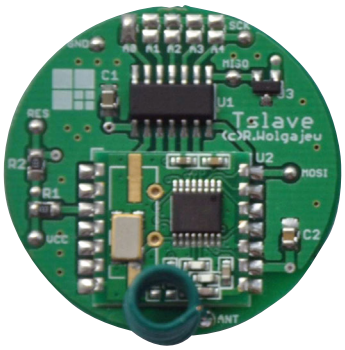


Autor składa podziękowania panu Sławomirowi Szwedzie z firmy Unisystem za dostarczenie wyświetlacza OLED znajdującego zastosowanie w niniejszym projekcie.

Dodatkowe materiały do pobrania ze strony www.media.avt.pl

W ofercie AVT* AVT-5635

Projekty pokrewne na www.media.avt.pl:

AVT-5623	4-kanałowy termometr z interfejsem Wi-Fi (EP 4/2018
AVT-5566	THPStation – rozbudowany termometr z Wi-Fi (EP 1/2017
AVT-1941	8-kanałowy termometr z I²C (EP 1/2017
AVT-5573	Nieskomplikowany termometr-rejestrator (EP 11/2016
AVT-5535	Termometr 2-kanałowy z interfejsem Bluetooth (EP 4/2016
AVT-5518	Termometr bezprzewodowy (EP 11/2015
AVT-1863	Termometr z interfejsem Bluetooth (EP 8/2015
AVT-1790	Termometr XXL (EP 2/2014
AVT-5489	8-kanałowy termometr z alarmem i wyświetlaczem LCD (EP 11/2013
AVT-5420	Wielopunktowy termometr z rejestracją (EP 10/2013
AVT-1734	Termometr do wędzarni (EP 4/2013
AVT-5373	Tlogger – rejestrator temperatury (EP 12/2012
AVT-1705	Moduł do pomiaru temperatury z interfejsem RS485 (EP 9/2012
AVT-1697	Wielogabarytowy termometr LED (EP 8/2012
AVT-5389	4-kanałowy termometr z wyświetlaczem LED (EP 5/1012
AVT-5330	Termometr PC (EP 2/2012
AVT-5301	Wskaźnik komfortu cieplnego z wbudowanym kalendarzem sezonowym (EP 7/2011
AVT-1582	Domowy termometr RGB (EP 8/2010
AVT-5230	Rejestrator temperatury z interfejsem USB (EP 4/2010
AVT-5205	System pomiaru temperatury z termoparą typu K (EP 10/2009
AVT-5117	Termometr USB (EP 11/2007
AVT-5108	2-kanałowy termometr z dwukolorowym wyświetlaczem LED (EP 8/2007
AVT-957	Moduł pomiaru temperatury (EP 11/2006
AVT-2787	PC – Termometr – termometr internetowy (EdW 5/2006
AVT-918	Termometr z termoparami J albo K (EP 2/2006
AVT-5041	Termometr MIN-MAX (EP 11/2001

* **Uwaga!** **Elektroniczne zestawy do samodzielnego montażu. Wymagana umiejętność lutowania!**
Podstawową wersją zestawu jest wersja [**B**] nazywana potocznie KITem (z ang. zestaw). Zestaw w wersji [**B**] zawiera elementy elektroniczne (w tym [**UK**] – jeśli występuje w projekcie), które należy samodzielnie wlutować w dołączoną płytkę drukowaną (PCB). Wykaz elementów znajduje się w dokumentacji, która jest podlinkowana w opisie kitu.
Mając na uwadze różne potrzeby naszych klientów, oferujemy dodatkowe wersje:
- wersja [**C**] zmontowany, uruchomiony i przetestowany zestaw [**B**] (elementy wlutowane w płytkę PCB)
- wersja [**A**] płytka drukowana bez elementów i dokumentacja

Kity w których występuje układ scalony wymagający zaprogramowania, posiadają następujące dodatkowe wersje:
- wersja [**A+**] płytka drukowana [**A**] + zaprogramowany układ [**UK**] i dokumentacja
- wersja [**UK**] zaprogramowany układ

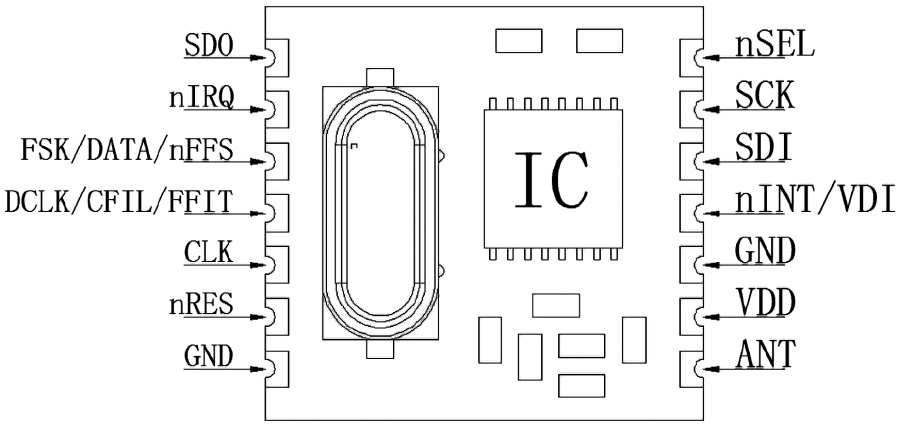
Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik pdf! Podczas składania zamówienia upewnij się, którą wersję zamawiasz!
http://sklep.avt.pl. W przypadku braku dostępności na ***http://sklep.avt.pl***, osoby zainteresowane zakupem płytek drukowanych (PCB), prosimy o kontakt via email: kity@avt.pl.

