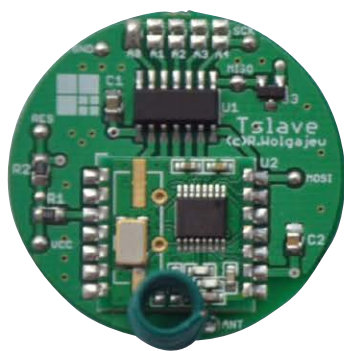




Autor składa podziękowania panu Sławomirowi Szwedzie z firmy Unisystem za dostarczenie wyświetlacza OLED znajdującego zastosowanie w niniejszym projekcie.

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-kanalowy termometr z interfejsem Wi-Fi (EP 4/2018)
AVT-5566	THPStation – rozbudowany termometr z Wi-Fi (EP 1/2017)
AVT-1941	8-kanalowy termometr z I ² C (EP 1/2017)
AVT-5573	Nieskomplikowany termometr-rejestrator (EP 11/2016)
AVT-5535	Termometr 2-kanalowy 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-kanalowy 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-kanalowy 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-kanalowy 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 wylutować 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 wylutowane w płytce 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 na 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.

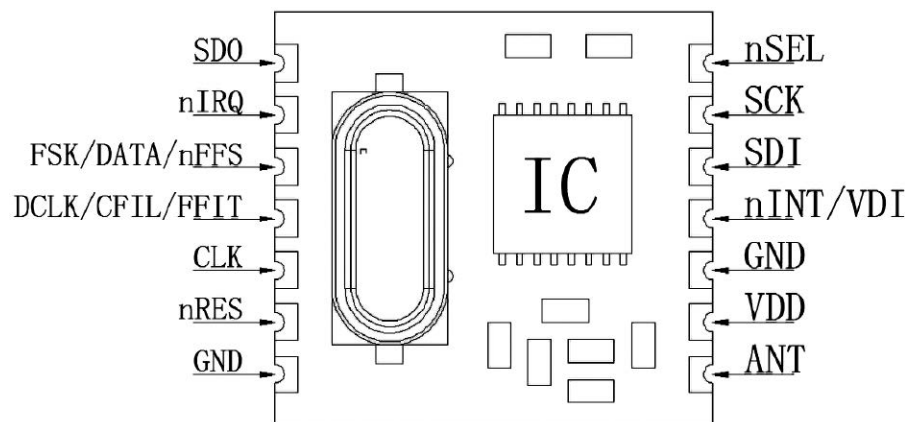
Bezprzewodowy, energooszczędny system pomiaru temperatury (1)

Stałem 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

REKLAMA



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, 1PA
WriteCMD(0xA640);
WriteCMD(0xC647);
WriteCMD(0XC777);
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
```

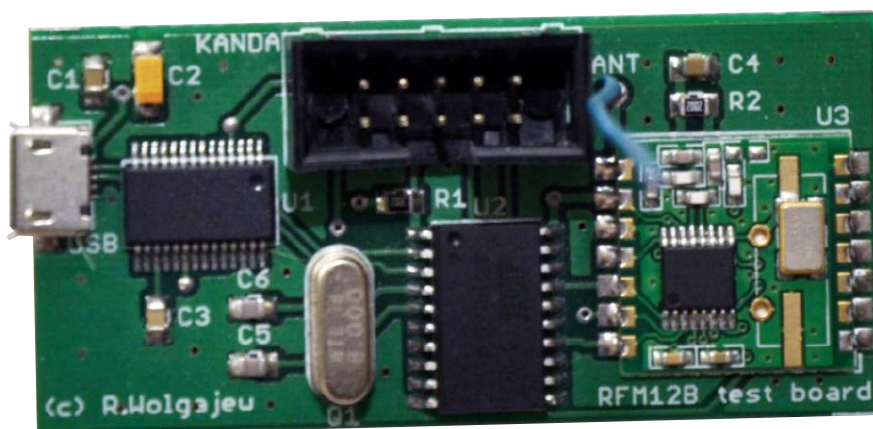
płytki drukowanej modułu w wersji SMD, natomiast w tabeli 1 rozmieszczenie tych wyprowadzeń, które są używane w opisywanej aplikacji.

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.

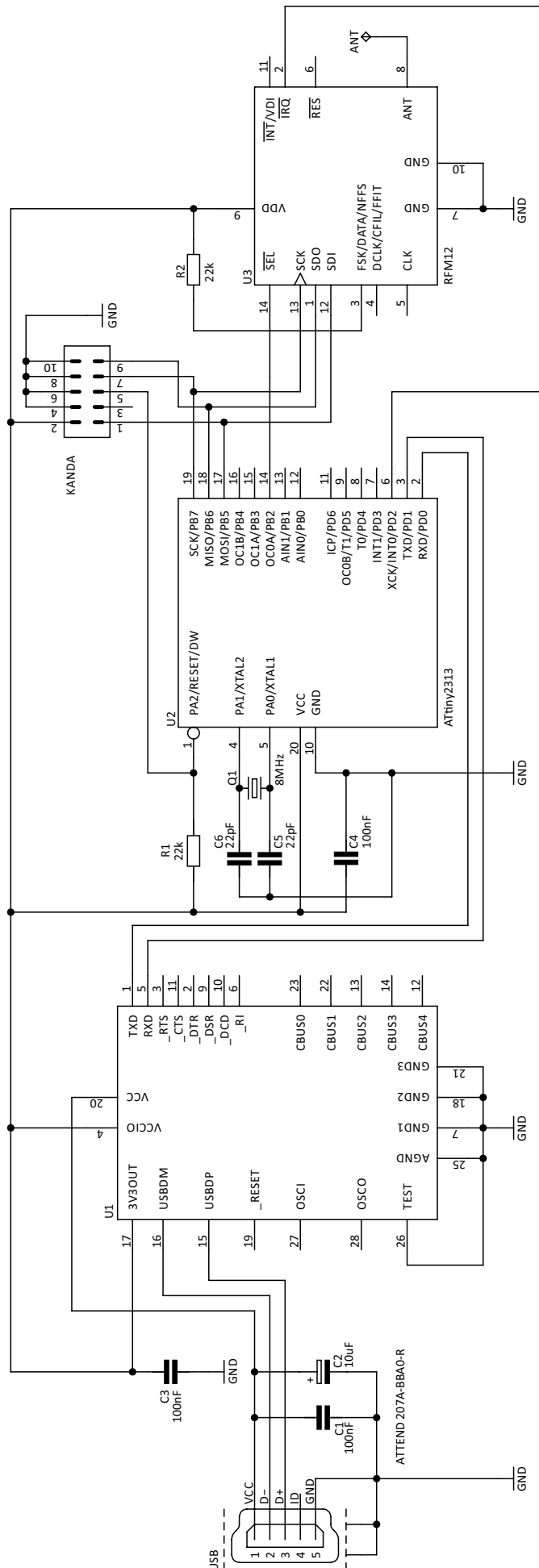
Po lekturze dostępnych rozwiązań programowych oraz niekompletnej dokumentacji producenta (w tym przykładowych

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↔U-SART 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 pisanie 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 bufor nadawczo-odbiorczy 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 przesyłać 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

AA	AA	AA	2D	D4	Dane...	AA	AA
----	----	----	----	----	---------	----	----

Rysunek 4. Przykładowa ramka danych modułu RFM-12B

AA	AA	AA	2D	D4	SIZE	Dane...	CRC	AA	AA
----	----	----	----	----	------	---------	-----	----	----

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_WAKEUP_TIMER (1<<1)
#define DISABLE_CLOCK_OUTPUT (1<<0)
//Rejestr częstotliwości nośnej transmisji radiowej
#define FREQUENCY_REG 0xA000 //DEF: 0x600
//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 BANDWIDTH_400KHZ (1<<5)
#define BANDWIDTH_340KHZ (2<<5)
#define BANDWIDTH_270KHZ (3<<5)
#define BANDWIDTH_200KHZ (4<<5)
#define BANDWIDTH_134KHZ (5<<5)
#define BANDWIDTH_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 wcze-

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 konfigująca wyprowadzenia programowego interfejsu SPI

