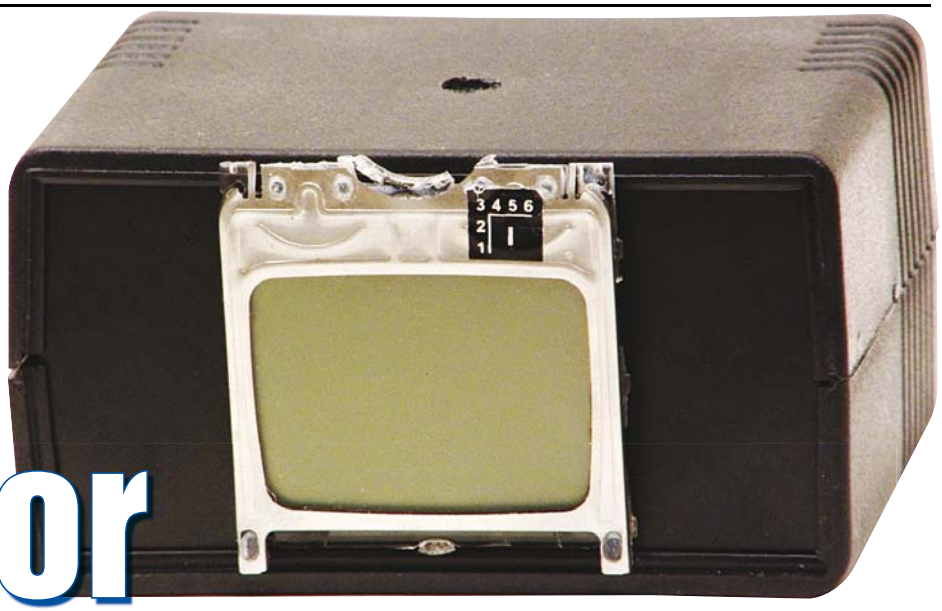
kit 2923 AVT



Monitor systemu komputerowego

Chciałbym zaproponować Czytelnikom budowę prostego układu, służącego do monitorowania pracy komputera. Idea budowy takiego urządzenia powstała dość dawno, podczas zaciekłego grania na laptopie. Momentami gra spowalniała i zastanawiałem się, co jest tego przyczyną. Mając do dyspozycji układ opisany w dalszej części artykułu, można bez problemu określić, jak wykorzystywany jest procesor, pamięć RAM, dysk twardy, łącze internetowe oraz akumulator. Pozwala to zareagować w sytuacji, gdy zasoby komputera są na wyczerpaniu lub łatwiej stwierdzić, co jest przyczyną niewystarczającej wydajności komputera. Informacje o stanie systemu są prezentowane w oknie aplikacji oraz na wyświetlaczu LCD pochodzącym z telefonu Nokia 3310. Komunikacja z urządzeniem odbywa się za pomocą portu USB, co oprócz łatwej instalacji umożliwia zasilenie układu z tego portu i zwiększa komfort użytkowania.

Czasami na forach internetowych spotyka się pytania w stylu „jak podłączyć LCD do portu LPT". Sugeruje to, że istnieje zapotrzebowanie na wyświetlacze podłączane do komputera, które wyświetlają dodatkowe informacje. W zasadzie niniejsze urządzenie jest prostym konwerterem USB->LCD. Oznacza to, że zapisując dane do portu szeregowego można przesłać je bezpośrednio na wyświetlacz. Bardzo proste komendy sterujące umożliwiają kasowanie wyświetlacza oraz zapis dowolnego tekstu do wybranej linii, choćby ze zwykłego HyperTerminala. Osoby posiadające umiejętność pisania aplikacji, mogą uzyskać szereg ciekawych efektów, choćby prezentację aktualnie odtwarzanego utworu po przygotowaniu wtyczki dla WinAMP-a. Szczegóły zapisu danych do wyświetlacza przedstawiono w dalszej części artykułu.

Do pracy urządzenia niezbędne jest zaprogramowanie mikrokontrolera z rodziny AVR. Czytelnicy z mniejszym doświadczeniem mogą poprosić o pomoc kolegę lub zakupić zaprogramowany układ w sklepie AVT. Program w postaci wynikowej, tzn. przygotowanej do natychmiastowego wgrania do procesora bez żadnych zmian, dostępny jest na Elportalu. Bardziej zaawansowani Czytelnicy znajdą tam również kod w wersji źródłowej, co pozwoli wprowadzać własne modyfikacje, jak choćby inny niż standardowy krój czcionki (szczegóły w dalszej części artykułu).

