Wyświetlacz LCD sterowany z portu USB

AVT–952

Możliwość bezpośredniego sterowania wyswietlaczem LCD z komputera, np. przez interfejs RS232, może być czasami bardzo cenna, tyle że... we wspólczesnych komputerach nie zawsze mamy do dyspozycji port szeregowy. Do tego celu trzeba więc zaprząc wszechobecne USB.

Rekomendacje:

zastosowania wyświetlacza LCD sterowanego bezpośrednio z portu USB w dużej mierze zależą od pomysłowości jego użytkownika i umiejętności napisania własnego oprogramowania uruchamianego na komputerze PC. Mogą to być np. różnego rodzaju ostrzeżenia lub informacje o stanie pracy komputera, które będą widoczne nawet po uśpieniu monitora.

PODSTAWOWE PARAMETRY

- Płytka o wymiarach 38x38 mm
- Zasilanie bezpośrednio z portu USB
- Obsługa wyświetlacza alfanumerycznego ze sterownikiem zgodnym z HD44780
- Obsługa własnych znaków zapisywanych w pamięci CG-RAM wyświetlacza

Dodatkowe materiały do artykułu publikujemy na CD-EP oraz www.ep.com.pl

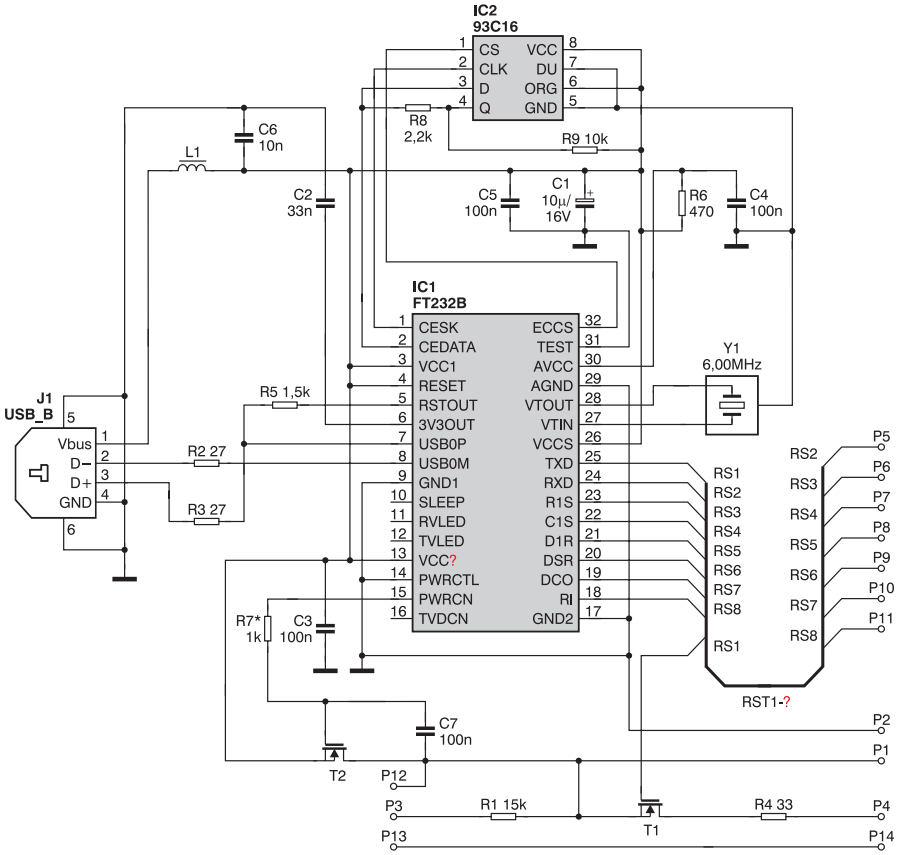
Temat sterowania alfanumerycznym wyświetlaczem LCD przy pomocy portów komputera doczekał się już wielu opracowań. Do tej kolekcji dołączymy prezentowany projekt. Wykorzystano w nim popularny układ FT232BM „wzmocniony" o łatwy w użyciu i modyfikacji komponent Delphi pozwalający na nieskomplikowane dopisywanie własnych aplikacji sterujących.