```
void SPInit(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);
}
```

śniej, 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 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 OUPUT_PWR_0 (0<<0)
#define OUPUT_PWR_3 (1<<0)
#define OUPUT_PWR_6 (2<<0)
#define OUPUT_PWR_9 (3<<0)
#define OUPUT_PWR_12 (4<<0)
#define OUPUT_PWR_15 (5<<0)
#define OUPUT_PWR_18 (6<<0)
#define OUPUT_PWR_21 (7<<0)
//Rejestr danych nadajnika (do zapisu)
#define TX_WRITE_REG 0xB800 //DEF: 0xAA
//Rejestr syntezy 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 wspomniano 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 dosto-

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;
}
```

sowaną do potrzeb tworzonej aplikacji. Dalej pokażę 2 proste funkcje narzędziowe, dzięki którym możliwe jest wprowadzenie modułu w tryb niskiego poboru mocy (pobór rzędu 1 μ A!), jak i wyjście z tego trybu. Te funkcje pokazano na [listingu 6](#).

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|BANDWIDTH_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|OUPUT_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 przerw
        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 dokładniej 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 wprowadzenia *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 przerwania 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).

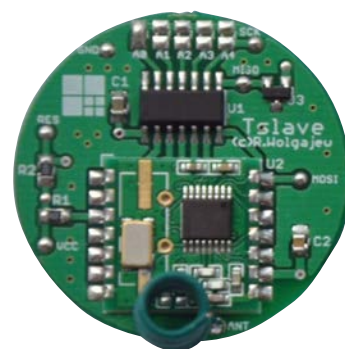
Z analizy kodu funkcji pokazanej na list. 8 wynika, że przygotowuje ona ramkę transmisji według pokazanego wcześniej wzorca,

posługując się globalną zmienną tablicową *RFM12B.Buffer*, po czym włącza nadajnik transceivera RFM-12B, co powoduje wygenerowanie szeregu przerw 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 przerw, 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

www.ep.com.pl/kap



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-kanalowy termometr z interfejsem Wi-Fi (EP 4/2018)
AVT-5566	THPStation - rozbudowany termometr z Wi-Fi (EP 1/2017)
AVT-1941	8-kanalowy termometr z I ² C (EP 1/2017)
AVT-5573	Nieskomplikowany termometr-rejestrator (EP 11/2016)
AVT-5535	Termometr 2-kanalowy 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-kanalowy 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-kanalowy termometr z wyświetlaczem LED (EP 5/2012)
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-kanalowy 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 (2)

Stałem 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.

W pierwszej części artykułu opisano oprogramowanie systemu do pomiaru temperatury. W części drugiej zostanie zaprezentowany schemat ideowy oraz będzie opisana budowa urządzenia.

Kończąc opis oprogramowania, opiszę funkcję inicjującą odbieranie danych – pokazano ją na **listingu 9**. Ta funkcja, 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**.

Pora na „wisienkę na torcie”, czyli funkcję obsługi przerwania zewnętrznego

(od wyprowadzenia nIRQ modułu), która to jest „silnikiem” całego mechanizmu realizując rzeczywiste nadawanie lub odbieranie danych. Tę funkcję przedstawiono na **listingu 10**. Zaopatrzone ją w bardzo bogate komentarze, więc nie wymaga ona dodatkowego opisywania. Warto jednak zauważyć, że cały proces wysyłania czy odbierania danych odbywa się w tle nie wstrzymując działania pętli głównej aplikacji. Bieżący stan realizowanego procesu możemy każdorazowo sprawdzić testując wartość zmiennej globalnej **RFM12B.Status**.

Uff, to tyle, jeśli chodzi o nasze ciekawe pierzyferium, w związku z czym pora na przedstawienie szczegółów konstrukcyjnych

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_Mgmt_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);
    }
}
```

Listing 10. Funkcja obsługi przerwania zewnętrznego realizująca rzeczywiste nadawanie lub odbieranie danych

```
ISR(RFM12_IRQ_NAME)
{
    uint8_t frameSize, CRC8 = 0;
    //Reagujemy tylko na zbocze opadające przerwania Pin Change Interrupt 1
    if(!(RFM12_IRQ_PIN & (1<<RFM12_IRQ_NR)))
    {
        //Transceiver w trybie odbiornika
        if(RFM12B.Status == RECEIVING)
        {
            //Odczyt odebranego bajta danych - sprawdzamy, przy okazji czy nie przekroczono maksymalnej długości ramki
            //Jeśli długość przekroczono to odrzucamy taką ramkę danych (jako błędną) i wyłączamy tor radiowy odbiornika
            if(RFM12B.Idx < BUFFER_SIZE) RFM12B.Buffer[RFM12B.Idx++] = SPIsendWord(RX_FIFO_READ_REG) & 0xFF;
            else
            {
                SPIsendWord(Power_Mgmt_Reg|ENABLE_CRYSTAL_OSC|DISABLE_CLOCK_OUTPUT);
                RFM12B.Status = FAULTY_PACKET; //Zmiana statusu transceivera
            }
        }

