Magistrala I^2C

Charakterystyka ogólna

Na szynę I^2C składają się tylko dwie linie: linia danych (oznaczana jako SDA, ang. serial data) i linia zegara taktującego transmisję (oznaczana jako SCL, ang. serial clock). Tymi dwiema liniami przesyłane są wszystkie informacje: dane, adresy i sygnały sterujące. Obie linie szyny I^2C - SDA i SCL - są liniami dwukierunkowymi i przez rezystor podciągający podłączone są do dodatniego napięcia zasilającego. Gdy na szynie nie odbywa się żadna transmisja to na obu liniach jest poziom wysoki. Obwody wyjściowe podłączonych układów muszą być typu otwarty-dren lub otwarty-kolektor, ze względu na konieczność realizacji funkcji galwaniczne-AND. Dane na szynie I^2C mogą być przesyłane z prędkością do 100kb/s w trybie standardowym (ang. Standard-mode) lub do 400kb/s w trybie szybkim (ang. Fast-mode). Liczba układów podłączonych do szyny jest zależna od maksymalnej pojemności szyny wynoszącej 400pF.

Każdy układ podłączony do szyny I^2C rozpoznawany jest przez swój adres i może pracować zarówno jako nadajnik, jak i odbiornik. Oczywiście sterownik wyświetlacza LCD jest tylko odbiornikiem, ale np. pamięć może być jednym i drugim. Dodatkowo podczas transmisji podłączone układy dzielą się na master i slave. Master jest to układ inicjujący transmisję i generujący sygnał zegarowy umożliwiający tą transmisję. W tym momencie każdy zaadresowany układ jest układem slave.

Transmisja na szynie I^2C jest transmisją start-stopową, tzn. każdy transfer rozpoczyna się od bitu START, po którym przesyłane są informacje (w przeciwieństwie jednak do interfejsu RS-232C ich ilość nie jest ograniczona), i kończy się bitem STOP.

Szyna I^2C jest szyną typu multi-master. To znaczy, że może być podłączonych wiele układów mogących kontrolować transmisję. Zwykle układami master są mikrokontrolery. Rozpatrzmy taką sytuację, gdy podłączone są dwa mikrokontrolery. Może to wyglądać np. tak:

1. Przypuśćmy, że mikrokontroler A chce wysłać informację do mikrokontrolera B:

-mikrokontroler A (master) adresuje mikrokontroler B (slave)

- mikrokontroler A (nadajnik-master) wysyła dane do mikrokontrolera B (odbiornik-slave)

-mikrokontroler A kończy transmisję.

2. Jeśli mikrokontroler A chce odebrać informacje od mikrokontrolera B:

-mikrokontroler A (master) adresuje mikrokontroler B (slave)

- mikrokontroler A (odbiornik-master) odbiera dane od mikrokontrolera B (nadajnik-slave)

-mikrokontroler A kończy transmisję.

W obu przypadkach master (mikrokontroler A) generuje przebieg zegarowy oraz kończy transmisję.

Możliwość podłączenia kilku mikrokontrolerów do szyny I^2C powoduje, że może zaistnieć sytuacja, w której kilka układów master chce rozpocząć transmisję w tej samej chwili. Aby uniknąć zgubienia danych stosowana jest specjalna procedura, pozwalająca wyłonić, który master będzie nadawał. Procedura ta oparta jest na funkcji galwaniczne-AND, która to wynika ze sposobu

podłączenia wszystkich układów do szyny I^2C.

