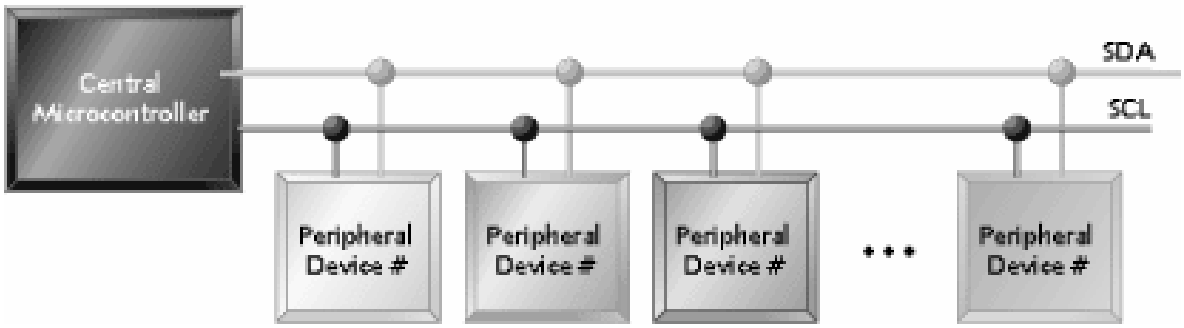
OBSŁUGA INTERFEJSU TWI (I²C) NA MIKROKONTROLERZE ATMEGA16

wydanie drugie

Opracowanie zawiera treści różnych publikacji takich jak:
książki, datasheety, strony internetowe



Cezary Klimasz
Kraków 2010
cezary.klimasz@gmail.com

Spis treści

1. **Przedstawienie standardu I²C ..... str. 3**
2. **Wykorzystanie TWI ..... str. 4**
   - **opis rejestrów TWI ..... str. 10**
   - **korzystanie z two-wire serial interface ..... str. 14**
3. **Pamięć EEPROM 24C16 ..... str. 27**
4. **Obsługa pamięci EEPROM w języku C ..... str. 30**
   - **program pierwszy ..... str. 31**
   - **program drugi ..... str. 34**
5. **Podsumowanie ..... str. 40**
6. **Bibliografia ..... str. 40**

1. Przedstawienie standardu

WPROWADZENIE

I²C – szeregowa, dwukierunkowa magistrala służąca do przesyłania danych w urządzeniach elektronicznych. Została opracowana przez firmę *Philips* na początku lat 80. Znana również pod akronimem *IIC*, którego angielskie rozwinięcie *Inter-Intergrated Circuit* oznacza "pośredniczący pomiędzy układami scalonymi". **Standard I²C** określa dwie najniższe warstwy modelu: warstwę fizyczną i warstwę łącza danych.

Jako, że firma *Philips* jest właścicielem prawnym standardu *I²C* inne firmy używają różnych nazw w odniesieniu do tego standardu. W wypadku mikrokontrolerów **AVR**, **Atmel** używa nazwy **TWI** (*two-wire interface*).

OPIS STANDARDU

Standard został opracowany na początku lat 80. (określany obecnie jako tryb standardowy pracy) i cechowały go:

- prędkość transmisji **100 kbps**,
- 7-bitowa przestrzeń adresowa.

W kolejnych latach standard ten rozszerzano poprzez zwiększanie możliwych prędkości transmisji, zmianę zakresu tolerancji napięcia (np. *High Speed Mode* – **3,4Mbps**, napięcie tolerancji w stanie wysokim: **+2,3 ÷ +5,5 V**).

I²C do transmisji wykorzystuje dwie dwukierunkowe linie: **SDA** (**Serial Data Line**) i **SCL** (**Serial Clock Line**). Obydwie linie są na stałe podciągnięte do źródła zasilania poprzez rezystory podciągające[1]. *I²C* używa logiki dodatniej. Wszystkie nadajniki są typu otwarty kolektor lub otwarty dren, a więc na liniach występuje tzw. iloczyn na drucie ("1" jest recesywna, a "0" dominujące). Pozwala to na wykrywanie kolizji. Każde urządzenie nadając "1" jednocześnie sprawdza, czy na magistrali rzeczywiście pojawił się stan wysoki. Jeżeli tak nie jest, oznacza to, iż inne urządzenie nadaje w tym samym czasie i urządzenie zaprzestaje nadawania.

[1] tzw. rezystory pullup

Podstawowa wersja *I²C* zakłada istnienie tylko jednego urządzenia, które może inicjować transmisję (**master**), ale dzięki istnieniu mechanizmu detekcji kolizji, możliwa jest praca w trybie **multi-master**. Ponieważ dane nadawane są w kolejności od najstarszego bitu do najmłodszego, w przypadku jednoczesnego nadawania, urządzenie nadające adres o wyższym numerze wycofa się pierwsze, co wynika z binarnego sposobu zapisywania liczb. Występuje tu zatem arbitraż ze stałym przydziałem priorytetów, określonym przez adres urządzenia typu *slave*. Urządzenia o niższych adresach mają wyższy priorytet od urządzeń o adresach wyższych.

Zmiana na linii danych podczas transmisji może następować jedynie, gdy linia zegara znajduje się w stanie niskim. Nie dotyczy to specjalnych sytuacji: *bitu startu* i *bitu stopu*. **Bit startu** ma miejsce, gdy linia danych zmienia swój stan z "*1*" na "*0*", podczas wysokiego stanu linii zegara, co ma miejsce w momencie rozpoczynania każdej transmisji danych. Po zakończeniu transmisji generowany jest **bit stopu**, czyli przejście linii danych w stan wysoki przy wysokim stanie linii zegara. Standard zakłada magistralowe połączenie urządzeń. Długość linii ograniczona jest jedynie jej maksymalną pojemnością, która wynosi *400 pF*.

2. Wykorzystanie TWI

Datasheet ATmega16 AVR®

INTERFEJS TWI[2]

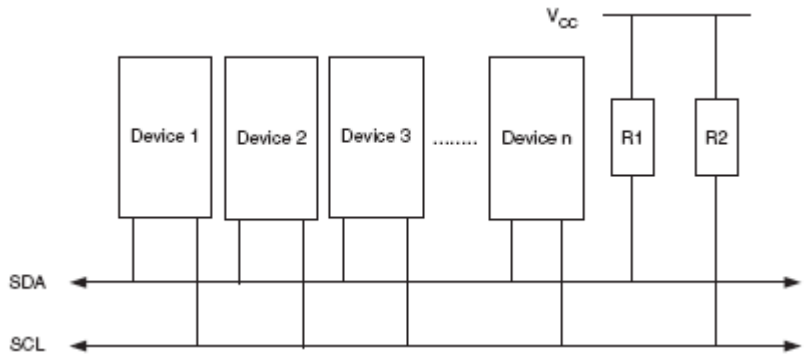
Głównymi cechami interfejsu **TWI** w mikrokontrolerach *AVR* jest:

- prostota i elastyczność interfejsu komunikacyjnego – potrzebne tylko dwie linie,
- możliwe operacje *master*/*slave*,
- dualność układów (zarówno nadajnik jak i odbiornik),
- *7-bitowy* system adresowania pozwalający na zapisanie *128* różnych adresów,
- wbudowany arbitraż,
- prędkość transmisji z częstotliwością *400kHz*,
- redukcja szumów.

[2] Interfejs TWI jest odpowiednikiem interfejsu I²C (lub IIC). Odmienne nazwy wynikają z praw autorskich firmy Philips do nazwy I²C. Atmel używa swojego nazewnictwa – Two-wire serial interface (TWI).

DEFINICJA MAGISTRALI

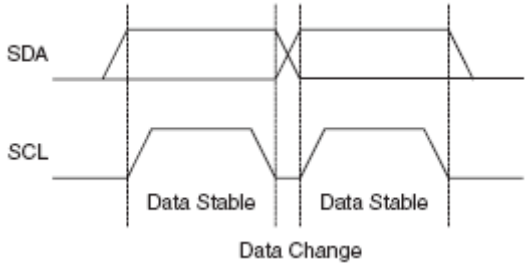
Two-wire Serial Interface (*TWI*) jest idealnym sposobem transmisji pomiędzy typowymi aplikacjami mikrokontrolerowymi[3]. **Protokół TWI** pozwala na podłączenie nawet *128* różnych układów, używając do tego jedynie dwóch linii danych. Jednej dla sygnału zegarowego **SCL**, drugiej linii danych **SDA**. Jedyne wymagania ze strony sprzętowej to podciągnięcie tych linii przez rezystory do zasilania. Wszystkie układy podłączone do magistrali mają indywidualne adresy, mechanizm i system obsługi magistrali umieszczony jest w protokole *TWI*. Poniżej przedstawiono schemat logiczny magistrali *TWI*.



Rysunek. *Połączenia pomiędzy urządzeniami w komunikacji I²C*

TRANSMISJA DANYCH
WYGLĄD RAMKI

Każdy bit danych wysyłany na magistralę *TWI* jest zsynchronizowany z impulsem na linii zegarowej. Poziom na linii danych musi być stabilny podczas kiedy na linii danych jest stan wysoki. Jedynym wyjątek stanowią warunki generowania startu oraz zatrzymywania transmisji. Poniżej przedstawiono rozwiązanie prawidłowe.



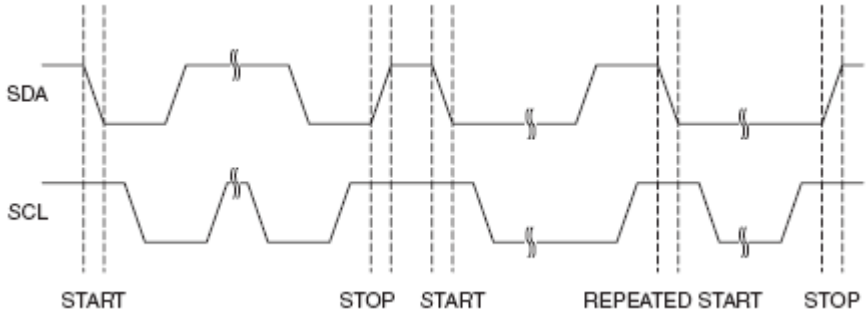
Rysunek. *Synchronizacja linii SDA z linią zegarową SCL*

.

[3] określenie „idealnym" zostało zaczerpnięte z dokumentacji ATmela

WARUNKI STARU I STOPU TRANSMISJI

Układ *Master* inicjalizuje i kończy przesył danych. Transmisja jest inicjowana w momencie kiedy *Master* wyda **żądanie startu**. Transmisja jest kończona jeśli *Master* wyśle odpowiedni rozkaz na magistralę. Pomiędzy żądaniami startu i stopu, magistrala jest uważana za zajętą i żaden inny *Master* nie powinien próbować korzystać z magistrali. Specjalny przypadek występuje gdy pojawiło się nowe żądanie startu w trakcie trwania poprzednich żądań. Taki przypadek określany jest jako **powtórzone żądanie startu** i jest używane w sytuacji kiedy *Master* chciałby rozpocząć nową transmisję nie czekając na zwolnienie magistrali. Po powtórzonym starcie, magistrala uważana jest za zajętą dopóki nie nastąpi stop. **START** i **STOP** są sygnalizowane przez zmianę poziomów na linii danych *SDA* kiedy linia *SCL* jest w stanie wysokim. Poniższy diagram przedstawia omówione generowanie startu i stopu.



