1 Magistrala I2C

1.1 Opis magistrali

Magistrala wprowadzona przez firmę Philips i przeznaczona do komunikacji wewnątrz urządzeń elektronicznych (*Integrated circuit to Integrated circuit*), angielska wymowa (ang. *I squared C*). Typowo stosowana do łączenia mikrokontrolera z takimi urządzeniami jak wyświetlacze, pamięci EEPROM, zegary RTC, przetworniki AD i DA, klawiatury. Jest to magistrala szeregowa.

Najważniejsze własności magistrali I^2C:

- Magistrala 2 przewodowa w skład której wchodzą sygnały: dane – SDA – (serial data line), zegar - SCL – (serial Clock)
- Każdy dołączony do magistrali układ ma unikalny adres. Adresowanie 7 bitowe (128 urządzeń) lub 10 bitowe (1024 urządzeń). Rodzaj transmisji master – slave
- Magistrala dopuszcza istnienie wielu układów typu Master. Stosowany jest arbitraż.
- Magistrala jest szeregowa, dwukierunkowa, zorientowana na transmisje 8 bitowe.
- Szybkości magistrali: standard mode - do 100kbit/s, fast-mode - do 400 kbit/s, high-speed mode - do 3.4 Mbit/s.
- Liczba dołączonych urządzeń ograniczona pojemnością – do 400 pF.
- Wbudowane układy zapobiegające przepięciom.
- Zasięg do kilku metrów

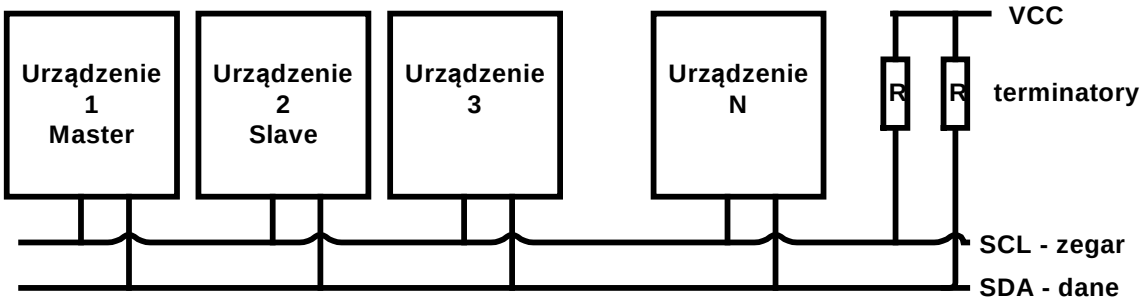
Inne własności:

- Małe zużycie mocy
- Odporność na zakłócenia
- Szeroki zakres napięć zasilania
- Szeroki zakres temperatur pracy

Definicje:

Termin	Znaczenie
Nadajnik	Wysyła dane na magistralę I2C
Odbiornik	Odbiera dane z magistrali I2C
Master	Inicjalizuje transmisje, generuje sygnał zegarowy i kończy transakcje
Slave	Urządzenie które jest adresowane przez master
Multi master	Więcej niż jeden master próbuje objąć magistralę
Arbitraż	Zapewnia że tylko jeden master obejmie kontrolę nad magistralą w sytuacji gdy więcej jednostek próbuje objąć magistralę. Zapewnia że komunikat pozostanie prawidłowy

Tabela 1-1 Definicje używane w technologii I2C

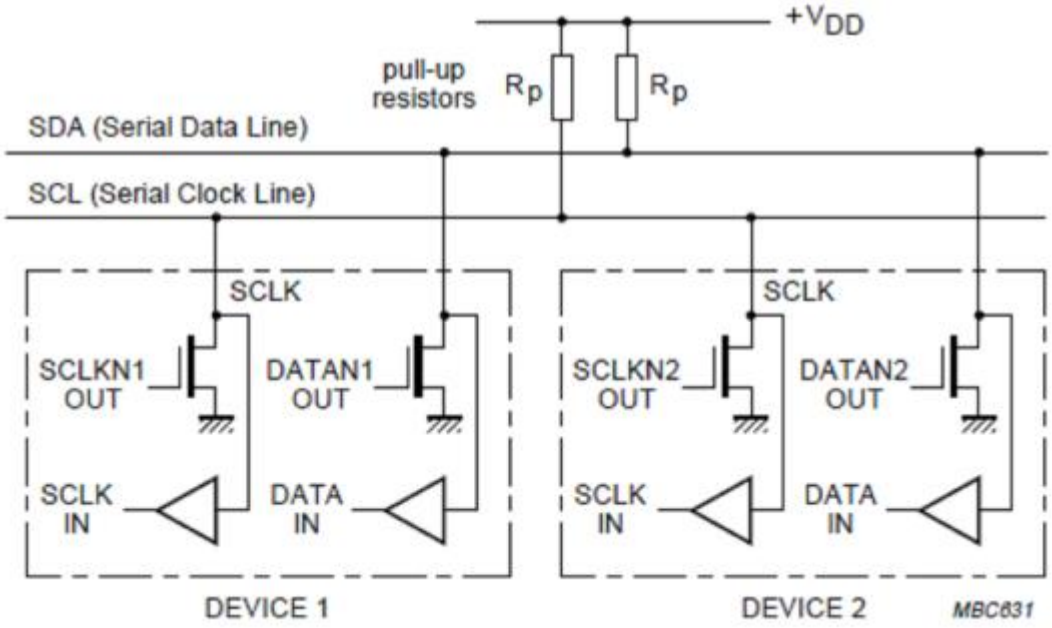


Rys. 1-1 Budowa magistrali i2c

1.2 Warstwa fizyczna

Standard I2C wykorzystuje dwie dwukierunkowe linie: danych SDA i zegara SCL.

Użyty jest standard logiki dodatniej a więc wysokiemu poziomowi napięcia odpowiada logiczna 1 a niskiemu logiczne 0. Magistrala podłączona jest do napięcia +5V poprzez rezystor. Linie te są typu otwarty kolektor ze słabą 1 i mocnym 0. Znaczy to że gdy jeden z układów wystawi jedynkę a drugi 0 na magistrali będzie 0.

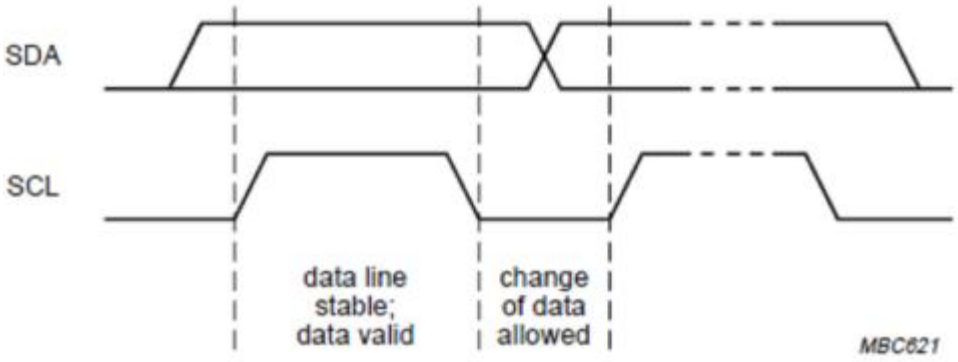


Rys. 1-1 Dołączenie nadajnika i odbiornika do magistrali

1.3 Warstwa łącza danych

Przesyłania danych

Zmiana danych może nastąpić gdy zegar SCL jest w poziome LOW. W czasie gdy zegar jest w stanie HIGH dane musza być stabilne.



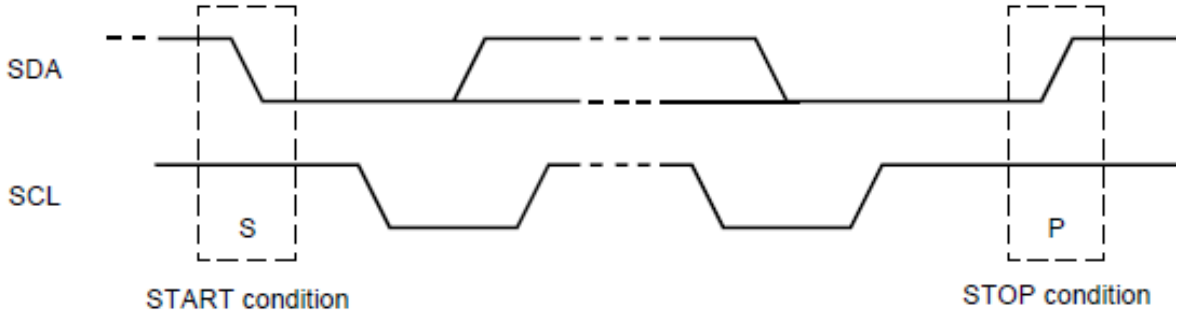
Rys. 1-2 Zmiana danych możliwa gdy sygnał zegara jest w stanie HIGH

Warunek startu i stopu

W czasie transmisji zdefiniowane są znaczniki startu i stopu transmisji.

Znacznik startu polega na tym że w czasie wysokiego poziomu zegara następuje zmiana linii sygnałowej z HIGH na LOW.

Znacznik stopu wystąpi gdy w czasie wysokiego poziomu zegara następuje zmiana linii sygnałowej SDA z LOW na HIGH.



