Barometr MEMS Freescale MPL115A1 SPI

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 15031, pass: 40nep417

Od kilku lat coraz większą popularność zyskują układy MEMS, czyli miniaturowe układy elektro-mechaniczne. Najbardziej typowe zastosowania tej technologii to m.in. akcelerometry, żyroskopy, czujniki ciśnienia oraz przepływomierze, które ze względu na niewielkie wymiary mogą być z łatwością wbudowywane w monitorowane urządzenia. Przykładem tego typu czujnika może być barometr Freescale MPL115A1. W artykule przedstawiono sposób komunikacji z tym czujnikiem oraz konwersji otrzymywanych z niego danych.

Fotografia 1. Moduł KAmod BAR SPI

Układ Freescale MPL115A jest miniaturowym czujnikiem ciśnienia i temperatury wykonanym w technologii MEMS. Ciśnienie mierzone jest w zakresie od 50 do 115 kPa, z dokładnością do 1 kPa. Temperatura powietrza mierzona jest w zakresie od –40 do +105°C, z dokładnością około 0,2°C. Układ produkowany jest w dwóch wersjach. Wersja MPL115A1 wyposażona jest w interfejs SPI, a A2 – w I²C. Wewnątrz układu znajduje się 10-bitowy przetwornik analogowo-cyfrowy o czasie konwersji około 3 ms. Typowe zastosowania opisywanego barometru to m.in.: stacje pogodowe, wysokościomierze, systemy klimatyzacji oraz układy monitorujące pracę innych urządzeń (np. dysków twardych). By ułatwić sobie zapoznanie z opisywanym czujnikiem skorzystać można z oferowanego przez kamami.pl modułu KAmodBAR, którego głównym elementem jest opisywany układ MPL115A. Moduł występuje zarówno w wersji z interfejsem I²C jak i SPI. Drugiej z nich, pokazanej na **fotografii 1**, przyjrzymy się bliżej.

Aby móc połączyć się czujnikiem, trzeba podłączyć go do mikrokontrolera wyposażonego w interfejs SPI. Autor wykorzystał w tym celu zestaw ZL27ARM z mikrokontrolerem STM32F103 z rdzeniem ARM Cortex-M3, a program obsługujący czujnik napisał w języku C z wykorzystaniem bibliotek *STM32F10x_StdPeriphDrv*. Ze względu na uniwersalność języka C oraz powszechność interfejsu SPI, przedstawione fragmenty programu powinien być łatwe do zaimplementowania w dowolnym środowisku programowania lub do przeniesienia na inny rodzaj procesora.

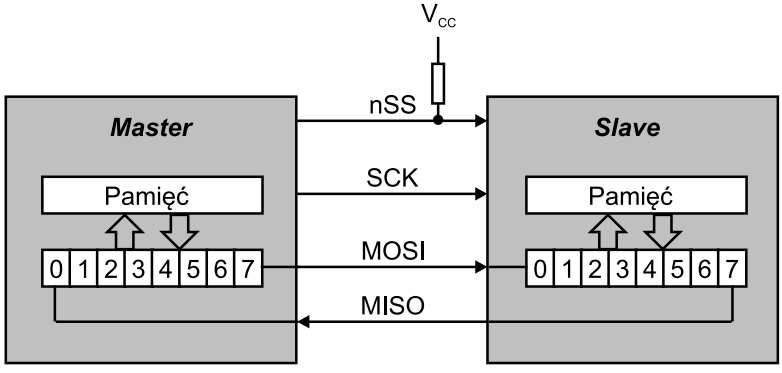
Interfejs SPI wykorzystuje 4 linie: SCK – służącą do przesyłania sygnału zegarowego, MOSI – służącą do przesyłania danych z układu nadrzędnego (*master*) do podrzędnego (*slave*), MISO – służącą do przesyłania danych z układu podrzędnego do nadrzędnego oraz SS – służącą do wskazywania przez układ nadrzędny, z którym układem podrzędnym chce się komunikować. Schemat połączeń pokazany jest na **rysunku 2**. Charakterystyczną cechą interfejsu SPI jest jednoczesna, dwukierunkowa wymiana danych bit po bicie pomiędzy układami. Rejestry nadawcze układów łączone są w tym celu w jeden rejestr przesuwny. Przy kolejnych taktach zegara na linii SCK, bit 7 z układu nadrzędnego trafia na pozycję 0 w rejestrze układu podrzędnego, natomiast 7-my bit z rejestru w układzie podrzędnym trafia na pozycję 0 w rejestrze układu nadrzędnego. Jednocześnie, pozostałe bity są przesuwane o jedna pozycję w prawo. Po wykonaniu ośmiu lub szesnastu (w zależności od długości rejestrów) przesunięć dane z obu układów zostają zamienione miejscami.

