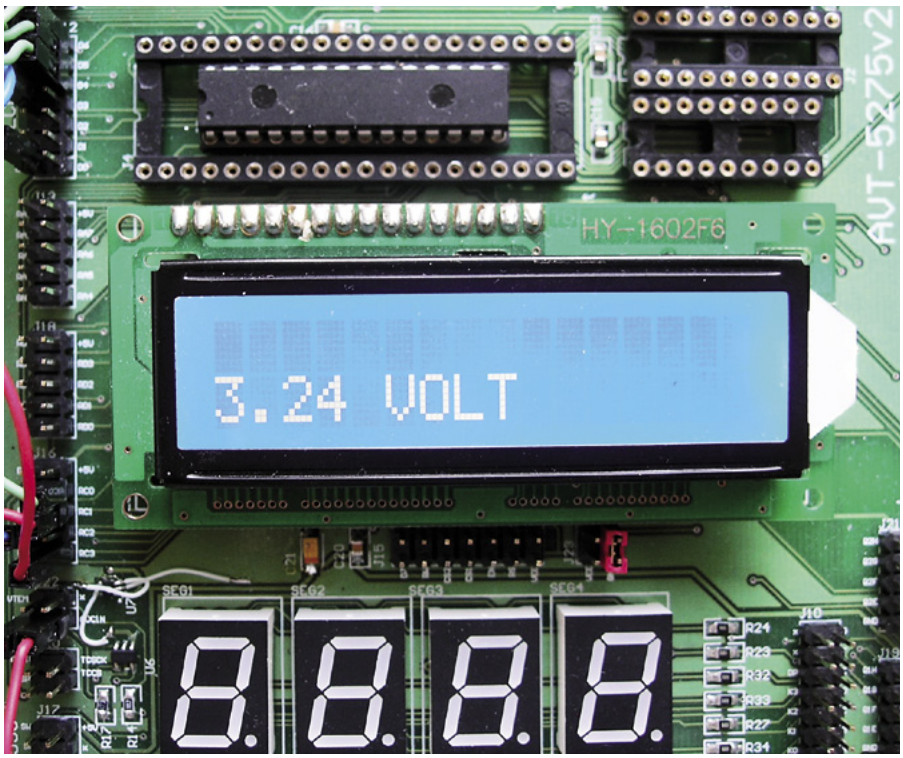
Kurs programowania mikrokontrolerów PIC(8)

Obsługa interfejsu szeregowego I²C

Interfejs I²C został zaprojektowany przez firmę Philips i jest przeznaczony do lokalnej wymiany danych pomiędzy układami. W najczęściej stosowanej konfiguracji do magistrali jest dołączony pojedynczy układ master (sterownik nadrzędny, najczęściej mikrokontroler) i jeden lub kilka układów slave (układów peryferyjnych). W bardziej zaawansowanych aplikacjach można do jednej magistrali dołączać wiele układów master. Komplikuje to jednak wymianę informacji, bo konieczne staje się wprowadzenie arbitrażu, ponieważ kiedy dwa lub więcej układów master próbuje uzyskać dostęp do magistrali w tym samym czasie, to pojawia się kolizja.

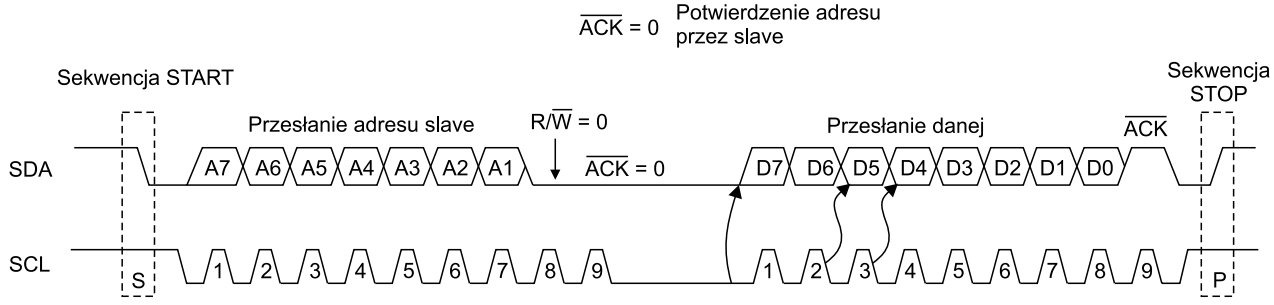


Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 15031, pass: 40nep417
• poprzednie części kursu

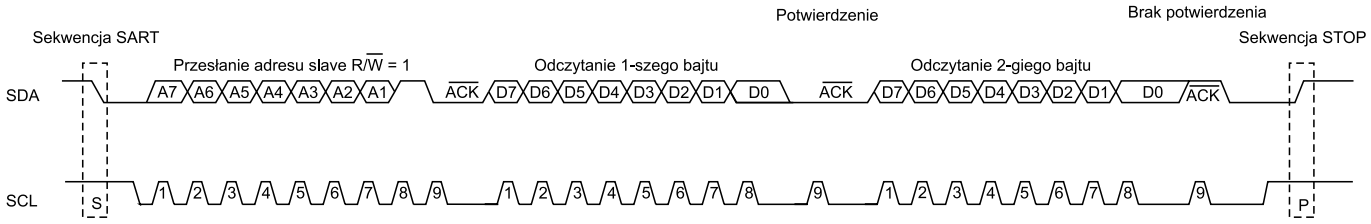
Układ *master* kontroluje cały ruch danych na magistrali: inicjuje i kończy transmisję, określa jej kierunek i generuje sygnał zegarowy transmisji na linii SCK. Funkcje układu *master* spełnia w większości wypadków pełni mikrokontroler. Układy slave to zewnętrzne pamięci, scalone czujniki (na przykład termometry), zegary czasu rzeczywistego RTC, ekspandery linii portów i wiele innych. Układem *slave* może też być odpowiednio zaprogramowany mikrokontroler.

