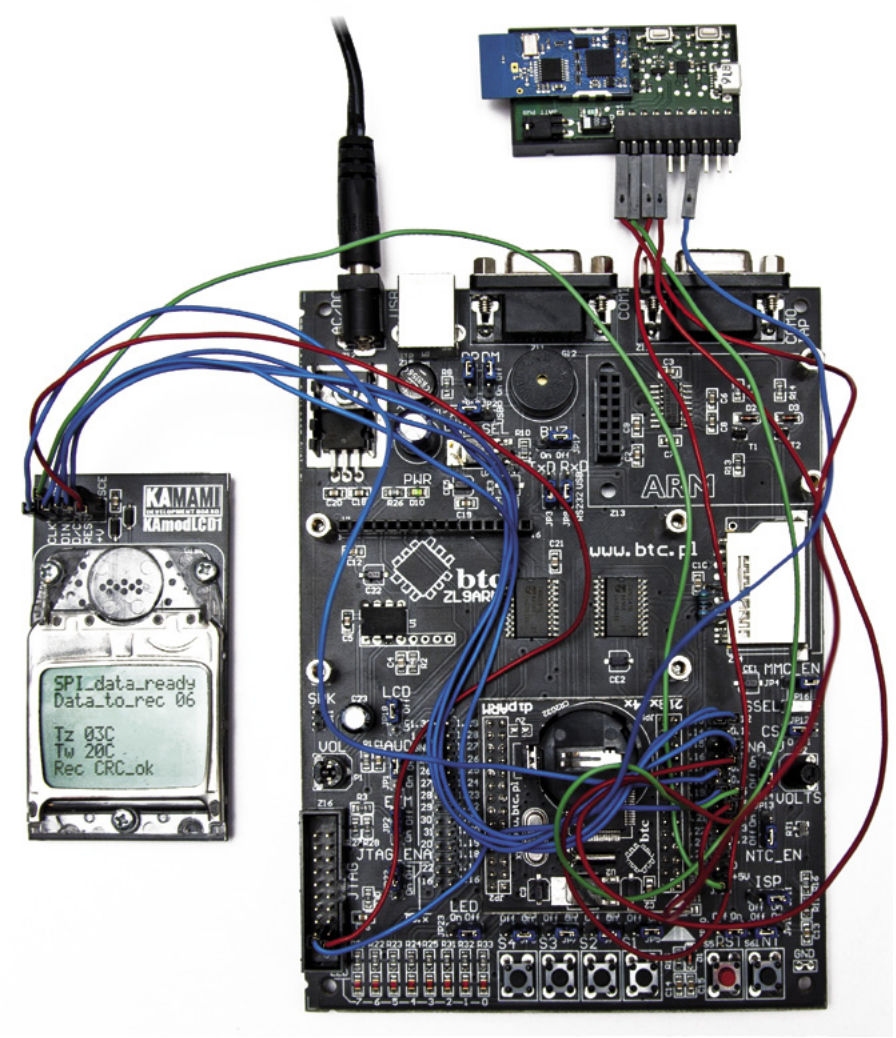
IQRF więcej niż radio

Host radiowy

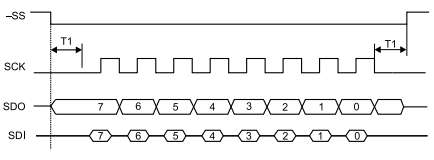
Każdy, kto zamierza wykorzystać w praktyce moduły radiowe firmy Microrisc musi dojść do etapu projektu sterownika nadrzędnego – hosta. W artykule opisano sposób wykonania i oprogramowania hosta radiowego zbudowanego z użyciem modułu TR52B.



Moduł TR52B może komunikować się z układem nadrzędnym poprzez interfejs szeregowy SPI. Można również używać innych interfejsów, na przykład I²C, ale wtedy na programiście spoczywa obowiązek całkowitego oprogramowania transmisji nie tylko po stronie hosta, ale także i modułu radiowego. Transmisja przez SPI jest wspierana przez wbudowany OS i co ważne odbywa się w tle programu głównego realizowanego przez moduł. Jest ona wyczerpująco opisana w dokumencie „*SPI Implementation in IQRF TR module*".

Poprzednio, przy okazji opisywania testowania połączenia pomiędzy zaprogramowanym modułem TR52B i aplikacją IQRF IDE (przykład 4) dokładnie opisałem podstawowe zasady wymiany informacji poprzez interfejs SPI. Tu dla przypomnienia opiszę jedynie najważniejsze z nich.