Opis układu

Schemat urządzenia przedstawiono na **rysunku 1**. Układ składa się z pięciu bloków. Pierwszy z nich stanowi stabilizator S78DL33. Dostarcza on napięcie 3,3V, które jest wymagane przez sterownik wyświetlacza LCD. W tym miejscu można zastosować dowolny odpowiednik o wydajności prądowej na poziomie 100mA lub więcej, ale należy zwrócić uwagę na sposób jego wlutowania. Na środkowym padzie znajduje się masa, natomiast wejście, przez bezpiecznik polimerowy, jest przyłączone do złącza USB. Lutując inny stabilizator, należy się upewnić, czy wyprowadzenia zostaną wlutowane w taki właśnie sposób. Kondensatory C11–C14 stanowią filtry zmniejszające tętnienia napięcia w ścieżce zasilającej.

Drugi blok, na który składają się elementy z sygnaturą 2x (np. U21, R21, C22, etc.) stanowi proste, regulowane źródło prądowe, które zapewnia wydajność na poziomie od zera do około 80mA. Pobranie większej wartości prądu wymagałoby wymiany bezpiecznika na odpowiednik dla większego prądu oraz rezystora R23 i/lub R22. Źródło to zostało przewidziane do podświetlenia wyświetlacza LCD dla Czytelników, którzy chcieliby uzyskać podświetlenie i zwiększenie widoczności wskazań urządzenia. Optymalnym elementem oświetlającym wydaje się być dioda LED o mocy 0,5 lub 1W.

Blok trzeci z oznaczeniami 3x składa się z konwertera USB<->RS232 oraz złącza USB, do którego należy podłączyć przewód łączący z komputerem. Bezpiecznik F31 zabezpiecza port przed nadmiernym poborem prądu oraz zwarciami. Co prawda każdy komputer powinien mieć stosowne zabezpieczenia, ale należy traktować je jako ostateczność i przewidzieć własny sposób zapobiegania przeciążeniom. Warto wspomnieć, że rezystory R31 oraz R32 też pełnią funkcję zabezpieczającą. Jak można wywnioskować ze schematu, mikrokontroler zasilany jest napięciem 3,3V, natomiast układ FT232 napięciem 5V. Brak tych rezystorów spowodowałby nadmierny przepływ prądu i uszkodzenie portów. Dzięki obecności diod zabezpieczających, wbudowanych w porty mikrokontrolera oraz ograniczeniu prądu przez R31 i R32, taki sposób podłączenia nie wywołuje uszkodzenia mikrokontrolera.

Czwarty blok stanowią w zasadzie dwa złącza. Pierwsze z nich (JP41) przeznaczono do podłączenia wyświetlacza. Dołączone kondensatory mają za zadanie filtrowanie napięcia zasilającego oraz wewnętrznego napięcia wytwarzanego przez przetwornicę modułu LCD. Drugie złącze (JP42) przeznaczono do ewentualnego podłączenia programatora ISP.

Piąty blok składa się przede wszystkim z mikrokontrolera ATmega8. Istnieje możliwość wykorzystania wersji niskonapięciowej (z sufiksem L) oraz „zwykłej". Kondensatory C51... C53 mają za zadanie, standardowo, filtrować napięcie zasilające i ograniczać ilość „śmieci"

wprowadzanych do linii zasilających przez pracujący procesor. Do komunikacji z komputerem wykorzystany został asynchroniczny port szeregowy, więc konieczne było zapewnienie stabilnego sygnału zegarowego i dlatego znajduje się tutaj rezonator kwarcowy Q51.

Oprogramowanie

Program został napisany w języku C++ z użyciem środowiska WinAVR. Zdefiniowane zostały dwie klasy: *usart* oraz *lcd3310*. Pierwsza, jak łatwo się domyślić, zajmuje się obsługą portu szeregowego. W zasadzie komunikacja z komputerem odbywa się za pomocą portu USB, ale układ FT232 sprawia, że zarówno od strony komputera, jak i mikrokontrolera, widoczny jest port szeregowy (RS232). Upraszcza to znacznie wymianę informacji. Klasa *usart* odpowiada za buforowany odbiór danych. Oznacza to, że odczytaniu każdego znaku towarzyszy przerwanie, które umieszcza go w specjalnym buforze. Za pomocą funkcji *bytesToRead* możliwe jest sprawdzenie, ile nieodczytanych znaków się w nim znajduje, a za pomocą funkcji *readByte* dane te są odczytywane. Za konfigurację portu, m.in. ustawienie prędkości transmisji (115200 bodów) odpowiada konstruktor tej klasy.

