Moduł z przekaźnikami - ARD Relay Shield

Magistrala I^2C

Wstęp

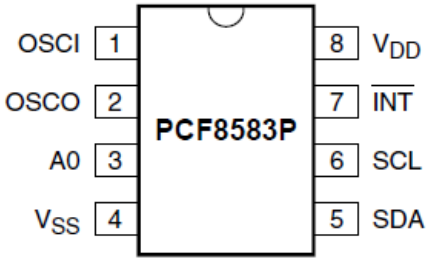
Powiem szczere, że moduł przekaźnikowy jest tylko i wyłącznie małym pretekstem do omówienia bardziej rozbudowanej kwestii jaką jest magistrala I^2C i jej obsługa z poziomu Arduino. Na początek powiem tylko, że magistrala ta jest stosunkowo popularna i znajdziemy wiele układów wykorzystujących ją do komunikacji z mikrokontrolerami. Podczas tej lekcji poznamy 3 z nich: zegar czasu rzeczywistego PCF8583, pamięć EEPROM 24C02 oraz ekspander wejściowo wyjściowy PCF8574(A). Ale zacznijmy od paru słów dot. samej magistrali I^2C.

Magistrala I^2C (czyt. i kwadrat c)

Magistrala ta składa się z 2 linii łączących układ z mikrokontrolerem, są to linie SDA i SCL. Pierwsza z nich to dwukierunkowa linia danych, natomiast druga stanowi sygnał zegarowy dla układów podłączonych pod magistralę, który jest generowany przez mikrokontroler (pracujący jako układ master, czyli inicjalizujący transmisję). Linie te powinny posiadać podciąganie do zasilania poprzez rezystory o wartości ok. 4,7kΩ (tzn. jeden rezystor powinien łączyć linię SDA z zasilaniem +5V, a drugi linię SCL z napięciem +5V)[1]. Do magistrali tej możemy podpiąć równolegle wiele układów, gdyż każdy model posiada inny adres[2], a czasem istnieje także możliwość ustawienia adresu danego układu za pomocą dodatkowych linii w które niektóre modele są wyposażone. Wszystkie te informacje znajdziemy zazwyczaj w karcie katalogowej danego układu. Nasz procesor posiada sprzętowe wsparcie dla tej magistrali i jeśli zostanie aktywowany odpowiedni moduł to wtedy piny A5 i A4 funkcjonują odpowiednio jako linie SCL i SDA magistrali. Czas zatem przejść do pierwszych ćwiczeń praktycznych.

Zegar czasu rzeczywistego PCF8583

Pierwszym układem jaki poznamy będzie zegar czasu rzeczywistego PCF8583. Poniżej na rysunku przedstawiam rozkład wyprowadzeń tego zegarka oraz ich opis.

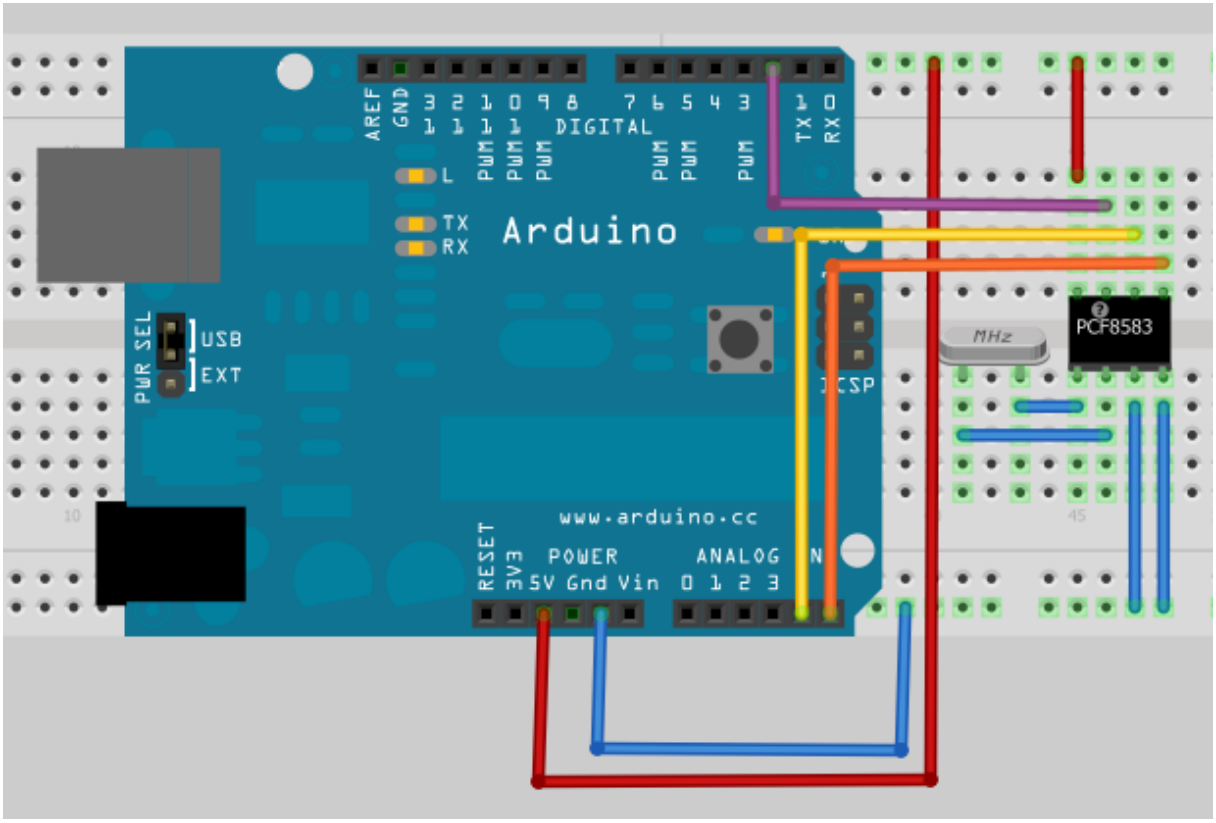


PIN	Funkcja
OSCI	wejście sygnału z kwarcu „zegarkowego” 32 768 Hz
OSCO	wyjście sygnału do kwarcu
A0	dodatkowa linia adresowa, jeśli zwarta do masy to układ ma adres 0xA0, jeśli do zasilania to adres układu jest równy 0xA2
Vss	masa układu

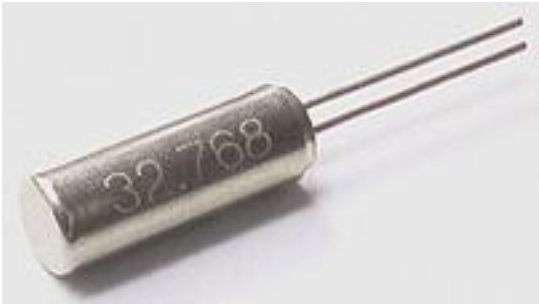
[1] rezystory te nie zostały zaznaczone na rysunkach połączeń, aby nie wprowadzać zamieszania. Oczywiście każdorazowo należy obowiązkowo podłączyć te rezystory podciągające – w przeciwnym razie narażamy się na nieprawidłową pracę całej magistrali.
[2] zdarzają się czasem wyjątki i niektóre modele mają takie same adresy

