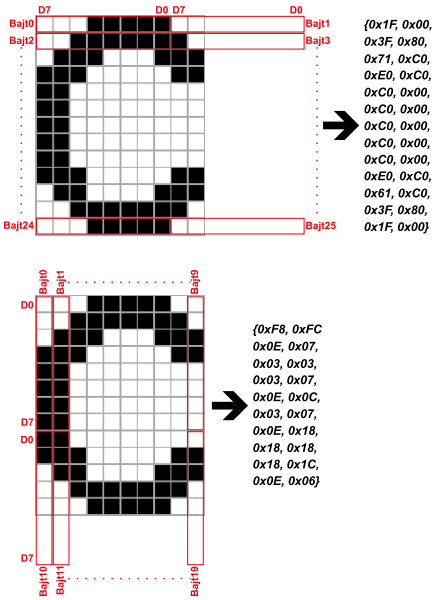
PixeLab – fabryka fontów

Projektowanie czcionek ekranowych w systemach mikroprocesorowych

Większość elektroników-programistów wcześniej lub później napotyka potrzebę użycia wyświetlacza graficznego w projektowanym systemie mikroprocesorowym. Nic w tym dziwnego, ponieważ peryferia tego typu dają znacznie większe możliwości utworzenia atrakcyjnego interfejsu użytkownika, niż proste wyświetlacze alfanumeryczne. Co więcej, dobrej wyświetlacze graficzne typu COG (z kontrolerem na szkle) osiągnęły już taki pułap cenowy, dzięki któremu z powodzeniem mogą stać się alternatywą dla alfanumerycznych.

Dodatkowe informacje:
Plik instalacyjny aplikacji zamieszczono w materiałach dodatkowych. Nosi nazwę PixeLab_v1.0.7z i jest zabezpieczony hasłem „pixelab".

W większości wypadków, budowa systemu mikroprocesorowego z użyciem wyświetlacza graficznego wymaga implementacji skutecznego mechanizmu obsługi czcionek ekranowych oraz dostarczenia samych wzorców czcionek, co nie jest zadaniem łatwym. Oczywiście, istnieją nieliczne wyświetlacze graficzne mające tryb znakowy z zaimplementowaną obsługą jednego lub więcej rodzajów czcionek, ale moim zdaniem są to rozwiązania, które mają więcej wad, niż zalet, gdyż ograniczają programistę w kwestii budowy interfejsu użytkownika. Tak, czy inaczej i niezależnie od zastosowanego wyświetlacza graficznego, niezbędne jest narzędzie, za pomocą którego można wygenerować odpowiednie tablice wzorców czcionek, ponieważ tak naprawdę wyświetlanie czcionki na ekranie wyświetlacza graficznego ogranicza się do wyświetlenie prostej bitmapy reprezentującej wygląd wybranego znaku.



Rysunek 1. Sposób podziału przykładowego znaku „c" na poszczególne bajty danych w zależności od organizacji pamięci ekranu

W swojej praktyce wielokrotnie stosowałem różne wyświetlacze graficzne, od monochromatycznych o rozdzielczości 128×64 piksele, poprzez doskonałej jakości wyświetlacze OLED (kolorowe i monochromatyczne), a skończywszy na kolorowych wyświetlaczach TFT o dużej rozdzielczości, wyposażonych w zintegrowany, pojemnościowy panel dotykowy. Za każdym razem, niezależnie od wykorzystywanych funkcji narzędziowych, podstawą do implementacji obsługi takich czcionek był program, za pomocą którego było możliwe szybkie wygenerowanie stosownych tablic i struktur opisujących używany font. W tym celu posługiwałem się wieloma programami komputerowymi (darmowymi i płatnymi), przy udziale których w sposób „prawie" automatyczny byłem w stanie wygenerować stosowne zbiory danych. Niestety, jak pokazała moja wieloletnia praktyka, programy takie pozostawiały sporo do życzenia lub też nie do końca nadawały się do zastosowania w przypadku konkretnego wyświetlacza graficznego i związanej z nim organizacji pamięci obrazu, co pociągało za sobą żmudne i długotrwałe procesy edycyjne, jakim musiałem poddać tak otrzymane zbiory danych. Powstał, więc pomysł, by przygotować aplikację PC, która umożliwiałaby automatyzację tego procesu a jednocześnie dawała możliwość przygotowania stosownych struktur danych dla różnych typów wyświetlaczy graficznych dostosowanych każdorazowo do specyficznej organizacji pamięci ekranu danego typu peryferium.