Magistrala I²C jest zbudowana dwóch linii: SDA (danych) i SCL (zegarowa).Obydwie linie (SDA i SCL) są dwukierunkowe i muszą być dołączone do plusa zasilania przez rezystory zasilające. Gdy dane nie są przesyłane, to wyjścia układów dołączonych do magistrali znajdują się w stanie wysokiej impedancji i dzięki rezystorom zasilającym utrzymuje się na nich poziom wysoki. Te wyjścia są typu otwarty kolektor lub otwarty dren, co umożliwia realizację funkcji „iloczynu na drucie" (*wired–AND*) niezbędnej do prawidłowego działania procedur potwierdzania odbioru bitów, zgłoszenia się układów i innych związanych z transmisją.

Standardowo dane mogą być przesyłane z prędkością do 100 kbit/s lub z podwyższoną prędkością do 400 kbit/s. Istnieją również inne rozwiązania, ale są stosowane najczęściej. Liczba układów dołączonych do wspólnej magistrali jest ograniczona jej pojemnością – nie może być większa niż 400 pF i jest sumą pojemności ścieżek linii SDA lub



Rysunek 1. Zapisanie danej do układu slave



Rysunek 2. Odebranie 2 bajtów z układu slave

SCL oraz pojemności wejść i wyjść układów dołączonych do linii.

Każdy układ peryferyjny *slave* dołączony do jednej magistrali I²C musi mieć nadany unikalny adres uzupełniony o bit R/W (kierunku transmisji danych). W większości wypadków są produkowane i stosowane układy ze standardowym adresem 7-bitowym. Również zaawansowane aplikacje z wieloma masterami nie są zbyt często wykorzystywane w praktyce. Z tych powodów skupimy się na obsłudze adresowania 7-bitowego z jednym układem Master.

Przesyłanie danych magistralą I²C

Jak już wspomniano, transmisją danych przez magistralę I²C steruje układ *master*. Transmisję rozpoczyna sekwencja *Start*: poziom na linii SDA zmienia się z wysokiego na niski, podczas gdy na linii SCL panuje poziom wysoki. Jest to sekwencja unikalna, ponieważ w trakcie przesyłania danych poziom linii SDA może zmienić się tylko przy niskim poziomie linii zegarowej SCL. Po wykonaniu sekwencji *Start* magistrala przechodzi w stan zajętości, aż do momentu, kiedy *master* wygeneruje na magistrali sekwencję *Stop* kończącą transmisję. Sekwencja *Stop* również jest unikalna: poziom linii SDA zmienia się z niskiego na wysoki, gdy na linii SCL występuje poziom wysoki. Po wykonaniu sekwencji *Stop* magistrala przechodzi w stan spoczynkowy (*idle state*) i jest gotowa na następne przesłanie danych. 8-bitowe słowo danych (bajt) jest przesyłane w trakcie 8 taktów sygnału na linii zegarowej SCL począwszy od bitu najbardziej do najmniej znaczącego. Dane na linii SDA muszą być stabilne, gdy linia zegarowa SCL ma poziom wysoki, a mogą się zmieniać gdy SCL jest na poziomie niskim. Każdy bajt danych odebranych przez Slave'a może być przez niego potwierdzony bitem potwierdzenia przesyłanym w trakcie dziewiątego impulsu zegarowego. W trakcie trwania tego impulsu układ *slave* wymusza poziom niski linii SDA, co jest traktowane jako potwierdzenie odebrania danych. Jeżeli brak potwierdzenia wystąpił po wysłaniu przez *mastera* adresu układu *slave*, może to oznaczać, że adres nie jest prawidłowy, układ *slave* jest nieobecny lub nie jest gotowy do jego odebrania. *Master* może wtedy wygenerować sekwencję *Stop* i zakończyć transfer danych. Sygnał braku potwierdzenia w czasie przesyłania danych (ale nie adresu) może też służyć do zasygnalizowania układowi *master*, że *slave* zakończył odbiór danych. Wtedy *master* generuje sekwencję *Stop* i transfer danych jest zakończony. Podobnie wygląda potwierdzenie danych odczytywanych z układu *slave*. Po odczytaniu ostatniego, 8 bitu *master* generuje dziewiąty impuls zegarowy i na linii danych wystawia bit potwierdzenia (SDA=0) lub braku potwierdzenia (SDA=1).

Slave ma możliwość czasowego zablokowania transmisji przez wyzerowanie linii CLK (wymuszenie poziomu niskiego). Taki stan nazywany jest nazywany stanem oczekiwania (*wait state*) i trwa przez czas potrzebny *slave* na przetworzenie ostatnio odebranej danej. *Master* musi sprawdzić czy po dziewiątym impulsie zegarowym sygnał na linii SCL przyjmuje poziom wysoki. Jeżeli nie, to musi zaczekać, aż tak się stanie.