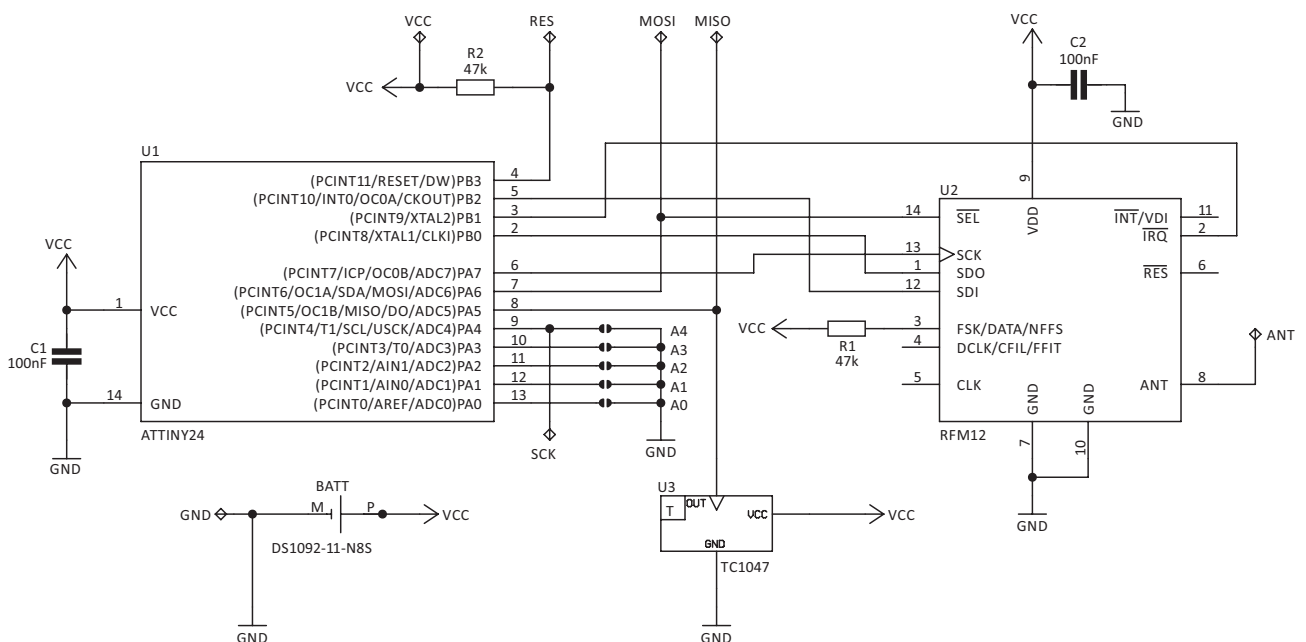
        //Sprawdzamy, czy otrzymano już kompletną ramkę danych o długości Size (bajt 0. ramki) plus 2. Dodatkowe
        //bajty, które muszą zostać odebrane to: rozmiar ramki oraz suma CRC8 (bajty postambuły: 0xAA pomijamy)
        frameSize = RFM12B.Buffer[0]; //Rozmiar przesłanej ramki danych (tylko dane użyteczne)
        if(RFM12B.Idx == frameSize+2)
        {
            //Wyłączenie toru radiowego odbiornika
            SPIsendWord(Power_Mgmt_Reg|ENABLE_CRYSTAL_OSC|DISABLE_CLOCK_OUTPUT);
            //Obliczenie CRC8 z odebranych bajtów danych (w tym bajta Size - nr 0.) i porównanie z przesłanym CRC8 (bajt Size+1)
            for(uint8_t i=0; i<= frameSize; ++i) CRC8 = _crc_ibutton_update(CRC8, RFM12B.Buffer[i]);
            //Zmiana statusu transceivera i sygnalizacja nadejścia nowej ramki danych w przypadku zgodnej sumy CRC8
            if(RFM12B.Buffer[frameSize+1] == CRC8) RFM12B.Status = NEW_PACKET; else RFM12B.Status = FAULTY_PACKET;
        }
        else if(RFM12B.Status == BROADCASTING) //Transceiver w trybie nadajnika
        {
            //Wysyłamy bajt danych
            SPIsendWord(TX_WRITE_REG|RFM12B.Buffer[RFM12B.Idx]);
            //Sprawdzamy, czy pozostało coś jeszcze do wysłania i jeśli nie to wyłączamy tor radiowy nadajnika
            if(RFM12B.Idx == 0)
            {
                SPIsendWord(Power_Mgmt_Reg|ENABLE_CRYSTAL_OSC|DISABLE_CLOCK_OUTPUT);
                RFM12B.Status = PACKET_SENT; //Zmiana statusu transceivera
            }
            else RFM12B.Idx--;
        }
    }
}
```

przedmiotu niniejszego artykułu, a mianowicie bezprzewodowego, energooszczędnego systemu wielopunktowego pomiaru temperatury, który w założeniach

charakteryzować się ma następującymi cechami funkcjonalnymi:

- Obsługa do 32 adresowalnych węzłów pomiarowych (układów Slave).

- Zasilanie bateryjne i niski pobór mocy węzła pomiarowego.
- Pozostawianie w uśpieniu węzłów pomiarowych i cykliczne wybudzanie



Rysunek 6. Schemat ideowy węzła pomiarowego

Listing 11. Funkcja *main* węzła pomiarowego

```

int main(void)
{
    char Data[3];
    //Podciągnięcie portu adresu do VCC
    ADDRESS_PORT |= (1<<PA4)|(1<<PA3)|(1<<PA2)|(1<<PA1)|(1<<PA0);
    //Redukcja poboru mocy przez wyłączenie modułów (lub ich zegarów): TIMER1, TIMER0, USI
    PRR = (1<<PRTIM1)|(1<<PRTIM0)|(1<<PRUSI);
    //Wyłączenie komparatora analogowego dla zmniejszenia poboru mocy
    ACSR = (1<<ACD);
    //Uruchomienie i konfiguracja Watchdoga: Watchdog Timeout Interrupt Enable, Timeout: 1024K cycles -> 8.0 s
    WDTCR = (1<<WDIE)|(1<<WDP3)|(1<<WDP0);
    //Uruchomienie i konfiguracja RFM12B, w tym interfejsu SPI
    RFM12bInit(869000, 8000, ADDRESS_PIN & 0b1111);
    sei();
    while(1)
    {
        set_sleep_mode(SLEEP_MODE_PWR_DOWN);
        sleep_enable();
        sleep_cpu();
        //W tym miejscu CPU oczekuje na wybudzenie przez Watchdog
        sleep_disable();
        //Wykonanie niezbędnych pomiarów
        Data[0] = ADDRESS_PIN & 0b1111; //Adres sprzętowy naszego urządzenia
        Data[1] = ADCmeasure(ADC_CHANNEL_TEMP); //Wykonanie pomiaru temperatury
        Data[2] = ADCmeasure(ADC_CHANNEL_VCC); //Wykonanie pomiaru napięcia zasilania
        ADC_STOP_MEASURE; //Wyłączenie ADC dla oszczędzania energii
        //Wychodzimy z trybu power-down modułu RFM12b
        RFM12bPowerUp();
        //Inicjujemy wysłanie ramki danych
        RFM12bStartTx(Data, 3, MASTER_ID);
        //Czekamy na zakończenie transmisji
        while(RFM12B.Status != PACKET_SENT);
        //Wprowadzamy moduł RFM12B w tryb power-down
        RFM12bPowerDown();
    }
}

```

się, któremu towarzyszy przesyłanie wartości mierzonej temperatury oraz stanu baterii zasilającej.

- Efektowny, graficzny interfejs użytkownika po stronie układu nadrzędnego (układu Master).
- Wysoka ergonomia obsługi całego systemu i brak konieczności konfiguracji.
- Kontrola aktywności węzłów pomiarowych przez układ nadrzędny.

Zacznijmy zatem od układu podrzędnego, czyli węzła pomiarowego, którego schemat pokazano na **rysunku 6**. Jego „sercem” jest niewielki mikrokontroler ATtiny24 sterujący pracą modułu transceivera dzięki realizacji programowej obsługi interfejsu SPI oraz obsłudze przerwania zewnętrznego **Pin Change Interrupt 1** (wyprowadzenie PCINT9) odpowiedzialnego za mechanizm wysyłania danych. Ponadto, dzięki wykorzystaniu przetwornika A/C wbudowanego w mikrokontroler, możliwy stał się pomiar temperatury przetwornika temperatura/napięcie pod postacią układu TC1047 oraz pomiar napięcia baterii zasilającej.

Na pierwszy „rzut oka” nie wydaje się, aby mikrokontroler w jakikolwiek sposób używał przetwornika A/C do pomiaru napięcia baterii zasilającej, ponieważ żadne z jego wejść zewnętrznych nie jest używane w tym celu. To prawda. Patrząc na schemat układu i nie mając do dyspozycji listingu programu można by wysnuć taki wniosek, jednak przetwornik A/C mierzy w takim przypadku specjalne, wewnętrzne napięcie odniesienia $V_{BG}=1,1$ V, dzięki temu, iż wewnętrzny, analogowy multiplexer przetwornika może zostać właśnie w ten sposób ustawiony. Napięciem odniesienia jest w takim wypadku napięcie zasilające mikrokontroler, czyli napięcie dostarczane na wyprowadzenie VCC.

Listing 12. Plik nagłówkowy obsługi wyświetlacza OLED ze sterownikiem SSD1309