Moja wiedza w temacie projektowania aplikacji PC uległa, mówiąc oględnie, przedawnieniu, gdyż sięga jeszcze czasów mojego ulubionego Delphi (a wcześniej Pascala), poprosiłem swojego kolegę, Marcina Popławskiego o pomoc w realizacji takiej, uniwersalnej aplikacji. Efektem naszej współpracy jest, nie boję się tego powiedzieć, jedna z najlepszych aplikacji do implementacji fontów w systemach mikroprocesorowych – **PixeLab**. Zanim jednak przejdę do szczegółów dotyczących samej aplikacji, przedstawić muszę podstawowe informacje, które niezbędne są z punktu widzenia mechanizmu obsługi czcionek ekranowych (w naszym wydaniu rastrowych, czyli nieskalowalnych). Analizując sposób organizacji pamięci ekranu wyświetlaczy graficznych, z jakimi miałem okazję się spotkać (dotyczy to głównie wyświetlaczy monochromatycznych) można wyróżnić dwa, główne sposoby podziału pamięci ekranu w świetle znaczenia bajtów danych do nich przesyłanych:

Podział horyzontalny, czyli taki, gdzie kolejne, wysyłane do wyświetlacza bajty danych reprezentują stan ośmiu, następujących po sobie pixeli obrazu ułożonych w poziomie.

Podział wertykalny, czyli taki, gdzie kolejne, wysyłane do wyświetlacza bajty danych reprezentują stan ośmiu, następujących po sobie pixeli obrazu ułożonych w pionie.

Jest to podstawowa różnica, jeśli chodzi o sposób generowania danych reprezentujących poszczególne znaki wybranej czcionki

Listing 1. Struktura *fontType* opisująca najważniejsze parametry zestawu znaków

```
typedef struct
{
    uint8_t Height;  //Wysokość znaków (px)
    uint8_t FirstCharCode;  //Kod ASCII pierwszego znaku w tablicy wzorców
znaków
    uint8_t Interspace;  //Szerokość odstępu pomiędzy znakami (px)
    uint8_t SpaceWidth;  //Szerokość spacji (px)
    charType *CharInfo;  //Wskaźnik do tablicy opisującej parametry
poszczególnych znaków
    uint8_t *Bitmap;  //Wskaźnik do tablicy wzorców poszczególnych znaków

} fontType;
```

ekranowej i dla niektórych typów wyświetlaczy (zwłaszcza TFT) może podlegać konfiguracji sprzętowej. Ta podstawowa różnica zmienia sposób, w jaki należy analizować „treść" obrazu poszczególnych znaków a co za tym idzie, zmienia wynikową tablicę danych reprezentujących interesujący nasz znak. Sposób podziału przykładowego znaku „c" na poszczególne bajty danych, narysowanego przy użyciu czcionki Arial 18pt, w zależności od organizacji pamięci ekranu pokazano na **rysunku 1**.

Jak widać, wielkość wynikowej tablicy danych reprezentującej edytowany znak (umieszczonej w tablicy wzorców znaków) zależy w tym przypadku od sposobu interpretacji poszczególnych bajtów danych. W celu efektywnej implementacji mechanizmu wyświetlania tak zdefiniowanych czcionek ekranowych wprowadzono 2 nowe, srukturalne typy danych oraz tablicę zawierającą wzorce znaków. Wspomniane elementy to:

Struktura *fontType*, która opisuje najważniejsze parametry całego zestawu znaków (**listing 1**).

Struktura *charType*, która opisuje parametry poszczególnych znaków znajdujących się w tablicy wzorców (**listing 2**).

Tablica wzorców znaków typu *uint8_t* zawierająca wzorce poszczególnych znaków w ramach całego zestawu znaków.