W opisywanym układzie, linie interfejsu noszą nieco inne, niż podane wyżej oznaczenia. Linia nSS oznaczana jest jako CS, linia zegarowa SCK oznaczana jest jako SCLK, zaś linie MOSI i MISO noszą oznaczenia DIN i DOUT. Jak już wspomniano, w celu obsługi czujnika, został on podłączony do zestawu ZL27ARM, a do komunikacji został wykorzystany interfejs SPI1 mikrokontrolera STM32, którego linie wyprowadzone są na porcie GPIOA, czyli na złączu JP6 zestawu. Wykonane połączenia przedstawione są w **tabeli 1**. Drobnym utrudnieniem był fakt, że na płytce modułu KAmodBAR, przy złączu Con2 nie ma opisów poszczególnych linii. Warto więc wiedzieć, że pin nr 1 znajduje się na lewo, po stronie napisu „Con2", natomiast pin 6 – na prawo, po stronie kondensatora C1.

Opisywany czujnik jest wyposażony w niewielką pamięć podzieloną na 19 re-

Tabela 1. Wykaz połączeń pomiędzy zestawem ZL27ARM, a modułem KAmodBAR SPI

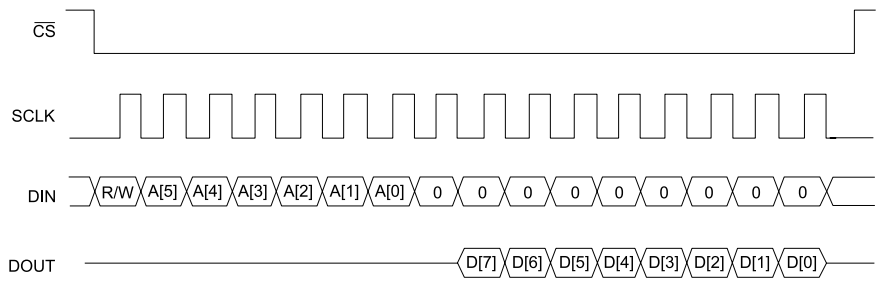
ZL27ARM – JP6 (GPIOA)	KAmod BAR Con2
PA4	Pin 2 – CS (SS)
PA5	Pin 5 – SCLK (SCK)
PA6	Pin 4 – DOUT (MISO)
PA7	Pin 3 – DIN (MOSI)
GND	Pin 6 – GND
+3,3V	Pin 1 – +V



Rysunek 2. Połączenie układów interfejsem SPI

Tabela 2. Mapa rejestrów układu MPL115A

Adres rejestru	Nazwa rejestru	Opis rejestru
$00	POUTH	Starszy bajt wartości ciśnienia
$01	POUTL	Młodszy bajt wartości ciśnienia
$02	TOUTH	Starszy bajt wartości temperatury
$03	TOUTL	Młodszy bajt wartości temperatury
$04	COEF1	Współczynnik korekcyjny a_0 – starszy bajt
$05	COEF2	Współczynnik korekcyjny a_0 – młodszy bajt
$06	COEF3	Współczynnik korekcyjny b_1 – starszy bajt
$07	COEF4	Współczynnik korekcyjny b_1 – młodszy bajt
$08	COEF5	Współczynnik korekcyjny b_2 – starszy bajt
$09	COEF6	Współczynnik korekcyjny b_2 – młodszy bajt
$0A	COEF7	Współczynnik korekcyjny c_{12} – starszy bajt
$0B	COEF8	Współczynnik korekcyjny c_{12} – młodszy bajt
$0C	COEF9	Współczynnik korekcyjny c_{11} – starszy bajt
$0D	COEF10	Współczynnik korekcyjny c_{11} – młodszy bajt
$0E	COEF11	Współczynnik korekcyjny c_{22} – starszy bajt
$0F	COEF12	Współczynnik korekcyjny c_{22} – młodszy bajt
$10	PRESS	Start konwersji ciśnienia
$11	TEMP	Start konwersji temperatury
$12	BOTH	Start konwersji obu wielkości



