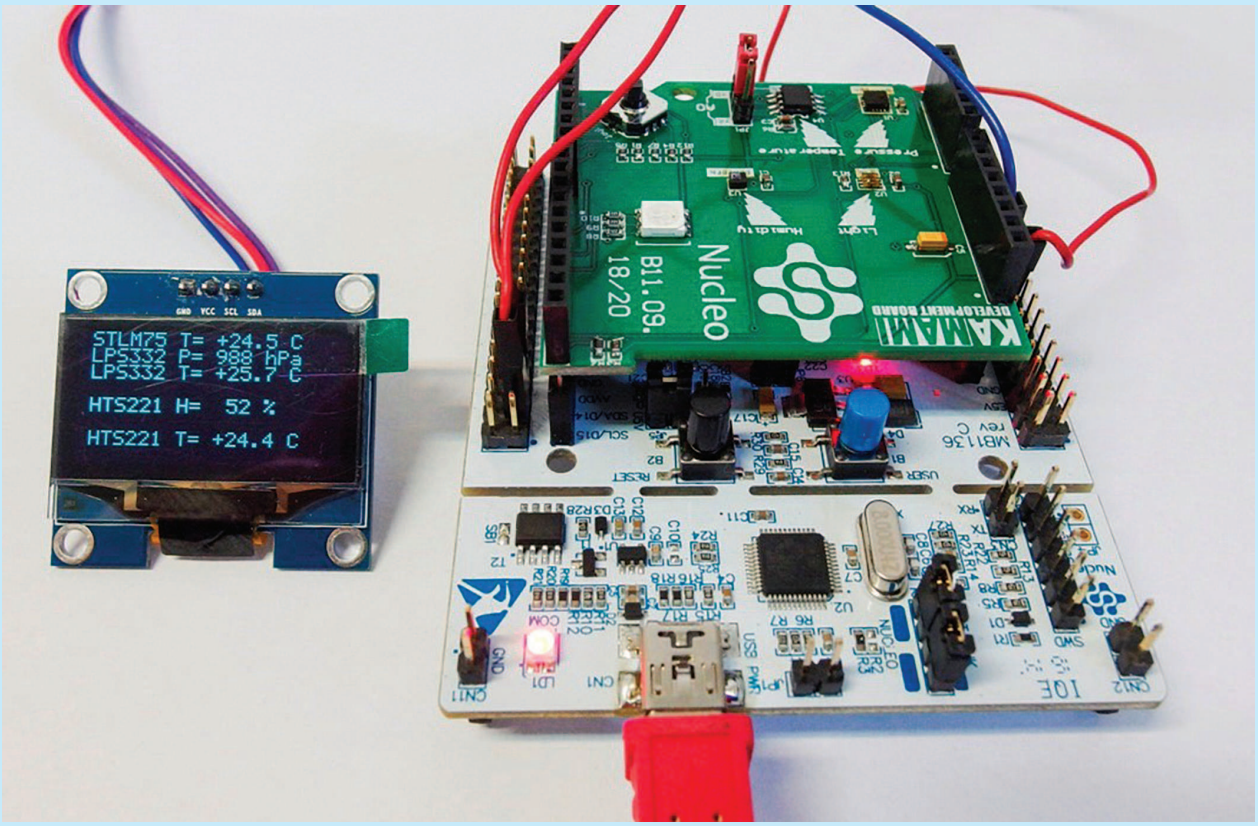


Nucleo – meteo

Użycie płytki Nucleo do budowy termometru

Firma STM od jakiegoś czasu stosuje prostą i skuteczną taktykę promowania swoich mikrokontrolerów 32-bitowych – dostarcza tanie zestawy ewaluacyjne oraz bezpłatne, programowe wsparcie w postaci bibliotek obsługujących układy peryferyjne. Wśród użytkowników wszystkich rodzin mikrokontrolerów STM32 najbardziej znane są zestawy Discovery, od najprostszych – na przykład STM32F0 Discovery – do rozbudowanych – na przykład STM32L1 Discovery lub STM32F4 Discovery. Wszystkie zestawy Discovery są wyposażone w niezależny programator/debuger STLink V2. Programator może współpracować z mikrokontrolerem zestawu Discovery, lub po przełożeniu dwu zworek programować zewnętrzny mikrokontroler. ST Link jest wspierany przez wszystkie liczące się implementacje środowisk projektowych IDE w tym przez środowiska oferowane przez firmę Keil (ARM) i IAR.

Jest cała rzesza entuzjastów elektroniki, którzy potrzebują środowisk tanich i łatwych w użyciu, a jednocześnie elastycznych i dających duże możliwości. Każdy z wielkich producentów mikrokontrolerów próbuje dostarczać takie kompletne środowiska, ale wielką popularność zdobył niezależny system Arduino, który został zaprojektowano jako platforma sprzętowo-programowa opartą o 8-bitowe mikrokontrolery AVR. Środowisko IDE jest bezpłatne i ma wbudowany kompilator języka zbliżonego do C. Fizycznie platforma jest podzielona na płytkę z mikrokontrolerem i płytki rozszerzeń łączonych za pomocą gniazd i wtyków. Duża popularność Arduino spowodowała „wysyp" rozszerzeń i programowych bibliotek obsługujących układy peryferyjne zamontowane na tych płytkach. Ponieważ w założeniach Arduino nie ma konieczności stosowania mikrokontrolerów ATmega, a biblioteki napisane w języku wyższego rzędu można łatwo przekompilować na dowolny mikrokontroler, to wkrótce zaczęły powstawać moduły z mikrokontrolerami innych firm.

Firma STM, pomimo niewątpliwego sukcesu zestawów Discovery, poszła dalej i zaoferowała nowe zestawy Nucleo sprzętowo zgodne ze standardem Arduino R3i, oferowane z różnymi typami mikrokontrolerów STM32 od rodziny F0 (STM32F030, STM32F071), poprzez F1 (STM32F103), F3 (STM32F302, STM32F334) do F4 (STM32F401, STM32F411).

Oprócz typowego złącza standardu Arduino, na płytce umieszczono złącza standardu Morpho zaprojektowanego przez STM. Umieszczenie złącz Morpho powoduje, że jest możliwe dołączenie dużo większej liczby linii mikrokontrolera, niż w wypadku standardowego złącza Arduino R3.

Płytka Nucleo (**fotografia 1**) jest podzielona fizycznie na 2 części: część programatora/ debugera ST-Link V2-1 i część mikrokontrolera ze złączami, przyciskami USER, RESET i 2 diodami LED. Na płytce jest miejsce na oscylator kwarcowy taktujący mikrokontroler, ale oscylator nie jest wlutowany. Domyślnie mikrokontroler jest taktowany wbudowanym w mikrokontroler generatorem RC. Do taktowania RTC zamontowano dodatkowy kwarc zegarkowy o częstotliwości 32,768 kHz.

Wyposażenie płytki w elementy dodatkowe jest stosunkowo ubogie. Wynika to z idei Arduino: bazowy moduł mikrokontrolera ma za zadanie sterować wymiennymi modułami z układami peryferyjnymi dołączanymi poprzez zestandaryzowane złącza.

Ponieważ płytka ma zamontowany stabilizator +3,3 V zasilający mikrokontroler, to producent przewidział możliwość zasilania z 3 źródeł:

- ze złącza USB programatora STLink z ograniczeniem do 100mA,
- z napięcia +5 V podawanego na wyprowadzenie E5V (złącze Morpho CN7),
- z napięcia VIN z zakresu 7…12 V DC (złącze Arduino CN6).

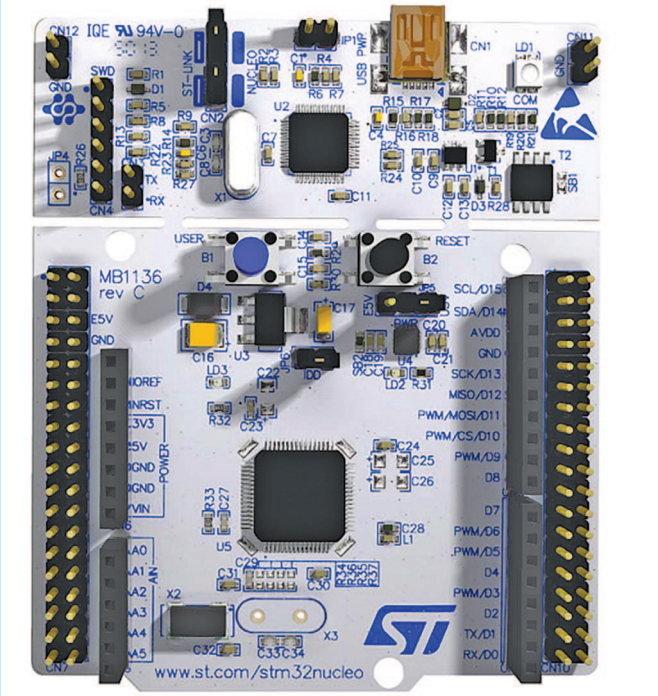
Podobnie jak w zestawach Discovery, STLink może programować zewnętrzne mikrokontrolery po przestawieniu zworek CN2. Sygnały SWD są dostępne na złączu CN4. Płytka zestawu ma podłużne frezowania na granicy programatora, tak by można było programator odłamać od reszty układu i używać go autonomicznie. Takiej możliwości zestawy Discovery nie miały.

Fabrycznie skonfigurowany moduł jest gotowy do pracy z programatorem i zasilaniem poprzez złącze USB. Programator ST Link V2-1 wymaga do prawidłowej pracy zainstalowania dedykowanego drivera dla systemu Windows. Driver należy pobrać z firmowej strony *www.st.com* i zainstalować. Po jego zainstalowaniu można dołączyć Nucleo do portu USB komputera. ST Link V2-1 jest widziany w komputerze jako osobny napęd z zapisanym plikiem strony mbed, czyli wirtualnego IDE z kompilatorem C/C++.

Po połączeniu się z programatorem można wykonać upgrade firmware. Na stronie *http://developer.mbed.org/teams/ST/wiki/Nucleo-Firmware* jest umieszczona najnowsza wersja firmware. Po ściągnięciu pliku trzeba go rozpakować i uruchomić plik exe. Aktualizacja modułu Nucleo dołączonego do komputera jest wykonywana automatycznie. Aktualizowanie firmware jest dobrym zwyczajem, ponieważ używa się najnowszej wersji, która przynajmniej w teorii jest najbardziej stabilna i ma najmniej błędów.

Mbed

Tak przygotowany moduł jest gotowy do pracy. W tym momencie trzeba wybrać środowisko projektowe IDE, w którym będziemy tworzyli i testowali oprogramowanie oraz kompilator języka C lub C++. W modułach Nucleo są montowane mikrokontrolery STM32, a w moim module testowym został zamontowany mikrokontroler



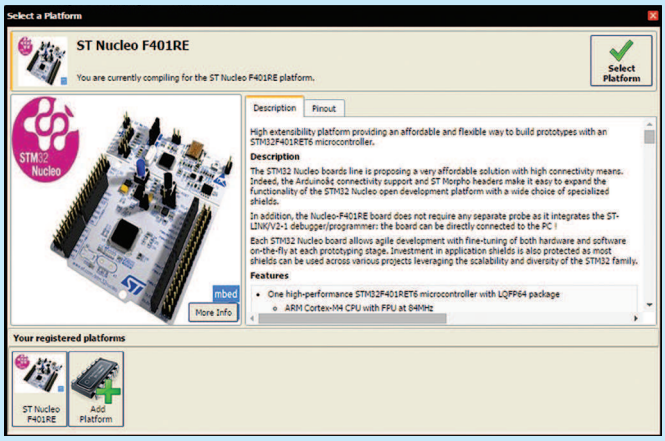
Fotografia 1. Płytka zastawu Nucleo

STM32F401. Do tej pory używałem środowiska uVision4 i pierwsze próby zamierzałem wykonać pisząc proste programy testowe w uVision4. Niestety, pierwsza próba zakończyła się niepowodzeniem. Mój mikrokontroler to jedna z nowszych propozycji STM – STM32F401 nie znalazł się na liście mikrokontrolerów wspieranych przez uVision4. Wsparcie IDE dla określonych typów mikrokontrolera polega na przygotowanych przez producenta definicji rejestrów sterujących, skryptów linkera, plików startup itp. Brak wsparcia powoduje, że trzeba by samodzielnie to uzupełnić, a nie jest to zadanie łatwe i na pewno czasochłonne.