Warto podkreślić, iż strukturę *charType* wprowadzono wyłącznie z uwagi na fakt, że poszczególne znaki (ich wzorce) mogą się różnić, jeśli chodzi o ich szerokość, gdyż łatwo się domyślić, że dla przykładu, szerokość znaku „w" będzie większa, aniżeli szerokosc znaku „i", więc bez wprowadzenia tejże struktury (a w zasadzie tablicy takich struktur) nie byłoby możliwe znalezienie w tablicy wzorców znaków (tablicy ***uint8_t***) początku wzorca konkretnego znaku (a w zasadzie jego offsetu w odniesieniu do początku tablicy). Dla znaków o stałej szerokości (tzw. *fixed*) struktura taka nie byłaby potrzebna, gdyż szerokość każdego znaku byłaby taka sama w związku, z czym parametr ten można by uczynić elementem struktury ***fontType***. Przejdźmy zatem do przykładu praktycznego, w zakresie którego pokażę wspomniane powyżej 3 elementy danych (2 struktury i jedną tablicę) dla przypadku zestawu znaków składającego się wyłącznie z dwóch znaków „a" i „c" wygenerowanych dla czcionki Arial 18pt. Wspomniane elementy danych pokazano na **listingu 3**.

Jak widać, tablica struktur typu ***charType*** zawiera „pusty" wpis {0,0} dotyczący znaku „b", który nie występuje w naszej tablicy wzorców znaków. Tego rodzaju podejście upraszcza w sposób znaczący funkcję wyświetlającą znaki na ekranie wyświetlacza graficznego, gdyż łatwiej odnajdujemy

Listing 2. Struktura *charType* opisująca parametry znaku

```
typedef struct
{
    const uint8_t Width;  //Szerokość wybranego znaku (px)
    const uint16_t Offset;  //Offset wzorca wybranego znaku w tablicy wzorców
znaków

} charType;
```

Listing 3. Elementy danych dla przypadku zestawu znaków składającego się wyłącznie z dwóch znaków „a" i „c".

```
fontType Arial18font PROGMEM =
{
    13, //Wysokość znaków (px)
    ,a', //Kod ASCII pierwszego znaku w tablicy wzorców znaków
    1, //Szerokość odstępu pomiędzy znakami (px)
    1, //Szerokość spacji (px)
    Arial18characters, //Wskaźnik do tablicy opisującej parametry
poszczególnych znaków
    Arial18bitmaps  //Wskaźnik do tablicy wzorców poszczególnych znaków
};

charType Arial18characters[] PROGMEM =
{
    {11, 0}, // Znak: a (97)
    {0, 0}, // Znak: b (98)
    {10, 26}, // Znak: c (99)
}

uint8_t Arial18bitmaps[] PROGMEM =
{
// Znak: a, Offset: 0 , Szerokość: 11
    0x1F, 0x00,
    0x7F, 0x80,
    0xE1, 0xC0,
    0xC0, 0xC0,
    0x00, 0xC0,
    0x07, 0xC0,
    0x3F, 0xC0,
    0x78, 0xC0,
    0xC0, 0xC0,
    0xC0, 0xC0,
    0xE3, 0xC0,
    0x7F, 0xC0,
    0x3C, 0x60,
//Znak: c, Offset: 26 , Szerokość: 10
    0x1F, 0x00,
    0x3F, 0x80,
    0x71, 0xC0,
    0xE0, 0xC0,
    0xC0, 0x00,
    0xC0, 0x00,
    0xC0, 0x00,
    0xC0, 0x00,
    0xC0, 0x00,
    0xE0, 0xC0,
    0x61, 0xC0,
    0x3F, 0x80,
    0x1F, 0x00
}
```