Schemat przejściówki USB<–>LCD pokazano na **rys. 1**. Zauważamy od razu brak jakiegokolwiek mikrokontrolera, który w tradycyjnych rozwiązaniach tego typu przyjmuje komendy i dane przez UART z aplikacji PC i na tej podstawie realizuje różne operacje sterowania. Tutaj zamiast typowej funkcji FT232, jaką jest konwersja USB<–>RS232 wykorzystano dodatkowy tryb pracy kostki, tzw. *BitBang*.

W trybie *BitBang* linie portu UART mogą być dowolnie skonfigurowane jako wyjściowe lub wejściowe linie danych, których zapis lub odczyt odbywa się cyklicznie, zgodnie z ustawioną prędkością. Nic więc nie stoi na przeszkodzie, aby takim 8–bitowym portem wysterować bezpośrednio popularny kontroler LCD HD44780 ustawiony na pracę z 4–bitowym słowem danych.

W projekcie zastosowano typowy, zalecany przez producenta schemat aplikacyjny układu FT232 zasilanego z magistrali, uzupełniony o kilka elementów sterowania podświetleniem i kontrastem. Zastosowanie miniaturowych elementów SMD pozwoliło na złożenie modułu na niewielkiej, dwuwarstwowej płytce (**rys. 2**). Nie obyło się jednak niestety się bez niespodzianek wynikających z braku – w czasie opracowania PCB – pełnej informacji o nieoczekiwanym zachowaniu się dodatkowych linii kontrolnych (PWRCTL, PWREN) układu FT232BM pracującego w trybie *BitBang*. Okazało się, że PWRCTL zmienia funkcję z wejścia na wyjście ze stanem wysokim, zaś PWREN staje się wejściem z pullupem, co uniemożliwia jego dalsze funkcjonowanie jako włącznika zasilania zewnętrznego obciążenia. Serwis Ftdi–Chip również przyznał się do błędu w kostce i obiecywał poprawienie w nowszych wydaniach. W związku z tym, płytki czekały przez dłuższy czas na nowsze serie FT232 – i niestety się nie doczekały. Zamiast poprawiać stare, producent stworzył zupełnie nowe serie układów (FT2232C, FT232R) o znacznie rozszerzonych możliwościach i uproszczonym stosowaniu. W końcu okazało się, że układu FT232BM w opisywanym module nie da się zmusić do całkowicie poprawnego działania.

Jednak przygotowane płytki nie zostały skazane na kasację. Drobne przeróbki pozwoliły na uruchomienie modułu, może nie do końca zgodnie ze specyfikacją USB, ale zapewniając zakładaną funkcjonalność sterowania wyświetlaczem. Polegały one na:



Rys. 1. Schemat adaptera USB<–>LCD

WYKAZ ELEMENTÓW

Rezystory

(0805 z wyjątkiem R4)

R1: 15 kΩ

R2, R3: 27 Ω

R4: 33 Ω 1206 – dobrany w zależności od prądu podświetlenia LED

R5: 1,5 kΩ

R6: 470 Ω

R7: 1 kΩ

R8: 2,2 kΩ

R9: 10 kΩ

Kondensatory

(0805 z wyjątkiem C1)

C1: 10 µF/16 V SMD tantal

C2: 33 nF

C3...C5, C7: 100 nF

C6: 10 nF

Półprzewodniki

U1: FT232BM

U2: 93C46

Q1: BSP 315 (jest dla obecnych potrzeb mocno przewymiarowany, jednak chodziło o uzyskanie na nim jak najmniejszego spadku napięcia)

Q2: MMBF 2202

Inne

L1: koralik ferrytowy przewlekany

J1: gniazdo USB typu B

Y1: rezonator ceramiczny 6,000 MHz

listwa goldpin

potencjometr regulacji kontrastu – Helitrim 20 kΩ

– przecięciu ścieżki łączącej pin 14 (PWRCTL) z masą – układ pracuje właściwie z wejściem „pływającym”,

