STEVAL-MKI063V1

Przykład oprogramowania akcelerometru i magnetometru LSM303DLH

Technologia MEMS na dobre zagościła już w takich zastosowaniach jak akcelerometry, magnetometry, żyroskopy, czujniki ciśnienia czy mikrofony, a w ofercie ST Microelectronics można znaleźć szeroką gamę tego typu układów. Oprócz samych układów, STM udostępnia także moduły demonstrujące możliwości poszczególnych grup produktów. Jednym z takich modułów jest STEVAL-MKI1063V1 oparty na układzie LSM303DLH zawierającym akcelerometr i magnetometr. W artykule opisano moduł oraz przedstawiono sposób komunikacji z nim z poziomu komputera PC na przykładzie aplikacji napisanej w języku C++/CLI (.NET).

Moduł demonstracyjny

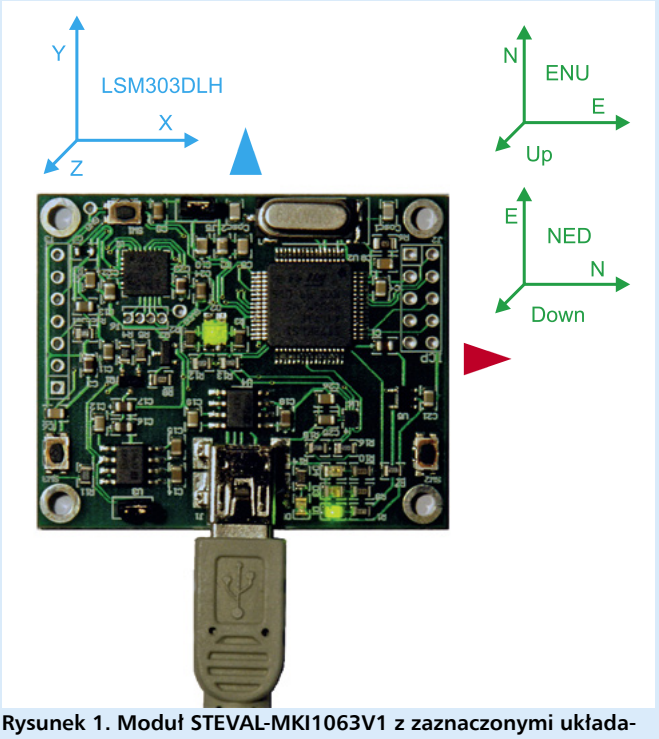
Pokazany na **rysunku 1** moduł STEVAL-MKI1063V1 jest oparty na układzie LSM303DLH. Jest to układ typu *system-in-package* zawierający 3-osiowy akcelerometr i 3-osiowy magnetometr. Oprócz wspomnianego układu, moduł zawiera także mikrokontroler ST7, którego zadaniem jest zapewnienie komunikacji przez złącze USB pomiędzy komputerem PC a układem. Na płytce modułu umieszczono także 3 przyciski, SW1 służy do zerowania układu natomiast SW2 i SW3 nie mają przypisanych funkcji i można je wykorzystać dowolnie.

Poznawanie możliwości modułu najlepiej jest zacząć od pobrania ze strony dedykowanej modułowi STEVAL-MKI1063V1 (*http://www.st.com/internet/evalboard/product/250958.jsp*) pakietu o nazwie *MEMS demonstration kit SW package*. Zawiera on program *Unico*, który służy do wizualizacji danych ze wszystkich modułów demonstracyjnych MEMS od STM. Oprócz *Unico*, pakiet ten zawiera między innymi sterownik wirtualnego portu szeregowego dla mikrokontrolerów ST7 i STM32 oraz pliki źródłowe programów i wykorzystywanych przez nie bibliotek obsługi układów umieszczonych na poszczególnych modułach demonstracyjnych. Należy pamiętać, że podczas instalacji programu *Unico* sterowniki portu szeregowego nie są automatycznie instalowane i należy tę czynność wykonać samodzielnie. Sterownik ten (*USB driver for ST7*) można także pobrać osobno, spod ww. adresu.

Po podłączeniu modułu i wykryciu go przez system Windows można uruchomić program *Unico*. Na ekranie startowym należy wybrać rodzaj modułu, który podłączyliśmy. Następnie, po uruchomieniu się zasadniczej części *Unico*, trzeba wybrać odpowiedni port COM, a następnie kliknąć ikonę nawiązania połączenia. Po połączeniu można, po wybraniu ikony *Start*, rozpoczynamy obserwację napływających danych. Kolejne ikony pozwalają przywołać ekrany, na których można zmienić konfigurację układu, obejrzeć i modyfikować zawartość rejestrów, a także obserwować i wizualizować odbierane dane na kilka sposobów. Jeden ze sposobów wizualizacji pokazano na **rysunku 2**.

Układ LSM303DLH

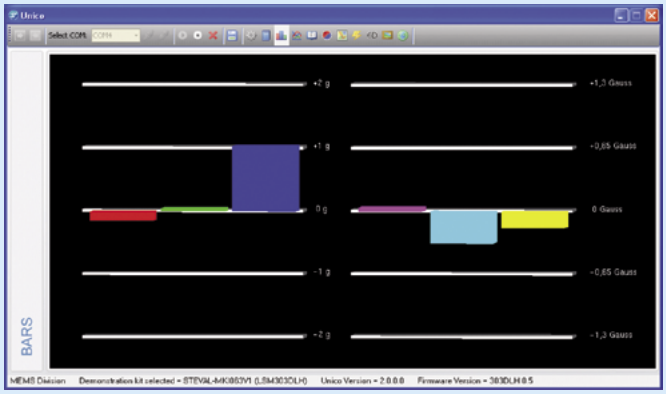
Jak już wspomniano, głównym elementem modułu demonstracyjnego jest układ LSM303DLH, zawierający w sobie 3-osiowy akcelerometr (o dostępnych zakresach ±2, ±4 lub ±8 g) i 3-osiowy magnetometr (o zakresach ±1,3, ±1,9, ±2,5, ±4,0, ±4,7, ±5,6 i ±8,1 Gaussa). Ponadto układ ma m.in. możliwość wykrywania swobodnego spadku oraz zgłaszania przerwań, których źródła można konfigurować. Komunikacja pomiędzy mikrokontrolerem a układem odbywa się z wykorzystaniem interfejsu I²C. Układ wykonuje pomiary z rozdzielczością 12 bitów, a wyniki udostępnianie są w postaci słów 16-bitowych, z tym, że dane z akcelerometru wyrównane są do lewej,



