Okrągły wyświetlacz OLED (1)

Autor składa podziękowania panu Sławomirowi Szweda z firmy Unisystem za dostarczenie wyświetlacza opisywanego w artykule. Więcej informacji nt. wyświetlacza oraz karta katalogowa są dostępne na stronie ***www.unisystem.pl.***

Okrągły wyświetlacz nie jest podzespołem często stosowanym przez elektroników, chociaż jest idealny w niektórych zastosowaniach. Pierwszym, naturalnym wydaje się budowanie zegarów elektronicznych, w tym zegarków naręcznych typu smartwatch. W artykule opisano sposób aplikacji takiego wyświetlacza – część pierwsza zawiera omówienie technik programowania sterownika wyświetlacza w języku C, natomiast druga opis zestawu ewaluacyjnego.

W trakcie swojej długoletniej przygody z elektroniką wielokrotnie spotykałem się z koniecznością zastosowania wyświetlacza będącego głównym elementem graficznego interfejsu użytkownika. Nie powinno to dziwić, bo przecież najprostszy nawet wyświetlacz znacznie poszerza możliwości w zakresie interakcji urządzenia z użytkownikiem, dając przy okazji spore pole do popisu programiście czy grafikowi. Muszę przyznać, że najciekawsze z mojego punktu widzenia były zawsze graficzne wyświetlacze OLED, ponieważ, po pierwsze, pozwalają popuścić wodze fantazji w kwestii projektowania interfejsu użytkownika, a po drugie, dzięki niewiarygodnej wręcz jakości obrazu, czynią każde urządzenie niesamowicie atrakcyjnym wizualnie. Niestety w przypadku tych podzespołów pewien problem zawsze stanowiła cena, gdyż nie należą one do rozwiązań budżetowych. Na szczęście sytuacja ta uległa „dobrej zmianie" i niewielki graficzny wyświetlacz OLED możemy zakupić już za kilkanaście złotych, czego dobrym przykładem jest jeden z moich ostatnich projektów „Watch. Zegarek naręczny" opublikowany na łamach „Elektroniki Praktycznej" 12/2015 i 01/2016. Zastosowany w nim niewielki wyświetlacz OLED o rozdzielczości 128×64 piksele może stanowić niezły punkt startowy dla wielu urządzeń przenośnych.

Nie byłbym jednak sobą, gdybym spoczął na laurach i zadowolił się tanim, chińskim modułem, który, z uwagi na swoją grubość, nie będzie optymalnym wyborem w przypadku każdego urządzenia tego typu. Tak się ostatnio złożyło, iż dość dobrze znana na rynku tego rodzaju produktów tajwańska firma Winstar wprowadziła do sprzedaży okrągłe, niewielkie wyświetlacze OLED z kontrolerem na szkle (COG), które idealnie wpisują się w dzisiejszą modę urządzeń z rodziny „wearable" czy IoT, a których to cena zachęca do zastosowań w konstrukcjach amatorskich. Mowa o serii wyświetlaczy **WEO128128B**, które charakteryzują się następującymi, ważniejszymi parametrami:

- Rozdzielczość 128×128 pikseli.
- Możliwość wyświetlania 16 poziomów szarości, co poprawia estetykę wyświetlanych piktogramów.
- Średnica obszaru aktywnego 30 mm.
- Wymiary modułu 37 mm×41 mm×2 mm (tylko 2 mm grubości!).
- Wbudowany kontroler typu SSD1327ZB.
- Zasilanie 3 V.
- Szeroki wybór dostępnych interfejsów sterujących: równoległy 6800/8080, szeregowe I²C oraz SPI.
- Mały pobór mocy.
- Dostępne kolory świecenia: żółty, biały, niebieski.
- Długi czas bezawaryjnej pracy (rzędu 50 tys. godzin)
- Jasność 100 cd/m².
- Kontrast 2000:1.
- Szeroki kąt patrzenia w pionie i poziomie rzędu 160°.

Wygląd wyświetlacza OLED z rodziny WEO128128B pokazano na **fotografii 1**.

Prawda, że wyświetlacz robi wrażenie? Idealnie nadaje się do budowy wszelkiego rodzaju zegarków naręcznych, pulsometrów, urządzeń wspomagających aktywny tryb życia czy wykorzystujących technologię GPS.