Rysunek. *Transmisja TWI – Start, Stop, Repeated Start*

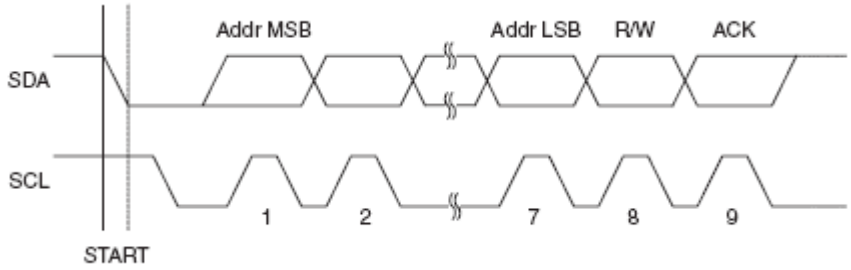
FORMAT ADRESU

Wszystkie adresy przesyłane magistralą *TWI* są **9 bitowe**. Zawierają *7* bitowy adres, następnie jeden **kontrolny bit READ/WRITE** i jeden **bit potwierdzający** (tzw. *acknowledge bit*). Jeśli ustawiony jest bit *READ/WRITE (***RW=1***)* operacja **odczytywania** jest wykonywana, w przeciwnym przypadku (**RW=0**) operacja **zapisywania** powinna być wykonana. Kiedy układ *Slave* rozpozna, że przesyłany adres jest jego adresem, powinien potwierdzić odbiór poprzez ustawienie linii *SDA* w stan niski w dziewiątym cyklu zegara (**ACK**[4]). Jeśli adres *Slave'a* jest zajęty lub z jakiś powodów nie może odpowiedzieć na wezwanie *Mastera*, linia *SDA* powinna być pozostawiona w stanie wysokim w cyklu zegara *ACK*. Master może wtedy wysłać żądanie *STOP* transmisji, lub powtórzyć *START*, aby

[4] ang. ACKnowledge bit

rozpocząć nową transmisję. Adres *Slave'a* oraz bit *READ/WRITE* są nazywane **SLA+R** lub **SLA+W** w odpowiednich przypadkach (w zależności czy zapis, czy odczyt).

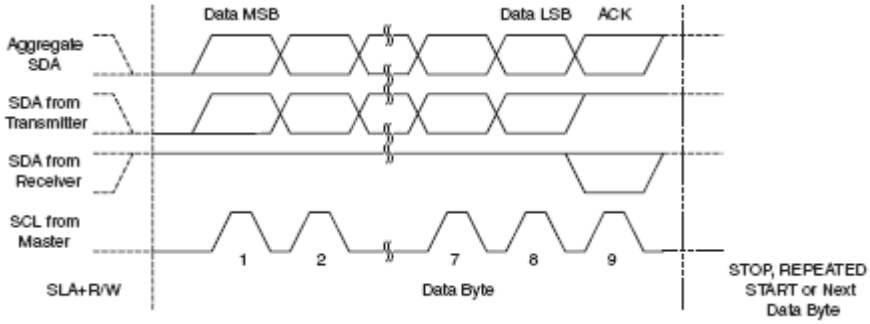
Najstarszy bit (**MSB**) adresu jest wysyłany jako pierwszy. Adres *Slave'a* może zostać dowolnie wybrany, ale trzeba pamiętać, że adres *0000 000* jest zarezerwowany dla wywołania ogólnego. Kiedy wygenerowane jest wywołanie ogólne, wszystkie układy *Slave* powinny odpowiedzieć poprzez ustawienie linii *SDA* w stan niski w cyklu *ACK*. Wywołanie ogólne jest używane wtedy gdy *Master* chce wysłać tą samą wiadomość do wszystkich układów na magistrali.



Rysunek. *Adresowanie układu na magistrali TWI*

FORMAT DANYCH

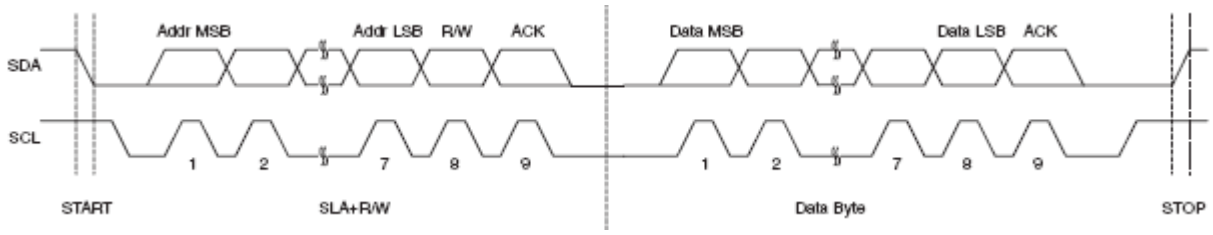
Wszystkie przesyłane dane przez magistralę są dziewięciobitowe. Zawierają bajty danych i bit potwierdzenia. Podczas transmisji danych układ *Master* generuje sygnał zegarowy oraz żądania *START* i *STOP*, podczas kiedy odbiornik jest odpowiedzialny za potwierdzanie odbioru. Potwierdzenie (*ACK*) jest sygnalizowane przez odbiornik poprzez ustawianie linii danych *SDA* do poziomu niskiego, podczas dziewiątego cyklu zegarowego. Jeśli odbiornik zostawi *SDA* w stanie wysokim, generowane jest **NACK**. Kiedy odbiornik otrzyma ostatni bajt lub z jakiegoś powodu nie może odebrać więcej bajtów, powinien poinformować o tym wysyłając **NACK** po ostatnim otrzymanym bajcie.



Rysunek. *Transmisja danych z nadajnika do odbiornika*

WYSYŁANIE ADRESU I DANYCH

Dane transmitowane powinny zawierać: sekwencję **START**, **SLA+R/W**, **jedną lub więcej paczek danych** oraz **żądanie STOP**. Niedozwolone jest przesyłanie pustych wiadomości, zawierających *START* i *STOP*. Poniżej przedstawiono typową transmisję.



Rysunek. *Typowa transmisja danych na magistrali*

MAGISTRALA MULTI-MASTER

Protokół *TWI* zakłada możliwość podłączenia kilku układów Master do jednej magistrali. Specjalne algorytmy zostały zaimplementowane celem stworzenia arbitrażu komunikacji. Szerszy opis znajduje się w *Datasheecie* **ATmega16**.

BIT RATE GENERATOR UNIT

Moduł ten kontroluje okres trwania sygnału *SCL* kiedy obsługiwany jest tryb *Master*. Okres trwania *SCL* jest kontrolowany przez rejestr **TWI Bit Rate Register** (**TWBR**) oraz przez preskaler ustawiony w rejestrze **TWI Status Register** (**TWSR**). Obsługa *Slave* nie jest zależna od generatora oraz od preskalera, ale częstotliwość taktowania procesora *slave'a* (*CPU clock*) musi być co najmniej 16 razy wyższa niż częstotliwość niż częstotliwość zegarowa *SCL*. Częstotliwość *SCL* oblicza się ze wzoru:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

TWBR jest wartością *TWI Bit Rate Register*

TWPS to wartość bitów preskalera w rejestrze *TWI Status Register*

BUS INTERFACE UNIT

Jednostka ta zawiera dane oraz adres *Shift Register* (**TWDR**), kontrole *START*/*STOP* oraz wspomaganie sprzętowe arbitrażu. Rejestr *TWDR* zawiera adres i dane do transmisji lub adres i dane odebrane. *TWDR* może posiadać może również bit *N(ACK)* do wysłania lub odebrania. **Bit N(ACK)** nie jest bezpośrednio dostępny z poziomu oprogramowania. Jednak, kiedy dane są odbierane, może być ustawiony lub wyczyszczony przez manipulowanie rejestrem **TWI Control Register** (**TWCR**). Kiedy ustawiony jest tryb nadawania, wartość *N(ACK)* może być reprezentowana przez wartość w rejestrze **TWSR**.

Kontrolka *START/STOP* jest odpowiedzialna za generowanie i wykrywanie odpowiednich akcji: *START, REPRATED START i STOP*.

ADDRESS MATCH UNIT

Jednostka ta sprawdza, czy długość otrzymanego adresu w rejestrze **TWI Address Register** jest *7-bitowa*.

CONTROL UNIT

Jednostka monitoruje magistralę *TWI* oraz generuje odpowiedzi zgodne z ustawieniami w rejestrze **TWI Control Register** (**TWCR**). Kiedy jakieś wydarzenie wymaga uwagi ze strony zdarzeń na magistrali *TWI*, flaga **TWI Interrupt Flag** (**TWINT**) jest ustawiona. W następnym cyklu zegarowym rejestr *TWI Status Register* (*TWSR*) jest uaktualniany przez kod statusu identyfikujący odpowiednie zdarzenie. Rejestr *TWSR* zawiera tylko istotne wiadomości o statusie. Tak długo jak flaga *TWINT* jest ustawiona, linia *SCL* jest ustawiana na poziom niski. Pozwala to oprogramowaniu aplikacji na zakończenie zadań wykonywanych przez wystąpieniem przerwania *TWI*.

Flaga *TWINT* jest ustawiona w następujących sytuacjach:

- po wysłaniu przez *TWI* żądań: *START/REPEATED START*,
- po wysłaniu bitów *SLA+R/W*,
- po wysłaniu danych adresowych,
- po tym jak *TWI* straci arbitraż,

- jeśli *TWI* wykryje: adresowanie do układu lub wywołanie ogólne,
- po tym jak *TWI* odbierze bajty z danymi,
- po odebraniu *STOP* lub *REPEATED START*,
- w wypadku niedozwolonych akcji *START* lub *STOP*.

OPIS REJESTRÓW TWI

TWI Bit Rate Register – TWBR

Bit	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bity 7..0 – TWI Bit Rate Register

Rejestr odpowiedzialny za wybór współczynnika podziału dla generatora. Generator ten odpowiada za częstotliwość która jest dzielona przez sygnał zegarowy *SCL* w trybie pracy *Master*.

TWI Control Register – TWCR

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Rejestr ten jest używany do kontrolowania operacji związanych z interfejsem. Ustawienia bitów tego rejestru odpowiadają za: inicjację *TWI*, inicjację dostępu układu *Master* do magistrali przez zastosowanie akcji *START*, do generowania potwierdzeń odbiornika, do generowania akcji *STOP* oraz do kontroli zatrzymania magistrali w chwili kiedy dane są wysyłane na magistralę. Wskazują także kolizję zapisu jeśli dane zapisywane do rejestru *TWDR* nie mogą zostać zapisane gdyż rejestr w tym momencie jest niedostępny.

Bit 7 – TWINT: TWI Interrupt Flag

Bit ten jest ustawiany przez *hardware* w chwili, gdy *TWI* zakończy aktualną pracę i będzie oczekiwać od aplikacji odpowiedzi. Jeśli pierwszy bit z rejestru *SREG* oraz bit *TWIE* z rejestru *TWCR* są ustawione, wtedy mikrokontroler skacze do wywołania **TWI Interrupt Vector**. W chwili gdy flaga *TWINT* jest ustawiona, krótkie okresy zegarowe *SCL* są rozciągane. Flaga ta musi być wyczyszczona przez oprogramowanie poprzez zapisanie logicznej jedynki do niej. Flaga jest automatycznie czyszczona sprzętowo kiedy zostanie wywołane przerwanie. Również może być wyczyszczona w momencie startu obsługi *TWI*, dlatego wszystkie dostępne rejestry: *TWI Address Register (TWAR)*, *TWI Status Register (TWSR)* oraz *TWI Data Register (TWDR)* muszą być zdefiniowane przez wyczyszczeniem flagi.

Bit 6 – TWEA: TWI Enable Acknowledge Bit

Bit ten kontroluje generowanie impulsu potwierdzenia. Jeśli bit ten zapisany jest jako jeden, wtedy *ACK* generowane jest przez *TWI* w następujących sytuacjach:

- *Slave* odbiera z magistrali swój własny adres,
- odebranie wywołania ogólnego, bit *TWGCE* w rejestrze *TWAR* musi być ustawiony,
- dane zostały odebrane w trybie *Master* lub *Slave*.

Zapisanie tego bitu jako zero powoduje, że urządzenie zostaje wirtualnie i tymczasowo odłączone od magistrali *TWI*. Rozpoznanie adresowe urządzenie może być wznowione poprzez ponowne zapisanie bitu *TWEA* jako jeden.

Bit 5 – TWSTA: TWI START Condition Bit

Aplikacja zapisuje ten bit jako jeden kiedy chce stać się układem *Master*. Sprzętowe *TWI* sprawdza, czy jeśli magistrala jest dostępna oraz generuje akcje *START*, jeśli magistrala jest wolna. Z kolei jeśli magistrala nie jest wolna, *TWI* czeka dopóki nie wykryje żądania *STOP* i wtedy generuje żądanie *START*, domagając się statusu *Master*. *TWSTA* musi być wyczyszczone przez oprogramowanie kiedy zostanie wysłana akcja *START* na magistralę.