Bezpośrednio po sekwencji *Start*, *master* wysyła bajt, w którym na 7 najstarszych bitach jest zapisany adres *slave*, a na najmłodszym bicie jest zapisana informacja o kierunku przesyłania danych (R/W). Kiedy bit R/W jest wyzerowany, to zaadresowany układ *slave* ma obowiązek odbioru jednego lub więcej bajtów danych przesyłanych po

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SMP	CKE	D/!A	P	S	R/!W	UA	BF

Bit 7 SMP: 1 zablokowany tryb prędkości slew rate dla trybu standard speed (100 kHz)
0 odblokowany tryb slew rate dla trybu high speed (400 kHz)
Bit 6 CKE tryby pracy magistrali SMB (dla I²C nie ma znaczenia)
Bit 5 D/!A 1 ostatnio odebrany/wysłany bajt był daną (tylko I²C Slave)
0 ostatnio odebrany/wysłany bit był adresem (tylko I²C Slave)
Bit 4 P 1 ostatnio wykryto bit STOP
0 nie wykryto ostatnio bitu STOP
Bit 3 S 1 ostatnio wykryto bit START
0 nie wykryto ostatnio bitu START
Bit 2 R/!W 1 trwa transmisja danych (I²C Master)
0 transmisja danych zakończona – magistrala w trybie idle state (I²C Master)
Bit 1 UA 1 konieczne wysłanie 2 bajtu adresu dla adresowania 10 bitowego (tryb Slave adresowanie 10-bitowe)
0 nie trzeba wysyłać drugiego bajtu adresu
Bit 0 BF odbieranie danych
1 odbiór zakończony SSPBUF jest pełny
0 odbiór danych trwa
Wysyłanie danych
1 wysyłanie danych trwa (tylko 8 bitów danych) – SSPBUF jest pełny
0 wysyłanie zakończone (tylko 8 bitów danych) – SSPBUF jest pusty

Rysunek 3. Rejestr SSPSTAT

B7	B6	B5	B4	B3	B2	B1	B0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0

Bit 7 WCOL I²C Master 1 zapis do SSPBUF kiedy magistrala nie jest w stanie gotowości do wysłania (kolizja na magistrali) Musi być zerowany programowo.
0 nie ma kolizji na magistrali
Bit 6 SSPOV 1 Wpisanie do SSPBUF nowo odebranej wartości, kiedy poprzednia nie została odczytana. Musi być zerowany programowo (dla wysyłania danych nie ma znaczenia)
0 nie ma przepełnienia SSPBUF
Bit 5 SSPEN 1 odblokowanie interfejsu i połączenie wyprowadzeń SDA i SCL do linii portu
0 zablokowanie interfejsu i zwolnienie linii portu
Bit 4 CKP w trybie master nie używane
w trybie Slave wymuszanie stanu na linii SCK
Bity 3:0 SSPM3, SSPM2, SSPM1, SSPM0
1111 I²C Slave adresowanie 10 bitowe – odblokowane przerwania od sekwencji START I STOP
1110 I²C Slave adresowanie 7 bitowe odblokowane przerwania od sekwencji START I STOP
1011 I²C Master Firmaware controlled
1000 I²C Master FCLK=Fosc/(4*SSPADD+1)
0111 I²C Slave adresowanie 10 bitowe
0110 I²C Slave adresowanie 7 bitowe

Rysunek 4. Rejestr SSPCON1 – tryb I²C

bajcie adresowym. Liczba bajtów odbieranych przez *slave* zależy od jego budowy wewnętrznej.

Na **rysunku 1** pokazano przesłanie adresu układu *slave* z bitem R/W=0, a po nim jednego bajtu danych. Po odebraniu adresu *slave* przesłał bit potwierdzenia ACK=0 sygnalizując, że adres jest poprawny i zgłaszając swoją aktywność. Po przesłaniu bajtu danych *slave* nie potwierdził jego odbioru (ACK=1). Master w takiej sytuacji może zakończyć transmisję danych, wygenerować sekwencję *Stop* i zwolnić magistralę.

Po zakończeniu transferu *master* może ponownie wygenerować sekwencję startu i zaadresować inny lub ten sam układ *slave*. Istnieje też możliwość wygenerowania ponownej sekwencji startu przez układ *master* i zaadresowania innego układu docelowego bez wcześniejszego generowania sekwencji stopu.

Odczytywanie danych z układu *slave* rozpoczyna się wysłaniem bajtu adresu z ustawionym bitem R/W. Po odebraniu potwierdzenia, *master* generuje 8 impulsów zegarowych na linii SCK, a *slave* transmituje w ich takt odczytywane dane. Na **rysunku 2** pokazano odczytanie 2 bajtów za pomocą I²C. Po odczytaniu pierwszego bajtu *master* wysyła bit potwierdzenia ACK sygnalizując, że jeszcze będzie odczytywał dane. Po odczycie 2 bajtu master go nie potwierdza i kończy transfer sekwencją *Stop*.

Interfejs I²C w PIC18F2320

Mikrokontroler PIC18F2320 ma wbudowany sprzętowy moduł MSSP tj. *Master Synchronous Serial Port*. MSSP może pracować w trybach: SPI lub I²C. Interfejs I²C wspiera sprzętowo tryby pracy: *master*, *multi-master* (klika układów Master dołączonych do jednej magistrali) i *slave*. Tu zajmiemy się przykładami programów dla najczęściej spotykanego i najbardziej przydatnego trybu *master*.