Rys. 1-3 Warunek startu i stopu

Wykrywanie kolizji

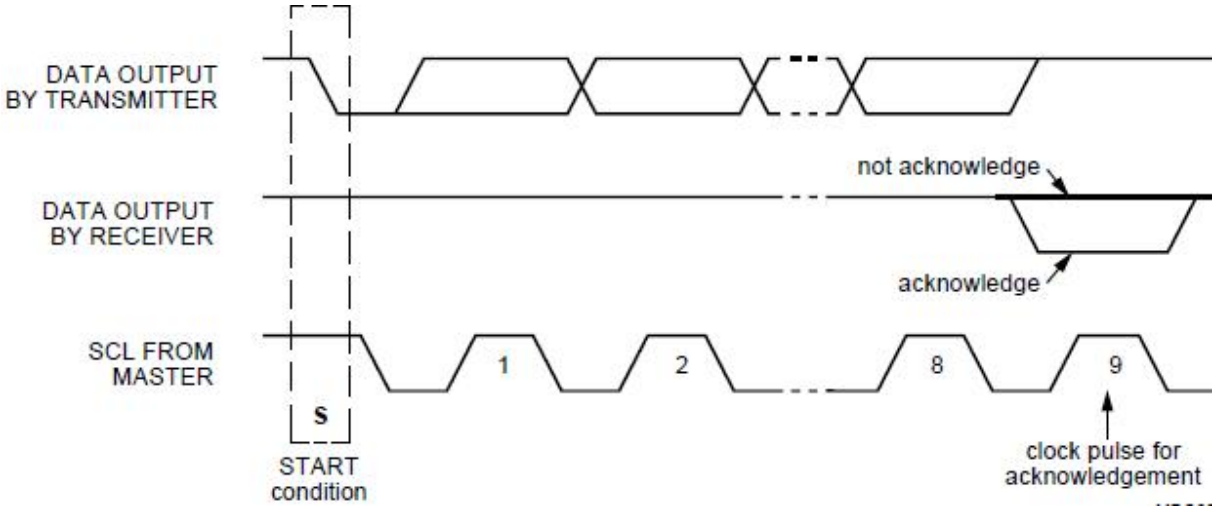
Magistrala pozwala na wykrywanie kolizji. Każde urządzenie wystawiające jedynkę jednocześnie sprawdza, czy na magistrali rzeczywiście pojawił się stan wysoki. Jeżeli tak nie, oznacza to, iż inne urządzenie nadaje w tym samym czasie zero i urządzenie zaprzestaje nadawania.

Format danych

Przesyłane są dane 8 bitowe począwszy od najstarszego do najmłodszego bitu. Liczba przesłanych bajtów nie jest ograniczona. Jeżeli SLAVE nie może odebrać bądź wysłać bajtów gdyż zajęty jest innymi czynnościami może zasygnalizować to stronie MASTER poprzez przytrzymanie linii SCL zegara w stanie LOW. Transmisja się wznowi gdy SLAVE zwolni sygnał SCL.

Potwierdzanie danych

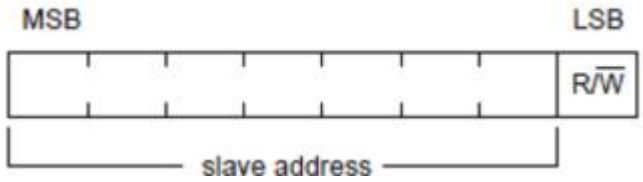
Każdy odebrany bajt musi być potwierdzony przez odbiornik. Odbywa się to poprzez wystawienie sygnału LOW w czasie gdy dziewiąty impuls zegarowy jest w poziomie HIGH.



Rys. 1-4 Potwierdzanie danych przez SLAVE

Adresowanie

Początkowo standard zakładał 7-bitową przestrzeń adresową, co dawało możliwość zaadresowania do 128 urządzeń. Po wysłaniu znacznika startu wysyłany jest 7 bitowy adres urządzenia SLAVE a następnie bit kierunku. Wartość LOW tego bitu oznacza transmisję od mastera do slave'a (zapis), podczas gdy wartość HIGH kierunek przeciwny (odczyt).

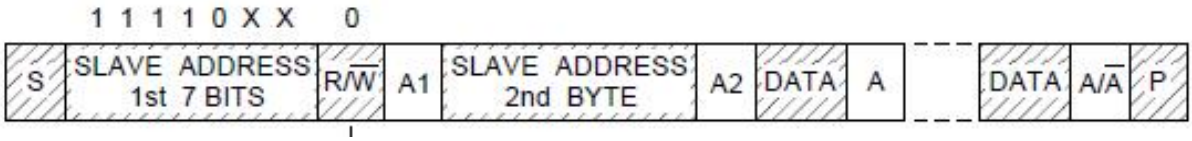


Rys. 1-5 Adresowanie 7 bitowe

Część adresów jest zarezerwowana, pozostawiając do dyspozycji 112 wartości. Jednym z zarezerwowanych adresów jest adres 0, który powoduje wysłanie danych do wszystkich urządzeń podłączonych do magistrali.

Adresowanie 10 bitowe

Wersja 1.0 magistrali pozwala na adresowanie 10-bitowe. W takim przypadku pierwszy przesyłany bajt zawiera 5 z góry ustalonych bitów (11110XX) oraz dwa najstarsze bity adresu 10-bitowego oznaczone jako XX. Drugi bajt zawiera pozostałe 8 bitów adresu. Potem następuje normalna transmisja danych.



Rys. 1-6 Adresowanie 10 bitowe

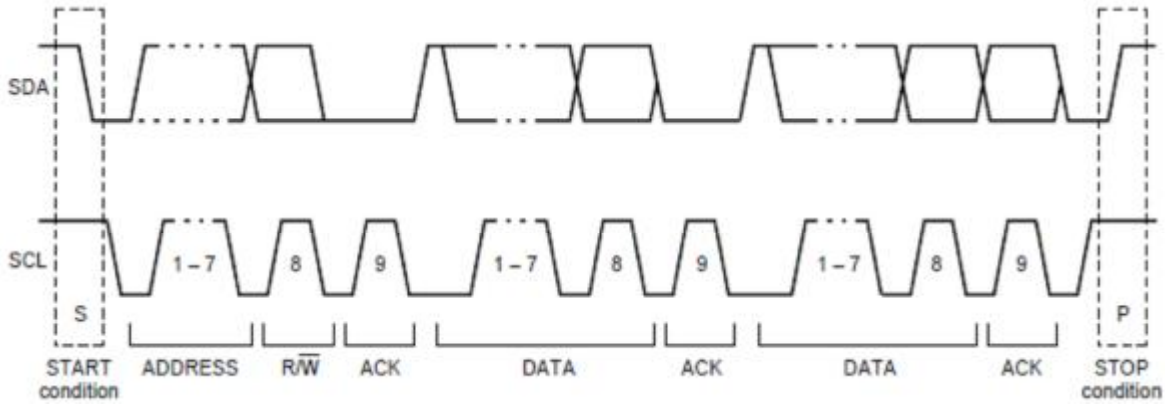
Port szeregowy

Oznaczenie	Długość	
S	1	Bit startu
P	1	Bit Stopu
Rd/Wr	1	Bit Odczyt (Rd) / Zapis (Wr) Rd = 1, Wr = 0
A, NA	1	Potwierdzenie, brak potwierdzenia
Addr	7 lub 10	Adres 7 bitowy, może być rozszerzony do 10
Comm	8	Polecenie (Command)
Data	8	Dane
Count	8	Licznik zawierający długość bloku danych
[…]		Dane przesyłane przez urządzenie I2C (Slave) do kontrolera (Master)

Tab. 1-1 Symbole opisu transmisji I2C

Przesyłanie danych

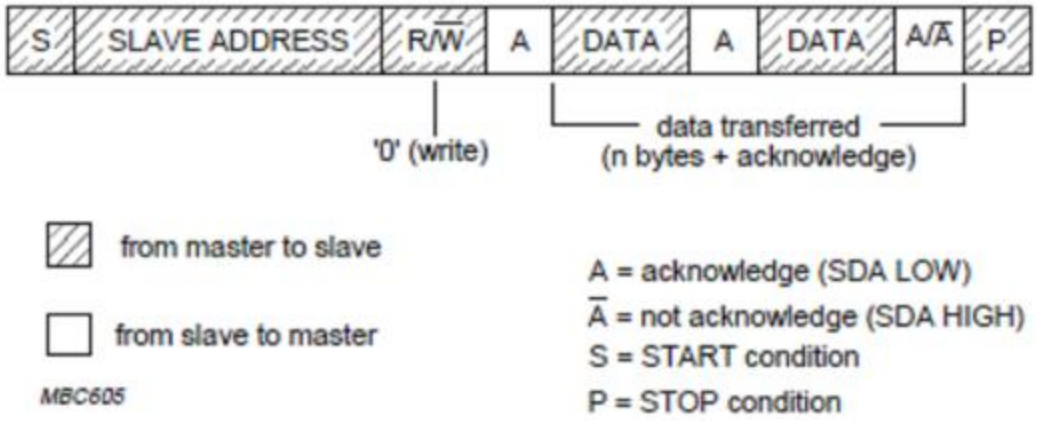
Po pierwszym bajcie (po drugim gdy występuje adresowanie 10 bitowe) przesyłane zostają dane. Może być dowolna liczba bajtów danych. Po zakończeniu przesyłania MASTER generuje warunek stopu.