Moduł TR52B jest urządzeniem *SPI slave*, a rolę *SPI master* może pełnić np. mikrokontroler. W takiej konfiguracji układ master generuje sygnał zegarowy oraz sygnał aktywacji interfejsu w układzie slave (*SS, Slave Select*). Rozdzielone linie danych wyjściowych MOSI i danych wejściowych MISO (od strony układu nadrzędnego master) pozwalają na transmisję w trybie dupleks. Kiedy master chce wysłać na przykład 8-bitowa daną, to generuje 8 taktów sygnału zegarowego transmisji. W standardzie SPI dane są zapisywane do układu slave i jednocześnie z niego odczytywane w czasie narastającego lub opadającego zbocza sygnału zegarowego.



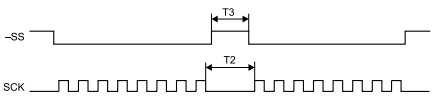
Rysunek 1 Przesyłanie danych za pomocą interfejsu SPI

W wypadku modułów TR52B dane są zapisywane i odczytywane narastającym zboczem zegara (**rysunek 1**).

Obsługa SPI przez OS nakłada pewne ograniczenia czasowe w trakcie przesyłania danych Na **rysunku 2** pokazano ograniczenia czasowe transmisji. Pomiędzy przesłaniem bajtów musi być zachowany odstęp minimum 100 µs dla zegara SCK=250 kHz. Typowy czas T2 jest równy 500 µs. Jest to dość istotne ograniczenie i w praktyce musi być dokładnie przestrzegane.

Sprzęt

Do budowy hosta można użyć dowolnego mikrokontrolera. Nie musi mieć wbudowanego sprzętowego interfejsu SPI, bo łatwo go emulować programowo, ale interfejs SPI jest tak popularny, że większość mikrokontrolerów ma wbudowany taki moduł i warto go wykorzystać. Ja użyłem zestawu ewaluacyjnego ZL9ARM z mikrokontrolerem LPC2148. Mikrokontroler ma odpowiednio duże zasoby, w tym 2 sprzętowe interfejsy SPI i co ważne – jest zasilany napięciem +3,3 V. Zapewnia to zgodność poziomów

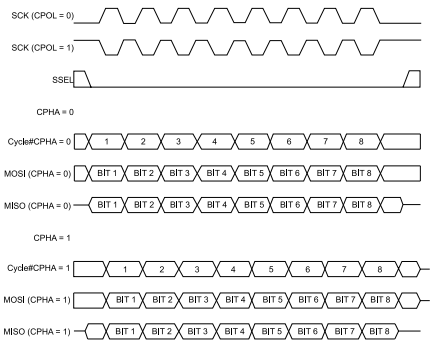


SCK	250 kHz maks,
T1 – czas pomiędzy opadającym zboczem SS i opadającym zboczem SCK (rysunek 1)	10 µs
T2 – opóźnienie pomiędzy kolejnym bajtami	100 µs min., 500 µs typowo
T3 – impuls na SS pomiędzy bajtami	20 µs min.

Rysunek 2. Ograniczenia czasowe przy przesyłaniu danych za pomocą SPI

Bit	Symbol	Opis
1:0	--	Rezerwa
2	BitEnable	Stała długość danych 8 bitów Długość danych jest programowana bitami 11:8
3	CPHA	0 – dane są próbkowane pierwszym zboczem SCK 1 – dane są próbkowane drugim zboczem SCK
4	CPOL	0 – SCK aktywne w stanie wysokim (zbocze narastające) 1 – SCK aktywne w stanie niskim (zbocze opadające)
5	MSTR	0 – SPI SLAVE 1 – SPI MASTER
6	LSBF	dane są wysyłane od bitu b7 (MSB) dane są wysyłane od bitu b0 (LSB)
7	SPIE	– przerwanie od SPI zablokowane Przerwanie od SPI odblokowane
11:8	BITS	1000 8 bitów 1001 9 bitów 1010 10 bitów 1011 11 bitów 1100 12 bitów 1101 13 bitów 1110 14 bitów 1111 15 bitów 0000 16 bitów
15:12	----	rezerwa

