CODE END 2 */
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        /* USER CODE END WHILE */
        /* USER CODE BEGIN 3 */
        bmp_read_temp_and_pressure(&bmp);
        temperature = bmp_get_temperature(&bmp);
        pressure = bmp_get_pressure(&bmp);
        altitude = bmp_get_altitude(pressure, 1013.25);
        sprintf(output, „%+05.1f *C      %04.0f hPa     %+07.2f m
n.p.m.  \r”, temperature, pressure, altitude);
        HAL_UART_Transmit(&huart2, output, strlen(output), 100);
        HAL_Delay(100);
    }
    /* USER CODE END 3 */
}
```

Listing 2. Plik nagłówkowy *bmp180.h*

```
#include „i2c.h”

typedef struct bmp_state {
    I2C_HandleTypeDef * i2c;
    int16_t AC1, AC2, AC3, B1, B2, MB, MC, MD;
    uint16_t AC4, AC5, AC6;
    int32_t UT, UP, B5;
} bmp_state;

bmp_state bmp_init(I2C_HandleTypeDef * i2c);
uint8_t bmp_read_data(bmp_state * state, uint8_t reg);
void bmp_write_data(bmp_state * state, uint8_t reg, uint8_t val-
ue);
void bmp_read_compensation_data(bmp_state * state);
void bmp_read_temp_and_pressure(bmp_state * state);
double bmp_get_temperature(bmp_state * state);
double bmp_get_pressure(bmp_state * state);
double bmp_get_altitude(double p, double p0);
```



Rysunek 13. Odbiór danych w programie PuTTY

(„bmp180.c”) zawiera kod źródłowy tych funkcji. Aby korzystając z biblioteki wykonać pomiar i wydobyć wartości zmierzone, należy kolejno: znacjonalizować strukturę przechowującą odczytane wartości (*bmp_state bmp = bmp_init(&hi2c1);*), odczytać do tej struktury dane kompensujące wyniki pomiarów (*bmp_read_compensation_data(&bmp);*), wykonać pomiary temperatury i ciśnienia (*bmp_read_temp_and_pressure(&bmp);*), odczytać skompensowaną wartość temperatury (*double temperature = bmp_get_temperature(&bmp);*), ciśnienia atmosferycznego (*doubel pressure = bmp_get_pressure(&bmp);*) oraz obliczyć na podstawie ciśnienia wysokość nad poziomem morza (*double altitude = bmp_get_altitude(pressure, 1013.25);*). Zwracana przez funkcje wartości podawane są w następujących jednostkach:

- temperatura – stopnie Celsjusza,
- ciśnienie – hektopaskale,
- wysokość – metry nad poziomem morza.

Następnie, w pliku „main.c” za pomocą funkcji *sprintf()* oraz *HAL_UART_Transmit()* parametry te zapisywane są do zmiennej typu *string* i wysyłane do komputera interfejsem UART (**rysunek 13**). Znaki „\r” oraz „\n” użyte w instrukcji *sprintf()* oznaczają, kolejno: powrót na początek linii oraz przejście do nowej linii. Dzięki zastosowaniu jedynie instrukcji „\r” w drugim wywołaniu funkcji *sprintf()* wyniki wyświetlane w oknie PuTTY są cały czas nadpisywane w tej samej linii.

Interfejs I²C, podobnie jak pozostałe omawiane, może być używany w trybie DMA oraz w przerwaniach. Wywołania funkcji są niemal identyczne jak dla interfejsu SPI, czy UART.

Aleksander Kurczyk

Listing 3. Plik biblioteki *bmp180.c*

```
#include „bmp180.h”

bmp_state bmp_init(I2C_HandleTypeDef * i2c)
{
    bmp_state state;
    state.i2c = i2c;
    state.AC1 = 0;
    state.AC2 = 0;
    state.AC3 = 0;
    state.AC4 = 0;
    state.AC5 = 0;
    state.AC6 = 0;
    state.B1 = 0;
    state.B2 = 0;
    state.MB = 0;
    state.MC = 0;
    state.MD = 0;
    state.UT = 0;
    state.UP = 0;
    state.B5 = 0.0;
    return state;
}

