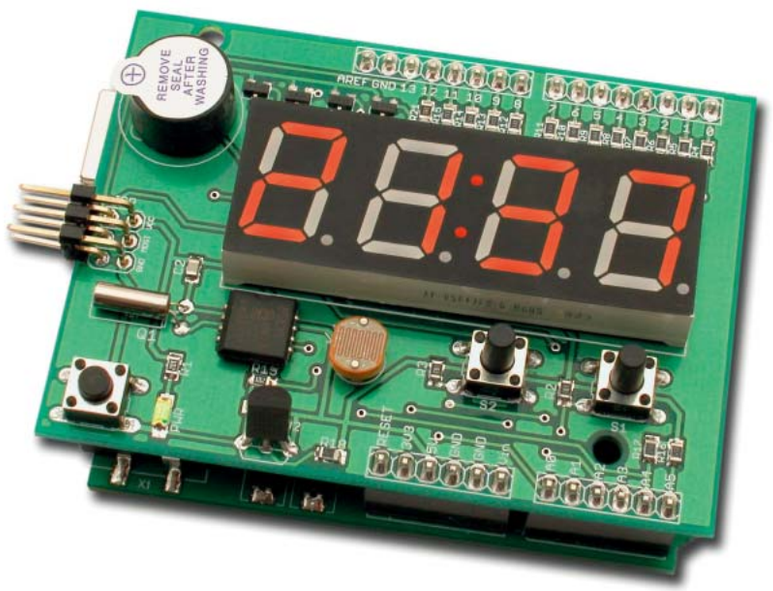
Klocki dla Arduino (2)

Procedury obsługi wybranych modułów dodatkowych dla Arduino – wyświetlacz LED i zegar PCF8583

Arduino jest nie tylko świetną platformą prototypową. Ogromy wybór modułów dodatkowych umożliwia budowanie funkcjonalnych urządzeń. Unika się przy tym konieczności zaprojektowania płytki drukowanej i jej montażu. W artykule prezentujemy kilka „klocków", z których można poskładać funkcjonalne urządzenia oraz metody ich obsługi programowej.



„Klocki" – kanapki dla Arduino, oferowane przez wielu niezależnych producentów oraz do samodzielnego wykonania, których dokumentacja jest dostępna w Internecie, pozwalają na szybkie wykonanie zegara, woltomierza, układu sterującego i wielu, wielu innych. Przyjrzyjmy się przykładom programów wykonanych dla Arduino Duemilanove oraz modułów opisywanych w Elektronice Praktycznej.

Obsługa wyświetlacza LED

7-segmentowe wyświetlacze LED wydają się być ponadczasowe. Pomimo tego, że w handlu są dostępne doskonałe wyświetlacze LCD o nieporównywalnych możliwościach, to jednak wyświetlacze LED są nadal chętnie używane w wielu aplikacjach, w których jest wystarczające wyświetlanie liczb i nieskomplikowanych komunikatów. W tym przykładzie użyjemy modułu AVT-1616 w wersji 2, który jest wyposażony nie tylko w 4-pozycyjny, 7-segmentowy wyświetlacz LED, ale również termometr 1-Wire, zegar czasu rzeczywistego PCF8583, fotorezystor, dwa przyciski ogólnego przeznaczenia oraz przycisk zerowania. Wyposażenie płytki predysponuje ją przede wszystkim do wykonania zegara – zbudujemy go w tym przykładzie.

Na początku przyjmijmy pewne założenia. Czas będzie wskazywany na wyświetlaczu LED, a odmierzany przez układ scalony zegara RTC PCF8583. Na początek, dla uproszczenia, przyjmijmy, że nie będziemy używali kalendarza, a jedynie liczników godzin, minut i sekund. Nie będziemy też używali termometru 1-Wire i fotorezystora do regulowania natężenia oświetlenia – zajmiemy się nimi w następnym przykładzie. Nastawy czasu zegara oraz przełączanie trybów będzie wykonywane za pomocą przycisków SW1 (pierwszy od prawej) i SW2 (drugi od prawej). Podczas wyświetlania godziny dwukropek będzie migotał, a gdy będą wyświetlane sekundy lub godziny/minuty, dwukropek będzie świecił się nieprzerwanie.