```

//Ustawienia konfiguracji połączeń
#define DATA_PORT PORTD
#define DATA_DDR DDRD
#define CONTROL_PORT PORTC
#define CONTROL_DDR DDRC
#define WR_NR PC3 //Write (0)
#define CS_NR PC0 //Chip select (0)
#define DC_NR PC2 //Data (1)/Command (0)
#define RST_NR PC1 //Reset (0)
#define RESET_WR CONTROL_PORT &= ~(1<<WR_NR)
#define SET_WR CONTROL_PORT |= (1<<WR_NR)
#define RESET_CS CONTROL_PORT &= ~(1<<CS_NR)
#define SET_CS CONTROL_PORT |= (1<<CS_NR)
#define RESET_DC CONTROL_PORT &= ~(1<<DC_NR)
#define SET_DC CONTROL_PORT |= (1<<DC_NR)
#define RESET_RST CONTROL_PORT &= ~(1<<RST_NR)
#define SET_RST CONTROL_PORT |= (1<<RST_NR)
//Definicje rozkazów sterujących sterownika SSD1309
#define SET_CONTRAST_CTRL 0x81
#define OUTPUT_FOLLOWS_RAM 0xA4
#define OUTPUT_IGNORES_RAM 0xA5
#define SET_NORMAL_DISPLAY 0xA6
#define SET_INVERSE_DISPLAY 0xA7
#define SET_DISPLAY_ON 0xAF
#define SET_DISPLAY_OFF 0xAE
#define SET_MEM_ADDR_MODE 0x20
#define HORIZONTAL_ADDRESSING 0x00
#define VERTICAL_ADDRESSING 0x01
#define PAGE_ADDRESSING 0x02
#define SET_COLUMN_ADDRESS 0x21
#define SET_PAGE_ADDRESS 0x22
#define SET_DISPLAY_START_LINE 0x40
#define SET_SEGMENT_REMAP_NORMAL 0xA0
#define SET_SEGMENT_REMAP_INVERSED 0xA1
#define SET_MULTIPLEX_RATIO 0xA8
#define MASTER_CONFIGURATION 0xAD
#define SET_COM_SCAN_DIR_NORMAL 0xC0
#define SET_COM_SCAN_DIR_INVERSED 0xC8
#define SET_DISPLAY_OFFSET 0xD3
#define SET_COM_PINS_HARDWARE_CONFIG 0xDA
#define SET_DISPLAY_CLOCK 0xD5
#define SET_PRECHARGE_PERIOD 0xD9
#define SET_VCOMH_DESELECT_LVL 0xDB

