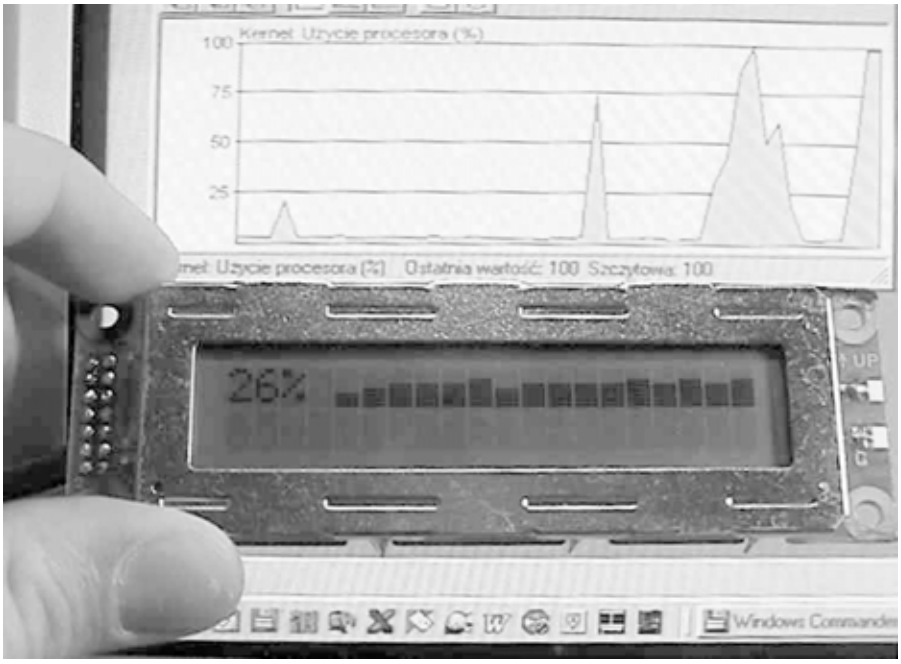
CPU-meter do PC-ta



Przedstawiony tutaj sprzętowy wskaźnik obciążenia procesora PC-ta wykorzystuje prosty i łatwo dostępny sprzęt (port Centronics) oraz dosyć złożony niskopoziomowy software w Delphi. Na szczęście większość pracy została już wykonana i tylko czeka na połączenie w działającą całość...
***Rekomendacje**: dla osób chcących rozbudować swój komputer o prosty, lecz efektowny wskaźnik obciążenia, będący również doskonałą bazą dla dalszych eksperymentów.*

Miernik obciążenia procesora jest obecny w niemal każdym systemie operacyjnym, często bywa wręcz zintegrowany z powłoką, jak w przypadku Sun Solaris. Popularne Windowsy również standardowo wyposażone są w *Monitor Systemu* i *Miernik Zasobów*. Proponujemy tutaj pewną alternatywę dla typowego, programowego wyświetlania czasu zajętości procesora przez przeniesienie aktualnego wskazania wraz z wykresem na zewnątrz komputera. Idealnym rozwiązaniem wydaje się zastosowanie powoli odchodzącego w niepamięć interfejsu Centronics do sterowania wyświetlaczem alfanumerycznym LCD. Ma to tym większy sens, że port równoległy bardzo często bywa niewykorzystany, jako że jego rolę przejmuje powoli USB. Całość projektu została przygotowana dla wyświetlaczy LCD kompatybilnych ze standardem HD44780. Dostępność tego typu wyświetlaczy obecnie nie jest żadnym problemem. Tak więc kluczem obwodu elektrycznego jest interfejs portu równoległego z wyświetlaczem. Sposób realizacji tego interfejsu pokazano na **rys. 1**. Jak widać, schemat nie jest szczególnie skomplikowany, komentarz do niego nie musi być zbyt obszerny.