Druga klasa, *lcd3310*, jest odpowiedzialna za obsługę wyświetlacza (zaskakujące, prawda? :)). Konstruktor wykonuje typowe operacje, takie jak ustawienie portów I/O, reset modułu LCD czy konfiguracja poszczególnych rejestrów. Najbardziej pożądaną funkcją jest *write*, w argumencie której podaje się łańcuch tekstowy (tzw. *string*), przeznaczony do wyświetlenia na wyświetlaczu. Posiada ona w zasadzie dwa bloki kodu. Ma z nich odpowiada za dekodowanie poleceń zaczynających się znakiem @. Po pojawieniu się takiego znacznika pobierany jest kolejny bajt i następuje wykonanie odpowiedniej akcji. Przejściu do wybranej linii towarzyszy automatyczne czyszczenie jej zawartości. Ma to szczególne znaczenie przy prezentowaniu danych o zmiennej długości, np. transfer z dysku twardego może mieć raz 1000kB, a innym razem 100kB. Bez wyczyszczenia zawartości linii ujrzelibyśmy na ekranie zapis: *HD RD:100kBB*. To ostatnie *B* byłoby pozostałością po poprzednim zapisie. Zapewne pojawi się pytanie: dlaczego wcześniej nie wyczyścić całego wyświetlacza odpowiednim poleceniem? Odpowiedź jest dość prosta – towarzyszy temu zauważalne i irytujące migotanie prezentowanych informacji.

Drugi fragment kodu, poprzedzony komentarzem *zapisywanie znaków*, odpowiada za konwersję przesłanego znaku na ciąg bitów zrozumiałych dla modułu LCD. Potrzebna jest spora tablica zawierająca stosowne ciągi kodowe dla poszczególnych znaków. Pojedynczy znak ma organizację 8x5 pikseli, przy czym jeden z wierszy (najniższy) jest zawsze pusty i tym samym służy za separator pomiędzy poszczególnymi liniami tekstu. W efekcie znaki mają rozmiar 7x5 pikseli. Najbardziej ciekawą rzeczą jest tu zmienna *offset*, w której zapisywany jest adres komórki w tablicy *chargen*. Dzięki niej możemy odwołać się do właściwego miejsca w pamięci zawierającego informację o tym, jak ma być zbudowany

```
//zapis polecenia
void lcd3310::write_cmd (char data){
  //linie sygnalowe
  cbi(LCD_PORT_REG, LCD_DC) ;
  cbi(LCD_PORT_REG, LCD_SCE) ;

  //zapisz slowo danych
  for(unsigned char t=0 ; t<8 ; t++){
    if(bit_is_set(data, 7)){
      sbi(LCD_PORT_REG, LCD_SDIN) ;
    }else{
      cbi(LCD_PORT_REG, LCD_SDIN) ;
    }
    _delay_us(1) ;
    sbi(LCD_PORT_REG, LCD_SCK) ;
    _delay_us(1) ;
    cbi(LCD_PORT_REG, LCD_SCK) ;
    data = data << 1 ;
  }
  _delay_us(1) ;
  sbi(LCD_PORT_REG, LCD_SCE) ;
}
```