Rysunek 1. Moduł STEVAL-MKI1063V1 z zaznaczonymi układami współrzędnych

Tabela 1. Zakresy i czułości magnetometru

Zakres [gauss]	Wzmocnienie osi X i Y [LSB/gauss]	Wzmocnienie osi Z [LSB/gauss]
±1,3	1055	950
±1,9	795	710
±2,5	635	570
±4,0	430	385
±4,7	375	335
±5,6	320	285
±8,1	230	205



Rysunek 2. Jeden ze sposobów wizualizacja danych odbieranych przez program *Unico*

a z magnetometru do prawej strony słowa. Ponadto, czułość magnetometru w osi Z jest o około 10% niższa niż w osiach X i Y, co należy uwzględniać przy przetwarzaniu danych z pomiarów. Odpowiednie wartości wzmocnień podane są w **tabeli 1**.

Warto w tym miejscu zauważyć, że układ LSM303DLH został wycofany z oferty ST Microelectronics, a jego miejsce zajęły układy LSM303DLM i LSM303DLHC. Ten pierwszy różni się od wersji DLH przede wszystkim niższym poborem mocy oraz nieco innymi czułościami magnetometru, jest jednak zgodny pod względem zakresów pomiarów oraz znaczenia rejestrów (z wyjątkiem zamiany adresów rejestrów wynikowych dla osi Y i Z magnetometru). Natomiast w układzie DLHC m.in. akcelerometr ma dodatkowo zakres ±16g, inaczej zorientowane są osie czujników, dostępnych jest więcej opcji konfiguracji, a co za tym idzie, niektóre rejestry mają inną zawartość i znaczenie niż w pozostałych dwóch układach.

Komunikacja z modułem

Jak wspomniano, moduł można podłączyć do komputera poprzez interfejs USB i obserwować wyniki jego działania w programie *Unico*. Jest to jednak mało przydatne z punktu widzenia możliwości wykorzystania modułu do własnych aplikacji. Na szczęście z modułem można skomunikować się samodzielnie poprzez wirtualny port COM, a polecenia wydawane są w postaci dość prostego zestawu komend (**tabela 2**). Wysyłane są one w postaci tekstowej i muszą być zakończone znakiem 0x10 (\r) co sprawia, że można sterować modułem nawet z poziomu zwykłego terminala. Na przykład komenda **mw000C\r* oznacza zapis wartości 0x0C do rejestru magnetometru o adresie 0x00. Parametry połączenia to: szybkość 115200 bps, 8 bitów danych, baz kontroli parzystości, 1 bit stopu, bez kontroli przepływu.

Tabela 2. Najważniejsze komendy sterujące modułem

Komenda	Znaczenie
*zoff	Aktywacja LSM303DLH
*zon	Dezaktywacja LSM303DLH
*ver	Pobranie numeru wersji oprogramowania modułu
*dev	Pobranie nazwy układu (zwraca: LSM303DLH)
*start	Rozpoczęcie wysyłania przez moduł danych w postaci zakodowanej
*stop	Zatrzymanie wysyłania danych przez moduł
*debug	Rozpoczęcie wysyłania przez moduł danych w jawnej postaci tekstowej (do celów diagnostycznych)
*waadd	Zapis wartość dd do rejestru o adresie aa (obie wartości podawane szesnastkowo) – akcelerometr
*raa	Odczyt zawartości rejestru o adresie aa (adres podawany szesnastkowo) – akcelerometr
*mwaadd	Zapis wartość dd do rejestru o adresie aa (obie wartości podawane szesnastkowo) – magnetometr
*mraa	Odczyt zawartości rejestru o adresie aa (adres podawany szesnastkowo) – magnetometr

Po nawiązaniu połączenia szeregowego, układ LSM303DLH można aktywować komendą **zoff*. Dopiero później można wykonać inne komendy. Wyjątkiem są jedynie komendy **dev* i **ver*, na które moduł reaguje również wtedy, gdy układ LSM303DLH jest nieaktywny.

Po wysłaniu komendy **start* moduł zaczyna automatycznie wysyłać dane z układu LSM303DLH. Są one zwracane w postaci pakietów zawierających 19 znaków i zakończonych sekwencją *\r\n*. Kolejne bajty zawierają:

- Literę „*s*”.
- Literę „*t*”.
- Starszy bajt wyniku pomiaru przyśpieszenia w osi X.
- Młodszy bajt wyniku pomiaru przyśpieszenia w osi X.
- Starszy bajt wyniku pomiaru przyśpieszenia w osi Y.
- Młodszy bajt wyniku pomiaru przyśpieszenia w osi Y.
- Starszy bajt wyniku pomiaru przyśpieszenia w osi Z.
- Młodszy bajt wyniku pomiaru przyśpieszenia w osi Z.
- Starszy bajt wyniku pomiaru natężenia pola magnetycznego w osi X.
- Młodszy bajt wyniku pomiaru natężenia pola magnetycznego w osi X.
- Starszy bajt wyniku pomiaru natężenia pola magnetycznego w osi Y.
- Młodszy bajt wyniku pomiaru natężenia pola magnetycznego w osi Y.
- Starszy bajt wyniku pomiaru natężenia pola magnetycznego w osi Z.
- Młodszy bajt wyniku pomiaru natężenia pola magnetycznego w osi Z.
- Zgłoszenie przerwania 1 (0x00 – brak przerwania, 0x40 – zgłoszenie aktywne).
- Zgłoszenie przerwania 2 (0 x00 – brak przerwania, 0x40 – zgłoszenie aktywne).
- Stan przycisków SW (0x00 – oba zwolnione, 0x01 – wciśnięty SW2, 0x02 – wciśnięty SW3, 0x03 – wciśnięte SW2 i SW3).