Magistrala danych D0...D7 jest sterowana bezpośrednio przez wyjścia danych portu Centronics. Bit przełączający między zapisem i odczytem z wyświetlacza (R/!W) został sprzętowo ustawiony na zapis. Powodem tego jest brak możliwości prostego odczytu danych z wyświetlacza, ponieważ z magistrali danych portu równoległego w standardowym trybie (SPP) nie można czytać. Dlatego implementacja odczytu danych z LCD (w tym również dosyć przydatnej flagi zajętości) bardzo by się skomplikowała i została odłożona na przyszłość. Pozostałe bity sterujące pracą wyświetlacza, tzn. przełączanie dane/komenda (RS) oraz taktowanie (E), są obsługiwane przez odpowiednie sygnały portu równoległego (SELECT IN oraz STROBE). Standardowy wyświetlacz alfanumeryczny LCD wymaga jeszcze podania dodatkowego napięcia sterującego kontrastem na wejście oznaczone Vo. W niektórych modelach wystarcza napięcie bliskie 0V, a w innych niezbędne jest wręcz napięcie ujemne. W tym celu do obwodu elektrycznego dodany został opcjonalny moduł zasilania -5 V zbudowany w oparciu o przetwornicę ICL7660. Potencjometr 10 kΩ umożliwia płynną regulację na-

List. 1. Program odpowiedzialny za wysyłanie danych do portu I/O

```
procedure prt(portn: word; val: byte);
begin
  asm
    mov al,val
    mov dx,portn
    out dx,al
  end;
end;
```

pięcia kontrastu pomiędzy -5 V a +5 V. W przypadku pominięcia przetwornicy należy drugi zacisk potencjometru zewrzeć zworką do masy. W tym wypadku oczywiście nie jest możliwa generacja napięć ujemnych, ale w wielu nowych typach wyświetlaczy nie ma to znaczenia.

Teraz oprogramowanie

Po omówieniu prostego w realizacji obwodu elektrycznego pora zająć się „inteligencją" urządzenia, która oczywiście została zaszyta w oprogramowaniu. Oprogramowanie to można podzielić na dwie części: procedury sterujące wyświetlaczem oraz moduł pomiaru obciążenia procesora.

Podstawą obsługi wyświetlacza jest oczywiście umiejętność programowania poszczególnych sygnałów sterujących. W tym przypadku można to osiągnąć przez pisanie do przestrzeni adresowej portu LPT. Przestrzeń ta rozciąga się przez kilka kolejnych bajtów adresu bazowego, typowo 278h lub 378h i jest krótko podsumowana w **tab. 1**.

List. 2. Procedura wysyłająca dane i polecenia do sterownika wyświetlacza LCD

```
procedure write_LPT(a: byte; sterowanie:boo-
lean);
begin
  if sterowanie=TRUE then
  begin
    prt(LPT+$02,$09);  { RS=0, E=0 }
    prt(LPT+$00,a);    { rozkaz }
    prt(LPT+$02,$01);  { RS=0, E=1 }
    prt(LPT+$02,$09);  { RS=0, E=0 }
  end else
  begin
    prt(LPT+$02,$08);  { RS=1, E=0 }
    prt(LPT+$00,a);    { dana }
    prt(LPT+$02,$00);  { RS=1, E=1 }
    prt(LPT+$02,$08);  { RS=1, E=0 }
  end;
end;
```

Podstawową procedurą przy obsłudze portu jest wysłanie danej wartości na port I/O (odpowiada za to program pokazany na **list. 1**), zaimplementowany dla języków Pascal firmy Borland (np. Turbo Pascal i Delphi) dzięki wbudowanemu asemblerowi. Parametrami tej procedury są oczywiście konkretna wartość oraz adres I/O, na który wartość ta ma zostać wysłana.

Dzięki powyższej procedurze można wysyłać dane lub rozkazy do wyświetlacza. Wysłanie danej od rozkazu różni się *de facto* tylko stanem sygnału sterującego RS, w związku z czym warto zintegrować obie te funkcje w jedną procedurę, pokazaną na **list. 2**. W procedurze tej zmienna globalna LPT oznacza adres bazowy wybranego portu.