SDA	linia danych magistrali I^2C
SCL	linia zegarowa magistrali I^2C
/INT	linia generująca przerwania według konfiguracji układu
Vdd	Zasilanie układu

Zgodnie z tym opisem podłączmy zatem tę „kostkę” do naszego Arduino w następujący sposób:



Zwykle jednak spotkamy się z kwarcem zegarkowym o częstotliwości 32,768kHz w takiej obudowie:



zamiast przedstawionej na rysunku dużej wersji:

Po wykonaniu tych połączeń możemy przystąpić do pierwszych testów odczytu danych z zegara:

```
#include <Wire.h>//dołączamy bibliotekę do obsługi I2C

byte bcd2byte(byte x){
  return (x>>4)*10+(x&0x0F);
}

void setup()
{
  Wire.begin();//inicjalizujemy magistralę
  Serial.begin(9600);
}

byte val=0,val_old=255;

void loop()
{
  Wire.beginTransmission(0xA0/2);//wysyłamy adres układu, z którym chcemy
się komunikować (z powodu przesunięcia w bibliotece musimy adres podzielić
przez 2 i "odciąć" najmłodszy bit)
  Wire.send(0x02);//wysyłamy adres komórki wewnątrz układu, z której chcemy
przeprowadzić odczyt (pod adresem 0x02 znajduje się rejestr sekund)
  Wire.endTransmission();//wstrzymujemy transmisję (nadawanie)
  Wire.requestFrom(0xA0/2,1);//przygotowujemy układ od odczytu jednego
bajtu danych
  val=Wire.receive();//odbieramy bajt danych
  val=bcd2byte(val);//konwertujemy go z kodu BCD
  if(val!=val_old){//jeśli wartość się zmieniła to ją wyświetlamy
    val_old=val;
    Serial.println(val,DEC);
  }
}
```

Wyjaśnienia może wymagać tylko kwestia konwersji z kodu BDC. Otóż nasz zegar przechowuje dane zakodowane w formacie BCD – czyli 4 młodsze bity przechowują cyfrę jedności, a 4 starsze bity cyfrę dziesiątek. Do konwersji wykorzystujemy logiczne operacje bitowe i przesunięcia bitowe, które postaram się pokrótce wyjaśnić.

Koniunkcja

Koniunkcja realizowana jest za pomocą operatora &. Daje na odpowiadającym bicie 1 wtedy i tylko wtedy gdy na odpowiadających bitach w dwóch zmiennych znajduje się 1. Czyli np.:

- 0b1111000 & 0b00110011 = 0b00110000
- 0b0101010 & 0b00000000 = 0b00000000

Alternatywa

Alternatywa realizowana jest operatorem | (pionowa kreska, zwykle na klawiaturach slash + SHIFT). Daje na odpowiadającym bicie 1 wtedy i tylko wtedy gdy choć jeden z odpowiadających bitów w 2 zmiennych znajduje się 1. Np.:

- 0b00001111 | 0b11110000 = 0b11111111
- 0b01010101 | 0b11110000 = 0b11110101

Negacja

Zmienia wszystkie 0 na 1 o na odwrót, realizowana jest operatorem ~ (tylda, ` + SHIFT, czasem trzeba wpisać 2 znaki w systemach Windows) Np.:

- ~0b00001111 = 0b11110000
- ~0b01010100 = 0b10101011

Przesunięcia bitowe

Przesunięcia bitowe powodują przesunięcie bitów w zmiennej o określoną ilość miejsc w prawo lub lewo. Nowe bity są wypełniane zerami. Przesunięcia są realizowane operatorami << i >>. Np.:

- 0b00001111<<2 = 0b00111100
- 0b11001100<<4 = 0b11000000
- 0b10101010<<1 = 0b01010100
- 0b00001111>>2 = 0b00000011
- 0b11001100>>3 = 0b00011001

Odczyt pozostałych informacji z zegara

Teraz pora na odczyt pozostałych danych z naszego zegara. Musimy jednak wcześniej zwrócić uwagę na dodatkową kwestię – otóż niektóre bajty zawierają podwójne dane – np. rejestr miesiąca zawiera także w 3 najstarszych bitach zapisane dane o dniu tygodnia. My jak na razie chcemy tylko i wyłącznie odczytać pełne informacje o godzinie i dniu miesiąca oraz miesiącu (z powodu tego, iż komórka przechowująca dane o roku może przechowywać tylko liczby z zakresu 0-3 informujące jedynie o roku przestępnym pełną obsługę roku zostawimy sobie na później). Dokonamy tego po prostu odczytując kolejne bajty z kolejnych adresów następujących po rejestrze sekund[3]. Później musimy odfiltrować oczywiście oczekiwane dane używając odpowiednich operacji bitowych. Operacji tych dokonamy w następującym programie:

```
#include <Wire.h>//dołączamy bibliotekę do obsługi I2C

byte bcd2byte(byte x){
  return (x>>4)*10+(x&0x0F);
}

void setup()
{
```

[3] Pod koniec tej części kursu umieszczam skrócony spis rejestrów w układzie PCF8583. Pełne informacje znajdziecie zawsze w karcie katalogowej układu.

```
  Wire.begin();//inicjalizujemy magistralę
  Serial.begin(9600);
}

byte s=0,s_old=255;
byte m,h,d,ms;

void loop()
{
  Wire.beginTransmission(0xA0/2);//wysyłamy adres układu, z którym chcemy
się komunikować (z powodu przesunięcia w bibliotece musimy adres podzielić
przez 2 i "odciąć" najmłodszy bit)
  Wire.send(0x02);//wysyłamy adres komórki wewnątrz układu, z której chcemy
przeprowadzić odczyt (pod adresem 0x02 znajduje się rejestr sekund)
  Wire.endTransmission();//wstrzymujemy transmisję (nadawanie)
  Wire.requestFrom(0xA0/2,5);//przygotowujemy układ od 5 bajtów danych
  s=bcd2byte(Wire.receive());//odbieramy bajt danych
  m=bcd2byte(Wire.receive());
  h=bcd2byte(Wire.receive());
  d=bcd2byte(Wire.receive()&0b00111111);//filtrujemy dodatkowo informacje o
roku
  ms=bcd2byte(Wire.receive()&0b00011111);//filtrujemy dodatkowo informacje
o dniu tygodnia
  if(s!=s_old){//jeśli minęła sekunda to aktualizujemy dane
    s_old=s;
    if(h<10)Serial.print("0");
    Serial.print(h,DEC);
    Serial.print(":");
    if(m<10)Serial.print("0");
    Serial.print(m,DEC);
    Serial.print(":");
    if(s<10)Serial.print("0");
    Serial.print(s,DEC);
    Serial.print(" ");
    if(d<10)Serial.print("0");
    Serial.print(d,DEC);
    Serial.print(".");
    if(ms<10)Serial.print("0");
    Serial.println(ms,DEC);
  }
}
```

Proste, prawda?

