Obsługa wyświetlaczy kolorowych

Wyświetlanie bitmap

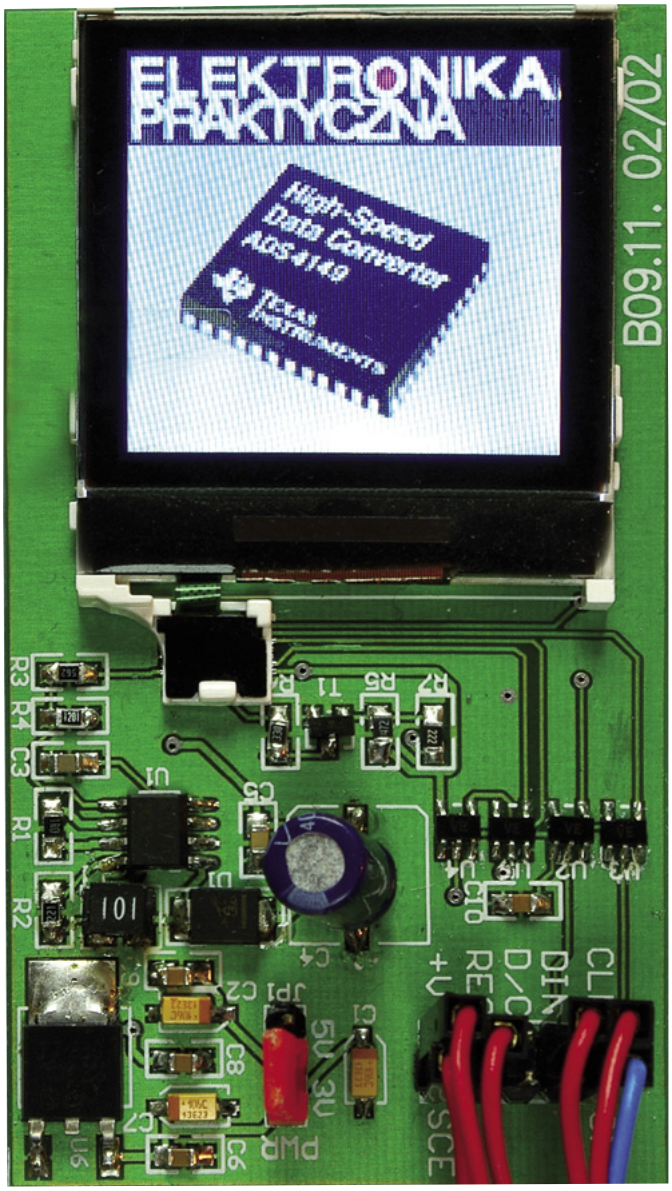
Wyświetlacze monochromatyczne są często używane w różnych konstrukcjach amatorskich. Inaczej sytuacja przedstawia się w przypadku wyświetlaczy kolorowych, nawet mimo tego, że kolor pozwala na znacznie lepsze pokazanie informacji. Ten artykuł ma na celu ułatwienie zastosowania popularnego wyświetlacza kolorowego we własnym urządzeniu. Opisuje sposoby sterowania wyświetlaczem oraz wyświetlania plików BMP.

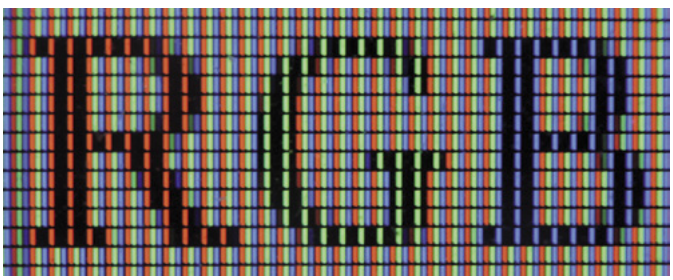
Ceny monochromatycznych wyświetlaczy graficznych są już tak niskie, że ich użycie we własnej aplikacji jest w zasięgu nawet niezbyt zasobnego hobbysty. Najbardziej popularne są wyświetlacze monochromatyczne bez możliwości sterowania jasnością (luminancją) piksela. Piksel jest albo „zapalony", albo „zgaszony". W pamięci programu sterownika takiego wyświetlacza jednemu pikselowi odpowiada jeden bit pamięci. Ponieważ dane są najczęściej zapisywane jako słowa 8 bitowe, to przyjęło się, że jeden bajt odpowiada wyświetlaniu linijki o długości 8 pikseli. O tym czy te linijki są ułożone pionowo, czy poziomo decydują właściwości układu scalonego sterownika wyświetlacza. Najczęściej po wpisaniu jednego bajtu jest wyświetlana linijka pionowa.