Jak już stwierdzono wcześniej, działanie portu równoległego w trybie podstawowym nie pozwala na odczyt danej przez magistralę D0...D7. Nie można w związku z tym odczytać statusu sterownika wyświetlacza, a konkretnie czy jest gotowy do przyjęcia kolejnej danej (bit BF). Rozwiązano ten problem w nieco prymitywny, aczkolwiek skuteczny sposób, mianowicie przez opóźnienie o ustalanej programowo długości. Długość tego opóźnienia można zoptymalizować w ostatniej fazie testów dla konkretnego komputera. Opóźnienie to nie jest jednak obciążeniem dla systemu (nie wliczając inicjalizacji), bowiem jest zrealizowane w Delphi za pomocą timerów, które uaktywniają się tylko w określonych interwałach czasowych.

List. 3. Procedury realizujące: konfigurację podczas inicjalizacji, czyszczenie ekranu, powrót kursora, przesunięcie kursora, wpisywanie znaków do CGRAM

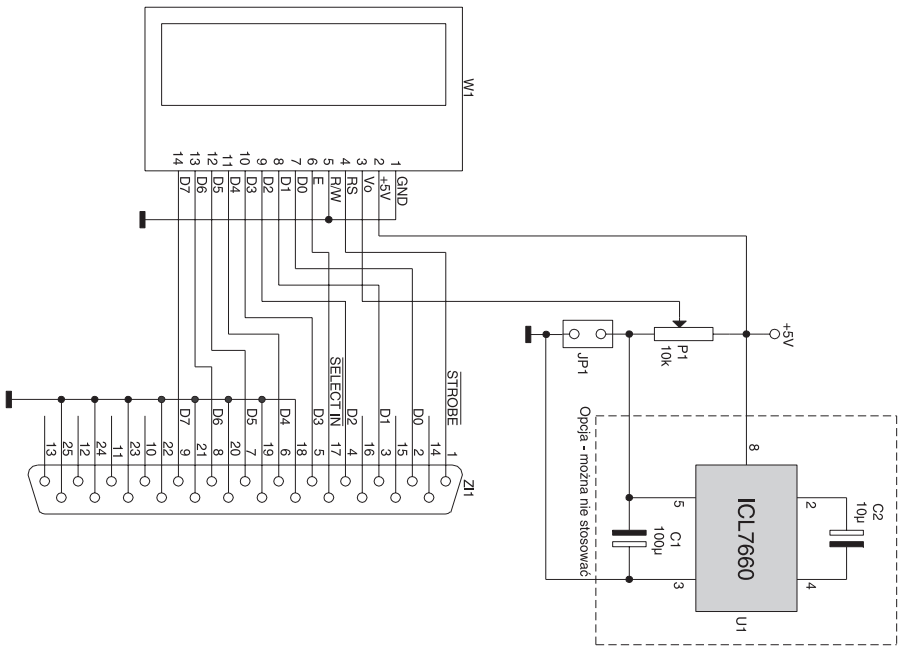
```
procedure
lcd_define(d0,d1,d2,d3,d4,d5,d6,d7:byte);
{ definicja znaku do CGRAM }
begin
  sleep(10);
  write_LPT(d0,FALSE);
  sleep(10);
  write_LPT(d1,FALSE);
  sleep(10);
  write_LPT(d2,FALSE);
  sleep(10);
  write_LPT(d3,FALSE);
  sleep(10);
  write_LPT(d4,FALSE);
  sleep(10);
  write_LPT(d5,FALSE);
  sleep(10);
  write_LPT(d6,FALSE);
  sleep(10);
  write_LPT(d7,FALSE);
  sleep(10);
end;

procedure lcd_init;
{ inicjalizacja wyswietlacza LCD }
begin
  sleep(10);
  write_LPT(32+16+8,TRUE);
  sleep(10);
  write_LPT(16+4,TRUE);
  sleep(10);
  write_LPT(8+4,TRUE);
  sleep(10);
  write_LPT(4+2,TRUE);
  sleep(10);
  write_LPT(1,TRUE);
  sleep(1000);
  write_LPT(64,TRUE);
  sleep(10);
  lcd_define(0,0,0,0,0,0,0,31);
  lcd_define(0,0,0,0,0,0,0,31);
  lcd_define(0,0,0,0,0,31,31,31);
  lcd_define(0,0,0,0,0,31,31,31);
  lcd_define(0,0,0,31,31,31,31,31);
  lcd_define(0,0,0,31,31,31,31,31);
  lcd_define(0,31,31,31,31,31,31,31);
  lcd_define(31,31,31,31,31,31,31,31);
  sleep(10);
end;

procedure lcd_clear;
{ czyszczenie ekranu wyswietlacza }
begin
  write_LPT(1,TRUE);
end;

procedure lcd_home;
{ powrot kursora }
begin
  write_LPT(2,TRUE);
end;

procedure lcd_shift;
{ przesuniecie zawartosci wyswietlacza }
begin
  write_LPT(16+8+4+1,TRUE);
end;
```