Cóż mamy już zatem w miarę działający zegarek… odliczający czas od włączenia zasilania. Przydałaby się zatem oczywiście możliwość zapisu czasu do naszego zegarka. Dokonamy tego wysyłając do naszego układu (po rozpoczęciu transmisji i wysłaniu adresu komórki) więcej bajtów danych. Oczywiście musimy pamiętać o konwersji danych z postaci dziesiętnej na kod BCD. Oto prosty przykład:

```
#include <Wire.h>

byte bcd2byte(byte x){
  return (x>>4)*10+(x&0x0F);
}

byte byte2bcd(byte x){
  return ((x/10)<<4)+(x%10);
```

```
}

void setup()
{
  Wire.begin();
  Serial.begin(9600);
  Wire.beginTransmission(0xA0/2);
  Wire.send(0x02);//poniżej zapisujemy dane do kolejnych komórek pamięci
(układ automatycznie zwiększa adres)
  Wire.send(byte2bcd(27));//sekundy
  Wire.send(byte2bcd(10));//minuty
  Wire.send(byte2bcd(12));//godziny
  Wire.send(byte2bcd(4));//dni
  Wire.send(byte2bcd(8));//miesiące
  Wire.endTransmission();
}

byte s=0,s_old=255;
byte m,h,d,ms;

void loop()
{
  Wire.beginTransmission(0xA0/2);
  Wire.send(0x02);
  Wire.endTransmission();
  Wire.requestFrom(0xA0/2,5);
  s=bcd2byte(Wire.receive());
  m=bcd2byte(Wire.receive());
  h=bcd2byte(Wire.receive());
  d=bcd2byte(Wire.receive()&0b00111111);
  ms=bcd2byte(Wire.receive()&0b00011111);
  if(s!=s_old){//jeśli minęła sekunda to aktualizujemy dane
    s_old=s;
    if(h<10)Serial.print("0");
    Serial.print(h,DEC);
    Serial.print(":");
    if(m<10)Serial.print("0");
    Serial.print(m,DEC);
    Serial.print(":");
    if(s<10)Serial.print("0");
    Serial.print(s,DEC);
    Serial.print(" ");
    if(d<10)Serial.print("0");
    Serial.print(d,DEC);
    Serial.print(".");
    if(ms<10)Serial.print("0");
    Serial.println(ms,DEC);
  }
}
```

Oczywiście docelowo wypadałoby sprawić, aby początkowy zapis danych był przeprowadzany na życzenie użytkownika i zapisywał dane przez niego wprowadzone, choćby w terminalu RS232, ale te kwestie pozostawiam już Wam do samodzielnego opracowania, gdyż na podstawie dotychczas poznanych informacji zadanie takie nie powinno stanowić dla Was najmniejszego problemu.

Niestety powyższy program nie wykorzystuje w pełni całych możliwości układu. Otóż możemy bardzo łatwo posłużyć się jego wyjściem INT, które domyślnie generuje przerwanie co 1s, czyli czas co jaki

chcemy odczytywać czas z zegara. Pora zatem wykorzystać to przerwanie do odczytu czasu, możemy to zrobić na przykład tak:

```
#include <Wire.h>

byte bcd2byte(byte x){
  return (x>>4)*10+(x&0x0F);
}

byte byte2bcd(byte x){
  return ((x/10)<<4)+(x%10);
}

void setup()
{
  Wire.begin();
  Serial.begin(9600);
  Wire.beginTransmission(0xA0/2);
  Wire.send(0x02);//poniżej zapisujemy dane do kolejnych komórek pamięci
(układ automatycznie zwiększa adres)
  Wire.send(byte2bcd(27));//sekundy
  Wire.send(byte2bcd(10));//minuty
  Wire.send(byte2bcd(12));//godziny
  Wire.send(byte2bcd(4));//dni
  Wire.send(byte2bcd(8));//miesiące
  Wire.endTransmission();
  pinMode(2,INPUT);
  digitalWrite(2,HIGH);
  attachInterrupt(INT0,odczyt,FALLING);
}

volatile byte s=0,s_old=255;
volatile byte m,h,d,ms;

void loop()
{
  if(s!=s_old){//jeśli minęła sekunda to aktualizujemy dane
    s_old=s;
    if(h<10)Serial.print("0");
    Serial.print(h,DEC);
    Serial.print(":");
    if(m<10)Serial.print("0");
    Serial.print(m,DEC);
    Serial.print(":");
    if(s<10)Serial.print("0");
    Serial.print(s,DEC);
    Serial.print(" ");
    if(d<10)Serial.print("0");
    Serial.print(d,DEC);
    Serial.print(".");
    if(ms<10)Serial.print("0");
    Serial.println(ms,DEC);
  }
}

void odczyt(){
  interrupts();
  Wire.beginTransmission(0xA0/2);
  Wire.send(0x02);
  Wire.endTransmission();
```

```
  Wire.requestFrom(0xA0/2,5);
  s=bcd2byte(Wire.receive());
  m=bcd2byte(Wire.receive());
  h=bcd2byte(Wire.receive());
  d=bcd2byte(Wire.receive()&0b00111111);
  ms=bcd2byte(Wire.receive()&0b00011111);
}
```