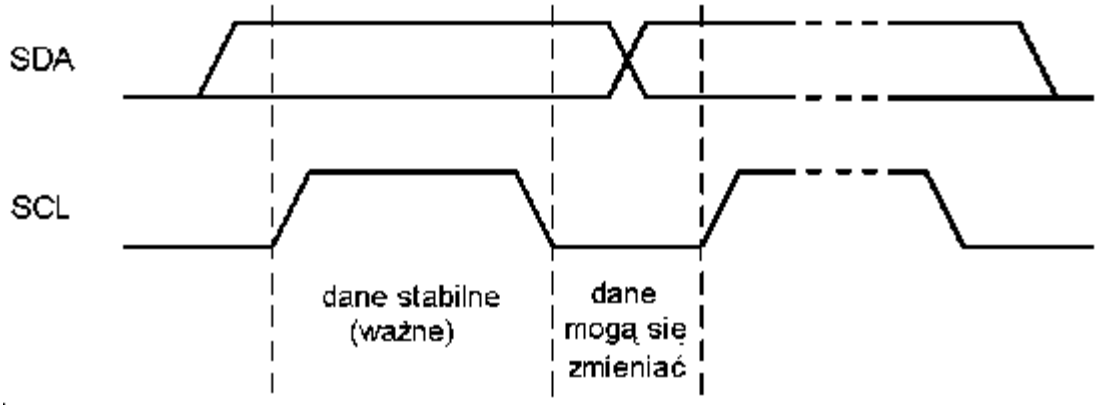
Jeżeli dwa lub więcej układów master rozpoczyna transmisję, to układ, który pierwszy wygeneruje bit o poziomie logicznym '1', gdy inne układy generują '0', traci arbitraż. Sygnał zegarowy na linii SCL podczas procedury arbitracyjnej jest funkcją galwaniczne-AND sygnałów zegarowych generowanych przez układy master, chcące zainicjować transmisję.

Generowanie sygnału zegarowego na szynie I^2C jest zawsze zadaniem układu typu master. Każdy master generuje swój własny sygnał zegarowy podczas transmisji danych. Sygnał zegarowy z układu master może być zmieniony tylko, wtedy gdy zbyt wolny układ slave wymusi poziom niski lub podczas procedury arbitracyjnej.

Transfer - na poziomie bitów

Ze względu na możliwość podłączenia układów wykonanych w różnych technologiach (NMOS, CMOS, bipolarne), napięcia dla poziomów logicznych '0' i '1' nie są zdefiniowane i zależą od odpowiedniego poziomu napięcia zasilającego Vdd. Napięcia poniżej poziomu 0,3Vdd rozpoznawane są jako poziom niski, zaś napięcia powyżej poziomu 0,7Vdd - jako poziom wysoki (dla Vdd=5V: VLmax=1,5V VHmin=3V) Dla każdego przesyłanego bitu generowany jest jeden impuls zegarowy.

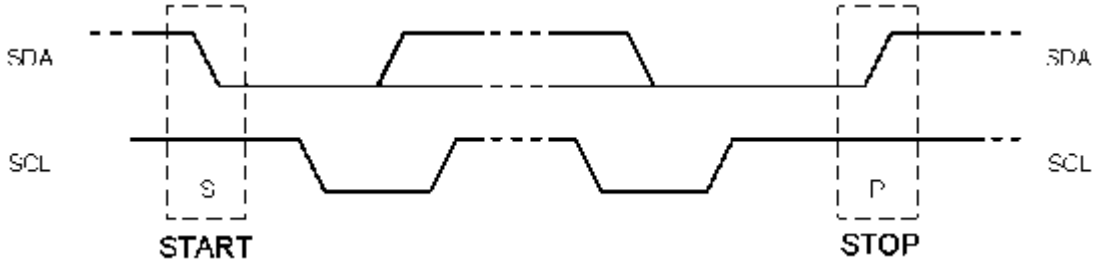
Dane na linii SDA muszą być stabilne podczas wysokiego poziomu sygnału zegarowego. Stan na linii danych może się zmieniać tylko podczas niskiego stanu sygnału zegarowego (na linii SCL).



START i STOP

Najważniejszymi sytuacjami podczas transmisji na szynie I^2C są bity START i STOP.