Rys. 1. Schemat elektryczny interfejsu

List. 4. Najważniejsze fragmenty programu

```
{ zmienne globalne }
var
  TestForm: TTestForm;
  buffer: string;
  akt,LPT,bufl: integer;
.
.
.
procedure TTestForm.TimerTimer(Sender: TObject);
{ procedura obslugi timera glownego, okres ustawiany przez użytkownika }
var i : Integer;
    j : Double;
    s : string;
begin
  CollectCPUData;
  j := GetCPUUsage(0)*100;
  s := Format('%1.0f%% ',[j]);
  TestForm.Caption:='CPU Usage '+s;
  Application.Title:='CPU Usage '+s;
  for i := 1 to 4 do buffer[i]:= s[i];
  for i := bufl+6 downto 6 do buffer[i]:= buffer[i-1];
  buffer[5]:= chr(trunc(j/13)+8);
  lcd_home;
  akt:= 1;
  Timer1.Enabled:= true;
end;

procedure TTestForm.FormCreate(Sender: TObject);
{ procedura przy inicjalizacji programu }
var i : Integer;
    f : TIniFile;
begin
  f := TIniFile.Create('CpuUsage.Ini');
  with f do
  begin
    LPT:= ReadInteger('Configuration','Address',$278);
    bufl:= ReadInteger('Configuration','Buffer',20);
    Timer.Interval:= ReadInteger('Configuration','Refresh',5000);
  end;
  akt:= 0;
  lcd_init;
  SpinEdit1.Value:= bufl;
  SpinEdit2.Value:= Timer.Interval div 1000;
  buffer:= '                ';
  if LPT = $278 then RadioButton1.Checked:= true
    else RadioButton1.Checked:= false;
  if LPT = $378 then RadioButton2.Checked:= true
    else RadioButton2.Checked:= false;
  f.Free;
end;

procedure TTestForm.CzasNaZnak(Sender: TObject);
{ procedura obslugi timera taktowania LCD, okres np. 10ms }
begin
  if (akt < bufl+6) then
  begin
    write_LPT(ord(buffer[akt]),FALSE);
    akt:=akt+1;
  end else Timer1.Enabled:= false;
end;

procedure TTestForm.set278(Sender: TObject);
{ procedura obslugi przycisku 278h }
begin
  LPT:=$278;
  lcd_init;
end;

procedure TTestForm.set378(Sender: TObject);
{ procedura obslugi przycisku 378h }
begin
  LPT:=$378;
  lcd_init;
end;

procedure TTestForm.buf_change(Sender: TObject);
{ procedura przy zmianie wartosci SpinEdita długości bufora }
begin
  bufl:=SpinEdit1.Value;
end;

procedure TTestForm.refresh_change(Sender: TObject);
{ procedura przy zmianie wartosci SpinEdita interwału pomiaru }
begin
  Timer.Interval:= SpinEdit2.Value * 1000;
end;

procedure TTestForm.FormClose(Sender: TObject; var Action: TCloseAction);
{ procedura przy zakończeniu aplikacji }
var
  f : TIniFile;
begin
  f := TIniFile.Create('CpuUsage.Ini');
  with f do
  begin
    WriteInteger('Configuration','Address',LPT);
    WriteInteger('Configuration','Buffer',bufl);
  WriteInteger('Configuration','Refresh',Timer.Interval);
  end;
  f.Free;
end;
```