Bit 4 – TWSTO: TWI STOP Condition Bit

Jeśli w trybie *Master* nada się *TWSTO* jedynkę logiczną, wtedy generowane jest żądanie *STOP*. Kiedy akcja ta jest wykonywana na magistrali, bit ten jest automatycznie czyszczony.

Bit 3 – TWWC: TWI Write Collion Flag

Bit ten jest ustawiany w momencie próby zapisu do rejestru *TWI Data Register – TWDR*, kiedy *TWINT* ustawiony jest na zero. Flaga jest czyszczona poprzez zapis do *TWDR* (bit *TWINT* jest w stanie wysokim).

Bit 2 – TWEN: TWI Enable Bit

Bit ten uaktywnia obsługę *TWI* oraz aktywuje interfejs *TWI*. Kiedy bit *TWEN* jest równy jeden, wtedy *TWI* przejmuje kontrolę pinów *I/O*, dokładniej *SCL* i *SDA*, włączając przy tym ogranicznik *SR* oraz filtr impulsów. Jeśli zapisany jako zero, *TWI* jest wyłączone, wszystkie transmisje są przerywane bez względu na trwające operacje.

Bit 1 – Res: Reserved Bit

Ustawiony zawsze na zero – bit zastrzeżony.

Bit 0 – TWIE: TWI Interrupt Enable

Odpowiada za odblokowanie przerwań *TWI*. Bit powiązany z flagą *TWINT*.

TWI Status Register – TWSR

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

Bity 7..3 – TWS: TWI Status

Te pięć bitów odzwierciedla status logiki *TWI* oraz magistrali *TWI*.

Bit 2 – Res: Reserved Bit

Bity 1..0 – TWPS: TWI Prescaler Bits

Ustawienia tych bitów definiują preskaler. Poniżej znajduje się tabela z dostępnymi wartościami:

TWPS1	TWPS0	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

Rysunek. *Ustawienie preskalera za pomocą bitów TWPS1 i TWPS0*

TWI Data Register – TWDR

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

W trybie nadajnika, *TWDR* zawiera następny bajt przeznaczony do transmisji. W trybie odbiornika, *TWDR* zawiera ostatni odebrany bajt. Możliwy jest zapis do rejestru jeśli *TWI* nie jest w trakcie przenoszenia bajtu.

Bity 7..0 – TWD: TWI Data Register

Bity te zawierają następny bajt danych przeznaczony do wysłania, lub ostatni odebrany z magistrali bajt.

TWI (Slave) Address Register – TWAR

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

Rejestr TWAR powinien być uzupełniony siedmiobitowym adresem slave'a. W systemie multi-master, TWAR musi być ustawione również w wypadku układów Master. Najmłodszy bit tego rejestru jest używany do aktywacji rozpoznawania adresu wywołania ogólnego (*0x00*).

Bity 7..1 – TWA: TWI (Slave) Address Register

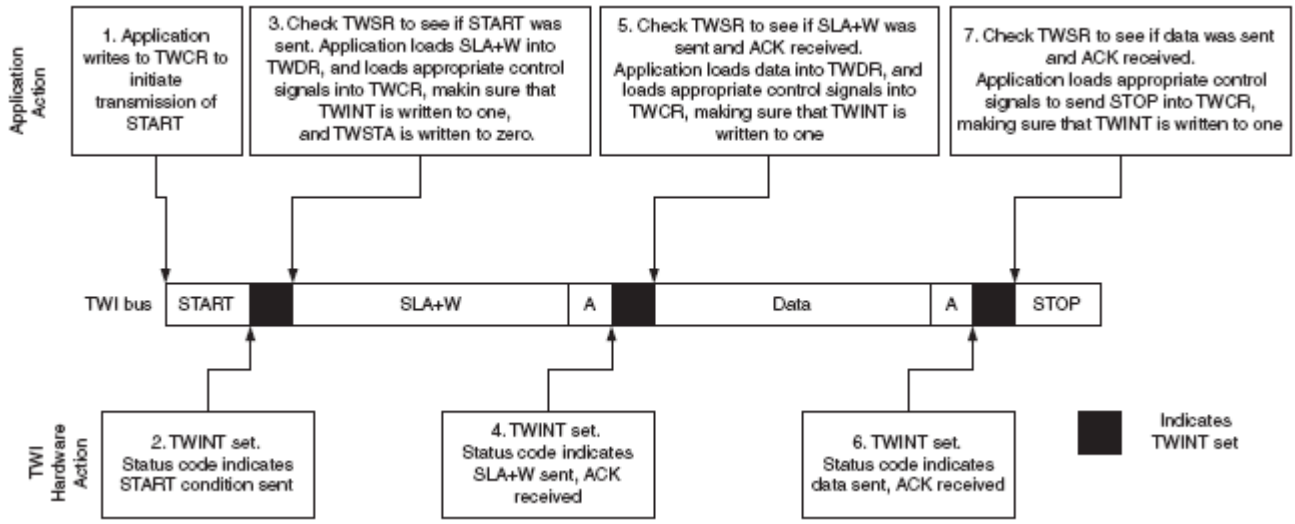
Bity te stanowią adres układu Slave.

Bit 0 – TWGCE: TWI General Call Recognition Enable Bit

Jeśli ustawiony wtedy aktywne jest rozpoznawanie wywołania ogólnego.

KORZYSTANIE Z TWO-WIRE SERIAL INTERFACE

Poniżej przedstawiony prosty przykład opisujący transmisję przez magistralę *TWI*. W tym przykładzie *Master* chce przesłać pojedynczy bit danych do układu *Slave*.



Rysunek. *Typowa obsługa urządzenia poprzez interfejs TWI*

1. Pierwszym krokiem nawiązania transmisji jest wysłanie żądania *START*. Realizowane jest to poprzez zapis odpowiedniego bitu w rejestrze *TWCR*.
2. Kiedy żądanie *START* zostało przesłane, flaga *TWINT* w rejestrze *TWCR* jest ustawiana i rejestr *TWSR* jest uaktualniany z kodem statusu wskazującym na pomyślność akcji *START*.
3. Aplikacja powinna sprawdzić wartość *TWSR* aby upewnić się czy żądanie zostało pomyślnie wysłane. Jeśli *TWSR* zawiera inną wartość niż oczekiwana przez aplikacje, można podjąć specjalne akcje, np. wygenerowanie błędu. Zakładając, że kod statusu jest poprawny, aplikacja musi załadować bit *SLA+W* do rejestru *TWDR*. Trzeba pamiętać, że rejestr *TWDR* jest używany zarówno dla adresu jak i dla danych. Po załadowaniu rejestru *TWDR* przez *SLA+W*, określona wartość musi być zapisana do rejestru *TWCR* zgodnie z ustawieniami sprzętowymi *TWI* do transmisji *SLA+W* określonego w rejestrze *TWDR*. *TWI* nie rozpocznie żadnego działania tak długi jak bit *TWINT* w rejestrze *TWCR* będzie ustawiony.

Natychmiast po tym jak aplikacja wyczyści bit *TWINT*, *TWI* zainicjalizuje transmisję z danymi adresowymi

4. Kiedy dane z adresem zostaną dostarczone, flaga *TWINT* w rejestrze *TWCR* zostanie ustawiona oraz rejestr *TWSR* zostanie zaktualizowany zgodnie z kodem statusu przypisanego do poprawności transmisji.
5. Oprogramowanie powinno zbadać wartość rejestru *TWSR*, aby upewnić się, że paczka z adresem została pomyślnie przesłana, i że wartość bitu *ACK* jest zgodna z oczekiwaną. Jeśli zaś rejestr *TWSR* zawiera inne dane od oczekiwanych, oprogramowanie powinno podjąć odpowiednią akcję. Jeśli stwierdzono poprawność wartości *TWSR*, aplikacja musi załadować otrzymane dane adresowe do rejestru *TWDR*.
6. Kiedy pakiet danych zostanie przesłany flaga *TWINT* w rejestrze *TWCR* zostanie ustawiona, a rejestr *TWSR* zostanie zaktualizowany.
7. Aplikacja powinna sprawdzić poprawność rejestru *TWSR* oraz wartość bitu *ACK*.
8. Transmisja zostaje zakończona żądaniem *STOP*.

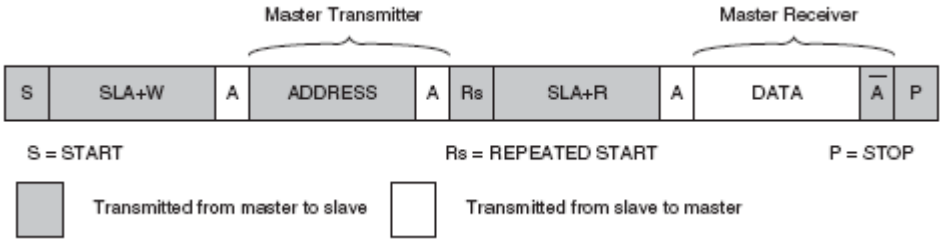
Poniżej przedstawiono zasięgniętą z dokumentacji *ATmega16* tabelę z poszczególnymi operacjami w języku C wraz z tłumaczeniem.

Lp.	Kod w języku C	Objaśnienia
1	`TWCR = (1<<TWINT)|(1<<TWSTA)| (1<<TWEN)`	Wysłanie żądania START.
2	`while (!(TWCR & (1<<TWINT))) ;`	Oczekiwanie na ustawienie flagi TWINT. Sygnalizuje to, że przesłanie START zostało przesłane.
3	`if ((TWSR & 0xF8) != START) ERROR();`	Sprawdzenie wartości statusu w rejestrze TWI Status Register
4	`TWDR = SLA_W;` `TWCR = (1<<TWINT) | (1<<TWEN);`	Załadowanie wartości SLA+W do rejestru TWDR. Wyczyszczenie bitu TWINT w rejestrze TWCR, aby możliwe było wysłanie adresu.
5	`while (!(TWCR & (1<<TWINT))) ;`	Oczekiwanie na ustawienie flagi TWINT. Sygnalizuje to, że bit SLA+W został wysłany, zaś bit ACK/NACK został odebrany.

6	`if ((TWSR & 0xF8) != MT_SLA_ACK)` `ERROR();`	Sprawdzenie wartości w statusu w TWSR.
7	`TWDR = DATA;` `TWCR = (1<<TWINT) | (1<<TWEN);`	Załadowanie danych do rejestru TWDR. Wyczyszczenie bitu TWINT, aby możliwa była transmisja danych.
8	`while (!(TWCR & (1<<TWINT)))` `;`	Oczekiwanie na ustawienie flagi TWINT. Sygnalizuje do wysłanie danych i otrzymanie bitu ACK/NACK.
9	`if ((TWSR & 0xF8) != MT_DATA_ACK)` `ERROR();`	Sprawdzenie wartości TWSR.
10	`TWCR = (1<<TWINT)|(1<<TWEN)|` `(1<<TWSTO);`	Wysłanie żądania STOP.

OPCJE TRANSMISJI

Interfejs *TWI* obsługuje jeden z czterech trybów. Nazwy tych trybów to: Master Transmitter (**MT**), Master Receiver (**MR**), Slave Transmitter (**ST**) oraz Slave Receiver (**SR**). Kilka z nich może być użytych w tej samej aplikacji. Za przykład niech posłuży zapisywanie danych do układu **EEPROM** w trybie **MT**, odczytywanie z kolei odbywa się w trybie **MR**. Poniżej przedstawiono opisany przykład.



Rysunek. *Przykład odczytu danych z pamięci EEPROM*

TRYB MASTER TRANSMITTER (MT)

W tym trybie pracy, określona ilość bajtów danych jest wysyłana do odbiornika *Slave*. Wysłanie żądania *START* jest rozkazem wejścia w ten tryb pracy. Format wysyłanego adresu

zależy od trybu pracy. Bit *SLA+W* jest przesyłany w trybie *MT*, zaś bit *SLA+R* w trybie *MR*. Żądanie *START* zostaje wywołane następującymi zapisami w rejestrze *TWCR*.

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	1	0	X	1	0	X

Bit *TWEN* musi być ustawiony aby aktywny był interfejs *TWI*. Bit *TWSTA* musi być ustawiony aby wysłać żądanie *START*, zaś bit *TWINT* po to aby wyczyścić flagę *TWINT*.