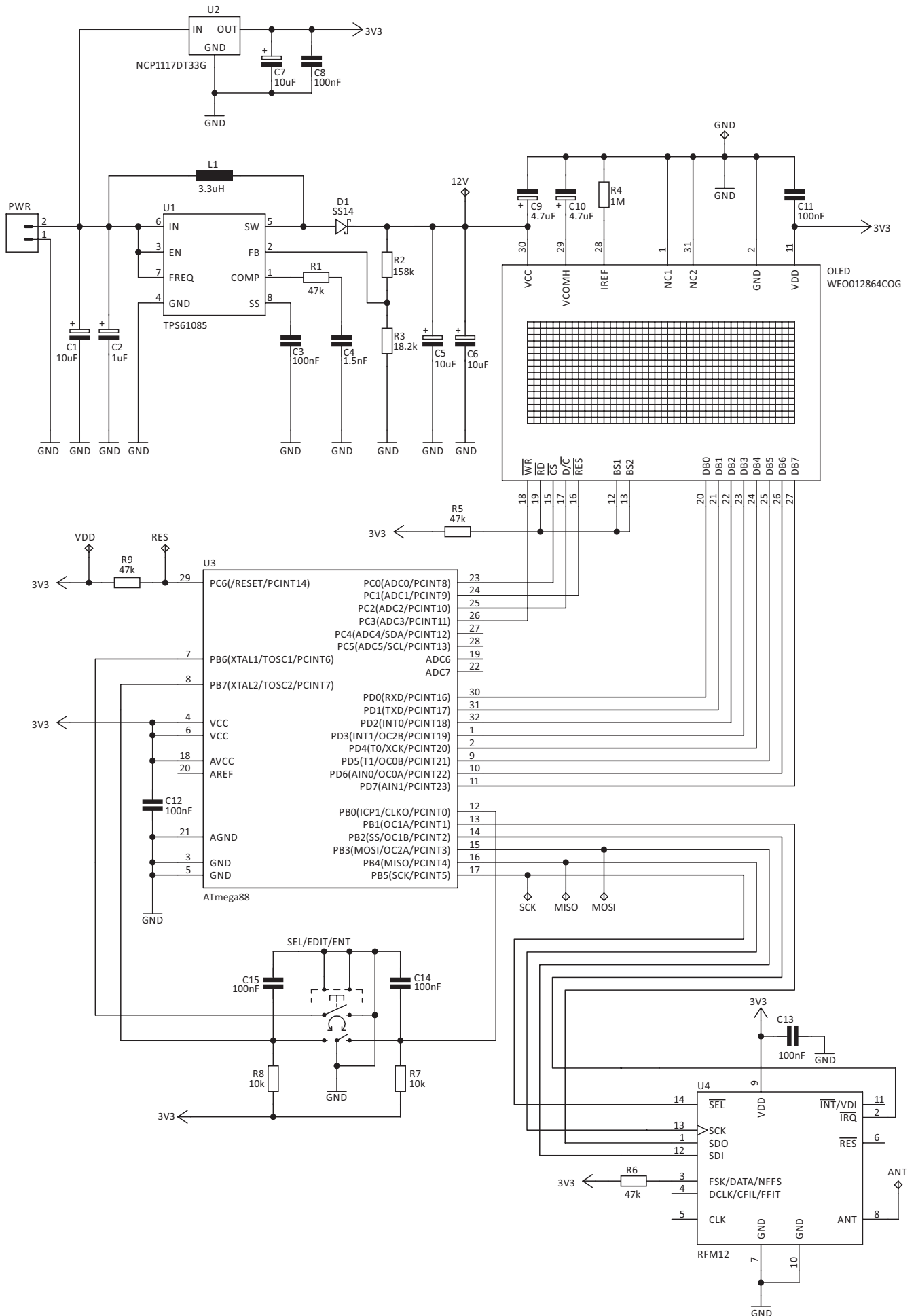
```

Skoro mierzone napięcie ma wartość stałą (1,1 V), zaś napięcie odniesienia wartość zmienną, łatwo możemy ustalić jego wartość. Dzięki temu „trickowi” stał się możliwy pomiar napięcia baterii zasilającej bez angażowania dodatkowego pinu mikrokontrolera, którego de facto już nie mamy. Dodatkowo, mikrokontroler obsługuje 5 wyprowadzeń adresowych A0...A4, pozwalających na ustawienie adresu sprzętowego węzła pomiarowego.

Zgodnie z tym, co napisano wcześniej, węzeł pomiarowy powinien charakteryzować się minimalnym zapotrzebowaniem na energię elektryczną, ponieważ jest zasilany

niewielką baterią CR2032. W związku z powyższym zastosowano następujące mechanizmy programowo-sprzętowe:

- Wyłączono wszystkie nieużywane peryferia mikrokontrolera (komparator analogowy, TIMER1, TIMER0, USI).
- Wprowadzono mikrokontroler w tryb obniżonego poboru energii Power-down, z którego jest wybudzany cyklicznie co 8 sekund przez odpowiednio skonfigurowany układ Watchdog, którego zadziałanie nie powoduje zresetowania mikrokontrolera, tylko wywołanie stosownego



Rysunek 7. Schemat ideowy układu nadrzędnego systemu pomiarowego

przerwania systemowego mającego możliwość wybudzania mikrokontrolera. Wybudzony mikrokontroler inicjuje transmisję danych, po czym przechodzi ponownie w tryb Power-down.

- Nieużywany transceiver RFM-12B wprowadzany jest każdorazowo w tryb obniżonego poboru energii.

Dzięki opisanym mechanizmom zapobieganie na prąd zmniejszono do około 40 μ A w trybie obniżonego poboru energii oraz 20 mA w czasie transmisji danych trwającej 10 ms (przy pełnej mocy nadawania). Osiągnięte wyniki zapewniają zadowalający czas pracy węzła pomiarowego na jednej baterii zasilającej.

Na **listingu 11** pokazano funkcję **main** programu obsługi węzła pomiarowego, realizującą całą, założoną funkcjonalność. Transceiver pracując wyłącznie w trybie nadajnika wysyła każdorazowo 3 bajty danych użytecznych: swój adres sprzętowy, wartość zmierzonej temperatury zewnętrznej oraz wartość napięcia zasilania. Nie jest sprawdzana obecność częstotliwości nośnej, a więc zajętość pasma transmisji, ale przeprowadzone testy praktyczne wykazały, że ryzyko kolizji danych jest naprawdę pomijalne, zwłaszcza przy tak zbudowanym systemie pomiarowym. Oczywiście, wysyłanie mierzonej temperatury co 8 sekund może okazać się zbyt częste jak i niepotrzebne w kontekście „drenowania” zasobów baterii zasilającej, lecz dzięki nieskomplikowanej korekcie możemy ten czas łatwo modyfikować.

Tyle w kwestii konstrukcji oprogramowania węzła pomiarowego, a zatem czas na przedstawienie schematu ideowego jednostki nadrzędnej, której zadaniem jest obsługa węzłów pomiarowych i wizualizacja przesyłanych danych. Schemat ideowy układu nadrzędnego pokazano na **rysunku 7**.

Tym razem do czynienia mamy z nieco bardziej rozbudowanym systemem mikroprocesorowym, którego „sercem” jest popularny mikrokontroler ATmega88 sterujący pracą modułu transceivera dzięki realizacji programowej obsługi interfejsu SPI oraz obsłudze przerwania zewnętrznego **Pin Change Interrupt 0** (wyprowadzenie PCINT2) odpowiedzialnego za mechanizm odbierania danych. Ponadto, mikrokontroler zajmuje się też obsługą enkodera obrotowego ze zintegrowanym przyciskiem, stanowiącego element interfejsu użytkownika oraz, co ważniejsze,

realizuje obsługę doskonałej jakości wyświetlacza graficznego OLED o przekątnej 2,7” o grubości zaledwie 2 mm (!) i rozdzielczości 128×64 piksele, wyposażonego w sterownik ekranu SSD1309. Ten element potrzebuje do działania napięcia 12 V, więc wykonano przetwornicę step-up z użyciem układu TPS61085. Warto również podkreślić, iż nie bez powodu wybrano ten rodzaj wyświetlacza – jego właściwości użytkowe w porównaniu z typowym elementem LCD są nie do przecenienia. Co więcej, cena tego modułu jest zbliżona do ceny analogicznego elementu wykonanego w technologii LCD, co nie pozostawia nam właściwie wyboru.

W związku z powyższym oraz z uwagi na fakt, iż wyświetlacz z tego rodzaju sterownikiem nie był dotychczas przedmiotem żadnego opracowania, postanowiłem przedstawić podstawowe funkcje obsługi tego peryferium. Zaczniemy standardowo od przedstawienia krótkiego pliku nagłówkowego, w którym zdefiniowano adresy rejestrów sterujących, a który to pokazano na **listingu 12**. Aby jednak zacząć współpracę z tym arcyciekawym wyświetlaczem niezbędne są dwie podstawowe funkcje narzędziowe, które pozwolą na wysłanie do panelu OLED rozkazów sterujących, dzięki którym jest możliwa konfiguracja sterownika

SSD1309 lub też danych obrazu. Interpretacja rodzaju danych, które mają zostać odebrane przez sterownik ekranu jest zdeterminowana stanem wyprowadzenia DC. Jest to typowe rozwiązanie w sterowaniu wyświetlaczami wszelkiego typu. Poziom niski na tym wyprowadzeniu decyduje o tym, iż przesyłana wartość zostanie zinterpretowana jako komenda sterująca zaś poziom wysoki spowoduje, iż dana zinterpretowana zostanie jako dana pamięci ekranu. Wspomniane, podstawowe funkcje narzędziowe zamieszczono na **listingu 13**.

Dysponując podstawowymi funkcjami narzędziowymi z list. 13 możemy przystąpić

Listing 13. Podstawowe funkcje narzędziowe odpowiedzialne za wysyłanie rozkazów sterujących lub danych obrazu

```
void writeCmd(uint8_t Command)
{
    RESET_CS;
    RESET_DC;
    RESET_WR;
    DATA_PORT = Command;
    SET_WR;
    SET_DC;
    SET_CS;
}

void writeData(uint8_t Data)
{
    RESET_CS;
    RESET_WR;
    DATA_PORT = Data;
    SET_WR;
    SET_CS;
}
```

Listing 14. Funkcja inicjująca sterownik ekranu SSD1309 wyświetlacza OLED z rodziny WE0912864

```
void OLEDinit(void)
{
    //Port danych OLEDa jako wyjścia ze stanem "0"
    DATA_DDR = 0xFF;
    //Porty sterujące OLEDa jako wyjścia ze stanem "1"
    CONTROL_PORT |= (1<<WR_NR)|(1<<CS_NR)|(1<<DC_NR)|(1<<RST_NR);
    CONTROL_DDR |= (1<<WR_NR)|(1<<CS_NR)|(1<<DC_NR)|(1<<RST_NR);
    //Zerowanie sprzętowe sterownika SSD1309
    RESET_RST;
    _delay_ms(1);
    SET_RST;
    _delay_ms(1);
    //Konfiguracja wszystkich parametrów sprzętowych sterownika SSD1309
    writeCmd(SET_DISPLAY_OFF);
    writeCmd(SET_MEM_ADDR_MODE);
    writeCmd(HORIZONTAL_ADDRESSING);
    writeCmd(SET_CONTRAST_CTRL);
    writeCmd(0x80);
    writeCmd(SET_DISPLAY_START_LINE|0); //Start Line = 0
    writeCmd(SET_SEGMENT_REMAP_INVERSED);
    writeCmd(OUTPUT_FOLLOWS_RAM); //Entire Display On
    writeCmd(SET_NORMAL_DISPLAY);
    writeCmd(SET_MULTIPLEX_RATIO);
    writeCmd(0x3F); //1/64 Duty
    writeCmd(MASTER_CONFIGURATION);
    writeCmd(0x8E); //Select external VCC supply
    writeCmd(SET_COM_SCAN_DIR_INVERSED);
    writeCmd(SET_DISPLAY_OFFSET); //Set Display Offset
    writeCmd(0x00); //Default: 0
    writeCmd(SET_DISPLAY_CLOCK);
    writeCmd(0xF0); //ew. 0x00: 105HZ
    writeCmd(SET_PRECHARGE_PERIOD); //Set Pre-charge Period
    writeCmd(0xF1); //ewentualnie FF
    writeCmd(SET_COM_PINS_HARDWARE_CONFIG);
    writeCmd(0x12); //Alternative COM Pin
    writeCmd(SET_VCOMH_DESELECT_LVL);
    writeCmd(0x40); //ew. 0xFF: -0.84xVCC
    OLEDclearArea(0, 0, 127, 7); //CLS
    writeCmd(SET_DISPLAY_ON);
}
```

Listing 15. Funkcja narzędziowa odpowiedzialna za ustawienie aktywnego obszaru ekranu, w ramach którego przeprowadzany jest zapis do pamięci ekranu sterownika SSD1309