Bezprzewodowy, energooszczędny system pomiaru temperatury (1)

Stanąłem przed wyzwaniem skonstruowania prostego systemu bezprzewodowego pomiaru temperatury, który, po pierwsze, nie wymagałby żadnej konfiguracji, a po drugie, odznaczałby się walorem w postaci małego poboru energii przez węzły pomiarowe. Zależało mi też na użyciu łatwo dostępnych, tanich modułów radiowych mających tryb oszczędzania energii. Prezentowane urządzenie opracowałem w efekcie tak sformułowanych wymagań.

Rekomendacje: *urządzenie może przydać się w domu, magazynie, szklarni – wszędzie tam, gdzie jest niezbędny pomiar temperatury w wielu miejscach.*

Prace konstrukcyjne rozpocząłem od poszukiwania takich modułów radiowych i dość szybko natknąłem się na bardzo popularne układy RFM-12B pracujące w paśmie 433, 868 lub 915 MHz (w zależności od wersji) będące przedstawicielem całej rodziny modułów radiowych produkowanych przez firmę HopeRF. Moduły, o których mowa stanowią kompletne rozwiązanie toru radiowego nadawczo-odbiorczego dostarczając wygodny interfejs komunikacyjny SPI pozwalający na przeprowadzenie pełnej konfiguracji elementu w ramach dostępnej szerokiej palety ustawień jak i sterowanie komunikacją radiową. Na **rysunku 1** pokazano wygląd



Rysunek 1. Wygląd płytki drukowanej modułu RFM-12B w wersji SMD (na podstawie dokumentacji producenta)

Tabela 1. Rozmieszczenie i opis wybranych wyprowadzeń modułu RFM-12B

Nazwa wyprowadzenia	Opis
nINT/VDI	Wejście przerwania zewnętrznego lub VDI (wskaźnik nadejścia danych)
VDD	Napięcie zasilania, maksymalnie 3,8 V
SDI	Wejście interfejsu SPI
SCK	Przebieg zegarowy interfejsu SPI
nSEL	Wejście Slave Select interfejsu SPI
SDO	Wyjście interfejsu SPI
nIRQ	Wyjście zgłoszenia przerwania (aktywny poziom niski)
CLK	Opcjonalny przebieg taktujący dla zewnętrznego mikrokontrolera
nRES	Wejście zerowania modułu (aktywny stan niski)
GND	Masa zasilania
ANT	Wyjście antenowe

Listing 1. Przykładowy kod inicjalizacji modułu RFM-12B znaleziony w Internecie

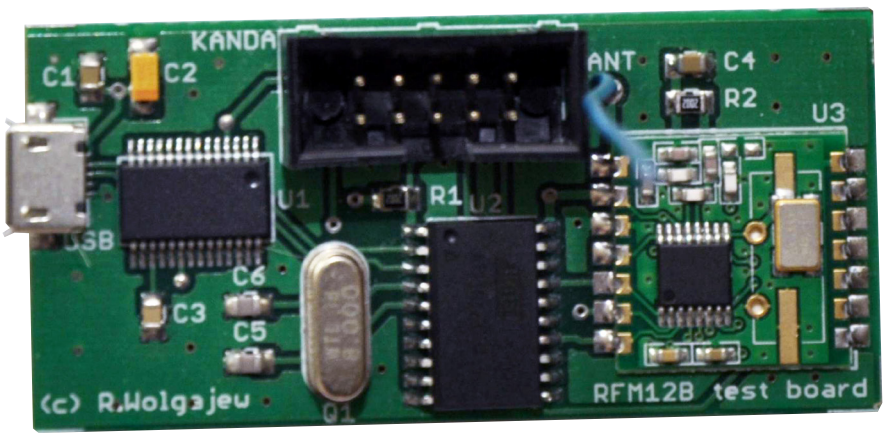
```
WriteCMD(0x80D8);   //enable register,433MHz,12.5pF
WriteCMD(0x8208);   //Turn on crystal,!PA
WriteCMD(0xA640);
WriteCMD(0xC647);
WriteCMD(0XCC77);
WriteCMD(0x94A0);   //VDI,FAST,134kHz,0dBm,-103dBm
WriteCMD(0xC2AC);
WriteCMD(0xCA80);
WriteCMD(0xCA83);   //FIFO8,SYNC,
WriteCMD(0xC49B);
WriteCMD(0x9850);   //!mp,9810=30kHz,MAX OUT
WriteCMD(0xE000);   //NOT USE
WriteCMD(0xC80E);   //NOT USE
WriteCMD(0xC000);   //1.0MHz,2.2V
```

płytki drukowanej modułu w wersji SMD, natomiast w **tabeli 1** rozmieszczenie tych wyprowadzeń, które są używane w opisywanej aplikacji.