Rysunek 3. Schemat komunikacji z układem MPL115A1

jestrów (**tabela 2**). Jeżeli chcemy wykonać pomiar, należy wysłać komendę (bajt o wartości 0) do rejestru PRESS, TEMP lub BOTH. Wywołana zostaje w ten sposób konwersja, odpowiednio, ciśnienia, temperatury lub obu tych wielkości. Przed odczytem wyników należy odczekać co najmniej 3 ms co wynika z szybkości działania przetwornika. Ponadto, układ wymaga, aby pomiędzy wysłaniem adresu rejestru, a odczytem jego wartości wystąpiły co najmniej 32 ns przerwy.

Zmierzone wartości odpowiadające temperaturze i ciśnieniu odczytuje się z rejestrów TOUTH (starsze 8 bitów temperatury), TOUTL (najmłodsze 2 bity temperatury), POUTH (starsze 8 bitów ciśnienia) i POUTL (najmłodsze 2 bity ciśnienia). Ponieważ wszystkie informacje odczytywane z układu zajmują od dziesięciu do szesnastu bitów oznacza to, że w celu odczytu każdej wartości konieczny jest odczyt dwóch rejestrów, a następnie połączenie ich zawartości. Na przykład w przypadku odczytu temperatury należy pobrać zawartość rejestru TOUTH, przesunąć ją o 8 bitów w lewo, połączyć z zawartością rejestru TOUTL i całość przesunąć o 6 bitów prawo. W rezultacie otrzymamy 10-bitową wartość temperatury T_{ADC} wyznaczoną przez przetwornik A/C.

Wysłanie poszczególnych bajtów komendy i danych przedstawia schemat pokazany na **rysunku 3**. Wynika z niego, że gdy transmisja się nie odbywa, linia SCLK powinna być w stanie niskim, a aktywnym zboczem sygnału zegarowego powinno być zbocze pierwsze, czyli narastające. Kontroler SPI po stronie mikrokontrolera powinien pracować w trybie *master*, a długość słowa danych powinna być ustawiona na 8 bitów. Podczas komunikacji z układem, najpierw wysyła się bit kierunku transmisji danych (z lub do rejestru układu) i 6 bitów adresu wewnętrznego rejestru, z którego chcemy wartość odczytać lub, do którego chcemy ją zapisać. Ponieważ mikrokontroler musi wysłać 8 bitów słowa, konieczne jest dodatkowe wysłanie bitu o wartości 0. Oznacza to, że przed wysyłaniem adres trzeba po prostu przesunąć w lewo o jeden bit. Wówczas 7 pierwszych (starszych) przesyłanych bitów jest zgodnych z wymaganiami układu MPL115, zaś dodatkowy, zerowy bit układ zignoruje. Następnie przesyła się bajt o wartości 0. W przypadku, gdy odczytujemy zawartość któregoś z rejestrów, bajt ten potrzebny jest tylko ze względu na sposób działania interfejsu SPI i łączy się z koniecznością wykonania transmisji jednocześnie w obu kierunkach.

Procedurę transmisji jednego bajtu realizuje funkcja *SPI_Transmit( )* przedstawiona na **listingu 1**. Natomiast funkcja *MPL115_ReadTemp( )* z **listingu 2** pokazuje, w jaki sposób można odczytać temperaturę zmierzoną przez układ.