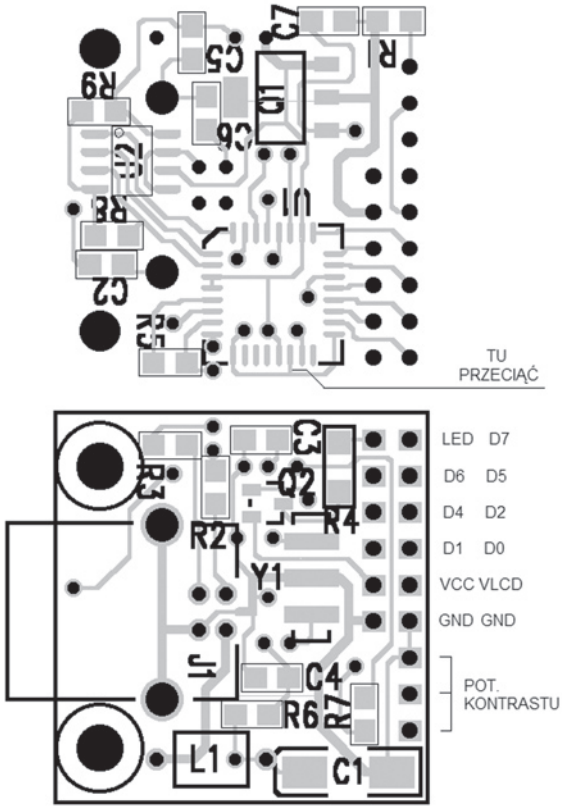
– dolutowaniu rezystora 10...15 kΩ pomiędzy pin PWREN, a masę (miniaturowy rezystor osiowy dobrze dopasowuje się między polami lutowniczymi rezystora R7 i kondensatora C1 (czarne kropki na schemacie) – w ten sposób wysoki stan na wyjściu PWREN wyłącza tranzystor Q2, a stan niski oraz niepożądane przejście w tryb wejścia z pullupem ok. 200 kΩ pozostawiają zasilanie wyświetlacza włączone. Możliwe, że taki układ nie zachowa się prawidłowo podczas wejścia hosta w uśpienie, nie zostało to sprawdzone.

Linie sterujące są wyprowadzone na dwurzędową listwę goldpin 2x6 (LED – podświetlenie, VLCD – napięcie matrycy odpowiedzialne za kontrast). Linie danych odpowiadają funkcji pinów FT232 w trybie *BitBang* (zauważmy, że nie ma tu linii D3 – została zużyta na sterowanie podświetleniem). Dodatkowe 3 pola lutownicze służą do podłączenia zewnętrznego potencjometru kontrastu (15...20 kΩ). W pozbawionym obudowy prototypie został tam bezpośrednio wlutowany pionowy helitrim 20 kΩ.

Wyświetlacz 1*16 został do testów podłączony taśmą. Przypisanie linii sterujących HD44780 do wyjść modułu jest w zasadzie dowolne (analogiczne ustawienia wprowadzimy w kodzie programu PC) Jednak dla wygody dobrze jest zapewnić zgodność linii danych (D4...D7), gdyż oszczędzi to zbędnych konwersji oryginalnych wartości bajtów na wartości wynikające z rzeczywistego polutowania. Tak też zostało zrobione w próbnym układzie (D4...D7

