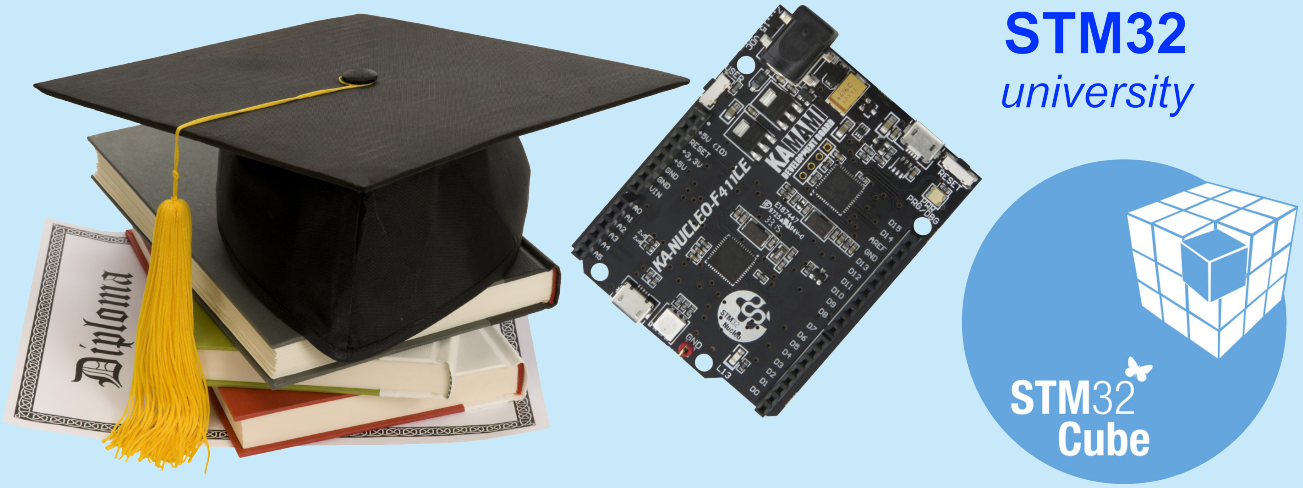


Programowanie STM32F4(7)

W artykule zajmiemy się konwerterem A/C wbudowanym w układ mikrokontrolera i za jego pomocą zmierzymy temperaturę panującą na układzie. Do wykonania tego ćwiczenia nie będzie potrzebne żadne dodatkowe urządzenie – wystarczy dowolna płytka rozwojowa z układem STM32F4 wyposażonym w przetwornik A/C oraz komputer z zainstalowanym oprogramowaniem STM32CubeMX i środowiskiem System Workbench for STM32.

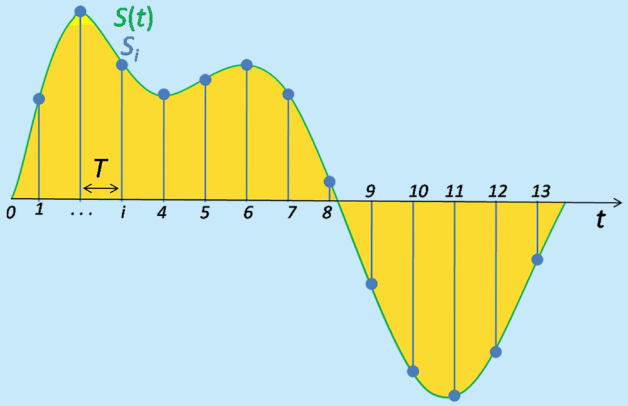
W używanym przeze mnie układzie STM32F411CEU6 do dyspozycji użytkownika jest jeden 13-kanałowy przetwornik A/C, z czego 10 kanałów jest skojarzonych z funkcjami alternatywnymi 10 wyprowadzeń mikrokontrolera, a 3 z nich są dołączone do czujników wbudowanych w układ. Przetwornik pracuje z maksymalną rozdzielczością 12 bitów, umożliwiając pomiar wartości z dokładnością do 806 mikrowoltów (przy standardowej konfiguracji napięcia odniesienia 3,3 V).

Jak działa przetwornik A/C?

Czujniki analogowe dostarczają nam informacje o mierzonych wartościach w postaci ciągłych, analogowych zmian napięcia lub innego parametru przepływającego przez nie prądu elektrycznego (natężenia, amplitudy, częstotliwości oscylacji). Najpopularniejszym typem czujników są te operujące na napięciu. Nasz przetwornik dokonuje pomiaru różnicy potencjałów pomiędzy wyprowadzeniem danego kanału a wspólną masą. Czujniki zewnętrzne mają najczęściej 3 wyprowadzenia – napięcie zasilania/odniesienia, masę oraz pin, na którym ustawiane jest napięcie pomiędzy napięciem odniesienia a masą.