Do celów przeprowadzenia testów konieczne było ściągnięcie i zainstalowanie nowszej wersji uVision5. Ja jednak zdecydowałem się wykorzystać inną możliwość. Programowanie modułów Nucleo jest możliwe w wirtualnym środowisku mbed, które wspiera programowanie wybranych platform projektowych produkowanych przez firmy NXP, Freescale, STM i Nordic.

Wirtualne środowisko wymusza niekonwencjonalne podejście do organizacji projektu i samego programowania. Do tej pory programista kupował komercyjne narzędzia projektowe lub instalował wersje bezpłatne, składające się z IDE, kompilatora i bibliotek. Wszystkie pliki projektu były umieszczone na lokalnym dysku. Idea narzędzi wirtualnych lub inaczej mówiąc – idea programowania w chmurze jest inna. Dostęp do środowiska projektowego jest możliwy z dowolnego komputera z zainstalowaną przeglądarką stron WWW i połączonego z Internetem. Nie trzeba instalować żadnego oprogramowania, a wszystkie pliki projektu są umieszczone gdzieś w przestrzeni wirtualnej i podczas normalnej pracy nie mamy ich kopii na dysku lokalnym.

Jak wspomniałem, plik ze stroną środowiska mbed jest umieszczony w folderze napędu widzianego w komputerze po dołączeniu modułu Nucleo z programatorem ST Link V2-1. Wystarczy kliknąć na ten plik i otworzyć stronę *https://developer.mbed.org/account/login*, na której można utworzyć nowe konto użytkownika lub zalogo-



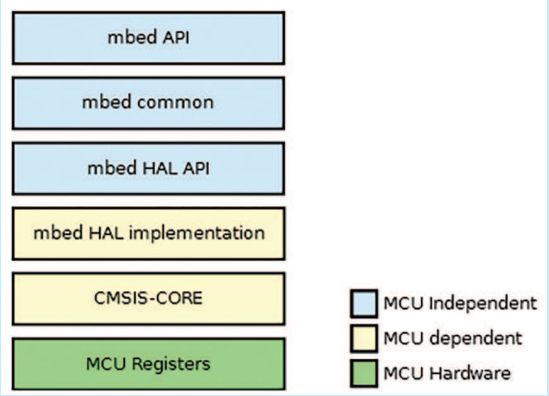
Rysunek 2. Okno wyboru platformy *mbed*

wać się do istniejącego konta. Po zalogowaniu się okno IDE jest otwierane po kliknięciu na przycisk „Compiler" w prawym górnym rogu strony. Jeżeli już wcześniej pracowaliśmy z mbed, to po zalogowaniu otworzą się aktywne projekty w momencie zamykania strony z IDE.

Po otwarciu strony, po pierwszym zalogowaniu trzeba utworzyć własny projekt lub otworzyć projekt przykładowy. Wcześniej wspomniałem, że mbed wspiera wiele platform projektowych różnych producentów. Jeżeli w czasie logowania do portu USB jest dołączony moduł Nucleo, to zostanie automatycznie rozpoznany (w naszym wypadku *ST Nucleo F401 RE*) a jego symbol zostanie wyświetlony w prawym górnym rogu strony IDE. Po kliknięciu na ten symbol otwiera się okno z opisem platformy, możliwością dodawania kolejnych platform i możliwością wyboru aktywnej. Dodatkowo, jest tam umieszczony opis wyprowadzeń modułu Pinout, jak pokazano na **rysunku 2**.

Tworzenie nowego projektu rozpoczynamy od kliknięcia na *New* → *Program*. Otwiera się okno, w którym można wybrać platformę (*Platform*), szablon (*Template*) oraz nazwę projektu (*Program name*). Nowy projekt zostanie umieszczony w oknie *Program Workspace*. Do tak utworzonego projektu można dodawać pliki źródłowe (*New* → *New File*) z rozszerzeniami C, CPP lub H, nowe foldery i biblioteki.

Dodawanie plików źródłowych nie wymaga szerszego komentarza. Zatrzymamy się na chwilę przy dodawaniu bibliotek. Użytkownik może dodawać biblioteki do swojego projektu po kliknięciu prawym przyciskiem myszy na nazwę projektu w oknie *Program Workspace* i wybraniu z listy rozwijanej polecenia *Import Library* → *From Import Wizard*. Zostanie wyświetlone okno wyboru *Import a Library From Import mbed.org*. Na liście jest umieszczona oficjalna biblioteka *mbed* oraz szereg bibliotek utworzonych przez różnych użytkowników. Teraz wybierzemy oficjalną biblioteka mbed i dodamy ją do projektu.



Rysunek 3. Model warstwowy biblioteki *mbed library*

Używanie różnego rodzaju bibliotek dostarczanych bezpłatnie lub kupowanych jest powszechną praktyką programistów. Przykładem są biblioteki CMSIS (*Cortex Microcontroller Software Interface*) stosowane przez użytkowników mikrokontrolerów z rdzeniem Cortex. Biblioteka w znacznym stopniu przyspiesza programowanie układów peryferyjnych. Programista nie musi wgłębiać się dokładnie w strukturę rejestrów konfiguracyjnych, nie musi znać znaczenia poszczególnych bitów tych rejestrów. Oczywiście, każdy producent musi przygotować dla swoich mikrokontrolerów własną wersję biblioteki, ale funkcje biblioteczne są takie same dla Corteksów. Niejako przy okazji w bibliotekach są umieszczane szablony przykładowych projektów dla najbardziej popularnych środowisk IDE. Biblioteki CMSIS mogą zawierać błędy, mogą nie być napisane optymalnie, w wypadku, kiedy coś pójdzie nie tak mogą zatrzymywać cały program, ale na pewno znacząco przyspieszają programowanie. Doświadczony programista na pewno poradzi sobie z drobnymi błędami, a coraz nowsze wersje bibliotek działają coraz lepiej.

Biblioteki związane z pracą układów peryferyjnych są tak tworzone, aby użytkownik miał do dyspozycji funkcje API i nie musiał znać jak są konfigurowane i jak pracują układy peryferyjne. Tak też wykonano bibliotekę *mbed library*. Na **rysunku 3** pokazano model warstwowy tej biblioteki. Kolorem niebieskim oznaczono warstwy niezależne od typu mikrokontrolera (producenta, rodziny), kolorem żółtym warstwy zależne od typu mikrokontrolera, a kolorem zielonym warstwę najniższą, związaną z rejestrami konfiguracyjnymi mikrokontrolera.

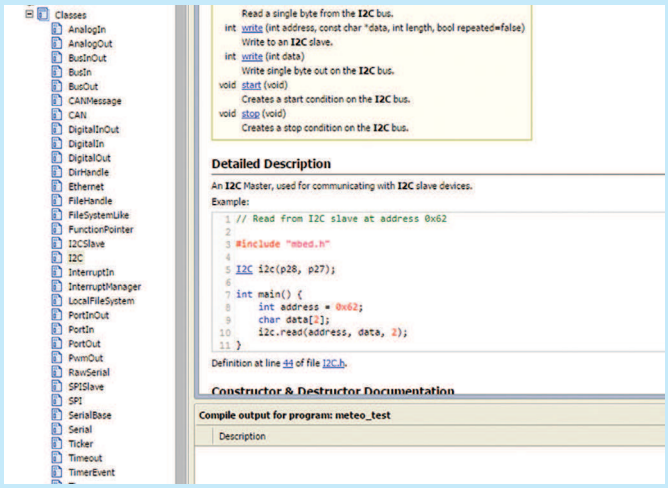
Taka struktura pozwala na tworzenie programów w taki sam sposób, niezależnie od wybranej platformy wspieranej przez *mbed*. A co jeżeli platforma nie jest wpierana?. Wtedy trzeba sobie napisać port (*porting*) do nowego typu mikrokontrolera. W plikach pomocy są zamieszczone wskazówki jak to zrobić, ale jest to zadanie dla bardziej doświadczonych programistów. Dla tych mniej doświadczonych pozostaje nadzieja, że oferta wspieranych modułów będzie się sukcesywnie poszerzać i uda się dobrać coś do własnych potrzeb.

Idea bibliotek mbed pokazuje trend odchodzenia od programowania przez użytkownika układów peryferyjnych. Wystarczy wiedza, że układ jest taki a taki i może wykonać określone funkcje. Resztę wykonają funkcje biblioteki. To kolejny krok w oddalaniu się programistów od struktury mikrokontrolera. Dzisiaj mało kto zna listę rozkazów i potrafi programować w asemblerze. Zapewne za jakiś czas niewielu programistów będzie znało budowę układów peryferyjnych i sposób ich programowania. Wraz ze znacznym zwiększeniem się możliwości mikrokontrolerów ich programowanie coraz bardziej przypomina programowanie w systemie operacyjnym z BIOS-em.

Wróćmy jednak do naszego programu tworzonego w środowisku *mbed*. Po dołączeniu biblioteki *mbed* mamy do dyspozycji szereg gotowych klas z funkcjami obsługi układów peryferyjnych: wejść (A/C) i wyjść (C/A)

analogowych, interfejsów CAN, I²C, SPI, UART, PWM, do manipulowania poziomami na liniach portów, obsługi timera (w tym licznika Tick). Oprócz tego, można używać funkcji związanych z układem przerwań, funkcji systemu plików i funkcji związanych z Ethernetem.

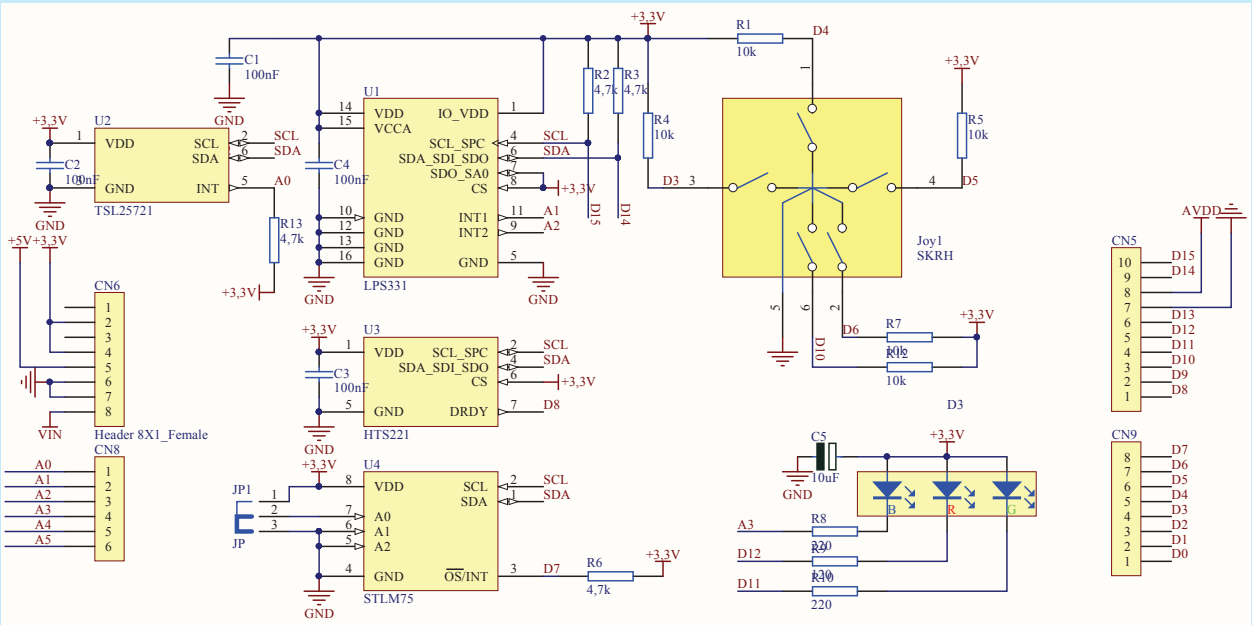
Na **rysunku 4** pokazano okno IDE *mbed* z otwarta zakładką *Classes* biblioteki *mbed* (okno *Program Workspace*) i wybranym interfejsem I²C. W oknie z prawej strony wyświetla się coś w rodzaju pliku pomocy z opisem funkcji klasy i przykładami użycia.