Istnieją też wyświetlacze monochromatyczne z możliwością modulacji jasności świecenia piksela. Każdy piksel może przybierać określoną liczbę stopni „szarości". Wtedy na piksel w pamięci obrazu przypada więcej niż jeden bit. Na przykład kiedy pikselowi są przyporządkowane 4 bity, to piksel może świecić z 16 poziomami jasności.

Organizacja pamięci obrazu

Do niedawna wyświetlacze kolorowe były bardzo drogie i trudno było je spotkać w niskobudżetowych interfejsach użytkownika. Również dzisiaj klasyczne wyświetlacze o relatywnie dużych wymiarach (porównywalnych z wyświetlaczami monochromatycznymi) nie są tanie, jednak w można użyć tanich wyświetlaczy z telefonów komórkowych. Jednym z parametrów wyświetlaczy (i ich sterowników) jest głębia koloru, czyli liczba możliwych do uzyskania kolorów pojedynczego piksela. Jest to parametr znany większości użytkowników komputerów. Jak łatwo domyśleć się, im więcej kolorów może odtworzyć wyświetlacz, tym wyświetlane obrazy będą bardziej zbliżone do rzeczywistości. W technice komputerowej standardem jest głębia 32-bitowa. To znaczy, że w pamięci komputera (lub karty graficznej) na atrybut koloru przeznaczono 32 bity (4 bajty) i każdy z pikseli może przybierać jeden z 2^32=4294967296 kolorów. Jeżeli komputer PC ma ustawioną rozdzielczość 2280×900 pikseli, to potrzebna wielkość pamięci do zapamiętania pojedynczego, statycznego obrazu wynosi 2280×900×4=10368000 bajtów. Całe szczęście, że w interesujących nas wyświetlaczach nie ma takich wymagań odnośnie do wielkości pamięci, bo raczej trudno byłoby go

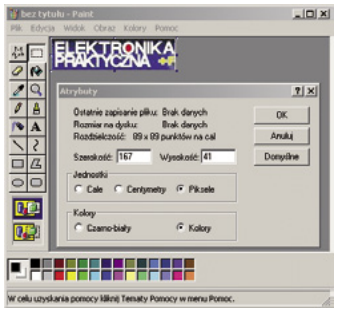




Rysunek 1. Świecące piksele RGB kolorowego wyświetlacza LCD

Rysunek 2. Położenie składowych 12-bitowego koloru w pamięci RAM obrazu dla 2 pikseli

stosować z taką rozdzielczością i głębią kolorów. Możliwości kolorowych wyświetlaczy telefonów komórkowych są zdecydowanie mniejsze. Pokażemy to na przykładzie wyświetlacza do telefonu komórkowego Nokia 6110.

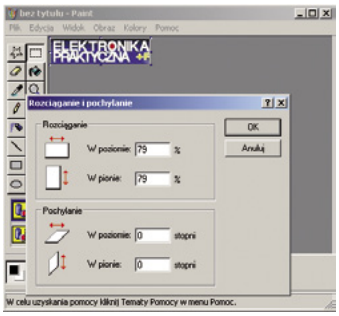


Rysunek 3. Atrybuty obrazka wklejonego z *MWSnap*

Wyświetlacz z Nokii 6110 ma rozdzielczość 132×132 piksele i 12-bitową głębię kolorów, czyli każdemu pikselowi w pamięci obrazu jest przypisanych 12 bitów.

Piksel wyświetlacza kolorowego jest zbudowany z 3 punktów. Każdy z tych punktów świeci w jednym z 3 podstawowych kolorów: czerwonym (R), zielonym (G) i niebieskim (B).

Te punkty maja identyczną budowę warstwy ciekłego kryształu, ale przed każdym z nich jest umieszczony inny filtr kolorowy. Powiększenie wyświetlacza w widoku od czoła pokazano na **rysunku 1**. Jasność świecenia każdego z punktów jest regulowana, a przez ta regulację otrzymujemy wypadkowy kolor poprzez składanie się z 3 podstawowych kolorów i sumaryczną jasność piksela.