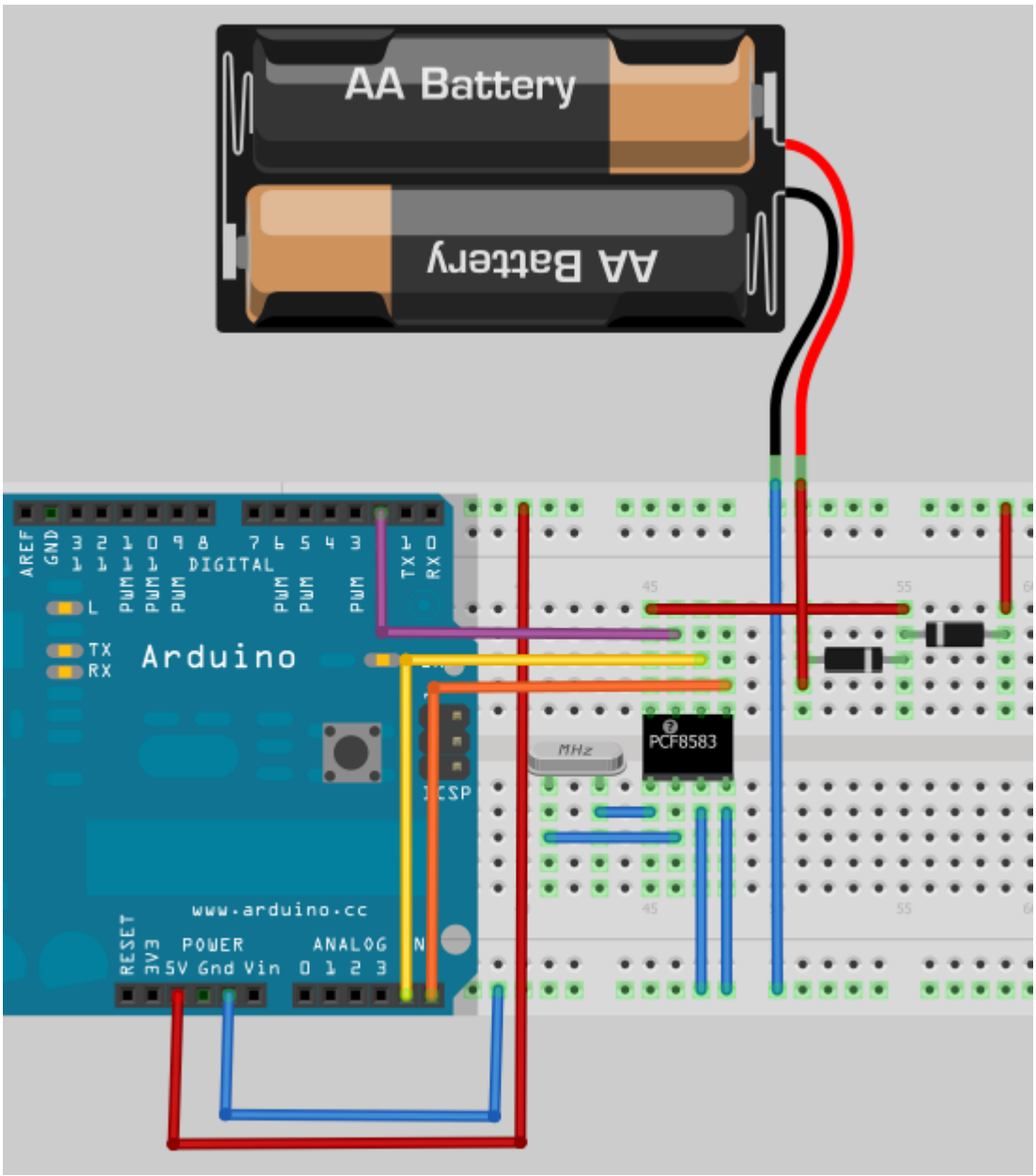
Wyjaśnienia może wymagać kolejne włączenie przerwań w przerwaniu INT0. Otóż domyślnie po wykryciu danego przerwania i podczas wykonywania jego programu blokowane są wszystkie inne przerwania, tak, aby nie zakłócać pracy wyzwolonego przerwania. W naszym przypadku jednak procedury obsługi magistrali I^2C także wykorzystują przerwania sprzętowe modułu I^2C/TWI i musimy ręcznie odblokować system przerwań. Uważajmy jednak na takie działania, gdyż jeśli np. przerwanie INT0 występowałoby tak często, że między dwoma przerwaniami nie starczyłoby czasu na wykonanie programu zawartego w danym przerwaniu, to wtedy nastąpiłoby całkowite zablokowanie pracy procesora.

Teraz przejdźmy do kwestii obsługi roku w naszym zegarze. Otóż możemy wykorzystać do tego dodatkową wolną pamięć ram dostępną pod adresami wyższymi lub równymi 0x10. Na szczęście nie będziemy musieli sami zaprzątać sobie głowy takimi sprawami, gdyż znów z pomocą przychodzi nam gotowa biblioteka do Arduino, którą specjalnie na potrzeby kursu zmodyfikowałem, aby ułatwić pracę w stosunku do oryginalnej. Myślę, że kod nie wymaga żadnego dodatkowego komentarza.

```
#include <Wire.h>
#include <PCF8583.h>

PCF8583 zegar(0xA0);

void setup(){
  Serial.begin(9600);
  zegar.day=31;
  zegar.month=12;
  zegar.year=2010;
  zegar.hour=23;
  zegar.minute=59;
  zegar.second=50;
  zegar.set_time();
}

byte old_second=255;

void loop(){
  zegar.get_time();
  if(zegar.second!=old_second){
    old_second=zegar.second;
    if(zegar.hour<10)Serial.print("0");
    Serial.print(zegar.hour,DEC);
    Serial.print(":");
    if(zegar.minute<10)Serial.print("0");
    Serial.print(zegar.minute,DEC);
    Serial.print(":");
    if(zegar.second<10)Serial.print("0");
    Serial.print(zegar.second,DEC);
    Serial.print(" ");
    if(zegar.day<10)Serial.print("0");
    Serial.print(zegar.day,DEC);
```

```
    Serial.print(".");
    if(zegar.month<10)Serial.print("0");
    Serial.print(zegar.month,DEC);
    Serial.print(".");
    Serial.println(zegar.year);
  }
}
```

Aby nasz układ zliczał czas nawet po odłączeniu zasilania od płytki możemy wyposażyć go w dodatkową baterię podtrzymującą o napięciu ok. 3V (np. popularne i łatwo dostępne CR2032, do których dostaniemy też wygodne obudowy). Jedyne co musimy dodać do naszego układu to właśnie tę baterię i 2 diody 1N4148 według rysunku:



Pamięć EEPROM

Czasem w naszych programach zajdzie konieczność przechowania danych tak, aby zostały one zachowane po odłączeniu zasilania. Może to być np. jakieś ustawienie działania programu, hasła, numery seryjne kart zbliżeniowych itp. Oczywiście można by zastosować kartę SD, jednak w wielu przypadkach 1 czy 2 GB miejsca to duży nadmiar i niepotrzebna komplikacja całego układu. Możemy zatem przechowywać takie dane w nieulotnej pamięci EEPROM. I tu do wyboru mamy 2 możliwości, albo wykorzystanie wewnętrznej pamięci EEPROM procesora (512B dla wersji m168 i 1024B dla wersji m328(p)), albo skorzystać z układu lub układów podpinanych do magistrali I^2C (np. AT24C02).

Wbudowana pamięć EEPROM

Obsługa wbudowanej pamięci EEPROM procesora jest niezwykle prosta. Przedstawia ją poniższy przykład:

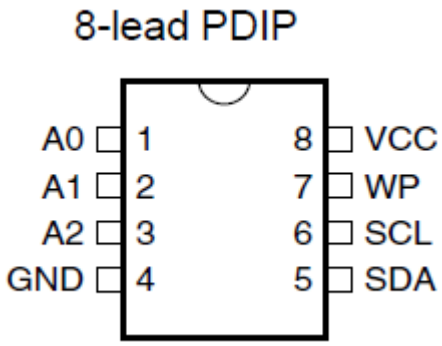
```
#include <EEPROM.h>

int odbierz(){//funkcja odbioru liczb z części 6 przystosowana do odbioru liczb typu int (bez znaku)
  byte odebrany=0;
  int liczba=0;
  do{
    if(Serial.available()>0){
      odebrany=Serial.read();
      if(odebrany!=10&&odebrany!=13){//filtrujemy znaki CR i LF (tak też ustawiamy terminal RS232)
        liczba*=10;
        liczba+=odebrany-'0';
      }
    }
  }while(odebrany!=10);//jeśli odebraliśmy znak LF to wychodzimy z pętli
  return liczba;//zwracamy odebraną liczbę
}

int address=0;
byte value=0;

void setup(){
  Serial.begin(9600);
}

void loop(){
  Serial.print("Podaj adres w EEPROM: ");
  address=odbierz();
  Serial.println(address,DEC);
  value=EEPROM.read(address);
  Serial.print("Aktualna wartosc: ");
  Serial.println(value,DEC);
  Serial.print("Podaj nowa wartosc do zapisania: ");
  value=odbierz();
  EEPROM.write(address,value);
  Serial.println(value,DEC);
  Serial.println("Wartosc zapisano");
}
```

Kod jest moim zdaniem na tyle prosty i czytelny, iż także nie wymaga dodatkowego komentarza z mojej strony. Jeśli chcemy sprawdzić działanie pamięci możemy po zapisaniu paru bajtów danych odłączyć zasilanie płytki i później po ponownym jej podłączeniu do komputera sprawdzić, że wartości

pod poszczególnymi adresami są takie, jak zapisaliśmy. Chyba nie muszę wspominać, że adres w procesorach ATMega168 musi mieścić się w zakresie 0-511, a w procesorach ATMega328(p) 0-1023. Dodatkowo chciałbym zwrócić uwagę na kwestię trwałości wbudowanej pamięci EEPROM – producent podaje dla każdej komórki minimalną trwałość 100 000[4] cykli zapisu – może wydawać się wam to duża liczba cykli, ale jeśli umieścilibyście operacje zapisu jednej komórki w jakiejś pętli to bardzo szybko moglibyście tę kromkę unicestwić – dlatego pamiętajcie o tym, aby z rozwagą korzystać z zapisu danych do pamięci EEPROM.