w takim wypadku parametry interesującego nas znaku (jego szerokość i offset wzorca). W jaki sposób odnajdujemy w tablicy wzorców znaków interesujący nas wzorzec? Powiedzmy, że szukamy wzorca dla znaku „c" (kod ASCII = 99). W pierwszym kroku sprawdzamy w strukturze ***fontType*** jaki jest kod ASCII pierwszego znaku znajdującego się w tablicy wzorców znaków i odnajdujemy tam znak „a" o kodzie ASCII równym 97. W takim wypadku od kodu ASCII szukanego znaku („c") odejmujemy kod ASCII pierwszego znaku ('a") otrzymując wartość 2 (99-97), która wskazuje nam na drugi element tablicy opisującej kolejne znaki, czyli tablicy ***charType***. Drugim elementem tej tablicy (pamiętajmy, że w języku C elementy tablicy numerowane sa od 0) jest struktura **{10, 26}**, która mówi nam, że szukany znak „c" ma szerokość 10 pikseli, zaś początek wzorca tego znaku znajduje się pod indexem 26 w tablicy wzroców znaków. Prawda, że proste! Zobaczmy w takim razie, jak wykorzystać tak przygotowane dane. Pierwszą funkcją, jaka niezbędna jest z punktu widzenia obsługi czcionek ekranowych, jest funkcja, przy pomocy której ustawiamy parametry bieżącej czcionki, a która to korzysta z globalnej zmiennej *currentFont* typu *fontType*, zaś jej argumentem jest wskaźnik do struktury danych opisujących wykorzystywany zestaw znaków. Listing tejże funkcji pokazano na **listingu 4**.

Wywołanie tej funkcji powoduje przyporządkowanie wszystkim polom zmiennej strukturalnej *currentFont* typu *fontType* parametrów bieżącej czcionki ekranowej. Jak wykorzystać te wszystkie dane w przypadku konkretnego sterownika ekranu pokażę na przykładzie sterownika SSD1963 stosowanego nader często w modułach wyświetlaczy TFT. Będziemy korzystali z wyświetlacza, dla którego każdy piksel obrazu opisują 2 kolejne bajty danych przechowujące informacje o jego kolorze w formacie RGB565, czyli 5 bitów odpowiedzialnych za składową czerwoną (red), 6 bitów odpowiedzialnych za składową zieloną (green) i 5bitów odpowiedzialnych za składową

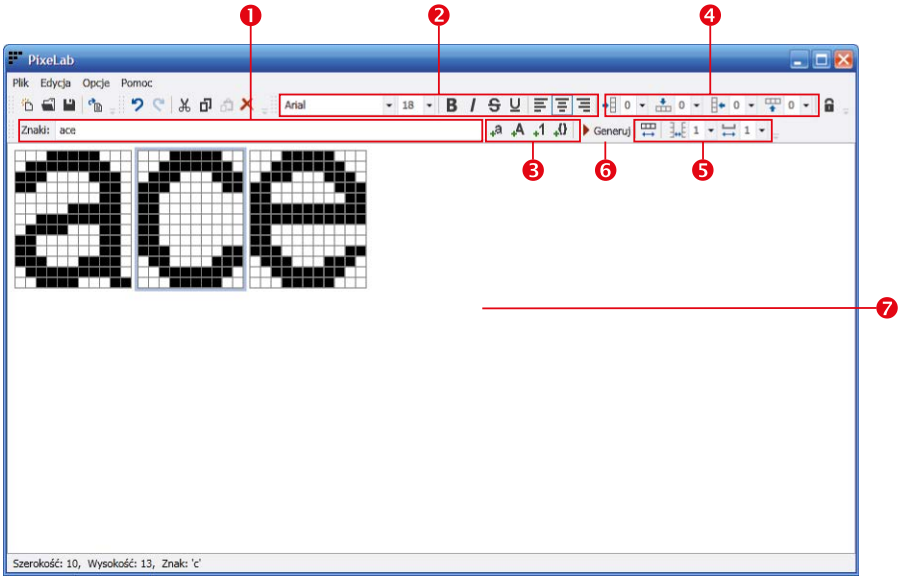
niebieską (blue). Wyświetlacz ze sterownikiem SSD1963 ma horyzontalną organizację pamięci ekranu, co oznacza, że przesłanie każdych 2 bajtów do pamięci ekranu przesuwa wskaźnik pamięci ekranu w poziomie o jeden piksel, aż do osiągnięcia maksymalnej wartości indeksu w poziomie (np. 319, dla wyświetlacza o rozdzielczości w poziomie równej 320), po czym następuje przejście do kolejnej linii (zwiększenie wskaźnika pamięci w pionie) i zerowanie wskaźnika pamięci ekranu w poziomie. Sterownik tego wyświetlacza, podobnie jak ma to miejsce w innych przypadkach, umożliwia także zdefiniowanie tzw. okna zapisu do pamięci ekranu, czyli zawężenie pamięci ekranu do obszaru okna określonego parametrami (współrzędne X i Y) jego dwóch wierzchołków: lewego górnego i prawego dolnego. Takie podejście znacznie upraszcza wyświetlanie czcionek ekranowych, gdyż nie musimy co wiersz (lub kolumnę) ustawiać wskaźnika do miejsca zapisu, gdyż ten inkrementowany jest automatycznie w obrębie okna zapisu (tę funkcjonalność realizuje funkcja *setActiveWindow*). Listing stosownej funkcji odpowiedzialnej za wyświetlenie znaku pokazano na **listingu 5**.