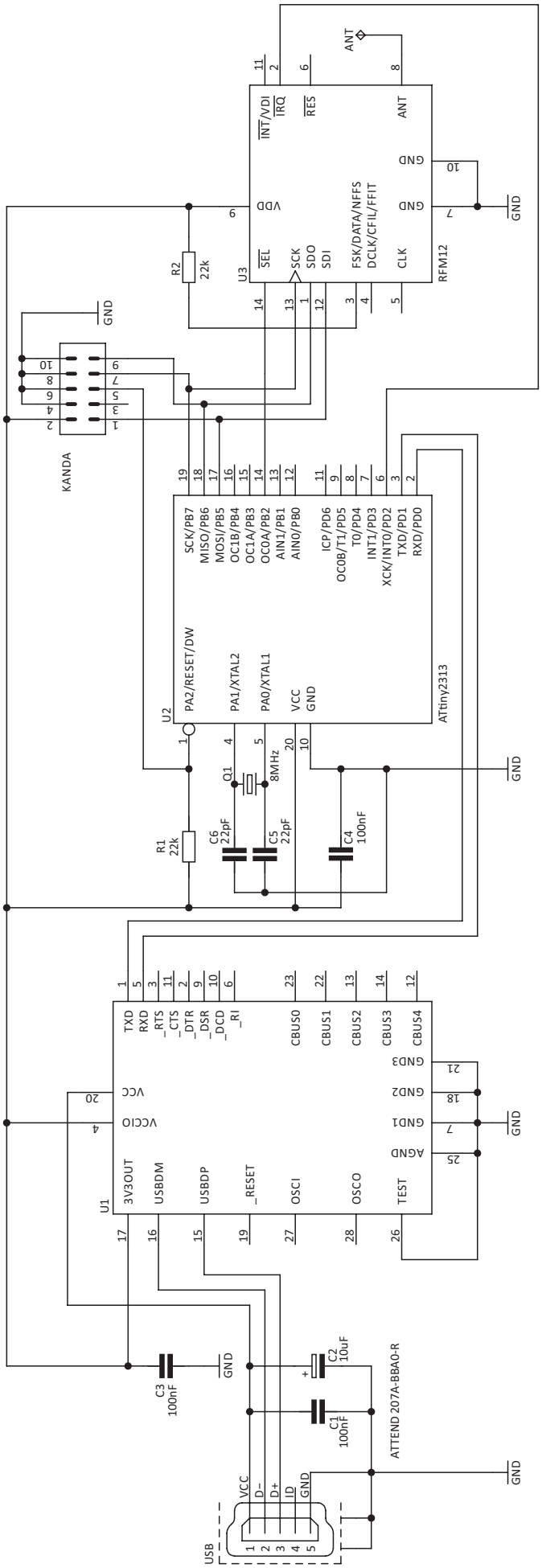
Przeglądając Internet natknąłem się na wiele przykładów obsługi wspomnianych modułów, jednak za każdym razem było to powielanie tej samej, dość niezrozumiałej konfiguracji, która z przekazów Internautów lepiej lub gorzej sprawdzała się w aplikacji. Przyczyną tego stanu rzeczy jest z jednej strony dość obszerna lista ustawień modułu, a z drugiej ogólnikowa i nieprecyzyjna dokumentacja producenta. Przykładową konfigurację modułu zamieszczono na **listingu 1**. O ile z załączonych komentarzy możemy mniej więcej zorientować się, jakiego rodzaju ustawienia są dokonywane, o tyle wprowadzane wartości liczbowe nie pozwalają na ustalenie, w jaki sposób należy je zmienić, by dokonać stosownych korekt.

Po lekturze dostępnych rozwiązań programowych oraz niekompletnej dokumentacji producenta (w tym przykładowych kodów obsługi) spodziewałem się pewnych problemów podczas uruchamiania transmisji z użyciem modułu. Zdecydowałem się na dość nietypowy krok, a mianowicie na wykonanie systemu deweloperskiego pozwalającego na łatwe debugowanie kodu, który umożliwiłby wygodne zapoznanie się z możliwościami naszego bohatera. Schemat urządzenia, o którym mowa pokazano na **rysunku 3**. Zaprojektowano nieskomplikowany system procesorowy, którego „sercem" jest mikrokontroler ATtiny2313 sterujący pracą modułu transceivera, zaś przy pomocy układu FT232RL, realizujący transmisję USB↔USART pozwalający, po pierwsze, na zasilanie naszego systemu bezpośrednio z gniazda micro-USB, zaś po drugie i najważniejsze, dający możliwość debugowania kodu, dzięki transmisji, którą możemy realizować poprzez wspomniany interfejs i prosty program terminalowy. Muszę z satysfakcją przyznać, że dzięki tak zbudowanemu systemowi deweloperskiemu proces uruchamiania modułów RFM-12B oraz pisania oprogramowania był czystą przyjemnością, dlatego polecam jego wykonanie. Warto również zauważyć, że system ten wyposażono w gniazdo programowania standardu KANDA przez co ułatwiono samo programowanie mikrokontrolera sterującego. Widok obwodu drukowanego zmontowanego systemu deweloperskiego pokazano na **fotografii 2**.

Wróćmy zatem do naszego bohatera, czyli modułu transceivera. Jak wspomniałem we wstępie, moduły RFM-12B wyposażono w szereg 16-bitowych rejestrów konfiguracyjnych, za których pomocą (i dzięki interfejsowi SPI) możemy dokonać stosownych ustawień konfiguracyjnych oraz zarządzać transmisją danych. Co więcej, moduł ma bufory nadawczo-odbiorcze oraz generuje stosowne przerwania, dzięki którym w dość łatwy sposób możemy zarządzać transmisją danych. Wprowadzono natomiast pewne wymagania dotyczące budowy ramki danych, dzięki którym zautomatyzujemy proces ich odbierania. Taką, przykładową ramkę danych pokazano na **rysunku 4**.