Rysunek 4. Skalowanie obrazka

Te ogólne informacje są niezbędne żeby zrozumieć czego potrzebuje sterownik żeby wyświetlić żądany kolor piksela. A potrzebuje informacji z jaką jasnością ma świecić każda ze składowych podstawowych RGB.

Dla 12-bitowej głębi kolorów na każdą składową przypada po 4 bity, więc ich jasność może się zmieniać w 16 krokach. Oprócz rozdzielczości bitowej musimy znać położenie składowych we wpisywanym do sterownika słowie. W sterowniku firmy Epson wyświetlacza Nokii 6110 ułożenie bitów składowych 12-bitowego koloru jest takie, jak to pokazano na **rysunku 2**.

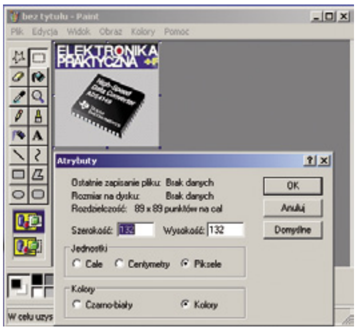
Aby optymalnie wykorzystać pamięć obrazu sterownika, składowe koloru dla 2 kolejnych pikseli upakowano w 3 kolejnych bajtach. Do zapełnienia całego ekranu wyświetlacza będziemy potrzebować (132×132×3)/2=25136 bajtów.

Generowanie bitmap

Już wiemy czego potrzebuje sterownik wyświetlacza. Zewnętrzny sterownik (host) musi odczytywać dane zapisane w pamięci wewnętrznej lub na przykład z karty SD i przesyłać je do sterownika wyświetlacza. W tym celu musimy przygotować tablicę z danymi,

w której będą zapisane składowe koloru każdego piksela. Kiedy po raz pierwszy używamy kolorowego wyświetlacza graficznego stajemy przed problemem, w jaki sposób wygenerować taką tablicę. To zadanie jest dobrze podzielić na etapy.

Po pierwsze, trzeba będzie utworzyć kolorową bitmapę o rozdzielczości 132×132 piksele. Postaram się pokazać ten proces na przykładzie. Niezbędne będzie użycie oprogramowania graficznego: programu do zapisywania zrzutów z ekranu i do obróbki i zapisywania bitmap. Ponieważ będziemy pracować z małymi bitmapami, to do obróbki plików zupełnie wystarczający będzie np. *Paint Brush* z Windowsa, a do zapisywania zrzutów ekranowych bezpłatny, polski program *WMSnap* autorstwa Mirka Wojtowicza (www.mirekw.com). *WMSnap* ma wbudowaną użyteczną funkcję zrzucania dowolnego prostokąta i zapisywania w schowku systemowym Windows. Za jego pomocą ze strony *www.ep.com.pl* skopiowałem fragment okładki jednego z numerów EP, a następnie fragment banera reklamowego. Oba te obrazy zapisujemy, lub bezpośrednio wklejamy używając schowka systemowego do tworzonej bitmapy.



Rysunek 5. Gotowa bitmapa o wymiarach 132×132 pikseli

Fragment okładki z napisem Elektronika praktyczna ma wymiary 167×41 pikseli i nie zmieści się w całości na ekranie o rozdzielczości 132×132 piksele. Przed wykorzystaniem należy zmienić jego wymiary, tak aby miał szerokość 132 pikseli. Można to zrobić programem Paint. Na **rysunku 3** pokazano atrybuty obrazka przed skalowaniem. Narzędziem *Rozciąganie i pochylanie* skalujemy obrazek. Przeskalowany obrazek zachowuje proporcje i ma szerokość 132 pikseli (**rysunek 4**).

Na końcu pracy z obrazkiem można go obrócić o kąt 90 lub 180°. W praktyce często dużo łatwiej jest obrócić obrazek w programie graficznym, niż potem modyfikować jego orientację za pomocą komend sterownika.

Następnie zmieniamy atrybuty na 132×132 piksele i wklejamy drugi obrazek, tak jak pokazano na **rysunku 5**. Wynik zapisujemy na dysku, ponieważ w kolejnym etapie trzeba go przekonwertować na tablicę, którą będzie mogła odczytać procedura sterownika – hosta.