W tym przykładzie wykonamy program zegara realizujący podstawowe funkcje, działający zgodnie z podanymi założeniami. W kolejnym artykule opiszemy, w jaki sposób wykorzystać pozostałe peryferia i wykonamy nieco bardziej skomplikowany.

Wyświetlacz LED na płytce AVT1616 ma 4 cyfry. Ich katody są wspólne i dołączone do wyprowadzeń portu PORT D, natomiast anody są rozdzielone i sterowane za pomocą wyprowadzeń 0…3 portu PORT B. Poziomem aktywnym załączającym zasilanie anody i powodującym świecenie segmentu jest poziom niski. Na przykład, jeśli chcemy załączyć świecenie segmentu „A" pierwszej cyfry, należy wyzerować wyprowadzenia PORT D0 oraz PORTD B0 (będziemy je w skrócie nazywali PD0 i PB0). Wyzerowanie wyprowadzenia PB1, ustawienie PB0 oraz pozostawienie PD0 bez zmian powoduje zaświecenie się segmentu „A" drugiej cyfry itd. Taka technika wyświetlania nosi nazwę multipleksowania i wymaga, aby wyświetlacz był „przemiatany" z taką częstotliwością, aby ludzkie oko nie zauważyło migotania – zwykle 50 Hz lub więcej.

Dwukropek ma własną anodę oznaczoną na schemacie „UC". Doprowadzono do niej napięcie +5 V, natomiast katoda za pośrednictwem rezystora ograniczającego prąd jest dołączona do PB4. Łatwo domyślić się, że w takiej sytuacji wystarczy poziom niski na wyprowadzeniu PB4, aby dwukropek zaświecił się.

Układ scalony zegara RTC PCF8583 jest dobrze znany Czytelnikom EP. Ma on interfejs I²C i jest dołączony do wyprowadzeń PC5 (SCL) i PC4 (SDA) mikrokontrolera płytki Arduino. Przyciski są dołączone do PC1 (SW1) oraz PC2 (SW2) i zasilane przez rezystory zamontowane na płytce. Ich przyciśnięcie powoduje zwarcie wyprowadzenia do masy, co łatwo testuje się w programie. Na płytce jest też zamontowany piszczek, którego można użyć np. jako sygnalizatora alarmu. Steruje się nim za pomocą wyprowadzenia PB5, którego wyzerowanie powoduje głośną sygnalizację dźwiękową.

Maszyna stanów

Długo zastanawiałem się na tym, w jaki sposób wykonać funkcję obsługi wyświetlacza. Zwykle robiłem ją z użyciem procedu-

Listing 1. Uproszczony wygląd pętli głównej programu obsługi zegara

```
//główna pętla programu
void loop()
{
/*program ma 6 stanów, pętla jest wykonywana nieskończenie,
więc można użyć instrukcji „for”, która będzie wykonywana
w nieskończonej pętli poprawnie numerując stany:
  - stan 0: obsługa klawiszy i nastaw,
  - stan 1: odczyt zegara RTC,
  - stan 2...5: wyświetlanie poszczególnych cyfr.
  Numery stanów zawiera zmianna „numer_stanu”, są one
  rozpatrywane w klauzuli switch-case.
*/
  for (numer_stanu = 0; numer_stanu < 6; numer_stanu++)
  {
    switch (numer_stanu)
    {
//obsługa klawiszy
      case 0:
        .
        .
        break
//odczyt zegara
      case 1:
        .
        .
        break
//wyświetlenie cyfry 1 (pierwsza od prawej)
      case 2:
        wyswietl_LED(1);
        break;
//wyświetlenie cyfry 2
      case 3:
        wyswietl_LED(2);
        break;
//wyświetlenie cyfry 3
      case 4:
        wyswietl_LED(3);
        break;
//wyświetlenie cyfry 4
      case 5:
        wyswietl_LED(4);
        break;
//wyświetlenie cyfry 4 (pierwsza od lewej)
      default:
        break;
    }
  }
}
```

ry obsługi przerwania któregoś z timerów, co było bardzo wygodne. Wystarczy wywoływać ją z pewną częstotliwością, pobierać w niej zawartość bufora do wyświetlania i wyświetlać znak po znaku. Zapewnia to „płynność” obsługi i jednakową jasność świecenia wszystkich znaków (przy jednakowym natężeniu prądu płynącego przez segmenty zależy ona przede wszystkim od czasu świecenia cyfry). Jednak Arduino IDE w swojej podstawowej wersji, bez bibliotek rozszerzeń, nie pozwala na użycie przerwań timerów do czegoś innego, niż odmierzanie opóźnień w funkcji *delay* i podobnych. Dlatego jako swego rodzaju ćwiczenie postanowiłem zastosować nieco zapomnianą technika programowania – utworzyć w programie maszynę stanów.