Próbkowanie

Ponieważ poziom napięcia stale zmienia się w czasie, aby umożliwić jego zapis cyfrowy, należy pobrać w określonych odstępach czasowych jedną lub więcej próbek, czyli wartości napięcia w konkretnych chwilach (**rysunek 1**). Jeśli korzystamy z przetwornika A/C do zapisu fali akustycznych lub innych sygnałów zmiennych w czasie, zgodnie z prawem Shannona sygnał taki musimy próbkować z częstotliwością równą co najmniej dwukrotności maksymalnej częstotliwości występującej w przetwarzanym sygnale.

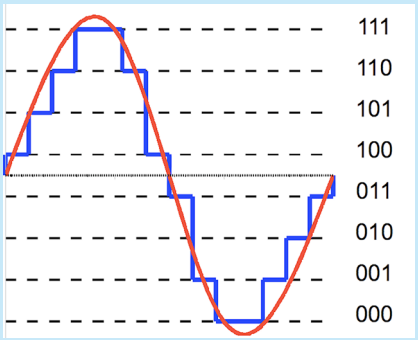


Rysunek 1. Próbkowanie (źródło: https://en.wikipedia.org/wiki/File:Signal_Sampling.png)

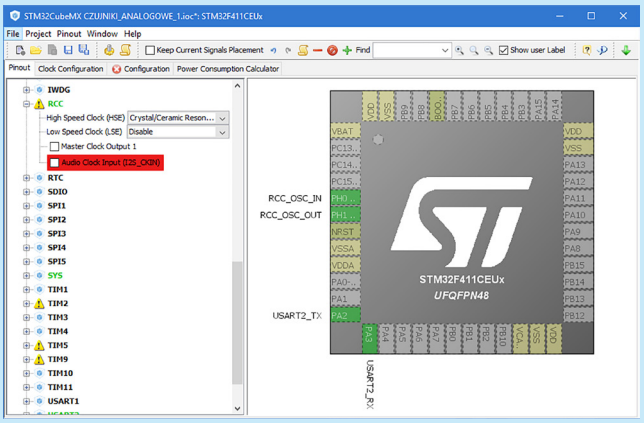
Kwantyzacja

Po próbkowaniu zapis sygnału wciąż nie jest jednak cyfrowy. Poziomy napięć w próbkach nadal są analogowe i mają nieskończoną precyzję. Dlatego procesowi próbkowania zawsze towarzyszy proces kwantyzacji, czyli przypisania wartościom napięcia z pewnego zakresu możliwie najbliższych im wartości cyfrowych, zapisanych w postaci liczb binarnych ze stałą precyzją (**rysunek 2**). Przykładowo, liczb z zakresu od 0 do 4095 dla napięcia z zakresu 0,0…3,3 V, gdzie 0 odpowiada napięciu 0 V, a 4095 – 3,3 V.

Podstawowym parametrem pracy przetwornika A/C jest jego rozdzielczość, zwyczajowo określana w postaci liczby bitów, na których zapisywane/kwantyzowane są wartości kolejnych próbek. Im wyższa ich liczba, tym lepiej. Może się to jednak odbić na szybkości



Rysunek 2. Kwantyzacja (źródło: https://en.wikipedia.org/wiki/File:3-bit_resolution_analog_comparison.png



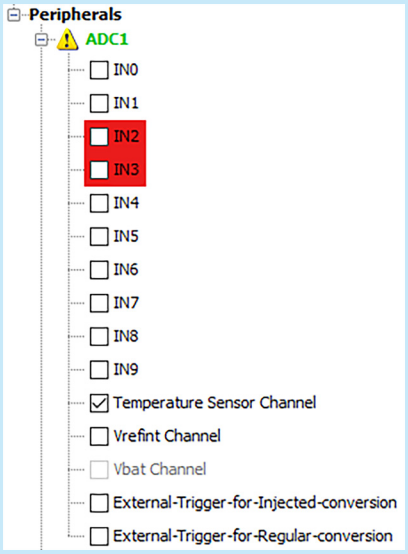
Rysunek 3. Konfiguracja wyprowadzeń układu w programie STM32CubeMX

pracy układu przetwornika. Typowe rozdzielczości to 8, 12, 16 lub 24 bity, odpowiadające kolejno 256, 4096, 65536 oraz 16777216 różnym możliwym do zidentyfikowania wartościom.