Bajty 0 i 1 zawsze zawierają sekwencję „*st*” co pozwala sprawdzić, czy pakiet został odebrany prawidłowo.

Patrząc na listę częstotliwości próbkowania dostępnych dla akcelerometru (0,5, 1, 2, 5 i 10 Hz w trybie *low-power* oraz 50, 100, 400 i 1000 Hz w trybie normalnym) i magnetometru (0,75, 1,5, 3, 7,5, 15, 30 i 75 Hz) można zauważyć, że nie są one ze sobą zgodne. Powstaje więc pytanie, z jaką częstotliwością moduł będzie wysyłać dane do komputera PC. W przypadku modułu, do którego

Rysunek 3. Ekran programu demonstracyjnego

dostęp miał autor, była to częstotliwość wybrana w ustawieniach akcelerometru. Analiza kodu źródłowego *firmware* dostarczanego wraz z *Unico* wykazała jednak, że w niektórych jego wersjach wybierana jest wyższa z częstotliwości wybranych dla obu układów, co wydaje się być bardziej uzasadnionym rozwiązaniem. Niezależnie od wyboru, wysyłane pakiety zawierają zawsze komplet wartości, z tym, że dane z czujnika „wolniejszego" są kilkukrotnie powtarzane w kolejnych pakietach. Domyślna konfiguracja układu jest następująca: częstotliwość próbkowania akcelerometru 50 Hz, zakres akcelerometru ±2 g, częstotliwość próbkowania magnetometru 15 Hz, zakres magnetometru ±1,5 Gaussa. Wybór częstotliwości próbkowania wiąże się z automatycznym wyborem nastaw wewnętrznego filtru dolnoprzepustowego (antyaliasingowego). Oprócz tego, dla akcelerometru można także włączyć filtrowanie danych filtrem górnoprzepustowym, co m.in. pozwala usunąć z sygnału składową stałą. Jeżeli chcemy zmienić konfigurację układu, należy zatrzymać wysyłanie danych komendą **stop*, a następnie poprzez komendy **waadd* i **mwaadd* ustawić odpowiednie wartości w rejestrach układu. W **tabeli 3** zawarto zestawienie najważniejszych z nich.

Tabela 3. Najważniejsze rejestry układu LSM303DLH

Nazwa rejestru	adres	Znaczenie
Akcelerometr		
CTRL_REG1_A	0x20	Aktywacja akcelerometru, wybór częstotliwości próbkowania, aktywacja pomiaru w poszczególnych osiach
CTRL_REG2_A	0x21	Ustawienia filtru górnoprzepustowego
CTRL_REG3_A	0x22	Konfiguracja zgłaszania przerwań
CTRL_REG4_A	0x23	Wybór trybu pomiaru, zapisu danych big/little endian, zakresu pomiarowego
STATUS_REG_A	0x27	Gotowość danych dla poszczególnych osi
OUT_X_L_A	0x28	Młodszy bajt wyniku dla osi X
OUT_X_H_A	0x29	Starszy bajt wyniku dla osi X
OUT_Y_L_A	0x2A	Młodszy bajt wyniku dla osi X
OUT_Y_H_A	0x2B	Starszy bajt wyniku dla osi X
OUT_Z_L_A	0x2C	Młodszy bajt wyniku dla osi X
OUT_Z_H_A	0x2D	Starszy bajt wyniku dla osi X
INT1_CFG_A	0x30	Włączenie źródeł przerwań przerwania 1
INT1_SRC_A	0x31	Sygnalizacja źródeł przerwań przerwania 1
INT2_CFG_A	0x34	Włączenie źródeł przerwań przerwania 2
INT2_SRC_A	0x35	Sygnalizacja źródeł przerwań przerwania 2
Magnetometr		
CRA_REG_M	0x00	Wybór częstotliwości próbkowania
CRB_REG_M	0x01	Wybór zakresu pomiarowego
OUT_XH_M	0x03	Starszy bajt wyniku dla osi X
OUT_XL_M	0x04	Młodszy bajt wyniku dla osi X
OUT_YH_M	0x05	Starszy bajt wyniku dla osi Y
OUT_YL_M	0x06	Młodszy bajt wyniku dla osi Y
OUT_ZH_M	0x07	Starszy bajt wyniku dla osi Z
OUT_ZL_M	0x08	Młodszy bajt wyniku dla osi Z

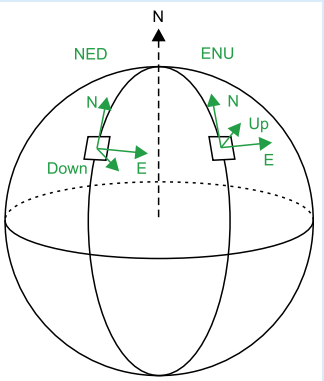
Program demonstracyjny

Aby zaprezentować sposób komunikacji z modułem STEVAL-MKI1063V1, napisano aplikację w języku C++/CLI w środowisku Microsoft Visual C++ 2008 Express. Jej ekran pokazany jest na **rysunku 4**. Pozwala ona na skonfigurowanie parametrów pomiaru, a następnie obserwowanie danych nadchodzących z modułu. Dodatkowo, w programie zaimplementowano cyfrowy kompas z funkcją automatycznej kalibracji.