Listing 4. Listing funkcji odpowiedzialnej za ustawienie parametrów bieżącej czcionki ekranowej

```
void setCurrentFont(const fontType *Font)
{
    currentFont.Height = pgm_read_byte(&Font-> Height);
    currentFont.FirstCharCode = pgm_read_byte(&Font->FirstCharCode);
    currentFont.Interspace = pgm_read_byte(&Font->Interspace);
    currentFont.SpaceWidth = pgm_read_byte(&Font->SpaceWidth);
    currentFont.CharInfo = (charType*) pgm_read_word(&Font->CharInfo);
    currentFont.Bitmap = (uint8_t*) pgm_read_word(&Font->Bitmap);
}
```

Listing 5. Listing funkcji odpowiedzialnej za wyświetlenie znaku dla sterownika SSD1963

```
void drawChar(uint8_t X, uint8_t Y, char Char)
{
    register uint8_t Width, widthIndex, heightIndex, widthByteNr, readByte,
pixelsNr;
    uint16_t Offset;
/* Odczytujemy offset położenia wzorca znaku ASCII (zmienna char) w tablicy
wzorców znaków */
    Offset = pgm_read_word(&currentFont.CharInfo[Char-currentFont.
FirstCharCode].Offset);
/* Odczytujemy szerokość znaku ASCII (zmienna char) */
    Width = pgm_read_byte(&currentFont.CharInfo[Char-currentFont.
FirstCharCode].Width);
/* Ustawiamy aktywne okno pamięci obrazu DDRAM sterownika SSD1963 by uprościć
zapis danych */
    setActiveWindow(X, Y, X+Width-1, Y+currentFont.Height-1);
/* Komenda inicjująca zapis do pamięci ekranu DDRAM */
    writeCommand(CMD_WRITE_MEM_START);
    for(heightIndex = 0; heightIndex<currentFont.Height; ++heightIndex)
    {
        for(widthIndex = 0, widthByteNr = 0; widthIndex<Width; widthIndex +=
8, ++widthByteNr)
        {
/* Odczytujemy kolejny bajt definicji znaku umieszczony w tablicy Bitmap pod
odpowiednim adresem */
            readByte = pgm_read_byte(&currentFont.Bitmap[Offset++]);
/* Dla czcionek o szerokości niebędącej wielokrotnością liczby 8 każdy,
ostatni w wierszu bajt definicji wzorca nie jest w pełni wykorzystany, jeśli
chodzi o poszczególne bity, w związku z czym musimy ustalić użyteczną liczbę
pikseli przeznaczonych do przesłania do sterownika ekranu */
            pixelsNr = ((widthByteNr+1)*8)<=Width?  8:Width-(widthByteNr *8);
            for(uint8_t i=0; i<pixelsNr; ++i)
            {
/* Wysyłamy 2 bajty do pamięci ekranu reprezentujące kolor jednego piksela
(lub tła-czarny) */
                if(readByte & 0x80) drawPixel(0xFF);
                else drawPixel(0x00);
                readByte<<=1;
            }
        }
    }
}
```