Rysunek 3. Rejestr S0SPCR

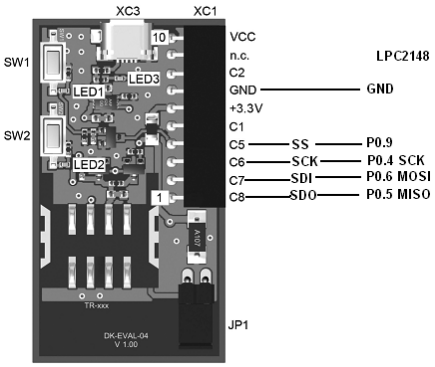


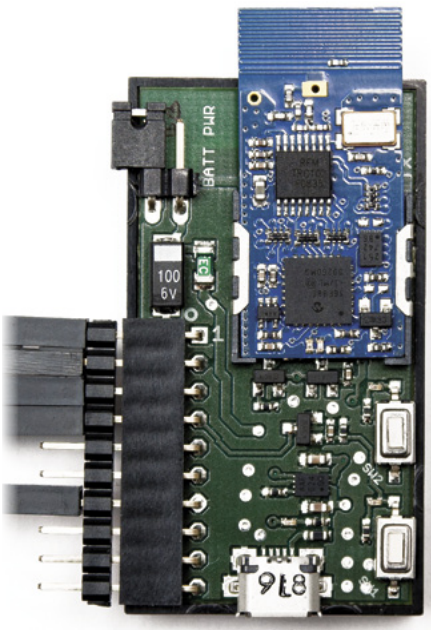
Rysunek 4. Przebiegi czasowe interfejsu SIP0

logicznych z modułem radiowym, który jest zasilany takim samym napięciem. Host miał mieć funkcję edukacyjną, więc wyposażyłem go w monochromatyczny wyświetlacz od telefonu komórkowego Nokia 3310 na module KAmodLCD1.

LPC2148 ma wbudowane 2 interfejsy oznaczone jako SPI0 i SPI1. Interfejs SPI1 (inaczej nazwany SSP) jest rozbudowany: ma 8 poziomowy bufor FIFO i można go zaprogramować również do pracy z magistralą SSI (*Texas Instruments*) oraz Microwire (*National Semiconductor*). Wbudowany bufor FIFO znacznie ułatwia intensywną wymianę danych pomiędzy układami. Jednak nie zdecydowałem się użyć go w przykładowych programach komunikacji z modułem radiowym. Uznałem, że obsługa buforów FIFO nieco zaciemni przykłady. Dlatego SPI1 został użyty do obsługi wyświetlacza, a SPI0 do komunikacji z modułem TR52B.

Moduł SPI0 może być zaprogramowany jako master lub slave do przesyłania danych w trybie *full duplex*. Długość danych ustawia się od 8 do 16 bitów na transfer (ramkę), z krokiem co 1 bit. Prędkość transmisji (częstotliwość sygnału zegarowego) jest ustawiana przez programowanie dedykowanego rejestru *Clock Counter Register*. Przed użyciem moduł SPI wymaga zainicjowania. W mikrokontrolerach LPC2148 przypisanie linii wewnętrznych interfejsów do fizycznych wyprowadzeń jest programowane przez zapisywanie rejestrów PINSEL0 i PINSEL1. Domyślnie po zerowaniu do wyprowadzeń są dołączone linie portów. Jeżeli chcemy stosować interfejsy SPI, trzeba ich linie przypisać do wyprowadzeń mikrokontrolera. InterfejsSPI0 jest konfigurowany 16-bitowym rejestrem S0SPCR (**rysunek 3**). Na **rysunku 4** pokazano przebiegi czasowe sygnałów





Rysunek 5. Połączenie TR52B z hostem

interfejsu zależnie od wybranego za pomocą bitów CPOL i CPHA trybu pracy SPI. Z porównania rysunków 1 i 4 wynika, że trzeba

Listing 1. Inicjalizacja modułów SPI hosta

```
void init_hwd(void){
char i, temp;
  //linia zerowania wyświetlacza
  IO0DIR|=RES;//RES wyjście
  IOCLR0|=RES;//RES stan niski
  PINSEL0=0x00001540;//przypisanie  linii portu do SPI0
  PINSEL1=0x000000A8;//przypisanie linii portu do SPI1
  //konfiguracja SPI0
  S0SPCR=0x0020;//SPI0 Master
  S0SPCCR=16;//prędkość na SPI0
  //konfiguracja SPI1 (SSP)
  SSPCR0=0x0907;//konfiguracja SPI1
  SSPCPSR=20;//prędkość transmisji na SPI1
  SSPCR1=2;//odblokowanie SSP1
  for(i=0;i<8;i++) //wyczytanie bufora FIFO SPI1
   temp=SSPDR;
  //konfiguracja linii wyświetlacza i linii SSEL modułu TR52B
  IO0DIR|=DC;//C/D wyjście
  IOSET0|=DC;//C/D stan wysoki
  IO0DIR|=SSEL;//SSEL wyjście
  IOSET0|=SSEL;//SSEL stan wysoki
  IO0DIR|=SCE;//SCE wyjście
  IOSET0|=SCE;//SSEL1 stan wysoki
}
```