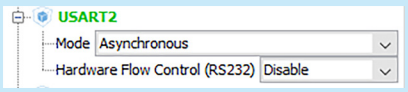
Pierwszy przykład – odczyt wartości temperatury na układzie mikrokontrolera

W ramach pierwszego przykładu odczytamy temperaturę zmierzoną za pomocą termometru wbudowanego w układ mikrokontrolera i prześlemy jej wartość do komputera interfejsem UART.

1. Uruchamiamy narzędzie STM32CubeMX. Tworzymy w nim nowy projekt i w oknie wyboru mikrokontrolera wybieramy posiadany przez nas model mikrokontrolera. Dla przypomnienia – podczas tworzenia tego kursu wykorzystywaną przeze mnie płytką rozwojową jest zestaw KA-NUCLEO–F411CE z układem STM32F411CEU6.
2. Na pierwszej planszy głównego okna programu CubeMX – „Pinout", podobnie jak w poprzednich częściach, z listy po lewej stronie rozwijamy pozycję „RCC" i z pola „High Speed Clock (HSE)" wybieramy opcję „Crystal/Ceramic Resonator" (**rysunek 3**).
3. Następnie rozwijamy pozycję ADC1 i zaznaczamy na liście jedynie pozycję „Temperature Sensor Channel" (**rysunek 4**). Na tej liście znajdują się wszystkie kanały wejściowe pierwszego i jedynego (w przypadku tego konkretnego układu) konwertera cyfrowo-analogowego. Pozycje od IN0 do IN9 odpowiadają wyprowadzeniom układu – odpowiednio PA0-PA7 oraz PB0 i PB1. Jeśli zostaną zaznaczone, na tych pinach będzie możliwy odczyt analogowych wartości przyłożonych napięć pomiędzy danym pinem a wspólną masą. „Temperature Sensor Channel", „Vrefint Channel", „Vbat Channel" to kanały wewnętrzne – przyłączone do wbudowanych w układ czujników bądź punktów pomiarowych. **„Temperature Sensor Channel"** to kanał wbudowanego termometru. **„Vrefint Channel"** to kanał napięcia referencyjnego, stabilizowanego wewnątrz układu. Za jego pomocą dokonywane są pozostałe pomiary. **„Vbat Channel"** umożliwia pomiar napięcia zasilania. W tym przykładzie jest wykorzystany wbudowany termometr, zasada wykonywania pomiarów jest jednak identyczna, również w przypadku pozostałych, w tym zewnętrznych czujników.



Rysunek 4. Konfiguracja kanałów przetwornika analogowo-cyfrowego w programie STM32CubeMX