W czasie pracy z wyświetlaczami monochromatycznymi korzystałem z gotowego programu napisanego przez Jerzego Szczesiula. W tym przypadku takiego programu nie miałem, ale po przeszukaniu Internetu natrafiłem na program *bmp2c* autorstwa Sebestien'a Riou. Program jest dostępny pod adresem *http://sourceforge.net/projects/bmp2c/*. Po ściągnięciu i rozpakowaniu otrzymujemy program z przykładami, dokumentacja i źródłami dla *Borland Builera C++*. Jak się okazało w praktyce, ten program jest zupełnie wystarczający dla naszych celów mimo bardzo uproszczonego interfejsu użytkownika.

Bmp2c wymaga podania: nazwy pliku z rozszerzeniem .ini z zapisanymi pleceniami do konwersji, nazwy pliku z bitmapą i opcjonalnej nazwy pliku tekstowego z wygenerowaną tablicą w języku C. Jeżeli nie podamy tej nazwy, to program nazwie plik tak samo, jak plik bitmapy, ale będzie on miał rozszerzenie .c i format tekstowy.

Sposób konwertowania danych jest zapisany w pliku z rozszerzeniem ini. Plik otwieramy w dowolnym edytorze tekstowym. Na pierwszy rzut wydaje się, że konfigurowanie programu jest trudne, ale to tylko pierwsze wrażenie. Nas będą interesowały 2 parametry: *data_size* i *data_map*. Pierwszy określa ile bitów ma głębia kolorów. Dla kolorów 12-bitowych wpisujemy *data_size=12*. Drugi parametr określa położenie bitów składowych kolorów w wygenerowanych słowach tablicy wyjściowej. W naszym przypadku zapiszemy:

data_map=r7 r6 r5 r4 g7 g6 g5 g4 b7 b6 b5 b4

Jak widać, zapisujemy najstarsze wagi bitów. Młodsze są zarezerwowane domyślnie dla kolorów o większej liczbie bitów.

Sterowanie wyświetlaczem

Do tej pory o tym nie wspomniałem, ale sterownik Epson wyświetlacza ma jeszcze jeden tryb koloru: kolor 8-bitowy. Mimo, że piszę o tym dopiero teraz uważam, że ten tryb może być podstawowym dla wyświetlaczy stosowanych w interfejsach użytkownika sterowników mikroprocesorowych. Mamy w nim do dyspozycji 256 kolorów i jeżeli nie chcemy wyświetlać wielobarwnych bitmap (na przykład zdjęć), to jest to w zupełności wystarczające. W porównaniu z trybem 12-bitowym ten tryb ma pewną zaletę: bitmapy zajmują w pamięci hosta nie 12 bitów na piksel, ale 8 bitów na piksel. 8 bitowa organizacja uprasza też procedury wyświetlania. Oczywiście dotyczy to tylko hosta, bo słowa 8-bitowe po wpisaniu do sterownika są i tak przez niego konwertowane na 12-bitowe.

Listing 1. Fragment tablicy wygenerowanej przez program *bmp2c*

```
#define sx 132;//width of the picture
#define sy 132;//height of the picture
const unsigned char bmp[] = {0x20, 0x00, 0x25, 0x25, 0x4a, 0x6e,
0x25, 0x25, 0x4a, 0x4a, 0x2a, 0x6e, 0x4a, 0x4e, 0x4a, 0x2a, 0x4a, 0x73, 0x4e, 0x4e,
0x4a, 0x2a, 0x2a, 0x4e, 0x2a, 0x4a, 0x4e, 0x26, 0x4e, 0x4e, 0x26, 0xdb, 0xdb, 0xdb,
0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb,
0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb,
0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb,
0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb,
0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb,
0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb,
0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb, 0xdb,
0x20, 0x01, 0x01, 0x01, 0x01, 0x25, 0x01, 0x05, 0x25, 0x25, 0x05, 0x25, 0x25, 0x26,
0x26, 0x26, 0x25, 0x2a, 0x2a, 0x26, 0x25, 0x2a, 0x26, 0x2a, 0x05, 0x26, 0x25, 0x25,
```