Rysunek 4. Okno z elementami *mbed* i plikiem pomocy

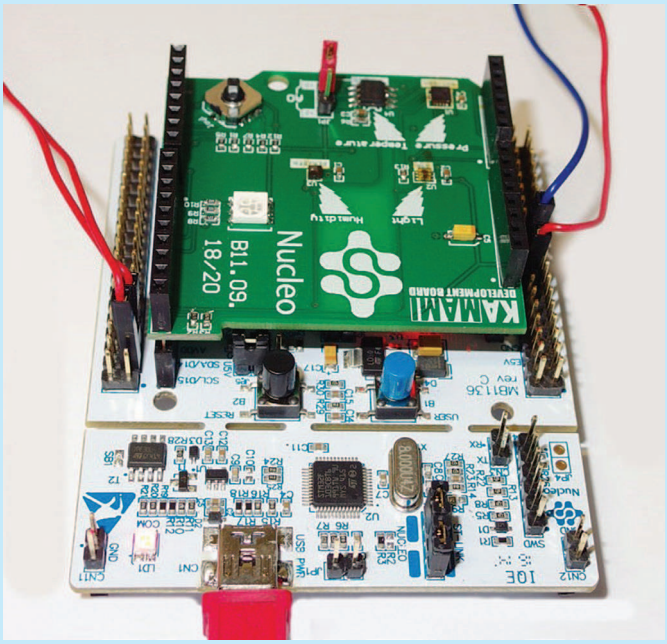
Testy praktyczne

Otrzymałem do dyspozycji moduł ST Nucleo F401RE z mikrokontrolerem STM32F401RET6 w obudowie z 64 wyprowadzeniami, wyposażony w 512 kB pamięci Flash oraz 96 kB pamięci RAM. Mikrokontroler ma bardzo wydajny rdzeń Cortex M4 taktowany z częstotliwością



Rysunek 5. Schemat płytki rozszerzeń (pdf)

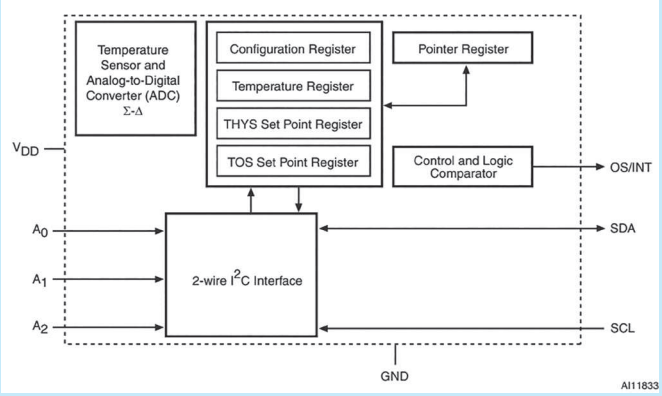
maksymalną 84 MHz. Lista układów peryferyjnych jest również bogata: 12-bitowy przetwornik A/C, interfejsy SPI, I²C, UART, USB (OTG), moduły RTC, CRC. Złącza Morpho umożliwiają dołączenie za pomocą przewodów z nasadkami na goldpiny dowolnego układu peryferyjnego. Wykorzystałem to do przyłączenia już opisywanego na łamach EP wyświetlacza OLED, sterowanego za pomocą magistrali I²C. Drugim, bardzo ważnym elementem testu była płytka zgodna ze standardem Arduino R3, zawierająca: cyfrowy termometr STLM75, cyfrowy ciśnieniomierz LPS331, cyfrowy higrometr HTS221, miernik natężenia światła, 3-kolorową diodę LED i miniaturowy joystick. Wszystkie cyfrowe układy pomiarowe zostały wyposażone w interfejs I²C. Schemat płytki pokazano na **rysunku 5**, a widok płytki rozszerzeń połączonej z modułem Nucleo F401RE na **fotografii 6**.



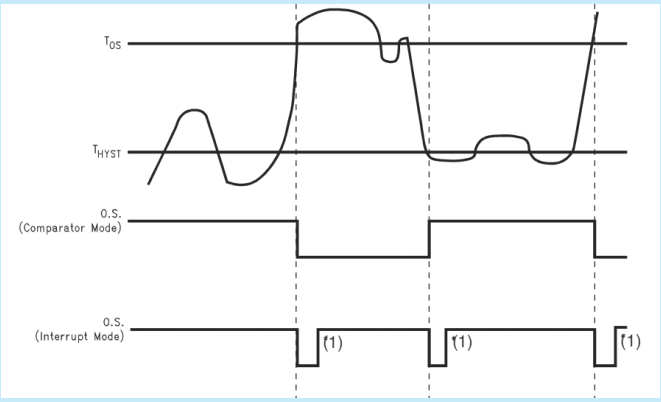
Fotografia 6. Widok połączonych płytek Nucleo i płytki rozszerzenia

Termometr STLM75

Poza pomiarem napięcia, pomiar temperatury jest chyba najczęściej wykonywanym przez różne urządzenia mikrokontrolerowe. Niezwykła popularność tego pomiaru zaowocowała opracowaniem wielu scalonych termometrów z wyjściem cyfrowym. Typowy pomiar po konwersji



Rysunek 7. Schemat blokowy termometru STLM75



Rysunek 8. Tryby pracy wyjścia OS/INT

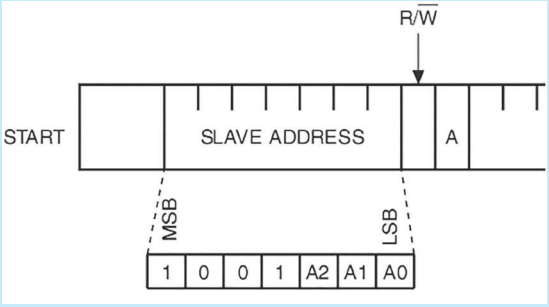
temperatura/napięcie, na przykład mierzenie spadku napięcia na złączu P-N w zależności od temperatury, sprawia kilka problemów technicznych. Trzeba to napięcie wzmocnić, a potem przekonwertować za pomocą przetwornika A/C. Specjalizowany termometr scalony ma wbudowane niezbędne bloki funkcjonalne, a mikrokontroler jedynie odczytuje zmierzoną temperaturę poprzez jakiś interfejs.

STLM75 potrafi mierzyć temperaturę z rozdzielczością 0,5°C w zakresie od -55…+125°C, który to pozwala na pomiary w większości typowych zastosowań. Dokładność pomiaru dla całego zakresu pomiarowego wynosi ±3°C. Może to oznaczać, że mimo iż zmierzyliśmy temperaturę +13,5°C, to w rzeczywistości w najgorszym wypadku może mieć ona wartość +10,5°C lub +16,5°C. Dla temperatury pokojowej taki pomiar nie ma praktycznie sensu. Jednak warto zauważyć, że jest to maksymalna odchyłka dla całego zakresu pomiarowego. Termometr jest tak skalibrowany, że typowa dokładność pomiaru dla temperatury otoczenia nie powinna być gorsza niż ±0,5°C.

Konwersja napięcia proporcjonalnego do mierzonej temperatury jest wykonywana przez 9-bitowy przetwornik

Temperatura	Cyfrowa wartość wyjściowa	
	binarnie	heksadecymalnie
+125C	0 1111 1010	0xFA
+25C	0 0011 0010	0x32
+0,5C	0 0000 0001	0x01
0C	0 0000 0000	0x00
-0,5C	1 1111 1111	0x1FF
-25C	1 1100 1110	0x1CE
-40C	1 1011 0000	0x1B0
-55C	1 1001 0010	0x192

Rysunek 9 zależność pomiędzy temperaturą, a cyfrową wartością wyjściową



Rysunek 10. Adresowanie STLM75

sigma-delta (**rysunek 7**). Wyprowadzenie OS/INT typu otwarty dren (wymaga podciągania do plusa zasilania) jest przeznaczone do sygnalizacji przekroczenia nastawionych progów temperatury. W trybie przerwania (INT) na tym wejściu pojawiają się impulsy w momencie przekroczenia progów, a w trybie termostatu na tym wyjściu zmienia się poziom w zależności od tego czy temperatura jest niższa, czy wyższa niż progowa (**rysunek 8**). Temperatury progu *Tos* i histerezy *Thys* są programowane przez użytkownika przez zapisanie odpowiednich rejestrów. Z wyjściem OS/INT jest powiązany filtr służący do eliminowania zakłóceń. Po przekroczeniu progu jest odliczana zaprogramowana liczba cykli czasowych (od 1 do 6) i po tym czasie ponownie jest sprawdzany warunek przekroczenia progu. Jeżeli próg jest przekroczony, to zmienia się poziom na OS/INT.

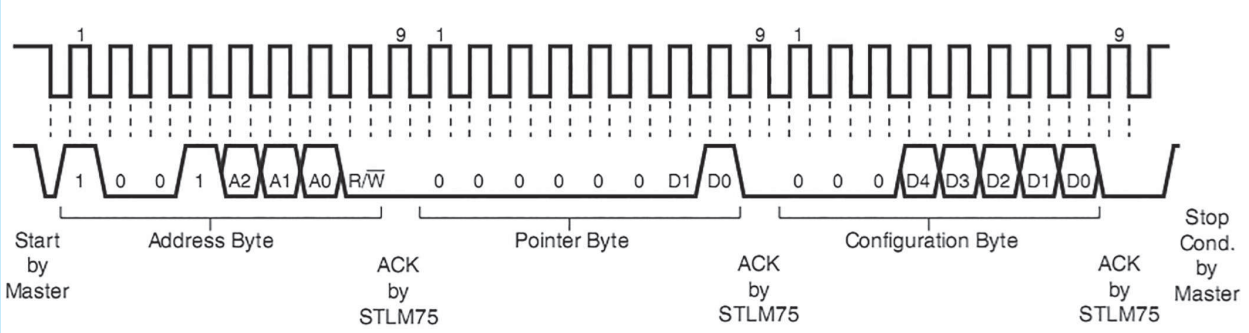
Jak wspomniano, termometr mierzy temperaturę z rozdzielczością 0,5°C w zakresie -55…+125°C. Wynik konwersji jest zapisywany na 9 bitach w kodzie U2. Dla temperatury dodatniej (najstarszy bit wyniku konwersji wyzerowany) 8 najstarszych bitów zawiera część całkowitą zmierzonej temperatury, a najmłodszy bit określa rozdzielczość 0,5°C. Dla temperatury ujemnej (najstarszy bit wyniku konwersji ustawiony) trzeba wykonać konwersję polegającą na zanegowaniu wszystkich bitów i dodaniu 1. Potem wyznaczenie temperatury przebiega tak, jak dla temperatury dodatniej. Na **rysunku 9** pokazano zależność pomiędzy temperaturą i wartością odczytana z rejestru termometru.

Do komunikacji z mikrokontrolerem jest przeznaczony interfejs I²C. Adres *Slave* układu jest określany przez poziomy logiczne na 3 wyprowadzeniach adresowych: A0, A1 i A2. Na **rysunku 10** pokazano sposób adresowania układu. Dla *A0=A1=A2=0* adres dla zapisu danych wynosi 0x90 (R/W=0), a dla odczytu 0x91 (R/W=1).

Zapisanie danych do układu standardowo rozpoczyna się od wysłania sekwencji startu, a potem jest przesyłany adres Slave. Jeżeli adres jest prawidłowy, to termometr potwierdza go w dziewiątym takcie zegara. Po potwierdzeniu adresu można wysyłać do układu dane.