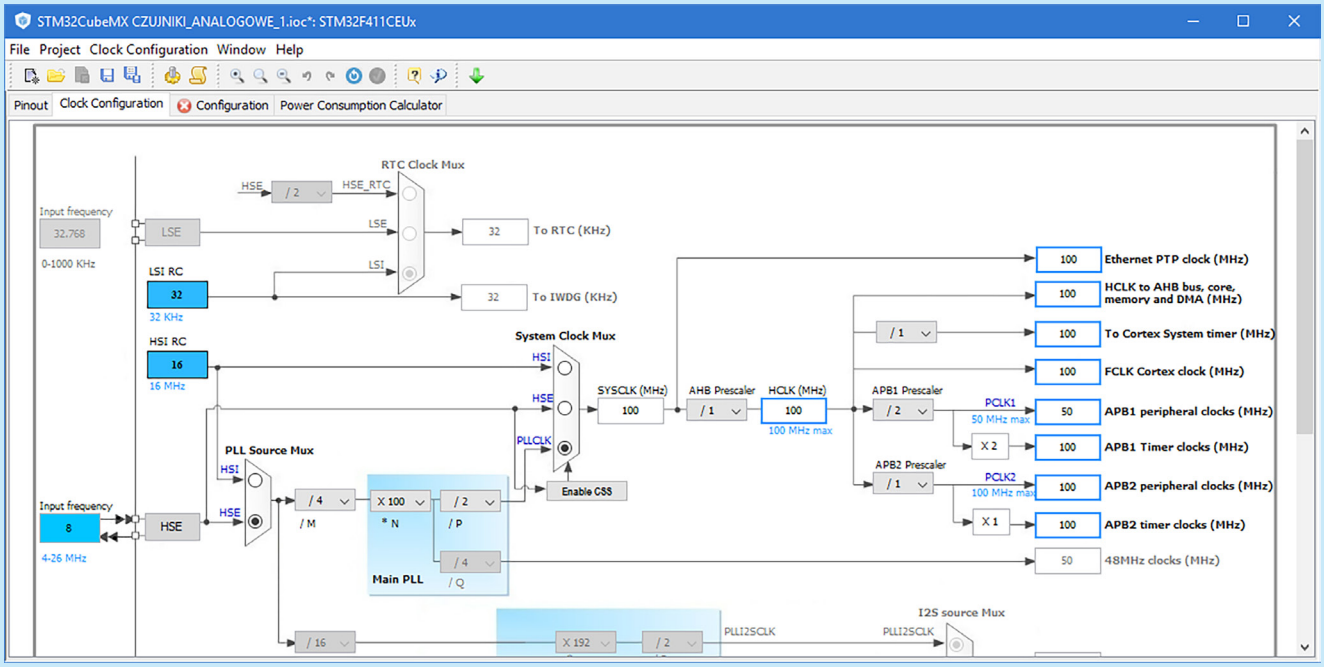


Rysunek 5. Konfiguracja wyprowadzeń interfejsu UART w programie STM32CubeMX

4. Do wykonania projektu potrzebny będzie nam również peryferial UART. Najlepiej ten przyłączony do programatora, znajdującego się na płytce, aby możliwe było przesłanie danych do komputera tym samym kablem USB, którym programujemy mikrokontroler. W przypadku płytki rozwojowej KA-NUCLEO–F411CE skorzystamy z interfejsu „USART2" na pinach PA2 i PA3. W tym celu rozwijamy pozycję „USART2" i z pola „Mode" wybieramy opcję „Asynchronous" (**rysunek 5**).
5. Przechodzimy do kolejnej planszy – „Clock Configuration" i ustawiamy na niej parametry tak, jak w poprzednich częściach. W pole „HSE – Input Frequency" wpisujemy wartość częstotliwości zewnętrznego oscylatora kwarcowego podłączonego do układu. Na płytce KA-NUCLEO–F411CE jest to 8 MHz. Z pola „PLL Source Mux" wybieramy opcję „HSE", z pola „System Clock Mux" – „PLLCLK". W pole „HCLK" wpisujemy wartość maksymalną częstotliwości taktowania obsługiwanej przez nasz układ (u mnie 100 MHz). Dokładne znaczenie tych parametrów zostało omówione w pierwszej części cyklu (**rysunek 6**).
6. Dalej przechodzimy do zakładki „Configuration" i konfigurujemy wykorzystywane peryferiale – „ADC1" i „USART2". W wypadku interfejsu „UART2", upewniamy się jedynie, czy domyślnie ustawione są następujące parametry: „Baud Rate" równy „1115200 bps", „Word Length" – „8 bitów" oraz „Stop Bits" – „1". Znaczenie tych parametrów omówione zostało w trzeciej części kursu.

W konfiguracji konwertera „ADC1" dokonujemy jednak kilku zmian. W sekcji „ADC_Settings" ustawiamy następujące parametry:

- **Clock Prescaler** – to nic innego, jak kolejny dzielnik częstotliwości sygnału taktującego, wchodzącego na układ konwertera. Układ „ADC1" taktowany jest sygnałem z linii APB2 (PCLK2), dla której częstotliwość wybrana została w poprzedniej zakładce głównego okna programu CubeMX. W układzie STM32F411CEU6 jego wartość maksymalna (a także obecnie