Gdy na linii SDA stan logiczny zmienia się z '1' na '0', podczas gdy na linii SCL jest stan '1', to sytuacja taka nazywa się START. Natomiast gdy na linii SDA stan logiczny zmienia się z '0' na '1', podczas wysokiego poziomu na linii SCL, to sytuacja taka nazywa się STOP.



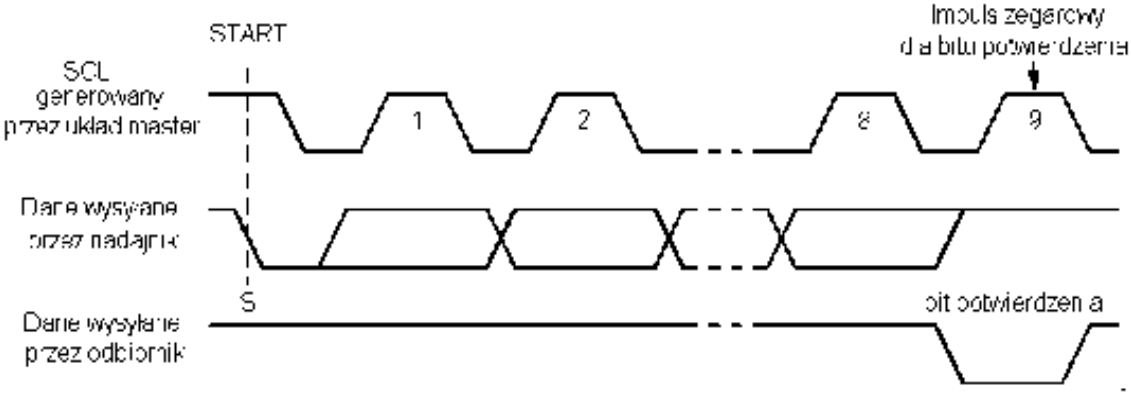
START i STOP generowane są zawsze przez układ master. Po wygenerowaniu START'u szyna jest zajęta, natomiast po wygenerowaniu STOP'u szyna jest wolna. Wykrywanie START'u i STOP'u jest łatwe, gdy podłączone urządzenia zawierają odpowiedni interfejs sprzętowy. Mikrokontrolery bez tego interfejsu, w celu wykrycia transmisji, muszą próbkować linię SDA co najmniej dwa razy w ciągu cyklu zegarowego.

Transfer danych

Informacje wysyłane na linię SDA muszą być 8-bitowe. Liczba bajtów, które mogą być przesłane nie jest limitowana. Każdy bajt musi zostać potwierdzony bitem potwierdzenia. Podczas transmisji danych pierwszy wysyłany jest bit najstarszy. Jeżeli odbiornik nie może odebrać następnego kompletnego bajtu danych, ze względu na konieczność wykonania jakichś innych funkcji, to może wymusić poziom '0' na linii SCL, w tym momencie nadajnik przejdzie w stan oczekiwania. Transmisja będzie kontynuowana, gdy odbiornik zwolni linię SCL.

Potwierdzenie

Transmisja z potwierdzeniem jest jedyną możliwą na szynie I^2C. Impuls zegarowy dla bitu potwierdzenia jest generowany, tak jak zawsze, przez układ master. Na czas impulsu zegarowego dla bitu potwierdzenia nadajnik zwalnia linię SDA (poziom wysoki). Odbiornik musi wymusić niski poziom na linii SDA podczas tego impulsu zegarowego, tak aby poziom niski był stabilny podczas wysokiego poziomu na linii SCL.



Zwykle zaadresowany odbiornik musi wygenerować bit potwierdzenia, po każdym odebranym bajcie (jest jeden wyjątek, dla zachowania kompatybilności z szyną CBUS). Gdy odbiornik-slave nie może potwierdzić swojego zaadresowania (na przykład nie może odbierać, ponieważ wykonuje jakieś funkcje w czasie rzeczywistym), to musi pozostawić linię SDA w stanie wysokim. Master może wtedy wygenerować STOP, aby przerwać transmisję.