Programowanie konfiguracji i sterowanie pracą interfejsu odbywa się przez zapisywanie 3 rejestrów: SSPSTAT, SSPCON1 i SSPCON2. Znaczenie poszczególnych bitów rejestrów pokazano na **rysunkach 3, 4 i 5**. W trybie pracy jako *master* moduł MSSP generuje sygnał zegarowy na linii SCL. Źródłem tego sygnału jest blok programowanego generatora prędkości transmisji (*Baud Rate Generator Block*, **rysunek 6**). Na wejście 6-bitowego licznika BRG zliczającego w dół jest podawany sygnał zegara o częstotliwości Fosc/4. Wartość podziału częstotliwości jest wpisywana do siedmiu młodszych bitów rejestru SSPADD. Częstotliwość sygnału na linii SCL (CLOKOUT) jest równa

$$F_{SCL} = \frac{F_{osc}}{4 \cdot (SSPADD + 1)}$$

Jeżeli np. chcemy uzyskać częstotliwość 100 kHz przy F_{OSC}=8 MH, to SSPADD+1 musi być równa:

B7	B6	B5	B4	B3	B2	B1	B0
GCEN	ACKSTAT	ACKDT	ACKEN	RCEB	PEN	RSEN	SEN

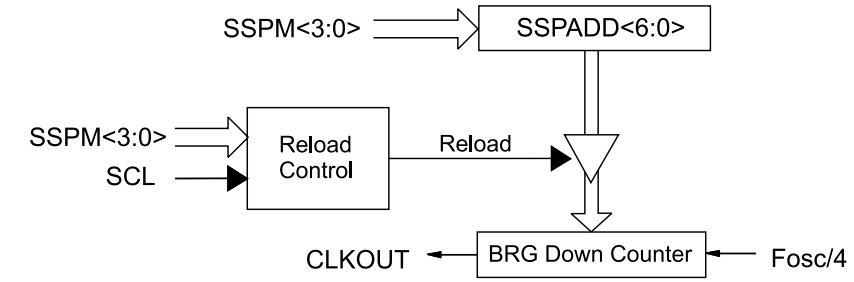
Bit 7 GCEN — odebranie adresu Genearal Call (0000h) – tylko tryb I²C Slave
Bit6 ACKSTAT — 1 nie odebrano potwierdzenia ze Slave; 0 odebrano potwierdzenie ze Slave
Bit 5 ACKDT — tryb Master odbiór; 1 wysyłanie braku potwierdzenia odebranego bajtu; 0 wysyłanie potwierdzenia odebranego bajtu
Bit 4 ACKEN — tryb Master odbiór; 1 Inicjacja sekwencji wysyłania bitu potwierdzenia o wartości z bitu ACKDT; 0 sekwencja potwierdzenia zablokowana
Bit 3 RCEN — 1 odblokowanie odbioru danych I²C Master; 0 odbiór danych zablokowany
Bit 2 PEN — tylko tryb I²C Master; 1 inicjacja sekwencji STOP (zerowana sprzętowo po zakończeniu sekwencji STOP); 0 sekwencja STOP zakończona
Bit 1 RSEN — tylko tryb Master; 1 inicjacja sekwencji powtórzonego startu (RESTART) (zerowana sprzętowo po zakończeniu sekwencji RESTART); 0 sekwencja RESTART zakończona
Bit 0 SEN — 1 inicjacja sekwencji START (zerowana sprzętowo po zakończeniu sekwencji START); 0 sekwencja START zakończona.

Rysunek 5. Rejestr SSPCON2

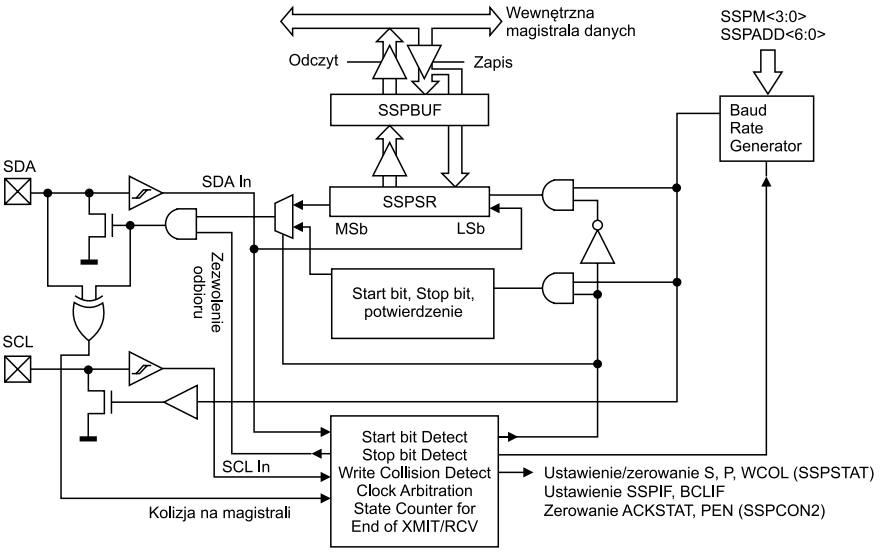
$$\frac{8\ MHz}{4} = 2\ MHz$$
$$\frac{2\ MHz}{100\ kHz} = 20$$

Do SSPADD trzeba zatem wpisać wartość 19 dziesiętnie.

