Jak „oswoić” i skłonić do pracy transceiver ISM CC1100, część 2

W artykule znajdują się wskazówki jak wykorzystać i oprogramować układ radiowego transceivera firmy Texas Instruments. Poza tym będzie też garść praktycznych porad dla tych Czytelników, którzy po raz pierwszy stykają się z opisywanym układem. W tej części artykułu podajemy kilka podstawowych pojęć i przydatnych w późniejszej pracy procedur.

Procedury dostępu do rejestrów CC1100

Jak wynika z opisu zamieszczonego w poprzednim fragmencie dostęp do wewnętrznych rejestrów CC1100 poprzez magistralę SPI można zrealizować używając 5 różnych komend. Są to:

SpiReadReg – odczyt z rejestru w trybie „single byte”

SpiWriteReg – zapis do rejestru w trybie „single byte”

SpiReadBurstReg – odczyt rejestrów w trybie „burst”

SpiWriteBurstReg – zapis rejestrów w trybie „burst”

SpiWriteCommand – przesłanie do rejestru komendy

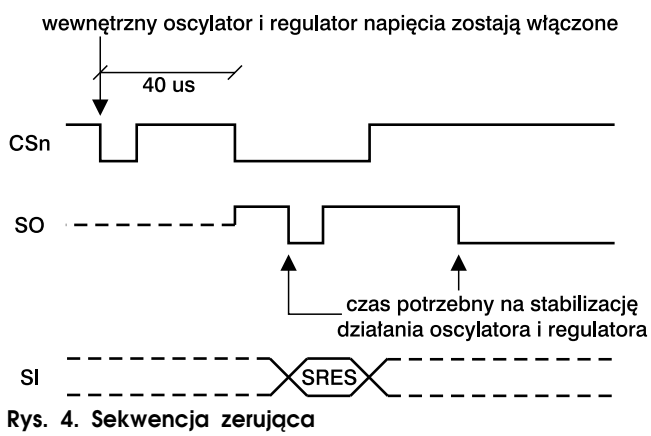
Dla lepszej czytelności, na zamieszczonych listingach zostanie pokazany przykładowy kod 5 komend, choć oczywiście wszystkie warianty zapisu i odczytu rejestrów można opracować w jednej procedurze. Procedury są bardzo proste i mam nadzieję czytelne, a do omówienia jako przykład niech posłuży kod procedury *SpiReadReg*.

Procedura jest wywoływana z parametrem `adres`, w którym przekazywany jest numer odczytywanego rejestru w trybie „single byte”. Po zakończeniu procedura zwraca wartość odczytaną z rejestru. W pierwszej instrukcji na wszelki wypadek zerowany jest stan wyjścia taktującego magistralę SPI, a w następnej poprzez podanie stanu niskiego na wejście CSn magistrala jest otwierana. Potwierdzeniem otwarcia magistrali jest stan niski na wyjściu SO CC1100. Dalej zerowane są bity b.7 i b.6 adresu oraz jest ustawiany b.7 przy

List. 3. Procedura SpiReadReg

```
//odczyt grupowy zawartości rejestrów
//-------------------------------------------
void SpiReadBurstReg(unsigned char adres, unsigned char *pbufor, unsigned
char ile)
//we: adres - adres pierwszego odczytywanego rejestru,
//*pbufor - wskaźnik do buforu gdzie mają się znaleźć odczytane z rejestrów
dane
//ile - liczba rejestrów do odczytu
{
 unsigned char x, y, maska, wartosc;

 SCLK_RF_PORT =0;
 CS_RF_PORT =0;
 while (SO_RF_PIN !=0);//oczekiwanie na stan niski na wyjściu SO
 adres =adres & 0b00111111; //wyzerowany b.7 i b.6
 adres =adres | 0b11000000; //b.7 i b.6 =1 odczyt w trybie „burst”
 maska =0b10000000;
 //wysłanie adresu pierwszego rejestru
 for (x=0; x<8; x++)
 {
  if ((adres &maska) ==0) SI_RF_PORT =0;
  else SI_RF_PORT =1;
  #asm(„nop”);
  SCLK_RF_PORT =1;
  #asm(„nop”);
  SCLK_RF_PORT =0;
  maska =maska >>1;
 }
 //odczyt kolejnych rejestrów
 for (y=0; y<ile; y++)
 {
  wartosc =0;
  maska =0b10000000;
  for (x=0; x<8; x++)
  {
   SCLK_RF_PORT =1;
   #asm(„nop”);
   if (SO_RF_PIN ==1) wartosc =wartosc |maska;
   #asm(„nop”);
   SCLK_RF_PORT =0;
   maska =maska >>1;
  }
  *pbufor =wartosc;
  pbufor++;
 }
 CS_RF_PORT =1;
}
```