```
//Column: 0...127, Page: 0...7
void OLEDSetActiveWindow(uint8_t startColumn, uint8_t startPage, uint8_t endColumn, uint8_t endPage)
{
    writeCmd(SET_COLUMN_ADDRESS);
    writeCmd(startColumn);
    writeCmd(endColumn);
    writeCmd(SET_PAGE_ADDRESS);
    writeCmd(startPage);
    writeCmd(endPage);
}
```

Listing 16. Funkcja narzędziowa odpowiedzialna za wyświetlenie obrazka

```
void OLEDdrawBitmap(uint8_t Column, uint8_t Page, const uint8_t *Bitmap)
{
    register uint8_t Width, Height;
    register uint16_t bytesToSend;
    Width = pgm_read_byte(Bitmap++); //Pierwszy bajt tablicy Bitmap to szerokość: 0...127
    Height = pgm_read_byte(Bitmap++)>>3; //Drugi bajt tablicy Bitmap to wysokość: 8, 16, 24...64 -> przeliczamy na bajty
    bytesToSend = Width*Height; //Liczba bajtów przeznaczonych do wysłania do OLEDA
    OLEDsetActiveWindow(Column, Page, Column+Width-1, Page+Height-1);
    while(bytesToSend--) writeData(pgm_read_byte(Bitmap++));
}
```

Listing 17. Definicja nowego typu danych odpowiedzialnego za przechowywanie parametrów bieżącej czcionki ekranowej

```
typedef struct
{
    uint8_t Width; //Rzeczywista szerokość znaku (px)
    uint8_t Height; //Rzeczywista wysokość znaku (bajty)
    uint8_t Interspace; //Odstęp pomiędzy znakami (px)
    uint8_t BytesPerChar; //Liczba bajtów danych tablicy wzorców na 1 znak
    uint8_t FirstCharCode; //Kod ASCII pierwszego znaku
    const uint8_t *Bitmap; //Wskaźnik to tablicy zawierającej wzorce poszczególnych znaków
} fontDescription;
```

Listing 18. Funkcja odpowiedzialna za ustawienie bieżącej czcionki ekranowej

```
void setFont(const fontDescription *Font)
{
    CurrentFont.Width = pgm_read_byte(&Font->Width); //Rzeczywista szerokość czcionki
    CurrentFont.Height = pgm_read_byte(&Font->Height); //Rzeczywista wysokość czcionki
    CurrentFont.Interspace = pgm_read_byte(&Font->Interspace); //Odstęp pomiędzy znakami
    CurrentFont.BytesPerChar = pgm_read_byte(&Font->BytesPerChar); //Liczba bajtów na definicję pojedyn. znaku
    CurrentFont.FirstCharCode = pgm_read_byte(&Font->FirstCharCode); //Kod ASCII definicji pierwszego znaku
    CurrentFont.Bitmap = (uint8_t*)pgm_read_word(&Font->Bitmap); //Wskaźnik do tablicy wzorców tej czcionki
}
```

Listing 19. Funkcja odpowiedzialna za rysowanie znaków, przy użyciu bieżącej czcionki ekranowej

```
void OLEDdrawChar(char Character, uint8_t Column, uint8_t Page, uint8_t Inverted)
{
    register uint8_t readByte;
    const uint8_t *dataPointer;
    register uint8_t bytesToSend;
    //Ustalamy adres początku wzorca znaku ASCII, który zamierzamy wyświetlić
    dataPointer = &CurrentFont.Bitmap[(CurrentFont.BytesPerChar*(Character-CurrentFont.FirstCharCode))];
    //Określamy okno zapisu by uprościć samą procedurę zapisu
    OLEDsetActiveWindow(Column, Page, Column+CurrentFont.Width-1, Page+CurrentFont.Height-1);
    //Określamy liczbę bajtów do wysłania
    bytesToSend = CurrentFont.BytesPerChar;
    while(bytesToSend--)
    {
        readByte = Inverted? ~pgm_read_byte(dataPointer++):pgm_read_byte(dataPointer++);
        writeData(readByte);
    }
}
```

Wykaz elementów:

Węzeł pomiarowy

Rezystory:

R1, R2: 47 kΩ (SMD 0805)

Kondensatory:

C1, C2: 100 nF (SMD 0805)

Półprzewodniki:

U1: ATtiny24 (SOIC14)

U3: TC1047A (SOT23)

Inne:

U2: RFM12B-866MHz (SMD)

BATT: koszyk baterii CR2032 typu

CONNFY DS1092-11-NBS

Moduł nadrzędny

Rezystory: (SMD 0805)

R1, R5, R6, R9: 47 kΩ

R2: 158 kΩ/1%

R3: 18,2 kΩ/1%

R4: 1 MΩ

R7, R8: 10 kΩ

Kondensatory: (SMD 0805)

C1, C5...C7: 10 μF/16 V (SMD „A”/3216-18R)

C2: 1 μF/16 V (SMD „A”/3216-18R)

C3, C8, C11...C15: 100 nF

C4: 1,5 nF

C9, C10 : 4,7 μF/20 V (SMD „A”/3216-18R)

Półprzewodniki:

U1: TPS61085 (TSSOP8)

U2: NCP1117DT33G (TO252)

U3: ATmega88 (TQFP32)

D1: SS14 (SMA)

Inne:

U4: RFM12B-866 MHz (SMD)

L1 - dławik mocy 3,3 μH typu DLG-0504-3R3

SELECT/ENTER - enkoder ze zintegrowanym

przyciskiem

PWR - gniazdo męskie katowe 90° 2pin

(NSL25-2W)

ZIF - złącze typu ZIF do montażu

powierzchniowego (raster 0,5 mm, 31-pin,

górny kontakt)

OLED - wyświetlacz OLED Winstar

WE0012864KSP3N00000 (128×64 px)

do inicjalizacji naszego wyświetlacza, ponieważ jak każde peryferium tego typu wymaga on ustawienia szeregu rejestrów konfiguracyjnych sterownika ekranu, których to wartości zależne są od właściwości wbudowanego panelu OLED, jak i specyfikacji producenta modułu. Stosowną funkcję inicjalizacyjną pokazano na **listingu 14**. Brak przeprowadzenia stosownej inicjalizacji w zasadzie uniemożliwia poprawne funkcjonowanie modułu OLED, ponieważ domyślne ustawienia rejestrów sterujących układu SSD1309 zwykle odbiegają od tych, wymaganych przez producenta panelu. Dalej, na **listingu 15** pokazano kolejną funkcję narzędziową odpowiedzialną za ustawienie aktywnego obszaru ekranu, w ramach którego przeprowadzany jest zapis do pamięci ekranu sterownika SSD1309. Jest ona bardzo użyteczna, gdyż jej użycie upraszcza a zarazem znacznie przyspiesza wyświetlanie obrazów, których rozmiar jest inny aniżeli całkowity, fizyczny rozmiar ekranu. Warto w tym miejscu zauważyć, iż jeden bajt pamięci obrazu odpowiada ośmiu pikselom obrazu ułożonym w pionie, stąd całą wysokość obrazu równą 64 piksele podzielono na 8 tzw. stron (Pages).

Dalej, na **listingu 16** pokazano funkcję umożliwiającą wyświetlenie obrazka,

którego treść znajduje się w tablicy, będącej argumentem wywołania, przy czym dwa pierwsze bajty tej tablicy określają szerokość i wysokość obrazka, zaś pozostałe bajty przechowują treść obrazka.

W tej chwili przyszedł czas na obsługę ostatniego elementu interfejsów graficznych, czcionek ekranowych! Aby jednak umożliwić wygodną obsługę wielu czcionek ekranowych, konieczne było wprowadzenie nowego typu danych, którego definicję pokazano na **listingu 17**. Bazując na zdefiniowanej strukturze wykonano funkcję, która korzystając z globalnej zmiennej

REKLAMA