Występują trzy główne tryby pracy magistrali:

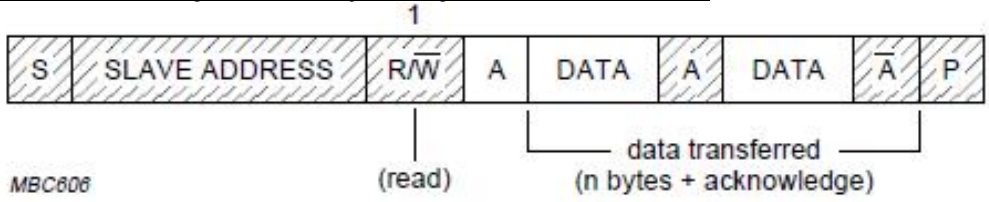
- MASTER pisze do SLAVE
- MASTER czyta ze SLAVE
- Tryb mieszany – kierunek zmienia się w czasie transmisji.

MASTER pisze do SLAVE



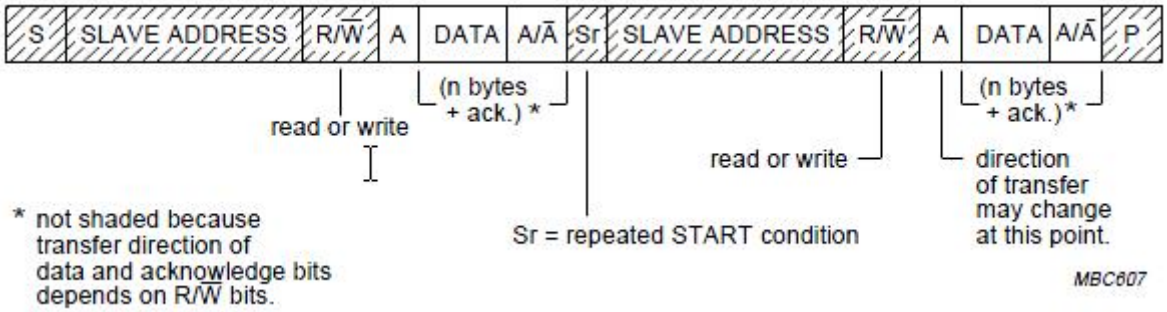
Rys. 1-7 MASTER pisze do SLAVE

MASTER czyta dane przesyłane ze SLAVE



Rys. 1-8 MASTER czyta dane przesyłane ze SLAVE

Tryb mieszany



Rys. 1-9 Kierunek transmisji może się zmieniać w trakcie pojedynczej transakcji

Arbitraż

W systemie I2C może dojść do sytuacji, w której więcej niż jedno urządzenie zechce przejąć kontrolę nad łączem. Zdefiniowano więc specjalną procedurę arbitrażową, która wyłania jedno urządzenie MASTER spośród urządzeń rywalizujących o sterowanie transmisją. Arbitraż wykorzystuje dwie własności magistrali. Po pierwsze jednostka wystawiająca określony poziom na linii SDA może sprawdzić czy stan ten został rzeczywiście osiągnięty. Po drugie magistrala implementuje słabą jedynkę i mocne zero. Tak więc gdy dwa rywalizujące urządzenia MASTER wystawiają swój adres na magistrale, zwycięża to o niższym adresie, gdyż zero jest mocniejsze niż jeden. Widzące to urządzenie o wyższym adresie musi wycofać się z rywalizacji

1.4 Interfejs I2C w mikrokontrolerze

Stan linii SDA i SCL może być monitorowany przez porty procesora. Zajmuje to jednak dużo czasu i lepiej gdy procesor wyposażony jest w kontroler magistrali I2C który wysyła i odbiera bajty.

1.4.1 Kontroler magistrali I2C w procesorze PXA255 (Xscale)

Komunikacja z kontrolerem odbywa się poprzez rejestry widoczne w przestrzeni pamięci lub we/wy.

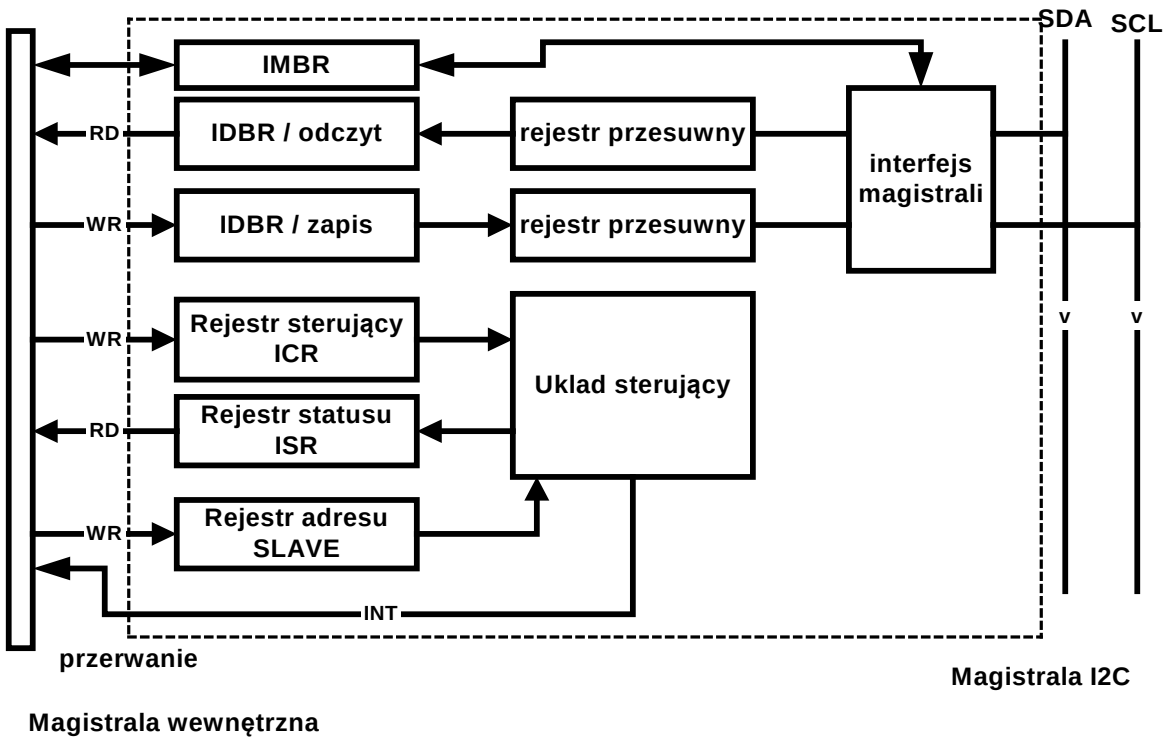
Jednostka sterująca I2C składa się z:

- Interfejsu do 2 przewodowej magistrali I2C
- 2 rejestrów przesuwnych 8 bitowych do konwersji danych z postaci równoległej na szeregową i odwrotnie
- Zestawu rejestrów statusowych i sterujących

Kontroler programuje się poprzez wpis do rejestru sterującego ICR. Status dostępny jest w rejestrze statusowym ISR. Dane odebrane należy odczytać z rejestru danych IDBR, tam też wpisuje się dane do wysyłania. Konwersja z postaci równoległej na szeregową i odwrotnie odbywa się poprzez rejestry przesuwne.

Przerwania generowane są gdy:

- Bufor odbiorczy jest pełny
- Bufor nadawczy jest pusty
- Wybrany zostanie adres SLAVE jednostki
- Nieudanego arbitrażu
- Błędu magistrali



Rys. 1-10 Jednostka sterująca I2C kontrolera PXA255 (Xscale)

Skrót	Nazwa rejestru	Opis
IMBR	Rejestr monitorowania stanu magistrali	Bit 0 – stan SDA Bit1 – stan SCL
IDBR	Rejestr danych	Dane wysyłane i odbierane zawarte są w bitach 0 - 7
ICR	Rejestr sterujący	Zawiera definicje trybu pracy
ISR	Rejestr statusu	Zawiera znaczniki odebrania bajtu, wysłania bajtu, warunku stopu, błędu magistrali, itp.
ISAR	Rejestr adresu SLAVE	Gdy jednostka pracuje jako SLAVE rejestr ten zawiera adres jednostki

Tab. 1-2 Rejestry jednostki sterujące I2C na przykładzie procesora PXA255 (Xscale)

1.5 Instalacja i konfiguracja sterowników

Sprzętowy kontroler I2C obsługiwany jest zazwyczaj przez jądro systemu Linux. Aby tak się stało należy zainstalować odpowiednie moduły sterownika. Ich instalacja zależy od wersji systemu operacyjnego. BeagleBone Black posiada trzy kontrolery I2C ale dla użytkownika dostępne sa dwa.