Rys. 4. Sekwencja zerująca

List. 4. Procedura SpiWriteBurstReg

```
//grupowy zapis do kolejnych rejestrów
//-----------------------------------------------
void SpiWriteBurstReg(unsigned char adres, unsigned char *pbufor, unsigned char ile)
//we: adres - adres pierwszego zapisywanego rejestru,
//*pbufor - wskaźnik do buforu z wartościami które mają być wpisywane do rejestrów
//ile - liczba rejestrów do zapisu
{
 unsigned char x, y, maska, wartosc;
 SCLK_RF_PORT =0;
 CS_RF_PORT =0;
 while (SO_RF_PIN !=0);//oczekiwanie na stan niski na wyjściu SO
 adres =adres & 0b00111111; //wyzerowany b.7 i b.6
 adres =adres | 0b01000000; //b.7=0, b.6=1 zapis w trybie „burst”
 maska =0b10000000;
 //wysłanie adresu pierwszego rejestru
 for (x=0; x<8; x++)
 {
  if ((adres &maska) ==0) SI_RF_PORT =0;
  else SI_RF_PORT =1;
  #asm(„nop”);
  SCLK_RF_PORT =1;
  #asm(„nop”);
  SCLK_RF_PORT =0;
  maska =maska >>1;
 }
 //zapis do kolejnych rejestrów
 for (y=0; y<ile; y++)
 {
  maska =0b10000000;
  wartosc =*pbufor;
  for (x=0; x<8; x++)
  {
   if ((wartosc &maska) ==0) SI_RF_PORT =0;
   else SI_RF_PORT =1;
   #asm(„nop”);
   SCLK_RF_PORT =1;
   #asm(„nop”);
   SCLK_RF_PORT =0;
   maska =maska >>1;
  }
  pbufor++;
 }
 CS_RF_PORT =1;
}
```

List. 5. Procedura SpiWriteCommand

```
//wysłanie pojedynczego rozkazu
//--------------------------------------
void SpiWriteCommand(unsigned char adres)
{
//we: adres rejestru rozkazów
 unsigned char x, maska;
 SCLK_RF_PORT =0;
 CS_RF_PORT =0;
 while (SO_RF_PIN !=0);//oczekiwanie na stan niski na wyjściu SO
 adres =adres & 0b00111111;//wyzerowany b.7 i b.6
 adres =adres | 0b00000000;   //b.7 i b.6 =0 zapis pojedynczego rozkazu
 maska =0b10000000;
 //wysłanie adresu rejestru
 for (x=0; x<8; x++)
 {
  if ((adres &maska) ==0) SI_RF_PORT =0;
  else SI_RF_PORT =1;
  #asm(„nop”);
  SCLK_RF_PORT =1;
  #asm(„nop”);
  SCLK_RF_PORT =0;
  maska =maska >>1;
 }
 CS_RF_PORT =1;
}
```

List. 6. Procedura zerowania

```
//sekwencja zerowania transceivera po włączeniu zasilania
//-------------------------------------------------------------
void Rf_Power_Up_Reset(void)
{
 #define CCxxx0_SRES         0x30 //kod komendy SRES
 unsigned char maska, x;
 SCLK_RF_PORT =1;
 SI_RF_PORT =0;
 CS_RF_PORT =0;
 #asm(„nop”);
 #asm(„nop”);
 CS_RF_PORT =1;
 SCLK_RF_PORT =0;
 delay_us(40);          //pauza 40ms
 CS_RF_PORT =0;
 while (SO_RF_PIN !=0); //oczekiwanie na stan niski linii SO
 //wysłanie komendy CCxxx0_SRES
 SpiWriteCommand(CCxxx0_SRES);
 while (SO_RF_PIN ==0); //oczekiwanie na stan wysoki linii SO
 while (SO_RF_PIN !=0); //oczekiwanie na ponowny stan niski linii SO
 CS_RF_PORT =1;
}
```

wyzerowanym b.6, co oznacza, że będzie przeprowadzona procedura odczytu rejestru w trybie „single byte”. Następnie w pętli wysyłany jest bit po bicie bajt adresu. Bajt maski z przesuwaną jedynką służy do ustawiania na linii SI poziomu odpowiadającego stanom kolejnych bitów. Zamieszczone w kodzie komendy asemblerowe `nop` mają za zadanie wydłużyć zwłokę czasową pomiędzy kolejnymi krokami, ale nie są niezbędne. Z kolei w pętli w podobny sposób odczytywana jest bit po bicie zawartość rejestru. Zarówno zapis kolejnego bitu adresu, jak i odczyt kolejnego bitu danej taktowany jest impulsami zegara podawanymi na linię SCLK.

Pierwszy krok – zerowanie CC1100

Mając opracowane procedury dostępu do wewnętrznych rejestrów układu można przystąpić do takiego jego oprogramowania, aby działał w sposób przez nas oczekiwany. Jednak po włączeniu zasilania należy na samym początku przeprowadzić zalecaną przez producenta procedurę zerowania. Układ posiada co prawda wewnętrzne mechanizmy zerujące, ale jeśli czas narastania napięcia zasilającego jest zbyt powolny i nie ma pewności czy wewnętrzny oscylator osiągnął właściwą stabilność, warto przeprowadzić zalecaną procedurę. Składa się na nią wygenerowanie na linii CSn odpowiednich impulsów, obserwacja linii SO oraz wysłanie do układu komendy zerującej SRES. Całą sekwencję procedury zerującej pokazano na **rys. 4**, a na **list. 6** przedstawiono procedurę zerowania.

Linia `delay_us(40);` oznacza, że następuje odwołanie do podprogramu pauzy o długości 40 µs. Taką pauzę może też generować pętla typu `for(;;)` o liczbie powtórzeń zależnej od szybkości działania mikrokontrolera.

Udało nam się częściowo „oswoić” układ CC1100, ale to jeszcze za mało do tego, by skłonić go do pracy. Pełnię umiejętności nabędziemy w trzeciej części artykułu.

Ryszard Szymaniak, EP
ryszard.szymaniak@ep.com.pl