Listing 2. Zapisanie i odczytanie bajtu prze moduł SPI0

```
unsigned char  PutSPI(unsigned char data){
char status,data_out;
  delay();//delay();delay();delay();delay();//opóźnienie
  IOCLR0|=SSEL;//linia SSEL aktywna
  S0SPDR=data;
  while((S0SPSR&0x80)==0);//czekaj na ustawienie SPIF
  status=S0SPSR;          //zerowanie SPIF
  data_out=S0SPDR;
  IOSET0|=SSEL;
  return(data_out);
}
```

wyzerować bity CHPA, CPOL, LSFB=0 oraz BitEnable. Po zaprogramowaniu interfejsu ustawiamy częstotliwość zegara zapisując rejestr S0SPCCR. Na listingu 1 zamieszczono procedurę inicjalizacji obu interfejsów SPI.

Zainicjowany moduł interfejsu zaczyna wysyłać i odbierać dane po zapisaniu rejestru S0SPDR. Rejestr można zapisać i odczytywać tylko wtedy, kiedy zakończono poprzednią operację transferu danych. Jest to sygnalizowane ustawieniem flagi SPIF w rejestrze statusowym S0SPSR. W przeciwnym wypadku wystąpi kolizja sygnalizowana bitem WCOL (dla zapisu) i ROVR (dla odczytu). Na listingu 2 pokazano procedurę zapisu bajtu (*unsigned char*), która jednocześnie zwraca odczytany bajt.

Po prawidłowym skonfigurowaniu modułu można podłączyć linie interfejsu SPI mikrokontrolera z modułem TR52B. Ja do tego celu użyłem firmowego modułu DK-EVAL-04 (**rysunek 5**). W trakcie pracy TR52B jest zasilany z baterii umieszczonej w DK-EVAL-04.

Jak już wiemy host jest masterem, a moduł układem slave. Obsługa interfejsu SPI w OS IQRF jest zaimplementowana jako maszyna stanu. Moduł TR52B może sam sprawdzić stan magistrali SPI używając funkcji *getStatusSPI()*. Host musi sekwencyjnie pytać moduł radiowy o status SPI wysyłając komendę *SPI_CHECK* (bajt 0x00). Na to pytanie slave odpowiada bajtem statusu (**rysunek 6**).

Procedura GetStatus pokazana na listingu 3 wysyła do TR52B bajt komendy SPI_CHECK i zwraca odczytany bajt statusu.

Jak widać odczytywanie statusu jest tak proste jak to tylko możliwe. Host wysyła jeden bajt bez sumy kontrolnej, a TR52B zwraca również jeden bajt. Jest to uzasadnione tym, że testowanie statusu może być wykonywane dość często. W procedurze z listingu 3 dla celów testowych jest wyświetlany typ wybranych statusów. W trakcie normalnej pracy hosta można te komunikaty usunąć, lub zostawić tylko informację o błędach.

Aby moduł radiowy mógł komunikować się z hostem musi zostać skonfigurowany komendą *enableSPI()*. Jeżeli się jej nie wykona, to na zapytanie hosta o status moduł odpowie bajtem 0x00 (SPI nieaktywne). Po wykonaniu komendy *enableSPI()*, kiedy nie ma danych do przesłania, moduł radiowy na zapytanie o status odpowiada bajtem 0x80 (SPI gotowe). Jeżeli host „spodziewa się" danych z modułu, to oczekuje najpierw statusu 0x80. Jest to informacja, że urządzenia mogą przesyłać dane pomiędzy sobą. Moment, w którym moduł ma rzeczywiste dane do przesłania jest sygnalizowany statusem o wartości 0x40+*ld*, gdzie *ld* jest liczbą bajtów. Jeżeli host odbierze taki status, to może odebrać dane wysyłając komendę *SPI_CMD*. Składa się ona z kodu (0xF0), bajtu PTYPE, bajtów danych i bajtu sumy kontrolnej CRC (**rysunek 7**).