Listing 1

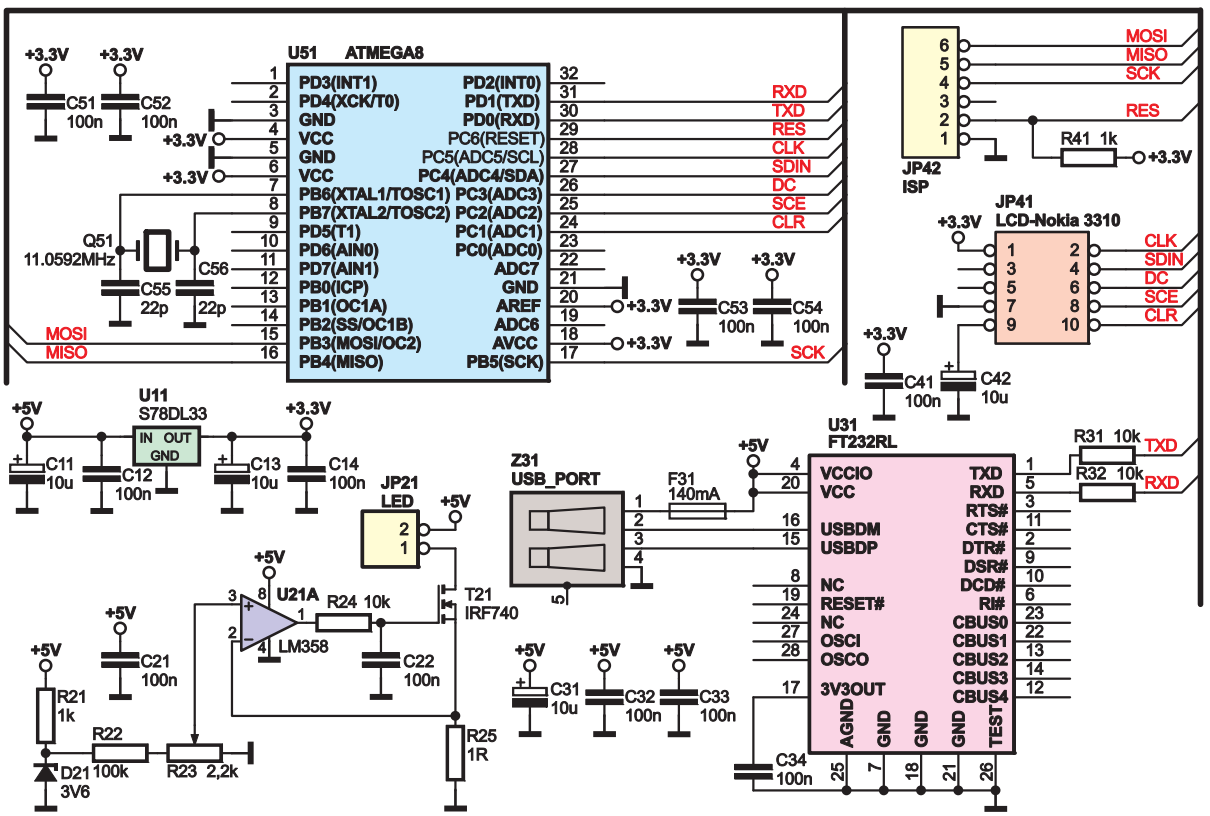
dany znak. Jak już wspomniałem, poszczególne znaki składają się z 5 bajtów, więc pierwszy znajduje się w komórkach od 0 do 4, drugi od 5 do 9, itd. – w ogólności n*5. Każdy wyświetlany znak ma swój kod liczbowy (ASCII), na podstawie którego można określić, z jakiego miejsca tablicy pobrać informacje o wyglądzie znaku. Przykładowo litera ‘A’ ma kod 65, co oznacza, że w tablicy *chargen* znajduje się na pozycji od 65*5 do 65*5+4. Tu pojawia się pewien haczyk, mianowicie pierwsze 32 znaki tablicy ASCII nie są normalnie wyświetlane, zatem zmarnujemy 32*5 bajtów. W związku z tym zostały one pominięte i obecnie litera A zaczyna się w tablicy od pozycji (65-32)*5, co ma swoje odzwierciedlenie przy wyznaczaniu wartości zmiennej *offset*:

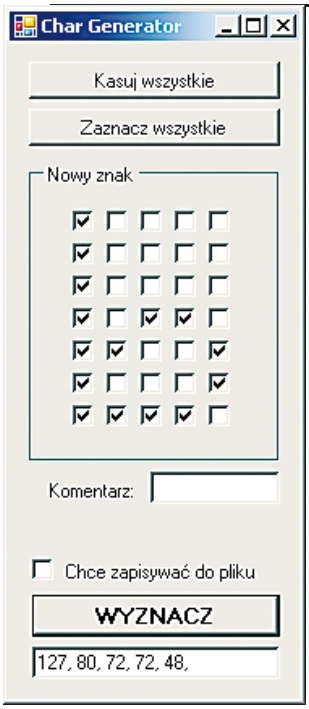
*unsigned int offset = (*src - 32)*5 ;*

Wyświetlenie znaku alfanumerycznego wymaga wysłania do LCD pięciu bajtów z tablicy *chargen,* począwszy od komórki reprezentowanej przez zmienną *offset* oraz bajtu zerowego stanowiącego separator pomiędzy sąsiednimi znakami.