Nie wdając się nadmiernie w szczegóły można powiedzieć, że maszyna stanów przyjmuje sekwencyjnie określone stany, w których są realizowane pewne czynności. Na potrzeby tego przykładu programowania wykonałem procedurę, która przyjmuje 6 stanów, ponumerowanych od 0 do 5 i umieściłem ją w pętli głównej programu. Uproszczona wersję tej procedury zamieszczono na **listingu 1**. Ze względu na obsługę różnych wariantów wyświetlania jest ona dosyć obszerna – jej pełną wersję można znaleźć w materiałach dodatkowych, natomiast tu pokazano tylko tyle, aby zrozumieć zasadę działania.

Za numerowanie i przez to zmianę stanów jest odpowiedzialna pętla *for(numer_stanu = 0; numer_stanu < 6; numer_stanu++);*. Jest ona wykonywana w nieskończonej pętli *loop* przypominającej pętlę *while (1)* znaną z języka C, więc za każdym razem, po zakończeniu iteracji, jest wywoływana na nowo i zmienna *numer_stanu* ponownie przyjmuje wartości z zakresu 0…5. Poszczególnym wartościom zmiennej *numer_stanu* odpowiada czynność wykonywana przez program obsługi. I tak, gdy zmienna ma wartość:

- 0, to są odczytywane przyciski,
- 1, to jest odczytywany licznik czasu zegara RTC,
- 2, to jest wyświetlana pierwsza cyfra od lewej,
- 3, to jest wyświetlana druga cyfra od lewej,
- 4, to jest wyświetlana trzecia cyfra od lewej,
- 5, to jest wyświetlana czwarta cyfra od lewej.

Tu pętla *for* kończy pracę i *loop* wymusza jej ponowny przebieg. W ten sposób stany są zmieniane cyklicznie i nieprzerwanie. Taka implementacja maszyny stanów powoduje, że CPU spędza w poszczególnych stanach tyle czasu, ile potrzebuje – maszyna pracuje asynchronicznie, ponieważ czasy trwania poszczególnych stanów są niezdefiniowane, a przez to również czas przejścia od stanu do stanu. Może to wpłynąć na jasność świecenia poszczególnych cyfr. Przyjrzyjmy się jednak temu, co ma do wykonania nasza maszyna.

Stan numer 0 to obsługa przycisków. Przy normalnym użytkowaniu zegara, mikrokontroler szybko „dowie się” za pomocą instrukcji *if*, że żaden z przycisków nie jest wciśnięty i przejdzie do kolejnego stanu. Jeśli będziemy w trybie ustawiania zegara, to musimy zgodzić się na pewne kompromisy, ale prędkość obsługi przycisków i tak nadal będzie bardzo duża.

Stan numer 1 odpowiada za odczyt liczników czasu zegara RTC. Przy stosunkowo dużej szybkości pracy interfejsu I²C odczyt zegara zajmuje bardzo mało czasu i jego wpływ na świecenie cyfr, podobnie jak obsługi przycisków, jest pomijalnie mały.

Stany numer 2…5 odpowiadają za obsługę wyświetlania poszczególnych cyfr. Jest ono realizowane za pomocą tej samej funkcji o nazwie *wyświetl_LED()*, której argumentem jest numer wyświetlanej cyfry. Samą funkcję umieszczono na **listingu 2**. W tym momencie dosyć jest zauważyć, że na jej końcu dodano instrukcję *delay(5)*, której zadaniem jest wydłużenie czasu wyświetlania każdej cyfry. Można zaryzykować twierdzenie, że czasy obsługi stanów 0 i 1 jest pomijalnie mały w porównaniu z czasem obsługi stanów 2…5 i nie wpływa na jasność świecenia cyfr. To twierdzenie okazało się słuszne również w praktyce.