Wykorzystane w funkcjach stałe są zdefiniowane następująco:

```
//interfejs
#define MPL115_SPI  SPI1
//port GPIO
#define MPL115_GPIO GPIOA
//SPI SS
#define MPL115_CS  GPIO_Pin_4
//SPI MOSI
```

Listing 1. Funkcja *SPI_Transmit( )*

```
char SPI_Transmit(char cData)
{ //Transmisja jednego bajtu po SPI
  volatile short int i;
  SPI_I2S_SendData(MPL115_SPI, cData);
  while (SPI_I2S_GetFlagStatus(MPL115_SPI, SPI_I2S_FLAG_RXNE) == RESET);
  for (i=0; i<20; i++);
  return SPI_I2S_ReceiveData(MPL115_SPI);
}
```

List. 2. Funkcja *MPL115_ReadTemp( )*

```
void MPL115_ReadTemp(long int * RawTemp)
{ //odczyt temperatury
  short int val1, val2;
  volatile long int i;

  SPI_CS_Enable();
  SPI_Transmit(MPL115_WRITE|(MPL115_TEMP<<1));
  SPI_Transmit(0x00);
  SPI_CS_Disable();

  for (i=0; i<20000; i++);  //3ms przerwy na wykonanie konwersji

  SPI_CS_Enable();
  SPI_Transmit(MPL115_READ|(MPL115_TOUTH<<1));
  val1=SPI_Transmit(0x0);
  val1=(val1<<8);
  SPI_Transmit(MPL115_READ|(MPL115_TOUTL<<1));
  val2=SPI_Transmit(0x0);
  val1=val1 | val2;
  val1=(val1>>6);
  * RawTemp =val1;
  SPI_CS_Disable();
}
```

Listing 3. Definicja typu *sCoeffs*
```
typedef struct {
  signed short int sia0;
  signed short int sib1;
  signed short int sib2;
  signed short int sic12;
  signed short int sic11;
  signed short int sic22;
} sCoeffs;
```

```
#define MPL115_SDI GPIO_Pin_7
//SPI MISO
#define MPL115_SDO GPIO_Pin_6
//SPI SCL
#define MPL115_CLK GPIO_Pin_5
//zapis danych
#define MPL115_WRITE    0x00
//odczyt danych
#define MPL115_READ     0x80
//MSB cisnienia
#define MPL115_POUTH    0x00
//LSB cisnienia
#define MPL115_POUTL    0x01
//MSB temperatury
#define MPL115_TOUTH    0x02
//LSB temperatury
#define MPL115_TOUTL    0x03
```

Występujące w listingach funkcje *SPI_CS_Enable( )* i *SPI_CS_Disable( )* sterują stanem linii CS. Warto zauważyć, że układ wymaga, by czas pomiędzy aktywacją linii CS, a rozpoczęciem transmisji pierwszego bitu wynosił co najmniej 125 ns. Konieczne więc było umieszczenie w funkcji *SPI_CS_Enable( )* krótkiej pętli opóźniającej.

Mierzona wartość ciśnienia atmosferycznego zależna jest od temperatury i wysokości nad poziomem morza, na której znajduje się czujnik. Aby możliwe było wyznaczenie ciśnienia na określonej wysokości, konieczne jest skorygowanie odczytu z uwzględnieniem pomiaru temperatury. Stąd obecność w barometrze także termometru. Ponadto, ponieważ charakterystyki układu nie są idealnie liniowe, barometr jest fabrycznie kalibrowany, a odpowiednie współczynniki wielomianu korekcyjnego potrzebne do wyznaczenia poprawnej wartości ciśnienia zapisane są na stałe w każdym układzie. Każdy ze współczynników jest liczbą 16-bitową, stałoprzecinkową i zajmuje dwa rejestry COEFx. Aby ułatwić odczyt i operowanie na współczynnikach można zdefiniować typ strukturalny *sCoeffs* (**listing 3**).

Do odczytu współczynników i ich umieszczenia w zmiennej strukturalnej typu *sCoeffs* służy funkcja *MPL115_ReadCoeffs( )* pokazana na **listingu 4**. Wykorzystuje ona wewnętrznie funkcję *MPL115_ReadCoeffParts( )* pokazaną na **listingu 5**. Funkcja ta przyjmuje jako parametry wywołania adresy dwóch, 8-bitowych rejestrów, w których znajdują się poszczególne części współczynnika, po czym zwraca jego wartość jako liczbę 16-bitową.