STLM75 ma w swojej strukturze 4 rejestry: 16-bitowy rejestr mierzonej temperatury *TEMP*, 8-bitowy konfiguracji *CONF*, 16-bitowy histerezy *Thys* i 16-bitowy progu temperatury *Tos*. Poza rejestrem odczytywanej temperatury *TEMP*, który można tylko odczytywać, pozostałe rejestry można również zapisywać. Każdy rejestr musi być zaadresowany przed dostępem. Pierwszą ważną sekwencją protokołu komunikacji z STLM75 jest zapisywanie jednego bajta rejestru konfiguracyjnego. Polega ona na:

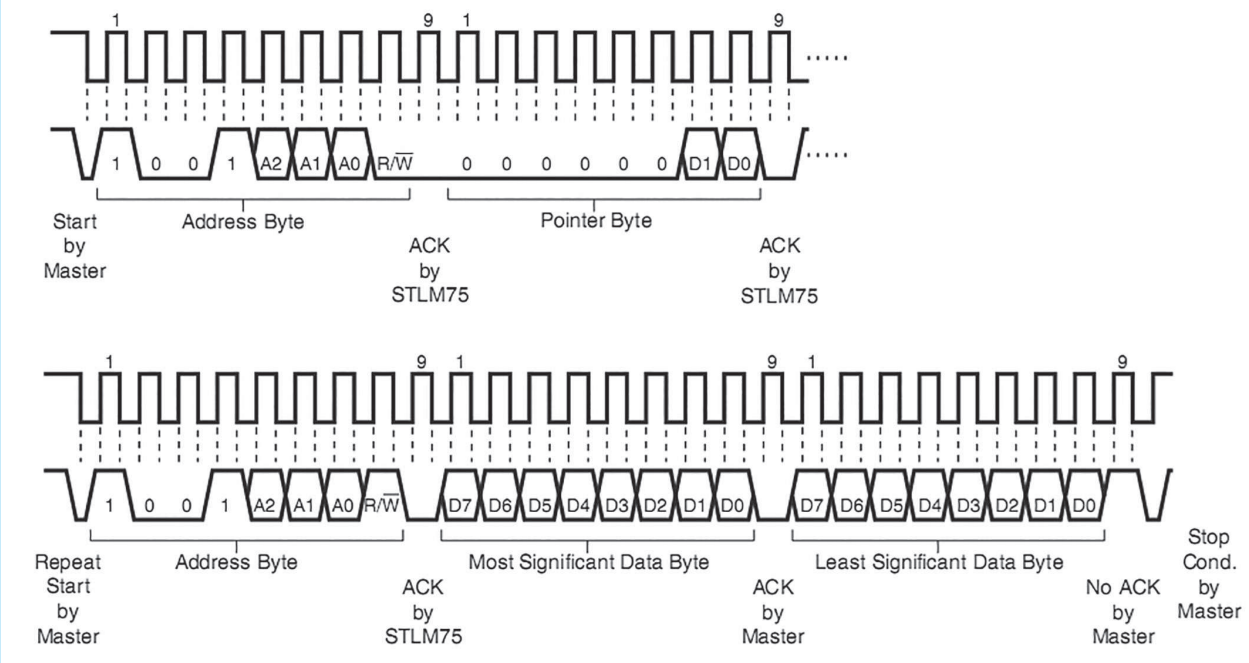
- Wysłaniu sekwencji START.
- Wysłaniu adresu Slave z bitem R/W=0 (i odebraniu potwierdzenia ACK).
- Wysłaniu bajta z adresem rejestru =1.



Rysunek 11. Sekwencja zapisu rejestru konfiguracyjnego

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	0	FT1	FT0	POL	M	SD

SD – Shutdown – ustawienie tego bitu włącza tryb oszczędzania energii
M – Termostat mode M=0 tryb termostatu , M=1 tryb przerwań (Interrupt)
POL – polaryzacja wyjścia OS/INT POL=0 aktywny stan niski, POL=1 aktywny stan wysoki
FT1:FT0 ilość cykli eliminacji zakłóceń.
Rysunek 12. Rejestr konfiguracyjny



Rysunek 13. Sekwencja odczytu rejestru STLM75

- Wysłaniu bajta rejestru konfiguracyjnego.
- Wysłaniu sekwencji STOP.

Zostało to pokazane na **rysunku 11**.

Po włączeniu zasilania wszystkie bity rejestru konfiguracyjnego są domyślnie wyzerowane. Rejestry temperatury *Thys* oraz *Tos* są zapisywane podobnie, jak rejestr konfiguracyjny, tylko po wysłaniu bajta z adresem (równego 2 lub 3) wysyłane są dwa kolejne bajty rejestru 16-bitowego.

Kolejną ważną sekwencją jest odczytywanie dowolnego rejestru polegające na:

- Wysłaniu sekwencji START.
- Wysłaniu adresu Slave z bitem R/W=0 (i odebraniu potwierdzenia ACK).
- Wysłaniu bajta z adresem rejestru (dla odczytania rejestru temperatury wyzerowany).
- Wysłania powtórnej sekwencji START.
- Wysłaniu adresu Slave z bitem R/W=1 (i odebraniu potwierdzenia ACK).
- Odczytaniu dwu bajtów rejestru.
- Wysłaniu sekwencji STOP.

Zostało to pokazane na **rysunku 13**.

Już wiemy, jak wyglądają sekwencje zapisywania i odczytywania danych poprzez I²C. Teraz korzystając ze środowiska *mbed* i jego bibliotek napiszemy i uruchomimy program odczytujący i wyświetlający temperaturę z termometru STLM75.

Przy okazji opisywania w jednym z artykułów obsługi małego wyświetlacza OLED – używanego również tutaj do prezentowania wyników pomiarów pokazywałem jak skonfigurować i używać interfejsu I²C z wykorzystaniem procedur biblioteki CMSIS. Sama konfiguracja jest dość rozbudowana i składa się z konfigurowania linii portów SDA i SCL oraz samego interfejsu I²C. Jeżeli do transferu danych chcemy wykorzystać kanał DMA, to dochodzi też konfigurowanie kanału DMA. Ponadto, aby mikrokontroler pracował poprawnie, trzeba skonfigurować układ zegara taktującego rdzeń i peryferia.

Listing 1. Zapis rejestru konfiguracji
```
#define STLM75_ADD 0x90
I2C i2ct(I2C_SDA, I2C_SCL);
int STLM75Config(char config)
{
    char data_config[2];//bufor danych
    data_config[0] = 0x01;//adres rejestru konfiguracji
    data_config[1] = config;
//przesłanie 2 bajtów z sekwencją STOP
    int status = i2ct.write(STLM75_ADD, data_config, 2, 0);
    return(status);
}
```

Listing 2. Odczyt rejestrów temperatury
```
#define STLM75_ADD 0x90
I2C i2ct(I2C_SDA, I2C_SCL);
int TempRead(char reg, char *data_read)
{
    int err;
    char data_write[2];//bufor danych do zapisu
    data_write[0] =reg;//rejestr temperatury=0
//wysłanie adresu rejestru  bez sekwencji STOP
    err=i2ct.write(STLM75_ADD, data_write, 1, 1);
    if(err!=0) return(err); //powrót z błedem
    err= i2ct.read(STLM75_ADD, data_read, 2, 0); //
rejestr temperatury
    return(err);
}
```

Rysunek 15. Wyświetlanie temperatury mierzonej przez STLM75

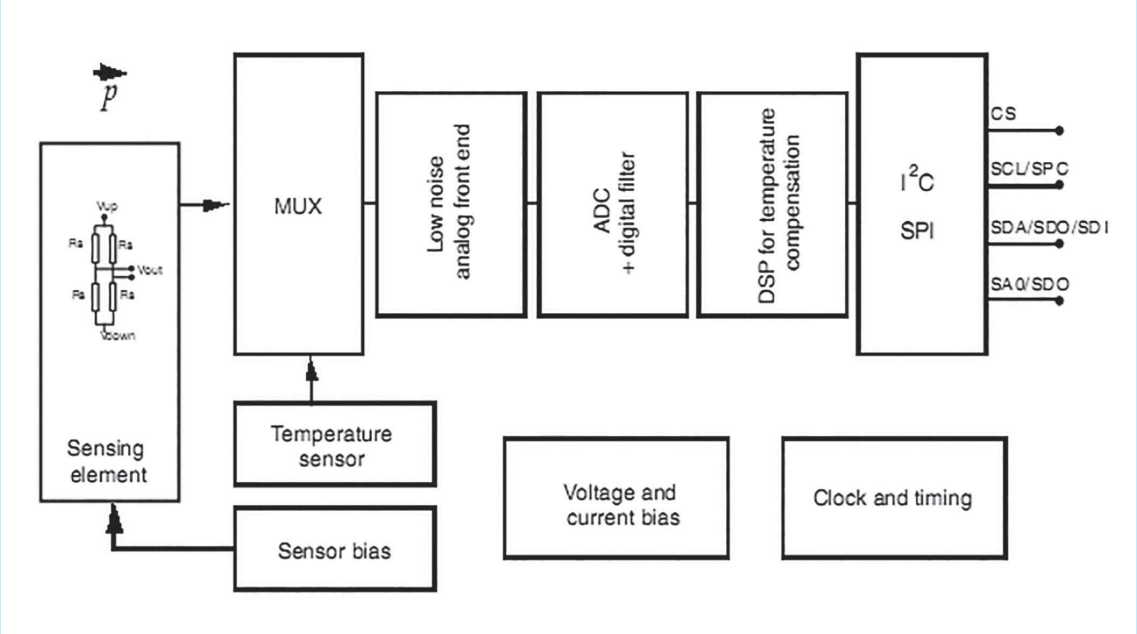
Moduł I²C mikrokontrolerów STM32 może pracować w trybie poolingu, przerwań lub DMA. To wszystko powoduje, że skonfigurowanie interfejsu, potem napisanie obsługi nie jest zadaniem banalnym i wymaga sporo pracy. Zobaczymy teraz jak to wygląda w przypadku *mbed*. W pierwszym kroku trzeba skonfigurować interfejs. Do tego celu jest przeznaczona klasa I2C (*PinName SDA*, *PinName SCL*). Cała definicja interfejsu sprowadza na przykład się do *I2C i2ct(I2C_SDA, I2C_SCL);*. Podajemy tylko definicje linii portów, do których będą doprowadzone sygnały SDA i SCL. W tym wypadku nie musimy znać, które linie mikrokontrolera to będą, ponieważ w *mbed* są zdefiniowane stałe I2C_SDA i I2C_SCL w taki sposób, aby linie interfejsu „trafiały" do pinów złącza Arduino R3 opisanych SDA/D14 i SCL/D15. Są to standardowe wyprowadzenia linii interfejsu I²C dla rozszerzeń Arduino, w tym również dla naszego modułu z termometrem, ciśnieniomierzem i higrometrem.

Tylko tyle? Właśnie, jak się ma w pamięci konfigurowanie interfejsu z CMSIS, to wydaje się to niewiarygodnie łatwe. I prawda, jest łatwe, ale tylko dla modułu Nucleo i pozostałych modułów wspieranych przez *mbed*. Po prostu, cała konfiguracja jest ładowana na podstawie informacji o dołączonym module. „Czarną robotę" wykonali za nas programiści piszący porty dla wspieranych modułów. Ma to swoje zalety, ale tak naprawdę nie wiemy jak pracuje interfejs czy wykorzystuje sprzętowy interfejs I²C, czy jest realizowany programowo. W niektórych zastosowaniach może to mieć znaczenie. Oczywiście, nie musimy korzystać z biblioteki *mbed* i możemy napisać własną obsługę korzystając na przykład z biblioteki CMSIS. Kiedy się na tym zastanawiałem doszedłem jednak do wniosku, że skorzystam z bibliotek mbed i zobaczę, jak szybko i co ważne – czy z dobrym

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
TD8	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0	0	0	0	0	0	0	0

TD8 – znak (ujemy/dodatni, TD0 – rozdzielczość 0,5˚C
Rysunek 14. Rejestr temperatury

Listing 3. Odczytanie i wyświetlenie temperatury
```
//inicjalizacja bufora wyświetlania
char TempCelsiusDisplay[] = „+abc.d C";
void STLMTempRead(char x, char y)
{
    char data_write[5];
    char data_read[5];
    data_write[0] =0;//rejestr temperatury
//odczytanie rejestru temperatury
    i2ct.write(STLM75_ADD, data_write, 1, 1); // bez sekwencji STOP
    i2ct.read(STLM75_ADD, data_read, 2, 0);
// w zmiennej tempval dane tak jak na rysunku 14
    int tempval = (int)((int)data_read[0] << 8) | data_read[1];
//wyliczenie temperatury w stopniach Celsiusza
    tempval >>= 7;//przesuń o 7 pozycji w prawo
    if (tempval <= 256)
    { //określenie znaku temperatury
        Display[0] = ‚+';
    } else {
        Display[0] = '-';
        tempval = 512 - tempval;
    }
// wartość po przecinku (0,5stC)
    if (tempval & 0x01)
    {
        Display[5] = 0x05 + 0x30;//konwersja do ASCII
    } else {
        sDisplay[5] = 0x00 + 0x30;
    }
// wyliczenie części całkowitej
    tempval >>= 1;
    Display[1] = (tempval / 100) + 0x30;
    if(Display[1]==0x30)//nie wyświetlamy zera z przodu wyniku
    Display[1]=' ';
    Display[2] = ((tempval % 100) / 10) + 0x30;
    Display[3] = ((tempval % 100) % 10) + 0x30;
    OledTxtRam(Display,x,y);//wyświetlenie wyniku na ekranie wyświetlacza
}
```



Rysunek 16. Schemat blokowy układu LPS331

skutkiem da się wykorzystać wsparcie dla modułów Nucleo.