Listing 2. Wyświetlanie bitmapy w trybie 8-bitowym

```
void LCDWriteBmp(void) {
long j;

WriteSpiCommand(DATCTL);
WriteSpiData(7); // P1: 0x00 = adres strony dekr., adres
kolumn dekr.  ,mod. licznika stron
WriteSpiData(0x00); // P2: 0x00 = RGB (wartosc domyslna)
WriteSpiData(0x01); // P3:tryb 8 bitowy

WriteSpiCommand(CASET); //zakres licznika kolumn
WriteSpiData(0);
WriteSpiData(131);

WriteSpiCommand(PASET);//zakres licznika stron
WriteSpiData(0);
WriteSpiData(131);

WriteSpiCommand(RAMWR);   //zapis do pamieci
for(j = 0; j < 17424; j++) {
WriteSpiData(bmp[j]);
}
// powrot do ustawien dla wysietlania tekstu
WriteSpiCommand(DATCTL);
WriteSpiData(0x01); // P1: 0x01 = adres strony
dekrtement, adres kolumn inkr ,mod. licznika stron
WriteSpiData(0x00); // P2: 0x00 = RGB
WriteSpiData(0x02); // P3: 0x02 = powrót do trybu 12
bitowego
WriteSpiCommand(DISON);
}
```

R	R	R	G	G	G	B	B

Rysunek 6. Położenie składowych koloru w bajcie w trybie 8-bitowym

Listing 3 wyświetlenie bitmapy o dowolnych wymiarach

```
void LCDWriteBmp12(unsigned char sx, unsigned char sy, unsigned char x, unsigned char y) {
unsigned short  j,k,w0,w1;

WriteSpiCommand(DATCTL);
WriteSpiData(7); // P1: 0x00 = adres strony dekr., adres kolumn dekr.  ,mod. licznika stron
WriteSpiData(0x00); // P2: 0x00 = RGB (wartosc domyslna)
WriteSpiData(0x02); // P3:tryb 12 bitowy

WriteSpiCommand(CASET); //zakres licznika kolumn
WriteSpiData(x);
WriteSpiData(x+sx-1);

WriteSpiCommand(PASET);//zakres licznika stron
WriteSpiData(y);
WriteSpiData(y+sy-1);
k=0;
WriteSpiCommand(RAMWR);

for(j=0;j<(sx*sy/2);j++){
  w0=bmp[k++]; //pobranie z tablicy składowych koloru 2 kolejnych pikseli
  w1=bmp[k++];
   WriteSpiData((w0 >> 4) & 0xFF);  //konwersja na tryb 12bitowy
   WriteSpiData(((w0 & 0xF) << 4) | ((w1 >> 8) & 0xF));
   WriteSpiData(w1 & 0xFF);

}
// powrot do ustawien dla wysietlania tekstu
WriteSpiCommand(DATCTL);
WriteSpiData(0x01); // P1: 0x01 = adres strony dekrtement, adres kolumn inkr ,mod. licznika stron
WriteSpiData(0x00); // P2: 0x00 = RGB
WriteSpiData(0x02); // P3: 0x02 = tryb 12 bitowy

}
```

Jak w przypadku kolorów 12-bitowych, musimy znać położenie bitów składowych kolorów podstawowych. Pokazane je na **rysunku 6**.

W pliku .ini programu bmp2c wpisujemy

data_size=8

data_map = r7 r6 r5 g7 g6 g5 b7 b6

W wygenerowanej tablicy deklarowany jest typ zmiennych *short*. Dla mikrokontrolerów 32-bitowych ten typ oznacza dane 16-bitowe i przez to tablica zajmie dwukrotnie więcej miejsca, niż wynika z prostego rachunku. Moje próby z wyświetlaczem były przeprowadzane z użyciem mikrokontrolera STM32 z rdzeniem ARM. Przy użyciu bezpłatnej ewaluacyjnej wersji pakietu *Keil uVision4*, po zdeklarowaniu tablicy jako *short* wielkość kodu wynikowego przekraczała dozwolony dla tej wersji rozmiar 32 kB. Dlatego dla trybu 8-bitowego trzeba zmienić deklarację typu z *short* na *unsigned char*.