```

Ustawienia fusebitów modułu nadrzędnego:
CKSEL3...0: 0010
SUT1...0: 10
CKDIV8: 0
CKOUT: 1
DWEN: 1
EESAVE: 0
Ustawienia fusebitów węzła pomiarowego:
CKSEL3...0: 0010
SUT1...0: 10
CKDIV8: 0
CKOUT: 1
DWEN: 1

```



Rysunek 8. Przykładowy wygląd interfejsu użytkownika

`static fontDescription CurrentFont` pozwala na ustawienie bieżącej czcionki ekranowej – listing 18. Funkcję umożliwiającą rysowanie znaków przy użyciu bieżącej czcionki ekranowej zamieszczono na listingu 19. Jak zwykle, do wygenerowania plików zawierających wzorce czcionek, oparte na czcionkach systemu Windows, polecam doskonały program **PixelLab** autorstwa **Marcina Popławskiego**, którego szczegółowy opis znalazł

się w Elektronice Praktycznej 06/2015. Tyle w kwestii obsługi naszego ciekawego wyświetlacza graficznego, którego zastosowanie, mam nadzieję, znacznie uatrakcyjni prezentowane urządzenie.

Obsługa

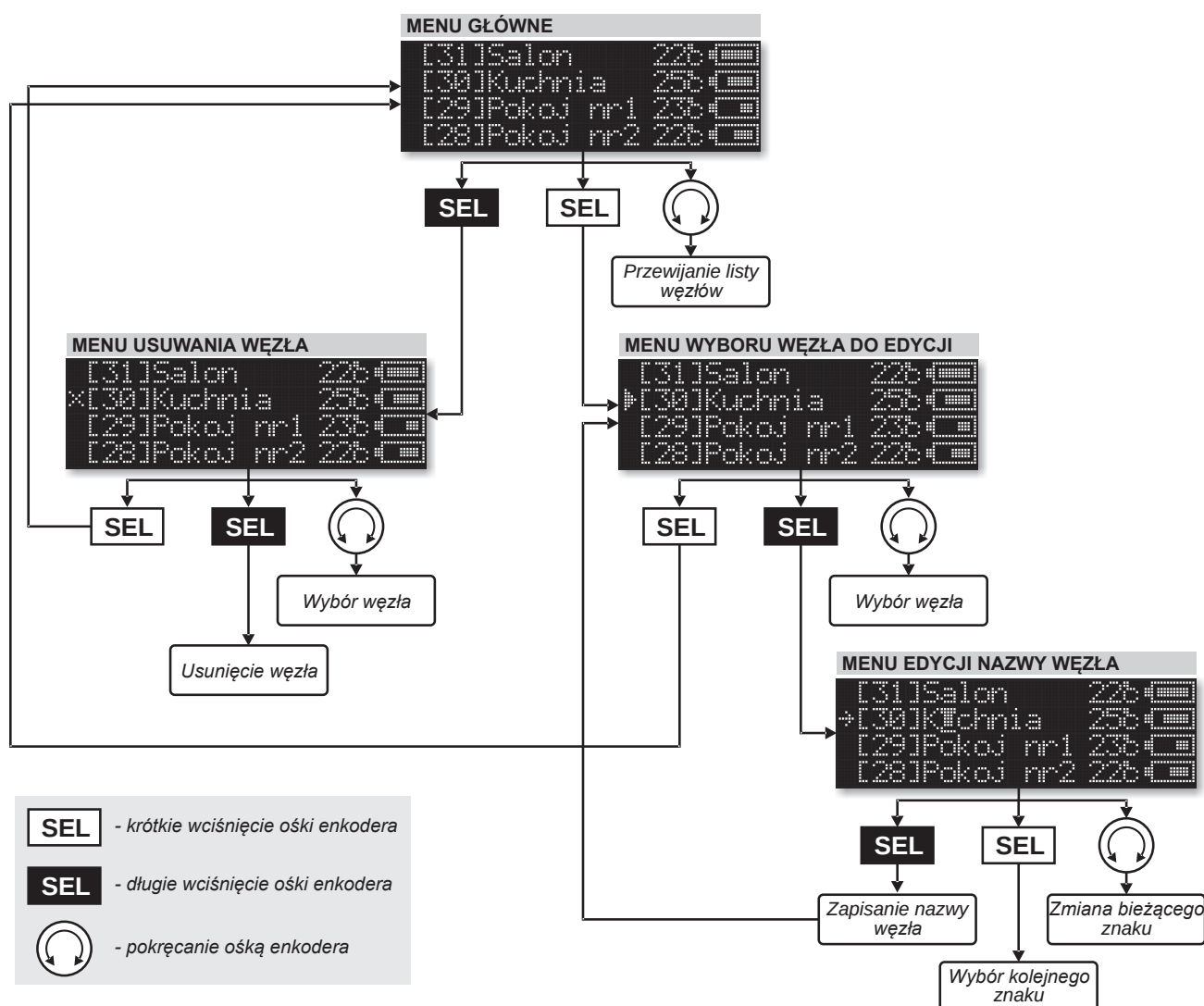
Projektując system bezprzewodowego pomiaru temperatury przyjąłem jako główne założenie maksymalną ergonomię obsługi, jak i brak potrzeby konfiguracji. W związku z powyższym, wszystko, co musimy zrobić, to nadać każdemu węzłowi sieci niepowtarzalny adres sprzętowy, posługując się zworkami na polach lutowniczych A0...A4, po czym włączyć go umieszczając baterię zasilającą. Po krótkiej chwili moduł taki zostanie wyświetlony w liście modułów w ramach graficznego interfejsu użytkownika modułu nadrzędnego i jeśli nie jest jeszcze znany (ma nowy, nieznany adres sprzętowy) zostanie mu nadana domyślna nazwa „No name”, którą następnie możemy poddać edycji. Przykładowy wygląd interfejsu użytkownika pokazano na rysunku 8. Na pierwszym miejscu jest pokazywany adres sprzętowy każdego węzła sieci, jego

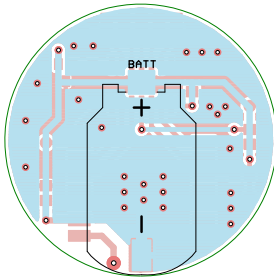
nazwa nadana przez użytkownika, następnie zmierzona temperatura (lub symbol ‘—’, w wypadku, gdy węzeł sieci przestanie wysyłać dane przez czas dłuższy niż 1 minuta) oraz stan baterii zasilającej w formie symbolu graficznego.

W ramach graficznego interfejsu użytkownika jednorazowo można wyświetlić dane 8 węzłów sieci. Ponadto, korzystając z enkodera z wbudowanym przyciskiem, możliwe jest przesuwanie listy węzłów (gdy jest ich więcej, niż 8), edycja ich nazw lub usuwanie węzłów. A wszystkie te czynności w wygodny sposób wykonujemy wyłącznie przy użyciu wspomnianego enkodera obrotowego. Wygląd wszystkich ekranów Menu wraz z funkcjonalnością enkodera obrotowego (w tym wbudowanego przycisku) pokazano na rysunku 9.

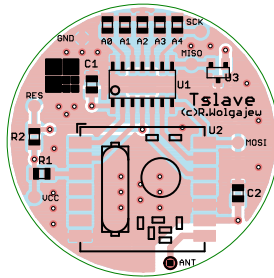
Montaż