Zewnętrzna pamięć EEPROM na magistrali I²C (AT24Cxx)

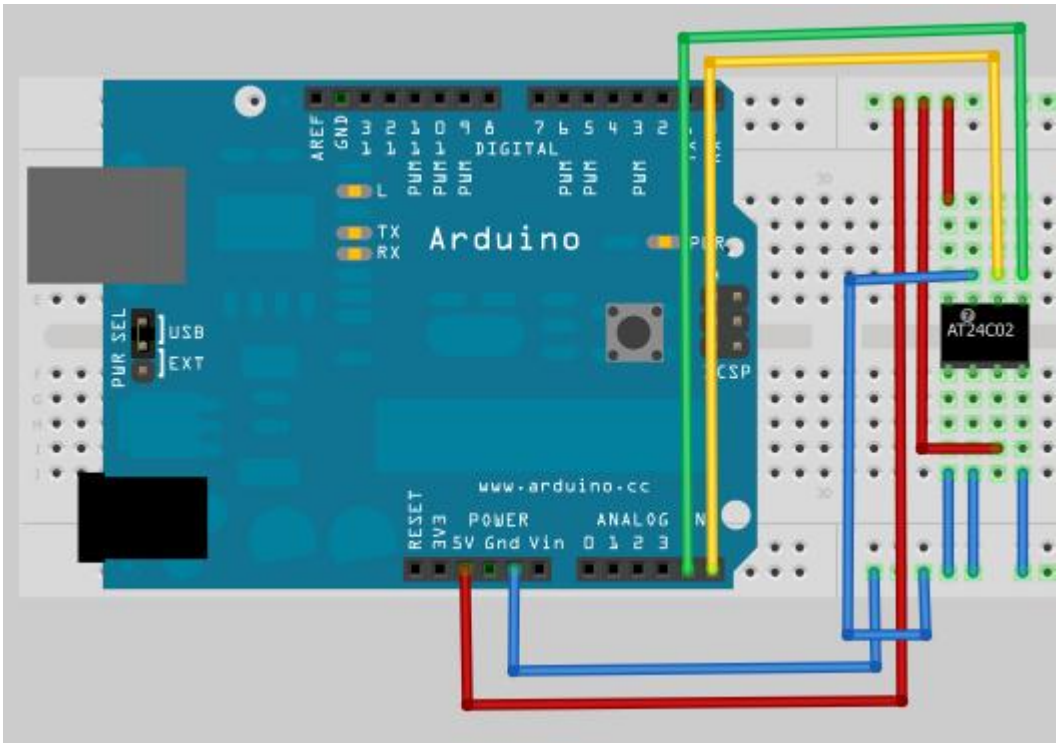
Drugą możliwością jest zastosowanie zewnętrznej pamięci EEPROM, np. AT24C01/02/04/08/16, które mają odpowiednio pojemności 128/256/512/1024/2048B. Jakie zalety mają jednak te pamięci w porównaniu z wbudowaną pamięcią EEPROM? Po pierwsze pozwalają one rozszerzyć dodatkowo pamięć EEPROM, jeśli ta w procesorze nie będzie nam wystarczać. Ponadto pamięci z tej serii posiadają większą trwałość (deklarowana minimalna trwałość to 1000 000 cykli) niż pamięć w procesorze. Poznajmy teraz rozkład wyprowadzeń tych układów oraz funkcje poszczególnych pinów:



PIN	Funkcja
A0	Piny umożliwiające ustalanie adresu układu – szerszy opis w dalszej części kursu
A1	
A2	
GND	Masa układu
SDA	Linia danych magistrali I²C
SCL	Linia zegarowa magistrali I²C
WP	Pin sprzętowej ochrony przed zapisem – zwarty do GND – brak ochrony, zwarty do VCC – blokada zapisu do układu
VCC	Zasilanie układu

Podłączymy zatem naszą pamięć AT24C02 do procesora w następujący sposób, podłączając piny A0 i A1 do masy, a pin A2 do VCC:

[4] oczywiście jest to minimalna gwarantowana trwałość w ekstremalnych warunkach, co oznacza, że komórka może wytrzymać w rzeczywistości więcej cykli



Zaglądamy jeszcze do datasheetu pamięci w celu sprawdzenia jej adresu na magistrali I^2C:

1K/2K	1	0	1	0	A_2	A_1	A_0	R/W
	MSB							LSB
4K	1	0	1	0	A_2	A_1	P0	R/W
8K	1	0	1	0	A_2	P1	P0	R/W
16K	1	0	1	0	P2	P1	P0	R/W

Widzimy bez problemu że adres naszego układu (zgodnie z połączeniami linii A0-A2 z rysunku) będzie miał adres bazowy 0b10101000. Do adresowania komórki z której ma zostać przeprowadzany odczyt lub zapis wykorzystywany jest w przypadku układów AT24C01 i 02 tylko pierwszy bajt przesłany po rozpoczęciu transmisji (dokładnie tak jak w przypadku zegara PCF8583). W przypadku układów z pamięcią większą niż 256 bajtów (modele AT24C04/08/16) do adresowania wykorzystywana jest także część adresu kosztem usunięcia możliwości zmiany jego bitów za pomocą linii adresowych (oznaczone jako P0, P1 i P2). Możemy zatem wtedy interpretować taki układ jako większą ilość układów AT24C02 zamkniętych w jednej kostce i posiadających kolejne adresy. UWAGA! Zwróćcie uwagę na to, że niektóre pamięci (z wymienionych wcześniej: AT24C16) lub niektóre ustawienia