Fotografia 1. Wygląd wyświetlacza OLED z rodziny WEO128128B

Opisując te arcyciekawe peryferia mógłbym, wzorem innych moich publikacji z tego zakresu, ograniczyć się do szczegółowego opisu procedur sterujących niezbędnych do ich obsługi, jednak tym razem będzie inaczej. Postanowiłem opracować ciekawy moduł ewaluacyjny, za którego pomocą zainteresowani czytelnicy będą mogli przetestować prezentowane oprogramowanie, jak też, bez wielkich przeróbek, wykonać praktyczne urządzenie przenośne. Moduł będzie opisany w drugiej części artykułu.

Niezależnie od wszystkiego, tak, czy inaczej, zacząć musimy od bohatera naszego artykułu – modułu wyświetlacza OLED typu WEO128128BLPP3N00000. Jest to wyświetlacz o białym kolorze pikseli, wyposażony w giętką, 24-wyprowadzeniową taśmę FPC przeznaczoną do wsunięcia w gniazdo ZIF o rastrze wyprowadzeń 0,5 mm. Rozkład i opis wyprowadzeń umieszczono w **tabeli 1**. Zgodnie z tym, co napisano i o czym można się przekonać spoglądając na tab. 1, nowe wyświetlacze z rodziny WEO128128B mają kilka rodzajów interfejsów sterujących, jednak z uwagi na fakt, iż w naszym docelowym systemie mikroprocesorowym nie dysponujemy bardzo szybkim mikrokontrolerem posłużymy się równoległym, 8-bitowym interfejsem danych/rozkazów sterujących, dla którego piny BS[2:1] wyświetlacza powinny zostać połączone z napięciem zasilania. Tego typu rozwiązanie pozwoli na osiągnięcie maksymalnej szybkości transmisji danych, co przełoży się na szybkość rysowania grafik ekranowych. Co więcej, zastosujemy jeszcze dwie inne „sztuczki" sprzętowe, które w dalszym stopniu przyczynia się do zwiększenia wspomnianej prędkości transmisji.

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 38323, pass: 5ednzyrt

Tabela 1. Rozmieszczenie wyprowadzeń taśmy ZIF wyświetlacza WEO128128BLPP3N00000

Pin	Nazwa	Znaczenie
1	**VSS**	Masa zasilania
2	**VCC**	Napięcie zasilania OLED (typ. 14,5 V/25 mA)
3	**VCOMH**	Wejście napięcia odniesienia VCOMH. Napięcie to może zostać dołączone z zewnątrz lub może też być dostarczone przez sterownik SSD1327. W drugim wypadku pomiędzy wejście VCOMH a masę (VSS) należy dołączyć kondensator.
4	**VCI**	Napięcie zasilania kontrolera i magistrali danych (typ. 3 V)
5	**VDD**	Napięcie zasilania rdzenia (2,4...2,6V) dostarczone z zewnątrz lub generowane (z VCI) przez wewnętrzny regulator sterownika ekranu (ustawienie programowalne). Niezbędny kondensator włączony pomiędzy to wyprowadzenie a masę zasilania.
6	**BS1**	Wybór aktywnej magistrali sterującej: BS[2:1] = 00 → SPI, BS[2:1] = 01 → I2C, BS[2:1] = 11 → równoległa 8080, BS[2:1] = 10 → równoległa 6800
7	**BS2**	
8	**VSS**	Masa zasilania
9	**IREF**	Wejście referencyjne dla prądu ISEG sterownika kolumn. Dla prądu odniesienia IREF=10 μA, pomiędzy to wejście a masę należy dołączyć rezystor o wartości 1 MΩ.
10	**CS**	Sygnał aktywacji sterownika ekranu.
11	**RES**	Sygnał zerowania sterownika ekranu.
12	**DC**	Sygnał wyboru rodzaju danych sterownika ekranu (0: rozkaz, 1: dane obrazu). W razie wyboru interfejsu I²C sygnał ten ustala adres sprzętowy modułu wyświetlacza.
13	**WR**	Sygnał zapisu sterownika ekranu.
14	**RD**	Sygnał odczytu sterownika ekranu.
15	**D0**	Interfejs równoległy lub: • w wypadku interfejsu I²C pin D2 połączony z D1 stanowi sygnał SDA, zaś pin D0 sygnał SCL, • w wypadku interfejsu SPI pin D0 stanowi sygnał SCLK, zaś D1 sygnał SDIN.
16	**D1**	
17	**D2**	
18	**D3**	
19	**D4**	
20	**D5**	
21	**D6**	
22	**D7**	
23	**VCC**	Napięcie zasilania (typ. 14,5 V/25 mA)
24	**VSS**	Masa zasilania