TWI sprawdzi magistralę i wygeneruje żądanie *START* tak szybko jak magistrala będzie wolna. Następnie flaga *TWINT* zostanie sprzętowo ustawiona zaś status *TWSR* przyjmie wartość *0x08*. W tym trybie musi zostać przesłany bit *SLA+W*. Jest to czynione poprzez zapis *SLA+W* w rejestrze *TWDR*. Od tego czasu bit *TWINT* powinien być czysty aby kontynuować wysyłanie danych. Jest to realizowane za pomocą zapisania następujących wartości w rejestrze *TWCR*.

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	0	0	X	1	0	X

Schemat działania powtarza się dopóki ostatni bajt danych nie zostanie przesłany. Transmisja jest kończona w momencie wygenerowania żądania *STOP* lub powtórnego startu. Żądanie *STOP* generowane jest przez ustawienie odpowiednich wartości w rejestrze *TWCR*. Ilustruje to diagram poniżej

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	0	1	X	1	0	X

Z kolei żądanie *REPEATED START* generowane jest zapisanie następujących wartości do rejestru TWCR:

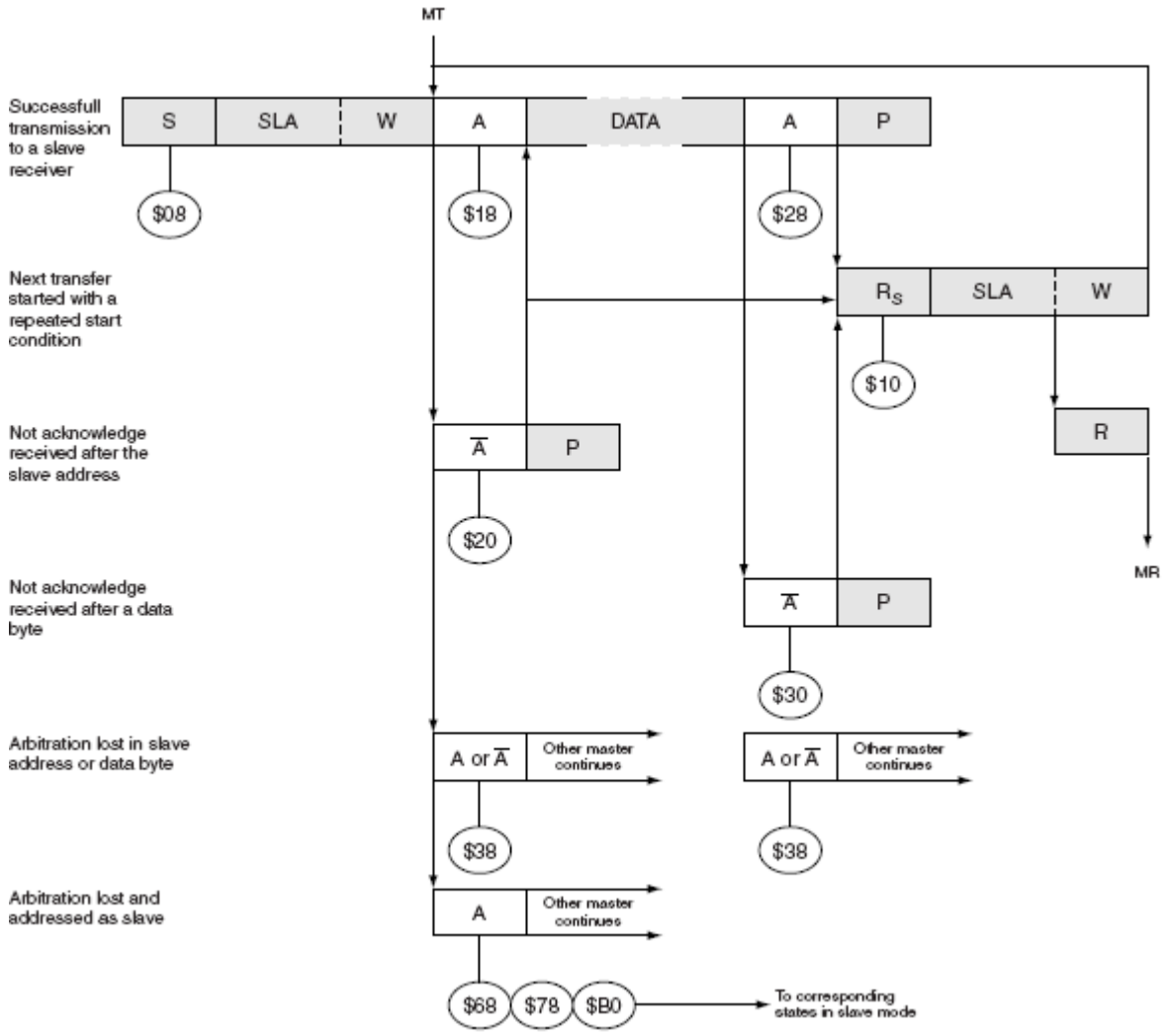
TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	1	0	X	1	0	X

Po wywołaniu powtórnego startu (stan *0x10*) interfejs *TWI* może połączyć się z tym samym układem *Slave*, lub z nowym z pominięciem transmisji żądania *STOP*. *REPEATED START*

umożliwia przełączanie pomiędzy układami podrzędnymi, trybami *Master Transmitter*, *Master Receiver* bez tracenia kontroli magistrali.

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x08	A START condition has been transmitted	Load SLA+W	0	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received
0x10	A repeated START condition has been transmitted	Load SLA+W or	0	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received
		Load SLA+R	0	0	1	X	SLA+R will be transmitted; Logic will switch to Master Receiver mode
0x18	SLA+W has been transmitted; ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
0x20	SLA+W has been transmitted; NOT ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
0x28	Data byte has been transmitted; ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
0x30	Data byte has been transmitted; NOT ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
0x38	Arbitration lost in SLA+W or data bytes	No TWDR action or	0	0	1	X	Two-wire Serial Bus will be released and not addressed Slave mode entered
		No TWDR action	1	0	1	X	A START condition will be transmitted when the bus becomes free

Rysunek. *Tabela statusów dla trybu Master Transmitter*



Rysunek. *Ilustracja zasady działania trybu Master Transmitter*

TRYB MASTER RECEIVER

W tym trybie określona ilość bajtów danych jest odbierana przez układ Master. Dane te są nadawana przez *Slave Transmitter*. Aby wejść w ten tryb należy wysłać żądanie *START*.

Żądanie *START* jest generowane przez ustawienie następujących kombinacji bitów w rejestrze *TWCR*:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	1	0	X	1	0	X

Bit *TWEN* musi być ustawiony aby aktywny był interfejs *TWI*. Bit *TWSTA* odpowiedzialny jest za wysłanie żądania *START*. Bit *TWINT* musi być ustawiony, aby wyczyszczona była flaga *TWINT*. *TWI* testuje magistralę i jeżeli ta jest wolna wysyła żądanie *START*. Po wysłaniu

żądania flaga *TWINT* jest sprzętowo ustawiana i kod w rejestrze *TWSR* powinien wynosić ***0x08***. Bit *SLA+R* w trybie *MR* musi być przesłany. Jest to wykonywane przez zapisanie bitu *SLA+R* w rejestrze *TWDR*. Po tym bit *TWINT* powinien być wyczyszczony aby kontynuować transmisję. Jest to osiągane za pomocą zapisania następujących bitów w rejestrze *TWCR*:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	0	1	X	1	0	X

Kiedy wysłany zostanie *SLA+R* oraz bit potwierdzenia zostanie odebrany ustawiana jest flaga *TWINT* oraz wystawiana odpowiednia wartość w rejestrze *TWSR*. Możliwe kody statusów to: ***0x38, 0x40, 0x48***. Odebrane dane mogą być odczytane z rejestru *TWDR* kiedy flaga *TWINT* ustawiona jest sprzętowo na poziom wysoki.

Taki schemat działania powtarza się dopóki ostatni bajt danych nie zostanie odebrany. Po ostatnim bajcie *MR* powinien poinformować *ST* przez wysłanie *NACK* po ostatnim odebrany bajcie danych. Transfer danych jest kończony przez wygenerowanie żądania *STOP* lub powtórzonego *START*. *STOP* generowane jest przez następujące ustawienie rejestru *TWCR*:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	0	1	X	1	0	X

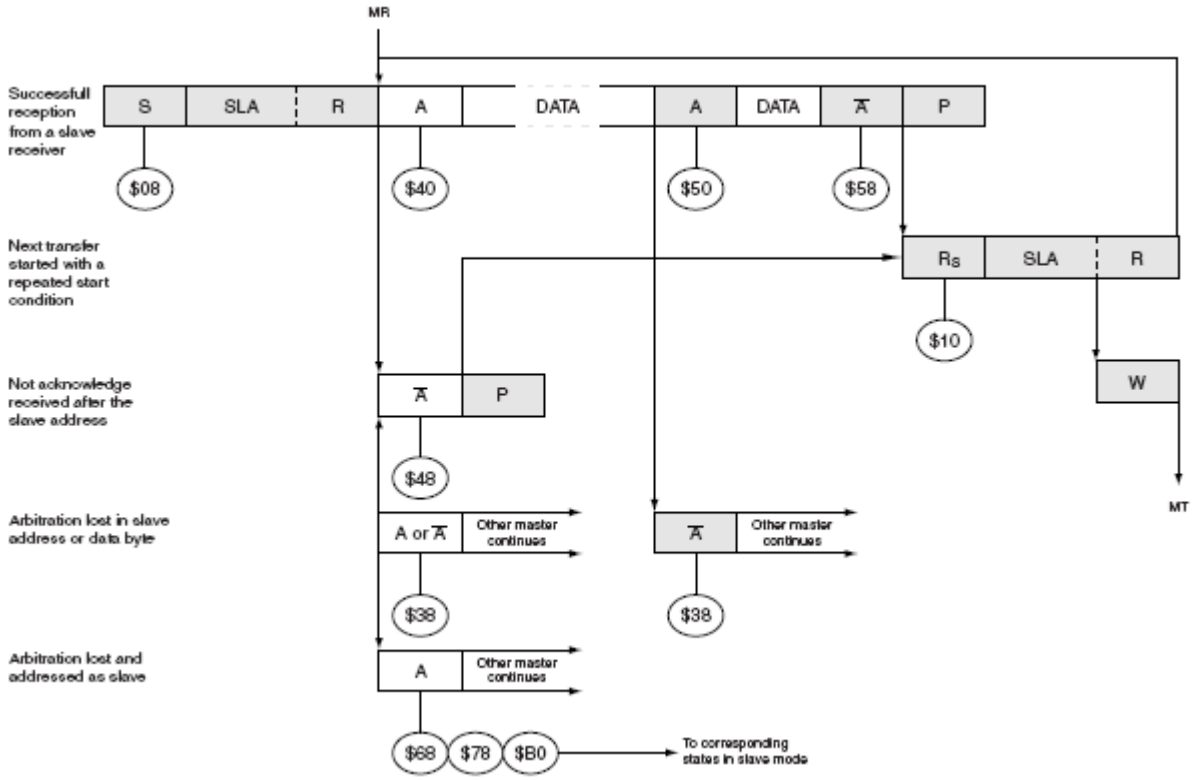
Z kolei *REPEATED START* jest generowane przez poniższe ustawienia w rejestrze *TWCR*:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	1	0	X	1	0	X

Po wystąpieniu powtórnego startu (***0x10***) interfejs *TWI* może połączyć się z tym samym układem podrzędnym, lub z nowym, z pominięciem transmisji żądania *STOP*. Umożliwia to przełączanie *Mastera* pomiędzy: układami podrzędnymi, *MT*, *MR* bez tracenia kontroli nad magistralą.