Na **rysunku 7** pokazano schemat blokowy modułu MSSP pracującego w trybie *master*. Każda z sekwencji sterujących pracą magistrali jest wykonywana sprzętowo. Trzeba pamiętać, że nie ma tu mechanizmu kolejkowania. Jest to istotna informacja, bo oznacza to, że nie można zainicjować następnej sekwencji dopóki niż zakończy się poprzednia i magistrala nie wejdzie w stan *idle state*. Nie można na przykład wpisać adresu *slave* do rejestru SSPBUF, jeżeli nie zakończyła się sekwencja *Start*. Każde takie działanie powoduje anulowanie sekwencji i wygenerowanie informacji o kolizji na magistrali. Z zakończeniem każdej z sekwencji jest związany mechanizm zgłaszania przerwania (można z niego korzystać lub nie). Takie rozwiązanie umożliwia napisanie obsługi transmisji „w tle" w bardziej zaawansowanych aplikacjach. W opisywanych przykładach nie bę-



Rysunek 6. Generator prędkości transmisji



Rysunek 7. Moduł MSSP w trybie I2C Master

dziemy wykorzystywać mechanizmu przerwań i komentować zachowania się bitu flagi zgłoszenia przerwania MSSPIF.

Po włączeniu zasilania moduł MSSP jest domyślnie wyłączony. Przed użyciem trzeba go włączyć ustawiając bit SSPEN i skonfigurować do pracy w trybie *I2C Master*. Konieczne jest też zaprogramowanie częstotliwości sygnału wyjściowego generatora prędkości transmisji.

Na **listingu 1** pokazano procedurę *OpenI2C*. Wpisanie do rejestru SSPCON wartości 0x08 włącza tryb pracy *master*. Do SSPADD jest wpisywane 19, co dla Fclk=8 MHz daje prędkość transmisji 100 kbit/s. Po ustawieniu linii portów SDA i SCL jako wejściowe i odblokowaniu odbioru (ustawienie bitu RCEN), można odblokować interfejs.

Sekwencja *Start* jest inicjowana ustawieniem bitu SEN rejestru SSPCON2. Przed tym magistrala musi być w stanie *idle state*. Każdy stan sekwencji trwa czas równy okresowi zegara generatora prędkości transmisji TBRG. Po sekwencji *Start* moduł zeruje bit SEN i ustawia bit zgłoszenia przerwania SSPIF. Jeżeli przerwania nie są odblokowane, to przerwanie nie będzie obsłużone. Żeby zapobiec konfliktowi na magistrali zapisanie do SSPBUF adresu *slave* jest możliwe dopiero po wyzerowaniu SEN. Procedura sprawdza czy ustawienie SEN nie wywołuje konfliktu na magistrali (jeżeli SEN było ustawione, a magistrala nie była w stanie *idle state*). Jeżeli zostanie wykryta kolizja, to sekwencja się nie wykona, a funkcja zwróci wartość błędu *–1*. Prawidłowo wykonana sekwencja *Start* powoduje zwrócenie przez funkcję *0*.

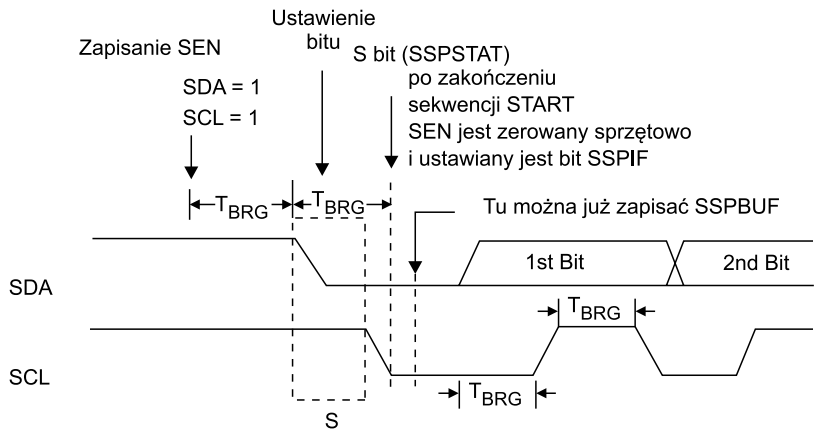
Listing 1. Procedura inicjalizacji MSSP w trybie I2C Master
```
void OpenI2C(void){
  SSPADD=19;              //SCL=100kHz
  SSPCON=8;               //MSSP I2C Master
  TRISC3=1;               //SDA i SCL wejściowe
  TRISC4=1;
  RCEN=1;                 //odblokuj odbiór
  SSPEN=1;                //odblokuj MSSP
}
```

Listing 2. Wysłanie sekwencji START
```
char i2c_start(void){
  SEN=1;                  //inicjowanie sekwencji START
  if(BCLIF==1)            //błąd kolizji magistrali
  return(-1);
  while(SEN);             //czekaj na koniec sekwencji START
  return(0);
}
```

Listing 3. Wysłanie danej na magistralę
```
char i2c_write(unsigned char data){
  SSPBUF=data;            //zapisanie danej do SSPBUF - start transmisji
  if (WCOL ){             // sprawdzenie czy nie ma kolizji na magistrali
    WCOL=0;               //programowe zerowanie WCOL
    return ( -1 );
  }
  else
  {
    while(RW);            //czkaj na wysłanie danej i odebranie bitu ACK
    return(ACKSTAT);
  }
}
```