Po pierwsze, sygnał wyboru sterownika ekranu CS podłączymy na stałe z masą zasilania aktywując sterownik SSD1327, przez co funkcje odpowiedzialne za transmisję danych nie będą musiały obsługiwać tego sygnału. Po drugie, przyjmiemy, że stanem spoczynkowym sygnału wyboru rodzaju danych DC będzie logiczna jedynka, co oznacza, że przesyłane dane są domyślnie danymi treści obrazu. To pozwoli na dalsze przyspieszenie stosownych funkcji graficznych. Nasza biblioteka nie będzie również obsługiwać sygnału odczytu magistrali sterującej RD, gdyż nie korzystamy w niej z możliwości odczytu danych ze sterownika ekranu. Sygnał ten będzie stale logiczną jedynką.

Listing 1. Funkcje odpowiedzialne za wysyłanie rozkazów lub danych obrazu

```
void writeCommand(uint8_t Command)
{
    RESET_DC;  //DC=0 wskazuje na rozkaz sterujący
    RESET_WR;
    DATA_PORT = Command;
    SET_WR;  //Zapis danych następuje na rosnącym zboczu sygnału WR
    SET_DC;  //DC domyślnie ustawiine, by przyspieszyć transfer danych obrazu
}

void writeData(uint8_t Data)
{
    RESET_WR;
    DATA_PORT = Data;
    SET_WR;  //Zapis danych następuje na rosnącym zboczu sygnału WR
}
```

No dobrze, mamy już garść podstawowych informacji dotyczących naszego wyświetlacza, więc pora zacząć „zabawę" w programowanie! Aby jednak zacząć współpracę z tym arcyciekawym peryferium niezbędne są dwie podstawowe funkcje, które pozwolą na wysłanie do OLED rozkazów sterujących, dzięki którym możliwa jest konfiguracja sterownika SSD1327 lub też danych obrazu. Interpretacja rodzaju danych, które mają być odebrane przez sterownik ekranu jest determinowana poziomem na wejściu DC. Jest to typowe rozwiązanie w sterowaniu wyświetlaczami. Poziom niski na tym wyprowadzeniu decyduje o tym, iż przesyłana wartość zinterpretowana zostanie jako komenda sterująca, natomiast wysoki, iż dana zostanie zinterpretowana jako dana pamięci ekranu. Wspomniane, podstawowe funkcje zamieszczono na **listingu 1**.

Dysponując funkcjami z list. 1 możemy przystąpić do inicjalizacji naszego wyświetlacza, ponieważ wymaga on skonfigurowania szeregu rejestrów sterownika ekranu, których to wartości zależne są od właściwości kontrolowanego wyświetlacza OLED, jak i specyfikacji producenta modułu. Funkcję inicjalizacyjną pokazano na **listingu 2**. Brak przeprowadzenia inicjalizacji w zasadzie uniemożliwia poprawne funkcjonowanie modułu OLED, gdyż domyślne ustawienia rejestrów sterujących układu SSD1327 zwykle odbiegają od tych, wymaganych przez producenta wyświetlacza OLED. Aby jednak w pełni zrozumieć znaczenie poszczególnych ustawień sterownika ekranu, niezbędna jest znajomość zawartości stosownego pliku nagłówkowego, którego treść zamieszczono na **listingu 3**. Dalej, na **listingu 4** pokazano funkcję odpowiedzialną za ustawienie aktywnego obszaru ekranu, w którego ramach jest przeprowadzany zapis do pamięci ekranu sterownika SSD1327. Jej użycie upraszcza a zarazem