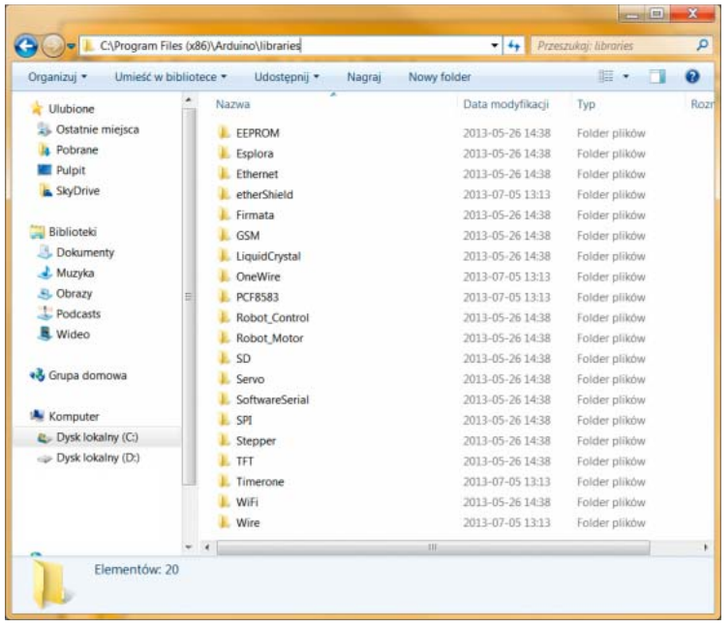
Zegarek z sekundnikiem

Opisaną maszynę stanów zastosowano w przykładowym programie do obsługi zegara. Cały program przykładowy jest dostępny w pliku *Obsluga_4xLED.ino* w materiałach dodatkowych do artykułu na serwerze FTP. Program jest opatrzony licznymi komentarzami, co ułatwi samodzielną analizę. W artykule skupimy się jedynie na opisaniu najważniejszych aspektów.

Każdy program w środowisku Arduino rozpoczyna się od zdefiniowania funkcji wyprowadzeń oraz nadania im poziomów początkowych. W środowisku Arduino IDE służy do tego specjalna funkcja *void setup()*, w której określa się funkcje poszczególnych bitów portów I/O i bloków peryferyjnych. Jest to o tyle wygodne, że podczas analizowania programu można szybko zorientować się odnośnie do funkcji pełnionych przez poszczególne wyprowadzenia. Zmusza też programistę do zachowania pewnego „porządku" w programie.

Ważną częścią programu obsługi zegara jest tablica dokonująca translacji cyfry na kod wyświetlacza 7-segmentowego. Wykonałem ja niejako w dwóch krokach, chociaż oczywiście można było uprościć deklarację kosztem jej czytelności. W kroku pierwszym są zdefiniowane wszystkie bity odpowiedzialne za świecenie pojedynczych segmentów. Na przykład, świeceniu segmentu „A" odpowiada bit numer 0, więc jego definicję można zapisać binarnie „11111110" lub szesnastkowo „0xFE". Podobnie segment „B", który jest dołączony do bitu numer 1 – „0xFD". Ponieważ są to liczby stałe i nie zmieniają się podczas pracy programu, to można je zdefiniować za pomocą dyrektywy *#define*.

Właściwą tablicę konwersji liczb na kod wyświetlacza 7-segmentowego należy umieścić w pamięci programu (Flash). W Arduino służy można posłużyć się w tym celu



Rysunek 1. Sposób skopiowania bibliotek do folderu programu

Listing 2. Procedura wyświetlająca poszczególne znaki (numer znaku jest przekazywany w argumencie wywołania funkcji)

```
//konwersja i wyświetlenie liczby na LED, liczby są w zmiennej globalnej
//bufor_led; argumentem funkcji jest numer wyświetlanej pozycji
void wyswietl_LED(byte numer_pozycji)
{
//wyłączenie wszystkich cyfr
  PORTB = PORTB | 0x0F;
//konwersja liczby na znak do wyświetlenia, wyświetlenie znaku
//argument numer pozycji jest o 1 większy od indeksu tablicy,
//więc pomniejszamy go o 1
//bufor_led zawiera liczby do wyświetlania, tablica wzorce_znaków
//pod numerem indeksu zawiera odpowiednie rozmieszczenie segmentów
//do wyświetlenia
  PORTD = wzorce_znakow[bufor_led[numer_pozycji-1]];
  PORTB = PORTB & anody_cyfr[numer_pozycji-1];
  delay(5);
}
```

instrukcją *const* służącą do definiowania stałych. Tablica o nazwie *wzorce_znaków* zawiera 11 bajtów, których pozycje odpowiadają konwertowanym cyfrom. I tak, na pozycji 0 umieściłem wzorzec znaku „0", a na pozycji 9 wzorzec znaku „9". Na pozycji 10 (przypomnijmy, że indeks w tablicy zaczyna się od 0) umieszczono kod wyłączający wszystkie segmenty, którego użyto do wygaszania zera nieznaczącego.