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x08	A START condition has been transmitted	Load SLA+R	0	0	1	X	SLA+R will be transmitted ACK or NOT ACK will be received
0x10	A repeated START condition has been transmitted	Load SLA+R or	0	0	1	X	SLA+R will be transmitted ACK or NOT ACK will be received
		Load SLA+W	0	0	1	X	SLA+W will be transmitted Logic will switch to Master Transmitter mode
0x38	Arbitration lost in SLA+R or NOT ACK bit	No TWDR action or	0	0	1	X	Two-wire Serial Bus will be released and not addressed Slave mode will be entered
		No TWDR action	1	0	1	X	A START condition will be transmitted when the bus becomes free
0x40	SLA+R has been transmitted; ACK has been received	No TWDR action or	0	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	0	0	1	1	Data byte will be received and ACK will be returned
0x48	SLA+R has been transmitted; NOT ACK has been received	No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
0x50	Data byte has been received; ACK has been returned	Read data byte or	0	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	0	0	1	1	Data byte will be received and ACK will be returned
0x58	Data byte has been received; NOT ACK has been returned	Read data byte or	1	0	1	X	Repeated START will be transmitted
		Read data byte or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		Read data byte	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset

Rysunek. *Tabela statusów dla trybu Master Receiver*



Rysunek. *Ilustracja zasady działania trybu Master Receiver*

TRYB SLAVE RECEIVER

W trybie tym określona ilość danych jest dostarczana z układu *Master Transmitter* (*MT*). Aby zainicjować tryb *SR*, należy zadeklarować odpowiednie ustawienia w rejestrach *TWAR* i *TWCR*.

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Value	Device's Own Slave Address							

Siedem najstarszych bitów odpowiada za adres, na który będzie reagować układ *Slave*. Jeśli ustawiony jest najmłodszy bit *TWGCE* wtedy *TWI* będzie odpowiadać na wywołanie ogólne (*0x00*), w innym przypadku będzie ignorować adres wywołania ogólnego.

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	0	1	0	0	0	1	0	X

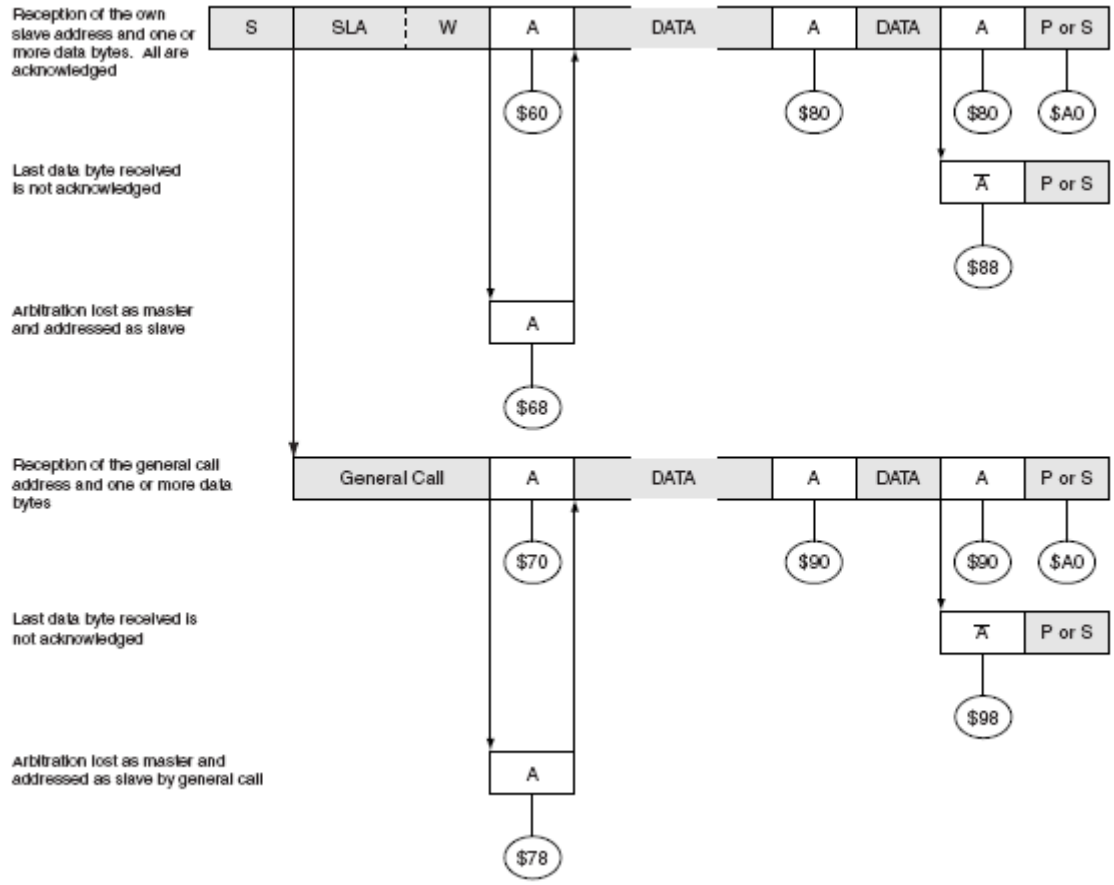
Bit *TWEN* musi być ustawiony aby aktywny był interfejs *TWI*. Bit *TWEA* musi być ustawiony aby aktywować potwierdzanie adresu Slave lub adresu wywołania ogólnego. *TWSTA* i *TWSTO* muszą być ustawione na zero.

Kiedy rejestry *TWAR* i *TWCR* zostaną ustawione, *TWI* czeka dopóki nie zostanie zaadresowane adresem *Slave* (lub adresem wywołania ogólnego). Po otrzymaniu odpowiedniego adresu Slave i zapisaniu bitu odebranego, flaga *TWINT* ustawiana jest, generowany jest kod statusu dostępnego w rejestrze *TWSR*. Jeśli bit *TWEA* zostanie wyczyszczony podczas transmisji, *TWI* zwróci „nie potwierdzenie" po ostatnim odebranym bajcie danych. Może być to używane to zasygnalizowania, że *Slave* nie jest w stanie przyjąć większej ilości danych. Kiedy *TWEA* ustawione na zero, *TWI* nie będzie potwierdzać otrzymania własnego adresu. Jednakże *TWI* będzie ciągle kontrolowana, dlatego rozpoznawanie adresu można przywrócić przez ponowne ustawienie bitu *TWEA*.

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x60	Own SLA+W has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x68	Arbitration lost in SLA+R/W as Master; own SLA+W has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x70	General call address has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x78	Arbitration lost in SLA+R/W as Master; General call address has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x80	Previously addressed with own SLA+W; data has been received; ACK has been returned	Read data byte or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	X	0	1	1	Data byte will be received and ACK will be returned
0x88	Previously addressed with own SLA+W; data has been received; NOT ACK has been returned	Read data byte or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		Read data byte or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = "1"
		Read data byte or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		Read data byte	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = "1"; a START condition will be transmitted when the bus becomes free
0x90	Previously addressed with general call; data has been received; ACK has been returned	Read data byte or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	X	0	1	1	Data byte will be received and ACK will be returned
0x98	Previously addressed with general call; data has been received; NOT ACK has been returned	Read data byte or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		Read data byte or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = "1"
		Read data byte or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		Read data byte	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = "1"; a START condition will be transmitted when the bus becomes free
0xA0	A STOP condition or repeated START condition has been received while still addressed as Slave	No action	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
			0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = "1"
			1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
			1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = "1"; a START condition will be transmitted when the bus becomes free

Rysunek. *Tabela statusów dla trybu Slave Receiver*

Ilustracja zasady działania trybu Slave Receiver



Rysunek. *Ilustracja zasady działania trybu Slave Receiver*

TRYB SLAVE TRANSMITTER

W tym trybie określona ilość bajtów danych jest wysyłana do *Master Receiver* (*MR*). Aby zainicjować transmisję należy ustawić odpowiednie bity w rejestrach *TWAR* i *TWCR*.

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Value	Device's Own Slave Address							

Siedem najstarszych bitów określa adres, na który *TWI* będzie odpowiadać. Najmłodszy bit *TWGCE* ustawiony na jeden powoduje iż *TWI* odpowiada na wywołania ogólne, w przeciwnym przypadku ignoruje adres wywołania ogólnego (***0x00***).

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	0	1	0	0	0	1	0	X

Bit *TWEN* musi być ustawiony aby aktywny był interfejs *TWI*. Bit *TWEA* musi być ustawiony aby aktywować potwierdzanie przez układ podrzędny otrzymania własnego adresu lub adresu wywołania ogólnego. *TWSTA*, *TWSTO* powinny być ustawione na zero.

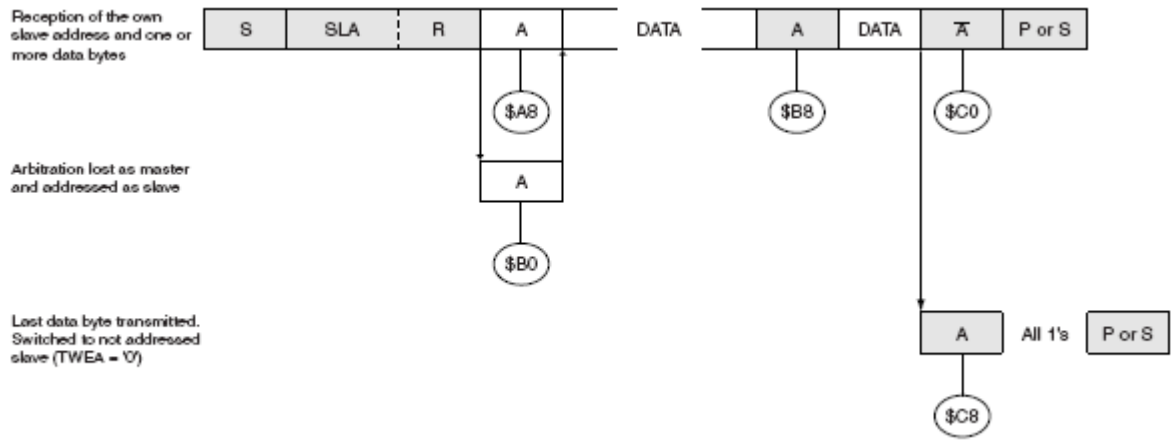
Kiedy zostaną zainicjowane rejestry *TWAR* i *TWCR*, *TWI* czeka dopóki nie zostanie zaadresowane własnym adresem *Slave'a* (lub wywołaniem ogólnym) podążającym za bitem kierunku danych. Jeśli bit ten wynosi jeden (odczyt) wtedy *TWI* obsługuje tryb *ST*, w przeciwnym przypadku obsługuje *SR*. Po otrzymaniu własnego adresu, ustawiana jest flaga TWINT oraz wystawiany odpowiedni kod statusu w rejestrze *TWSR*.

Jeśli bit *TWEA* ustawiony jest na zero podczas transmisji, *TWI* wysyła ostatni bajt z nadajnika. Stan *0xC0* lub *0xC8* wystąpi w zależności od tego czy Master Receiver wyśle *NACK* lub *ACK* po ostatnim bajcie.

W czasie kiedy *TWEA* jest równe zero, *TWI* nie odpowiada na swój adres. Jednak, magistrala *TWI* jest ciągle kontrolowana i rozpoznawanie adresu może zostać wznowione w dowolnym czasie przez ustawienie bitu *TWEA*. Taki zabieg pozwala na chwilowe odizolowanie *TWI* od magistrali.

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0xA8	Own SLA+R has been received; ACK has been returned	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
0xB0	Arbitration lost in SLA+R/W as Master; own SLA+R has been received; ACK has been returned	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
0xB8	Data byte in TWDR has been transmitted; ACK has been received	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
0xC0	Data byte in TWDR has been transmitted; NOT ACK has been received	No TWDR action or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		No TWDR action or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = "1"
		No TWDR action or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		No TWDR action	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = "1"; a START condition will be transmitted when the bus becomes free
0xC8	Last data byte in TWDR has been transmitted (TWEA = "0"); ACK has been received	No TWDR action or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		No TWDR action or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = "1"
		No TWDR action or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		No TWDR action	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = "1"; a START condition will be transmitted when the bus becomes free

Rysunek. *Tabela statusów dla trybu Slave Transmitter*



Rysunek. *Ilustracja zasady działania trybu Slave Transmitter*

RÓŻNE STANY

Występują dwa kody statusów, które nie zostały wcześniej zdefiniowane. Poniżej znajduje się tabela uzupełniająca dokumentację o te kody.