Oscyloskopy przenośne

VPS10 695 zł

Oscyloskop panelowy przeznaczony do nadzoru, kontroli urządzeń, do pracowni szkolnych, pokazów, testowania czujników itp.
1 kanał, 2 MHz

HPS50 1295 zł

Stworzony i zaprojektowany przez elektroników – entuzjastów dla elektroników – entuzjastów!!!
Urządzenie łączy w sobie wygodę użytkowania z praktycznością i wielozadaniowością.
Urządzenie sprawdzi się przy pomiarach wszelkiego rodzaju urządzeń audio-video, zasilaczy, układów cyfrowych, czujników.
1 kanał 2 MHz

HPS10SE 595 zł

Przenośny oscyloskop o wymiarach i cenie dobrej klasy multimetru.
Połączenie wysokiej czułości z dużą ilością funkcji pomiarowych pozwala na użytkowanie go w serwisach elektronicznych, samochodowych i oczywiście przez hobbystów.
1 kanał, 10 MHz
sonda pomiarowa i walizka w komplecie

HPS40 995 zł

HPS40 nie jest zwykłym multimetrem z wyświetlaczem graficznym, lecz pełnowartościowym, przenośnym oscyloskopem.
Niewątpliwą zaletą jest podświetlany wyświetlacz LCD i pięc różnych wariantów prezentacji pomiarów.
Oscyloskop przeznaczony jest do pomiarów we wszelkiego rodzaju urządzeniach audio-video, zasilaczach, układach cyfrowych, czujnikach, diagnostyce samochodowej, itd.
1 kanał, 12 MHz

Ceny z dnia 13.12.2010, sprzed zmiany stawki podatku VAT

AVT Korporacja
ul. Leszczynowa 11, 03-197 Warszawa
tel. 22 257 84 50, faks 22 257 84 55
www.sklep.avt.pl

velleman

Fotografia 7. Wyświetlenie bitmapy w trybie 8-bitowym.

Fragment tablicy wygenerowanej przez *bmp2c* pokazano na **listingu 1**. Deklarację typu danych zmieniono ręcznie z *short* na *unsigned char*. Przed właściwą tablicą są automatycznie zdefiniowane wymiary w pikselach konwertowanej bitmapy. Te definicje pozwalają procedurom wyświetlającym zorientować się w rozmiarze bitmapy w przypadku.

Na **listingu 2** pokazano procedurę wyświetlania pełnowymiarowej bitmapy zapisanej w głębi 8-bitowej.

Efekt działania procedury z list. 2 przy wyświetlaniu bitmapy z rys. 2 pokazano na **fotografii 7**. Bitmapa jest wyświetlana za pomocą wyświetlacza od telefonu Nokia 6110 w trybie 8-bitowym.

Program *bmp2c* w wersji dystrybuowanej przez autora nie potrafi zapisywać tablicy tak, aby przy 12 bitach na piksel w 3 kolejnych bajtach były zapisane składowe koloru dla 2 kolejnych pikseli. Dla składowych koloru dłuższych niż 8 bitów, składowe pojedynczego piksela są zapisywane na 16 bitach. Parametry konwersji będą następujące:

data_size=12

data_map = r7 r6 r5 r4 g7 g6 g5 g4 b7 b6 b5 b4

Jeżeli chcemy użyć trybu 12-bitowego, to należy wygenerowaną bitmapę przekonwertować.

W czasie konwersji są odczytywane po 2 słowa z tablicy bitmapy odpowiadające dwóm kolejnym pikselom, a następnie układane w trzech kolejnych bajtach, tak aby bity składowych koloru były zgodne z trybem 12-bitowym pokazanym na rys. 2. Na **listingu 3** zamieszczono procedurę, która realizuje to zadanie. Argumenty *sx* i *sy* określają wymiary bitmapy. Potrzebne wartości definiuje się w pliku wyjściowym program *bmp2c*. Argumenty *x* i *y* to współrzędne początku wyświetlania bitmapy. Dla pełnowymiarowych obrazów 132×132 piksele x=y=0. Komendy *CASET* i *PASET* na podstawie argumentów określają początek wyświetlania i wymiary bitmapy na ekranie LCD.

Pętla konwersji jest wykonywana (sx×sy/2)-razy, bo za każdym razem z tablicy są pobierane 2 słowa składowych. Następnie są one konwertowane na 3 bajty dla trybu 12-bitowego.

Przedstawione tutaj sposoby przygotowania tablic w C mogą być używane dla dowolnych kolorowych wyświetlaczy graficznych z różną głębią kolorów.

Tomasz Jabłoński
tomasz.jablonski@ep.com.pl