Listing 2. Funkcja inicjalizacyjna sterownika SSD1327 wyświetlacza WEO128128

```
void initOLED(void)
{
    //Port danych jako wyjściowy, równy 0x00
    DATA_DDR = 0xFF;
    //Port sterujący jako wyjściowy, równy 0xFF
    CTRL_PORT |= (1<<WR_PIN)|(1<<RD_PIN)|(1<<DC_PIN)|(1<<RST_PIN);
    CTRL_DDR |= (1<<WR_PIN)|(1<<RD_PIN)|(1<<DC_PIN)|(1<<RST_PIN);
    RESET_RST;  //Zerowanie sprzętowe sterownika SSD1327
    _delay_ms(10);
    SET_RST;
    _delay_ms(10);
    writeCommand(SET_COLUMN_ADDR);
    writeCommand(0x00);  //Start Column Address
    writeCommand((OLED_WIDTH/2)-1);  //End Column Address
    writeCommand(SET_ROW_ADDR);
    writeCommand(0x00);  //Start Row Address
    writeCommand(OLED_HEIGHT-1);  //End Row Address
    writeCommand(SET_CONTRAST_CTRL);
    writeCommand(0x9B);  //Contrast Level
    writeCommand(SET_REMAP);
    writeCommand(COLUMN_ADDR_REMAP|COM_REMAP|COM_SPLIT_ODD_EVEN);
    writeCommand(SET_DISP_START_LINE);  //Set Display Start Line
    writeCommand(0x00);
    writeCommand(SET_DISP_OFFSET);
    writeCommand(0x00);
    writeCommand(SET_DISP_MODE_NORMAL);
    writeCommand(SET_MULTIPLEX_RATIO);
    writeCommand(0x7F);       //Multiplex = 128
    writeCommand(SET_FUNC_SELECTION_A);
    writeCommand(ENABLE_INTERNAL_VDD);
    writeCommand(SET_PHASE_LENGTH);
    writeCommand(0xF1);  //705
    writeCommand(SET_DISP_CLOCK);  //Set Display Clock Divide Ratio/Oscilla-
tor Frequency
    writeCommand(0x00);
    writeCommand(SET_PRECHARGE_VOLTAGE);  //Set Prechange Voltage
    writeCommand(0x07);  //0.613 x VCC
    writeCommand(SET_VCOMH_VOLTAGE);  //Set VCOMH Voltage
    writeCommand(0x07);  //0.86 x VCC
    writeCommand(SET_SEC_PRECHARGE_PERIOD);  //Set Second Pre-charge period
    writeCommand(0x0F);
    writeCommand(SET_FUNC_SELECTION_B);
    writeCommand(INTERNAL_VSL|ENABLE_SECOND_PRECHARGE);
    writeCommand(SET_GRAY_SCALE_TABLE);
    writeCommand(0x01);  //A[2:0]
    writeCommand(0x01);  //B[2:0]
    writeCommand(0x10);  //B[6:4]
    writeCommand(0x01);  //C[2:0]
    writeCommand(0x10);  //C[6:4]
    writeCommand(0x01);  //D[2:0]
    writeCommand(0x10);  //D[6:4]
    writeCommand(0x01);  //E[2:0]
    writeCommand(0x10);  //E[6:4]
    writeCommand(0x01);  //F[2:0]
    writeCommand(0x10);  //F[6:4]
    writeCommand(0x01);  //G[2:0]
    writeCommand(0x10);  //G[6:4]
    writeCommand(0x01);  //H[2:0]
    writeCommand(0x10);  //H[6:4]
    writeCommand(SET_DISPLAY_ON);  //Set Display On
}
```