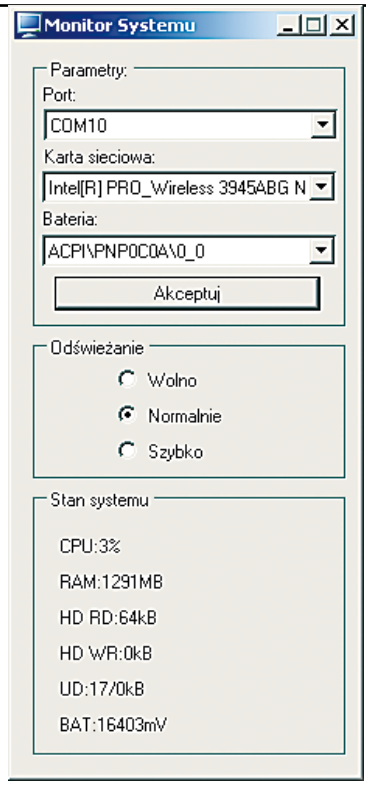
W klasie *lcd3310* zawarte są jeszcze dwie funkcje: *write_cmd* oraz *write_char*. Pierwsza z nich została przedstawiona na **listingu 1**. Różnią się ona między sobą jedynie pierwszą linią, gdyż chcąc zapisać komendę, należy podać zero na wejście *D/C*, a zapisując dane, należy podać jeden. Obie funkcje realizują uproszczony interfejs SPI, tzn. jednokierunkowy. Pobierane są bity z przesłanego argumentu funkcji

Rys. 1 Schemat ideowy





Rys. 2



Rys. 4

(począwszy od najstarszego) i sukcesywnie, w takt zegara, wystawiane na wejście danych LCD. Całość rozpoczyna się od wystawienia zera na wejście *SCE* (odpowiednik CS – wybór układu, czyli informacja, że magistrala ma być aktywna), a kończy ustawieniem jedynki na tym porcie.

Warto jeszcze wspomnieć o tablicy *chargen*, która znajduje się w pliku *chargen.h*. Czytelnicy, którzy chcieliby zmienić krój czcionki na własny, powinni zainteresować się tym właśnie plikiem. Wyznaczanie na piechotę wszystkich kodów jest bardzo męczące, dlatego napisałem prosty program, który znacznie przyspiesza ten proces – jest on widoczny na **rysunku 2**. Można go pobrać razem z resztą materiałów zamieszczonych w Elportalu. Obsługa aplikacji nie powinna sprawiać problemów. Dwa pierwsze przyciski (*kasuj wszystkie/zaznacz wszystkie*) umożliwiają, odpowiednio, wyłączenie bądź włączenie wszystkich punktów. W polu *nowy znak* można określić, jak będzie wyglądał dany symbol – pole zaznaczone odpowiada włączonemu pikselowi na LCD (czarna kropka). Po zaprojektowaniu własnego znaku należy kliknąć *Wyznacz* i otrzymany kod podmienić z oryginalnym kodem dla danego znaku w pliku *chargen.h*. Można także zaznaczyć pozycję *Chcę zapisywać do pliku*, co sprawi, że przy każdym kliknięciu *Wyznacz* dany kod zostanie dodany do pliku (w tym samym katalogu, w którym jest program). Jest to rozwiązanie wygodne, gdy wymienia się cały zestaw znaków – można wtedy przygotowywać kolejne znaki zgodnie z tablicą ASCII i klikać *Wyznacz*, a na końcu skopiować zawartość wygenerowanego pliku i zamienić z zawartością pliku oryginalnego. Tu uwaga – nie można zwyczajnie zamienić plików, gdyż program nie generuje poleceń w języku C++, należy otworzyć oba pliki i przekopiować tylko dane.