Wartość hex	status
00	SPI nie aktywne (wyłączone przez komendę disableSPI())
07	SPI zatrzymane przez komendę stopSPI()
3F	SPI nie gotowe (pełny bufor, ostatnie CRC prawidłowe). Dalsza transmisja odblokowywana funkcją startSPI(0)
3E	SPI nie gotowe (pełny bufor, ostatnie CRC nie prawidłowe). Dalsza transmisja odblokowywana funkcją startSPI(0)
41 do 40+Nmax	SPI gotowe do przesłania tego statusu -40hex bajtów
80	SPI gotowe – tryb komunikacji
81	SPI gotowe – tryb programowania
82	SPI gotowe – tryb debugowania
83	Zarezerwowane dla starszych wersji modułów radiowych
FF	SPI nie aktywne – błąd sprzętowy

Rysunek 6. Status modułu

Master	SPI_CMD	PTYPE	DM_1	DM_2	…	$DM_{SPIDLEN}$	CRCM
Slave	SPISTAT	SPISTAT	DS_1	DS_2	…	$DS_{SPIDLEN}$	CRCS

Rysunek 7. Komenda SPI_CMD

b7	b6	b5	b4	b3	b2	b1	b0
CTYPE		SPIDLEN					

Pole SPIDLEN – liczba przesyłanych bajtów (od 1 do Nmax=41)
Pole CTYPE
B7=1 dane przesyłane przez master są ważne i zostaną zapisane do bufferCOM
B7=0 bufferCOM nie zostanie zmieniony

Rysunek 8. Struktura bajtu PTYPE

Magistrala SPI po wysłaniu bajtu z mastera automatycznie odbiera bajt ze slave. Daje to różne możliwości. Host – master może chcieć odczytać dane z modułu radiowego – slave, ale nie musi w tym celu przesyłać ważnych danych. Bajty wysyłane do slave służą tylko do wygenerowania cykli zegarowych taktujących przesyłaniem danych. Oczywiście nie musi, ale może. O tym czy dane przesyłane do modułu są ważne, czy tylko służą do odczytania danych z modułu decyduje host w bajcie PTYPE wysyłanym jako drugi po kodzie komendy. PTYPE pełni dwie funkcje: zawiera informację o liczbie bajtów danych przesyłanych do modułu i informację, czy dalsze dane mają być przez moduł radiowy

Listing 3 Odczytanie statusu

```
unsigned char  GetStatus(void){
unsigned char status;
  delay();
  status=PutSPI(0);//wyślij komendę SPI_CHECK
  switch (status){
  case 0:
    DispLcd("SPI_not_active",0,0);
    break;
  case 0x3e:
    DispLcd("SPI_CRC_Err ",0,0);
    break;
  case 0x3f:
    DispLcd("SPI_full_buf  ",0,0);
    break;
  case 0x80:
    DispLcd("SPI_ready     ",0,0);
    break;
  }
  if(status>=0x40&&status<0x80)
  DispLcd("SPI_data_ready ",0,0);
  return(status);
}
```

Listing 4. Procedura wysłania komendy SPI_CMD

```
#define DATAS 1 //dane ważne
#define DATANS 0 //tylko odczytanie danych z modułu

unsigned char PutCmd(unsigned char ld, unsigned char trt, unsigned char *buf){
unsigned char crc,crco,i, ldd;
  ldd=ld;
  if(trt==DATAS)  //dane ważne dla modułu radiowego
    ld|=0x80;
  crc=0xf0;
  buf_cmd[0]=PutSPI(0xf0);  //wysłanie kodu komendy
  crc^=ld;  //oblicz CRC
  buf_cmd[1]=PutSPI(ld);  //wyślij PTYPE
  for(i=0;i<ldd;i++){  //wyślij zadaną ilość danych
    crc^=*(buf+i);  //oblicz crc
    buf_cmd[i+2]=PutSPI(*(buf+i));
  }
  crc^=0x5f;  //ostatni bajt CRC
  buf_cmd[i+2]=PutSPI(crc);  //wyślij CRC danych nadawanych
  crco=ld;  //CRC danych odbieranych =PTYPE
  for(i=2;i<ld+2;i++)
    crco^=buf_cmd[i];  //wyliczenie CRC danych odbieranych
  crco^=0x5f;
  if(crco==buf_cmd[i])
  return(0);  //powrót bez błędu CRC
  else return(1);  //powrót z błędem CRC
}
```

przyjęte jako ważne, czy odrzucone. Strukturę bajtu PTYPE pokazano na **rysunku 8**.