Do obliczenia ciśnienia na podstawie pomiaru konieczne jest więc nie tylko wykonanie pomiarów ciśnienia i temperatury, ale

Listing 4. Funkcja *MPL115_ReadTemp()*
```
void MPL115_ReadCoeffs(sCoeffs * Coeffs)
{ //Odczyt wspolczynnikow do obliczenia cisnienia
  Coeffs->sia0  = MPL115_ReadCoeffParts(MPL115_COEF1,  MPL115_COEF2);
  Coeffs->sib1  = MPL115_ReadCoeffParts(MPL115_COEF3,  MPL115_COEF4);
  Coeffs->sib2  = MPL115_ReadCoeffParts(MPL115_COEF5,  MPL115_COEF6);
  Coeffs->sic12 = MPL115_ReadCoeffParts(MPL115_COEF7,  MPL115_COEF8);
  Coeffs->sic11 = MPL115_ReadCoeffParts(MPL115_COEF9,  MPL115_COEF10);
  Coeffs->sic22 = MPL115_ReadCoeffParts(MPL115_COEF11, MPL115_COEF12);
}
```

Listing 5. Funkcja *MPL115_ReadCoeffParts()*
```
signed int MPL115_ReadCoeffParts(unsigned char MSBaddr,
                                 unsigned char LSBaddr)
{ //Odczyt dwoch czesci wspolczynnika i polaczenie ich w jedna wartosc
  signed int retVal;
  signed char valMSB, valLSB;
  SPI_CS_Enable();
  SPI_Transmit(MPL115_READ|(MSBaddr<<1));
  valMSB=SPI_Transmit(0x0);
  SPI_CS_Disable();
  SPI_CS_Enable();
  SPI_Transmit(MPL115_READ|(LSBaddr<<1));
  valLSB=SPI_Transmit(0x0);
  SPI_CS_Disable();
  retVal = (signed int) valMSB << 8;
  retVal += (signed int) valLSB & 0x00FF;
  return retVal;
}
```

Tabela 3. Formaty zapisu współczynników korekcyjnych

	Współczynnik korekcyjny					
	a_0	b_1	b_2	c_{12}	c_{11}	c_{22}
Całkowita liczba bitów	16	16	16	14	12	11
Bit znaku	1	1	1	1	1	1
Liczba bitów cz. całkowitej	12	2	1	0	0	0
Liczba bitów cz. ułamkowej	4	13	14	13	11	10
Pozycja separatora części ułamkowej	-	-	-	9	11	15

Listing 6. Funkcja *MPL115_PressCalc()*
```
void MPL115_PressCalc(long int * CalcPress,
                      unsigned short int * RawTemp,
                      unsigned short int * RawPress,
                      sCoeffs * Coeffs)
{ //Obliczanie ciśnienia
  long int decPcomp;
  signed long lt1, lt2, lt3, si_c11x1, si_a11, si_c12x2;
  signed long si_a1, si_c22x2, si_a2, si_a1x1, si_y1, si_a2x2;
  unsigned int uiPadc, uiTadc;
  uiPadc=*RawPress;
  uiTadc=*RawTemp;
  // Krok 1 c11x1 = c11 * Padc
  t1 = (signed long) Coeffs->sic11;
  lt2 = (signed long) uiPadc;
  lt3 = lt1*lt2;
  si_c11x1 = (signed long) lt3;
  // Krok 2 a11 = b1 + c11x1
  lt1 = ((signed long) Coeffs->sib1)<<14;
  lt2 = (signed long) si_c11x1;
  lt3 = lt1 + lt2;
  si_a11 = (signed long)(lt3>>14);
  // Krok 3 c12x2 = c12 * Tadc
  lt1 = (signed long) Coeffs->sic12;
  lt2 = (signed long) uiTadc;
  lt3 = lt1*lt2;
  si_c12x2 = (signed long)lt3;
  // Krok 4 a1 = a11 + c12x2
  lt1 = ((signed long)si_a11<<11);
  lt2 = (signed long)si_c12x2;
  lt3 = lt1 + lt2;
  si_a1 = (signed long) lt3>>11;
  // Krok 5 c22x2 = c22*Tadc
  lt1 = (signed long) Coeffs->sic22;
  lt2 = (signed long)uiTadc;
  lt3 = lt1 * lt2;
  si_c22x2 = (signed long)(lt3);
  // Krok 6 a2 = b2 + c22x2
  lt1 = ((signed long) Coeffs->sib2<<15);
  lt2 = ((signed long)si_c22x2>1);
  lt3 = lt1+lt2;
  si_a2 = ((signed long)lt3>>16);
  // Krok 7 a1x1 = a1 * Padc
  lt1 = (signed long)si_a1;
  lt2 = (signed long)uiPadc;
  lt3 = lt1*lt2;
  si_a1x1 = (signed long)(lt3);
  // Krok 8 y1 = a0 + a1x1
  lt1 = ((signed long) Coeffs->sia0<<10);
  lt2 = (signed long)si_a1x1;
  lt3 = lt1+lt2;
```