Za pomocą przytoczonych procedur można łatwo realizować podstawowe funkcje wyświetlacza LCD, takich jak:
- konfiguracja podczas inicjalizacji,
- czyszczenie ekranu,
- powrót kursora,
- przesunięcie,
- definicja znaków do CGRAM (niezbędna do wykresu słupkowego).

a) b)

Rys. 2. Wygląd interfejsu aplikacji sterującej

Przykłady procedur realizujących powyższe operacje przedstawiono na **list. 3**.

Internetowe *open-source* są nieprzebranym źródłem pomysłów i niemal gotowych projektów. Również dzięki tej idei udało się prosto i bezboleśnie zrealizować pomiar obciążenia procesora. Kluczem okazał się tutaj gotowy komponent do Delphi realizujący tego typu pomiar. Spośród paru testowanych modułów najlepiej chyba w tej roli spisał się pakiet Alexeya Dynnikova (*aldyn@-chat.ru*) o nazwie *adCPU*. Moduł ten jest dostępny jako freeware pod adresem *http://www.aldyn.ru*. Instalacja modułu praktycznie ogranicza się do dołączenia pliku źródłowego do projektu. Dostajemy do dyspozycji trzy funkcje do pomiaru obciążenia dowolnego procesora w systemie:
- *GetCPUCount* - zwraca liczbę procesorów w systemie,
- *CollectCPUData* - zbiera informacje o aktualnym obciążeniu każdego procesora,
- *GetCPUUsage(n)* - zwraca poprzednio zebrany pomiar obciążenia dla procesora n.

Tab. 1. Przestrzeń adresowa portu Centronics

Ofset	Odczyt/ Zapis	Numer bitu	Opis
Baza + 0 Dane	Zapis	7	D7
		6	D6
		5	D5
		4	D4
		3	D3
		2	D2
		1	D1
		0	D0
Baza + 1 Status	Tylko odczyt	7	Busy
		6	/Acknowledge
		5	Paper End
		4	Select
		3	/Error
		2	/IRQ
		1	Zarezerwowane
		0	Zarezerwowane
Baza + 2 Sterowanie	Odczyt/ Zapis	7	Zarezerwowane
		6	Zarezerwowane
		5	Direction
		4	Enable IRQ
		3	/Select In
		2	Initialize
		1	/Autofeed
		0	/Strobe

Rys. 3. Przykładowe wskazanie na wyświetlaczu

Nie wnikając w zasadę działania procedur autorstwa Alexeya Dynnikova (które można poznać, analizując niezbyt długi kod źródłowy), można w prosty sposób zmierzyć obciążenie procesora. Praktyczna realizacja w projekcie Delphi składałaby się z części inicjalizacyjnej oraz cyklicznie wywoływanej procedury pomiaru i wyświetlania. Kluczem jest cykliczny pomiar wartości obciążenia, który może być wywoływany sygnałem timera. Po zebraniu danych należy również zaktualizować bufor poprzednich wartości w celu stworzenia wykresu. Byłaby to bardzo dobra koncepcja, gdyby nie konieczność stosowania sztywnych opóźnień przy sterowaniu wyświetlaczem. Taki czas bezczynności byłby dużym obciążeniem dla systemu. W związku z tym wprowadzono drugi timer, który z częstotliwością wielokrotnie większą taktuje dane dla wyświetlacza po pomiarze obciążenia. Timer ten jest aktywny przez parę cykli po pomiarze, po czym po przetransmitowaniu całości danych sam się deaktywuje. Realizacja programowa nie jest skomplikowana i można ją natychmiast zrozumieć spojrzawszy w kod źródłowy projektu (kluczowe fragmenty są przedstawione na **list. 4**).