Po aktywacji sterownika kontrolery I2C powinny one być widoczne w katalogu /dev jako: /dev/i2c-0, /dev/i2c-1 itd. Numery 0,1, oznaczają kolejne kanały.

```
root@beaglebone:~# ls /dev/i2c*
/dev/i2c-0  /dev/i2c-1
```

Przykład 1-1 Testowanie obecności sterowników I2C

Obecność sterowników można też sprawdzić w katalogu **/sys/class/i2c-dev**

```
root@beaglebone:~# ls /sys/class/i2c-dev/
i2c-0  i2c-1
```

Przykład 1-2 Testowanie obecności sterowników I2C

1.6 Narzędzie i2ctools

Pakiet i2c-tools zawiera programy narzędziowe przydatne przy uruchamianiu układów dołączonych do komputera za pomocą interfejsu I2C. Pakiet i2c-tools obejmuje programy:

- i2cdetect – wykrywanie urządzeń i2c dołączonych do magistrali
- i2cdump - odczyt wielu rejestrów urządzenia i2c określonego adresem
- i2cset - zapis ustalonej wartości do wskazanego rejestru określonego adresem urządzenia
- i2cget - odczyt wartości ze wskazanego rejestru określonego adresem urządzenia

Podane powyżej programy są bardzo przydatne gdyż umożliwiają przetestowanie działania urządzeń dołączonych do magistrali i2c bez potrzeby pisania programów.

i2cdetect - identyfikacja urządzeń i2c

Polecenie i2cdetect służy do wykrywania magistral i2c a także do sprawdzenia jakie urządzenia dołączone są do danej magistrali.

`i2cdetect [-y] i2cbus`	Wykrywanie urządzeń na danej magistrali
`i2cdetect -F i2cbus`	Testowanie opcji podanej magistrali
`i2cdetect -V`	Wypisanie wersji programu
`i2cdetect -l`	Testowanie dostępnych magistral

Parametr	Znaczenie
`-y`	Tryb interakcyjny zabroniony
`i2cbus`	Numer (1,2, itd) bądź nazwa **`(/dev/i2c-0,/dev/i2c-1,...)`** magistrali i2c
`-F`	Testowanie opcji implementowanych przez dany kontroler magistrali
`-l`	Testowanie dostępnych magistral i2c

Aby wykryć magistrale i2c dostępne w komputerze BBB wpisujemy na konsoli polecenie: **`i2cdetect –l`** i otrzymamy wyniki jak poniżej:

```
#i2cdetect –l
i2c-0   i2c     OMAP I2C adapter        I2C adapter
i2c-1   i2c     OMAP I2C adapter        I2C adapter
```

Pisząc polecenie **`i2cdetect –y –r 1`** możemy wykryć jakie urządzenia dołączone są do magistrali **`/dev/i2c-1`**. Otrzymujemy wtedy następujący diagram:

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Ekran 1-1 Zestawienie urządzeń i2c wykrytych na magistrali /dev/i2c-1

i2cdump – testowanie rejestrów urządzenia i2c

Polecenie i2cdump służy do testowania stanu rejestrów urządzenia dołączonego do magistrali i2c. Polecenie może być wykonane na podane poniżej sposoby:

`i2cdump [-y] i2cbus address [mode] [bank[bankreg]]`	Wypisanie rejestrów urządzenia
`i2cdump -V`	Wypisanie wersji programu

Parametr	Znaczenie
-y	Tryb interakcyjny zabroniony
i2cbus	Numer (1,2, itd) bądź nazwa **(/dev/i2c-0,/dev/i2c-1,…)** magistrali i2c
address	Adres urządzenia
mode	Opcjonalny parametr b, w, s, i. b – odczytywane są bajty, w - odczytywane są słowa, b – odczytywany jest blok, c – odczytywane są kolejne bajty.
bank, bankreg	Wyjaśnienie trybów podane w [4].
-V	Program poda wersję po czym się zakończy.

Program wypisuje na konsolę zawartość rejestrów urządzenia którego adres podanego jako parametr.

i2cset – zapis do rejestru urządzenia

Program **i2cset** służy do wpisu bajtu do określonego rejestru urządzenia o podanym adresie na ustalonej magistrali i2c. Może on być uruchomiony jak podano poniżej.

i2cset [-y] i2cbus chip-address data-address value [mode] [mask]

Parametr	Znaczenie
i2cbus	Numer bądź nazwa magistrali i2c (1,2, itd)
chip-address	Adres układu z przedziału od 0x00 do 0x7F
data-address	Adres rejestru urządzenia z zakresu od 0x00 do 0xFF
value	Wartość wpisywana do rejestru o adresie podanym wyżej
mode	Opcjonalny parametr b lub w. Gdy podamy b zapisywany będzie jeden bajt , gdy podamy w zapisywane będzie słowo (dwa bajty). Brak parametru oznacza tryb b.
–y	Pominięcie potwierdzeń. Bez tej opcji program zapyta czy wprowadzone parametry są prawidłowe
-V	Program poda wersję po czym się zakończy.

i2cget – odczyt z rejestru urządzenia

Program **`i2cget`** służy do odczytu bajtu lub słowa z określonego rejestru urządzenia o podanym adresie na ustalonej magistrali i2c. Może on być uruchomiony jak podano poniżej.

```
i2cget [-f] [-y] i2cbus chip-address [data-address [mode]]
```

Znaczenie parametrów jak poprzednio.

1.7 Przykład – akcelerometr i żyroskop MPU-6050

Urządzenie do śledzenia ruchu stosowane w telefonach komórkowych i innych urządzeniach.

- Akcelerometr w trzech osiach x,y,z – podaje wartość przyspieszenia dla każdej osi
- Żyroskop w trzech osiach x,y,x. Podaje prędkość kątową , bazuje na sile Coriolisa.
- Procesor ruchu (ang. *Motion processor*)
- Interfejs i2c
- 1024 bajtowa kolejka FIFO

Podłączenie urządzenia podaje poniższa tabela.

Złącze MPU-6050	Złącze BBB	Oznaczenie BBB	Uwagi
SCL	P9-19	I2C2_SCL	Zegar
SDA	P9-20	I2C2_SDA	Dane
VCC	P9-3	Zasilanie +5V	
GND	P9-2	GND	

Tab. 1-3 Podłączenie czujnika MPU-6050 do BBB

Obecność i adres urządzenia wykrywamy poleceniem i2cdetect:

```
root@beaglebone:~# i2cdetect -y -r 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Tab. 1-4 Wykrywanie adresu czujnika MPU-6050 na magistrali i2c BBB

Widzimy że urządzenie posiada adres 0x68. Urządzenie posiada wewnętrzne rejestry 8 bitowe od 0x0D do 0x75. Opisane zostały w dokumentacji [5]. Ważniejsze są podane poniżej.

Rejestr	Adres
ACCEL_XOUT_H	**0x3B**
ACCEL_XOUT_L	**0x3C**
ACCEL_YOUT_H	**0x3D**
ACCEL_YOUT_L	**0x3E**
ACCEL_ZOUT_H	**0x3F**
ACCEL_ZOUT_L	**0x40**
GYRO_XOUT_H	**0x43**
GYRO_XOUT_L	**0x44**
GYRO_YOUT_H	**0x45**
GYRO_YOUT_L	**0x46**
GYRO_ZOUT_H	**0x47**
GYRO_ZOUT_L	**0x48**

Tab. 1-5 Niektóre rejestry czujnika MPU-6050

Aby urządzenie uruchomić należy wpisać 0 do rejestru 0x6B zarządzania energią jak poniżej:

```
i2cset –y 1 0x68 0x6B 0
```

Następnie odczytujemy rejestry urządzenia poleceniem i2cdump

```
root@beaglebone:~# i2cdump -y 1 0x68
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: fd 03 02 3f c0 be fc 4e 02 b1 06 50 28 4f 4e b7
10: a5 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30: 00 00 00 00 00 00 00 00 00 00 01 0d 28 f0 00 34
40: 34 f0 20 fe c4 ff 81 00 1c 00 00 00 00 00 00 00
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 36 41
70: 00 00 00 00 00 68 00 00 00 00 00 00 00 00 00 00
80: fd 03 02 3f c0 be fc 4e 02 b1 06 50 28 4f 4e b7
90: a5 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
b0: 00 00 00 00 00 00 00 00 00 00 01 0d cc f0 c8 34
c0: 5c f0 20 fe ca ff b1 00 0b 00 00 00 00 00 00 00
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 39 34
f0: 00 00 00 00 00 68 00 00 00 00 00 00 00 00 00 00
```

Przykład 1-3 Odczyt wszystkich rejestrów czujnika MPU-6050

Poszczególne rejestry urządzenia odpowiadające poszczególnym funkcjom można odczytać za pomocą polecenie i2cget. Przykładowo starszą część przyspieszenia w osi X odczytujemy za pomocą polecenia:

`i2cget –y 1 0x68 0x3B`

a młodszą za pomocą polecenia

`i2cget –y 1 0x68 0x3B`

```
root@beaglebone:~# i2cget -y 1 0x68 0x3B
0x0b
root@beaglebone:~# i2cget -y 1 0x68 0x3C
0xa4
```

Przykład 1-4 Odczyt wybranych rejestrów czujnika MPU-6050

Program obsługi podany jest na stronie:

https://www.teachmemicro.com/beaglebone-black-mpu6050-i2c-tutorial-part-2/

```
// Obsluga akcelerometru - zyroskopu MPU-6050
// Komputer BeagleBone Black
// Na podstawie
// https://www.teachmemicro.com
// beaglebone-black-mpu6050-i2c-tutorial-part-2/
// J. Ulasiewicz 2017

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define  deviceADDR      " 0x68"
#define  PWR_MGMT_1      " 0x6B"
#define  ACCEL_X_OUT_H   " 0x3B "
#define  ACCEL_X_OUT_L   " 0x3C "
#define  ACCEL_Y_OUT_H   " 0x3D "
#define  ACCEL_Y_OUT_L   " 0x3E "
#define  ACCEL_Z_OUT_H   " 0x3F "
#define  ACCEL_Z_OUT_L   " 0x40 "
#define  GYRO_X_OUT_H    " 0x43 "
#define  GYRO_X_OUT_L    " 0x44 "
#define  GYRO_Y_OUT_H    " 0x45 "
#define  GYRO_Y_OUT_L    " 0x46 "
#define  GYRO_Z_OUT_H    " 0x47 "
#define  GYRO_Z_OUT_L    " 0x48 "