Program główny przedstawiono na **listingu 2**. Nie jest on nadmiernie skomplikowany. Pierwsza pętla *while* będzie się wykonywać do momentu, aż do portu szeregowego nie trafi jakiś znak przesłany z komputera. Do tego czasu we wszystkich liniach będzie migać napis *Loading...* co pozwoli łatwo, bez użycia komputera, sprawdzić, czy urządzenie pracuje prawidłowo. Druga pętla *while* wykonuje się w nieskończoność (w praktyce – do odłączenia zasilania). Jej zadanie sprowadza się do rozpoznania jednego z trzech stanów pracy portu: oczekiwanie na nowe dane, odbiór nowych danych oraz zakończenie odbioru. Cała sztuczka polega na umieszczeniu opóźnienia – gdy dane są transmitowane, to w warunku *if* wartość zmiennej *odebrano* będzie różna od *bytesToRead*. Wynika to z prostej zależności – po odczytaniu liczby znaków w buforze zapamiętujemy je w zmiennej i czekamy 50ms. Jeżeli transmisja się nie zakończyła, to przez te 50ms odebrane zostały kolejne znaki i wartość zmiennej jest różna od liczby znaków dostępnych w buforze portu. Sztuczkę tę powtarzamy do momentu wystąpienia równości, co będzie miało miejsce także w sytuacji, gdy żadne dane nie są przesyłane, ponieważ zmienna *odebrano* będzie miała wartość zero, czyli tyle samo, ile zwraca funkcja *bytesToRead*. Stąd pojawia się konieczność dodania drugiego warunku sprawdzającego, czy w ogóle coś odebrano. Po stwierdzeniu, że przesyłanie tekstu się zakończyło, dane z bufora są przesyłane do wyświetlacza w celu zaprezentowania ich Czytelnikowi.

Listing 2

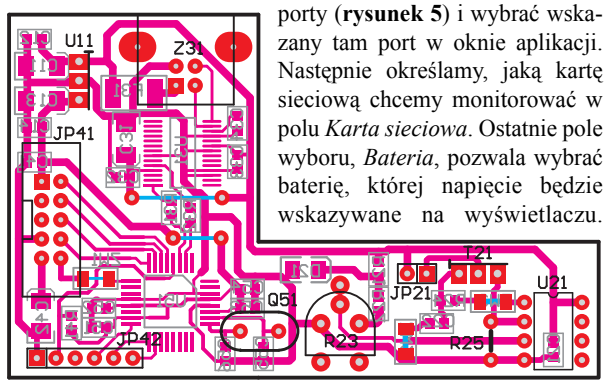
```
int main(){
  //zmienna pomocnicza
  unsigned char odebrano = 0 ;
  char buf[100] ;
  //opoznienie
  _delay_ms(10) ;
  while(port.bytesToRead()==0){
    lcd.init() ;
    lcd.write("@1Loading...") ; //zapisz do LCD
    lcd.write("@2Loading...") ; //zapisz do LCD
    lcd.write("@3Loading...") ; //zapisz do LCD
    lcd.write("@4Loading...") ; //zapisz do LCD
    lcd.write("@5Loading...") ; //zapisz do LCD
    lcd.write("@6Loading...") ; //zapisz do LCD
    _delay_ms(500) ;
  }
  //petla glowna
  while(1){
    //obsluga portu szeregowego
    odebrano = port.bytesToRead() ;
    _delay_ms(50) ;
    //zakonczono odbior ?
    if( (odebrano == port.bytesToRead()) && (odebrano!=0)){
      port.readBuffer(buf, odebrano) ; //odczytaj dane
      lcd.write(buf) ; //zapisz do LCD
    }

  }
}
```

Montaż i uruchomienie

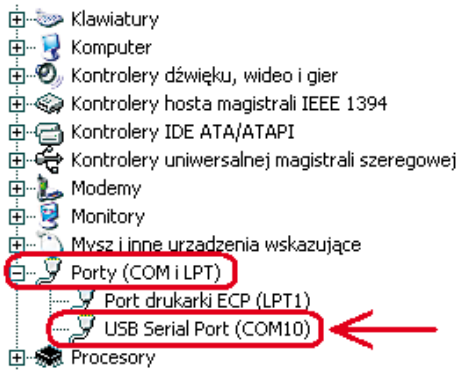
Układ można zmontować na płytce drukowanej pokazanej na **rysunku 3**. Montaż jest klasyczny i nie powinien sprawiać problemów. Przed pierwszym uruchomieniem urządzenia należy zainstalować bezpłatne sterowniki ze strony producenta układu FT232 – **http://www.ftdichip.com/**. Wybieramy tam *Drivers*, a następnie *VCP* i pobieramy sterowniki adekwatne do posiadanego systemu operacyjnego. Po zainstalowaniu sterownika można podłączyć urządzenie do portu USB, w efekcie czego system je zainstaluje.