Listing 6. c.d.
```
  si_y1 = ((signed long)lt3>>10);
  // Krok 9 a2x2 = a2 * Tadc
  lt1 = (signed long)si_a2;
  lt2 = (signed long)uiTadc;
  lt3 = lt1*lt2;
  si_a2x2 = (signed long)(lt3);
  // Krok 10 pComp = y1 + a2x2
  lt1 = ((signed long)si_y1<<10);
  lt2 = (signed long)si_a2x2;
  lt3 = lt1+lt2;
  //Konwersja na rzeczywista wartosc cisnienia
  //zaokraglona do pelnego hPa
  decPcomp = ((650*((signed long)lt3/8192))/1023)+500;
  *CalcPress=decPcomp;
}
```

Listing 7. Deklaracje zmiennych wykorzystywanych w listingu 8
```
  unsigned short int Temperatura, Cisnienie;
  sCoeffs  Wspolczynniki;
  long int TempObliczona, CisObliczone;
  unsigned char Tekst[17] = {"\0"};
```

Listing 8. Fragment programu wyświetlający zmierzone temperaturę i ciśnienie
```
  //Pobierz z układu MPL115 współczynniki wielomianu
  //do obliczania ciśnienia
  MPL115_ReadCoeffs(&Wspolczynniki);
  //odczytaj „surowe" temperaturę i ciśnienie
  MPL115_ReadPressTemp(&Temperatura, &Cisnienie);
  //korekta temperatury do realnych wartości,
  //uwaga! niezgodnie z nota aplikacyjna
  Temperatura-=38;
  //Oblicz temperaturę
  TempObliczona=(2500 -((Temperatura - 472)*10000 / 535))/10;
  //Oblicz ciśnienie
  MPL115_PressCalc(&CisObliczone, &Temperatura,
                   &Cisnienie,    &Wspolczynniki);
  //wyświetl wynik na LCD 16x2
  sprintf((char *)Tekst, "Temp.: %2d,%d stC\0",
          TempObliczona/10, abs(TempObliczona%10));
  LCD_WriteTextXY(Tekst,0,0);
  sprintf((char *)Tekst, "Cisn.: %4d hPa\0", CisObliczone);
  LCD_WriteTextXY(Tekst,0,1);
```

też odczytanie współczynników korekcyjnych. Do wyznaczenia prawidłowej wartości ciśnienia wykorzystuje się dwie zależności. Pierwsza z nich pozwala obliczyć skorygowaną, całkowitoliczbową wartość ciśnienia P_{kor}:

$$P_{kor} = a_0 + \left(b_1 + c_{11} \cdot P_{ADC} + c_{12} \cdot T_{ADC}\right) \cdot P_{ADC} + \left(b_2 + c_{22} \cdot T_{ADC}\right) \cdot T_{ADC}$$