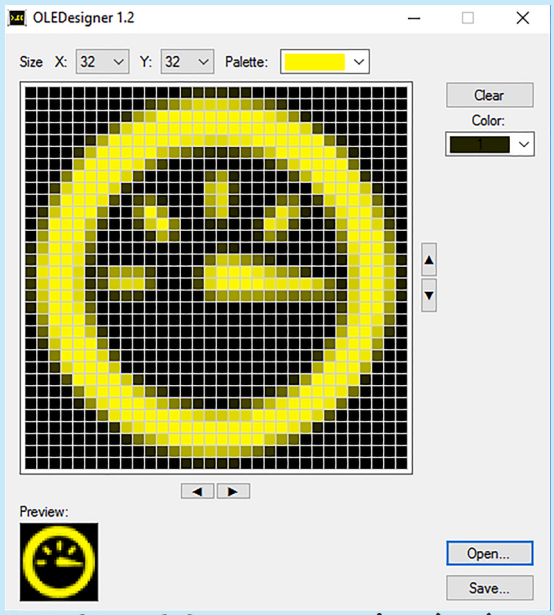
Listing 3. Plik nagłówkowy sterownika SSD1327 wyświetlacza WEO128128

```
#define DATA_PORT PORTB
#define DATA_DDR DDRB
#define CTRL_PORT PORTD
#define CTRL_DDR DDRD
#define WR_PIN PD6
#define RD_PIN PD7
#define DC_PIN PD5
#define RST_PIN PD4
#define RESET_DC CTRL_PORT &= ~(1<<DC_PIN)
#define SET_DC CTRL_PORT |= (1<<DC_PIN)
#define RESET_WR CTRL_PORT &= ~(1<<WR_PIN)
#define SET_WR CTRL_PORT |= (1<<WR_PIN)
#define RESET_RST CTRL_PORT &= ~(1<<RST_PIN)
#define SET_RST CTRL_PORT |= (1<<RST_PIN)
//Rozdzielczość ekranu (pozioma i pionowa)
#define OLED_WIDTH 128
#define OLED_HEIGHT 128
//Definicje komend sterujących
#define SET_COLUMN_ADDR 0x15
#define SET_ROW_ADDR 0x75
#define SET_CONTRAST_CTRL 0x81
#define SET_REMAP 0xA0
    #define COLUMN_ADDR_REMAP (1<<0)
    #define NIBBLE_REMAP (1<<1)
    #define HORIZONTAL_ADDR_INCR (0<<2)
    #define VERTICAL_ADDR_INCR (1<<2)
    #define COM_REMAP (1<<4)
    #define COM_SPLIT_ODD_EVEN (1<<6)
#define SET_DISP_START_LINE 0xA1
#define SET_DISP_OFFSET 0xA2
#define SET_DISP_MODE_NORMAL 0xA4
#define SET_DISP_MODE_INVERSE 0xA7
#define SET_MULTIPLEX_RATIO 0xA8
#define SET_FUNC_SELECTION_A 0xAB
    #define SELECT_EXTERNAL_VDD (0<<0)
    #define ENABLE_INTERNAL_VDD (1<<0)
#define SET_DISPLAY_OFF 0xAE
#define SET_DISPLAY_ON 0xAF
#define SET_PHASE_LENGTH 0xB1
#define SET_DISP_CLOCK 0xB3
#define SET_SEC_PRECHARGE_PERIOD 0xB6
#define SET_GRAY_SCALE_TABLE 0xB8
#define SET_PRECHARGE_VOLTAGE 0xBC
#define SET_VCOMH_VOLTAGE 0xBE
#define SET_FUNC_SELECTION_B 0xD5
    #define INTERNAL_VSL (0x60|(0<<0))
    #define ENABLE_EXTERNAL_VSL (0x60|(1<<0))
    #define DISABLE_SECOND_PRECHARGE (0x60|(0<<1))
    #define ENABLE_SECOND_PRECHARGE (0x60|(1<<1))
```

znacznie przyspiesza wyświetlanie obrazów, których wielkość jest inna, aniżeli całkowita, fizyczna rozdzielczość ekranu. Co ważne, dla uproszczenia zapisu do wspomnianej pamięci obrazu, przyjmuje się, iż dopuszczalne wartości argumentów dla osi X **muszą być parzyste**, co wydaje się być niewielkim ograniczeniem przedstawionej powyżej funkcji jak i następnych. Wynika to z faktu, iż sterownik SSD1327 ma dość nietypową organizację pamięci obrazu, gdzie każdy bajt tejże pamięci przechowuje dane dla 2 kolejnych pikseli obrazu (stąd możliwe jest wyświetlenie 16 poziomów jasności każdego piksela). Czas na funkcje odpowiedzialne za rysowanie prostych elementów graficznych, to jest funkcje umożliwiające wyświetlanie wypełnionego i „pustego" prostokąta na ekranie wyświetlacza, których to treść pokazano na **listingu 5**.