Gdy odbiornik-slave potwierdzi swoje zaadresowanie, ale potem podczas transmisji nie będzie mógł odbierać kolejnych danych, to master znowu będzie musiał przerwać transmisję. Sytuacja tak jest rozpoznawana, gdy slave nie wygeneruje bitu potwierdzenia po pierwszym kolejnym bajcie (zostawi linię SDA w stanie wysokim). Master wygeneruje wtedy sygnał STOP.

Gdy odbiornikiem jest układ master, to sygnalizuje on koniec danych nadajnikowi-slave nie generując bitu potwierdzenia po ostatnim bajcie wysłanym przez układ slave. Nadajnik-slave musi wtedy zwolnić linię danych, aby umożliwić układowi master wygenerowanie sygnału STOP.

Arbitraż i synchronizacja

-Synchronizacja

Każdy master generuje własny sygnał zegarowy na linii SCL, gdy chce przesłać dane szyną I^2C. Dane ważne są tylko podczas wysokiego poziomu na linii SCL. Synchronizacja zegarów

następuje przy pomocy połączenia galwaniczne-AND wszystkich układów do linii SCL. Zsynchronizowany sygnał zegarowy generowany jest z poziomem niskim określonym przez układ, który najdłużej utrzymał poziom '0' i z poziomem wysokim określonym przez układ, który najkrócej utrzymał poziom '1'.

-Arbitraż

Układ master może zacząć transmisję tylko wtedy gdy szyna jest wolna. Może się jednak zdarzyć, że dwa lub więcej układy master wygenerują sygnał START w tym samym czasie. Rozpocznie się wtedy procedura arbitracyjna. Arbitraż ma miejsce podczas wysokiego poziomu na linii SCL. W takim przypadku master, który nadaje poziom wysoki na linii SDA, podczas gdy drugi master nadaje poziom niski, przestanie nadawać, ponieważ stan na linii SDA nie będzie odpowiadał stanowi przez niego wygenerowanemu. Arbitraż może trwać przez wiele bitów. Jego pierwszym etapem jest porównywanie bitów adresowych. Jeżeli układy master starają się zaadresować te samo urządzenie, arbitraż przeciąga się na bity danych. Ponieważ adres i dane używane są do arbitracji, żadna informacja nie jest tracona podczas tego procesu. Master, który stracił arbitraż może generować sygnał zegarowy do końca bajtu, w którym stracił arbitraż.

Jeśli master pełni również funkcję slave i stracił arbitraż podczas etapu adresowania to jest możliwe, że wygrywający master próbuje go zaadresować. Master, który przegrał arbitraż musi czym prędzej przełączyć się w tryb slave.

Oczywiście w arbitrażu może uczestniczyć więcej układów master (zależy ile jest podłączonych).

Kontrola szyny I^2C zależy od wysłanego, przez konkurujące układy master, adresu i danych, tak więc zbędny jest jakikolwiek master centralny lub kolejność priorytetów.

Arbitraż nie może mieć miejsca między:

- powtórzonym sygnałem START i bitem danych,

- sygnałem STOP i bitem danych,

- powtórzonym sygnałem START i sygnałem STOP.

Adresowanie 7-bitowe

Po bicie START wysyłany jest 7-bitowy adres, uzupełniony ósmym bitem R/W określającym kierunek transmisji - '0' oznacza zapis, '1' oznacza odczyt. Transmisja danych zawsze kończy się bitem STOP generowanym przez układ master. Jeżeli master nadal chce komunikować się po szynie, to może wygenerować powtórzony sygnał START i zaadresować następny układ slave bez generowania bitu STOP.

Możliwe są więc następujące formaty wysyłanych danych:

-Nadajnik-master nadaje do odbiornika-slave. Kierunek transmisji jest niezmienny.