Po pierwszym uruchomieniu programu (**rysunek 4**) pokaże się informacja o błędzie inicjacji sprzętu, który wynika z braku pliku konfiguracyjnego. Jego utworzenie wymaga wybrania portu komunikacyjnego w polu *Port*. Jaki port wybrać? Najprościej skorzystać z panelu sterowania systemu Windows i tamtejszej ikony *System*. W otwartym okienku wybieramy zakładkę *Sprzęt* i następnie *Menedżer urządzeń*. Na liście trzeba odnaleźć porty (**rysunek 5**) i wybrać wskazany tam port w oknie aplikacji. Następnie określamy, jaką kartę sieciową chcemy monitorować w polu *Karta sieciowa*. Ostatnie pole wyboru, *Bateria*, pozwala wybrać baterię, której napięcie będzie wskazywane na wyświetlaczu.



Rys. 3

Rys. 5



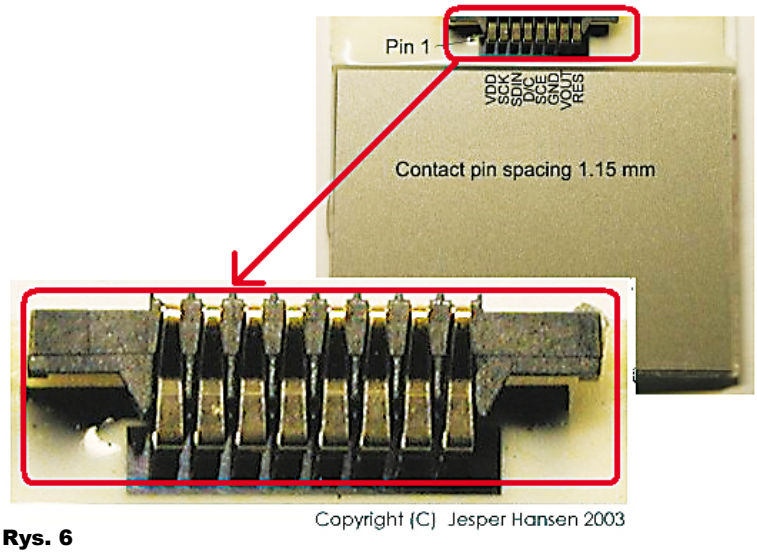
Niestety nie udało mi się znaleźć sposobu na zaprezentowanie pozostałej pojemności w procentach tak jak to robi Windows, więc uznałem, że jakimś pocieszeniem będzie monitorowanie napięcia, które zmniejsza się wraz z wyładowaniem baterii. Zależało mi bardzo na jakimś wskaźniku rozładowania akumulatora, gdyż czasami sprawdzenie jego stanu jest utrudnione (np. podczas grania), ale warto wiedzieć, ile jeszcze, mniej więcej, czasu pozostało. Na zakończenie należy kliknąć *Akceptuj*, co spowoduje otwarcie portu komunikacyjnego i wyświetlenie stosownego komunikatu. Przy następnym uruchomieniu nastawy zostaną odczytane z pliku. Aplikacja po zminimalizowaniu „ukrywa się" w systemowym trayu i nie zajmuje miejsca na pasku zadań.

Warto wspomnieć, że szybsza aktualizacja wskazań wybierana w polu *Odświeżanie* może powodować większe obciążenie procesora, co przyczyni się do skrócenia czasu pracy na baterii. W takiej sytuacji warto skorzystać z wolniejszego odświeżania.

Ostatnie pole aplikacji, *Stan systemu*, prezentuje monitorowane parametry:

- CPU – obciążenie procentowe procesora
- RAM – ilość wolnej pamięci RAM
- HD RD – aktualna prędkość odczytu z dysku twardego (zero, gdy dysk nie jest odczytywany)
- HD WR – aktualna prędkość zapisu dysku twardego (zero, gdy dysk nie jest zapisywany)
- UD – odpowiednio: upload oraz down-load, czyli prędkość wysyłania i pobierania danych z Internetu/sieci (rozdzielone ukośną kreską)
- BAT – napięcie akumulatora

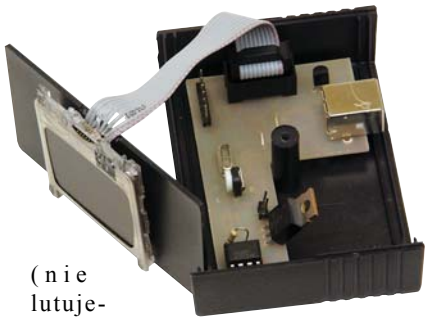
Parametry te są, oczywiście, identyczne z tym, co pokazuje się na wyświetlaczu.