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0xF8	No relevant state information available; TWINT = "0"	No TWDR action	No TWCR action				Wait or proceed current transfer
0x00	Bus error due to an illegal START or STOP condition	No TWDR action	0	1	1	X	Only the internal hardware is affected, no STOP condition is sent on the bus. In all cases, the bus is released and TWSTO is cleared.

Rysunek. *Tabela specjalnych stanów rejestru statusu*

WYBRANE PARAMETRY TWI

Symbol	Parameter	Condition	Min	Max	Units
V_{IL}	Input Low-voltage		-0.5	0.3 V_{CC}	V
V_{IH}	Input High-voltage		0.7 V_{CC}	V_{CC} + 0.5	V
f_{SCL}	SCL Clock Frequency	f_{CK}[4] > max(16f_{SCL}, 250kHz)[5]	0	400	kHz
Rp	Value of Pull-up resistor	$f_{SCL} \leq$ 100 kHz	$\frac{V_{CC} - 0,4V}{3mA}$	$\frac{1000ns}{C_b}$	Ω
		f_{SCL} > 100 kHz	$\frac{V_{CC} - 0,4V}{3mA}$	$\frac{300ns}{C_b}$	Ω

Rysunek. *Tabela wybranych informacji o TWI*

Więcej informacji znajduje się w *datasheet ATmega16* w rozdziale *Two-Wire Serial Interface.*

3. PAMIĘĆ EEPROM 24C16

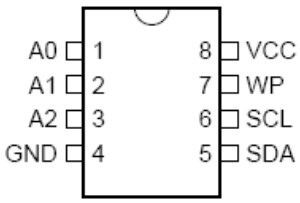
Przy prezentowaniu możliwości interfejsu *TWI* oraz magistrali *TWI* (**I²C**) założono prezentację implementacji **obsługi pamięci EEPROM**. Zanim jednak do implementacji dojdzie należy zapoznać się z notą katalogową układu **24C16**. Oczywiście wiele producentów wytwarza takie układy. Część z nich używa nazwy *I²C* lub *IIC* przy opisie rodzaju transmisji, *Atmel* używa nazwy *Two-wire interface (TWI)*. My skorzystam z kilku not katalogowych aby ogólnie omówić rozwiązania wykorzystywane przy korzystania z tych pamięci.

24C16

Układ ten jest pamięcią typu *EEPROM* o pojemności **16kb** (kilobitów). Zazwyczaj organizacja danych w układzie wygląda następująco: *8 x 2048 bitów* (*8 x 256B*). Interfejs komunikacyjny *2-wire serial* (kompatybilny z I²C). Układ może posiadać zabezpieczenie sprzętowe przed zapisem. Napięcia zasilające: nawet od **+1,8V** do **+5,5V**. Milion cyklów zapisu do standard, trwałość *100* lat.

Poniżej znajduje się wygląd wyprowadzeń układu w obudowie DIP8.

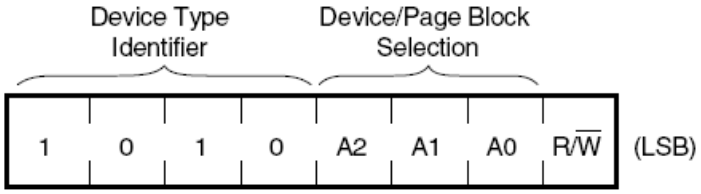
Pin Name	Function
A0 to A2	Address Inputs
SDA	Serial Data
SCL	Serial Clock Input
WP	Write Protect



Rysunek. *Wyprowadzenia pamięci 24C16 i ich przeznaczenie*

Niektórzy producenci wyeliminowali nóżki adresowe **A0, A1, A2** (*NC*). Nóżka **WP** chroni przed zapisem. **SCL, SDA** to linie magistrali *I²C*.

Poniżej znajduje się typowa ramka adresu *Slave* układu *24C16*. Ramka ta nosi nazwę **Device Address** i w obsłudze wysyłania i odbierania informacji jest nieodzowna.



Rysunek. *Wygląd ramki adresowej 24C16*

Warto dodać, że sprzętowo można ustawić odpowiednio bity *A2, A1 i A0*. Z kolei najmłodszy bit *R/W* równy jedynce logicznej odpowiada za odczyt w przeciwnym wypadku za zapis. Zazwyczaj w rozwiązaniach spotyka się podłączaniem wyprowadzeń *A2, A1, A0* do masy – wtedy pierwszy adres zapisu wynosi w systemie decymalnym *160*[5], zaś odczytu *161*.

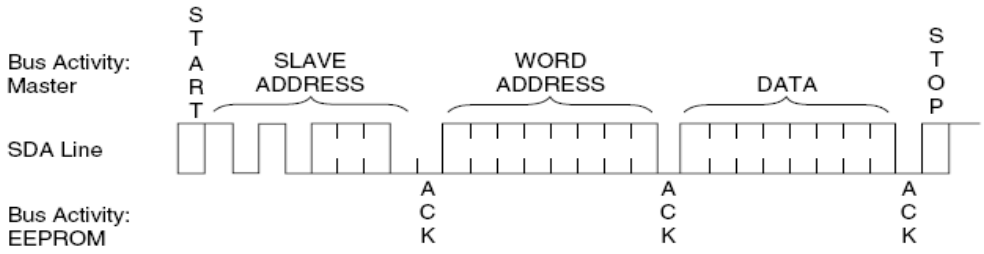
Trzeba nadmienić, że układ 24C16 posiada 8 adresów zapisu i tyle samo odczytu. Przykładowe adresy dla zapisu: *160, 162, 164, 166, 168, 170, 172, 174*.

TRYB ZAPISU

Algorytm zapisu pojedynczego bajtu przedstawiono poniżej.

- wysłanie sekwencji *START*,
- wysłanie bajtu adresu na magistralę z bitem *R/W* w stanie niskim (zapis),
- wysłanie bajtu adresu komórki pamięci w uprzednio zaadresowanym układzie pamięci,
- wysłanie bajtu danych przeznaczonego do zapisu,
- wysłanie sekwencji *STOP*.

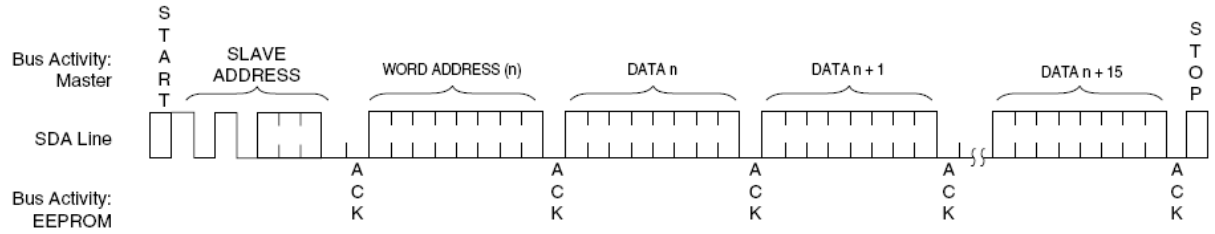
Poniższy rysunek ilustruje działanie algorytmu.



Rysunek. *Zapis bajtu do pamięci 24C16*

[5] $160_{(10)}=1010\ 0000_{(2)}$ (zapis)
$161_{(10)}=1010\ 0001_{(2)}$ (odczyt)

Pamięć potwierdza odebranie każdego bajtu poprzez ustawieniu na magistralę bitu ACK (stan niski *SDA*). Poniżej przedstawiono ramkę zapisu stronicowego danych.

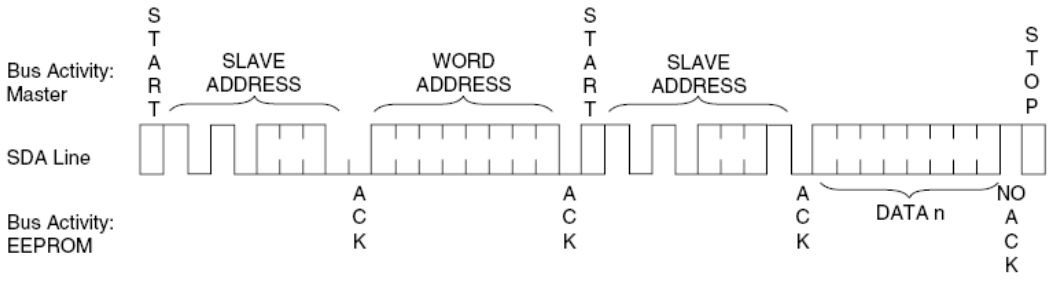


Rysunek. *Zapis stronicowy do pamięci 24C16*

TRYB ODCZYTU

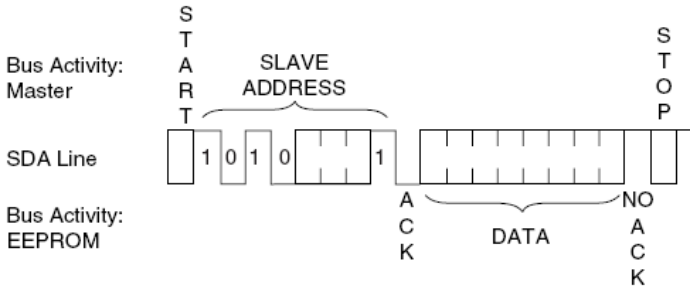
Algorytm odczytu pojedynczego bajtu przedstawiono poniżej.

- wysłanie sekwencji *START*,
- wysłanie bajtu adresu układu pamięci na magistralę z bitem *R/W* w stanie niskim,
- wysłanie bajtu adresu komórki pamięci, spod której nastąpi odczyt,
- wysłanie sygnału *START*,
- wysłanie bajtu adresu układu pamięci z bitem *R/W* w stanie wysokim (odczyt),
- odebranie bajtu danych wysłanych przez układ pamięci,
- wystawienie sygnału braku potwierdzenia (*NACK*),
- wysłanie sekwencji *STOP*.



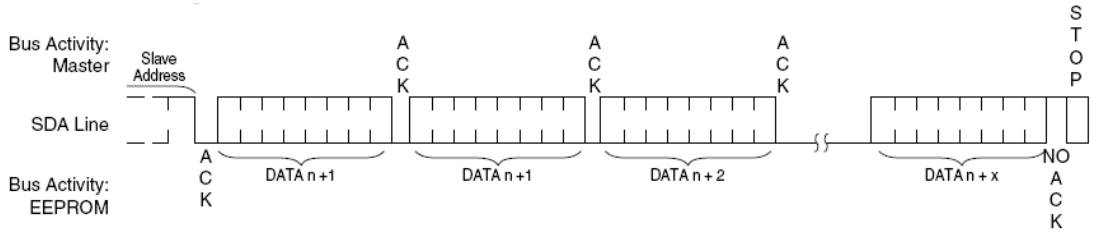
Rysunek. *Odczyt bajtu danych z komórki pamięci układu 24C16*

Odczyt bajtu z aktualnego adresu prowadzi się według poniższej ramki.



Rysunek. *Odczyt bajtu z aktualnego adresu komórki układu 24C16*

Odczyt sekwencyjny został przedstawiony w ramce poniżej.



Rysunek. *Odczyt sekwencyjny danych z pamięci układu 24C16*

Posiadając wiedzę na temat zapisu i odczytu do pamięci oraz znając możliwości modułu *TWI*, możemy przejść do implementacji programowej obsługi magistrali *I²C*.

4. OBSŁUGA PAMIĘCI EEPROM W JĘZYKU C

Głównym celem tego opracowania jest prezentacja implementacji programowej obsługi interfejsu *TWI* (*I²C*). Założono, że programowany **mikrokontroler będzie posiadał moduł sprzętowy TWI**. Wybrano kontroler **ATmega16**, posiadający **16kB pamięci flash** i przede wszystkim posiadający, nas interesujący, **interfejs TWI**. Środowiskiem do stworzenia programu będzie *WinAVR*. Z kolei jako język programowania przyjęto *Ansi C* (kompilator *GCC*). Założeniami pierwszego programu jest wysłanie i odebranie pojedynczego bajtu z pamięci układu **24C16**.