Po skonfigurowaniu interfejsu trzeba napisać funkcje realizujące sekwencje zapisywania i odczytywania danych za pomocą I²C. Tu czeka nas kolejna miła niespodzianka, czyli funkcje *write* i *read* klasy *I2C*.

Funkcja *write* ma kilka argumentów:

int address – adres slave zapisywanego układu.

*char *data* – wskaźnik na początek bufora z danymi do wysłania.

int length – liczba bajtów wysyłanych danych.

bool repeated – kończenie transferu sekwencją STOP.

Aby przesłać dane funkcją *write* trzeba najpierw zapisać bufor danymi do wysłania i ustalić adres slave odbiorcy. Funkcja po wywołaniu automatycznie wyśle sekwencję START i adres slave z bitem R/W=0, następnie zadaną liczbę bajtów danych z bufora i jeżeli argument *repeated* jest równy 1, to zakończy sekwencję zapisu. Jeżeli *repeated* jest równy 0, to na końcu wysyła sekwencję STOP. Zapis rejestru konfiguracji pokazany na **rysunku 11** może być wykonany przez procedurę z **listingu 1**.

Funkcja write zwraca 0, jeśli zapisanie danych się powiodło. W wypadku wystąpienia problemów, na przykład po podaniu błędnego adresu, błędu w połączeniach i innych, funkcja zwraca niezerowy kod błędu. Funkcja read ma podobne argumenty:

int address – adres slave zapisywanego układu.

*char *data* – wskaźnik na początek bufora z odbieranymi danymi.

int length – liczba odbieranych bajtów danych,

bool repeated – kończenie transferu sekwencją STOP.

Funkcja również zwraca 0, jeśli odczytywanie się powiedzie.

Rejestr konfiguracyjny pokazano na **rysunku 12**. Sekwencja odczytu zawartości rejestru jest podzielona na 2 części: najpierw jest zapisywany adres odczytywanego rejestru, a następnie są odczytywane 2 bajty (**rysunek 13**). Na **listingu 2** pokazano kompletną funkcję odczytywania rejestru temperatury. Jak już wiemy, temperatura jest zapisana na 9 bitach, a odczytane dane są zapisane na dwóch bajtach. Ułożenie 9 bitów w 16-bitowym rejestrze (dosunięte do lewej) pokazano na **rysunku 14**. Program, który będzie odczytywał i wyświetlał temperaturę musi:

- Przesunąć zawartość rejestru o 7 pozycji w prawo.
- Określić czy bit TD8 jest jedynką (temperatura ujemna), czy zerem (temperatura dodatnia). Dla temperatury ujemnej trzeba wykonać konwersję.
- Określić i zapamiętać poziom najmłodszego bitu i na jego podstawie wyświetlić po przecinku 0 lub 5.
- Przesunąć wynik o jedno miejsce w prawo.
- Przekonwertować wynik na postać znaków ASCII gotowych do wyświetlenia.

Na **listingu 3** pokazano funkcję służącą do odczytu i wyświetlenia temperatury na wyświetlaczu OLED, również komunikującym się z modułem Nucelo przez interfejs I²C, a na **rysunku 15** wynik działania tej funkcji.

Cyfrowy barometr LPS331

Pomiar ciśnienia atmosferycznego jest wykonywany na potrzeby meteorologii i do pomiaru wysokości względem ustalonego poziomu (na przykład poziomu lotniska). W odróżnieniu od pomiaru temperatury, pomiar ciśnienia wymaga stosowania mechanicznego przetwornika ciśnienie/przesunięcie. Taki przetwornik jest wyposażony w membranę, której ugięcie jest proporcjonalne do wartości ciśnienia. Przesunięcie jest mierzone przez zmianę rezystancji mostka R i proporcjonalną do niej zmianę napięcia. Sygnał napięciowy jest albo po wzmocnieniu wyprowadzany jako sygnał wyjściowy przetwornika, albo konwertowany na postać cyfrową przez wbudowany przetwornik A/C. Po konwersji analogowo-cyfrowej można wartość cyfrową przesłać do mikrokontrolera poprzez magistralę szeregową.

Układ LPS331 jest czujnikiem ciśnienia atmosferycznego mierzącym w zakresie 260…1260 mbar (hPa). Czujnikiem ciśnienia jest miniaturowa membrana wykonana w technologii MEMS z wykorzystaniem oryginalnego procesu VENSENS opracowanego w firmie ST. Ugięcie membrany powoduje niezrównoważenie rezystancyjnego mostka

Master	START	Adres Slave		Adres rejestru		Zapisywane dane		STOP
Slave			ACK		ACK		ACK	

Rysunek 17 Sekwencja zapisu rejestru

Listing 4. Zapisanie rejestru LPS331

```
#define LPS_ADD 0xBA
I2C i2cb(I2C_SDA, I2C_SCL);//I2C dla barometru LPS331
int LPS331WriteReg(unsigned char addr, char data)
{
    char data_config[2];
    data_config[0] = addr;//adres rejestru
    data_config[1] = data;//wpisywana dana
//zapisanie rejestru
    int status = i2cb.write(LPS_ADD, data_config, 2, 0);
    return(status);
}
```

Wheatstonea. Napięcie z wyjścia mostka jest wzmacniane w stopniu niskoszumnego wzmacniacza, potem konwertowane przez przetwornik A/C o rozdzielczości 24 bitów. Na wyjściu przetwornika umieszczono dolnoprzepustowy filtr antyaliasingowy. Ostatnim stopniem w torze obróbki sygnału jest blok DSP wykonujący kompensację temperaturową mierzonego ciśnienia (**rysunek 16**).

Do kompensowania gęstości powietrza układ ma wbudowany sensor temperatury. Komunikacja mikrokontrolera z barometrem może być realizowana przez interfejs I²C lub SPI. Na płytce Nucleo pozostawiono możliwość korzystania tylko z interfejsu I²C. Oprócz linii SDA i SCL, układ ma dwa w pełni programowane linie źródeł przerwania INT1 i INT2. Każda z linii może

Master	START	Adres Slave R/W=0		Adres rejestru		START	Adres Slave R/W=1			NACK	STOP
Slave			ACK		ACK			ACK	Odczytane dane		

Rysunek 18 Sekwencja odczytywania rejestru

Rejestr	ADRES hex	POR	Funkcja
REF_P_XL	08	0	Najmłodsza cześć rejestru ciśnienia referencyjnego
REF_P_L	09	0	Środkowa cześć rejestru ciśnienia referencyjnego
REF_P_XL	0A	0	Najstarsza cześć rejestru ciśnienia referencyjnego
WHO_AM_I	0F	0xBB	ID układu
RES_CONF	10	0x7A	Rejestr konfiguracji rozdzielczości
CTRL_REG1	20	0	Rejestr kontrolny 1
CTRL_REG2	21	0	Rejestr kontrolny 2
CTRL_REG3	22	0	Rejestr kontrolny 3
INT_CFG_REG	23	0	Rejestr konfiguracyjny układu przerwań
INT_SOURCE_REG	24	0	Rejestr konfiguracji źródła przerwań
THIS_P_LOW_REG	25	0	Młodsza część rejestru progu wyzwalania
THIS_P_HIGH_REG	26	0	Starsza część rejestru progu wyzwalania
STATUS_REG	27	0	Rejestr statusu
PRESS_POUT_P_XL	28	0	Najmłodsza cześć rejestru mierzonego ciśnienia
PRESS_OUT_L	29	0	Środkowa cześć rejestru mierzonego ciśnienia
PRESS_OUT_H	2A	0	Najstarsza cześć rejestru mierzonego ciśnienia
TEMP_OUT_L	2B	0	Najmłodsza cześć rejestru mierzonej temperatury
PRESS_OUT_H	2C	0	Starsza cześć rejestru mierzonej temperatury

Rysunek 19. Wykaz rejestrów LPS331

B7	B6	B5	B4	B3	B2	B1	B0
RFU	AVGT2	AVGT1	AVGT0	AVGP3	AVGP2	AVGP1	AVGP0

AVGT2:AVGT0	Tavg	AVGP3:AVGP0	Pavg
0	1	0	1
1	2	1	2
2	4	2	4
3	8	3	8
4	16	4	16
5	32	5	32
6	64	6	64
7	128	7	128
9		8	256
A		384	
		512	

Rysunek 20. Rejestr RES_CONF

sygnalizować ciśnienie wyższe od progu górnego, ciśnienie niższe od progu dolnego, ciśnienie poza ustawionymi progami (wyższe od progu górnego lub ciśnienie niższe od progu dolnego) oraz gotowość danych do odczytu.

Interfejs I²C układu LPS331

LPS331 jest na magistrali I²C układem slave z możliwością ustawienia 2 adresów zależnie od poziomu logicznego na wejściu SA0. Dla SA0=0 adres jest równy 0xB8 dla zapisywania danych i 0xB9 dla odczytywania, a dla SA0=1 adres jest równy 0xBA dla zapisywania danych i 0xBB dla odczytywania. Na płytce Nucleo SA0 jest dołączone do źródła zasilania +3,3 V i adresy są równe, odpowiednio: 0xBA i 0xBB. Układ jest konfigurowany za przez zapisywanie zestawu rejestrów. Odczytywane dane (ciśnienie, temperatura, rejestr statusu) są również umieszczone w rejestrach wewnętrznych. Żeby zapewnić sobie dostęp do rejestrów trzeba napisać dwie funkcje: zapisania rejestru o określonym adresie i odczytania rejestru o określonym adresie.

Sensor LPS331 ma wbudowany mechanizm autoinkrementacji adresu po odczytaniu rejestru. Żeby go uaktywnić trzeba ustawić najstarszy bitu adresu. W przedstawionych procedurach obsługi barometru, ze względu na czytelność przykładów, nie jest to wykorzystywane i każdy rejestr jest adresowany przed odczytaniem.

Sekwencja zapisu danej do rejestru rozpoczyna się od wysłania sekwencji START. Po niej jest wysyłany adres slave z bitem R/W=0 (0xBA). Po potwierdzeniu adresu przez LPS331 można wysyłać bajt adresu rejestru (*subaddres*) i bajt z zawartością zapisywanego rejestru (**rysunek 17**). Funkcję *LPS331WriteReg (unsigned char addr, char data)* służącą do zapisania pojedynczego rejestru pokazano na **listingu 4**.