Listing 4. Odbiór bajtu
```
unsigned char i2c_read(unsigned char ack ){
  RCEN = 1;               //start odbierania jednego bajtu z magistrali
  while (!BF );           //czekaj na koniec odebrania 8 bitów
  if(ack==0) ACKDT = 0;   //ustawienie bitu potwierdzenia
  else ACKDT=1;
  ACKEN = 1;              //start sekwencji wysyłania bitu potwierdzenia
  while(ACKEN);           //czekaj na zakończenie sekwencji
  return ( SSPBUF );      //powrót z odebranym bajtem
}
```

Listing 5 wysłanie sekwencji STOP
```
void i2c_stop(void){
  PEN=1;                  //inicjowanie sekwencji STOP
  while(PEN);             //czekaj na koniec sekwencji STOP
}
```



Rysunek 8. Wykonanie sekwencji Start

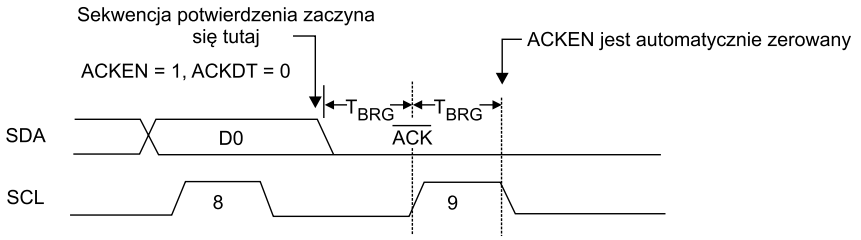
Wysłanie bajtu rozpoczyna się po jego wpisaniu do rejestru SSPBUF. Magistrala musi być w stanie *idle state* (nie jest wykonywana inna sekwencja), bo w przeciwnym przypadku wystąpi kolizja. Zapisanie SSPBUF powoduje ustawienie bitu BF. Bit BF jest zerowany po ósmym impulsie zegarowym na linii SCL, czyli po wysłaniu całego bajtu. Można to wykorzystać do testowania zakończenia transmisji. Jednak master wysyła na magistralę jeszcze dziewiąty impuls zegara, aby odczytać bit potwierdzenia. Trzeba też pamiętać o możliwości wystąpienia stanu oczekiwania *wait state* wymuszanego przez układ *slave*. Aby mieć pewność, że po wysłaniu bajtu magistrala przeszła w stan *idle state*, należy testować bit RW rejestru SSPSTAT. Bit ten jest ustawiany w momencie zapisania SSPBUF i zerowany po zwolnieniu magistrali.

Po zapisaniu rejestru SSPBUF są ustawiane bity BF i R/W. Warto też sprawdzić czy nie jest ustawiony bit kolizji WCOL (**listing 3**). Jego ustawienie spowoduje, ze dane nie zostaną przesłane. Procedura *i2c_write* testuje bit RW. Wyzerowanie RW oznacza zakończenie przesyłania danych łącznie z odebraniem bitu potwierdzenia (zapisanym w bicie ACKSTAT). Jeżeli procedura zwróci kod błędu –1, to oznacza, że dane nie zostały wysłane. W przeciwnym wypadku jest zwracana wartość bitu potwierdzenia odebranego z układu *slave*.

Zainicjowanie odbioru danych jest możliwe, gdy magistrala jest w stanie *idle state*. Ustawienie bitu RCEN powoduje, że bit BF jest zerowany. *Master* wysyła na magistralę 8 impulsów zegarowych, w takt których *slave* wysyła na magistralę dane. Zakończenie odbierania danych jest sygnalizowane wpisaniem 1 do rejestru BF. Mikrokontroler powinien wtedy odczytać rejestr SSPBUF. Jednak to nie koniec odbioru danych, ponieważ *master* powinien wysłać do układu *slave* bit potwierdzenia. Trzeba zainicjować sekwencję potwierdzenia przez ustawienie bitu ACKEN i wyzerowanie (potwierdzenie) lub ustawienie (brak potwierdzenia) ACKDT. Sekwencja potwierdzenia kończy się po sprzętowym wyzerowaniu bitu ACKEN (**rysunek 8**).

Na **listingu 4** zamieszczono kompletną procedurę odbioru bajtu i wysłania sekwen-

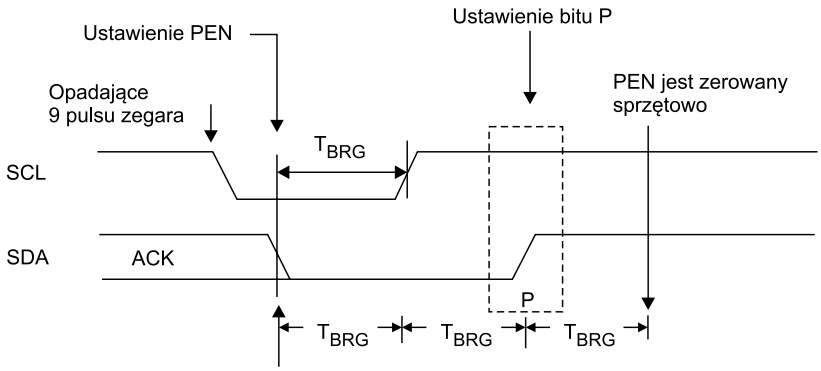
cji potwierdzenia. Argumentem procedury jest wartość bitu potwierdzenia, a zwracany jest odczytany bajt. Sekwencja *Stop* rozpoczyna się od ustawienia bitu PEN. Sekwencja wykona się prawidłowo, na opadającym zboczu dziewiątego impulsu zegara przy przesyłaniu potwierdzenia danej zapisywanej lub odczytywanej za pomocą interfejsu I²C (**listing 5**). W protokole wymiany informacji jest jeszcze sekwencja powtórzonego startu: *REPEATED START* (*RESTART*) pokazana na **rysunku 9**. Jest ona inicjowana przez ustawienie bitu RSEN. Wówczas linia SDA przyjmuje poziom wysoki, a linia SCL pozostaje bez zmian. Po czasie równym okresowi przebiegu sygnały wyjściowego generatora *baud rate generator* TBRG, linia SCL przyjmuje poziom wysoki, a następnie jest wykonywana standardowa sekwencja *Start*. Po jej wykonaniu bit RSEN jest zerowany sygnalizując zakończenie *RESTART* (**listing 6**).