List. 1. Procedury kontroli wyświetlacza – *FtBbLcd.pas*

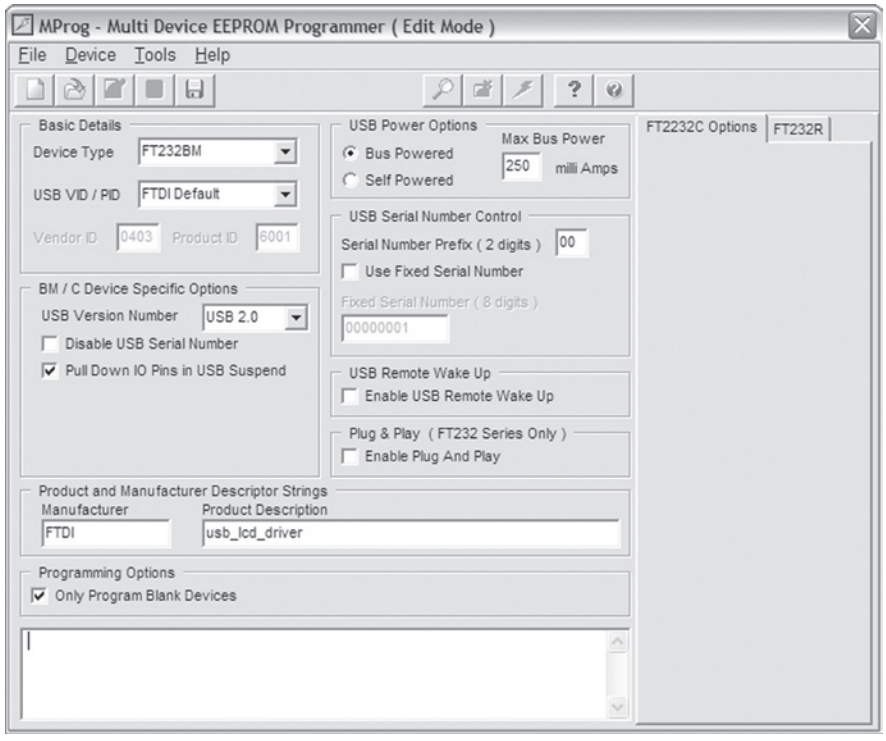
```
public
   { Public declarations }
   constructor Create(Owner:TComponent);override;

   property DevicePresent:Boolean read fDevicePresent write SetDevicePresent;
   property DeviceHandle:THandle read fDevHandle;
   property Light:Boolean read fLight write SetLight;
   property Line1:String write SetLine1;
   property Line2:String write SetLine1;
   property Line3:String write SetLine1;
   property Line4:String write SetLine1;
   property CgRam:String write SetCgRam;

 published
   { Published declarations }
   property DeviceDescription:String read fDeviceDescription write fDeviceDescription;
   property DeviceSerialNumber:String read fDeviceSerialNumber write fDeviceSerialNumber;
   property DeviceOpenBy:TDeviceOpenBy read fDeviceOpenBy write fDeviceOpenBy;
   property LinesNum:Integer read fLinesNum write SetLinesNum;
   property LineLength:Integer read fLineLength write SetLineLength;
   property DoubleLine:Boolean read fDoubleLine write fDoubleLine;
```



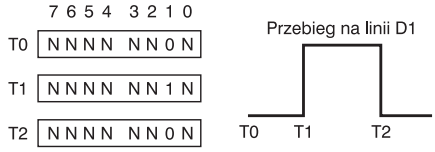
Rys. 2. Płytka drukowana modułu

– dane w trybie 4–bitowym, D2 – R/W, D1 – RS, D0 – E). Linia R/W została podłączona „do kompletu", chociaż program jej nie używa i utrzymuje cały czas w stanie niskim (można więc po prostu wejście R/W wyświetlacza dołączyć do masy).

Tryb *BitBang* wyklucza najprostszy sposób obsługi FT232 jako dodatkowego portu COM przy pomocy sterowników VCP. Musimy użyć sterowników FTD2xx, które instalujemy przy pierwszym podłączeniu układu do PC po monicie systemu (Windows XP) o wykryciu nowego urządzenia. FTD2xx są również niezbędne dla uruchomienia programu konfigurującego Mprog (dostarczanego przez FTDI). Przy pomocy Mprog ustawiamy potrzebne parametry pracy kostki: zasilanie z magistrali, prąd 250 mA, WakeUp i PnP wyłączone jako zbędne, nazwa „usb_lcd_driver" będzie używana w programie do połączenia się z układem, pulldown w trybie zawieszenia włączony (**rys. 3**). Zwróćmy uwagę, że po tych zmianach Windows potraktuje nasz moduł jako zupełnie nowe urządzenie i przy ponownym uruchomieniu będziemy musieli ponowić procedurę instalowania sterowników FTD2xx.



Rys. 3. Okno programu konfigurującego Mprog 2.8



Rys. 4. Sterowanie wybraną linią w trybie BitBang poprzez wysyłanie sekwencji bajtów

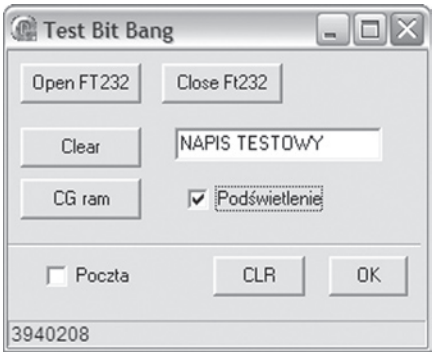
Po tych wszystkich przygotowaniach mamy gotowy sprzętowy interfejs, który wykorzystamy do programowego sterowania wyświetlaczem.