Na **listingu 1** pokazano deklarację kilku zmiennych należących do klasy *Form1* opisującej formularz okna programu. Zmienne te wykorzystywane są w dalszej części programu. Pierwsze trzy z nich to tablice daneS, daneP i danePKor, które zawierają, kolejno: dane „surowe", czyli odebrane z modułu STEVAL i odkodowane; dane przeliczone na jednostki fizyczne; dane przeliczone na jednostki fizyczne i skorygowane. Zmienna azymut zawiera kąt azymutu wyznaczony w celu narysowania wskaźnika kompasu. Na końcu listingu znajduje się uchwyt do zmiennej obiektowej serialPort, która jest odpowiedzialna za komunikację przez port szeregowy. Klasa SerialPort jest klasą standardową zawartą w bibliotekach środowiska .NET od wersji 2.0. Inicjalizacja większości z tych zmiennych odbywa się w konstruktorze klasy Form1 (**listing 2**). Z kolei na **listingu 3** przedstawiono deklarację klasy LSMKonfigClass, której zadaniem jest przechowywanie aktualnego zestawu nastaw konfiguracyjnych układu LSM303DLH.

Dwa główne elementy kodu programu to klasa watekOdczytuClass oraz metoda z klasy Form1 o nazwie buttonStart_Click(). Metoda ta jest wywoływana po naciśnięciu przez użytkownika przycisku *Start*. Realizowane w niej polecenia mają za zadanie przygotować transmisję z i do modułu STEVAL-MKI063V1, wykonać konfigurację układu LSM303DLH, a następnie rozpocząć zasadniczą część transmisji, czyli uruchomić wątek programu odpowiedzialny za odbiór danych pomiarowych.

Kod metody jest pokazany na **listingu 4**. Na jej początku znajdują się deklaracje tablic zawierających komendy, które trzeba wysłać do modułu STEVAL w zależności od wybranej przez użytkownika konfiguracji układu LSM303DLH. Pierwszą istotną operacją jest sprawdzenie czy port szeregowy nie jest już otwarty. Jeśli jest – zostaje zamknięty. Ma to na celu zakończenie poprzedniego połączenia, jeśli z jakiejś przyczyny nie został on zamknięty w prawidłowy sposób, np. nastąpiło wyłączenie modułu w czasie pomiaru. W kolejnym kroku następuje właściwe otwarcie portu z parametrami ustawionymi w konstruktorze klasy Form1 oraz przez użytkownika w oknie wyboru portu COM. Następnie, program wysyła do modułu STEVAL komendy **zoff* oraz **stop*. Pierwsza z nich aktywuje układ LSM303DLH, zaś druga zatrzymuje wysyłanie danych. Jest to konieczne, ponieważ układ, po połączeniu, nie musi być w stanie zatrzymania (np. wcześniej wykonane było połączenie przez terminal, które nie zostało zakończone komendą **stop*). Komendy wysyłane są za pomocą metody PortWrite( ), której kod pokazano na **listingu 5**. W kolejnych krokach, wysyłane są komendy zapisujące do rejestrów układu odpowiednie nastawy zakresów i częstotliwości próbkowania akcelerometru i magnetometru. Wybór komend zależy od ustawień dokonanych przez użytkownika z list rozwijalnych w oknie programu. Po zakończeniu konfiguracji pomiaru, jest tworzony nowy wątek o nazwie watekOdczytu. Jego zadaniem jest cykliczne odczy-



Rysunek 4. Orientacja przestrzenna osi w układach NED i ENU

Listing 1. Deklaracje kilku zmiennych wykorzystywanych w programie

```
//odczytane dane „surowe”
private: static array <System::Int16>^  daneS;
//odczytane dane po przeliczeniu na jednostki fizyczne
private: static array <System::Double>^ daneP;
//odczytane dane po przeliczeniu i korekcji (kalibracji)
private: static array <System::Double>^ danePKor;
//azymut kompasu
private: static System::Double          azymut;
//wybór układu osi ENU / NED
private: static bool                    osieENU;
//stan połączenia
private: static bool                    polaczono;
//polecenie zakończenia odbierania danych
private: static bool                    zakonczPomiar;
//liczba bajtów w buforze odczytu
private: static int                     zajetoscBufora;
//uchwyt do wątku odczytującego dane
private: Thread ^                       watekOdczytu;
//uchywt do obiektu portu szeregowego
private: static System::IO::Ports::SerialPort^
serialPort;
```

Listing 2. Fragment konstruktora klasy *Form1* odpowiedzialny za inicjalizację zmiennych z listingu 1

```
serialPort = gcnew System::IO::Ports::SerialPort;
serialPort->PortName = L”COM1”;
serialPort->BaudRate = 115200;
serialPort->DataBits = 8;
serialPort->Parity   = System::IO::Ports::Parity::None;
serialPort->StopBits = System::IO::Ports::StopBits::One;
serialPort->ReadTimeout  = 4000;
serialPort->WriteTimeout = 4000;
zajetoscBufora = 0;
polaczono      = false;
zakonczPomiar  = true;
delegat =gcnew UpdateUIDelegate(this, &Form1::UpdateUI);
daneS   =gcnew array<System::Int16>(7); //dane surowe
daneP   =gcnew array<System::Double>(7);//dane
przetworzone
danePKor=gcnew array<System::Double>(7);//dane
przetworzone i skorygowane
```

Listing 3. *Deklaracja klasy LSMKonfigClass*

```
private: ref class LSMKonfigClass{
  public: int    Az;      //akcelerometr - zakres
  public: double Mz;      //magnetometr - zakres
  public: int    McXY;    //magnetometr - wzmocnienie osi
X i Y
  public: int    McZ;     //magnetometr - wzmocnienie osi
Z
  public: bool   osieENU; //wybór układu współrzędnych
};
```