Teraz łatwo zauważyć, dlaczego tablice wzorców znaków definiowano dwuetapo-

Listing 3. Fragment maszyny stanu odpowiedzialnej za odczyt czasu, zależnie od zmiennej *tryb_wyswietlania*

```
//odczyt zegara
case 1:
  p.get_time();
//wyświetlenie czasu, tryb 0
  if (tryb_wyswietlania == 0)
  {
    bufor_led[3]=  p.minute % 10;
    bufor_led[2] = p.minute / 10;
    bufor_led[1]=  p.hour % 10;
    bufor_led[0] = p.hour / 10;
    if (bufor_led[0] == 0) bufor_led[0] = 10;  //wyłączanie zera nieznaczącego
    if (p.second%2 == 0) digitalWrite(pb4, HIGH);  //sterowanie dwukropkiem
(PB4)
    else digitalWrite(pb4, LOW);
  } else
//wyświetlenie sekund, tryb 1
    if (tryb_wyswietlania == 1)
    {
      bufor_led[3] = p.second % 10;
      bufor_led[2] = p.second / 10;
      bufor_led[1] = bufor_led[0] = 10;
      digitalWrite(pb4, LOW);
    } else
//wyświetlanie w czasie nastaw, tryb 3
      if (tryb_wyswietlania == 3)
      {
        bufor_led[3] = tmpminuty % 10;
        bufor_led[2] = tmpminuty / 10;
        bufor_led[1] = tmpgodziny % 10;
        bufor_led[0] = tmpgodziny / 10;
       }
  break;
```

wo. Poszczególne wiersze są reprezentowane przez iloczyn logiczny stałych definiujących zaświecanie się segmentów. W ten sposób jest bardzo łatwo modyfikować wygląd znaków i wyszukiwać ewentualne błędy.

Kody załączenia poszczególnych anod również zawarto w 4-bajtowej tablicy o nazwie *anody_cyfr*. Tu przypuszczalnie również można było zaoszczędzić kilka bajtów, ale z doświadczenia wiem, że mając do dyspozycji tablicę z kodami załączającymi anody jest łatwo zmieniać kolejność wyświetlania cyfr lub modyfikować program do użycia na innej płytce, z innymi połączeniami pomiędzy wyświetlaczem a mikrokontrolerem.

Funkcja wyświetlająca znak (list. 2) jako argument przyjmuje numer pozycji wyświetlacza. Poszczególne znaki do wyświetlenia są umieszczane w pamięci RAM, w zmiennej tablicowej o nazwie *bufor_led*. Aby wyświetlić znak wystarczy na odpowiedniej pozycji bufora umieścić liczbę z zakresu 0…10 (10 = zgaszenie wyświetlacza). Ponieważ pozycje znaków są numerowane od 1 (pierwszy z prawej) do 4 (pierwszy z lewej), natomiast indeks tablicy rozpoczyna się od 0, to znak do konwersji jest pobierany spod indeksu o jeden mniejszego *bufor_led[numer_pozycji-1]*. Następnie stanowi on indeks tablicy *wzorce_znaków* i po pobraniu odpowiedniego wzorca jest zapisywany do rejestru PORTD. Podobnie kod załączenia cyfry jest pobierany z tablicy *anody_cyfr* spod indeksu *numer_pozycji-1* za pomocą iloczynu bitowego (&) zmienia poziom na odpowiednim wyprowadzeniu portu B.