Na **rys. 2** przedstawiono wygląd formy surowego projektu w Delphi oraz gotowej aplikacji po skompilowaniu. Jak widać, z poziomu GUI można zmieniać długość bufora (przydatne przy wyświetlaczach o różnej liczbie znaków), adres portu oraz interwał cyklu pomiarowego.

Po każdorazowym pomiarze obciążenia procesora, aktualizowany jest bufor, w którym znajdują się poprzednie wartości pomiarów. Następnie aktualna wartość pomiaru jest transmitowana do wyświetlacza w postaci numerycznej (np. 56%) oraz poprzednie wartości z bufora w postaci wykresu słupowego (dzięki znakom zdefiniowanym do CGRAM). Przykładowe wskazanie na wyświetlaczu mogłoby więc wyglądać jak na **rys. 3**. Długość wykresu jest oczywiście zależna od długości bufora oraz ilości dostępnych pól na wyświetlaczu.

Jak już wspomniano, kluczowe fragmentu kodu źródłowego projektu przedstawiono na list. 4. Kod ten zawiera również wiele dodatkowych elementów, które z punktu widzenia tego projektu są mniej istotne, a służą głównie ergonomii użycia aplikacji. Są to m.in. zapamiętywanie i odtwarzanie poprzednich ustawień w pliku *CpuUsage.Ini* oraz formatowanie bufora i tytułu aplikacji. Czytelnicy bardziej zainteresowani stroną programistyczną z pewnością łatwo zrozumieją sens tego kodu i być może wzbogacą go o własne pomysły. Aplikację tę można w prosty sposób rozbudować o kolejne elementy, jak chociażby wyświetlanie innych danych w drugiej linijce wyświetlacza (np. aktualny czas systemowy). Dodatkowo można ją rozbudować o możliwość pracy w systemie wieloprocesorowym, np. przez cykliczne wyświetlanie wskazań dla kolejnych procesorów. W niektórych komputerach również adres portu równoległego nie jest ustawiony na żadną z „historycznych" wartości 278h lub 378h. W związku z tym, sposób ustawiania adresu można również łatwo zmodyfikować. Delphi daje w tym względzie wręcz nieograniczone możliwości.

Montaż i uruchomienie urządzenia nie powinny przysparzać żadnych trudności. Cały układ bez wyświetlacza mieści się wewnątrz obudowy złącza DB25. Otwartą kwestią pozostaje zasilanie, które niestety nie jest wyprowadzone na port Centronics, w związku z czym niezbędne jest dołączenie zasilacza zewnętrznego. Aby tego uniknąć, można zastosować sztuczkę polegającą na „podczepieniu" się do zasilania komputera na złączu klawiaturowym, gameporcie lub USB. W praktyce polecam rzadko wykorzystywany gameport (zasilanie na wszystkich skrajnych stykach - napięcie +5 V występuje na stykach 1, 8, 9 i 15, masa to styki 4 i 5).

Jarek Paluszyński
jarekp@ict.pwr.wroc.pl

WYKAZ ELEMENTÓW

Rezystory
P1: 10kΩ
Kondensatory
C1: 100µF/16V
C2: 10µF/16V
Półprzewodniki
U1: ICL7660/7660S
Różne
W1: dowolny wyświetlacz LCD 2x20 znaków
ZI1: wtyk DB25
JP1: goldpiny 1x2 + jumper