Po wysłaniu dwóch pierwszych bajtów moduł odpowiada statusem, a na każdy następny bajt danych przesyłanych przez hosta odpowiada swoimi danymi. Przypominam, że komenda *SPI_CMD* jest w tym wypadku wysyłana jako odpowiedź na odebranie statusu 0x40+ld, a to oznacza, że slave ma dla hosta dane do odebrania. Host może również wysłać dane do modułu w dowolnym momencie pod warunkiem, że wcześniej odczyta status 0x80.

Listing 5. Fragment programu inicjującego SPI oraz mierzącego temperaturę i wysyłającego 6 bajtów przez SPI (moduł połączony z hostem)

```
void APPLICATION()
{
  uns16 temperature;
  uns8 i, rec_ok;
  SWDTEN = 0;
  enableSPI();
  toutRF = 4;
  rec_ok=0;
  while (1){
    clrwdt();
    getTemperature();  // pomiar temperatury
    temperature.high8 = ADRESH & 0x03;  // zapis 10b wyniku
    temperature.low8  = ADRESL;  // w ADRESH i ADRESL
    temperature *= 75;
    temperature >>= 8;
    temperature -= 50;
    bufferCOM[0] = 'T'; bufferCOM[1] = 'w'; bufferCOM[2] = ' ';
    bufferCOM[3] = temperature.low8/10;
    bufferCOM[3] += '0';
    bufferCOM[4] = temperature.low8%10;
    bufferCOM[4] += '0';
    bufferCOM[5] = 'C';
    startSPI(6); // wysłanie przez SPI
.....
```

Listing 6. Oczekiwanie na odczytanie danych przez hosta.

```
while(1){
  clrwdt();//zeruj watchdoga
  if (getStatusSPI()) //czekaj na wysłanie bufora bufferCOM
    continue;
  else
    break;
}
```

Listing 7. Testowanie odbioru danych z hosta i poprawności CRC

```
if (_SPIRX)  // czy coś odebrano?
{
  clrwdt();
  if (_SPICRCok)  // suma CRC prawidłowa?
  {
    DLEN = SPIpacketLength;
    copyBufferCOM2RF();  // kopiuj dane z bufferCOM do bufferRF
    PIN = 0;
    RFTXpacket(); //wyślij dane drogą radiową
    startSPI(0); //zwolnienie magistrali
  }
}
```

Listing 8. Program modułu zdalnego TR52B

```
void APPLICATION()
{
uns16 temperature;
uns8 i, rec_ok;
  SWDTEN = 0;
  enableSPI();
  toutRF = 4;
  rec_ok=0;
  while (1){
    clrwdt();
    if (checkRF(5)){  // poziom sugnału odpowiedni?
      if (RFRXpacket()){  // odebrano pakiet?
        pulseLEDR();  // LED
        getTemperature();  //pomiar temperatury
        temperature.high8 = ADRESH & 0x03;  // wynik 10b
        temperature.low8  = ADRESL;
        temperature *= 75;         //konwersja
        temperature >>= 8;
        temperature -= 50;
        bufferRF[3] = temperature.low8/10;
        bufferRF[3] += '0';
        bufferRF[4] = temperature.low8%10;
        bufferRF[4] += '0';
        bufferRF[0] = 'T';
        bufferRF[1] = 'z';
        bufferRF[2] = ' ';
        bufferRF[5] = 'C';
        PIN = 0;
        RFTXpacket();
        pulseLEDG();               // obsługa LED
      }
    }
  }
}
```

Transmisja danych powinna zawsze być zabezpieczona sumą kontrolną CRC. Testowanie prawidłowości CRC zwiększa pewność, że przesyłane dane są prawidłowe. W systemie IRQF OS przesyłane dane są zabezpieczone CRC liczoną według zależności:

Dla danych wysłanych przez hosta CRCM = SPICMD ^ PTYPE ^ DM1 ^ DM2 ^ … . ^ DMspidlen ^ 0x5f, gdzie DM – 8 bitowe dane wysyłane przez mastera – hosta ,spidlen – ilość danych, a ^ = XOR

Dla danych odbieranych przez hosta CRCS = PTYPE ^ DS1 ^ DS2 ^ …… ^ DSspidlen ^ 0x5f,
gdzie DS – 8 bitowe dane przesyłane przez slave

Warto zauważyć, że do obliczania CRC dla danych odbieranych potrzebujemy wartości PTYPE wysyłanej w komendzie SPI_CMD do odebrania tych danych.