#define  cmdGet     "i2cget -y 1"
#define  cmdSet     "i2cset -y 1"
#define  MAX_BUFFER 256

int accel_x;
int accel_y;
int accel_z;
int gyro_x;
int gyro_y;
int gyro_z;

char data[5*MAX_BUFFER];
```

```
/* Wykonaj polecenie cmd w bash */
int exec(char* cmd) {
     FILE * stream;
     int cnt = 0;
     char buffer[MAX_BUFFER];
     strcpy(data,"");
     // Utworzenie procesu potomnego
     stream = popen(cmd, "r");
     if (stream == NULL) { perror("popen"); exit(0);}
     while (!feof(stream))
          if (fgets(buffer, MAX_BUFFER, stream) != NULL) {
              strcat(data,buffer);
               cnt++;
          }

     pclose(stream);
     // Konwersja z string hex do dec
     cnt = (int)strtol(data, NULL, 16);
     return cnt;
}

/*funkcja wykonuje seti2c */
int set(char* reg1,  char* reg2, int value){
     char str[100];
     char tmp[10];
     int res;
     sprintf(str,"%s %s %s %d ",cmdSet,reg1,reg2,value);
     res = exec(str);
     return(res);
}

/*funkcja wykonuje geti2c */
int get(const char* reg1, const char* reg2){
     char str[100];
     int res;
     sprintf(str,"%s %s %s ",cmdGet,reg1,reg2);
     res = exec(str);
     return(res);
}

int main(){
    int i, xl,xh,x,res;
     set(deviceADDR, PWR_MGMT_1, 0);    // Wlaczenie urzadzenia
     for(i=0;i<1000;i++) {
          xh = get(deviceADDR, ACCEL_X_OUT_H);
          xl = get(deviceADDR, ACCEL_X_OUT_L);
          accel_x = (xh<<8) + xl;
          xh = get(deviceADDR, ACCEL_Y_OUT_H);
          xl = get(deviceADDR, ACCEL_Y_OUT_L);
          accel_y = (xh<<8) + xl;
          xh = get(deviceADDR, ACCEL_Z_OUT_H);
```

```
            xl = get(deviceADDR, ACCEL_Z_OUT_L);
            accel_z = (xh<<8) + xl;
            xh = get(deviceADDR, GYRO_X_OUT_H);
            xl = get(deviceADDR, GYRO_X_OUT_L);
            gyro_x = ((xh<<8) + xl)/131;
            xh = get(deviceADDR, GYRO_Y_OUT_H);
            xl = get(deviceADDR, GYRO_Y_OUT_L);
            gyro_y = ((xh<<8) + xl)/131;
            xh = get(deviceADDR, GYRO_Z_OUT_H);
            xl = get(deviceADDR, GYRO_Z_OUT_L);
            gyro_z = ((xh<<8) + xl)/131;
            printf("x: %5d y: %5d z: %5d ",
                    accel_x,accel_y,accel_z);
            printf("x: %5d y: %5d z: %5d\n",
                    gyro_x,gyro_y,gyro_z);
          sleep(1);
      }
}
```

Przykład 1-5 Obsługa czujnika MPU-6050

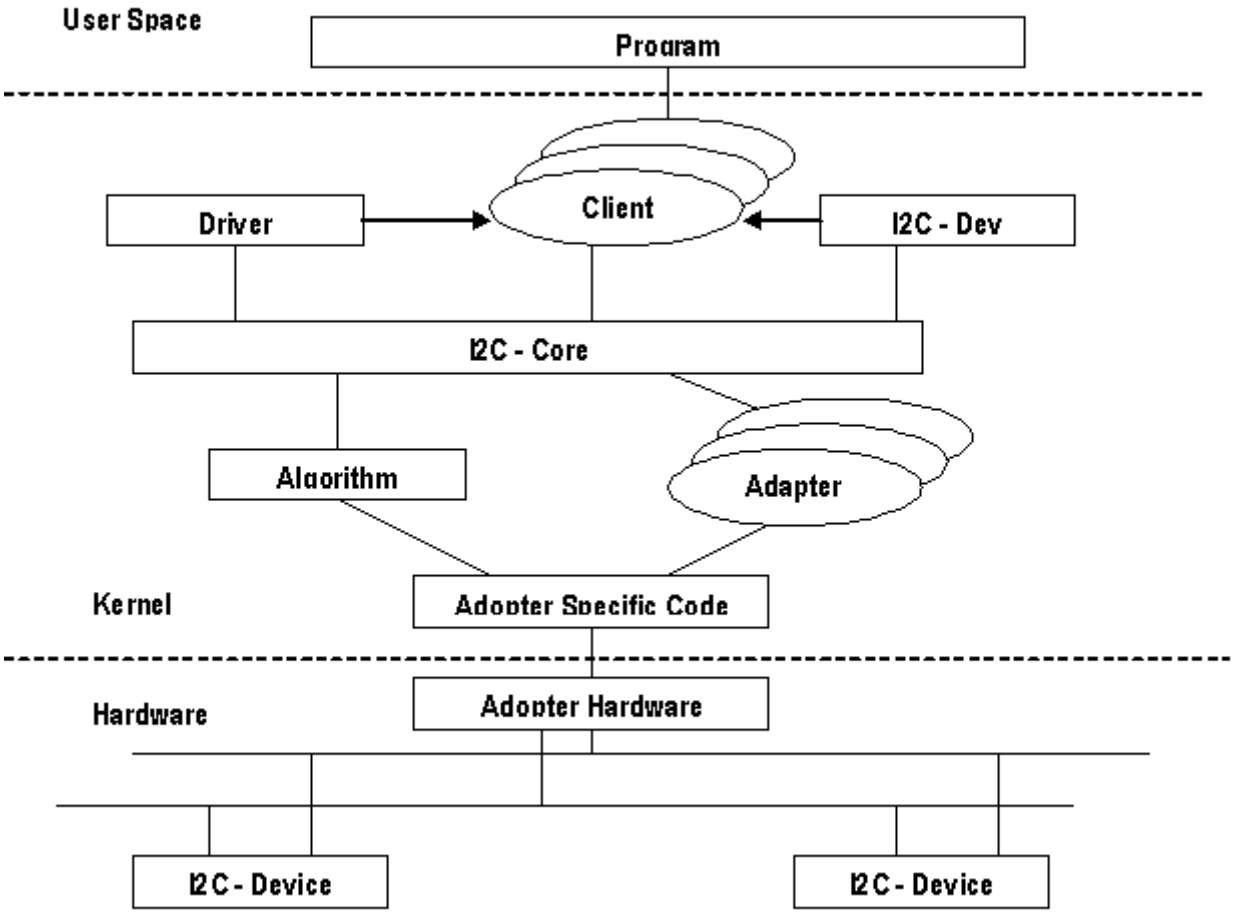
1.8 Komunikacja z urządzeniami I2C, biblioteka i2c_smbus

Problem z komunikacją:
Komunikacja z urządzeniem ma postać transakcji gdzie wymagane są operacje na bitach (start, stop, potwierdzenia, odczyt/zapis)

Tym niemniej w do komunikacji z urządzeniami I2C daje się wykorzystać abstrakcję pliku.
Nie wszystkie transakcje I2C dadzą się wyrazić poprzez użycie funkcji read, write. Powodem są:

- występowanie znaczników start/stop,
- konieczność generowania bitów potwierdzeń
- zmiana kierunków transmisji
- manewrowanie impulsami zegarowymi

W takim przypadku należy użyć funkcji interfejsowych biblioteki i2c_smbus.



Rys. 1-11 Dostęp do funkcji i2c (według http://processors.wiki.ti.com)

Program aplikacyjny komunikuje się z kontrolerem za pomocą polecenia ioctl.

```
int ioctl(int fildes, int request, ... /* arg */)
```

Gdzie:
fildes – uchwyt pliku
request – polecenie, jest ich wiele, zależą od urządzenia

Ustalenie adresu urządzenia slave

```
ioctl(file, I2C_SLAVE, long addr)
```

file Uchwyt pliku
addr Adres urządzenia slave

Ustalenie adresacji 7 lub 10 bitów

```
ioctl(file, I2C_TENBIT, long select)
```

file Uchwyt pliku
select 0 – adres 7 bit, 1 – adres 10 bit

Test funkcji kontrolera

```
ioctl(file, I2C_FUNCS , long *funcs)
```

file Uchwyt pliku
func Funkcje kontrolera (ustawiane bity)

Transakcja I2C

```
ioctl(file,I2C_RDWR,struct i2c_rdwr_ioctl_data
*msgset)
```

file Uchwyt pliku
msgset Wskaźnik na strukturę specyfikacji transakcji

```
struct i2c_rdwr_ioctl_data {
  // Wskaźnik na tablicę przesłań prostych
  struct i2c_msg *msgs;
  // Liczba przesłań
  int nmsgs;
}

struct i2c_msg {
  __u16 addr;  // slave address
  __u16 flags; // Flagi – specyfikują akcję
  __u16 len;   // długość komunikatu
  __u8 *buf;   // Wskaźnik na bufor
};
```

Funkcja realizuje odczyt lub zapis w zależności od ustawienia flag.
Interfejs realizuje także podstawowe funkcje:

```
int read(int fdes, void *bufor, int nbytes)
```

fdes Uchwyt do pliku zwracany przez funkcję open
bufor Bufor w którym umieszczane są przeczytane bajty
nbytes Liczba bajtów którą chcemy przeczytać.