Do obsługi interfejsu I²C oraz układu PCF8583 użyto bibliotek dostępnych na stronie Arduino. Są to *PCF8583.h* oraz *Wire.h*. Pliki źródłowe bibliotek należy skopiować do katalogu, w którym jest zainstalowane środowisko Arduino, do folderu *Libraries*. Dla biblioteki *Wire.h* należy utworzyć podkatalog */Wire*, natomiast dla *PCF8583.h* – */PCF8583*. Sposób zagnieżdżenia katalogów dla bibliotek pokazano na **rysunku 1**. Dołączenie bibliotek do programu głównego odbywa się za pomocą dyrektywy *#include*. Zalecam użycie plików dostępnych w materiałach dodatkowych, ponieważ te pobrane przeze mnie ze strony Arduino wymagały pewnych modyfikacji – nie chciały bezbłędnie kompilować się w najnowszym środowisku Arduino IDE ze względu na błędne deklaracje typów zmiennych.

Maszyna stanów utworzona dla potrzeb programu obsługi zegara musi rozróżniać trzy tryby pracy. Są to:

- tryb pracy normalnej, w którym mogą być wyświetlane sekundy (licznik pobierany z RTC),
- tryb pracy normalnej, w którym jest wyświetlane wskazanie godzin i minut (liczniki pobierane z RTC),
- tryb nastaw, w którym są wyświetlane zmienne służące do wykonania nastaw (aktualizowane za pomocą naciśnięć przycisków SW1/SW2).
- Tryby pracy są rozróżniane za pomocą zmiennej *tryb_wyswietlania*, która może mieć następujące wartości:
- 0 – wyświetlanie liczników godzin i minut (normalny tryb pracy zegara),
- 1 – wyświetlanie licznika sekund,
- 3 – wyświetlanie zmiennych *tmpgodziny* i *tmpminuty* służących do wprowadzenia nastaw czasu zegara.

Numer aktualnego trybu pracy zawarty w zmiennej *tryb_wyswietlania* jest zmieniany po naciśnięciu przycisku SW1 (przełączanie zegar/sekundnik) lub SW2 (po wejściu w tryb ustawiania zegara). Na **listingu 3** pokazano fragment maszyny stanów z pętli głównej odpowiedzialny za wyświetlanie odpowiednich zmiennych, zależnie od stanu zmiennej *tryb_wyswietlania*. Jak łatwo zauważyć, do bufora wyświetlacz LED wpisywane są zmienne zależnie od wartości *tryb_wyswietlania*.

Podobnie do wyświetlania, od trybu, w którym pracuje zegar jest uzależniona obsługa przycisków SW1 i SW2, jednak zasada działania jest zbliżona do procedury z list. 3 i składa się głównie z rozpatrywania licznych warunków, więc pominiemy ją w tym opisie. Z punktu widzenia tego opisu istotna jest jedynie wiedza, w jakim celu wprowadzono różne tryby wyświetlania.

Do obsługi zegara służą przyciski SW1 i SW2. Przycisk SW1 w trybie wskazywania czasu służy do przełączania wyświetlania pomiędzy zegarem a sekundnikiem. Przycisk SW2 powoduje wejście w tryb nastaw zegara, a w tym trybie służy do zmiany „w górę” minut, godzin oraz uruchomienia odmierzania czasu. Przycisk SW1 w trybie nastaw powoduje przejście od pozycji do pozycji, od minut do oczekiwania na start odmierzania czasu.

Aby ustawić zegar należy:

- nacisnąć SW2,
- nacisnąć SW2 tyle razy, aby wyświetlany licznik minut był zgodny z oczekiwaniami,
- nacisnąć SW1 i przejść w ten sposób do licznika godzin,
- nacisnąć SW2 tyle razy, aby wyświetlany licznik godzin był zgodny z oczekiwaniami,
- nacisnąć SW1, aby przejść do oczekiwania na uruchomienie licznika zegara,
- nacisnąć SW2, aby zapisać nastawę i uruchomić licznik zegara RTC, co jest sygnalizowane krótkim sygnałem dźwiękowym.

Ustawianie zegara nie jest wykonywane zbyt często i dlatego wykonano je „na skróty”, jak najprościej, godząc się na pewien kompromis pomiędzy estetyką a funkcjonalnością.

Podsumowanie

Program źródłowy jest przykładem zastosowania AVT1616 i AVT5272 i może być dowolnie wykorzystywany i modyfikowany na zasadach licencji GPL. Analizując jego kod można znaleźć odpowiedzi na pytania odnośnie do sposobu obsługi układu zegara RTC oraz wyświetlacza LED.

Jacek Bogusz, EP