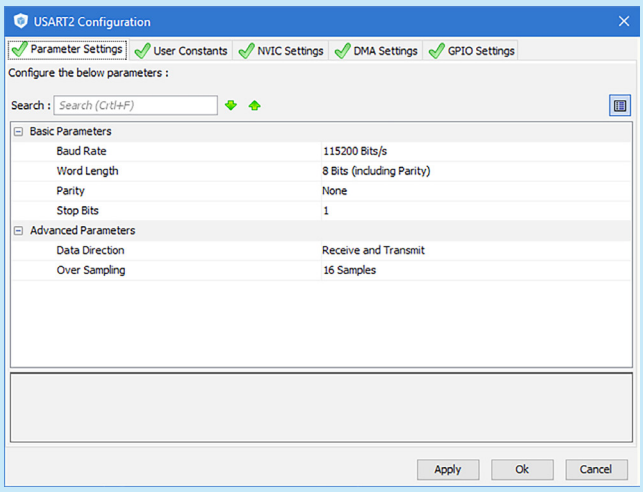


Rysunek 6. Konfiguracja sygnału taktującego w programie STM32CubeMX

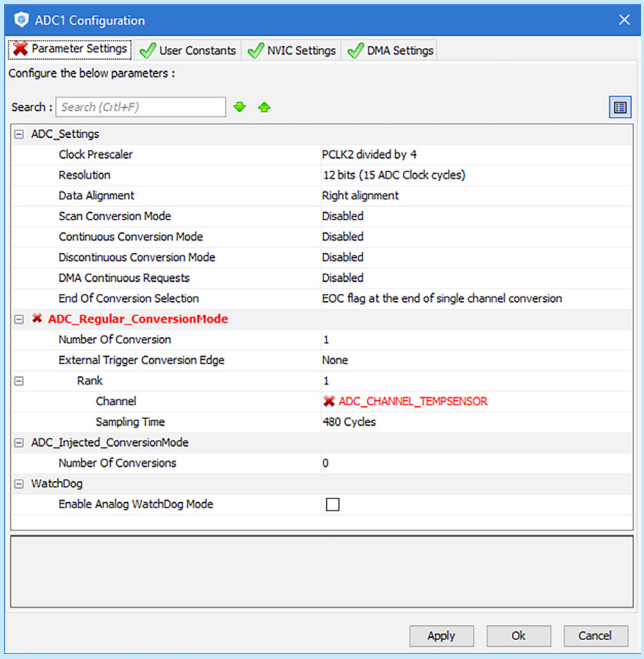
ustawiona) to 100 MHz. Dostępne wartości preskalera to 4, 6 i 8. Wartość ta wpływa na szybkość pracy konwertera. Jeśli jest niska, pomiar wykonywany jest szybciej, ale za to wynik może być obarczony błędem wynikającym z szybkozmiennych zakłuceń. W przykładzie wartość ta została ustawiona na możliwie najniższą – 4 (co daje nam częstotliwość taktowania przetwornika równą 25 MHz), dla pomiaru temperatury może ona jednak być dowolna.

- **Resolution** – tutaj ustawiamy możliwie dużą wartość. Jest to rozdzielczość pracy naszego konwertera. Im wyższa, tym dłużej trwa pomiar, jednak zyskujemy większą dokładność odczytanej wartości.
- **Data Alignment** – na „Right Alignment". Opcja ta decyduje o sposobie umieszczenia mierzonej wartości w 32-bitowym rejestrze/komórce pamięci. Jeśli rozdzielczość to 12 bitów, dane, w 32-bitowej komórce, zapisane mogą zostać na 12 najmłodszych bitach („z prawej") lub 12 najstarszych („z lewej").
- **Scan Conversion Mode** – na „Disabled". Opcja ta określa, czy wykonujemy pomiary po kolei ze wszystkich kanałów, czy pojedynczo. Korzystamy tylko z jednego kanału, jej ustawienie nie ma więc znaczenia.

Do pozostałych parametrów z sekcji „ADC_Settings" powrócimy w kolejnym przykładzie. W kolejnej sekcji – „ADC_Regular_ConversionMode" określamy, ile konwersji wartości analogowych z jakich kanałów ma zostać wykonanych po pojedynczym wywołaniu oraz kiedy nastąpić ma początek pomiaru. Ponieważ konwersję będziemy rozpoczynać w kodzie programu, a nie po sygnale z zewnętrznego pinu, wartość w polu „External Trigger Conversion Edge" ustawiamy na „None". Pole „Number of Conversion" decyduje o liczbie wartości mierzonych po wywołaniu konwersji. Pozostawiamy tutaj wartość domyślną – 1. Jej zmiana powodowałaby dodanie kolejnych sekcji „Rank", poniżej, decydujących o kanałach, z jakich pobierane są kolejne wartości oraz liczbie powtórzeń pomiarów. W jedynej sekcji „Rank" jako kanał źródłowy wybieramy „ADC_CHANNEL_TEMPSENSOR", a jako „Sampling Time" wybieramy największą wartość – 480, spowoduje ona uśrednienie



Rysunek 7. Konfiguracja parametrów pracy interfejsu UART w programie STM32CubeMX



Rysunek 8. Konfiguracja parametrów pracy przetwornika analogowo-cyfrowego w programie STM32CubeMX

wyniku z wielu pomiarów (wykonywanych przez 480 cykli zegara), mając oczywiście negatywny wpływ na czas trwania pomiaru, ale tym samym polepszając dokładność.