Rys. 6

Kilka słów o module LCD. Ma on niestety elektrody napylone na szklaną płytkę, więc demontaż ramki, w której jest sprzedawany, nie jest zbyt dobrym pomysłem. W modelu widocznym na fotografii tytułowej została ona obcięta tam, gdzie nie była potrzebna. Ramka ta gwarantuje docisk złącza do napylonych elektrod. Czasami się zdarza, że mimo to coś nie kontaktuje i właśnie z tego względu po uruchomieniu urządzenia pojawia się migający napis „Loading...". Miganie jest wynikiem naprzemiennego zapisywania danych i inicjacji wyświetlacza, co stwarza okazję do spokojnego sprawdzenia jakości połączenia.

Wyświetlacz jest podłączony taśmą 10-żyłową. Z jednej strony przylutowany jest wyświetlacz, a z drugiej zaciśnięte złącze IDC-10. Warto mieć na uwadze, że lutujemy przewody po kolei, z tym że trzeci oraz piąty obcinamy



(nie lutujemy), natomiast siódmy z ósmym zamieniamy miejscami. Wyprowadzenia LCD pokazano na **rysunku 6** (na podstawie [1]).

Urządzenie zostało zamontowane w obudowie KM-48NB. W tylnej ściance wyciąłem otwór na złącze USB, natomiast moduł LCD został przyklejony do przedniego panelu obudowy. W handlu dostępne są także przezroczyste panele, co daje możliwość schowania wyświetlacza w obudowie po opracowaniu sposobu na skuteczne przytwierdzenie złącza. Wtedy można umieścić wewnątrz również diodę LED i wykorzystać obecne na płytce źródło prądowe.

Płytka drukowana została przyklejona klejem na gorąco do spodu obudowy.

Możliwości zmian

Urządzenie dzięki prostemu interfejsowi komunikacyjnemu ułatwia wykorzystanie go do realizacji własnych pomysłów. Zapis informacji, jak już wspomniano, odbywa się za pomocą standardowego portu szeregowego RS232 (tak jest ono widziane przez komputer) z prędkością 115 200. Wybór linii, do której ma być zapisany ciąg znaków, odbywa się poleceniem *@n*, gdzie *n* to numer linii. Przykładowo wysłanie ciągu „@2druga linia @4czwarta linia" spowoduje, że w drugiej linijce pojawi się napis *druga linia*, a w czwartej – napis *czwarta linia*. Za pomocą polecenia *@c* można wyczyścić zawartość całego wyświetlacza. Cały problem sprowadza się zatem do napisania programu, który prześle pożądane dane do portu szeregowego. Pracę układu można przetestować, uruchamiając dowolny terminal (np. *Start->Uruchom* i polecenie *hypertrm*). Po skonfigurowaniu portu szeregowego przesłanie dowolnego ciągu znaków spowoduje zaprezentowanie go na wyświetlaczu. Polecenia sterujące zaczynające się od znaku @ mogą nie działać prawidłowo, gdyż parametr (czyli numer linii lub znak *c*) muszą być wysłane natychmiast po znaku @.

Źródła:

[1] http://www.techdesign.be/projects/datasheet/nokia_3310_LCD.jpg

Jakub Borzdyński
jakub.borzdynski@elportal.pl

Wykaz elementów

Rezystory

R21,R41	1kΩ SMD 0805
R22	100kΩ SMD 0805
R23	2,2kΩ PR
R24,R31,R32	10kΩ SMD 0805
R25	1Ω

Kondensatory

C11,C13,C31,C42	10µF tantal SMD (rozmiar B)
C12,C14,C21,C22,C32-C34,C41,C51-C54	100nF 2012(0805)
C55,C56	22pF 2012(0805)

Półprzewodniki

D21	3V6
JP21	LED
JP41	LCD-Nokia 3310
T21	IRF740
U11	S78DL33
U21	LM358
U31	FT232RL
U51	ATmega8

Pozostałe

F31	140mA (1812)
JP42	goldpin x6
Q51	11,0592MHz
Z31	USB PORT

Komplet podzespołów z płytką jest dostępny w sieci handlowej AVT jako kit szkolny AVT-2923.