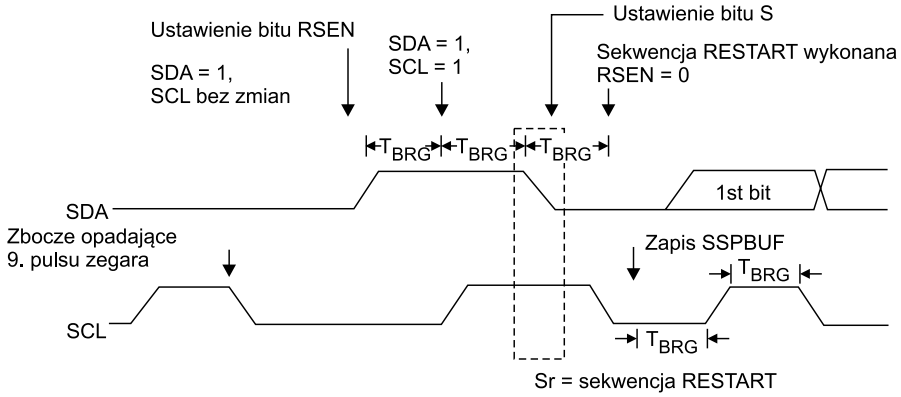
Rysunek 9. Sekwencja potwierdzenia (ACKDT=1)

Listing 6. Sekwencja REPEAT START

```
char i2c_rstart(void){
  RSEN=1;                 //incjowanie sekwencji RE START
  if(BCLIF==1)            //błąd kolizji magistrali
  {
    BCLIF=0;
    return(-1);
  }
  while(RSEN);            //czekaj na koniec sekwencji RESTART
  return(0);
}
```



Rysunek 10. Sekwencja Stop



Rysunek 11. Sekwencja Repeated Start

Pomiar napięcia za pomocą przetwornika MCP3021

Po garści wiedzy teoretycznej, w tym przykładzie programu użytkowego użyjemy modułu MSSP pracującego w trybie *master* do komunikacji ze scalonym przetwornikiem analogowo-cyfrowym MCP3021 zamontowanym na płytce AVT5275.

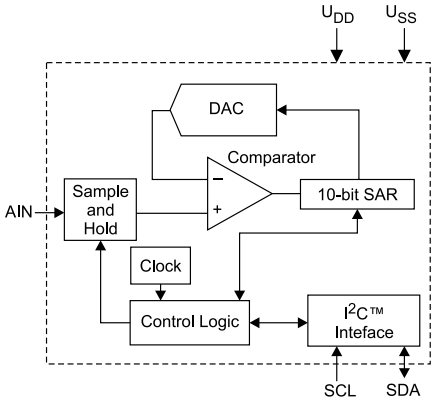
MCP3021 jest scalonym przetwornikiem A/D o rozdzielczości 10-bitów. Może być zasilany pojedynczym napięciem z zakresu 2,7…5,5 V i pobiera maksymalnie 250 μA. Napięcie mierzone jest podawane na wejście asymetryczne. Interfejs komunikacyjny I²C tego przetwornika może pracować z prędkością standardową 100 kHz i w trybie *Fast* z maksymalną prędkością 400 kHz. Przetwornik ma architekturę SAR. Jego schemat blokowy pokazano na **rysunku 10**. Wewnętrzny kondensator jest połączony z wejściem sygnału analogowego AIN i gromadzi ładunek przez czas akwizycji. Po rozpoczęciu konwersji analogowy klucz odłącza kondensator od AIN a jego ładunek jest konwertowany na liczbę 10-bitową. Przetwornik A/D pracuje synchronicznie i jest taktowany przez wbudowany generator, którego częstotliwość determinuje czasy akwizycji i przetwarzania. Typowo, czas akwizycji – oznaczony *tacq* – wynosi 1,12 μs, natomiast czas konwersji *tconv* ma wartość 8,96 μs. Oba czasy nie są niezależne od częstotliwości sygnału zegarowego na linii SCL.

Sekwencja przesyłania danych z przetwornika (*slave*) do mikrokontrolera (*master*) rozpoczyna się od wysłania bajtu adresu. Oczywiście, adres wysyła mikrokontroler, a odbiera przetwornik. 7-bitowy *adres slave* musi być uzupełniony najmłodszym bitem R/W. W starszym bajcie 4 najstarsze bity są zerowane. Pomimo, że w bajcie adresu są przewidziane 3 bity adresowe (A2, A1, A0; **rysunek 11**), to standardowo można kupić układy z A2=1, A1=0 i A0=1 (**rysunek 12**). Układy z innymi adresami są dostępne na zamówienie u producenta. W wypadku odczytywania danych z przetwornika bit R/W musi być ustawiony (odczyt danych). Możliwe jest wyzerowanie R/W, ale wówczas z przetwornika nie można odczytać danych, jednak po wysłaniu jego adresu układ odpowiada zerując bit ACK. Można to wykorzystać np. do sygnalizacji obecności przetwornika na magistrali.