Testowa aplikacja sterująca została napisana dla systemu Windows XP przy pomocy środowiska BDS 2005 PE (cały kod źródłowy jest udostępniony w materiałach pomocniczych). Funkcje obsługi trybu *BitBang* są zawarte w module *u_ft32bbapi.pas*. Natomiast wszystkie procedury kontroli wyświetlacza zostały opakowane w wygodną formę komponentu Delpni (*FtBbLcd.pas*). Jego metody i właściwości znacznie ułatwiają późniejsze stosowanie w programie (**list. 1)**:

DevicePresent służy do otwierania i zamykania urządzenia. *Light* włącza i wyłącza podświetlenie. *Line* oraz *CgRam* wpisują treść poszczególnych linii wyświetlacza oraz zawartość znaków użytkownika. Właściwość *DeviceOpenBy* ustawiamy na *OpenByDescription*, a jako opis (*DeviceDescription*) wstawiamy nazwę zaprogramowaną przed chwilą w *eeprom* modułu (*usb_lcd_driver*). *LinesNum* i *LineLength* określają rozmiar wyświetlacza, zaś *DoubleLine* przewiduje wykonanie niektórych jednoliniowych wyświetlaczy z adresowaniem drugiej połówki wiersza, jak drugiej linii (od 0x40 – tak było np. w testowanym wyświetlaczu).

W kodzie możemy zwrócić uwagę na technikę sterowania poszczególnymi liniami (np. E). Takie sterowanie wymaga operacji na poszczególnych bitach, które są w trybie *BitBang* nieosiągalne – zawsze wystawiamy na magistralę pełny kolejny bajt. Dlatego musimy w takim przypadku przygotować odpowiedni bufor (tablicę bajtów) ze zmienianym tylko potrzebnym bitem, a resztą pozostawioną bez zmian. Wysłanie takiego bufora daje w efekcie oczekiwane przebiegi na wybranej linii (**rys. 4**).

Komponent (będący w fazie testów) nie został zainstalowany w środowisku BDS, ale jest dynamicznie tworzony w aplikacji kontrolnej. Testowy program nie aspiruje do miana jakiegoś uniwersalnego narzędzia



Rys. 5. Okienko prostej aplikacji testowej

(typu np. *Smartie*) – ma posłużyć głównie jako przykład dla samodzielnych realizacji (**rys. 5**).

Ta mała aplikacja potrafi przetestować włączanie i wyłączanie modułu, podświetlenie, zapis i wyświetlenie zawartości CG–RAM oraz przepisanie pola edycyjnego na wyświetlacz. Jako przykład praktycznego wykorzystania została dołożona funkcja okresowego sprawdzania liczby nowych maili w wybranej skrzynce pocztowej i wyświetlenie tej informacji na wyświetlaczu (analogicznie jak w prezentowanym na łamach EP projekcie, gdzie do sygnalizacji służyły diody LED). Warto przy okazji zwrócić uwagę na to, jaki potencjał oferują bezpłatne pakiety komponentów dla Delphi. Przy pomocy zainstalowanego w BDS2005PE pakietu sieciowego *Indy 10*, taką – w rzeczywistości rozbudowaną i skomplikowaną operację – „załatwiamy" jednym krótkim kawałkiem kodu wykorzystującym metody komponentu *IdPOP3* (**list. 2**).

W projekcie Delphi musimy oczywiście podać wszystkie własne opcje dostępu do skrzynki pocztowej – załączony kod jest ich z oczywistych względów pozbawiony. W ogóle – ze względu na ochronę swoich danych – jeśli nasz program czy projekt ma trafiać w obce ręce raczej nie umieszczajmy w nim wszelkiego rodzaju haseł oraz identyfikatorów bezpośrednio ale wczytujmy dynamicznie z odpowiednio zaszyfrowanych plików, wpisów w rejestrze itp.

List.2. Sterowanie wyświetlaczem LCD za pomocą komponentu *IdPOP3*

```
function TM_form.CheckMails:Longint;
begin
     CheckTimer.Enabled:=False;
     try
     IdPOP3.Connect;
     Connected:=True;
     Result:=IdPOP3.CheckMessages;
     IdPOP3.Disconnect;
     Connected:=False;
     except
       StatBar.SimpleText:=('Błąd połączenia z serwerem POP3');
       Result:=0;
     end;
     CheckTimer.Enabled:=True;
end;

procedure TM_form.CheckTimerTimer(Sender: TObject);
begin
     if FindWindow(nil,'Skrzynka odbiorcza - Outlook Express') = 0 then
Inc(CheckCounter);
     if CheckCounter=300 then begin
     // 5 minut
        CheckCounter:=0;
     MailNum:=CheckMails;
     if (MailNum > LastMailNum) then begin
        FtLcd.DevicePresent:=True;
        FtLcd.Light:=True;
        FtLcd.Line1:='NOWE MAILE : '+IntToStr(MailNum);
        FtLcd.DevicePresent:=False;
        end;
     LastMailNum:=MailNum;
     end;
end;
```