- Master czyta dane od slave natychmiast po pierwszym bajcie. Po pierwszym potwierdzeniu nadajnik-master staje się odbiornikiem-master, a odbiornik-slave staje się nadajnikiem-slave. Potwierdzenie to generowane jest jeszcze przez układ slave. Bit STOP generowany jest przez układ master.

-Format mieszany. Podczas zmiany kierunku nadawania, powtarzany jest bit START i adres układu slave, ale ze zmienionym bitem R/W.

Uwagi.

1. Format mieszany może być używany np. do sterowania pamięcią szeregową. Pierwszy bajt danych określa wewnętrzny adres w pamięci. Po powtórzonym bicie START i adresie układu mogą być przesłane dane.

2. To czy adres wewnątrz pamięci będzie automatycznie zwiększany, czy zmniejszany zależy od projektanta danego układu.

3. Po każdym bajcie następuje bit potwierdzenia

Rozszerzenia specyfikacji szyny I^2C

Szyna I^2C z szybkością przesyłu 100kb/s i adresowaniem 7-bitowym istnieje już od około 15 lat. Kilka lat temu wprowadzono do niej pewne unowocześnienia:

- tryb szybki (ang. Fast-mode) - z prędkością przesyłu do 400kb/s

-adresowanie 10-bitowe - pozwalające zaadresować do 1024 urządzeń.

TWI Control Register – TWCR

Rejestr sterujący układu TWI. W rejestrze tym zawarte są następujące bity :

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	

Bit 7 – TWINT: TWI Interrupt Flag
Bit ten jest ustawiany sprzętowo po zakończeniu wykonywania przez układ TWI bieżącej operacji. Jeśli bit I w rejestrze statusu oraz bit TWIE w rejestrze TWCR jest ustawiony, to nastąpi wywołanie procedury obsługi przerwania od układi TWI. Bit TWINT jest zerowany programowo poprzez zapis jedynki. Należy pamiętać, że bit nie jest zerowany automatycznie po wywołaniu procedury obsługi przerwania. Należy również pamiętać, że wyzerowanie tego bitu rozpoczyna wykonywanie operacji przez układ TWI, tak więc wszelkie operacje na zawartości rejestrów TWAR, TWSR oraz TWDR muszą zostać zakończone przez wyzerowaniem bitu TWINT

Bit 6 – TWEA: TWI Enable Acknowledge Bit
Bit steruje generowaniem sygnału ACK. Jeśli bit TWEA jest ustawiony to sygnał ACK jest generowany w następujących okolicznościach :
- układ odebrał własny adres Slave
- odebrano sygnał globalnego wywołania przy ustawionym bicie TWGCE w rejestrze TWAR
- odebrano dane w trybie Master Receiver lub Slave Receiver

Bit 5 – TWSTA: TWI START Condition Bit
Bit ten jest ustawiany w celu rozpoczęcia transmisji w trybie Master.Układ TWI sprawdza czy

magistrala jest wolna i generuje sygnał START. Jeśli magistrala jest zajęta, układ TWI oczekuje na sygnał STOP i wystawia sygnał START w celu uzyskania statusu MASTER na magistali. Bit TWSTA musi zostać wyzerowany programowo po wygenerowaniu sygnału START.

Bit 4 – TWSTO: TWI STOP Condition Bit
Ustawienie bitu TWSTO powoduje wygenerowanie w trybie Master sygnału STOP na magistrali. Bit ten jest automatycznie zerowany po wygenerowaniu sygnału STOP.

Bit 3 – TWWC: TWI Write Collision Flag
Bit jest ustawiany przy próbie zapisu do rejestru TWDR przy wyzerowanym bicie TWINT. Bit jest zerowany po zapisie do rejestru TWDR przy ustawionym bicie TWINT.