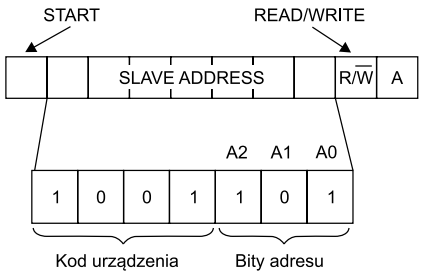
Po wysłaniu 7 bitów adresu i bitu R/W=1 trzeba odczekać czas równy sumie czasu akwizycji i czasu konwersji tconv+tacq= 10,08 μs. Inicjowanie przetwarzania jest wykonywane przy pierwszym opadającym zboczu przesyłania bitu R/W.

Na rysunku **13** jest pokazano sekwencję odczytu danych z przetwornika A/D. W tej procedurze, zamieszczonej na **listingu 7**, po wysłaniu adresu jest wykonywane opóźnienie 1 ms, aby przetwornik mógł odliczyć czas akwizycji i konwersji. Oczywiście, czas trwania opóźnienia wynoszący 1 ms jest większy od wymaganego (ok. 10 μs), jednak o ile spowalnia to akwizycję danych z przetwornika, to w niczym nie przeszkadza. Brak opóźnienia powoduje, że po wysłaniu następnego bajtu adresu magistrala zgłasza błąd kolizji – ustawiany jest bit WCOL i nie dochodzi do odczytu.

Testowanie działania przetwornika trzeba zacząć od połączenia linii *I2C_SCL* z linią *RC3* i *I2C_SDA* z linią *RC4* złącze J11). Wejście *ADC_IN* łączymy z wyprowadzeniem



Rysunek 12. Schemat blokowy MCP3021



Rysunek 13. Adresowanie MCP3021

VREG (suwak potencjometru PR2). *ADC_IN* i *VREG* sa dostępne na złączu J22.

Pętla nieskończona pokazana na **listingu 8** odczytuje wartość z przetwornika i wyświetla ją w postaci heksadecymalnej na wyświetlaczu LCD modułu AVT5275.

Listing 7. Odczytanie wartości pomiaru

```
unsigned int  read_adc(void){
unsigned int  res;
  i2c_start();
  i2c_write(0x9b);                //adres slave
  delay_ms(1);                    //opóźnienie
  res=i2c_read(0);                //starszy bajt wyniku
  res<<=8;
  res=res| i2c_read(1);           //młodszy bajt wyniku
  i2c_stop();
  return(res);
}
```

Listing 8. Pętla odczytywania i wyświetlania pomiarów

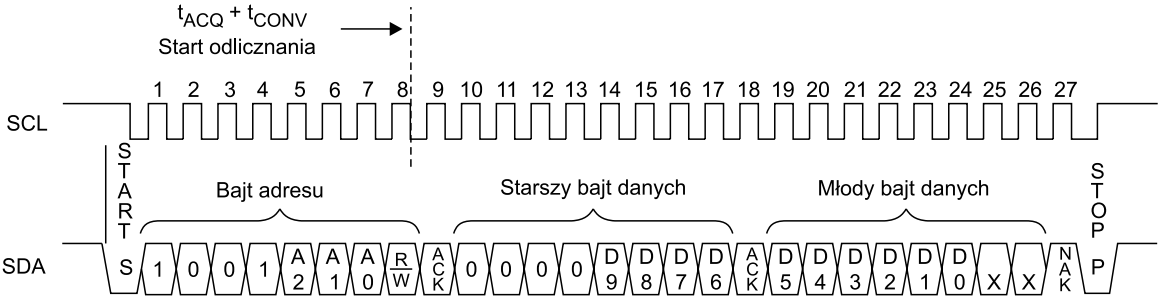
```
unsigned int adc;
while(1){
  adc=read_adc();
  DispHex(adc>>8,1,1);            //wyświetlanie 8 starszych bitów
  DispHex(adc,3,1);               //wyświetlanie 8 młodszych bitów
}
```

Listing 9. Pętla woltomierza

```
  while(1){
  delay_ms(400);
  adc=read_adc();                 //odczytanie przetwornika
  volt=(unsigned long)adc*488;    //konwersja
  volt=(volt/1000);
  DispDec(volt,1,2);              //wyświetlanie
}
```

Aby program spełniał funkcję woltomierza trzeba 10-bitową wartość odczytaną z rejestrów przetwornika poddać konwersji, podobnie jak to było w przykładzie z przetwornikiem A/C wbudowanym w strukturę mikrokontrolera. Napięcie referencyjne MCP3021 jest równe napięciu zasilania. Przy jego wartości wynoszącej 5 V i rozdzielczości 10-bitowej zmiana na najmłodszym bicie odpowiada zmianie napięcia równej 5 V/1024=0,0048828 V. Wartość z rejestru przetwornika mnożymy przez 488, a potem dzielimy przez 1000. Do wyświetlania napięcia mierzonego wykorzystamy dokładnie taka samą procedurę, jak w wypadku wyświetlania napięcia mierzonego przez wbudowany przetwornik. Fragment programu woltomierza zamieszczono na **listingu 9**.

Tomasz Jabłoński, EP



Rysunek 14. Sekwencja odczytywania danych z przetwornika za pomocą I²C