Jedyną niewdzięczną i pracochłonną czynnością pozostało przygotowanie tablic opisujących własne znaki w obszarze CG–RAM wyświetlacza. Dlatego przy okazji został szybko dopisany (również w BDS 2005PE) mały program narzędziowy do realizacji tylko tego konkretnego celu (**rys. 6).**

Program ten pozwala na graficzną edycję 8 znaków CG–RAM oraz przechowywanie 64 wzorów znaków w podręcznym magazynku. Każdy znak możemy wyposażyć w komentarz. Wyprodukowany na podstawie wprowadzonych szablonów graficznych kod tablicy (Delphi lub avr–gcc) nie jest zapisywany w oddzielnym pliku, ale bezpośrednio do schowka Windows, skąd wklejamy go sobie w potrzebnym miejscu programu – przykład na **list. 3**.

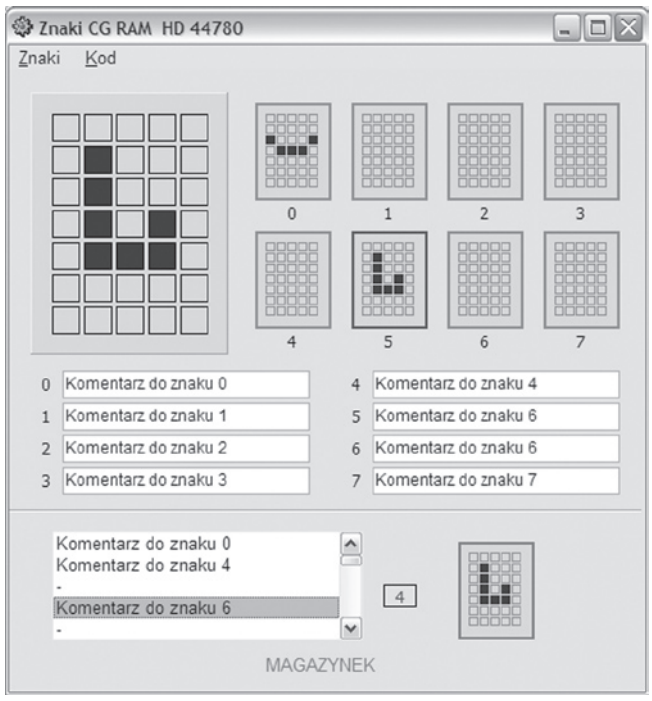
Kod źródłowy narzędzia jest również udostępniony jako całkowicie wolny – do wszelkich własnych przeróbek i udoskonaleń.

Zauważmy na koniec, że prezentowane oprogramowanie oraz procedura uruchamiania może być z powodzeniem zastosowana po ewentualnym samodzielnym przeprojektowaniu modułu na nowszą wersję FT232R.

Jerzy Szczesiul
jerzy.szczesiul@ep.com.pl

List. 3. Program do graficznej edycji znaków CG–RAM

```
const CgRamTable:array[0..63] of
Byte =
(
$10,$10,$12,$14,$18,$10,$1F,$00,
//znak 0: Komentarz do znaku 0
$15,$11,$19,$15,$13,$11,$11,$00,
//znak 1: Komentarz do znaku 1
$04,$0F,$10,$0E,$01,$01,$1E,$00,
//znak 2: Komentarz do znaku 2
$0E,$11,$11,$1F,$11,$11,$02,$00,
//znak 3: Komentarz do znaku 3
$04,$0E,$11,$10,$10,$11,$0E,$00,
//znak 4: Komentarz do znaku 4
$04,$1F,$02,$04,$08,$10,$1F,$00,
//znak 5: Komentarz do znaku 5
$1F,$10,$10,$1E,$10,$1F,$02,$00,
//znak 6: Komentarz do znaku 6
$04,$0E,$11,$11,$11,$11,$0E,$00
//znak 7: Komentarz do znaku 7
);
```



Rys. 6. Okno pomocniczego programu do generacji zawartości CG–RAM