Bit 2 – TWEN: TWI Enable Bit
Bit aktywujący interfejs TWI. Gdy bit TWEN jest ustawiony układ TWI przejmuje kontrolę nad wyprowadzeniami mikrokontrolera podłączonymi do linii SDA i SCL. Gdy bit TWEN jest wyzerowany, układ TWI jest wyłączany oraz wszelkie trwające operacje są przerywane.

Bit 1 – Res: Reserved Bit
Ten bit jest zarezerwowany.

Bit 0 – TWIE: TWI Interrupt Enable
Jeśli bit TWIE jest ustawiony, oraz bit I w rejestrze statusowym jest ustawiony, to przerwanie od układu TWI będzie aktywne tak długo, jak bit TWINT będzie ustawiony.

i2cmaster.h

```
#ifndef _I2CMASTER_H_
#define _I2CMASTER_H_
#endif

#include <avr/io.h>

#define DS1337ADDR  0xD0      //adres urzadzenia dla A0=gnd, dla vcc bedzie 0xA2;

#define I2C_READ    1
#define I2C_WRITE   0

#define sbi(PORT,PIN) ((PORT)|=1<<(PIN))
#define cbi(PORT,PIN) ((PORT)&=~(1<<(PIN)))

void czekaj(unsigned int useconds);

void i2c_init(void);            //inicjalizacja i2c

void i2c_stop(void);            //warunek stop, oddanie kontroli po transmisji

unsigned char i2c_start(unsigned char address);          //warunek startu, poczatek transmisji
                                                         //podaje adres i R/W, 0- ok, 1- blad

unsigned char i2c_rep_start(unsigned char address);      // repeated start, j/w

void i2c_start_wait(unsigned char address);              // j/w, ale czeka az urzadzenie bedzie gotowe

unsigned char i2c_write(unsigned char data);             //wysyla 1 bajt, 0-ok, 1-blad

unsigned char i2c_readAck(void);                         //odczyt z ACK (czyli zadanie kolejnych bajtow)

unsigned char i2c_readNak(void);                         //czytanie z NACK, czyli po odczycie warunek stopu
```