pinów adresowych mogą powodować konflikt adresów[5] z innymi układami, w szczególności z zegarem PCF8583, który w takim przypadku nie będzie mógł być podłączony równolegle z taką pamięcią. Zatem na początek, bez zbędnych wstępów napiszmy prosty program zapisujący podobnie jak poprzednio dane, tyle, że do pamięci EEPROM w układzie AT24C02:

```
#include <Wire.h>

int odbierz(){//funkcja odbioru liczb z części 6 przystosowana do odbioru
liczb typu int (bez znaku)
  byte odebrany=0;
  int liczba=0;
  do{
    if(Serial.available()>0){
      odebrany=Serial.read();
      if(odebrany!=10&&odebrany!=13){//filtrujemy znaki CR i LF (tak też
ustawiamy terminal RS232)
        liczba*=10;
        liczba+=odebrany-'0';
      }
    }
  }while(odebrany!=10);//jeśli odebraliśmy znak LF to wychodzimy z pętli
  return liczba;//zwracamy odebraną liczbę
}

const int at24c02_address=0b10101000/2;//adres układu piny A0 i A1 podpięte
do masy, pin A2 podpięty do VCC

void at24c02_write(byte address, byte value){
  Wire.beginTransmission(at24c02_address);
  Wire.send(address);//adres komórki
  Wire.send(value);//wartość do zapisania
  Wire.endTransmission();
}

byte at24c02_read(byte address){
  Wire.beginTransmission(at24c02_address);
  Wire.send(address);//adres komórki
  Wire.endTransmission();
  Wire.requestFrom(at24c02_address,1);
  return Wire.receive();//odczyt wartości
}

byte address=0;
byte value=0;

void setup(){
  Serial.begin(9600);
  Wire.begin();
}

void loop(){
  Serial.print("Podaj adres w EEPROM: ");
  address=odbierz();
  Serial.println(address,DEC);
  value=at24c02_read(address);
  Serial.print("Aktualna wartosc: ");
  Serial.println(value,DEC);
```

[5] posiadanie takiego samego adresu przez większą liczbę układów

```
  Serial.print("Podaj nowa wartosc do zapisania: ");
  value=odbierz();
  at24c02_write(address,value);
  Serial.println(value,DEC);
  Serial.println("Wartosc zapisano");
}
```

Do naszej pamięci dane możemy zapisywać i odczytywać też większymi partiami, tak jak kolejne rejestry zegara PCF8583, musimy jednak pamiętać, że przy zapisie musimy po każdym bajcie odczekać 5ms, gdyż musimy dać pamięci czas na zapisanie danych. Seryjne odczytywanie danych realizuje następujący program (niestety z powodu takiej a nie innej konstrukcji biblioteki seryjny zapis do pamięci nie jest możliwy seriami, gdyż biblioteka korzysta z bufora utrudniającego generowanie opóźnień, stąd też funkcja wykonuje określoną ilość pojedynczych zapisów):

```
#include <Wire.h>

int odbierz(){//funkcja odbioru liczb z części 6 przystosowana do odbioru
liczb typu int (bez znaku)
  byte odebrany=0;
  int liczba=0;
  do{
    if(Serial.available()>0){
      odebrany=Serial.read();
      if(odebrany!=10&&odebrany!=13){//filtrujemy znaki CR i LF (tak też
ustawiamy terminal RS232)
        liczba*=10;
        liczba+=odebrany-'0';
      }
    }
  }while(odebrany!=10);//jeśli odebraliśmy znak LF to wychodzimy z pętli
  return liczba;//zwracamy odebraną liczbę
}

const int at24c02_address=0b10101000/2;//adres układu piny A0 i A1 podpięte
do masy, pin A2 podpięty do VCC

void at24c02_write_buf(byte address, byte values[], int n){
  for(int i=0;i<n;i++){
      Wire.beginTransmission(at24c02_address);
      Wire.send(address+i);//adres komórki
      Wire.send(values[i]);
      Wire.endTransmission();//kończymy transmisję i wysyłamy dane
      delay(5);
  }
}

void at24c02_read_buf(byte address, byte values[], int n){
  Wire.beginTransmission(at24c02_address);
  Wire.send(address);//adres komórki
  Wire.endTransmission();
  Wire.requestFrom(at24c02_address,n);
  for(int i=0;i<n;i++)values[i]=Wire.receive();
}

byte address=0;
byte bufor[10];

void setup(){
```

```
  Serial.begin(9600);
  Wire.begin();
}

void loop(){
  Serial.print("Podaj adres w EEPROM: ");
  address=odbierz();
  Serial.println(address,DEC);
  Serial.print("10 kolejnych bajtow danych: ");
  at24c02_read_buf(address,bufor,10);
  for(int i=0;i<10;i++){
    Serial.print(bufor[i],DEC);
    Serial.print(" ");
  }
  Serial.println();
  Serial.print("Podaj 10 wartosci do zapisania: ");
  for(int i=0;i<10;i++){
    bufor[i]=odbierz();
    Serial.print(bufor[i],DEC);
    Serial.print(" ");
  }
  Serial.println();
  at24c02_write_buf(address,bufor,10);
  Serial.println("Wartosci zapisano");
}
```

Pamięci większe niż 256B[6]

Jak już wspomniałem pamięci o rozmiarach większych niż 256B (czyli 2kb = 2 kida bity = 2048 bitów) do adresowania kolejnych komórek (zapisanych na tzw. stronach pamięci) wykorzystywane są bity z adresu, zgodnie z przedstawioną wcześniej tabelką. Przedstawię poniżej prosty program zapisujący pojedyncze komórki do pamięci AT24C16, który korzysta z operacji bitowych, poznanych przez Was na początku tej lekcji.

```
#include <Wire.h>

int odbierz(){//funkcja odbioru liczb z części 6 przystosowana do odbioru
liczb typu int (bez znaku)
  byte odebrany=0;
  int liczba=0;
  do{
    if(Serial.available()>0){
      odebrany=Serial.read();
      if(odebrany!=10&&odebrany!=13){//filtrujemy znaki CR i LF (tak też
ustawiamy terminal RS232)
        liczba*=10;
        liczba+=odebrany-'0';
      }
    }
  }while(odebrany!=10);//jeśli odebraliśmy znak LF to wychodzimy z pętli
  return liczba;//zwracamy odebraną liczbę
```

[6] Mówię to o pamięciach z przedstawionej wcześniej serii, jednak istnieją też inne wersje podobnych pamięci, w których adresowanie wyższych adresów odbywa się np. za pomocą wysłania po rozpoczęciu transmisji 2 bajtów adresu. Szczegóły zawsze znajdziemy w kartach katalogowych (datasheet) układów, do których lektury zawsze namawiam przed użyciem jakiegokolwiek układu – w przeciwnym razie narażamy się na problemy lub nawet uszkodzenia sprzętu i elementów elektronicznych.

```
}

const int at24c16_address=0b10100000/2;//adres układu - podłączenie pinów
A0-A2 nie ma znaczenia - powinniśmy je podpiąć do masy

void at24c16_write(int address, byte value){
      Wire.beginTransmission(at24c16_address|(address>>8));//adres układu +
adres strony (przesuwamy adres o 8 bitów, aby otrzymać jedynie 3 najstarsze
jego bity i wstawiamy je do adresu bazowego)
      Wire.send(address&0xFF);//adres komórki (najmłodsze 8 bitów
odfiltrowane za pomocą operacji bitowych)
      Wire.send(value);
      Wire.endTransmission();//kończymy transmisję i wysyłamy dane
}

byte at24c16_read(int address){
  Wire.beginTransmission(at24c16_address|(address>>8));
  Wire.send(address&0xFF);//adres komórki
  Wire.endTransmission();
  Wire.requestFrom(at24c16_address|(address>>8),1);
  return Wire.receive();
}

int address=0;
byte value=0;

void setup(){
  Serial.begin(9600);
  Wire.begin();
}

void loop(){
  Serial.print("Podaj adres w EEPROM: ");
  address=odbierz();
  Serial.println(address,DEC);
  Serial.print("Wartosc pod adresem: ");
  value=at24c16_read(address);
  Serial.println(value,DEC);

  Serial.print("Podaj wartosc do zapisania: ");
  value=odbierz();
  Serial.println(value,DEC);
  at24c16_write(address,value);
  Serial.println("Wartosc zapisano");
}
```