gdzie P_{ADC} i T_{ADC} to zmierzone, „surowe", 10-bitowe wartości ciśnienia i temperatury, zaś a_0, b_1, b_2, c_{11}, c_{12} i c_{22} to współczynniki wielomianu odczytywane z pamięci układu. W praktyce, ponieważ współczynniki są stałe, wystarczy odczytać je raz, np. na początku programu. Ciśnieniu 50 kPa powinna odpowiadać wartość P_{kor} = 0, natomiast ciśnieniu 115 kPa wartość P_{kor} = 1023. Stąd wynika, że do wyznaczenia faktycznej wartości ciśnienia P konieczne jest zastosowanie dodatkowej zależności:

$$P[kPa] = \frac{115-50}{1023} \cdot P_{kor} + 50$$

Poszczególne wartości współczynników wielomianu, choć zajmują dwa rejestry, to zapisane są w różnych formatach, zestawionych w **tabeli 3**. Na przykład współczynnik a_0 zapisany jest jako bit znaku, 12 bitów części całkowitej i 3 bity części ułamkowej ($ZI_{11}I_{10}I_9I_8I_7I_6I_5I_4I_3I_2I_1I_0,F_2F_1F_0$) natomiast współczynnik c_{12} zapisany jest jako bit znaku i 13 bitów części ułamkowej dodatkowo przesuniętej o 9 pozycji w prawo ($Z,0000000 00I_{12}I_{11}I_{10}I_9I_8I_7I_6I_5I_4I_3I_2I_1I_0$).

Niejednolity zapis poszczególnych współczynników komplikuje wykonywanie obliczeń. Na szczęście nota aplikacyjna układu zawiera algorytm realizujący odpowiednie operacje, co znacznie ułatwia tworzenie własnego programu. Na **listingu 6** pokazana jest funkcja *MPL115_PressCalc( )* obliczająca wartość ciśnienia *P*. Jej parametrami są wskaźniki do zmiennych zawierających wynik obliczeń ciśnienia, „surowych" wartości temperatury i ciśnienia oraz do struktury zawierającej współczynniki korekcyjne.

Mierzona przez układ MPL115 „surowa" wartość temperatury T_{ADC} wykorzystywana jest do obliczenia rzeczywistego ciśnienia. Można ją jednak także przeliczyć na rzeczywistą wartość temperatury *T*. Zgodnie z notą katalogową, w temperaturze 25°C odczytana wartość T_{ADC} powinna wynosić 472. Jednocześnie wzrostowi temperatury o 1°C odpowiada zmiana wartości T_{ADC} o –5,35. Temperaturę *T* wyznaczmy więc korzystając z zależności:

$$T = 25°C - \frac{\left(T_{ADC} - 472\right)}{5{,}35\,\frac{1}{°C}}$$

Mając już wszystkie funkcje potrzebne do wykonania obliczeń można napisać główną część programu, realizującą np. wykonanie pomiaru i wyświetlenie jego rezultatów na wyświetlaczu (**listingi 7** i **8**).

Na koniec warto dodać, że wykorzystany przez autora egzemplarz układu MPL115 podawał zaniżony o kilka stopni pomiar temperatury. Jak się okazało po przejrzeniu forów internetowych, także inni użytkownicy tego układu niekiedy napotykali ten problem. Można mu łatwo zaradzić korygując programowo odczytaną wartość temperatury jeszcze przed wyznaczeniem ciśnienia. Gdy uda się już napisać całość programu obsługi czujnika, warto porównać odczytywane wartości z innymi źródłami (np. stacjami pogodowymi, termometrami itp.). W internecie znaleźć można również aktualne pomiary z różnych stacji pogodowych rozsianych po całym świecie, jest więc szansa na znalezienie takiej, która znajduje się gdzieś w pobliżu. Przy porównywaniu podawanego ciśnienia należy przy tym zwrócić uwagę, czy jest to ciśnienie lokalne, czy odniesione do poziomu morza. Warto pamiętać, że ciśnienie spada w przybliżeniu o około 11,5 hPa na każde 100 m wzrostu wysokości nad poziom morza.

Marek Galewski
marg@mech.pg.gda.pl