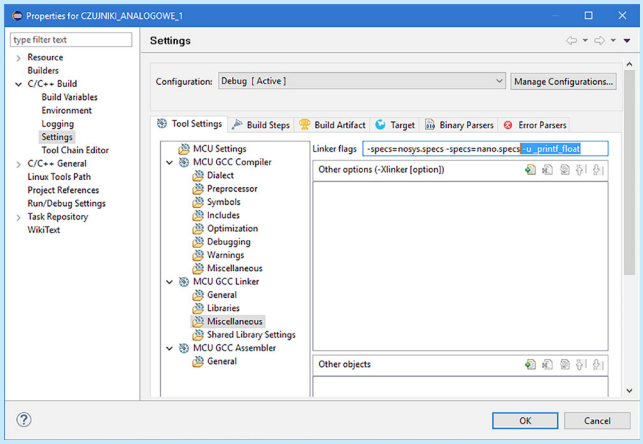
Konfigurację parametrów przetwornika A/C w programie STM32CubeMX pokazano na **rysunku 8**.

7. Po dokonaniu wszystkich zmian możemy wygenerować kod projektu i zaimportować go w środowisku System Workbench for STM32. Klikamy ikonę zębatki w pasku menu. W nowym oknie wybieramy nazwę projektu oraz ścieżkę dostępu do folderu, w którym ma być zapisany. W polu „Toolchain / IDE" wybieramy opcję „SW4STM32". W zakładce „Code Generator" możemy też zaznaczyć opcję „Generate peripheral initialization as pair od ‚.c/.h'...". Następnie klikamy OK.
8. W środowisku 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 wygenerowany projekt i klikamy przycisk „Finish".
9. Aby możliwe było korzystanie z wartości zmiennoprzecinkowych, w funkcji sprintf() konieczne jest dodanie parametru „-u _printf_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" (**rysunek 9**).
10. Modyfikujemy kod źródłowy plik „Src/main.c" zgodnie z **listingiem 1**. W kodzie z listingu nr 1, w sekcji „USER CODE 2" definiujemy kilka stałych pomocnych przy wykonywaniu obliczeń oraz uruchamiamy pomiar przetwornikiem analogowo-cyfrowym. Następnie, w pętli głównej, w sekcji „USER CODE 3" oczekujemy aktywnie na zakończenie pomiaru, pobieramy zmierzoną wartość do zmiennej, obliczamy kolejno: zmierzoną wartość napięcia (dzielimy skwantyzowany odczyt przez maksymalny możliwy i mnożymy uzyskaną wartość przez wartość napięcia odniesienia) oraz temperaturę w stopniach Celsjusza – korzystając ze wzoru (oraz stałych) podanych w karcie katalogowej układu tj. **((zmierzone napięcie – napięcie_w_temperaturze_25_stopni) / liczba_woltów_na_stopień_celsjusza) + 25**). Dalej wartość ta zapisywana jest do zmiennej typu string oraz wysyłana do komputera interfejsem UART. Przed ponownym wykonaniem pętli każemy naszemu mikrokontrolerowi odczekać jeszcze 100 milisekund i ponowić pomiar. Wartość „**%+5.2f \r**", użyta w wywołaniu funkcji sprintf(), oznacza zapis liczby zmiennoprzecinkowej w formacie ze znakiem (±), na pięciu znakach ASCII, z dwoma miejscami po przecinku, dwie spacje oraz znak powrotu na początku linii, tak aby wartość w oknie terminalu była cały czas nadpisywana.
11. Teraz możemy już uruchomić nasz kod – ikona młotka oraz robaka w środowisku System Workbench for STM32, i odczytać wartość temperatury, korzystając z programu PuTTY lub dowolnego innego emulatora terminalu.

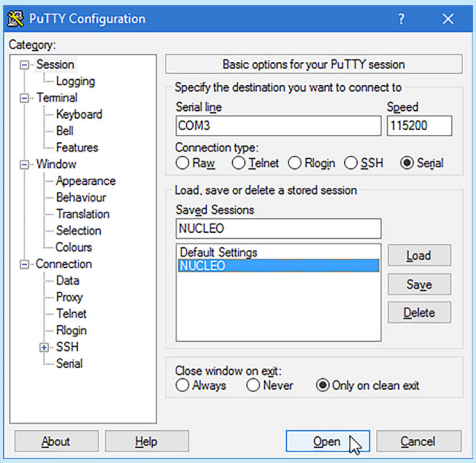
Ustawienia parametrów transmisji szeregowej pokazano na **rysunku 10**, a wynik pracy programu na **rysunku 11**.