tywanie bufora odbiorczego portu szeregowego, dekodowanie zawartych w nim pakietów danych oraz konwersja odczytanych danych. Wykorzystanie mechanizmu wielowątkowości jest konieczne, aby operacje odczytu nie zablokowały użytkownikowi możliwości interakcji z programem. Gdyby odczyt danych realizowany był w pętli wewnątrz metody buttonStart_Click( ), wówczas aż do zakończenia pętli interfejs użytkownika byłby nieaktywny. Nie można by więc nawet użyć przycisku *Stop*. Uruchomienie wątku pozwoli powrócić programowi do obsługi kolejki komunikatów systemu, a co za tym idzie, program będzie reagował na polecenia użytkownika, zaś odczyt danych i aktualizacja interfejsu nowymi wyświetlanymi wartościami pomiarów będą wykonywane w tle. Po przygotowaniu wątku odczytu następuje wysłanie do modułu komendy **start*. Od tego momentu moduł rozpoczyna wysyłanie pakietów danych. Następnie uruchamiany jest utworzony przed chwilą wątek. W przypadku wystąpienia błędów przy otwarciu portu lub podczas wysyłania komend system zgłasza wyjątki. Są one obsługiwane na końcu metody buttonStart_Click( ). Obsługa ta sprowadza się do wyświetlenia komunikatu o błędzie i zakończeniu transmisji. Występująca w tym fragmencie metoda StopTransimsji( ) wysyła komendę **stop* i zamyka port. Wywoływana jest ona w programie w sytuacji gdy użytkownik kliknie przycisk *Stop*, zamknie program lub, co opisano powyżej, gdy wystąpi wyjątek związany z błędem dostępu do portu szeregowego.

Jak już wspomniano, zasadniczą część programu związaną z odczytem kolejnych pakietów wysyłanych przez moduł STEVAL do komputera PC realizuje osobny wątek. Kod klasy watekOdczytuClass opisującej ten wątek pokazany jest na **listingu 6**. Konstruktor klasy, jako parametry wywołania pobiera ustawienia konfiguracji pomiaru, które wykorzystywane są później do konwersji danych, oraz uchwyt do okna formularza programu, który wykorzystywany jest w celu aktualizacji wartości wyświetlanych na tymże formularzu. Najważniejszą częścią klasy jest metoda PortRead( ). Rozpoczyna się ona od sprawdzenia, czy w buforze odbiorczym portu jest co najmniej 19 bajtów danych. Wartość ta wynika z długości pojedynczego pakietu danych. Jeśli w buforze jest wystarczająca liczba znaków, rozpoczyna się ich odczyt pakiet po pakiecie. Następnie wykonywane jest dekodowanie (wyniki zapisane w tablicy daneS) i przeliczenie odczytanych danych na fizyczne wartości przyśpieszenia i natężenia pola magnetycznego (wyniki zapisane w tablicy daneP). Kolejną wykonywaną operacją jest wprowadzenie korekt kalibrujących odczyty. W przypadku akcelerometru korekty te zostały wyznaczone *off-line*, tzn. zaobserwowano wskazania akcelerometru w poszczególnych osiach i na stałe zapisano odpowiednie wartości korygujące w kodzie programu. W przypadku magnetometru zastosowano procedurę automatycznej kalibracji. Należy tu zwrócić uwagę, że „surowe” odczyty z magnetometru często są obarczone dużymi błędami. W przypadku idealnym wszystkie odczytane wartości powinny opisywać punkty znajdujące się na powierzchni kuli. Tymczasem wzmocnienia w poszczególnych osiach nie są takie same (nawet po uwzględnieniu wzmocnień podanych w danych katalogowych układu), co sprawia, że zamiast kuli punkty reprezentujące odczyty tworzą elipsoidę. Ponadto, środek tej elipsoidy nie wypada w środku układu współrzędnych – jest ona dość znacznie przesunięta. Błędy te można na szczęście skorygować w dość prosty sposób. Wystarczy rejestrować maksymalne i minimalne wartości natężeń pola w poszczególnych osiach, a następnie, na ich podstawie wyznaczyć odpowiednie przesunięcia oraz wzmocnienia dla każdej z osi. Korekta odczytów wykonywana jest według wzoru:

$$W_{kor} = \left(W_{mierz} - A - W_{\min}\right)\left(A_{\max}/A\right)$$

gdzie

W_{kor} – wartość skorygowana

W_{mierz} – wartość zmierzona

A – amplituda sygnału $A = \left(W_{\max} - W_{\min}\right)/2$

W_{max} – wartość maksymalna sygnału

W_{min} – wartość minimalna sygnału

A_{max} – najwyższa amplituda dla wszystkich trzech osi magnetometru

W module, którym dysponował autor przesunięcia wynosiły nawet 40% odczytywanego zakresu (np. w jednej z osi wartości maksymalne były rzędu +0,05 Gaussa, minimalne –0,45 Gaussa, a prawidłowo powinno to być ±0,25 Gaussa). Z kolei różnice wzmocnień pomiędzy osiami nie przekraczały 5%. Zawarty na listingu 6 fragment dotyczący korekcji odczytów z magnetometru realizuje opisaną wyżej metodę. Przy takim rozwiązaniu, żeby skalibrować układ wystarczy po każdym uruchomieniu pomiarów kilkukrotnie obrócić go we wszystkich osiach. Uzyskuje się w ten sposób efekt automatycznej, samoczynnej kalibracji odczytów. Dodać należy rów-