```
int write(int fdes, void *bufor, int nbytes)
```

fdes Uchwyt do pliku zwracany przez funkcję open
bufor Bufor w którym umieszczane są bajty przeznaczone do zapisu
nbytes Liczba bajtów którą chcemy zapisać

W pliku i2c-dev.h zawarte są funkcje opakowujące ioctl co ułatwia programowanie.

function	description	parameters	return value
SMBus Access			
`write_quick(addr)`	Quick transaction.	int addr	long
`read_byte(addr)`	Read Byte transaction.	int addr	long
`write_byte(addr,val)`	Write Byte transaction.	int addr,char val	long
`read_byte_data(addr,cmd)`	Read Byte Data transaction.	int addr,char cmd	long
`write_byte_data(addr,cmd,val)`	Write Byte Data transaction.	int addr,char cmd,char val	long
`read_word_data(addr,cmd)`	Read Word Data transaction.	int addr,char cmd	long
`write_word_data(addr,cmd,val)`	Write Word Data transaction.	int addr,char cmd,int val	long
`process_call(addr,cmd,val)`	Process Call transaction.	int addr,char cmd,int val	long
`read_block_data(addr,cmd)`	Read Block Data transaction.	int addr,char cmd	long[]
`write_block_data(addr,cmd,vals)`	Write Block Data transaction.	int addr,char cmd,long[]	None
`block_process_call(addr,cmd,vals)`	Block Process Call transaction.	int addr,char cmd,long[]	long[]
I2C Access			
`read_i2c_block_data(addr,cmd)`	Block Read transaction.	int addr,char cmd	long[]
`write_i2c_block_data(addr,cmd,vals)`	Block Write transaction.	int addr,char cmd,long[]	None

Zasady programowania urządzeń I2C przy użyciu abstrakcji zapisu/odczytu z/do pliku:

1. Włączyć pliki nagłówkowe: **`<linux/i2c-dev.h>`**
2. Otworzyć urządzenie, jest widziane jako plik specjalny:
   `fh = open("/dev/i2c-1",O_RDWR);`
3. Poinformować sterownik z którym urządzeniem będziemy się komunikować: **`ioctl(fh, I2C_SLAVE, DEV_ADDR)`**
   `DEV_ADDR –` adres urządzenia
4. Użyć funkcji read, write do wysyłania i odbierania bajtów
5. Gdy nie jest to wystarczające użyć funkcji z biblioteki SMBus

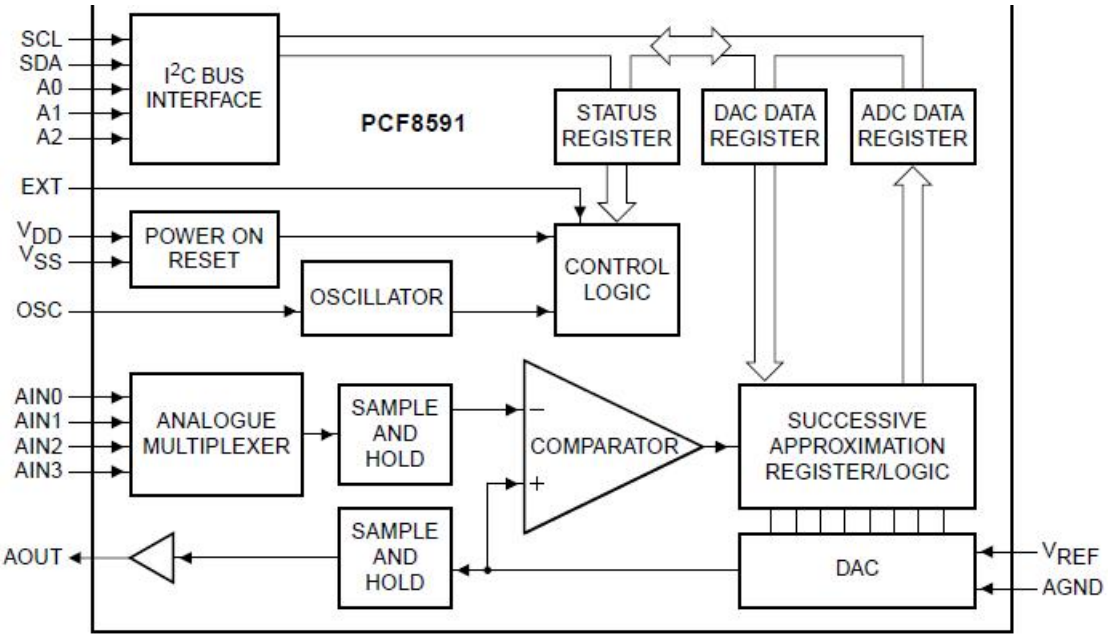
1.8.1 Układ PCF8591, przetwornik AD/DA z I2C – obsługa w języku C

Układ PCF8591 jest 4 kanałowym 8 bitowym przetwornikiem AD i 1 kanałowym przetwornikiem DA z interfejsem I2C.

Własności:

- Pojedyncze napięcie zasilania 2.5 – 6V
- Interfejs szeregowy I2C
- Trzy wejścia adresowe – 8 układów
- Częstotliwość próbkowania zgodna z szybkością magistrali
- 4 wejścia AC niesymetryczne lub 2 różnicowe
- Konwersja 8 bitowa metodą sukcesywnej aproksymacji
- 1 kanał 8 bitowego przetwornika DA

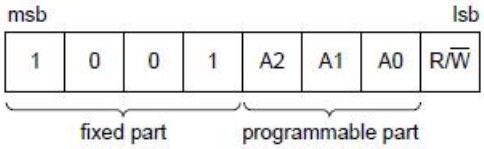
1.8.2 Budowa



Rys. 1-12 Budowa układu PCF8591

1.8.3 Adresowanie

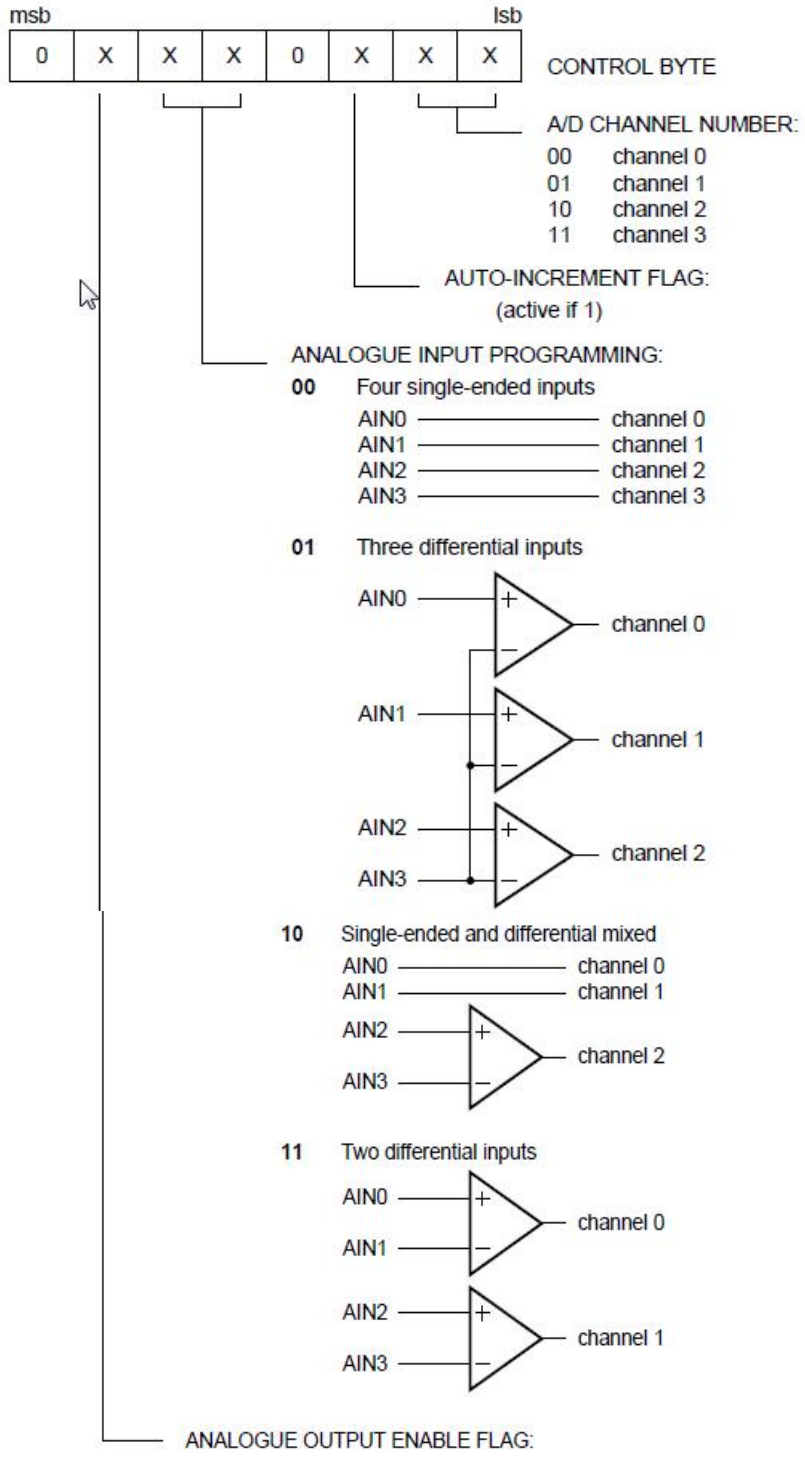
Urządzenie uaktywnia się po przesłaniu prawidłowego adresu. Zależy on od polaryzacji wejść A0, A1, A2. Gdy są na poziomie 0 to adres wynosi 1001000. W praktyce podaje się adres przesunięty o 1 pozycję w prawo.



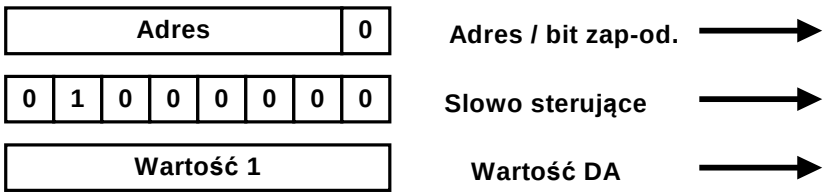
Rys. 1-13 Adresowanie układu PCF8591

1.8.4 Sterowanie układem

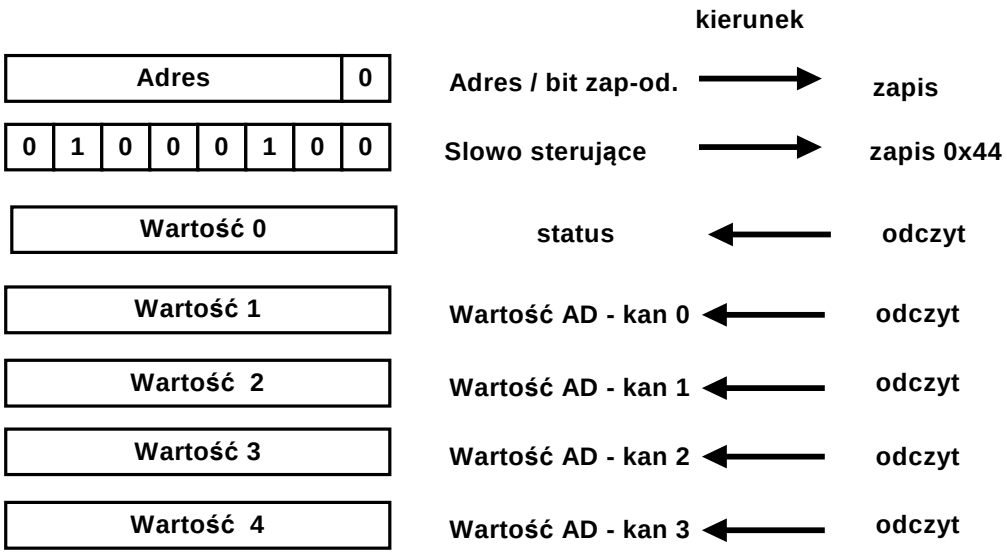
Drugi bajt przesłany do urządzenia traktowany jest jako słowo sterujące. Znaczenie bitów definiuje poniższy rysunek.



Przykładowo gdy wyślemy słowo sterujące 01000100 = 0x44 to będą odczytywane kanały AD 0,1,2,3 w trybie auto inkrementacji i aktywne będzie wyjście DA.



Rys. 1-14 Zapis wartości wyjściowej na przetwornik DA



Rys. 1-15 Odczyt wartości zmierzonej z przetwornika AD

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/i2c-dev.h>
#define PCF8591_ADR 0x48
#define PCF8591_DAC_ENABLE 0x40

int write_dac(int fh,char DAC_value);
int read_adca(int fh,int val[4]);

int main(int argc, char *argv[]){
      int fh;
      int i, kroki,chan,res;
      int aval[4];
      fh = open("/dev/i2c-1",O_RDWR);
      if (fh < 0) {
            fprintf(stderr, "I2C file error!\n"); exit(1);
      }
      printf("Uchwyt %d\n",fh);
      if (ioctl(fh, I2C_SLAVE, PCF8591_ADR) < 0) {
            fprintf(stderr, "I2C adress error!\n");
      exit(1);
      }
      val = 0x00;
     write_dac(fh,val);
      printf("\n");
      for(i=0;i<kroki;i++) {
         read_adca(fh,aval);
         printf(" %02d %02d %02d %02d \n",
        aval[0],aval[1],aval[2],aval[3]);
         val = (val+10) % 256;
         write_dac(fh,val);
         usleep(500000);
      }
      close(fh);
      return 0;
}

int write_dac (int fh, char value){
  int res;
  char buf[2];
  buf[0] = PCF8591_DAC_ENABLE;
  buf[1] = value;
  res = write(fh, buf, 2);
  if(res < 0) {  perror("write"); return -1;}
  printf (" DAC: %3d ",value);
  return res;
}
```

```
int read_adca(int fh, int val[4]){
  char vbuf[4];
  int res,i;
  vbuf[0] = 0x44;
  // Polecenie odczytu 4 kan z autoinkrementacja
  res = write(fh, vbuf, 1);
  if(res < 0) {  perror("write"); return -1;}
  // Bajt statusu odrzucany
  res = read(fh, vbuf, 1);
  if(res < 0) {  perror("read"); return -1;}
  // Odczyt 4 kanalow
  res = read(fh, vbuf, 4);
  if(res < 0) {  perror("read"); return -1;}
  for(i=0;i<4;i++) val[i] = vbuf[i];
  return res;
}
```