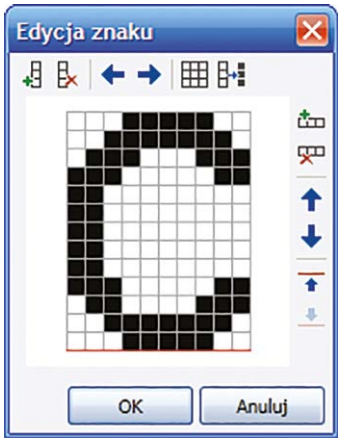


Rysunek 2. Wygląd głównego okna programu PixeLab

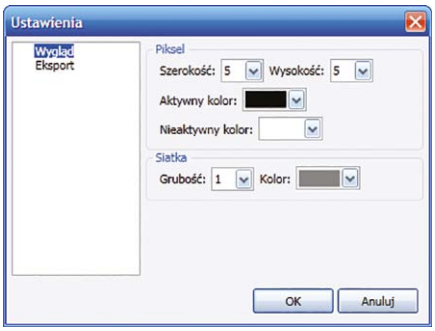
Jest to tylko zarys funkcji do wyświetlania znaków, jednak myślę, że zamieszczone komentarze wystarczą do zrozumienia sposobu, w jaki korzysta ona z danych reprezentujących parametry wybranej czcionki ekranowej (w przedstawionym przypadku, z części tychże danych). W tym momencie mamy już komplet informacji niezbędnych z punktu widzenia zrozumienia mechanizmów wyświetlania czcionek ekranowych, w związku z czym przejdźmy do niezmiernie ciekawej aplikacji, którą jest program *PixeLab* autorstwa Marcina Popławskiego. Wygląd okna programu pokazano na **rysunku 2**.

Główny obszar aplikacji to pole, które jest przeznaczone do wyświetlania wzorców uprzednio wygenerowanych znaków oraz ich późniejszej edycji (w razie potrzeby). W pasku menu aplikacji możemy wyróżnić następujące elementy dające dostęp do najczęściej wykonywanych akcji edycyjnych:

1. Pole umożliwiające wprowadzenie własnego, zupełnie dowolnego zestawu znaków, na podstawie którego wygenerowane zostaną wzorce tychże znaków (i niezbędne struktury danych). Wprowadzenie takiej możliwości umożliwia generowanie zestawu znaków zawierających wyłącznie interesujące nas wzorce znaków (tylko te, używane w danych programie aplikacji), co wyraźnie wpływa na zmniejszenie wynikowej tablicy wzorców znaków oszczędzając cenną pamięc programu mikrokontrolera.
2. Grupa ikon umożliwiająca wybór czcionki systemowej (oraz ustawienie jej parametrów), na podstawie której będą generowane wzorce znaków.
3. Grupa ikon umożliwiająca szybkie wybranie predefiniowanego zestawu znaków spośród następujących opcji: małe litery, wielkie litery, liczby, znaki specjalne.
4. Grupa ikon pozwalająca na dodanie dodatkowych marginesów (dodatkowych kolumn lub wierszy) dla wcześniej utworzonego i wybranego (podświetlonego) wzorca znaku. Należy mieć na uwadze, iż dodanie marginesu górnego czy dolnego (a więc de facto



Rysunek 3. Wygląd okna edycyjnego przykładowego wzorca znaku programu PixeLab

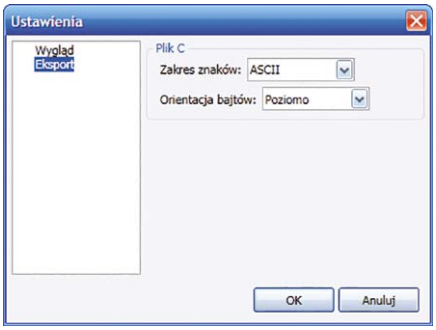


Rysunek 4. Wygląd okna ustawień aplikacji PixeLab w zakresie wyglądu pola reprezentującego wzorzec znaku

dodatkowego wiersza) spowoduje stosowną akcję dla wszystkich wygenerowanych wzorców znaków, nie zaś wyłącznie do wybranego, co wynika z faktu, iż wysokość wszystkich znaków musi być równa.

5. Grupa opcji pozwalająca na: wyrównanie szerokości wszystkich znaków (otrzymujemy zestaw znaków typu „fixed") oraz ustawienie wartości dla odstępu między znakami jak i szerokości dla znaku spacji (ostatnie dwie opcje wykorzystywane są w funkcjach wyświetlających ciągi znaków). Wyrównanie szerokości całego zestawu znaków polega de facto na ustawieniu szerokości każdego znaku na wartość znaku o największej szerokości z wygenerowanego zestawu znaków. Wspomniana opcja ma charakter „przełącznika", w związku z czym każdorazowe przyciśnięcie stosownej ikony powoduje zmianę tej właściwości całego zestawu znaków co jednocześnie możemy obserwować na ekranie głównym aplikacji.
6. Użycie ikony powoduje wygenerowanie wzorców znaków na podstawie wcześniej ustawionych parametrów i pokazanie ich na ekranie głównym aplikacji.