Listing 4. Kod metody *buttonStart_Click( )*

```
private: System::Void buttonStart_Click(System::Object^  sender, System::EventArgs^  e) {
  //tablica komend dla LSM303DLH - akcelerometr - zakres
  array <String ^>^ rejestrAzakresK = gcnew array<String ^>(3);
  rejestrAzakresK[0]=L"*w2380\r"; //2 g
  rejestrAzakresK[1]=L"*w2390\r"; //4 g
  rejestrAzakresK[2]=L"*w23A0\r"; //8 g
  //akcelerometr - zakres - wartości liczbowe
  array <int>^ rejestrAzakres = gcnew array<int>(3);
  rejestrAzakres[0]=2;
  rejestrAzakres[1]=4;
  rejestrAzakres[2]=8;
  //tablica komend dla LSM303DLH - akcelerometr - częstotliwość
  array <String ^>^ rejestrAczestK = gcnew array<String ^>(9);
  rejestrAczestK[0]=L"*w2047\r";  //   0,5 Hz - Low-power
  rejestrAczestK[1]=L"*w2067\r";  //   1   Hz - Low-power
  rejestrAczestK[2]=L"*w2087\r";  //   2   Hz - Low-power
  rejestrAczestK[3]=L"*w20A7\r";  //   5   Hz - Low-power
  rejestrAczestK[4]=L"*w20C7\r";  //  10   Hz - Low-power
  rejestrAczestK[5]=L"*w2027\r";  //  50   Hz
  rejestrAczestK[6]=L"*w202F\r";  // 100   Hz
  rejestrAczestK[7]=L"*w2037\r";  // 400   Hz
  rejestrAczestK[8]=L"*w203F\r";  //1000   Hz
  //tablica komend dla LSM303DLH - magnetometr - zakres
  array <String ^>^ rejestrMzakresK = gcnew array<String ^>(7);
  rejestrMzakresK[0]=L"*mw0120\r"; //1,3 gauss
  rejestrMzakresK[1]=L"*mw0140\r"; //1,9 gauss
  rejestrMzakresK[2]=L"*mw0160\r"; //2,5 gauss
  rejestrMzakresK[3]=L"*mw0180\r"; //4,0 gauss
  rejestrMzakresK[4]=L"*mw01A0\r"; //4,7 gauss
  rejestrMzakresK[5]=L"*mw01C0\r"; //5,9 gauss
  rejestrMzakresK[6]=L"*mw01E0\r"; //8,1 gauss
  //magnetometr - wzmocnienia w poszczególnych osiach
  array <int,2>^ rejestrMzakres = gcnew array<int,2>(7,3);
  rejestrMzakres[0,0]=13; rejestrMzakres[0,1]=1055; rejestrMzakres[0,2]=950; //1,3 gauss
  rejestrMzakres[1,0]=19; rejestrMzakres[1,1]= 795; rejestrMzakres[1,2]=710; //1,9 gauss
  rejestrMzakres[2,0]=25; rejestrMzakres[2,1]= 635; rejestrMzakres[2,2]=570; //2,5 gauss
  rejestrMzakres[3,0]=40; rejestrMzakres[3,1]= 430; rejestrMzakres[3,2]=385; //4,0 gauss
  rejestrMzakres[4,0]=47; rejestrMzakres[4,1]= 375; rejestrMzakres[4,2]=335; //4,7 gauss
  rejestrMzakres[5,0]=59; rejestrMzakres[5,1]= 320; rejestrMzakres[5,2]=285; //5,9 gauss
  rejestrMzakres[6,0]=81; rejestrMzakres[6,1]= 230; rejestrMzakres[6,2]=205; //8,1 gauss
  //tablica komend dla LSM303DLH - magnetometr - częstotliwość
  array <String ^>^ rejestrMczestK = gcnew array<String ^>(7);
  rejestrMczestK[0]=L"*mw0000\r"; //  0,75 Hz
  rejestrMczestK[1]=L"*mw0004\r"; //  1,5  Hz
  rejestrMczestK[2]=L"*mw0008\r"; //  3,0  Hz
  rejestrMczestK[3]=L"*mw000C\r"; //  7,5  Hz
  rejestrMczestK[4]=L"*mw0010\r"; // 15    Hz
  rejestrMczestK[5]=L"*mw0014\r"; // 30    Hz
  rejestrMczestK[6]=L"*mw0018\r"; // 75    Hz
  LSMKonfigClass^ LSMKonfig=gcnew LSMKonfigClass;
  buttonStart->Enabled=false;
  buttonStop->Enabled=true;
  buttonStop->Focus();
  try {
    //zamknięcie portu jeśli z jakiś przyczny był otwarty,
    //(np. wyłączenie modułu w czasie poprzedniego pomiaru)
    if (serialPort->IsOpen) {serialPort->Close(); Thread::Sleep(1000);}
    serialPort->PortName=comboBoxCOM->Items[comboBoxCOM->SelectedIndex]->ToString();
    serialPort->Open();
    polaczono=true;
    //aktywacja LSM303DLH
    PortWrite(L"*zoff\r");
    //zatrzymanie pomiaru (układ mógł być włączony np. z innej aplikacji)
    PortWrite(L"*stop\r");
    //wyczyszczenie bufora odczytu
    serialPort->DiscardInBuffer();
    //Odczyt ID układu
    PortWrite(L"*dev\r");
    labelID->Text=serialPort->ReadLine();
    //Wybór zakresu akcelerometru
    PortWrite(rejestrAzakresK[comboBoxAzakres->SelectedIndex]);
    LSMKonfig->Az=rejestrAzakres[comboBoxAzakres->SelectedIndex];
    //Wybór częstotliwości akcelerometru
    PortWrite(rejestrAczestK[comboBoxAczest->SelectedIndex]);
    //Wybór zakresu magnetomeru, przypisanie czułości
    PortWrite(rejestrMzakresK[comboBoxMzakres->SelectedIndex]);
    LSMKonfig->Mz  =rejestrMzakres[comboBoxMzakres->SelectedIndex,0];
    LSMKonfig->McXY=rejestrMzakres[comboBoxMzakres->SelectedIndex,1];
    LSMKonfig->McZ =rejestrMzakres[comboBoxMzakres->SelectedIndex,2];
    LSMKonfig->osieENU=radioButtonENU->Checked; //wybór osi
    //Wybór częstotliwości magnetometru
    PortWrite(rejestrMczestK[comboBoxMczest->SelectedIndex]);
    //przygotowanie wątku odczytującego wartości pomiarów
    watekOdczytuClass ^wOdczyt = gcnew watekOdczytuClass(this, LSMKonfig);
    watekOdczytu = gcnew Thread(gcnew ThreadStart(wOdczyt, &watekOdczytuClass::PortRead));
    //rozpoczęcie pomiarów,
    //układ będzie automatycznie wysyłał pakiety wyników
    PortWrite(L"*start\r");
    zakonczPomiar=false;
    //uruchomienie wątku odczytującego pakiety z modułu STEVAL
    watekOdczytu->Start();
    Thread::Sleep(10);
  }
  catch (...){
      MessageBox::Show("Błąd komunikacji z modułem STEVAL!\nSprawdź, czy jest podłączony
i aktywny.", "Błąd", MessageBoxButtons::OK, MessageBoxIcon::Error);
      StopTransmisji();
  }
}
```