PROGRAM PIERWSZY

Program pierwszy zakładał stworzenie funkcji elementarnych, takich jak: wysłanie sekwencji *START* na magistralę, zatrzymanie transmisji, odbiór i wysłanie pojedynczych bajtów itd. Procesor będzie pracował w trybie *Master Transmitter* (*MT*). Pierwszą stworzoną funkcją jest funkcja wysyłająca żądanie *START* na magistralę. Pomocny jest tutaj opis *TWI* (*datasheet ATmega16*). Należy zaznaczyć iż w prezentowanej obsłudze interfejsu *TWI* wykorzystuje się tzw. **polling** (pętle oczekujące na ustawienie danej flagi). W bardziej wyrafinowanych i krytyczno czasowych aplikacjach powinno wykorzystywać się dostępne przerwania.

(1)

```
//Funkcja inicjuje transmisje w trybie Master
void vTWIStart()
{
    TWCR=(1<<TWINT)|(1<<TWEN)|(1<<TWSTA);    //wyslanie sekwencji start
    while(!(TWCR&(1<<TWINT)));               //oczekiwanie na flage TWINT
}
```

Ustawienie bitu **TWINT** powoduje odblokowanie przerwań. Z kolei ustawienie **TWSTA** aktywuje wygenerowanie w trybie *Master* sekwencji startu transmisji. Bit **TWEN** odblokowuje interfejs *TWI*. Pętla *while()* wykonywana jest dopóki w rejestrze **TWCR** nie zostanie ustawiony bit **TWINT**. Bit ten jest ustawiany sprzętowo po zakończeniu bieżącej operacji przez interfejs. Kolejną przedstawioną funkcją jest funkcja wysyłająca sekwencję *STOP* na magistralę.

(2)

```
//Funkcja konczy transmisje
void vTWIStop()
{
    TWCR=(1<<TWINT)|(1<<TWEN)|(1<<TWSTO);    //wyslanie sekwenci stop
    while((TWCR&(1<<TWSTO)));                //oczekiwanie na bit stopu
}
```

Funkcja opiera się na ustawieniu wartości w rejestrze **TWCR**. Bit **TWSTO** odpowiedzialny jest za wygenerowanie żądania *STOP* i wysłanie go na magistralę. Po wykonaniu zadania bit **TWSTO** jest sprzętowo czyszczony. Pętla *while()* działa dopóki w rejestrze **TWCR** bit **TWSTO** nie będzie równy zero. Procedura poniżej przedstawia wysłanie pojedynczego bajtu.

(3)

```
//Funkcja wysyla bajt na magistrale
void vTWIWriteByte(unsigned char ucData)
{
    TWDR = ucData;                          //wyslanie bajtu
    TWCR = (1<<TWINT) | (1<<TWEN);
    while (!(TWCR & (1<<TWINT)));           //oczekiwanie na zakonczenie
}
```

Rejestr **TWDR** zawiera bajt danych do wysłania/odebrania. Aktywując flagę *TWINT* i interfejs uruchamiamy przesył tych danych ma magistralę. Poniżej przedstawiono analogiczną strukturę odpowiedzialną za odebranie bajtu danych.

(4)

```
//Funkcja odbiera bajt danych
unsigned char ucTWIReadByte(unsigned char ucAck)
{
    TWCR = (1<<TWINT)|(ucAck<<TWEA)|(1<<TWEN);  //ustawienie rejestru TWCR
    while (!(TWCR & (1<<TWINT)));               //oczekiwanie na flage
    return TWDR;                                //zwrocenie wartosci
}
```

Ustawienia bitu **TWINT** i **TWEN** są analogiczne do wcześniej przytoczonych kodów źródłowych. Z kolei ustawienie lub wyczyszczenie bitu **TWEA** jest odpowiedzialne za generowanie potwierdzenia odebrania wiadomości.

Mając elementarne funkcje obsługujące magistralę **TWI**, zajmiemy się teraz zastosowaniem ich do odczytu i zapisu pojedynczych bajtów wiadomości do układu *EEPROM 24C16*. Analizując opis sposobu komunikowania się z tym układem poniżej prezentowany jest kod zapisujący bajt danych do komórki pamięci układu *EEPROM*.

(5)

```
//Funkcja zapisuje pojedynczy bajt danych
//do ukladu EEPROM na magistrali I2C
void vEEPROMWriteData(unsigned char ucAddress, unsigned char ucData)
{
    vTWIStart();                //sekwencji start
    vTWIWriteByte(0xA0);        //wyslanie adresu ukladu Slave
    vTWIWriteByte(ucAddress);   //wyslanie adresu komorki do zapisu
    vTWIWriteByte(ucData);      //wyslanie bajtu danych do EEPROMa
    vTWIStop();                 //wyslanie sekwencji stop
    delayms(15);
}
```

Widać, że w kodzie wykorzystano funkcje stworzone wcześniej. Algorytm zapisu zaczerpnięto z opisu układu *24C16*. Najpierw wysyłamy żądanie *START*, następnie adres

układu wraz z bitem *R/W* [6]. Kolejno wysyła się adres komórki którą chcemy obsłużyć i wreszcie bajt danych do zapisu. Transmisję kończy się poprzez wysłanie sekwencji *STOP* na magistralę. Po zatrzymaniu transmisji należy odczekać tzw. *Time Write Cycle* – dla układów *24C16* jest to od *10* do *15ms* [7]. Poniższy fragment przedstawia odczyt pojedynczego bajtu danych z układu *EEPROM*.

(6)

```
//Funkcja czyta z danej komorki ukladu bajt danych
unsigned char ucEEPROMReadData(unsigned char ucAddress)
{
    unsigned char ucReadData;

    vTWIStart();                  // sekwencja start
    vTWIWriteByte(0xA0);          //wyslanie adresu ukl. w trybie write
    vTWIWriteByte(ucAddress);     //wyslanie adresu komorki do odczytu
    vTWIStart();                  //wyslanie sekwencji start
    vTWIWriteByte(0xA1);          //wyslanie adresu ukl. w trybie read
    ucReadData=ucTWIReadByte(0);//odczyt danych i wyslanie NACK
    vTWIStop();                   //koniec transmisji

    return ucReadData;            //zwrocenie wartosci
}
```

Algorytm odczytu, jak i zapisu oparty jest o datasheet układu *24C16*. Po wysłaniu sekwencji *START*, należy wysłać adres układu (tryb zapis), następnie adres komórki do odczytu. Kolejno trzeba wysłać jeszcze raz sekwencję start, a po niej adres *Slave* (do odczytu), po tym zabiegu można odczytać dane, zwracając na magistralę *NACK*. Transmisja kończy się wysłaniem żądania *STOP*.

Mając już wszystkie funkcje możemy przystąpić do napisania prostego programu mającego na celu zapisanie pierwszego bloku pamięci (adres *0xA0*) liczbami: *255-L,* gdzie L oznacza numer komórki. Następnie następuje odczyt tych danych. Założono, że wyniki będą przedstawione na wyświetlaczu *LCD*[8] . Możliwe jest oczywiście również inne przetwarzanie, np. wysłanie na port szeregowy itd.

[6] Przypomnienie: Adres *0xA0* odpowiada zapisowi dwójkowemu: *1010 0000*. Cztery najstarsze bity są bitami fabrycznymi układu 24C16, zaś najmłodsze są konfigurowalne. Kolejne trzy bity po bitach fabrycznych są bitami adresowymi (możliwe 8 różnych adresów). Ostatni bit jest bitem R/W odpowiadającym za tryb pracy zapis/odczyt

[7] Obsługa odstępów czasowych została zrealizowana przy pomocy bibliotek RKlibAVR (avr.elektroda.eu)

[8] Obsługa wyświetlacza LCD została przeprowadzona za pomocą bibliotek RKlibAVR (avr.elektroda.eu)

(7)

```
//Glowna funkcja programu
int main(void)
{
    int iCounter;                                   //licznik
    unsigned char ucMemoryData;

    LCD_init();                                     //inicjacja wyswietlacza
    LCD_clear();                                    //wyczyszczenie wyswietlacza

    for(iCounter=0;iCounter<255;iCounter++)         //zapis danych do pamieci EEPROM
    {
        ucMemoryData=255-iCounter;
        vEEPROMWriteData(iCounter, ucMemoryData); //procedura zapisujaca
    }

    for(iCounter=0;iCounter<255;iCounter++)         //odczyt danych z pamieci EEPROM
    {
        ucMemoryData=ucEEPROMReadData(iCounter);
        LCD_putint(ucMemoryData, 10);               //wyswietlenie na LCD
        delayms(50);                                //opoznienie aby mozliwy byl odczyt
    }
}
```

PROGRAM DRUGI

Założeniami drugiego programu była prezentacja możliwości wykorzystania pamięci celem odczytu ustawień zapisanych przed odłączeniem zasilania. Z punktu widzenia standardu *TWI* dodano **zapisywanie stronicowe** oraz **obsługę błędów**. W programie wykorzystano także obsługę czterech przycisków oraz prezentacje wyników na wyświetlaczu *LCD*. Program drugi, podobnie jak i pierwszy wykonano modułowo – odpowiednie funkcje mogą zostać użyte w innych programach. Również cechą wspólną dla obu programów jest wykorzystanie tzw. *pollingu* (oczekiwanie w pętli na ustawienie danej flagi). Jest to rozwiązanie dobre dla małych i niekrytyczno-czasowych aplikacji. Należy pamiętać, że w *pollingu* program czeka na ustawienie danej flagi, a w przypadku np. błędów hardware'u, flaga może nie być nigdy ustawiana, wtedy program zawiesza się. W przypadku skorzystania z przerwań (wektor *TWI_vect*) taki problem nie pojawia się.

Celem pierwszych działań jest stworzenie funkcji zapisu i odczytu stronicowego. Jest to wygodne w wypadku odczytu i zapisu większej ilości danych. Można oczywiście zrealizować to za pomocą wcześniej stworzonej funkcji, mianowicie wielokrotnego odczytu (zapisu) pojedynczego bitu, ale wymusza to stosowanie opóźnienia czasowego (dla zapisu *15ms*), po każdej sekwencji *STOP*. Poniżej przedstawiono algorytm zapisu stronicowego.

(1)

```
//Funkcja zapisuje stronicowo liczby do pamieci EEPROM
void vEEPROMWritePage(unsigned char ucAddress, unsigned char aucData[], int iDataCount)
{
    int iCounter;                   //licznik

    vTWIStart();                    //inicjujemy magistrale
    vTWIWriteByte(0xA0);            //wysylamy adres Slave'a
    vTWIWriteByte(ucAddress);       //wysylamy adres komor do zapisu

    for(iCounter=0; iCounter<iDataCount; iCounter++)
        vTWIWriteByte(aucData[iCounter]);   //zapisujemy kolejne bajty

    vTWIStop();                             //sekwencja stop
    delayms(15);                            //odczekujemy 15ms
}
```

W wywołaniu funkcji podajemy adres początkowy, dane do zapisu, oraz ilość tych danych. Algorytm zapisu pochodzi z dokumentacji układu *EEPROM*. Po wysyłaniu sekwencji start, adresu *Slave'a* oraz adresu komórki, następuje ciągły zapis bajtów do układu. Po zakończeniu, należy wysłać sekwencję stop, a następnie odczekać *15ms* [9]. Kolejną przedstawioną funkcją jest funkcja odczytu stronicowego. Widoczna jest poniżej.

(2)

```
//Funkcja odczytuje stronicowo pamiec EEPROM
void vEEPROMReadPage(unsigned char ucAddress, unsigned char aucData[], int iDataCount)
{
    int iCounter;               //licznik

    vTWIStart();                //wyslanie sekwencji start
    vTWIWriteByte(0xA0);        //wyslanie adresu ukl. w trybie write
    vTWIWriteByte(ucAddress);   //wyslanie adresu komorki do odczytu
    vTWIStart();                //wyslanie sekwencji start
    vTWIWriteByte(0xA1);        //wyslanie adresu ukl. w trybie read

    for(iCounter=0; iCounter<iDataCount; iCounter++)
        aucData[iCounter]=ucTWIReadByte(1);     //odczyt,wysylamy ACK

    aucData[iDataCount]=ucTWIReadByte(0);       //ostatnia dana, +NACK

}
```

Zajmiemy się teraz modyfikacją funkcji stworzonych w programie pierwszych. Do plików nagłówkowych możemy dodać plik zawierający makra TWI.