Przykład 1-6 Program sterowania przetwornikami AD/DA układu PCF8591

1.8.5 Przykład – akcelerometr i żyroskop MPU-6050

Uwaga!
Aby program dał się skompilować należy użyć:
sudo apt-get install libi2c-dev

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>

#include <linux/i2c.h>
#include "i2c-dev.h"
#define  PWR_MGMT_1    0x6B
#define  WHO_AM_I_REG   0x75
…
int deviceI2CAddress = 0x68; // address of MPU-6050 device
int accel_x;
int accel_y;
int accel_z;

int main(void) {
      int fd,res,xh,xl,i;
      long funkcje;
      unsigned char par;

      // Otwarcie urzadzenia /dev/i2c-1
      if ((fd = open("/dev/i2c-1", O_RDWR)) < 0){
```

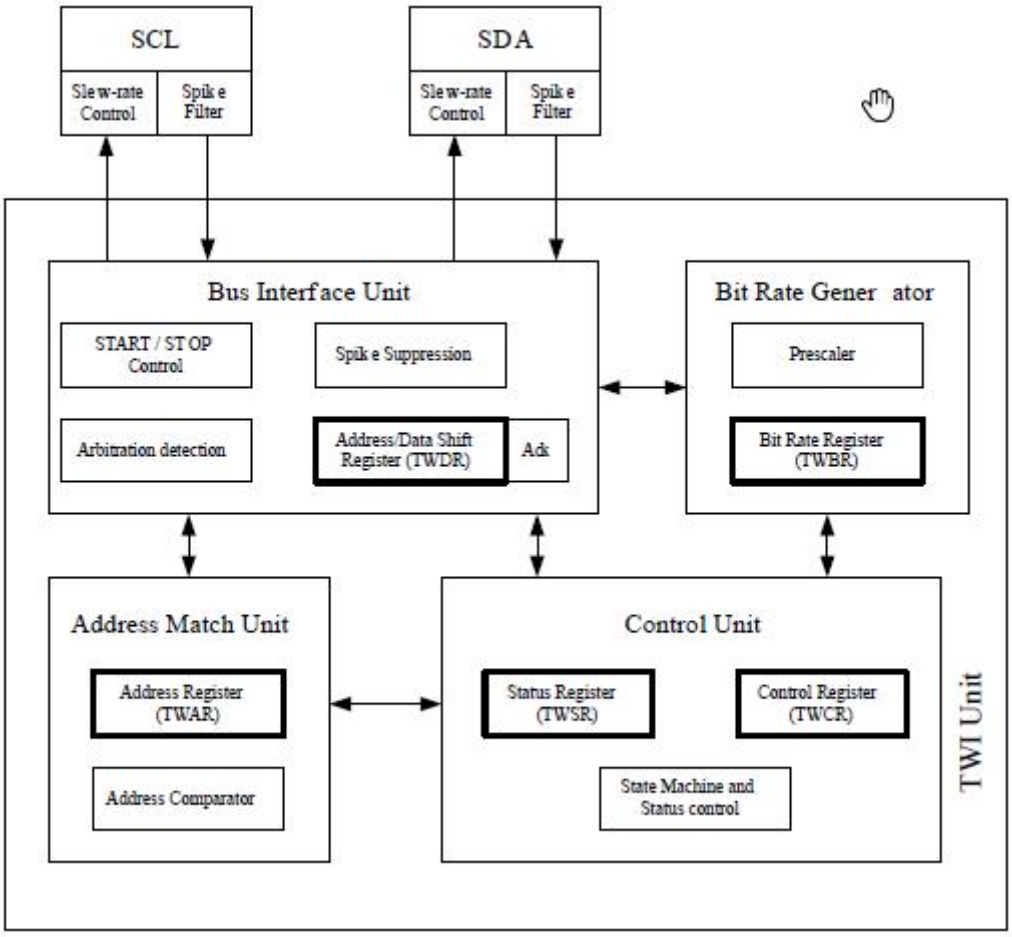
```
            perror("open");
            return 1;
      }
      printf("fd = %d\n",fd);
      // Polaczenie z MPU-6050 jako i2c slave
      if (ioctl(fd, I2C_SLAVE, deviceI2CAddress) < 0) {
            perror("device address");
            return 1;
      }

      if (ioctl(fd, I2C_FUNCS, &funkcje) < 0) {
            perror("funkcje");
            return 1;
      }
      printf("Funkcje kontrolera; %4x \n",funkcje);