Odczytywanie zawartości rejestru przebiega w dwóch etapach. Najpierw Master wysyła sekwencję START, adres slave z bitem R/W=0 (0xBA) i bajt adresu rejestru. W drugim etapie jest wysyłana powtórnie sekwencja START, adres slave z bitem R/W=1 (0xBB) i jest odczytywany jeden bajt zawartości zaadresowanego rejestru. Po odczytaniu danych Master nie wysyła potwierdzenia i wysyła sekwencję STOP kończącą transfer

Listing 5. Odczytanie rejestru LPS331
```
char LPS331ReadReg(char addr)
{
    char reg[2];
    char data[2];
    reg[0]=addr;
    i2cb.write(LPS_ADD,reg, 1, 1); //
zapisanie adresu rejestru
    i2cb.read(LPS_ADD, data, 1, 0);//
odczytanie zawartości rejestru
    return(data[0]);
}
```

B7	B6	B5	B4	B3	B2	B1	B0
PD	ODR2	ODR1	ODR0	DIFFEN	BDU	DELTA_EN	SIM

PD Power Down PD=0 power down, PD=1 tryb aktywny
ODR1:ODR1 – częstotliwość odczytywania danych wyjściowych
DIFF_EN odblokowanie układu przerwań DIFF_EN=0 nie aktywny DIFF_EN=1 aktywny
BDU – odświeżanie rejestru wyjściowego BDU=0 dane odświeżane ciągle, BDU=1 dane odświeżane po odczycie rejestrów ciśnienia i temperatury
DELTA_EN – odblokowanie rejestru delta pressure
SIM – konfiguracja interfejsu SPI (dla I²C bez znaczenia)

ODR2	ODR1	ODR0	Odświeżanie danych wyjściowych ciśnienia	Odświeżanie danych wyjściowych temperatury
0	0	0	Pomiar pojedynczy (One shot)	Pomiar pojedynczy (One shot)
0	0	1	1Hz	1Hz
0	1	0	7Hz	1Hz
0	1	1	12,5Hz	1Hz
1	0	0	25Hz	1Hz
1	0	1	7Hz	7Hz
1	1	0	12,5Hz	12,5Hz
1	1	1	25Hz	25Hz

Rysunek 21. Rejestr CTRL_REG1.

(**rysunek 18**). Funkcję odczytującą zwartość rejestru pokazano na **listingu 5**.

Rejestry barometru można podzielić na 2 grupy: rejestry konfiguracyjne i rejestry z mierzonymi wartościami ciśnienia i temperatury. Wykaz wszystkich rejestrów pokazano na **rysunku 19**. Barometr przed wykonywaniem pomiarów powinien być zainicjalizowany. W pierwszym kroku tej inicjalizacji możemy zmienić domyślne ustawienia precyzji pomiarów. Do tego celu jest przeznaczony rejestr *RES_CONF* o adresie 0x10 (**rysunek 20**). Precyzja pomiaru jest tym lepsza, im większa jest wartość *Tavg* dla pomiaru temperatury i *Pavg* dla pomiaru ciśnienia. Ostatnia wartość *Pavg=512* (*RES_CONF – 0x7A*) nie może być użyta w trybie ciągłego odczytu z częstotliwością 25 Hz (25 odczytów na sekundę – bity *ODR2:ODR0* rejestru *CTRL_REG1*). Wtedy maksymalna wartość *RES_CONF* jest równa 0x79.

Pomiar ciśnienia i temperatury może być wykonywany na żądanie (*One Shot*) lub w sposób ciągły. Tego wyboru dokonuje się przez zapisanie rejestru *CTRL_REG1* o adresie 0x20 (**rysunek 21**). Dla nas istotne są bity: *PD*, *ODR2…ODR0*, i *BDU*. Bit *PD* jest przeznaczony do wprowadzania układu w stan obniżonego poboru mocy i powrotu do stanu aktywnego. W trybie obniżonego poboru mocy powinien być modyfikowany rejestr *RES_CONF*. W czasie testów sprawdzałem działanie trybów pojedynczego wyzwalania pomiarów i ciągłego wyzwalania pomiarów. Tryb pojedynczego wyzwalania ustawia się wpisując do *ODR2:ODR0* wartość 0. Wyzwolenie pomiaru następuje po zapisaniu do rejestru *CTRL_REG2* o adresie 0x21 wartości 0x01. Po wykonaniu pomiaru i zapisaniu wyniku do rejestrów ciśnienia i temperatury *CTRL_REG2* jest automatycznie zerowany. Procedurę wyzwalania pomiaru pokazano na **listingu 6**. Jej argumentami są dwie współrzędne wyświetlania wyniku na ekranie OLED. Żeby ten tryb wprowadzić trzeba wywołać procedurę inicjalizacyjną (**listing 7**). Ciągłe odczytywanie pomiarów wymaga zmiany wartości bitów ODR2:ODR0 według zasady pokazanej na **rysunku 21**. Na **listingu 8** pokazano procedurę inicjalizującą ciągły tryb pracy z częstotliwością odświeżania pomiarów z częstotliwością 1 Hz. Po ustawieniu bitu BDU mamy pewność, że przy ciągłym odczycie ciśnienia i temperatury wszystkie bajty wyniku pomiaru zawierają wartości z ostatniego pomiaru. Gotowość danych do odczytu można testować odczytując rejestr statusowy *STATUS_REG* o adresie 0x27. Ustawienie bitu b0 (*T_DA*) oznacza dostępność ostatniego pomiaru temperatury, a ustawienie bitu b1 (*P_DA*) oznacza dostępność ostatniego pomiaru ciśnienia. Odczytywanie statusu wykonuje procedura *LPS331CheckStatus()* zamieszczona na **listin-**

Rysunek 22. Wyświetlanie temperatury i ciśnienia mierzonego przez czujniki STLM75 i LPS331

Listing 6. Wyzwolenie pomiaru One Shot i wyświetlenie odczytanego ciśnienia i temperatury
```
void LPS331OneShot(char x, char y)
{
    LPS331WriteReg(0x21,1);//One Shot start
    while(LPS331ReadReg(0x21)!=0);//czekaj na zakończenie pomiaru
//alternatywnie   while(LPS331ReadReg(0x27)==0);
    LPS331PressRead(x, y);//odczyt i wyświetlenie ciśnienia
    LPS331TempRead(x, y+1);//odczyt i wyświetlenie temperatury
}
```

Listing 7. Inicjalizacja trybu One Shot
```
void LPS331OneShotInit(void)
{
//CTRL_REG1 Power Down
    LPS331WriteReg(0x20,0 );
//RES_CONF 512 próbek do średniej ciśnienia i 128 próbek
//do średniej temperatury
    LPS331WriteReg(0x10, 0x7a);
//CTRL_REG1 Active mode, One Shot mode, Update after read
    LPS331WriteReg(0x20, 0x84);
}
```

gu 9. Na **listingu 10** pokazano fragment programu testujący bity statusu i odczytujący ciśnienie i temperaturę.

Ponieważ do konwersji ciśnienia na napięcie zastosowano 24-bitowy przetwornik A/C, to rejestr wyniku ma długość 24 bitów i jest zapisany w trzech rejestrach 8-bitowych: *PRESS_POUT_P_XL* (najmłodszy bajt pomiaru), *PRESS_OUT_L* i *PRESS_OUT_H* (najstarszy bajt wyniku). Po odczytaniu tych trzech rejestrów trzeba z nich złożyć liczbę 24-bitową zapisaną w 32-bitowym rejestrze typu *int*. Żeby otrzymać wynik w milibarach (hektopaskalach) musimy tę 24-bitową wartość podzielić przez 4096. Na **listingu 11** zamieszczono procedurę odczytującą 3 bajty ciśnienia, wykonującą konwersję wyniku na milibary i wyświetlającą go na wyświetlaczu OLED. Konwersja wyniku na postać znaków ASCII jest wykonywana przez standardową funkcję kompilatora *sprintf*.

Podobnie wygląda odczytanie temperatury. Dwa bajty wyniku należy złożyć w jedną liczbę 16-bitową ze znakiem. Ważna jest długość zmiennej – nie można użyć tu 32-bitowej zmiennej *int*, bo w obliczeniach będzie zawsze interpretowana jako liczba dodatnia, a temperatura odczytywana z barometru jest liczbą ze znakiem. Dlatego musi to być zmienna typu *short*.

Po złożeniu 16-bitowej zmiennej musimy ją przekształcić na liczbę zmiennoprzecinkową, bo będziemy wykonywać obliczenia zmiennoprzecinkowe w celu przekonwertowania wyniku na stopnie Celsjusza. Konwersja polega na podzieleniu odczytanej wartości przez 480 i dodaniu do wyniku wartości 42,5. Widać tu, że dla temperatur pokojowych wartość odczytana z barometru jest ujemna. Do przekształcenia wyniku z postaci zmiennoprzecinkowej jest zastosowana również standardowa funkcja *sprintf* formatowanego wyprowadzania znaków do bufora w pamięci RAM.

Na **rysunku 22** pokazano ekran wyświetlacza OLED z wyświetlaniem: temperatury z czujnika STLM75 oraz ciśnienia i temperatury z barometru LPS331. Temperatura odczytywana z barometru jest o 1,2°C wyższa od tej odczytywanej z barometru. Najprawdopodobniej jest to wynik rozbieżności w dokładności pomiaru obu czujników.

Trzeba pamiętać, że odczytane z barometru ciśnienie jest ciśnieniem rzeczywistym, to znaczy takim, jakie panuje w miejscu pomiaru. To ciśnienie będzie się różniło bardziej lub mniej od ciśnienia atmosferycznego podawanego w serwisach pogodowych. Ciśnienie rzeczywiste zależy od wysokości miejsca pomiaru liczonego od poziomu morza. Im wyżej mieszkamy, tym rzeczywiste mierzone ciśnienie będzie niższe. Jest to logiczne następstwo mniejszego słupa powietrza wywierającego nacisk na membranę barometru przy zwiększaniu wysokości pomiaru. Żeby uniezależ-

Listing 8. Odczyt ciągły z częstotliwością 1 pomiar na sekundę

```
void LPS331ContInit(void)
{
//CTRL_REG1 Power Down
    LPS331WriteReg(0x20,0 );
//RES_CONF 512 próbek do średniej ciśnienia i 128
//próbek do średniej temperatury
    LPS331WriteReg(0x10, 0x7a);
//CTRL_REG1 Active mode, Cont mode, Update after read
    LPS331WriteReg(0x20, 0x94);
}
```

Listing 9. Procedura odczytywania rejestru statusowego

```
if(LPS331CheckStatus()&2==2) LPS331PressRead(11,2);
if(LPS331CheckStatus()&1==1) LPS331TempRead(11,4);
```

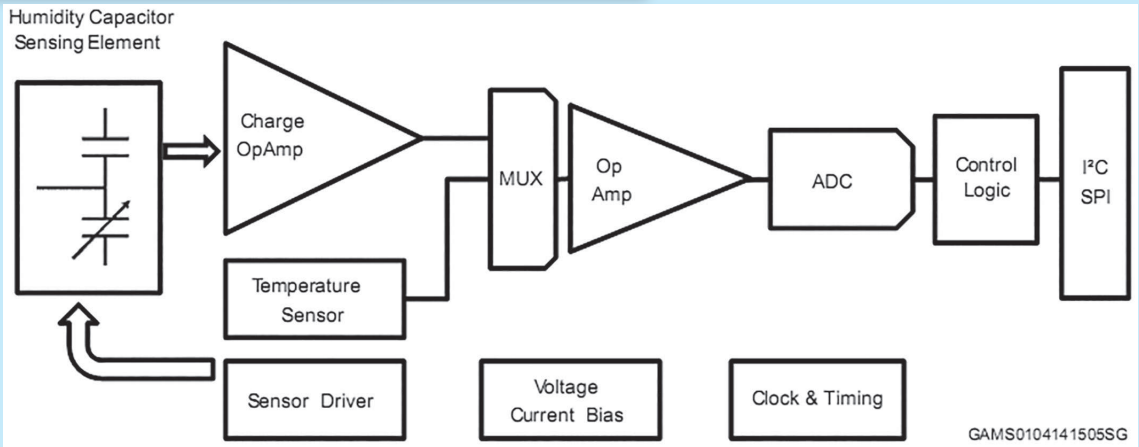
Listing 10. Fragment programu testowania statusu, odczytywania ciśnienia i temperatury

```
char LPS331CheckStatus(void)
{
    return(LPS331ReadReg(0x27));

}
```