Fotografia 2. Widok zmontowanego systemu deweloperskiego dla modułów RFM-12B



Rysunek 3. Schemat ideowy systemu deweloperskiego dla modułów RFM-12B

Jak widać, na początku należy wysłać należy 3 bajty preambuły o wartościach 0xAA (ewentualnie 0x55), których zadaniem jest dopasowanie wzmocnienia wejściowych układów RF po stronie odbiornika. Następnie wysyłamy 2 bajty synchronizacyjne o wartościach 0x2D i 0xD4, przy czym wartość drugiego bajta możemy zmieniać w ramach konfiguracji modułu. Służą one do synchronizacji mechanizmu odbierania danych po stronie odbiornika tzn. odbiornik po odebraniu tychże bajtów zaczyna wypełniać swój bufor odbiorczy (FIFO) kolejnymi bajtami przesyłanymi po bajtach synchronizacji, zgłaszając każdorazowo stosowne przerwanie (ściągając wyprowadzenie nIRQ do logicznego 0). Należy jednak zaznaczyć, iż jest to jeden z możliwych trybów pracy odbiornika (nazywany w dokumentacji mechanizmem FIFO), ponieważ może też zgłaszać odebranie każdego bajta danych (i napełniać bufor odbiorczy), w tym bajtów synchronizacji. Ustawienie to, jak wiele innych, podlega konfiguracji, jednak zdecydowanie polecam standardową metodę obsługi ramki danych wykorzystującą mechanizm synchronizacji.

Po bajtach synchronizacji przesłać należy użyteczne dane, a na samym końcu kilka (u nas 2) bajty postambuły o wartościach 0xAA (ewentualnie 0x55). Zgodnie z tym, co napisałem wcześniej, wartość jednego z bajtów synchronizacji, o wartości 0xD4 podlega konfiguracji, przez co może on niejako stanowić adres sprzętowy modułu, gdyż ten moduł zacznie zgłaszać przerwania towarzyszące odbieraniu danych użytecznych dopiero po otrzymaniu dwóch bajtów synchronizacji, w tym tego podlegającego konfiguracji. W ten sposób zaimplementujemy mechanizm sprzętowej adresacji modułów. Ponadto, aby zautomatyzować sposób odbioru pakietów danych, w ramach procedury obsługi przerwania zewnętrznego od wyprowadzenia nIRQ modułu, wprowadzimy dodatkowe pola w naszej ramce danych – ramkę po modyfikacji pokazano na **rysunku 5**. Wprowadzono do niej dodatkowy bajt *Size* określający rozmiar przesyłanej ramki danych (danych „użytecznych"), dzięki czemu, co zobaczycie później, w łatwy sposób ustalimy koniec ramki transmisji zgłaszając odebranie nowej ramki danych do programu głównego aplikacji. Dodatkowo, nasza ramka danych zawiera też bajt sumy kontrolnej CRC8, który umożliwi sprawdzenie poprawności danych. Wróćmy