Opis montażu systemu rozpoczniemy od montażu obwodu drukowanego węzła pomiarowego, którego schemat montażowy pokazano na rysunku 10. Montaż rozpoczynamy od przylutowania mikrokontrolera, następnie lutujemy czujnik TC1047, moduł

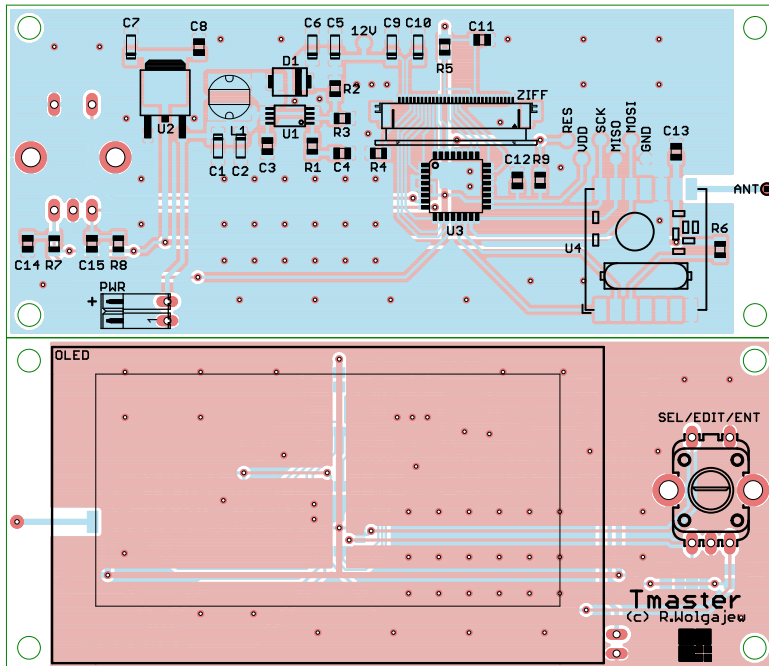




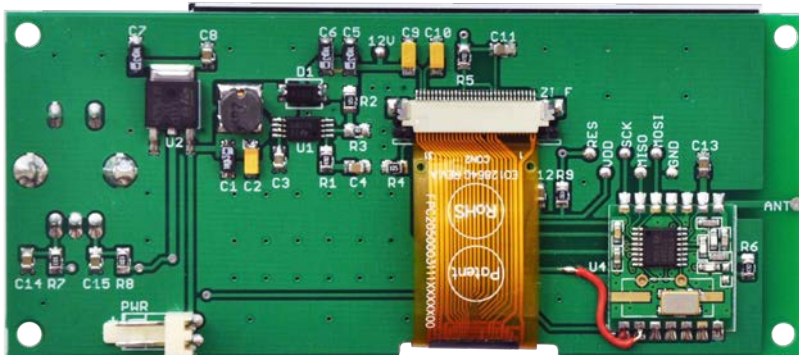
Rysunek 10. Schemat montażowy węzła pomiarowego



Fotografia 11. Widok zmontowanego węzła pomiarowego od strony TOP



Rysunek 12. Schemat montażowy modułu nadrzędnego



Fotografia 13. Wygląd zmontowanego obwodu drukowanego układu nadrzędnego od strony BOTTOM

RFM-12B a na końcu elementy bierne. Dalej, przechodzimy na warstwę BOTTOM, gdzie przylutowujemy koszyk baterii zasilającej. Do tak przygotowanej płytki przylutowujemy antenę nadawczą w postaci kawałka

przewodu o długości 164 mm. Widok zmontowanego węzła pomiarowego od strony TOP pokazano na **fotografii 11**.

W tym momencie przejdźmy do montażu modułu nadrzędnego. Jego schemat

montażowy pokazano na **rysunku 12**. Z uwagi na fakt, iż moduł wyświetlacza OLED jest dołączony do płytki naszego urządzenia z użyciem gniazda ZIF o bardzo gęstym rastrze wyprowadzeń (0,5 mm), montaż modułu nadrzędnego rozpoczynamy właśnie od przylutowania wspomnianego gniazda. Najprostszym sposobem montażu elementów o tak dużym zagęszczeniu wyprowadzeń, niewymagającym jednocześnie posiadania specjalistycznego sprzętu, jest użycie typowej stacji lutowniczej, dobrej jakości cyny z odpowiednią ilością topnika oraz dość cienkiej plecionki rozlutowniczej, która umożliwi usunięcie nadmiaru cyny pomiędzy wyprowadzeń złącza. Należy przy tym uważać by nie uszkodzić termicznie tegoż elementu. Jakość tak wykonanego połączenia sprawdzamy pod lupą korzystając z najprostszego miernika pozwalającego sprawdzić ciągłość połączeń. Wspomniana kontrola będzie znacznie łatwiejsza, jeśli zmontowaną płytkę przemyjemy alkoholem izopropylowym w celu wypłukania nadmiaru kalafonii lutowniczej. Następnie lutujemy mikrokontroler, moduł RFM-12B, pozostałe półprzewodniki a na samym końcu elementy bierne oraz złącze zasilające. Dalej, przechodzimy na warstwę TOP, gdzie przylutowujemy nieliczne elementy bierne oraz enkoder obrotowy. Na samym końcu podłączamy wyświetlacz OLED do złącza ZIFF po stronie BOTTOM, zaś sam element przyklejamy po stronie TOP (w miejscu wyznaczonym obrysem) korzystając z dwustronnej taśmy klejącej.

Podobnie, jak poprzednio, do tak przygotowanej płytki przylutowujemy antenę nadawczą w postaci kawałka przewodu o długości 164 mm. Wygląd zmontowanego obwodu drukowanego układu nadrzędnego od strony BOTTOM pokazano na **fotografii 13**.

Robert Wołgajew, EP

<http://sklep.avt.pl>