Teraz z kolei, przedstawię funkcję pozwalającą wyświetlać proste grafiki ekranowe. Jako, że zgodnie z tym, co napisano powyżej, sterownik SSD1327 ma dość nietypową organizację pamięci ekranu, do przygotowania tablic zawierających przeznaczone do wyświetlenia grafiki niezbędne okazało się wykorzystanie specjalistycznego oprogramowania. W tym celu posłużono się najnowszą wersją programu **OLEDesigner** (v.1.2), którego autorem jest Marcin Popławski. Jest to, co oczywiste, jedno z możliwych rozwiązań tego problemu, lecz moim zdaniem na tyle uniwersalne, komfortowe i łatwe w użyciu, iż może być podstawowym narzędziem dla monochromatycznych wyświetlaczy OLED o takiej organizacji pamięci obrazu. Wygląd okna najnowszej wersji programu OLEDesigner przedstawiono na **rysunku 2**. Za pomocą tego łatwego w użyciu, ale jakże użytecznego narzędzia, jesteśmy w stanie zaprojektować dowolny element przyszłego interfejsu użytkownika projektowanego urządzenia. Co więcej, bieżąca wersja programu obsługuje następujące formaty danych:

- OPF (Oled Picture File) czyli plik binarny o odpowiedniej strukturze przeznaczony do zastosowań w kompilatorze Bascom.



Rysunek 2. Wygląd programu OLEDesigner (v.1.2)

Listing 4. Funkcja ustawiająca aktywney obszar ekranu
```
//X1 i X2: 0...127 (tylko parzyste), Y1 i Y2: 0...127
void setActiveWindow(uint8_t X1, uint8_t Y1, uint8_t X2, uint8_t Y2)
{
    writeCommand(SET_COLUMN_ADDR);  //Set Column Address
    writeCommand(X1>>1);  //Adres startowy aktywnego obszaru pamięci ekranu dla osi X
    writeCommand(X2>>1);  //Adres końcowy aktywnego obszaru pamięci ekranu dla osi X
    writeCommand(SET_ROW_ADDR);  //Set Row address
    writeCommand(Y1);  //Adres startowy aktywnego obszaru pamięci ekranu dla osi Y
    writeCommand(Y2);  //Adres końcowy aktywnego obszaru pamięci ekranu dla osi Y
}
```

Listing 5. Funkcje wyświetlające wypełniony i pusty prostokąt
```
void drawFilledBox(uint8_t X1, uint8_t Y1, uint8_t X2, uint8_t Y2, uint8_t Color)
{
    uint16_t Pixels = ((X2-X1+1)/2)*(Y2-Y1+1);
    setActiveWindow(X1, Y1, X2, Y2);
    while(Pixels--) writeData(Color);
}

void drawRectangle(uint8_t X1, uint8_t Y1, uint8_t X2, uint8_t Y2, uint8_t Color)
{
    drawFilledBox(X1 , Y1 , X2 , Y1, Color);
    drawFilledBox(X1 , Y2 , X2 , Y2, Color);
    drawFilledBox(X1 , Y1 , X1 , Y2, Color);
    drawFilledBox(X2 , Y1 , X2 , Y2, Color);
}
```

są ponumerowane. Pod prawym przyciskiem myszy kryje się gumka, za pomocą której usuwamy niepotrzebny kolor. Pliki OPF jak i C mają następującą strukturę danych:

- pierwszy bajt określa szerokość obrazka,
- drugi bajt określa wysokość obrazka,
- pozostałe bajty to dane kolejnych pikseli obrazu poddane prostemu algorytmowi kompresji, przy czym zgodnie z organizacją pamięci sterownika SSD1327, każdy wspomniany bajt przechowuje informację o dwóch, kolejnych punktach obrazu.

Jak pokazały testy praktyczne, najlepszy stopień kompresji obrazków przechowywanych w ten sposób daje zmodyfikowany algorytm RLE (nazwany tutaj mRLE), którego ideę działania przedstawia **tabela 2**. Pomimo swojej prostoty zapewnia on całkiem spory stopień kompresji, co zdecydowanie przekłada się na zmniejszenie zajętości pamięci Flash mikrokontrolera. Pamiętamy przecież, że stosowne wzorce przechowujemy zazwyczaj w pamięci Flash. Na **listingu 6** przedstawiono funkcję odpowiedzialną za odczytanie, dekompresję i wyświetlenie tak zapisanego obrazka.