Listing 6. Kod klasy *watekOdczytuClass*

```
private: ref class watekOdczytuClass {
  private: Form1^ oknoForm1;
  private: LSMKonfigClass LSMKonfig;
  public:  watekOdczytuClass(Form1^ formularz, LSMKonfigClass^ LSMKonfigData) {
    oknoForm1 = formularz;
    LSMKonfig.Az    = LSMKonfigData->Az;
    LSMKonfig.Mz    = LSMKonfigData->Mz;
    LSMKonfig.McXY = LSMKonfigData->McXY;
    LSMKonfig.McZ   = LSMKonfigData->McZ;
    LSMKonfig.osieENU = LSMKonfigData->osieENU;
    osieENU=LSMKonfigData->osieENU;
  }

  public: System::Void PortRead(){
    int i;
    array<unsigned char>^ buf;
    System::Int16 slowo;
    double maxMX=-50, minMX=50, maxMY=-50, minMY=50, maxMZ=-50, minMZ=50;
    double ampMX, ampMY, ampMZ, maxampM;
    buf=gcnew array<unsigned char>(19);
    while ((!zakonczPomiar)&&(serialPort->IsOpen)){
      try {
        //co ~0.1 sekundy sprawdzamy zawartosc bufora
        //jesli sa dane, to odczytujemy i przetwarzamy po kolei
        //wszystkie pakiety
        Thread::Sleep(100);
        zajetoscBufora=serialPort->BytesToRead;
        while ((serialPort->BytesToRead>=19)&&(!zakonczPomiar)){
          serialPort->Read(buf,0,19);
          //bajty 2-7 - dane z akcelerometru
          for (i=1; i<4; i++){
            slowo=buf[i*2];
            slowo=slowo<<8;
            slowo=slowo+buf[i*2+1];
            daneS[i-1]=Convert::ToInt16(slowo)/16;
            daneP[i-1]=static_cast<System::Double>(daneS[i-1])*LSMKonfig.Az/static_cast<System::Double>(2048);
          }
          //Korekta odczytów akcelerometru dla każdego zakresu i osi inna
          switch (LSMKonfig.Az){
            case 2:{danePKor[0]=daneP[0]+0.002;
                    danePKor[1]=daneP[1]+0.025;
                    danePKor[2]=daneP[2]-0.043;} break;
            case 4:{danePKor[0]=daneP[0]+0;
                    danePKor[1]=daneP[1]+0.038;
                    danePKor[2]=daneP[2]-0.035;} break;
            case 8:{danePKor[0]=daneP[0]+0;
                    danePKor[1]=daneP[1]+0.037;
                    danePKor[2]=daneP[2]-0.035;} break;
          }
          //bajty 8-13 - dane z magnetometru
          for (i=4; i<7; i++){
            slowo=buf[i*2];
            slowo=slowo<<8;
            slowo=slowo+buf[i*2+1];
            daneS[i-1]=Convert::ToInt16(slowo);
            if (i<6) {daneP[i-1]=static_cast<System::Double>(daneS[i-1])/static_cast<System::Double>(LSMKonfig.McXY);}
            else {    daneP[i-1]=static_cast<System::Double>(daneS[i-1])/static_cast<System::Double>(LSMKonfig.McZ);}
          }
          //bajty 14 i 15 - źródła przerwań - nie wykorzystywane w tym programie
          //bajt 16 - stan SW2 i SW3
          slowo=buf[16];
          daneS[6]=Convert::ToInt16(slowo);
          daneP[6]=daneS[6];
          danePKor[6]=daneS[6];
          //bajt 17 i 18 - znaki \r\n
          //automatyczna kalibracja odczytów z magnetometru i wyliczenie azymutu dla kompasu
          if (daneP[3]>maxMX) {maxMX=daneP[3];}
            else if (daneP[3]<minMX) {minMX=daneP[3];}
          if (daneP[4]>maxMY) {maxMY=daneP[4];}
            else if (daneP[4]<minMY) {minMY=daneP[4];}
          if (daneP[5]>maxMZ) {maxMZ=daneP[5];}
            else if (daneP[5]<minMZ) {minMZ=daneP[5];}
          //jeśli amplituda==0 oznacza to, że jeszcze nie nadeszly odczyty
          //np. gdy czest. magnetometru < czest.akcelerometru
          //należy tymczasowo skorygować odczyty
          ampMX=(maxMX-minMX)/2; if (ampMX==0) {ampMX=1;}
          ampMY=(maxMY-minMY)/2; if (ampMY==0) {ampMY=1;}
          ampMZ=(maxMZ-minMZ)/2; if (ampMZ==0) {ampMZ=1;}
          maxampM=Math::Max(Math::Max(ampMX,ampMY),ampMZ);
          //ostateczna korekta wartości
          danePKor[3]=(daneP[3]-ampMX-minMX)*(maxampM/ampMX);
          danePKor[4]=(daneP[4]-ampMY-minMY)*(maxampM/ampMY);
          danePKor[5]=(daneP[5]-ampMZ-minMZ)*(maxampM/ampMZ);
          //wyznaczenie azymutu w zależności od wyboru układu osi
          if (osieENU){
            azymut=Math::Atan2(danePKor[3],-danePKor[4])*180/Math::PI+180;
          } else {
            azymut=Math::Atan2(danePKor[4],-danePKor[3])*180/Math::PI+180;
          }
        }
        //metoda odczytuje wszystkie pakiety danych
        //ale aktualizuje interfejs użytkownika tylko danymi
        //z ostatniego pakietu odczytanego w danej paczce pakietów
        oknoForm1->Invoke(oknoForm1->delegat);
      }
      catch (TimeoutException ^) { }
    }
    return;
  }
}; //Koniec klasy wątku odczytu
```

Listing 5. Kod metody *PortWrite( )*

```
private: System::Void PortWrite(String ^ command){
  int i;
  for (i=0; i<command->Length; i++){
    serialPort->WriteLine(command->Substring(i,1));
  }
}
```

nież, że przy kalibracji brane są pod uwagę odczyty ze wszystkich trzech osi magnetometru mimo, że do dalszych obliczeń związanych z wyznaczaniem azymutu dla kompasu wykorzystywane są tylko osie X i Y. Jak wspomniano, opisana metoda jest dość prosta, jednak do wielu zastosowań daje zadowalające rezultaty. Jeśli konieczna byłaby dokładniejsza kalibracja należałoby zastosować bardziej złożone metody aproksymacji parametrów elipsoidy.