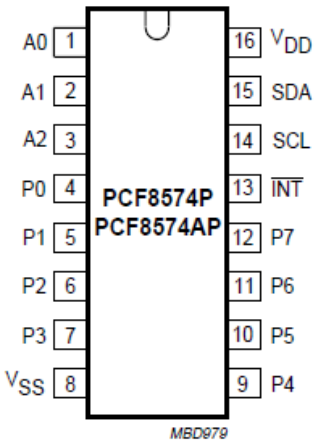
Na tym zakończymy już omawianie pamięci szeregowych EEPROM na magistrali I^2C, a przejdziemy do nieco ciekawszego zajęcia a mianowicie:

Ekspander portu I/O PCF8574

Kolejnym i ostatnim urządzeniem na magistrali I^2C jakie omówię szczegółowo w tym kursie jest wspomniany przeze mnie ekspander portów I/O PCF8574. Układ ten umożliwia rozszerzenie ilości pinów wejściowo/wyjściowych (czyli takich jak nasze piny cyfrowe D0-D13) z wykorzystaniem magistrali I^2C. Układ ten działa jednak w nieco inny sposób niż poprzednie. Otóż każdy wysłany do niego bajt jest przekazywany do rejestrów wyjściowych i jeśli dany bit jest ustawiony na 0, to

odpowiadający mu pin jest „konfigurowany" jako wyjście i podawany jest na nim stan niski. Z kolei jeśli bit jest ustawiony na 1 to odpowiadający mu pin jest „konfigurowany" jako wejście z podciąganiem. Każdy odczytany bajt zawiera informacje o aktualnym stanie wszystkich wyprowadzeń układu (zarówno tych skonfigurowanych jako wejścia jak i wyjścia). Daje to zatem możliwość sterowania urządzeniami stanem niskim (np. diodami LED) lub odczytu przycisków zwierających do masy. Dodatkowo każda zmiana stanu pinów skonfigurowanych jako wejście (spowodowana np. przyciśnięciem przycisku) powoduje wygenerowanie przerwania na odpowiedniej linii układu, przez co możemy łatwo rozwiązać sprawdzanie stanu podłączonych urządzeń.

Teraz tradycyjnie czas na układ wyprowadzeń:



PIN	Funkcja
A0,A1,A2	Piny służące do zmiany adresu układu
P0,...,P7	Piny wejściowo/wyjściowe
Vss	Masa układu
/INT	Przerwanie generowane przez układ
SCL, SDA	Odpowiednie linie magistrali I^2C
Vdd	Zasilanie układu

Ale tym razem będziecie musieli się sami trochę napracować i podłączyć układ do procesora zupełnie samodzielnie. Oprócz niezbędnych połączeń (zasilanie, linie danych) podłączcie na czas ćwiczeń wszystkie linie adresowe do masy, linię /INT do pinu D2 naszej płytki (przerwanie INT0), do portów P0, P1, P4 podłączcie przez rezystory 330Ω 3 diody LED (aktywowane stanem niskim podanym z układu PCF8574), a do pinów P2 i P3 podłączcie przyciski zwierające do masy.

Przed przystąpieniem do prezentacji przykładowego programu podaję jeszcze tabelkę zawierającą wyjaśnienie adresowania układu:

a. PCF8574. b. PCF8574A.

Wyjaśnienia może wymagać to, dlaczego podaję 2 tabelki – otóż omawiany przez nas układ występuje w 2 wersjach – z literką „A" w modelu lub bez niej. Kości z literką „A" w modelu są adresowane według tabelki b., zaś te bez literki „A" według tabelki a. A teraz kolej na program:

```
#include <Wire.h>

const int pcf8574_address=0b01110000/2;//adres układu - w moim przypadku
wersja z literką "A" - PCF8574AP

void pcf8574_write(byte value){
      Wire.beginTransmission(pcf8574_address);//adres układu
      Wire.send(value);//dane do wystawienia na piny
      Wire.endTransmission();
}

byte pcf8574_read(){
  Wire.requestFrom(pcf8574_address,1);
  return Wire.receive();//odczytujemy dane z układu
}

void setup(){
  Wire.begin();
  pinMode(2,INPUT);//konfiguracja pinu przerwania INT0
  digitalWrite(2,HIGH);
  attachInterrupt(INT0,klawisze,FALLING);//przypisanie do przerwania INT0
funkcji
}

volatile byte state=0b11111111;//zmienna przechowująca aktualny stan pinów
układu

void loop(){
  state&=~(1<<0);//zapalamy diodę podpiętą pod P0
  state|=(1<<1);//gasimy diodę podpiętą pod P1
  pcf8574_write(state);//wysyłamy dane do układu
  delay(1000);
  state&=~(1<<1);//zapalamy diodę podpiętą pod P1
  state|=(1<<0);//gasimy diodę podpiętą pod P0
  pcf8574_write(state);//wysyłamy dane do układu
  delay(1000);
}

void klawisze(){
  interrupts();//aktywujemy system przerwań, z którego korzysta biblioteka
Wire
  byte value = pcf8574_read();//odczytujemy stan pinów
  if(!(value&(1<<2))){//jeśli pin P2 został zwarty do masy
    state&=~(1<<4);//zapalamy diodę podpiętą pod P4
    pcf8574_write(state);//wysyłamy dane do układu
  }
  if(!(value&(1<<3))){//jeśli pin P3 został zwarty do masy
    state|=(1<<4);//gasimy diodę podpiętą pod P4
    pcf8574_write(state);//wysyłamy dane do układu
  }
}
```

O ile sama idea działania programu i odczytu/zapisu danych nie wymaga chyba żadnego wyjaśnienia, to na pewno muszę szerzej omówić operacje wykonywane z użyciem operatorów bitowych służące do sprawdzania stanów przycisków i zapalania/gaszenia diod.

Na wstępie powiem, że wszystkie operacje mają za zadanie jedynie ustawić, skasować lub sprawdzić stan określonego przez nas bitu.