i2cmaster.c

```
//Plik zawiera funkcje wykorzystywane w i2c dla trybu nadrzędnego (master)

#include <compat/twi.h>
#include "i2cmaster.h"
#include <util/delay.h>
#include "master.c"

#ifndef F_CPU
#define F_CPU 1000000UL
#endif
#define SCL_CLOCK  20000L // szybkosc i2c
#define sbi(PORT,PIN) ((PORT)|=1<<(PIN))
#define cbi(PORT,PIN) ((PORT)&=~(1<<(PIN)))

void i2c_init(void) //inicjalizacja sprzetowego i2c
{
		TWSR = 0;
		TWBR = ((F_CPU/SCL_CLOCK)-16)/2;  //na zadana czestotliwosc
		PORTC|=((1<<PC4)|(1<<PC5));		// wlaczenie podciagania dla i2c
}

unsigned char i2c_start(unsigned char address)
{
		unsigned char twst;		//informacja o statusie

		// wysyla warunek startu
		TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

		// czeka na koniec transmisji
		while(!(TWCR & (1<<TWINT)));

		// sprawdza czy nastapil blad
		twst = TW_STATUS & 0xF8;
		if ( (twst != TW_START) && (twst != TW_REP_START)) return 1;

		// wysyla adres
		TWDR = address;
		TWCR = (1<<TWINT) | (1<<TWEN);

		// czeka na koniec i ACK lub NACK
		while(!(TWCR & (1<<TWINT)));

		// kolejne sprawdzanie statusu
		twst = TW_STATUS & 0xF8;
		if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;

		return 0;
}

void i2c_start_wait(unsigned char address)
{
   unsigned char   twst;   //informacja o statusie

   while (1)
   {
		// wysyla warunek startu
		TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

		// czeka na koniec transmisji
		while(!(TWCR & (1<<TWINT)));
```

```
                // sprawdzanie statusu
                twst = TW_STATUS & 0xF8;
                if ( (twst != TW_START) && (twst != TW_REP_START)) continue;

                // wysylanie adresu
                TWDR = address;
                TWCR = (1<<TWINT) | (1<<TWEN);

                // czekanie na koniec transmisji
                while(!(TWCR & (1<<TWINT)));

                // sprawdzenie statusu
                twst = TW_STATUS & 0xF8;
                if ( (twst == TW_MT_SLA_NACK )||(twst ==TW_MR_DATA_NACK) )
                {
                    //urzadzenie na i2c zajete, wysyla STOP aby zakonczyc transmisje
                    TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

                    // czeka na warunek stopu
                    while(TWCR & (1<<TWSTO));

                   continue;
                }
                //if( twst != TW_MT_SLA_ACK) return 1;
                break;
    }
}


unsigned char i2c_rep_start(unsigned char address)        // repeated start
{
   return i2c_start(address);
}

void i2c_stop(void)
{               // warunek stopu
                TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
                // oczekiwanie na zakonczenie
                while(TWCR & (1<<TWSTO));
}


unsigned char i2c_write(unsigned char data)
{
                unsigned char twst;             //informacja o statusie

                // wysyla dane do urzadzenia, do ktorego wydano warunek startu
                TWDR = data;
                TWCR = (1<<TWINT) | (1<<TWEN);

                // oczekiwanie na zakonczenie transmisji
                while(!(TWCR & (1<<TWINT)));

                // sprawdzenie statusu
                twst = TW_STATUS & 0xF8;
                if( twst != TW_MT_DATA_ACK) return 1;
                return 0;
}

unsigned char i2c_readAck(void)                         // odczytanie pojedynczego bajtu
{
                TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
                while(!(TWCR & (1<<TWINT)));
```

```
   return TWDR;
}

unsigned char i2c_readNak(void)                    // odczyt bez ACK, czyli pojedynczy bajt
{
                TWCR = (1<<TWINT) | (1<<TWEN);
                while(!(TWCR & (1<<TWINT)));

   return TWDR;
}

void czekaj(unsigned int useconds)                 //opóźnienie w mikrosekundach
{
  int i,j;
  for (j=0;j<useconds;j++)
    for (i=0;i<26;i++) asm("nop");
}
```

master.c

```
#include <compat/twi.h>
#include "i2cmaster.h"
#include <util/delay.h>
#include "i2cmaster.c"

uint8_t value;

int main(void)
{
        sbi(DDRB,PB1);       //informacja o poprawnie odebranych danych
        cbi(PORTB,PB1);     //ustawienie w stan niski
        sbi(PORTC,PC4);     //Podciągnięcie linii SDA do 5V
        sbi(PORTC,PC5);     //Podciągnięcie linii SCL do 5V

        czekaj(1000);           //funkcja opóźniająca inicjalizację modułu TWI
        i2c_init();                     //inicjalizacja

        while(1)
        {
                i2c_start(DS1337ADDR+I2C_WRITE); //wysłanie sygnału start
                                                 //zawierającego adres urządzenia oraz
                                                 //inf. czy dane będą wysyłane czy odbierane
                i2c_write(0xf3+I2C_WRITE);       //wysłanie danych
                i2c_stop();                      //zakończenie transmisji

                i2c_start(DS1337ADDR+I2C_READ);
                value = i2c_readNak();           //odbiór danych
                i2c_stop();
                        asm volatile (
                                        "WDR"::);           //reset watchdoga
                        if(value == 0b10101010)
                        sbi(PORTB,PB1);                     //podciągnięcie portu na stan wysoki
                }
}
```