Mając te wszystkie wiadomości możemy napisać procedurę wysyłającą komendę SPI_CMD i jednocześnie odbierającą dane z modułu. Zamieszczono ją na **listingu 4**.

Argumentami tej procedury są:

ld – liczba danych do przesłania,

trt – identyfikator ważnych danych dla modułu,

*buf – wskaźnik na początek bufora z danymi do przesłania.

Jeżeli przesyłane dane mają być wpisane do *bufferCOM* modułu radiowego, to *trt* ma być równe *DATAS* (*PTYPE=ld|0x80*), a jeżeli nie, to równe *DATANS* (*PTYPE=ld*).

Procedura wysyła dane zawarte w buforze z argumentu, a odebrane z modułu TR52B dane są zapisywane do bufora buf_cmd. Są tu też obliczane obie sumy kontrolne: dla danych wysyłanych i odbieranych. Poza tym testowana jest poprawność CRC dla danych odbieranych i procedura zwraca kod 0x00 kiedy CRC jest poprawna, lub 0x01 kiedy wystąpił błąd.

Wyposażeni w procedury odczytywania statusu i wysyłania komendy SPI_CMD możemy odczytywać dane po SPI z modułu. Jednak, żeby to zrobić trzeba najpierw napisać program, który będzie odczytywać temperaturę z czujnika TR52B i przesyłać ją na żądanie hosta (**listing 5**).

Najpierw jest inicjowany interfejs SPI (*enableSPI*), a potem funkcja *getTemperature* odczytuje za pomocą wbudowanego przetwornika A/C temperaturę z czujnika umieszczonego na module. Po konwersji jest ona zapisana do bufora *bufferCOM* (uzupełniony o znaki „ Tw C"). Zapisany bufor *bufferCOM* jest gotowy do przesłania po wykonaniu komendy *startSPI(6)*. Kiedy teraz host zapyta o status, to TR52B odpowie bajtem 0x46. Jest to informacja dla hosta, że moduł ma do przesłania 6 bajtów i trzeba wysłać komendę *CMD_SPI* z bajtem *PTYPE=0x06*, lub *PTTPE=0x86*. Moduł radiowy powinien te-

Fotografia 9. Ekran wyświetlacza hosta w trakcie pomiaru temperatury

raz poczekać na zwolnienie magistrali zanim spróbuje przesłać następną porcję danych do hosta (**listing 6**). Po wykryciu, że host odebrał dane można sprawdzić, czy host przesłał dane do modułu i zapisał bufferCOM (**listing 7**).

Procedura z listingu 7 robi jeszcze coś. Jeżeli wykryto, że host przesłał ważne dane, to kopiuje zawartość bufferCOM do bufferRF i wysyła je drogą radiową do innego „zdalnego" modułu TR52B. W ten sposób host wysyłając ważne dane poprzez SPI do modułu może je przesłać drogą radiową do kolejnego modułu, lub do kilku innych modułów. Zawartość przesyłanej informacji może na przykład adresować jeden z kilku modułów i tylko ten zaadresowany zareaguje na odebraną informację. Sposób adresowania zależy tylko od inwencji programisty. W prezentowanym przykładzie radiowo wysyłam informację tylko do jednego modułu z zapytaniem o jego temperaturę. Zapytanie dochodzi do skutku, kiedy host wysyła do modułu połączone z nim po SPI ważne dane (bit b7 PTYPE jest ustawiony).

W tym momencie host odczytuje temperaturę modułu do niego podłączonego po SPI i radiowo wysyła zapytanie do zdalnego modułu. Zdalny moduł musi odebrać pakiet, odczytać i przekonwertować temperaturę i odesłać go do modułu przy hoście. Na listingu 8 jest pokazany kompletny program wykonujący to zadanie.

Po konwersji wartości temperatury na postać ASCII, do bufora bufferRF jest dodawany tekst „Tz C" a całość jest transmitowana do modułu połączonego z hostem. Po ich odebraniu z bufora *bufferRF* trzeba skopiować do *bufferCOM* za pomocą interfejsu SPI przesłać do hosta. Jest to wykonywane przez fragment programu pokazany na **listingu 9**.