(3)

```
#include <util/twi.h>
```

Jest to jedna z bibliotek *WinAVR*, zawierająca makra funkcji i adresów.

[9] Czas oczekiwania po sekwencji stop jest różny dla układów różnych producentów. Można tu zastosować opóźnienie nawet 5ms, ale my stosujemy bezpieczne ograniczenie 15ms.

Dalsze modyfikacje będą dotyczyły obsługi błędów transmisji. Kierować będziemy się wcześniej przytoczoną dokumentacją procesora *ATmega16* i interfejsu *TWI*. Na pierwszy rzut pójdzie sprawdzanie poprawności wysłania sekwencji start.

(4)

```
//Funkcja inicjuje transmisje w trybie Master
void vTWIStart()
{
    TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN); //sekwencja start
    while(!(TWCR&(1<<TWINT)));           //oczekiwanie na flage

    //0x08 - TW_START
    //0x10 - TW_REP_START
    if (((TWSR & 0xF8)!=0x08)&&((TWSR & 0xF8)!=0x10))
        LCD_putstr_P(PSTR("ERROR START!"));
}
```

Zmiany dotyczą sprawdzanie rejestru *TWSR* i jego zawartości. Operacja bitowa & powoduje maskowanie bitów tego rejestru [10]. Kod *0x08* odpowiada wysłaniu sekwencji start, zaś kod *0x10* wysłaniu powtórnego startu. Jeśli w rejestrze znajdują się takie wartości tzn., że sekwencja start została pomyślnie wysłana na magistralę. Należy nadmienić, że kody liczbowe używane przy obsłudze magistrali mogą być zamienione na makra określone w pliku *twi.h*. Np. dla kodu *0x08* makro to *TW_START*. Kolejnym etapem będzie zmodyfikowanie funkcji v*TWIStop()*.

(5)

```
//Funkcja konczaca transmisje
void vTWIStop()
{
    TWCR=(1<<TWINT)|(1<<TWEN)|(1<<TWSTO); //wyslanie sekwenci stop
    while((TWCR&(1<<TWSTO)));            //oczekiwanie na bitu stop

    //0xF8 TW_NO_INFO
    if (TWSR!=0xF8)
        LCD_putstr_P(PSTR("ERROR STOP!"));
}
```

W tym wypadku oczekiwana wartość szesnastkowa to *0xF8*. Jeśli zawartość rejestru *TWSR* będzie różnić się od oczekiwanej, wtedy wyświetlamy komunikat. Poniżej znajduje się kolejna funkcja, mianowicie zmieniona funkcja transmisji pojedynczego bajtu danych.

[10] Wartość *0xF8* odpowiada wartości binarnej *11111000*. Należy przypomnieć, że pięć najstarszych bitów tego rejestru odpowiada za generowanie odpowiednich kodów statusu interfejsu. Bity 2, 1 są zarezerwowane, zaś najmłodszy bit służy do ustawienia preskalera (w naszym wypadku preskaler równy zero).

(6)

```
//Funkcja wysyla dane na magistrale
void vTWIWriteByte(unsigned char ucData)
{
    TWDR = ucData;                    //wyslanie danych
    TWCR = (1<<TWINT) | (1<<TWEN);
    while (!(TWCR & (1<<TWINT))); //oczekiwanie na zakonczenie akcji

    //0x18 TW_MT_SLA_ACK
    //0x28 TW_MT_DATA_ACK
    //0x40 TW_MR_SLA_ACK - tryb Master Receiver
    if (((TWSR & 0xF8)!=0x18)&&((TWSR & 0xF8)!=0x28)&&((TWSR & 0xF8)!=0x40))
        LCD_putstr_P(PSTR("ERROR WRITE!"));
}
```

Kod *0x18* odpowiada wysłaniu adresu i odebraniu od *Slave*, bitu potwierdzającego *ACK*. Kod *TW_MT_DATA_ACK* odpowiada wysłaniu bajtu danych i otrzymaniu bitu *ACK*. Zaś *0x40* odpowiada wysłaniu *SLA+R* i odebraniu *NACK*.

(7)

```
//Funkcja odbiera dane z magistrali I2C
unsigned char ucTWIReadByte(unsigned char ucAck)
{
    TWCR = (1<<TWINT) | (ucAck<<TWEA) | (1<<TWEN);  //ustawienie rejestru TWCR
    while (!(TWCR & (1<<TWINT)));                    //oczekiwanie na flage

    //0x50 TW_MR_DATA_ACK
    //0x58 TW_MR_DATA_NACK
    if (((TWSR & 0xF8)!=0x50)&&((TWSR & 0xF8)!=0x58))
        LCD_putstr_P(PSTR("ERROR READ!"));

    return TWDR;                                     //zwrocenie wartosci
}
```

Powyższa funkcja pozwala odebrać bajt danych. Rejestr *TWSR* jest sprawdzany pod kątem wysłania bitu *ACK* lub *NACK* w trybie *Master Receiver*.

Przypominam, że w celu używania makr zamiast kodów liczbowych należy dołączyć do plików nagłówkowych plik *twi.h*.

W stosunku do programu pierwszego wyglądu nie zmieniły funkcje v*EEPROMWriteData()* oraz *ucEEPROMReadData()*.

Wszystkie potrzebne funkcje są już utworzone, należy teraz je wykorzystać.

W programie wykorzystywane są *4* przyciski: *up, down, next, save*. Założono, że użytkownik będzie miał cztery dane *A, B, C, D*, i ich wartości będzie mógł zmieniać. Do tego celu będą służyć **przyciski up** i **down**. Z kolei **przycisk next** będzie powodował przejście do kolejnej danej do zmiany. Koniec zmian będzie potwierdzany **przyciskiem save**. Zmiany zostaną zapisane na trwałe w pamięci zewnętrznej *EEPROM*.

(8)

```
//Funkcja wyswietla dane w postaci
//np. A=1 B=2 C=4 D=5
void vShowDataOnLCD(unsigned char aucData[])
{
    LCD_clear();
    LCD_putstr_P(PSTR("A="));
    LCD_putint(aucData[1], 10);
    LCD_putstr_P(PSTR(" B="));
    LCD_putint(aucData[2], 10);
    LCD_putstr_P(PSTR(" C="));
    LCD_putint(aucData[3], 10);
    LCD_putstr_P(PSTR(" D="));
    LCD_putint(aucData[4], 10);
}

//odczytujemy pamiec i pokazujemy
void vReadFromMemory(unsigned char aucData[])
{
    vEEPROMReadPage(1, aucData, 4);
    vShowDataOnLCD(aucData);
}
```

Procedura *vShowDataOnLCD()* wyświetla na wyświetlaczu dane przekazane w wywołaniu. Opiera się na obsłudze wyświetlacza za pomocą bibliotek (*rkavrlib*). Z kolei funkcja *vReadFromemory()* ma za zadanie odczytanie czterech komórek pamięci *EEPROM* od adresu *1*, a następnie wyświetlenie danych za pomocą *vShowDataOnLCD ()*.

Można przejść już do opisu pętli głównej programu. Najpierw ułatwmy sobie życie definiując makra.

(9)

```
#define PRESS PIND
#define up      0
#define down    1
#define next    2
#define save    3

unsigned char  ucData[8];                    //dane zapisywane
```

Makra te odnoszą się do portu *D* mikroprocesora oraz przycisków podłączonych do jego wyprowadzeń (kolejno *PIND0...PIND3*). Inicjujemy również tablicę globalną *ucData*. Poniżej znajduje się początek funkcji głównej programu.

(10)

```
//Glowna funkcja programu
int main(void)
{
    int iPosition;

    DDRD=0x00;                          //ustawienie jako port we
    PORTD=0xff;                         //podciagniecie
    LCD_init();                         //inicjacja wyswietlacza
    LCD_clear();                        //wyczyszczenie wyswietlacza
    vReadFromMemory(ucData);            //odczyt ustawien poczatkowych
    iPosition=1;                        //ustawienie pozycji aktualnej
```

Jak widać inicjuje się tutaj zmienne lokalne, ustawienia portów, obsługę wyświetlacza *LCD*. Na starcie odczytujemy również wstępną zawartość pamięci *EEPROM*.

W pętli głównej programu zawarto obsługę czterech przycisków. Założono, że aby wyeliminować drgania styków *30ms* jest wartością wystarczającą. Kolejnym poczynionym założeniem są zmiany wartości poszczególnych zmiennych (*A, B, C, D*). Założono, że liczby te będą przyjmować wartości od *0* do *9*. Wynika to z faktu możliwości wypisania danych na wyświetlaczu *LCD 2x16.* Oczywiście można regulować te liczby w zakresie 0.. 255 (jednego bajtu). Poniżej zaprezentowano dalszą część funkcji *main()*.

(11)

```
for(;;)
{
    if (bit_is_clear(PRESS, up))                //sprawdzamy przycisk UP
    {
        delayms(30);                            //30ms zwloki
        if (bit_is_clear(PRESS, up))
        {
            if (ucData[iPosition]<9)
                ucData[iPosition]++;            //zwiekszamy wartosc
            vShowDataOnLCD(ucData);             //wyswietlamy zmienione dane
            delayms(60);
        }
    }

    if (bit_is_clear(PRESS, down))              //sprawdzamy przycisk DOWN
    {
        delayms(30);                            //30ms zlowki - drgania stykow
        if (bit_is_clear(PRESS, down))          //sprawdzamy ponownie
        {
            if (ucData[iPosition]>0) ucData[iPosition]--;
            vShowDataOnLCD(ucData);
            delayms(60);
        }
    }

    if (bit_is_clear(PRESS, next))              //sprawdzamy przycisk NEXT
    {
        delayms(30);                            //eliminacja drgan stykow
        if (bit_is_clear(PRESS, next))          //sprawdzamy ponownie
        {
            if (iPosition<4) iPosition++; else iPosition=1;
            delayms(200);
        }
    }
```

(12)

```
            if (bit_is_clear(PRESS, save))              //sprawdzamy przycisk SAVE
            {
                delayms(30);                            //30ms przerwy
                if (bit_is_clear(PRESS, save))          //sprawdzamy ponownie
                {
                    vEEPROMWritePage(1, ucData, 4);     //zapisujemy do pamieci
                    LCD_xy(0,1);                        //wyswietlamy info
                    LCD_putstr_P(PSTR("Zapisano"));
                    delayms(300);                       //czekamy 300ms
                    vShowDataOnLCD(ucData);             //pokazujemy dane
                }
            }

        }
    }
```

Używając wewnętrznych makr *bit_is_clear()* lub *bit_is_set()* możemy łatwo sprawdzić stan portu mikroprocesora. Analiza zamieszczonego kodu nie powinna sprawić problemu.

5. Podsumowanie

Dzięki wspieraniu standardu **I²C** (**TWI**) przez firmę **Atmel** mamy możliwość korzystania z układów mikroprocesorowych posiadających wbudowaną obsługę interfejsu *I²C*. Znacznie ułatwia to tworzenie programów i jest bardzo pomocne w przypadku analizy błędów transmisji. Dzięki wbudowanej kontroli, na jednej szynie magistrali może znajdować się nawet kilka układów *Master*.

W opracowaniu tym korzystano bezpośrednio z interfejsu **TWI** jaki oferował mikrokontroler **ATmega16**. Można by pokusić się o stworzenie implementacji *I²C* od podstaw, jednak zapewne nie udałoby się stworzyć tak obszernego i bezpiecznego interfejsu jaki oferuje *Atmel*.

Autor ma nadzieje, że w sposób prosty i jasny wytłumaczył sposób i zasadę działania interfejsu *TWI* i że programy umieszczone w tymże opracowaniu będą pomocne.

6. Bibliografia

[1] Gręboszj J.: *Symfonia C++.* Kraków 1999, Kallimach

[2] Opracowanie własne – *Programowanie mikrokontrolerów AVR w języku C.* Kraków 2007

[3] Atmel : *ATmega16 datasheet*. Atmel 2007

[4] Atmel: *2-Wire Serial EEPROM. AT24C164*

[5] Fairchild Semiconductor: *NM24C16/17*, Luty 2000

[6] STMicroelectronics: *ST24C16*, Luty 1999

[7] *http://radzio.dxp.pl*

[8] *http://wikipedia.org*