Rysunek 4. Przykładowa ramka danych modułu RFM-12B

Rysunek 5. Zmodyfikowana ramka danych modułu RFM-12B

Listing 2. Plik nagłówkowy zawierający definicje do obsługi modułu RFM-12B

```
#define BUFFER_SIZE 32  //Rozmiar bufora nadawczo-odbiorczego
//Definicje portów sterujących
#define MOSI_PORT PORTB
#define MOSI_DDR DDRB
#define MOSI_NR PB3
#define MISO_PORT PORTB
#define MISO_PIN PINB
#define MISO_NR PB1
#define SCK_PORT PORTB
#define SCK_DDR DDRB
#define SCK_NR PB4
#define SS_PORT PORTB
#define SS_DDR DDRB
#define SS_NR PB5
//Definicje portu przerwania
#define RFM12_IRQ_PORT PORTB
#define RFM12_IRQ_PIN PINB
#define RFM12_IRQ_NR PB2
#define RFM12_IRQ_NAME PCINT0_vect
//Specyfikacja rejestrów i ich bitów konfiguracyjnych
//Rejestr statusów transceivera (bity 15...10 generują przerwania – wyprowadzenie nIRQ)
#define STATUS_REG 0x0000
  #define TX_REG_READY (1<<15)
  #define RX_FIFO_FULFILLED (1<<15)
  #define POWER_ON_RESET (1<<14)
  #define TX_REG_UNDERRUN (1<<13)
  #define RX_FIFO_OVERFLOW (1<<13)
  #define WAKE_UP_OVERFLOW (1<<12)
  #define LOW_ON_INTRPT_PIN (1<<11)
  #define LOW_BATT_DET (1<<10)
  #define FIFO_IS_EMPTY (1<<9)
  #define RSSI_ABOVE_LIMIT (1<<8)
  #define DATA_QUALITY_DET (1<<7)
  #define CLOCK_RECOVERY_LOCKED (1<<6)
//Rejestr podstawowej konfiguracji transceivera
#define CONFIGURATION_REG 0x8000  //DEF: 0x08
  #define ENABLE_TX_REG (1<<7)
  #define ENABLE_RX_FIFO (1<<6)
  #define BAND_433MHZ (1<<4)
  #define BAND_868MHZ (2<<4)
  #define BAND_915MHZ (3<<4)
  #define CRYSTAL_CAP_8_5 (0<<0)
  #define CRYSTAL_CAP_9_0 (1<<0)
  #define CRYSTAL_CAP_9_5 (2<<0)
  #define CRYSTAL_CAP_10_0 (3<<0)
  #define CRYSTAL_CAP_10_5 (4<<0)
  #define CRYSTAL_CAP_11_0 (5<<0)
  #define CRYSTAL_CAP_11_5 (6<<0)
  #define CRYSTAL_CAP_12_0 (7<<0)
  #define CRYSTAL_CAP_12_5 (8<<0)
  #define CRYSTAL_CAP_13_0 (9<<0)
  #define CRYSTAL_CAP_13_5 (10<<0)
  #define CRYSTAL_CAP_14_0 (11<<0)
  #define CRYSTAL_CAP_14_5 (12<<0)
  #define CRYSTAL_CAP_15_0 (13<<0)
  #define CRYSTAL_CAP_15_5 (14<<0)
  #define CRYSTAL_CAP_16_0 (15<<0)
//Rejestr zarządzania zasilaniem modułów transceivera
#define POWER_MANGMNT_REG 0x8200  //DEF: 0x08
  #define ENABLE_RECEIVER (1<<7)
  #define ENABLE_BASEBAND (1<<6)
  #define ENABLE_TRANSMITTER (1<<5)
  #define ENABLE_SYNTHESIZER (1<<4)
  #define ENABLE_CRYSTAL_OSC (1<<3)
  #define ENABLE_LOW_BATT_DET (1<<2)
  #define ENABLE_WAKEPU_TIMER (1<<1)
  #define DISABLE_CLOCK_OUTPUT (1<<0)
//Rejestr częstotliwości nośnej transmisji radiowej
#define FREQUENCY_REG 0xA000  //DEF: 0x680
//Rejestr prędkości transmisji tzw. bitrate
#define BITRATE_REG 0xC600  //DEF: 0x23
//Rejestr bajta synchronizacji
#define SYNCHRO_PATTERN_REG 0xCE00  //DEF: 0xD4
//Rejestr parametrów odbiornika
#define RX_CTRL_REG 0x9000  //DEF: 0x80
  #define P16_INTR_INPUT (0<<10)  //Pin 16, jako wejście przerwania
  #define P16_VDI_OUTPUT (1<<10)  //Jako wyjście VDI (Valid data indicator)
  #define VDI_RESPONSE_FAST (0<<8)
  #define VDI_RESPONSE_MED (1<<8)
  #define VDI_RESPONSE_SLOW (2<<8)
  #define VDI_RESPONSE_ALWAYS_ON (3<<8)
  #define BANDWITH_400KHZ (1<<5)
  #define BANDWITH_340KHZ (2<<5)
  #define BANDWITH_270KHZ (3<<5)
  #define BANDWITH_200KHZ (4<<5)
  #define BANDWITH_134KHZ (5<<5)
  #define BANDWITH_67KHZ (6<<5)
  #define LNA_GAIN_0 (0<<3)
  #define LNA_GAIN_6 (1<<3)
  #define LNA_GAIN_14 (2<<3)
  #define LNA_GAIN_20 (3<<3)
  #define DRSSI_THRSHLD_103 (0<<0)
  #define DRSSI_THRSHLD_97 (1<<0)
  #define DRSSI_THRSHLD_91 (2<<0)
  #define DRSSI_THRSHLD_85 (3<<0)
  #define DRSSI_THRSHLD_79 (4<<0)
  #define DRSSI_THRSHLD_73 (5<<0)
//Rejestr ustawień bufora FIFO odbiornika i układu Reset
#define RX_FIFO_SETTINGS_REG 0xCA00
  #define FIFO_DEPTH_8BITS (8<<4)
  #define SYNCHRO_PATT_2DD4 (0<<3)
  #define SYNCHRO_PATT_D4 (1<<3)
  #define FIFO_START_SYNCHRO (0<<2)
  #define FIFO_START_ALWAYS (1<<2)
  #define FIFO_FILL_ENABLE (1<<1)
  #define FIFO_FILL_DISABLE (0<<1)
  #define RESET_SENSITIVE (0<<0)
  #define RESET_NON_SENSITIVE (1<<0)
//Rejestr bufora FIFO odbiornika (do odczytu)
#define RX_FIFO_READ_REG 0xB000
```

tymczasem do konfiguracji naszego modułu radiowego. Nie będę w tym miejscu powielał dokumentacji producenta, lecz zaprezentuję czytelny plik nagłówkowy, w którego ramach zdefiniowano nazwy wszystkich portów sterujących, nazwy i wartości rejestrów konfiguracyjnych modułu RFM-12B oraz listę dostępnych ustawień. Ten plik pokazano na **listingu 2**. Zarówno nazwy rejestrów konfiguracyjnych, jak i listę dostępnych ustawień opisano w sposób niebudzący wątpliwości odnośnie do ich funkcjonalności. Zgodnie z tym, co napisano wcześniej, moduł RFM-12B wyposażono w szereg 16-bitowych rejestrów konfiguracyjnych, a co za tym idzie jest potrzebna funkcja, dzięki której jest możliwe wysłanie i odczytanie słowa 16-bitowego. Warto zauważyć, iż korzystamy tutaj z programowej implementacji obsługi interfejsu SPI. Funkcję pokazano na **listingu 3**. Zanim jednak skorzystamy z funkcji SPIsendWord() musimy odpowiednio skonfigurować wyprowadzenia sterujące interfejsu SPI, co można zrobić za pomocą funkcji pokazanej na **listingu 4**.