i2cslave.c

```
//Program wykorzystanie magistrali i2c w trybie podrzędnym
#include <avr/io.h>
#include <avr/interrupt.h>
#include <compat/twi.h>

#define sbi(PORT,PIN) ((PORT)|=1<<(PIN))
#define cbi(PORT,PIN) ((PORT)&=~(1<<(PIN)))

uint8_t value;                  //zmienna przechowująca dane
unsigned char twst;             //informacja o statusie

SIGNAL (SIG_2WIRE_SERIAL)
{
 switch(TWSR)
 {
                case 0x00:  //błąd (nieoczekiwany sygnał START lub STOP)
                sbi(TWCR,TWSTO);
                sbi(TWCR,TWINT);
                break;

                case 0x60:  //Odebrano własny SLA+W zwrócono ACK
                break;

                case 0x68:  //Utracono kontrolę nad magistralą, odebrano własny SLA+W zwrócono ACK
                break;

                case 0x80:  //Adresowanie SLA:odebrano słowo danych zwrócono ACK
                value = TWDR;
                PORTB = value;
                break;

                case 0x88:  //Adresowanie SLA:odebrano słowo danych zwrócono NACK
                value = TWDR;
                PORTB = value;
                break;

                case 0xA8:  //Odebrano własny SLA+R zwrócono ACK
                TWDR=0b10101010;

                // wysyla dane do urzadzenia, do ktorego wydano warunek startu
                TWCR = (1<<TWINT) | (1<<TWEN);

                // oczekiwanie na zakonczenie transmisji
                while(!(TWCR & (1<<TWINT)));

                // sprawdzenie statusu
                twst = TW_STATUS & 0xF8;
                if( twst != TW_MT_DATA_ACK)
                break;

                case 0xB8:  //Nadano słowo danych, odebrano ACk konieczne dalsze przesyłanie
                 TWDR=0b10101010;

                // wysyla dane do urzadzenia, do ktorego wydano warunek startu
                TWCR = (1<<TWINT) | (1<<TWEN);

                // oczekiwanie na zakonczenie transmisji
                while(!(TWCR & (1<<TWINT)));

                // sprawdzenie statusu
                twst = TW_STATUS & 0xF8;
```

```
                if( twst != TW_MT_DATA_ACK)
                break;

                case 0xC0:  //Nadano słowo danych, odebrano NACK konieczne przerwanie transmisji
                TWDR=0b10101010;

                // wysyla dane do urzadzenia, do ktorego wydano warunek startu
                TWCR = (1<<TWINT) | (1<<TWEN);

                // oczekiwanie na zakonczenie transmisji
                while(!(TWCR & (1<<TWINT)));

                // sprawdzenie statusu
                twst = TW_STATUS & 0xF8;
                if( twst != TW_MT_DATA_ACK)
                break;
 }
                sbi(TWCR,TWEA);         //włączenie generacji potwierdzeń ACK
                sbi(TWCR,TWINT);        //znacznik przerwania dla modułu TWI
                sbi(TWCR,TWEN);         //włączenie modułu TWI
                sbi(TWCR,TWIE);         //uaktywnienie przerwań dla modułu TWI
                asm volatile (          //reset watchdoga
                        "WDR"::);
}



int main()
{
  // twi
  DDRB=0xff;              //port z diodami na wyjście
  PORTB=0x00;             //ustawienie portu na stan niski
  cbi(PRR0,PRTWI);        //The Power Reduction TWI bit, PRTWI bit in
                          // “Power Reduction Register 0 - PRR0” on
                          //page 54 must be written to zero to enable
                          // the 2-wire Serial Interface. In ATmega2561
  sbi(TWCR,TWEA);         //włączenie generacji potwierdzeń ACK
  sbi(TWCR,TWINT);        //znacznik przerwania dla modułu TWI
  sbi(TWCR,TWEN);         //włączenie modułu TWI
  sbi(TWCR,TWIE);         //uaktywnienie przerwań dla modułu TWI

  TWAR = 0xD0;            //Adres urządzenia
  TWSR=0;                 //czyszczenie statusu TWI

  sei();                  //globalne zezwolenie na przerwania

  while(1){}
}
```