Listing 6. Funkcja odpowiedzialna za odczytanie, dekompresję i wyświetlenie obrazka
```
void drawPicture(uint8_t X1, uint8_t Y1, const uint8_t *Picture)
{
    register uint8_t byteA, byteB;
    uint16_t bytesToSend, repeats, i=2;
    //Obliczamy liczbę bajtów jakie należy wysłać do sterownika SSD1327
    bytesToSend = (pgm_read_byte(&Picture[0])>>1)*pgm_read_byte(&Picture[1]);
    //Ustawiamy aktywne okno pamięci obrazu sterownika SSD1327 by uprościć zapis danych
    setActiveWindow(X1, Y1, X1+pgm_read_byte(&Picture[0])-1, Y1+pgm_read_byte(&Picture[1])-1);
    do
    {
        if(bytesToSend == 1)  //Dla ostatniego bajta danych
        {
            writeData(pgm_read_byte(&Picture[i]));
            bytesToSend--;
        }
        else
        {
            byteA = pgm_read_byte(&Picture[i]);  //Odczytujemy bajt „i”
            byteB = pgm_read_byte(&Picture[i+1]);  //Odczytujemy bajt „i+1”
            if( byteA != byteB)  //Sprawdzamy czy kolejne bajty są różne
            {
                writeData(byteA);
                bytesToSend--;
                i++;
            }
            else
            //Jeśli kolejne bajty są takie same to ustalamy liczbę powtórzeń danego bajta (+2 powtórzenia
            //obligatoryjne) wynikające ze sposobu zapisu kompresji typu mRLE (zmodyfikowane RLE)
            {
                repeats = pgm_read_byte(&Picture[i+2])+2;  //Odczytujemy liczbę powtórzeń danego bajta + 2 bajty obligatoryjne
                while(repeats--) {writeData(byteA); bytesToSend--;} //Wysyłamy liczbę powtórzeń bajta
                i +=3;  //Przesuwamy indeks o 3 miejsca
            }
        }
    } while(bytesToSend);
}
```

- BMP o dowolnej palecie barw, gdyż program automatycznie przekształci taki obrazek do trybu monochromatycznego (oraz „przytnie” do formatu 128×128, jeśli rozdzielczość obrazka będzie nieodpowiednia).
- C z tablicą wzorca obrazka zapisaną przy użyciu kompresji mRLE (zmodyfikowany algorytm RLE).
- C z tablicą wzorca obrazka zapisaną przy użyciu typowej kompresji RLE.

Otrzymany w ten sposób obraz możemy dowolnie edytować jak i zapisywać/odczytywać korzystając z dowolnego z dostępnych formatów danych. Sama edycja jest bardzo intuicyjna. Po wyborze odpowiedniego koloru z rozwijanej listy i wciśnięciu lewego przycisku myszy nanosimy wybrany kolor na obraz. W celu łatwiejszej identyfikacji kolory w palecie

W tej chwili przyszedł czas na obsługę ostatniego elementu interfejsów graficznych, czcionek ekranowych! Aby jednak umożliwić wygodną obsługę wielu czcionek ekranowych, konieczne było

Listing 7. Definicja typu danych przechowującego parametry czcionki
```
typedef struct  //Deklaracja struktury przechowującej parametry bieżącej czcionki
{
    uint8_t Width;  //Rzeczywista szerokość znaku (px)
    uint8_t Height;  //Rzeczywista wysokość znaku (px)
    uint8_t Interspace;  //Odstęp pomiędzy znakami (px)
    uint8_t BytesPerChar;  //Liczba bajtów danych tablicy wzorców na 1 znak
    uint8_t FirstCharCode;  //Kod ASCII pierwszego znaku
    const uint8_t *Bitmap;  //Wskaźnik to tablicy zawierającej wzorce poszczególnych znaków
} fontDescription;
```