uint8_t bmp_read_data(bmp_state * state, uint8_t reg)
{
    uint8_t tmp = 0;
    HAL_I2C_Mem_Read(state->i2c, 0xEF, reg, 1, &tmp, 1, 100);
    return tmp;
}

void bmp_write_data(bmp_state * state, uint8_t reg, uint8_t value)
{
    HAL_I2C_Mem_Write(state->i2c, 0xEE, reg, 1, &value, 1, 100);
}

void bmp_read_compensation_data(bmp_state * state)
{
    state->AC1 = (bmp_read_data(state, 0xAA) << 8) + bmp_read_data(state, 0xAB);
    state->AC2 = (bmp_read_data(state, 0xAC) << 8) + bmp_read_data(state, 0xAD);
    state->AC3 = (bmp_read_data(state, 0xAE) << 8) + bmp_read_data(state, 0xAF);
    state->AC4 = (bmp_read_data(state, 0xB0) << 8) + bmp_read_data(state, 0xB1);
    state->AC5 = (bmp_read_data(state, 0xB2) << 8) + bmp_read_data(state, 0xB3);
    state->AC6 = (bmp_read_data(state, 0xB4) << 8) + bmp_read_data(state, 0xB5);
    state->B1 = (bmp_read_data(state, 0xB6) << 8) + bmp_read_data(state, 0xB7);
    state->B2 = (bmp_read_data(state, 0xB8) << 8) + bmp_read_data(state, 0xB9);
```

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **33372**, pass: **6nwd5fk4**

Listing 3. cd.

```
    state->MB = (bmp_read_data(state, 0xBA) << 8) + bmp_read_data(state, 0xBB);
    state->MC = (bmp_read_data(state, 0xBC) << 8) + bmp_read_data(state, 0xBD);
    state->MD = (bmp_read_data(state, 0xBE) << 8) + bmp_read_data(state, 0xBF);
}

void bmp_read_temp_and_pressure(bmp_state * state)
{
    bmp_write_data(state, 0xF4, 0x2E);
    HAL_Delay(5);
    state->UT = (bmp_read_data(state, 0xF6) << 8) + bmp_read_data(state, 0xF7);
    uint8_t OSS = 3; // „ultra high resolution”
    bmp_write_data(state, 0xF4, 0x34 + (OSS << 6));
    if(OSS == 0) HAL_Delay(5);
        else if(OSS == 1) HAL_Delay(8);
            else if(OSS == 2) HAL_Delay(14);
                else HAL_Delay(26);
    state->UP = ((bmp_read_data(state, 0xF6) << 16) + (bmp_read_data(state, 0xF7) << 8) + bmp_read_data(state, 0xF8)) >> 8-OSS;
}

double bmp_get_temperature(bmp_state * state)
{
    int32_t X1, X2, T;
    X1 = (state->UT - state->AC6) * state->AC5 / 32768;
    X2 = state->MC * 2048 / (X1 + state->MD);
    state->B5 = X1 + X2;
    T = (state->B5 + 8) / 16.0;
    return T / 10.0;
}

double bmp_get_pressure(bmp_state * state)
{
    int32_t B6, X1, X2, X3, B3, P;
    uint32_t B4, B7;
    uint8_t OSS = 3;

    B6 = state->B5 - 4000;
    X1 = (state->B2 * (B6 * B6 / 4096)) / 2048;
    X2 = state->AC2 * B6 / 2048;
    X3 = X1 + X2;
    B3 = (((state->AC1 * 4 + X3) << OSS) + 2) / 4;
    X1 = state->AC3 * B6 / 8192;
    X2 = (state->B1 * (B6 * B6 / 4096)) / 65536;
    X3 = ((X1 + X2) + 2) / 4;
    B4 = state->AC4 * (uint32_t)(X3 + 32768) / 32768;
    B7 = ((uint32_t)state->UP - B3) * (50000 >> OSS);
    if(B7 < 0x80000000) P = (B7 * 2) / B4; else P = (B7 / B4) * 2;
    X1 = (P / 256) * (P / 256);
    X1 = (X1 * 3038) / 65536;
    X2 = (-7357 * P) / 65536;
    P = P + (X1 + X2 + 3791) / 16;
    return P / 100.0;
}

double bmp_get_altitude(double p, double p0)
{
    return 44330 * (1 - pow((p/p0), (1/5.255)));
}
```