Jak napisano, wygenerowaniu zestawu wzorców znaków towarzyszy pojawienie się ich wzorców na powierzchni głównego obszaru aplikacji „7". Każdy z tak otrzymanych wzorców znaków możemy następnie poddać edycji poprzez podwójne kliknięcie na interesujący nas wzorzec. Wykonanie tej czynności przenosi nas do okna edycyjnego wzorca znaku, które pokazano na **rysunku 3**. Okno pozwala na dowolną edycję wcześniej wygenerowanego wzorca znaku. Mamy tutaj możliwość przesuwania wzorca znaku góra/dół i prawo/lewo, dodawania/usuwania dodatkowych wierszy i kolumn, inwersji wzorca znaku, czyszczenia oraz przesuwania linii bazowej, w odniesieniu do której wykonywane są przekształcenia tegoż wzorca. Co oczywiste, każde kliknięcie w samo pole wzorca znaku powoduje zapalenie/zgaszenie wybranego piksela dając dostęp do zupełnie swobodnej edycji. Należy zaznaczyć, iż dodanie nowego wiersza lub kolumny do wybranego wzorca znaku zostanie automatycznie anulowane w przypadku, gdy na nowo dodanym obszarze nie umieścimy aktywnego piksela (zapalonego), w związku z czym, jeśli zależy nam na zwiększeniu wysokości lub szerokości wzorca znaku (pamiętajmy, że zmiana wysokości odnosi się do całego zestawu znaków) bez zmiany jego treści musimy użyć grupy ikon pozwalających na dodanie dodatkowych marginesów. Tak utworzony zestaw znaków możemy następnie zapisać dla potrzeb późniejszego wykorzystania czy też edycji w specjalnym formacie programu PixeLab i, co najważniejsze, wygenerować interesujące nas pliki zawierające niezbędne struktury i tablice danych języka C (pliki *.h i *.c). W ten prosty, szybki i bardzo wygodny sposób możemy wygenerować wzorce znaków dla większości



Rysunek 5. Wygląd okna ustawień aplikacji PixeLab w zakresie mechanizmu generowania wzorców znaków

typów ciekłokrystalicznych wyświetlaczy LCD i to niezależnie czy do czynienia mamy z elementem monochromatycznym czy kolorowym typu TFT! Ostatnią możliwością, jaką daje nam program PixeLab jest możliwość zmiany kilku ustawień samej aplikacji.

Na **rysunku 4** pokazano okno ustawień aplikacji w zakresie wyglądu pola reprezentującego wzorzec znaku, zaś na **rysunku 5**. pokazano możliwości ustawień w zakresie mechanizmu generowania wzorców znaków. Mamy tu możliwość wyboru rodzaju zestawu znaków (w zasadzie rodzaju kodowania znaków) oraz zdecydowania o sposobie interpretacji wzorca znaku w procesie generowania stosownej tablicy, czyli sposobu analizowania „treści" obrazu poszczególnych znaków (horyzontalnie lub wertykalnie). Jak widać, aplikacja PixeLab, mimo że niepozorna, stanowić może potężne narzędzie w procesie projektowania każdego graficznego interfejsu użytkownika dla systemów z wbudowanym wyświetlaczem graficznym w sposób znaczący przyspieszając ten mozolny, zazwyczaj, proces. Niewątpliwą zaletą programu jest fakt, iż ten program jest w **darmowy** i nie ma żadnych mechanizmów ograniczających jego funkcjonalność. To niewątpliwy ukłon jego autora w kierunku całej rzeszy elektroników-programistów, z czego można brać tylko przykład. Biorąc to wszystko pod uwagę, tym bardziej polecić mogę tę aplikację jako najlepszą, której miałem okazji używać.

Robert Wołgajew, EP