      // Start urzadzenia -----------------------------
      res = i2c_smbus_write_byte_data(fd,PWR_MGMT_1,0);
      if(res < 0) perror("write");

      // Identyfikacja -----------------
      par = WHO_AM_I_REG;
      res  = i2c_smbus_read_byte_data(fd,par);
      if(res < 0) perror("read 1");
      printf("Who am I = %2X\n",res);

  for(i=0;i<20;i++) {
      xh = i2c_smbus_read_byte_data(fd,ACCEL_X_OUT_H);
      xl = i2c_smbus_read_byte_data(fd,ACCEL_X_OUT_L);
      accel_x = (xh<<8) + xl;
      xh = i2c_smbus_read_byte_data(fd,ACCEL_Y_OUT_H);
      xl = i2c_smbus_read_byte_data(fd,ACCEL_Y_OUT_L);
      accel_y = (xh<<8) + xl;
      xh = i2c_smbus_read_byte_data(fd,ACCEL_Z_OUT_H);
      xl = i2c_smbus_read_byte_data(fd,ACCEL_Z_OUT_L);
      accel_z = (xh<<8) + xl;
      printf("x:%5d y:%5d z:%5d
             \n",accel_x,accel_y,accel_z);
      sleep(1);
 }
      return 0;
}
```

Przykład 1-7 Program odczytu danych z czujnika ruchu MPU-6050

```
root@beaglebone:~/prog/accel# ./mpu6050-i2c
fd = 3
Funkcje kontrolera; efe000d
Who am I = 68
x: 55356 y: 65484 z:  9904
x: 55356 y:     4 z: 10120
x: 55544 y:   148 z:  9972
x: 55308 y: 65308 z:  9992
x: 55088 y: 65480 z:  9728
x: 55268 y:   252 z: 10100
```

Przykład 1-8 Wyniki działania programu odczytu z czujnika ruchu MPU-6050

1.8.6 Kontroler magistrali I2C w procesorze Atmega 328 (Arduinoe)



Rys. 1-2 Kontroler TWI w procesorze Atmega 328

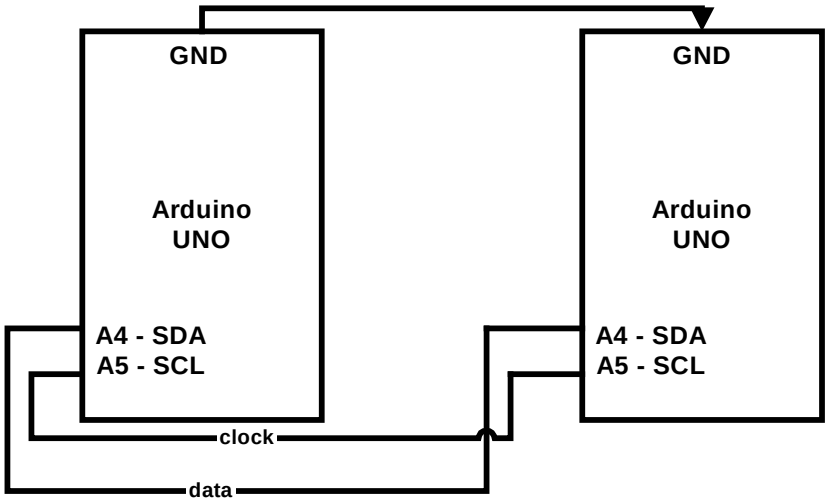
Praca z kontrolerem TWI polega na przesłaniu do rejestru sterującego polecenia a następnie obsługi przerwania z TWI. W ramach obsługi przerwania należy zidentyfikować jego przyczynę przez odczyt rejestru statusu TWSR a następnie zmodyfikować wspólne struktury danych i wysłać kolejne polecenie do jednego z rejestrów.

1.9 Bibliteka Wire - funkcje interfejsowe TWI (i2c) dla Arduino

Współpraca z magistralą TWI na poziome rejestrów i przerwań jest trudna. Biblioteka Wire dostarcza wygodnych funkcji interfejsowych. Opisuje je poniższa tabela.

Nazwa	Opis
`Wire.begin()`	Inicjalizacja interfejsu i2c - master
`Wire.begin(address)`	Inicjalizacja interfejsu i2c – slave, address jest adresem stacji slave
`Wire.beginTransmission(address)`	Funkcja rozpoczęcia transmisji do urządzenia Slave o adresie **`address`**. Kolejne bajty maja być pisane funkcja **`Wire.write`** a na koniec transakcji ma być wywołana funkcja **`endTransmission`**
`Wire.write (value)` **`Wire.write (string)`** **`Wire.write (buf,length)`**	Funkcja wysyłania danych: **`value`** , łańcucha **`string`** lub bufora o długości **`length`**. Wywołanie ma być poprzedzone przez **`Wire.beginTransmission`** a zakończone **`Wire.endTransmission()`**. Funkcja zwraca liczbę zapisanych bajtów.
`Wire.onReceive(handler)`	Rejestracja funkcji **`handler`** która ma być wywołana gdy slave odbierze dane.
`Wire.available()`	Funkcja zwraca liczbę odebranych bajtów które maja być odczytane funkcja **`Wire.read().`** Funkcja ma być wywołana na stacji master po funkcji **`Wire.requestFrom()`** lub na stacji slave wewnątrz handlera określonego w **`Wire.onReceive(handler).`**
`Wire.read()`	Funkcja zwraca odebrany bajt. Ma być wywołana w stacji master po funkcji **`Wire.requestFrom()`** lub na stacji slave.
`Wire.onRequest (handler)`	Rejestracja funkcji handler. Funkcja będzie wywołana na tym urządzeniu slave gdy master zażąda danych.
`Wire.setClock()`	Funkcja ustawia zegar kontrolera i2c. Standardowo jest to 100000 (100 KHz) a w trybie fast 400000 (40 KHz).

Tab. 1-6 Funkcje interfejsu I2C dla Arduino



Rys. 1-3 Połączenie dwóch Arduino poprzez interfejs i2c

1.9.1 Przykład - master pisze slave czyta

Przykład testu gdy master pisze a slave czyta dany jest poniżej. Połączenia należy wykonać zgodnie z Rys. 1-3.

```
#include <Wire.h>

void setup() {
  pinMode(LED_BUILTIN, OUTPUT); // Dioda wewn. jako wyjscie
  Wire.begin(); // Inicjalizacja sterownika
}

byte x = 0;

void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // zapal diode
  Wire.beginTransmission(8);  // pocz. transm. do urzadz. #8
  Wire.write("x is ");        // wysylaj 5 znakow
  Wire.write(x);              // wyslij 1 znak x
  Wire.endTransmission();     // koniec transmisji
  x++;
  delay(500);
  digitalWrite(LED_BUILTIN, LOW); // zgas diode
  delay(500);
}
```

Przykład 1-9 Master wysyła napis przez i2c

```
// Wire Slave Receiver
// by Nicholas Zambetti <http://www.zambetti.com>

#include <Wire.h>

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  Wire.begin(8);                // Ustalenie adresu i2c na #8
  Wire.onReceive(receiveEvent); // Rejestracja funkcji odbioru
  Serial.begin(9600);           // Init portu szeregowego
  Serial.println("I2C receiver");
}

void loop() {
  delay(105);
}

// Funkcja jest wykonywana gdy odbierane sa dane
// funkcja jest rejestrowana jako wykonywana przy odbiorze
void receiveEvent(int howMany) {
  digitalWrite(LED_BUILTIN, HIGH); // zapal diode
  while (Wire.available()> 1) { // czekaj na znaki
    char c = Wire.read();       // odbieraj znak
    Serial.print(c);            // drukuj znak
  }
  int x = Wire.read();    // odbieraj znak jako int
  Serial.println(x);      // Drukuj znak
  digitalWrite(LED_BUILTIN, LOW); // zgas diode
}
```

Przykład 1-10 Slave odbiera napis przez interfejs i2c

Literatura

[1] THE I 2C-BUS SPECIFICATION VERSION 2.1 JANUARY 2000
http://www.lm-sensors.nu/

[2] Embedded Systems Academy
http://www.esacademy.com/en/library/technical-articles-and-documents/miscellaneous/i2c-bus/

[3] Witryna xGoat – Using I2C from user space in Linux
https://xgoat.com/wp/2007/11/11/using-i2c-from-userspace-in-linux/

[4] lm-sensors – strona poświęcona monitorowaniu sprzetu.
http://www.lm-sensors.org/wiki/man/i2cdetect

[5] MPU-6500 Register Map and Descriptions Revision 2.1
https://www.invensense.com/wp-content/uploads/2015/02/MPU-6500-Register-Map2.pdf

[6] Witryna Texas Instruments
http://processors.wiki.ti.com/index.php/AM335x-PSP_04.06.00.02_Features_and_Performance_Guide#I2C_Driver