Wykonanie komendy startSPI(DLEN) spowoduje, że host po zapytaniu o status odczyta bajt 0x40+DLEN. Jak wiemy w takim przypadku wyczytywanie danych odbywa się komendą PutCmd (**listing 10**).

Listing 9. Odebranie danych przesłanych przez radio
```
if (RFRXpacket()){  // czy odebrano dane z radia
  pulseLEDG();  // sygnalizacja LED
  copyBufferRF2COM();  // kopiowanie bufferRF do bufferCOM
  waitDelay(2);
  startSPI(DLEN);  // wysłanie odebranej informacji przez SPI
}
//odczytaj status i wyjdź z pętli kiedy magistrala wolna
while(1){
  clrwdt();
  if (getStatusSPI()) continue; else break;
}
```

Listing 10. Odczytanie danych po SPI i wyświetlenie na wyświetlaczu (host).
```
if(PutCmd((status&0x0f),DATAS, buf_data)==0){
  DispLcd("Rec CRC_ok   ",0,5);
  if(buf_cmd[3]=='w')
  Poz(0,4);  //pozycja na LCD dla temperatury wewn. Tw
  else Poz(0,3); //pozycja na LCD dla temperatury zewn. Tz
  WriteChar(buf_cmd[2]);// wyświetlanie danych przesłanych SPI
  WriteChar(buf_cmd[3]);
  WriteChar(buf_cmd[4]);
  WriteChar(buf_cmd[5]);
  WriteChar(buf_cmd[6]);
  WriteChar(buf_cmd[7]);
} else{
  DispLcd("Rec data_error",0,4);
  DispLcd("Rec CRC_error ",0,5);}
}
```

Listing 10. Pętla odbioru i wyświetlania danych przez hosta
```
while(1){
  del();
  while(1){
    status=GetStatus();
    if(status==0x80)
    break;
    if((status&0x40)>=0x40&&status<=0x80)
    break;
  }
  //petla oczekiwania na dane
  while(1){
    status=GetStatus();
    if((status&0x40)>=0x40&&status<=0x80){
      DispLcd("Data_to_rec ",0,1);
      DispHex(status&0x3f);
      break;
    }
  };
  buf_data[0]=1;
  buf_data[1]=1;
  if(PutCmd((status&0x0f),DATAS, buf_data)==0){
    DispLcd("Rec CRC_ok   ",0,5);
    if(buf_cmd[3]=='w') Poz(0,4); else Poz(0,3);
    for(i=1; i++; i<8) WriteChar(buf_cmd[i]);
  } else {
    DispLcd("Rec data_error",0,4);
    DispLcd("Rec CRC_error ",0,5);}
  }
}
```

Fragment programu z list. 10 odczytuje dane przesyłane za pomocą SPI nie wiedząc czy pochodzą one z czujnika temperatury w module TR52B połączonego z hostem, czy z czujnika temperatury z modułu TR52B połączonego drogą radiową. Można to rozróżnić i wyświetlać pomiar w osobnych linijkach analizując zawartość danych. Tutaj jest analizowana zawartość drugiego bajtu pola danych. Jak pamiętamy, temperatura wewnętrzna jest przesyłana ze znakami „Tw", a zewnętrzna ze znakami „Tz". Na **listingu 11** pokazano kompletną pętla odbioru i wyświetlania danych przez hosta, natomiast na **fotografii 9** wynik jej funkcjonowania.

Podsumowanie

Przykładowe procedury starałem się w sposób jak najłatwiejszy do samodzielnej analizy. Dlatego nie ma w nich typowych zabezpieczeń typu timeout'y, a programy całkowicie wstrzymują swoje działanie czekając na przykład na status odebrania danych, czy zwolnienie magistrali. W praktyce takie rozwiązania mogą być możliwe nie do zaakceptowania, szczególnie w projektach, w których komunikacja z modułami radiowymi jest jednym z elementów większej całości. Wymianę informacji pomiędzy hostem, a modułem można zaimplementować jako maszynę stanu wykorzystując przerwania od licznika i modułu SPI. Innym rozwiązaniem mogłoby być zastosowania do obsługi jednego z wątków system RTOS. Mimo tego opisany projekt hosta mierzącego i wyświetlającego temperaturę z dwóch czujników został sprawdzony w praktyce, a w czasie trwających wiele godzin testów nie zauważyłem żadnych problemów z działaniem.

Tomasz Jabłoński, EP