Listing 3. Funkcja, dzięki której możliwe jest wysłanie (i jednoczesne odczytanie) 16-bitowego słowa poprzez interfejs SPI

```
uint16_t SPIsendWord(uint16_t Word)
{
    SS_PORT &= ~(1<<SS_NR);  //SS=0 (Slave Select)
    for(uint8_t i=0; i<16; ++i)
    {
        if(Word & 0x8000) MOSI_PORT |= (1<<MOSI_NR); else MOSI_PORT &= ~(1<<MOSI_NR);
        Word <<= 1;
        SCK_PORT |= (1<<SCK_NR);  //Slave zatrzaskuje bieżący bit
        asm("nop"); asm("nop");
        if(MISO_PIN & (1<<MISO_NR)) Word |= 0x01;  //Zapisujemy odczytany bit
        SCK_PORT &= ~(1<<SCK_NR);  //Slave wystawia kolejny bit
    }
    SS_PORT |= (1<<SS_NR);  //SS=1 (Slave Unselect)
    return Word;
}
```

Listing 4. Funkcja konfigurująca wyprowadzenia programowego interfejsu SPI

```
void SPIinit(void)
{
    MISO_PORT |= (1<<MISO_NR);  //Podciągnięcie MISO do VCC
    SS_PORT |= (1<<SS_NR);  //SS domyślnie w stanie wysokim
    MOSI_DDR |= (1<<MOSI_NR);  //MOSI, SCK i SS jako wyjścia
    SCK_DDR |= (1<<SCK_NR);
    SS_DDR |= (1<<SS_NR);
}
```

Listing 6. Funkcje zarządzające trybem niskiego poboru mocy modułu RFM-12B

```
void RFM12bPowerDown(void)
{
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
    {
        //Przed przejściem w tryb Power-Down odczytujemy rejestr statusu
        //by skasować ewentualne flagi przerwań
        SPIsendWord(STATUS_REG);
        //Wyłączenie zasilania wszystkich modułów transceivera
        SPIsendWord(POWER_MANGMNT_REG|DISABLE_CLOCK_OUTPUT);
    }
}

void RFM12bPowerUp(void)
{
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
    {
        //Załączenie zasilania niezbędnych (na tym etapie) modułów transceivera
        SPIsendWord(POWER_MANGMNT_REG|ENABLE_CRYSTAL_OSC|DISABLE_CLOCK_OUTPUT);
    }
}
```

Mamy już podstawowe funkcje narzędziowe, więc pora na wykonanie uniwersalnej funkcji pozwalającej na skonfigurowanie modułu RFM-12B – pokazano ją na **listingu 5**. Przyjmuje ona 3 argumenty:

Listing 2. c.d.

```
//Rejestr parametrów nadajnika
#define TX_CTRL_REG 0x9800  //DEF: 0x00
  #define FREQ_DEV_15KHZ (0<<4)
  #define FREQ_DEV_30KHZ (1<<4)
  #define FREQ_DEV_45KHZ (2<<4)
  #define FREQ_DEV_60KHZ (3<<4)
  #define FREQ_DEV_75KHZ (4<<4)
  #define FREQ_DEV_90KHZ (5<<4)
  #define FREQ_DEV_105KHZ (6<<4)
  #define FREQ_DEV_120KHZ (7<<4)
  #define FREQ_DEV_135KHZ (8<<4)
  #define FREQ_DEV_150KHZ (9<<4)
  #define FREQ_DEV_165KHZ (10<<4)
  #define FREQ_DEV_180KHZ (11<<4)
  #define FREQ_DEV_195KHZ (12<<4)
  #define FREQ_DEV_210KHZ (13<<4)
  #define FREQ_DEV_225KHZ (14<<4)
  #define FREQ_DEV_240KHZ (15<<4)
  #define OUPTUT_PWR_0 (0<<0)
  #define OUPTUT_PWR_3 (1<<0)
  #define OUPTUT_PWR_6 (2<<0)
  #define OUPTUT_PWR_9 (3<<0)
  #define OUPTUT_PWR_12 (4<<0)
  #define OUPTUT_PWR_15 (5<<0)
  #define OUPTUT_PWR_18 (6<<0)
  #define OUPTUT_PWR_21 (7<<0)
//Rejestr danych nadajnika (do zapisu)
#define TX_WRITE_REG 0xB800  //DEF: 0xAA
//Rejestr syntezera PLL
#define PLL_REG 0xCC00  //DEF: 0x77
//Rejestr modułu wybudzania Wake-up timer
#define WAKE_UP_TIMER_REG 0xE000


//Definicja nowego typu danych transceivera
typedef struct
{
    uint8_t Idx;  //Bieżący indeks zapisywanego/odczytywanego znaku
    uint8_t Buffer[BUFFER_SIZE];  //Bufor nadawczo-odbiorczy
    uint8_t Status;  //Status transceivera
} RFM12type;
//Definicja statusów transceivera
enum {IDLE, RECEIVING, NEW_PACKET, FAULTY_PACKET, BROADCASTING, PACKET_SENT};
extern volatile RFM12type RFM12B;
```

1. **Frequency** – częstotliwość transmisji radiowej wybrana z dopuszczalnego zakresu częstotliwości (dostępny w komentarzu).
2. **Bitrate** –prędkość transmisji bitowej przesyłanych danych, którą dobieramy według potrzeb w dość szerokim zakresie (600...115200 bps).
3. **myID** – adres sprzętowy urządzenia będący tak naprawdę drugim bajtem synchronizacji, o czym wspominano wcześniej.

Warto zauważyć, że większa prędkość transmisji zapewnia krótszy czas jej trwania, ale też jest ona bardziej narażona na zaburzenia. Widać również, że w ramach funkcji inicjalizacyjnej jest konfigurowane wiele parametrów nadajnika i odbiornika modułu RFM-12B, ale ich jednoznaczne nazwy pozwolą na ewentualną własną korektę dostosowaną do potrzeb tworzonej aplikacji. Dalej pokażę 2 proste funkcje narzędziowe, dzięki którym możliwe jest wprowadzenie modułu w tryb niskiego poboru mocy (pobór rzędu 1 μA!), jak i wyjście z tego trybu. Te funkcje pokazano na **listingu 6**.