Listing 7. Fragmenty kodu metody *UpdateUI( )* i zmiennych związanych z jej delegacją

```
public: delegate System::Void UpdateUIDelegate();
public: UpdateUIDelegate^ delegat;
public: System::Void UpdateUI() {
  labelZajBuf->Text = zajetoscBufora.ToString();
  labelZajBufP->Text = (zajetoscBufora/19).ToString();
  osieENU=radioButtonENU->Checked;
  //aktualizacja kolejnych obiektów label...
  //wartościami przyśpieszenia i natężenia pola
  labelAXS->Text=daneS[0].ToString();
  //[...]
  labelAXP->Text=danePKor[0].ToString("0.000");
  //[...]
  switch (daneS[6]){ //stan przycisków
    case 1: {panelSW2->BackColor=Color::Green; } break;
    case 2: {panelSW3->BackColor=Color::Green; } break;
    case 3: {panelSW2->BackColor=Color::Green;
             panelSW3->BackColor=Color::Green;} break;
    default : {panelSW2->BackColor=SystemColors::Control;
               panelSW3->BackColor=SystemColors::Control;} break;
  }
  labelAzymut->Text=azymut.ToString("0°");
  pictureBoxRoza->Refresh();
}
```

Jak wiadomo, akcelerometr służy do pomiaru przyśpieszeń, co pozwala pośrednio wyznaczyć także prędkość i zmianę położenia obiektu. Ponadto, może on służyć m.in. do określenia orientacji obiektu względem powierzchni Ziemi. Z kolei magnetometr pozwala określić orientację obiektu względem ziemskich biegunów magnetycznych. Z tego powodu jednym z podstawowych zastosowań, do których przewidziany jest opisywany układ MEMS jest nawigacja i wyznaczanie orientacji przestrzennej urządzenia, w którym się znajduje. W tego typu aplikacjach ważne jest zdefiniowanie wykorzystywanych globalnych i lokalnych układów współrzędnych. W przypadku układów lokalnych, w nawigacji lądowej najczęściej stosuje się układ *East-North-Up* (ENU), w którym oś X wskazuje kierunek wschodni, Y – północny, a Z – w górę. Natomiast w nawigacji morskiej i lotniczej – układ *North-East-Down* (NED) gdzie oś X wskazuje kierunek północny, Y – wschodni, a Z – w dół, czyli do wnętrza Ziemi (**rysunek 4**). W programie demonstracyjnym użytkownik ma możliwość wyboru, według którego układu mają być interpretowane dane. Przyjęto przy tym, że moduł leży płasko na poziomym stole i do określania kierunku wykorzystywane są tylko osie X i Y magnetometru. Gdy przyjmiemy układ ENU i moduł leży wierzchem (stroną z układem i mikrokontrolerem) do góry, wówczas krawędź zaznaczona na fotografii 1 niebieską strzałką pokazuje kierunek wskazywany na ekranie przez igłę kompasu. Gdy z kolei przyjmiemy układ NED i moduł leży stroną spodnią do góry, wówczas kierunek wskazywany przez igłę pokazuje krawędź zaznaczona na fotografii 1 strzałką czerwoną

Ostatnim zadaniem wykonywanym w metodzie *PortRead( )* jest wywołanie, poprzez delegata, metody aktualizującej interfejs użytkownika, tj. wyświetlającej odczytane wartości i rysującej wskaźnik kompasu oraz pola symbolizujące stan przycisków SW2 i SW3. Aktualizacja wykonywana jest przez metodę UpdateUI( ). W systemie Windows dostęp do kontrolek interfejsu ma tylko wątek, w którym zostały one utworzone, i dlatego wątek odczytu nie może tego robić bezpośrednio. Stąd potrzeba delegacji metody UpdateUI( ). Kod tej metody pokazany jest na **listingu 7**. Listingi 7 i 2 zawierają także pozostałe elementy związane z delegowaniem metody.

Na koniec warto zwrócić uwagę, że odczyty danych z bufora w metodzie PortRead( ) rozpoczynają się co około 0,1 s i realizowane są pakietami, czyli po 19 bajtów. Jeśli częstotliwość próbkowania akcelerometru ustawiona była na więcej niż 10Hz, wówczas w buforze znajduje się od 1 do ponad 50 pakietów danych. Metoda odczytuje i konwertuje wszystkie pakiety, ale aktualizuje interfejs użytkownika tylko danymi z ostatniego pakietu odczytanego w danej grupie pakietów. Oznacza to, że mamy dostęp do wszystkich pobranych wyników pomiarów, natomiast prezentowane są tylko niektóre z nich, ponieważ użytkownik i tak nie byłby w stanie odczytać tych wartości przy bardzo częstej ich aktualizacji.

Przedstawiony w artykule program demonstracyjny dedykowany jest dla modułu STEVAL-MKI063V1, jednak inne moduły STEVAL z układami MEMS mają takie same zasady komunikacji i obsługują te same polecenia (choć niektóre z nich mają ich więcej). W związku z tym, program można stosunkowo łatwo zmodyfikować i uzupełnić dopasowując go do innych modułów tej rodziny. Ponadto, przedstawiony program ma zawarte tylko podstawowe zabezpieczenia związane z błędami obsługi portu i odczytu danych. Zaprogramowanie pełnych zabezpieczeń wiązałoby się jednak z rozbudową kodu i mogłoby zaciemnić kluczowe fragmenty związane z obsługą modułu.

Marek Galewski
marg@mech.pg.gda.pl