Drugi przykład – odczyt wartości z konwertera ADC bezpośrednio do zmiennej (tryb DMA)

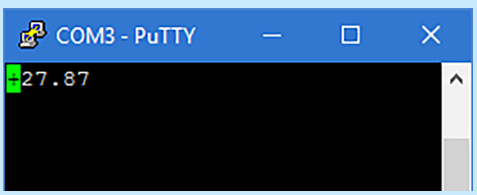
Obecnie, po rozpoczęciu pomiaru, procesor aktywnie oczekuje na jego zakończenie, nie mogąc



Rysunek 9. Zmiana parametrów wywołania linkera w środowisku IDE System Workbench for STM32



Rysunek 10. Ustawienia połączenia szeregowego w programie PuTTY



Rysunek 11. Temperatura w stopniach Celsjusza wyświetlana w programie PuTTY

Listing 1. Modyfikacje pliku źródłowego dla przykładu 1

```
/* USER CODE BEGIN 2 */
const double voltage_at_25 = 0.76;
const double volts_per_degree = 0.0025;
const double max_voltage = 3.3;
const double resolution = 4095.0;

HAL_ADC_Start(&hadc1);
/* USER CODE END 2 */

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

    /* USER CODE BEGIN 3 */
    if (HAL_ADC_PollForConversion(&hadc1, 100) == HAL_OK)
    {
        uint32_t reading = HAL_ADC_GetValue(&hadc1);
        double voltage = (reading / resolution) * max_voltage;
        double temperature = ((voltage - voltage_at_25) / volts_per_degree) + 25;

        uint8_t text[10];
        sprintf(text, „%+5.2f  \r", temperature);
        HAL_UART_Transmit(&huart2, text, strlen(text), 100);

        HAL_Delay(100);
        HAL_ADC_Start(&hadc1);
    }
}
/* USER CODE END 3 */
```

w tym czasie wykonywać innych czynności. W tak małym projekcie nie stanowi to problemu. W bardziej złożonych projektach, gdzie do obsłużenia jest kilka peryferiali, do wykonania kilka pomiarów oraz obliczeń, a wszystkim tym rzeczom należy zapewnić odpowiedni czas wykonania, może to stanowić problem. Rozwiązaniem jest tryb DMA, pozwalający na automatyczną transmisję danych z peryferiali do pamięci RAM mikrokontrolera, bez angażowania do tego zasobów procesora. W przypadku przetwornika analogowo-cyfrowego możemy uruchomić ciągły pomiar i zapisywać kolejne wyniki do zmiennej w pamięci, a następnie wykorzystać te dane w kodzie programu.

Aby to zrobić, modyfikujemy poprzedni przykład, zgodnie z poniższą instrukcją:

12. Uruchamiamy program STM32CubeMX, nawigujemy do zakładki „Configuration" i otwieramy ustawienia przetwornika ADC1. W polach „Contignous Conversion Mode" oraz „DMA Contignous Request" ustawiamy wartości „Enabled". Włączenie opcji „Contignous Conversion Mode" powoduje, że po zakończeniu pomiaru przetwornik automatycznie rozpocznie kolejny. Opcja „DMA Contignous Request" powoduje, że po każdym pomiarze uzyskana wartość będzie zapisywana do pamięci (**rysunek 12**).
13. Przechodzimy do zakładki „DMA Settings" i klikamy przycisk „Add". Z pola „DMA Request", w nowo utworzonym wpisie na liście kanałów DMA, wybieramy pozycję „ADC1". Następnie rozwijamy listę „Mode" i wybieramy z niej opcję „Circular" oraz zapisujemy ustawiania. W ten sposób uruchamiamy mechanizm DMA i każemy zapisywać zmierzone wartości cały czas do tej samej zmiennej, a nie na przykład do kolejnych komórek tablicy (**rysunek 13**).
14. Teraz możemy już wygenerować zmieniony projekt – klikając przycisk zębatki w pasku menu, przejść do środowiska IDE System Workbench i zmodyfikować kod w pliku „Src/main.c" zgodnie z **listingiem 2**.