Listing 11. Odczytanie i wyświetlenie ciśnienia

```
void LPS331PressRead(char x, char y)
{
    int  temp=0;
    char data_press[3];//bufor danych ciśnienia
    char str[20];//bufor łańcucha tekstowego
    data_press[0]=LPS331ReadReg(0x28);//rejestr  PRESS_POUT_P_XL
    data_press[1]=LPS331ReadReg(0x29);//rejestr  PRESS_OUT_L
    data_press[2]=LPS331ReadReg(0x2a);//rejestr  PRESS_OUT_H
//składanie liczby 24 bitowej
    temp= temp+data_press[2];
    temp=temp<<8;
    temp=temp+data_press[1];
    temp=temp<<8;
    temp=temp+data_press[0];
//konwersja - wynik w mbar
    temp=temp/4096;
//konwersja na postac znaków ASCII
    sprintf(str,"%u hPa",temp);
//wyświetlenie na wyświetlaczu OLED
    OledTxtRam(str,x,y);
}
```



Rysunek 23. Schemat blokowy czujnika HTS221

Master	START	Adres Slave		Adres rejestru		zapisywane dane		STOP
Slave			ACK		ACK		ACK	

Rysunek 24. Sekwencja zapisu rejestru HTS221

Master	START	AdresSlave R/W=0		adres rejestru		START	Adres Slave R/W=1			NACK	STOP
Slave			ACK		ACK			ACK	odczytane dane		

Rysunek 25. Sekwencja odczytu rejestru HTS221

nić wartość ciśnienia od wysokości wprowadzono pojęcie ciśnienia względnego – inaczej mówiąc zredukowanego do poziomu morza. Jest to taka wyliczona wartość ciśnienia jak gdyby pomiar był wykonywany w miejscu pomiaru, ale na poziomie morza. Żeby wyliczyć ciśnienie względne trzeba znać wysokość nad poziomem morza i temperaturę powietrza. Między innymi do tego celu w barometr wbudowano pomiar temperatury. W zasobach Internetu można znaleźć uproszczone wzory obliczania ciśnienia względnego i można się pokusić o jego wyliczanie na podstawie wysokości (można ją odczytać z odbiornika GPS). Dla przetestowania poprawności odczytywania ciśnienia posłużyłem się kalkulatorem ciśnienia ze strony *stacje-pogody.waw.pl/kalkulator_cisnien.php*. W moim wypadku dla temperatury powietrza ok. 0°C i wysokości pomiaru 228 m n.p.m. różnica pomiędzy zmierzonym ciśnieniem rzeczywistym, a ciśnieniem względnym wynosiła ok. 30 hPa. Jednak okazało się, że po przeliczeniu ciśnienie odpowiadało temu podawanemu przez serwisy pogodowe dla miejsca pomiaru.

Higrometr HTS221

Pomiar wilgotność względnej powietrza jest wykorzystywany w stacjach meteorologicznych, urządzeniach AGD (np. lodówkach) czujnikach wentylatorów łazienkowych, sterownikach klimatyzacji itp. Umieszczony na płytce czujnik HTS221 mierzy wilgotność w zakresie 20…80% rH z dokładnością ±4,5%. Pomiar może być wykonywany z częstotliwością z zakresu 1…12,5 Hz. Oprócz pomiaru wilgotności, czujnik mierzy temperaturę z zakresu -40…+120°C z dokładnością ±0,5°C w zakresie od +15…+40°C oraz ±1°C w zakresie 0…+60°C. Schemat blokowy układu pokazano na **rysunku 23**.

Podstawowym elementem toru pomiaru wilgotności jest sensor pojemnościowy, którego pojemność zmienia się w zależności od wilgotności otoczenia. Konwersja zmiany pojemności na sygnał napięciowy jest wykonywana we wzmacniaczu Charge OpAmp. Sygnały napięciowe z tego wzmacniacza lub z czujnika napięciowego są konwertowane na postać cyfrową przez 16-bitowy przetwornik A/C. Do komunikacji z mikrokontrolerem-hostem może być wykorzystany interfejs I²C lub SPI. Jak wspomniałem, we wszystkich czujnikach zamontowanych na płytce jest uaktywniony interfejs I²C, również w HTS221. Ten układ nie ma wyprowadzeń adresowych i adres slave ma jedną wartość równą 0xBE dla zapisu danych i 0xBF dla odczytu danych. Podobnie jak w wypadku pozostałych czujników, konfiguracja parametrów

Listing 12. Odczytanie i wyświetlanie temperatury
```
void LPS331TempRead(char x, char y)
{
    signed short  temp;//16 bitowa zmienna wyniku
    double temp1=0;
    char data_temp[3];
    char str[20];
//odczytanie rejestrów wyniku
    data_temp[0]=LPS331ReadReg(0x2b);
    data_temp[1]=LPS331ReadReg(0x2c);
//składanie wyniku
    temp=temp+data_temp[1];
    temp=temp<<8;
    temp=temp+data_temp[0];
//konwersja do zmiennej float
    temp1=(double)temp;
//konwersja odczytanej temperatury
    temp1=42.5+(temp1/480);
//zamiana na znaki ASCII
    sprintf(str,"%+ 4.1f C",temp1);
//wyświetlanie wyniku
    OledTxtRam(str,x,y);
}
```

Listing 13. Zapisanie rejestru HTS221
```
define HTS_ADD 0xBE
int HTS221WriteReg(unsigned char addr, char data)
{
    char data_config[2];
    data_config[0] = addr;//adres rejestru
    data_config[1] = data;//wpisywane dane
    int status = i2ch.write(HTS_ADD, data_config, 2, 0);
    return(status);
}
```

Listing 14. Odczytanie rejestru HTS221
```
char HTS221ReadReg(char addr)
{
    char reg[2];
    char data[2];
    reg[0]=addr; //adres rejestru
    i2ch.write(HTS_ADD,reg, 1, 1); // no stop
    i2ch.read(HTS_ADD, data, 1, 0);
    return(data[0]);
}
```

pracy i odczytywanie mierzonych wartości odbywa się przez zapisywanie i odczytywanie wewnętrznych rejestrów. Standardowo będziemy potrzebowali dwie funkcje: odczytania zawartości rejestru o podanym adresie i zapisania rejestru o podanym adresie.

Zapisanie rejestru rozpoczyna się od wysłania przez mikrokontroler sekwencji START, a po niej adresu slave 0xBE. Potwierdzenia adresu przez HTS221 pozwala na wysłanie przez mikrokontroler adresu rejestru, a po nim zapisywanej danej (**rysunek 24**). Na **listingu 13** zamieszczono procedurę zapisywania rejestru z dwoma argumentami: *addr* (adres rejestru) i *data* (dane do zapisania). Odczytanie rejestru rozpoczyna się od wysłania sekwencji START i adresu slave z bitem R/W=0 (zapis) i bajt adresu rejestru. Potem jest wysyłana powtórna sekwencja START, adres slave z bitem R/W=1, a po nim jest

Rejestr	ADRES hex	POR	Funkcja
WHO_AM_I	0F	0xBC	ID układu
AV_CONF	10	0x7A	Rejestr konfiguracji rozdzielczości
CTRL_REG1	20	0	Rejestr kontrolny 1
CTRL_REG2	21	0	Rejestr kontrolny 2
CTRL_REG3	22	0	Rejestr kontrolny 3
STATUS_REG	27	0	Rejestr statusu
HUMIDITY_OUT_L	28	0	Młodsza cześć rejestru mierzonej wilgotności
HUMIDITY_OUT_H	29	0	Starsza cześć rejestru mierzonej wilgotności
TEMP_OUT_L	2A	0	Najstarsza cześć rejestru mierzonego ciśnienia
TEMP_OUT_H	2B	0	Najmłodsza cześć rejestru mierzonej temperatury

Rysunek 25. Wykaz rejestrów sterujących i rejestrów wyniku pomiaru

B7	B6	B5	B4	B3	B2	B1	B0
PD	res	res	res	res	BDU	ODR1	ODR2

PD Power Down PD=0 power down, PD=1 tryb aktywny
BDU – odświeżanie rejestru wyjściowego BDU=0 dane odświeżane ciągle, BDU=1 dane odświeżane po odczycie rejestrów ciśnienia i temperatury
ODR1:ODR1 – częstotliwość odczytywania danych wyjściowych
00 – pomiar na żądanie One Shot
01 – 1 Hz
10 – 7 Hz
11 – 12,5 Hz
Rysunek 26. Rejestr CTRL_REG1

B7	B6	B5	B4	B3	B2	B1	B0
BOOT	res	res	res	res	res	HEATER	ONE_SHOT

BOOT – ustawienie tego bitu powoduje przepisanie ustawień kalibracyjnych z wewnętrznej pamięci FLASH do rejestrów kalibracyjnych
HEATER – ustawienie tego bitu włącza wewnętrzne podgrzewanie w celu usunięcia wilgoci z kondensacji pary wodnej zna czujniku wilgotności. Wyzerowanie bitu powoduje wyłączenie podgrzewania.
ONE_SHOT – ustawienie tego bitu inicjuje jednokrotny pomiar wilgotności i temperatury. Po wykonaniu pomiaru ONE_SHOT jest sprzętowo zerowany
Rysunek 27. Rejestr CTRL_REG2

Listing 15. Inicjalizacja higrometru
```
void HTS221Init(void)
{
    HTS221WriteReg(0x20,0 ); //CTRL_REG1 Power Down
    HTS221Cal(); //odczytanie wspolczynnikow kalibracji
    HTS221WriteReg(0x10, 0x1b); //dokladnosc pomiarow
    HTS221WriteReg(0x20, 0x84); //CTRL_REG1 Power On, One Shot, Update after read
}
```

Rejestr	Adres hex
H0_rH_x2	30
H1_rH_x2	31
T0_degC_x8	32
T1_degC_x8	33
T1/T0 msb	35
H0_T0_OUT	36
	37
H1_T1_OUT	3A
	3B
T0_OUT	3C
	3D
T1_OUT	3E
	3F

Rysunek 28. Rejestry kalibracyjne

Listing 16. Odczytanie rejestrów kalibracyjnych
```
void HTS221Cal(void)
{
    short T0,T1;
    struct
    {
        unsigned char H0_rH_x2;//addr 0x30
        unsigned char H1_rH_x2;//addr 0x31
        unsigned char T0_degC_x8;//addr 0x32
        unsigned char T1_degC_x8;//addr 0x33
        unsigned char T1_T0_msb;//addr 0x35
        short H0_T0_OUT;//addr 36, 37
        short H1_T0_OUT;//addr 3a, 3b
        short T0_OUT;//addr 3c, 3d
        short T1_OUT;//addr 3e, 3f
    }cal;
//współczynniki kalibracji dla wilgotnosci
    cal.H0_rH_x2=HTS221ReadReg(0x30);
    cal.H1_rH_x2=HTS221ReadReg(0x31);
    H0_rh=(float)cal.H0_rH_x2/2;
    H1_rh=(float)cal.H1_rH_x2/2;
    cal.H1_T0_OUT=HTS221ReadReg(0x3b);
    cal.H1_T0_OUT<<=8;
    cal.H1_T0_OUT|=HTS221ReadReg(0x3a);
    cal.H0_T0_OUT=HTS221ReadReg(0x37);
    cal.H0_T0_OUT<<=8;
    cal.H0_T0_OUT|=HTS221ReadReg(0x36);
    H0_cal=cal.H0_T0_OUT;
    H1_cal=cal.H1_T0_OUT;
//wspolczynniki kalibracji dla temperatury
    cal.T0_degC_x8=HTS221ReadReg(0x32);
    cal.T1_T0_msb=HTS221ReadReg(0x35);
    T0=cal.T1_T0_msb&3;
    T0<<=8;
    T0=T0|cal.T0_degC_x8;
    T0_degC=(float)T0/8;
    cal.T1_degC_x8=HTS221ReadReg(0x33);
    cal.T1_T0_msb=HTS221ReadReg(0x35);
    T1=(cal.T1_T0_msb&0x0c);
    T1>>=2;
    T1<<=8;
    T1=T1|cal.T1_degC_x8;
    T1_degC=(float)T1/8;
    cal.T0_OUT=HTS221ReadReg(0x3d);
    cal.T0_OUT<<=8;
    cal.T0_OUT|=HTS221ReadReg(0x3c);
    cal.T1_OUT=HTS221ReadReg(0x3f);
    cal.T1_OUT<<=8;
    cal.T1_OUT|=HTS221ReadReg(0x3e);
     T0_cal=cal.T0_OUT;
     T1_cal=cal.T1_OUT;
}
```

odczytywany jeden bajt zawartości zaadresowanego rejestru (**rysunek 25**). Funkcja *HTS221ReadReg* z **listingu 14** odczytuje i zwraca zawartość rejestru, którego adres jest umieszczony w argumencie *addr*. Wykaz rejestrów sterujących i rejestrów wyników pomiarów pokazano na **rysunku 26**.