Zatem po kolei:

1. Ustawianie bitu (np. `state|=(1<<4);`)
   Na początku następuje przesunięcie bitowe jedynki w zapisie binarnym

0	0	0	**0**	0	0	0	1

   o 4 bity w wyniku czego otrzymujemy:

0	0	0	**1**	0	0	0	0

   Następnie wartość ta jest łączona operatorem alternatywy z aktualną zawartością zmiennej state, np.:

1	1	1	**0**	1	1	1	0

   W wyniku czego zostaje ustawiony pożądany przez nas bit a wynik tej operacji jest zapisywany do zmiennej state:

1	1	1	**1**	1	1	1	0

2. Kasowanie bitu (np. `state&=~(1<<1);`)
   Na początku następuje przesunięcie bitowe jedynki w zapisie binarnym

0	0	0	0	0	0	**0**	1

   o 1 bit w wyniku czego otrzymujemy:

0	0	0	0	0	0	**1**	0

   Kolejnym krokiem jest zanegowanie tej wartości:

1	1	1	1	1	1	**0**	1

   Następnie wartość po negacji jest łączona operatorem koniunkcji z aktualną zawartością zmiennej state, np.:

1	1	1	1	1	1	**1**	0

   W wyniku czego pożądany bit zostaje skasowany, a wynik operacji jest zapisywany do zmiennej state:

1	1	1	1	1	1	**0**	0

3. Sprawdzanie czy dany bit jest ustawiony (np. `if(value&(1<<2))`)
   Na początku następuje przesunięcie bitowe jedynki w zapisie binarnym

0	0	0	0	0	**0**	0	1

   o 2 bity w wyniku czego otrzymujemy:

0	0	0	0	0	**1**	0	0

   Następnie łączymy tę wartość z wartością zmiennej value np.:

1	1	0	1	1	1	0	1

w wyniku czego otrzymujemy:

0	0	0	0	0	1	0	0

czyli wartość różną od zera, która jest interpretowana jako logiczna prawda. Jeśli zaś na odpowiedniej pozycji w zmiennej value znajdowałoby się 0, np.:

1	1	0	1	1	0	0	1

otrzymalibyśmy:

0	0	0	0	0	0	0	0

Czyli 0 interpretowane jako logiczny fałsz.

4. Sprawdzanie czy dany bit jest skasowany (np. `if(!(value&(1<<2)))`)
   Postępujemy tu dokładnie tak samo jak w punkcie 3, tyle ze później wynik całej operacji poddajemy logicznemu zaprzeczeniu – czyli logiczny fałsz (0) zmieniamy naprawdę, a logiczną prawdę (wartość większą do zera) zamieniamy na logiczny fałsz – dokonuje tego operator wykrzyknika „!”.

Polecam wszystkim szczególne przeanalizowanie powyższych metod operacji bitowych, gdyż jeśli w przyszłości będziecie chcieli przejść do programowania w czystym języku C to tego typu operacje stają się nieodzowne w przypadku jakiegokolwiek operowania na rejestrach procesora.

Moduł przekaźnikowy

UWAGA! Przed podłączeniem modułu przekaźnikowego koniecznie podłącz do płytki zewnętrzne zasilanie 12V. W przeciwnym razie płytka i/lub komputer mogą ulec uszkodzeniu.

UWAGA! Praca z wysokimi napięciami (~230V) stwarza zagrożenie dla Twojego zdrowia i życia. Wszystkie połączenia koniecznie wykonuj przy odłączonym zasilaniu. Jeśli nie masz doświadczenia w pracy z wysokimi napięciami koniecznie wykonuj wszystkie czynności pod okiem doświadczonej osoby. Zachowaj szczególną ostrożność, gdyż moduł posiada nieosłonięte złącza na których może występować po podłączeniu zewnętrznych urządzeń wysokie napięcie. Autor nie bierze odpowiedzialności za wszelkie szkody wynikłe podczas pracy z wysokimi napięciami.

Moduł przekaźnikowy oparty jest o układ PCF8574 (choć zawsze warto sprawdzić, czy nie trafiliśmy na wersję PCF8574A), który z pomocą odpowiednich rezystorów podciągających (zapewniających stan wysoki) steruje poprzez układ ULN2003 4 przekaźnikami zasilanymi napięciem 12V, a mogącymi sterować urządzeniami nawet do 230V. Pin A2 układu PCF8574 jest pozostawiony jako niepodłączony, zatem przyjmuje stan wysoki (A2 = 1). Piny A1 i A0 wyprowadzone są do zworek, którymi możemy ustawiać dodatkowo 2 bity adresu, więc możemy podłączyć nawet do 4 takich modułów. W swoim przykładzie założyłem obie zworki w pozycji 0, dzięki czemu układ ma adres 0b01001000. Przekaźniki są podłączone odpowiednio: Q4 do P0, Q3 – P1, Q2 – P2 i Q1 – P3, w programie zdefiniowałem odpowiednie identyfikatory ułatwiające pracę z modułem. Poniżej przedstawiam prosty program załączający po kolei wszystkie przekaźniki, a później je wyłączający.

```
#include <Wire.h>
```

```
#define Q4 0
#define Q3 1
#define Q2 2
#define Q1 3

byte state=0;
const int relay_address=0b01001000/2;

void setup(){
  Wire.begin();
  state=0;//wyłączenie wszystkich przekaźników
  Wire.beginTransmission(relay_address);
  Wire.send(state);
  Wire.endTransmission();
  delay(1000);
}

void loop(){
  state|=(1<<Q4);//włączamy przekaźnik 4
  Wire.beginTransmission(relay_address);
  Wire.send(state);
  Wire.endTransmission();
  delay(1000);
  state|=(1<<Q3);//włączamy przekaźnik 3
  Wire.beginTransmission(relay_address);
  Wire.send(state);
  Wire.endTransmission();
  delay(1000);
  state|=(1<<Q2);//włączamy przekaźnik 2
  Wire.beginTransmission(relay_address);
  Wire.send(state);
  Wire.endTransmission();
  delay(1000);
  state|=(1<<Q1);//włączamy przekaźnik 1
  Wire.beginTransmission(relay_address);
  Wire.send(state);
  Wire.endTransmission();
  delay(1000);

  state&=~(1<<Q4);//wyłączamy przekaźnik 4
  Wire.beginTransmission(relay_address);
  Wire.send(state);
  Wire.endTransmission();
  delay(1000);
  state&=~(1<<Q3);//wyłączamy przekaźnik 3
  Wire.beginTransmission(relay_address);
  Wire.send(state);
  Wire.endTransmission();
  delay(1000);
  state&=~(1<<Q2);//wyłączamy przekaźnik 2
  Wire.beginTransmission(relay_address);
  Wire.send(state);
  Wire.endTransmission();
  delay(1000);
  state&=~(1<<Q1);//wyłączamy przekaźnik 1
  Wire.beginTransmission(relay_address);
  Wire.send(state);
  Wire.endTransmission();
  delay(1000);
}
```

Dodatek – skrócony opis rejestrów układu PCF8583

Adres (HEX)	Opis rejestru
00	Rejestr kontrolny
01	Setne części sekundy
02	Sekundy
03	Minuty
04	Godziny
05	Rok/dzień miesiąca (rok zapisany w 2 najwyższych bitach)
06	Dzień tygodnia/miesiąc (dzień tygodnia zapisany w 3 najwyższych bitach)
07	Rejestr Timera
08	Rejestr kontrolny alarmu
09	Setne części sekundy alarmu
0A	Sekundy alarmu
0B	Minuty alarmu
0C	Godziny alarmu
0D	Dzień miesiąca alarmu
0E	Miesiąc alarmu
0F	Timer alarmu
10	Wolna pamięć RAM
…	
FF	

Dokładne opisy wszystkich rejestrów oraz ich poszczególnych bitów znajdują się w karcie katalogowej układu PCF8583.