Listing 8. Funkcja wybierająca bieżącą czcionkę ekranową
```
void setFont(const fontDescription *Font)
{
    CurrentFont.Width = pgm_read_byte(&Font->Width);  //Rzeczywista szerokość czcionki
    CurrentFont.Height = pgm_read_byte(&Font->Height);  //Rzeczywista wysokość czcionki
    CurrentFont.Interspace = pgm_read_byte(&Font->Interspace);  //Odstęp pomiędzy znakami
    CurrentFont.BytesPerChar = pgm_read_byte(&Font->BytesPerChar);  //Liczba bajtów na definicje pojedyn. znaku
    CurrentFont.FirstCharCode = pgm_read_byte(&Font->FirstCharCode);  //Kod ASCII definicji pierwszego znaku
    CurrentFont.Bitmap = (uint8_t*)pgm_read_word(&Font->Bitmap);  //Wskaźnik do tablicy wzorców tej czcionki
}
```

Listing 9. Funkcja odpowiedzialna za rysowanie znaków przy użyciu czcionki ekranowej

```
void drawText(uint8_t X1, uint8_t Y1, char *Text, uint8_t Color, uint8_t Background)
{
    register char Character;
    register uint16_t byteIndex, offset;
    register uint8_t corrWidth, bitIndex, readByte;
    //Korygujemy docelową szerokość w celu poprawnego ustawienia okna zapisu, które to zaokrąglamy do pełnych bajtów. Ma to uprościć
    //sprawdzenie poszczególnych bitów bajta i tak np. dla szerokości = 6, skorygowana szerokość będzie równa 8, dla 12 równa 16 itd.
    corrWidth = ((CurrentFont.Width/8) + (CurrentFont.Width%8 !=0 ? 1 : 0)) *8;
    while ((Character = *(Text++)))  //Kolejne znaki napisu wejściowego
    {
        setActiveWindow(X1, Y1, X1+corrWidth-1, Y1+CurrentFont.Height-1);
        offset = (Character-CurrentFont.FirstCharCode)*CurrentFont.BytesPerChar;  //Offset położenia wzorca w tablicy wzorców znaków
        for( byteIndex = offset; byteIndex < (offset + CurrentFont.BytesPerChar); byteIndex++)
        {
            //Odczytujemy kolejny bajt definicji znaku umieszczony w tablicy Bitmap
            readByte = pgm_read_byte(CurrentFont.Bitmap+byteIndex);
            //Kolejne pary bitów bajta: 0-1, 2-3, 4-5, 6-7 by utworzyć bajt wynikowy zawierający dane 2 pixeli w 16 odcieniach
            for( bitIndex=0; bitIndex<6; bitIndex+=2 )
            {
                if(readByte & (1<<bitIndex)) corrWidth= Color <<4; else corrWidth = Background <<4;  //corrWidth użyte jako temp
                if(readByte & (1<<(bitIndex+1))) corrWidth |= Color; else corrWidth |= Background;
                writeData(corrWidth );
            }
        }
        X1+=(CurrentFont.Width+CurrentFont.Interspace);  //Przesuwamy się o rzeczywistą szerokość znaku plus odstęp pomiędzy znakami
    }
}
```

Tabela 2. Zasada działania algorytmu kompresującego dane obrazków

Liczba powtórzeń	Dane przed kompresją	Dane po kompresji
1	0xAA	0xAA
2	0xAA 0xAA	0xAA 0xAA 0x00
3	0xAA 0xAA 0xAA	0xAA 0xAA 0x01
4	0xAA 0xAA 0xAA 0xAA	0xAA 0xAA 0x02
5	0xAA 0xAA 0xAA 0xAA 0xAA	0xAA 0xAA 0x03
257	257 x 0xAA	0xAA 0xAA 0xFF

wprowadzenie nowego typu danych, którego definicję pokazano na **listingu 7**. Bazując na zdefiniowanej powyżej strukturze, wprowadzono funkcję, która korzystając z globalnej zmiennej **static** *fontDescription* CurrentFont pozwala na ustawienie bieżącej czcionki ekranowej – pokazano ją na **listingu 8**. Wreszcie przyszedł czas na funkcję umożliwiającą rysowanie znaków przy użyciu bieżącej czcionki ekranowej – zamieszczono ją na **listingu 9.**

Jak zwykle, do wygenerowania plików zawierających wzorce czcionek, oparte na czcionkach systemu Windows, polecam doskonały program **PixeLab** autorstwa Marcina Popławskiego, którego szczegółowy opis znalazł się w Elektronice Praktycznej 06/2015.

Robert Wołgajew, EP

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **38323**, pass: **5ednzyrt**