Pomiary wilgotności i temperatury mogą być wykonywane na żądanie, po wysłaniu polecenia (one shot) lub w sposób ciągły, z określoną częstotliwością. Do programowania częstotliwości pomiarów jest przeznaczony rejestr *CTRL_REG1* o adresie 0x20 pokazany na **rysunku 26**. Bit *PD* tego rejestru jest przeznaczony do wprowadzania układu w stan obniżonego poboru mocy, a bit *BDU* określa czy rejestr z danymi wyjściowymi wilgotności i temperatury ma być odświeżany automatycznie po wykonaniu pomiarów, czy też po odczytaniu przez mikrokontroler 8 starszych bitów poprzedniego wyniku pomiaru. Po włączeniu zasilania bit *PD* jest wyzerowany i żeby można było wykonywac pomiary trzeba do *PD* wpisać jedynkę.

Wyzerowanie bitów *ODR1* i *ODR2* wprowadza układ w tryb pomiaru na żądanie. Żeby wyzwolić taki pomiar

trzeba ustawić bit *ONE_SHOT* w rejestrze *CTRL_REG2* (**rysunek 27**). Po wyzwoleniu pomiaru, ale też w trybie pomiaru ciągłego trzeba testować, czy wynik pomiaru został zapisany do rejestrów wyjściowych. Żeby to zrobić trzeba odczytać zawartość rejestru statusowego *STATUS_REG*. Ustawienie bitu *H_DA* (bit b1 *STATUS_REG*) oznacza, że wynik pomiaru wilgotności został zapisany do rejestrów wyjściowych *HUMIDITY_OUT_L* i *HUMIDITY_OUT_H*, a ustawienie bitu *T_DA* (bit b0 *STATUS_REG*) oznacza, że wynik pomiaru temperatury został wpisany do rejestrów. W czasie testów czujnika skonfigurowałem układ do pracy z wyzwalaniem na żądanie i z ustawionym bitem BDU. Procedurę inicjalizacyjną pokazano na **listingu 15**.

W poprzednio opisywanych czujnikach wartości odczytywane z wyjściowych rejestrów pomiarów trzeba było przekonwertować z formatu U2, ewentualnie przeskalować lub dodać offset. W HTS221 jest inaczej. Pierwszą zasadniczą różnicą jest umieszczenie w pamięci układu szeregu dodatkowych rejestrów kalibracji. Każdy układ w procesie produkcyjnym jest kalibrowany i w nieulotnej pamięci Flash są zapisywane dane do kalibracji. W czasie sekwencji włączania zasilania układu dane kalibracyjne są przepisywane z pamięci Flash do rejestrów kalibracyjnych pokazanych na **rysunku 28**. W trakcie inicjalizacji układu mikrokontroler musi odczytać dane kalibracyjne, po to by je potem wykorzystać przy każdorazowym odczycie wilgotności i temperatury. Procedurę odczytu rejestrów kalibracyjnych pokazano na **listingu 16**. W wyniku działania tej funkcji są zapisywane zmienne globalne pokazane na **listingu 17**. Te zmienne w połączeniu z 16-bitowymi danymi wyjściowymi będą służyły do wyliczenia mierzonej wartości. W przypadku temperatury wartości *T0_cal*, *T1_cal*, *T0_degC* i *T1degC* są współrzędnymi wyznaczającymi prostą kalibracji – zostało to pokazane na **rysunku 29**. Do zmiennej *Treg* jest wpisywana 16-bitowa wartość odczytana z rejestrów *TEMP_OUT_L* i *TEMP_OUT_H*. Mając wartości odczytane z rejestrów kalibracji i wartość *Treg* wyliczamy wartość temperatury w stopniach Celsjusza w następujący sposób: *T_C = (Treg - T0_cal))/(T1_cal - T0_cal) * (T1_degC - T0_degC) + T0_degC;*

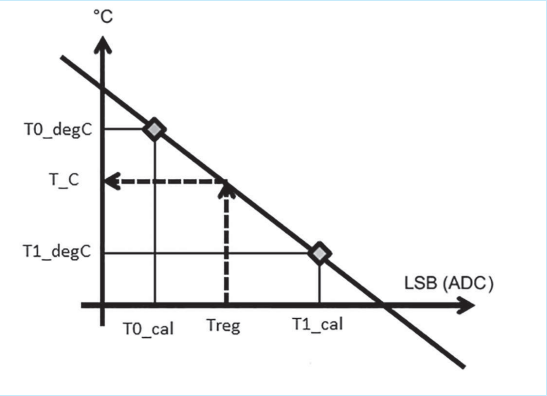
Na **listingu 18** pokazano kompletną procedurę wyzwolenia pojedynczego pomiaru, wyliczenia i wyświetlenia zmierzonej temperatury. Wartości kalibracyjne zostały jednokrotnie pobrane z rejestrów i wpisane do zmiennych w momencie inicjalizacji układu, i nie ma potrzeby ich pobierania przy każdym pomiarze. Podobnie wygląda sposób postępowania przy odczytywaniu, przeliczaniu i wyświetlaniu wilgotności. Wilgotność jest wyliczana w następujący sposób: *H_rh = (((H_T - H0_cal))/(H1_cal - H0_cal) * (H1_rh - H0_rh) + H0_rh* (**rysunek 30**). Procedurę odczytującą, przeliczającą i wyświetlającą wilgotność pokazano na **listingu 19**, natomiast wynik działania tej procedury na **rysunku 31**.

Przyznam się, że ten sposób wyliczania mierzonej wartości budzi moje wątpliwości. Z jednej strony, obliczenia nie są specjalnie trudne. Jednak, aby je wykonać z odpowiednią dokładnością trzeba użyć liczb zmiennoprzecinkowych. Nie ma z tym problemu, kiedy używamy 32-bitowego mikrokontrolera z odpowiednimi zasobami. Jednak, gdyby trzeba było zastosować ten czujnik we współpracy z bardzo uproszczonym mikrokontrolerem 8-bitowym, na przykład rodziny PIC10F,

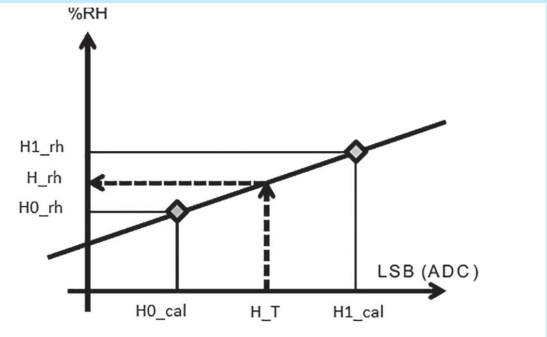
Listing 17. Zmienne kalibracji
```
// Temperaturea w stopniach C dla kalibracji
float T0_degC, T1_degC;
// Wyjsciowa wartosc temperatury dla kalibracji
int16_t T0_cal, T1_cal;
// Wilgotnosc dla kalibracji
float H0_rh, H1_rh;
// Wyjsciowa wartosc wilgotnosci dla kalibracji
int16_t H0_cal, H1_cal;
```

Listing 18. Odczytanie, obliczenie i wyświetlenie temperatury
```
//odczytanie rejestrów temperatury
short HTS221TempRead(void)
{
    short temp;
    temp=HTS221ReadReg(0x2B);//TEMP_OUT_H
    temp=temp<<8;
    temp=temp|HTS221ReadReg(0x2A);//TEMP_OUT_L
    return(temp);
}

//odczytanie i wyświetlenie temperatury
void HTS221ReadTemp(char x, char y)
{
    short Treg, tempt;
    float T_C,Temperature;
    char str[20];
//start pomiaru
    HTS221WriteReg(0x21,1);//start One shot
    while((HTS221ReadReg(0x27)&1)==0); //czekaj na dane pomiaru
    Treg=HTS221TempRead();//odczytaj rejestr temperatury
//wylicz temperaturę
//na podstawie danych kalibracji
    T_C=((float)(Treg-T0_cal))/(T1_cal-T0_cal)*(T1_degC-T0_degC)+T0_degC;
    tempt = (int16_t)(T_C *100);
    Temperature = (float)tempt)/100;
//wyswietl temperature
    sprintf(str,"%+ 4.1f C",Temperature);
    OledTxtRam(str,x,y);
}
```



Rysunek 29. Prosta kalibracji i wyliczanie temperatury w stopniach Celsiusa



Rysunek 30. Prosta kalibracji i wyliczanie wilgotności w %

Listing 19. Odczytanie, obliczenie i wyświetlenie wilgotności

```
short HTS221HumRead(void)
{
     short hum;
     hum=HTS221ReadReg(0x29);//HUMIDITY_OUT_H
     hum=hum<<8;
     hum=hum|HTS221ReadReg(0x28);//HUMIDITY_OUT_L
     return(hum);
}

//odczytanie i wyświetlenie wilgotności
void HTS221ReadHum(char x, char y)
{
     int16_t H_T, humt;
     float H_rh, Hum;
     char str[20];
     //start pomiaru
     HTS221WriteReg(0x21,1);//start One shot
     while((HTS221ReadReg(0x27)&2)==0);
     H_T = HTS221HumRead();//odczytaj rejestr wilgotnosci
     H_rh=((float)(H_T-H0_cal))/(H1_cal-H0_cal)*(H1_rh-H0_rh)+H0_rh;
     humt = (uint16_t)(H_rh * 100);
     Hum= ((float)humt)/100;
      sprintf(str,"% 4.0f",Hum);
      OledTxtRam(str,x,y);
      OledTxt("%", x+5, y);
}
```

Rysunek 31. Pomiary z wszystkich odczytywanych czujników

do zbudowania powiedzmy sterownika wentylatora łazienkowego, to byłoby to bardzo trudne do zrealizowania. Nie wiem dlaczego konstruktorzy nie zaimplementowali tych obliczeń wewnątrz układu, a wynik nie jest udostępniony w rejestrach wyjściowych. Być może w nowszych układach tak będzie.

Podsumowanie

Przetestowałem zestaw Nucleo z płytką rozszerzeń, na której umieszczono cztery czujniki przeznaczone do mierzenia parametrów meteorologicznych: temperatury, ciśnienia atmosferycznego i wilgotności. Przykładowe procedury pozwalają na odczytywanie pomiarów, ich konwersję i wyświetlanie na ekranie wyświetlacza. Używanie wirtualnego środowiska *mbed* znacznie uprościło i przyspieszyło programowanie obsługi czujników. Nie bez znaczenia jest fakt zastosowania jednego interfejsu I²C do komunikacji ze wszystkimi testowanymi czujnikami i wyświetlaczem OLED. Pomimo początkowych oporów *mbed* okazał się prawie porównywalny z komercyjnymi środowiskami typu uVision. Prawie, bo niestety jest pozbawiony obsługi sprzętowego debugera. Można sobie z tym brakiem poradzić przez utworzenie konsoli znakowej łącząc płytkę Nucleo z komputerem PC za pomocą łącza RS232 i wysyłać komunikaty debugowania. Jest to wbrew pozorom bardzo skuteczny sposób debugowania, ale wymaga posiadania komputera z portem RS232 lub konwertera USB/RS232. Mimo to wsparcie w postaci bibliotek *mbed* oraz gotowy, bardzo tani modułu Nucleo i płytki rozszerzeń, to idealne połączenie stanowiące gotowe rozwiązanie niezbędne do pierwszych eksperymentów, jak i do bardziej zaawansowanych testów.

Tomasz Jabłoński, EP