Listing 7. Funkcja do sprawdzania zajętości pasma radiowego

```
uint8_t RFM12bAirBusy(void)
{
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
    {
        //Sprawdzenie bitu RSSI_ABOVE_LIMIT w rejestrze statusowym
        //świadczącego o obecności sygnału RF
        if(SPIsendWord(STATUS_REG) & RSSI_ABOVE_LIMIT) return 1;
    }
    return 0;
}
```

W tym miejscu warto zauważyć, że moduł RFM-12B wyposażono w wygodny mechanizm detekcji częstotliwości nośnej, przez co, przed ewentualnym włączeniem nadajnika jest możliwe sprawdzenie zajętości

Listing 5. Funkcja umożliwiająca pełną konfigurację modułu RFM-12B

```
//Frequency: 430240...439750, 860480...879510, 900720...929270 MHz
//BitRate: 600...115200 bps

void RFM12bInit(uint32_t Frequency, uint32_t BitRate, uint8_t myID)
{
    uint16_t freqBits;
    //Inicjalizacja interfejsu SPI
    SPIinit();
    //Konfiguracja i uruchomienie przerwania zewnętrznego obsługującego sygnał
    //IRQ modułu (zbocze opadające)
    RFM12_IRQ_PORT |= (1<<RFM12_IRQ_NR);  //Podciągnięcie portu IRQ modułu do VCC
    PCICR = (1<<PCIE0);  //Uruchomienie przerwania Pin Change Interrupt 0
    PCMSK0 = (1<<PCINT2);  //Zmiana stanu na PB2/PCINT2 generuje przerwanie
    //Ustawienie pasma roboczego transceivera i wykonanie podstawowej konfiguracji
    if(Frequency >= 900000)  //915 MHz
    {
    SPIsendWord(CONFIGURATION_REG|ENABLE_TX_REG|ENABLE_RX_FIFO|BAND_915MHZ|CRYSTAL_CAP_12_0);
        freqBits = (10*(Frequency-900000))/75UL;
    }
    else if(Frequency >= 860000)  //868 MHz
    {
        SPIsendWord(CONFIGURATION_REG|ENABLE_TX_REG|ENABLE_RX_FIFO|BAND_868MHZ|CRYSTAL_CAP_12_0);
        freqBits = (Frequency-860000)/5UL;
    }
    else  //433 MHz
    {
        SPIsendWord(CONFIGURATION_REG|ENABLE_TX_REG|ENABLE_RX_FIFO|BAND_433MHZ|CRYSTAL_CAP_12_0);
        freqBits = (10*(Frequency-430000))/25UL;
    }
    //Ustawienie częstotliwości transmisji radiowej
    if(freqBits<96) freqBits = 96;
    if(freqBits>3903) freqBits = 3903;
    SPIsendWord(FREQUENCY_REG|freqBits);
    //Załączenie zasilania niezbędnych (na tym etapie) modułów transceivera
    SPIsendWord(POWER_MANGMNT_REG|ENABLE_CRYSTAL_OSC|DISABLE_CLOCK_OUTPUT);
    //Ustawienie prędkości transmisji - tzw. bitrate
    if(BitRate<2694) SPIsendWord(BITRATE_REG|(1<<7)|((43103/BitRate)-1)); else SPIsendWord(BITRATE_REG|((344828/BitRate)-1));
    //Ustawiamy definiowalny bajt synchronizacji, czyli de facto adres naszego urządzenia
    SPIsendWord(SYNCHRO_PATTERN_REG|myID);
    //Ustawienie parametrów odbiornika
    SPIsendWord(RX_CTRL_REG|P16_VDI_OUTPUT|VDI_RESPONSE_FAST|BANDWITH_134KHZ|LNA_GAIN_0|DRSSI_THRSHLD_103);
    //Ustawienie parametrów bufora FIFO odbiornika i układu Reset
    SPIsendWord(RX_FIFO_SETTINGS_REG|FIFO_DEPTH_8BITS|SYNCHRO_PATT_2DD4|FIFO_START_SYNCHRO|FIFO_FILL_DISABLE|RESET_NON_SENSITIVE);
    SPIsendWord(RX_FIFO_SETTINGS_REG|FIFO_DEPTH_8BITS|SYNCHRO_PATT_2DD4|FIFO_START_SYNCHRO|FIFO_FILL_ENABLE|RESET_NON_SENSITIVE);
    //Ustawienie parametrów nadajnika
    SPIsendWord(TX_CTRL_REG|FREQ_DEV_15KHZ|OUPTUT_PWR_0);
    //Ustawienie parametrów PLL według domyślnej wartości z dokumentacji (zalecane)
    SPIsendWord(PLL_REG|0x77);
    //Ustawienie parametrów modułu wybudzania Wake-up timer (wyłączenie)
    SPIsendWord(WAKE_UP_TIMER_REG|0x00);
    //Odczyt rejestru statusu - start modułu RFM12B
    SPIsendWord(STATUS_REG);
}
```