Listing 2. Modyfikacje pliku źródłowego dla przykładu 2

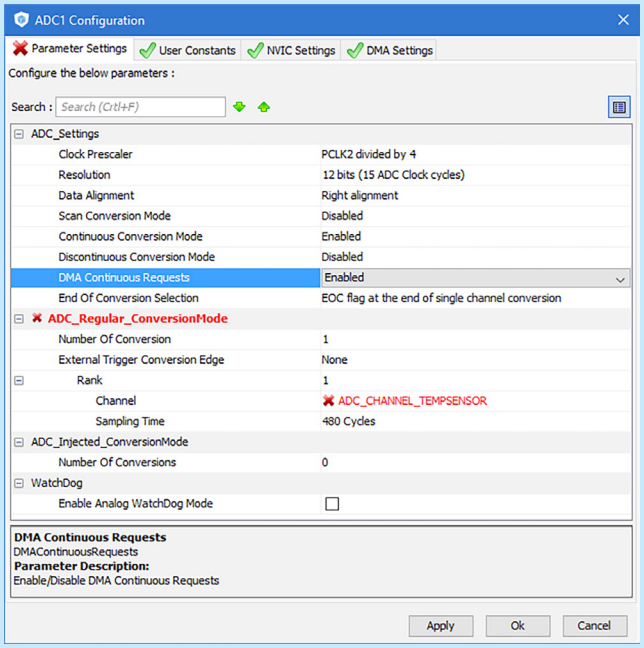
```
/* USER CODE BEGIN PV */
/* Private variables */
const double voltage_at_25 = 0.76;
const double volts_per_degree = 0.0025;
const double max_voltage = 3.3;
const double resolution = 4095.0;

volatile uint32_t reading = 0;
/* USER CODE END PV */

/* USER CODE BEGIN 2 */
HAL_ADC_Start_DMA(&hadc1, &reading, 1);
/* USER CODE END 2 */

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

/* USER CODE BEGIN 3 */
    double voltage = (reading / resolution) * max_voltage;
    double temperature = ((voltage - voltage_at_25) / volts_per_degree) + 25;
    uint8_t text[10];
    sprintf(text, „%+5.2f  \r", temperature);
    HAL_UART_Transmit(&huart2, text, strlen(text), 100);
    HAL_Delay(100);
}
/* USER CODE END 3 */
```

W kodzie z listingu 2, deklaracja stałych oraz zmiennej przechowującej zmierzoną wartość przeniesiona została do sekcji globalnej USER CODE PV, nie musimy też w kolejnych iteracjach pętli głównej, w sekcji USER CODE 2, oczekiwać na wykonanie pomiaru ani co iterację uruchamiać kolejnego. Teraz zawsze najnowsza zmierzona wartość znajduje się w zmiennej „reading", z której korzystamy przy wyliczaniu temperatury tuż przed jej wysyłką do komputera.

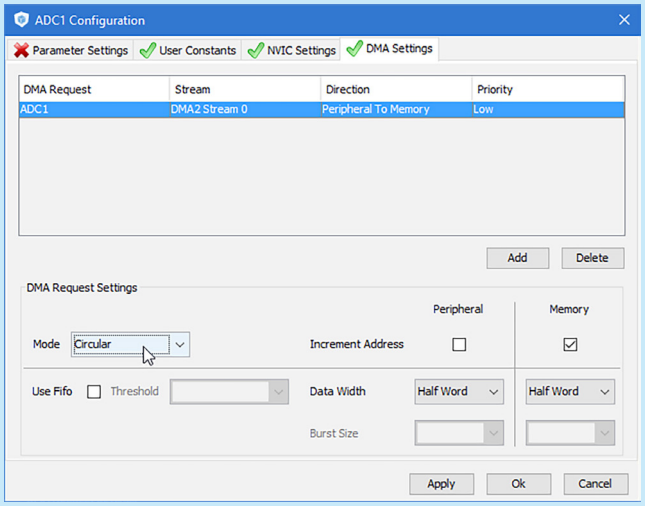
Do czego jeszcze możemy wykorzystać przetwornik analogowo-cyfrowy?

Przetwornik A/C może zostać wykorzystany w wielu sytuacjach. Nie służy on tylko do odczytu danych z czujnikach. Często jest wykorzystywany do przyjmowania sygnałów zmiennych w czasie – audio i innych, a także w roli wejścia, w interakcji z użytkownikiem. Do wejścia A/C możemy dołączyć potencjometr analogowy. Przekręcając go, użytkownik może zmieniać parametry pracy programu. Jeśli zabraknie nam wolnych pinów do podłączenia przycisków, konstruując odpowiednią siatkę rezystorów, możemy do portu A/C przyłączyć wiele przycisków, tak dobierając rezystancję, aby wciśnięcie innego przycisku dało w efekcie inne napięcie między pinem A/C a masą.

Aleksander Kurczyk



Rysunek 12. Zmiana parametrów pracy przetwornika A/C w programie STM32CubeMX



Rysunek 13. Konfiguracja kanału DMA na potrzeby transmisji danych z przetwornika do pamięci RAM mikrokontrolera w programie STM32CubeMX