Listing 8. Funkcja inicjująca proces wysyłania danych

```
void RFM12bStartTx(char *Data, uint8_t Size, uint8_t destID)
{
    uint8_t i, j, CRC8 = 0;
    //Jeśli podaliśmy parametr Size=0 to mamy do czynienia z łańcuchem tekstowym
    if(!Size) Size = strlen(Data);  //Obliczamy jego długość
    RFM12B.Status = BROADCASTING;  //Zmiana statusu transceivera
    //Zwiększamy długość ramki wynikowej o: 3 bajty preambuły, 2 bajty synchro, 1 bajt rozmiaru ramki,
    //1 bajt CRC8 i dwa bajty postambuły
    i = RFM12B.Idx = Size + 8;
    RFM12B.Buffer[i--] = 0xAA;  //Preambuła
    RFM12B.Buffer[i--] = 0xAA;
    RFM12B.Buffer[i--] = 0xAA;
    RFM12B.Buffer[i--] = 0x2D;  //Pierwszy bajt synchronizacji
    RFM12B.Buffer[i--] = destID;  //Drugi bajt synchronizacji będący adresem urządzenia docelowego
    RFM12B.Buffer[i--] = Size;  //Bajt rozmiaru ramki
    CRC8 = _crc_ibutton_update(CRC8, Size);  //Aktualizacja CRC8
    for(j = 0; j<Size; ++j)
    {
        RFM12B.Buffer[i--] = Data[j];
        CRC8 = _crc_ibutton_update(CRC8, Data[j]);  //Aktualizacja CRC8
    }
    RFM12B.Buffer[i--] = CRC8;  //Bajt CRC8
    RFM12B.Buffer[i--] = 0xAA;  //Postambuła
    RFM12B.Buffer[i--] = 0xAA;
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
    {
        //Włączenie toru radiowego nadajnika, co spowoduje zgłaszanie przerwań
        SPIsendWord(POWER_MANGMNT_REG|ENABLE_TRANSMITTER|ENABLE_SYNTHESIZER|ENABLE_CRYSTAL_OSC|DISABLE_CLOCK_OUTPUT);
    }
}
```

Listing 9. Funkcja inicjująca proces odbierania danych

```
void RFM12bStartRx(void)
{
    RFM12B.Status = RECEIVING;  //Zmiana statusu transceivera
    RFM12B.Idx = 0;  //Zerujemy indeks odbieranego znaku
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
    {
        //Zresetowanie bufora FIFO (konieczne, by uruchomić mechanizm detekcji słowa synchro) i włączenie toru radiowego odbiornika
        SPIsendWord(POWER_MANGMNT_REG|ENABLE_RECEIVER|ENABLE_BASEBAND|ENABLE_CRYSTAL_OSC|DISABLE_CLOCK_OUTPUT);
        SPIsendWord(RX_FIFO_SETTINGS_REG|FIFO_DEPTH_8BITS|SYNCHRO_PATT_2DD4|FIFO_START_SYNCHRO|FIFO_FILL_DISABLE|RESET_NON_SENSITIVE);
        SPIsendWord(RX_FIFO_SETTINGS_REG|FIFO_DEPTH_8BITS|SYNCHRO_PATT_2DD4|FIFO_START_SYNCHRO|FIFO_FILL_ENABLE|RESET_NON_SENSITIVE);
    }
}
```

pasma radiowego. Funkcjonalność tą realizuje funkcja pokazana na **listingu 7**, która korzysta z bitu RSSI_ABOVE_LIMIT znajdującego się w rejestrze statusowym modułu STATUS_REG, którego to ustawienie świadczy o obecności częstotliwości nośnej (a dokładnie o mocy sygnału radiowego powyżej ustawionego limitu – ustawienie to determinuje wartość bitów DRSSI_THRESHOLD w rejestrze RX_CTRL_REG).

Pora na główne funkcje umożliwiające nadawanie i odbieranie danych. Warto w tym miejscu zauważyć, że te funkcje wyłącznie inicjują nadawanie lub odbieranie danych, natomiast sam proces zachodzi w procedurze obsługi przerwania od wyprowadzenia *nIRQ* modułu. Jest to możliwe dzięki dwóm bitom rejestru statusowego oraz odpowiednim przerwaniom – TX_REG_READY oraz RX_FIFO_FULFILLED. Pierwsze z nich staje się aktywne po uruchomieniu nadajnika i powoduje zgłaszanie przerwań za każdym razem, gdy bufor nadawczy jest gotowy na przyjęcie nowych danych przeznaczonych do wysłania. Drugi z bitów zajmujący to same miejsce w rejestrze statusowym jest aktywny w trybie odbiornika i powoduje zgłaszanie przerwania za każdym razem, gdy w buforze odbiorczym znajduje się odpowiednia (konfigurowana, standardowo – 8) liczba bitów.

Funkcję inicjującą wysyłanie danych pokazano na **listingu 8**. Podczas jej wywołania podaje się 3 argumenty:

1. ***Data** –wskaźnik na tablicę danych przeznaczonych do wysłania.
2. **Size** – rozmiarem danych użytecznych przeznaczonych do wysłania (w wypadku łańcuchów tekstowych należy podać wartość 0).
3. **destID** – adres urządzenia docelowego (w formie drugiego bajta synchronizacji).

Z analizy kodu funkcji pokazanej na list. 8 wynika, że przygotowuje ona ramkę transmisji według pokazanego wcześniej wzorca, posługując się globalną zmienną tablicową *RFM12B.Buffer*, po czym włącza nadajnik transceivera RFM-12B, co powoduje wygenerowanie szeregu przerwań systemowych, w ramach obsługi których następuje faktyczne wysłanie tak przygotowanej tablicy danych.

Funkcję inicjującą odbieranie danych pokazano na **listingu 9**. Tym razem do czynienia mamy ze znacznie prostszą funkcją, która poza przygotowaniem zmiennych używanych w mechanizmie odbioru danych, uruchamia odbiornik transceivera RFM-12B, co powoduje wygenerowanie szeregu przerwań, w ramach obsługi których następuje faktyczne odebranie ramki danych. Jej odebraniu (z poprawną sumą CRC8) towarzyszy zmiana wartości globalnej zmiennej statusowej *RFM12B.Status* na predefiniowaną wartość *NEW_PACKET*. W tym momencie odebrane dane umieszczone zostają w globalnej zmiennej *RFM12B.Buffer*.

Robert Wołgajew, EP