Ramka cyfrowa do zdjęć

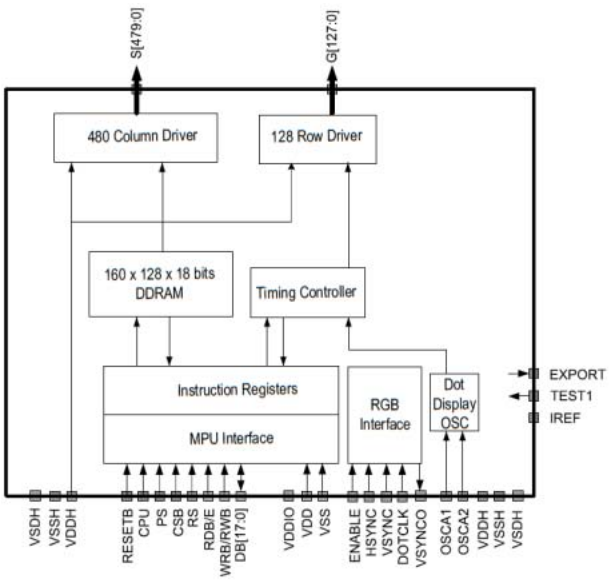
Obsługa kolorowego wyświetlacza OLED w języku C

Kiedy kilka lat temu pierwszy raz zajmowałem się tematyką obsługi paneli dotykowych, byłem niemal pewien, że tego rodzaju sposób interakcji urządzenia z użytkownikiem stanie się w przyszłości standardem dla elektroniki użytkowej i nie tylko. Nie czuję się bynajmniej, co oczywiste, żadnego rodzaju wizjonerem! Ot, taki scenariusz wydawał mi się najbardziej prawdopodobny biorąc pod uwagę rozwój interfejsów użytkownika opartych o wyświetlacze TFT oraz rosnące wymagania klientów. Inną sprawą jest, iż zastosowanie kolorowych wyświetlaczy TFT wyposażonych w panel dotykowy zapewnia naturalny sposób komunikacji z urządzeniem, którego możliwości ograniczone są wyłącznie wyobraźnią programisty czy grafika komputerowego.

Rekomendacje: *artykuł jest doskonałą bazą dla własnych projektów.*

Dzisiaj nie dziwi już fakt, że w wielu modelach samochodów, bynajmniej nie z segmentu premium, stosowane są rozbudowane systemy audiowizualne wyposażone w doskonałej jakości, duże wyświetlacze TFT oraz rozbudowane interfejsy użytkownika wzorowane na najlepszych rozwiązaniach dostępnych w urządzeniach do komunikacji mobilnej.

Co wspólnego ma ta krótka historia z naszym artykułem? To proste, pewną analogię, o której za chwilę. Od dosyć dawna w projektach swoich urządzeń stosuję wyświetlacze wykonane w technologii OLED i mimo tego, że proces ten rozpoczął się jakiś czas temu, jestem nadal pod ogromnym wrażeniem tych elementów elektronicznych. Nie sposób tutaj nie zauważyć wielkiego postęp, który dokonał się w tej technologii. Jego wynikiem jest znaczny wzrost trwałości tego rodzaju paneli oraz towarzyszący mu, ustawiczny spadek cen gotowych rozwiązań, których pułap sięga już typowych wyświetlaczy LCD. Te wszystkie wydarzenia skłaniają mnie do twierdzenia, iż przyszłość elektroniki w zakresie systemów wizualizacji oraz GUI, będzie należała do technologii OLED. Wystarczy skierować uwagę w stronę nowoczesnych, topowych modeli telefonów komórkowych, markowych przyrządów pomiarowych z tzw. wysokiej półki lub urządzeń medycznych, by



Rysunek 1. Uproszczony schemat funkcjonalny układu SEPS525 w zakresie sterownika matrycy OLED

W ofercie AVT*
AVT-5409 A

Podstawowe informacje:
- Napięcie zasilania: 4…6 V DC (lub 3 baterie AA).
- Maksymalny prąd obciążenia: 100 mA (zależnie od treści obrazu oraz ustawienia sterownika ekranu).

Dodatkowe materiały na CD lub FTP:
ftp://ep.com.pl, user: 62828, pass: 18ofqn10
- wzory płytek PCB
- karty katalogowe i noty aplikacyjne elementów oznaczonych w Wykazie elementów kolorem czerwonym

Projekty pokrewne na CD/FTP:
(wymienione artykuły są w całości dostępne na CD)

AVT-5246 Atframe – ramka do zdjęć cyfrowych EP 7/2010
AVT-5208 T-Mixer. Nowoczesny mikser audio z panelem dotykowym EP 11/2009

* Uwaga:
Zestawy AVT mogą występować w następujących wersjach:
AVT xxxx UK to zaprogramowany układ. Tylko i wyłącznie. Bez elementów dodatkowych.
AVT xxxx A płytka drukowana PCB (lub płytki drukowane, jeśli w opisie wyraźnie zaznaczono), bez elementów dodatkowych.
AVT xxxx A+ płytka drukowana i zaprogramowany układ (czyli połączenie wersji A i wersji UK) bez elementów dodatkowych.
AVT xxxx B płytka drukowana (lub płytki) oraz komplet elementów wymieniony w załączniku pdf
AVT xxxx C to nic innego jak zmontowany zestaw B, czyli elementy wlutowane w PCB. Należy mieć na uwadze, że o ile nie zaznaczono wyraźnie w opisie, zestaw ten nie ma obudowy ani elementów dodatkowych, które nie zostały wymienione w załączniku pdf
AVT xxxx CD oprogramowanie (nieczęsto spotykana wersja, lecz jeśli występuje, to niezbędne oprogramowanie można ściągnąć, klikając w link umieszczony w opisie kitu)
Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik pdf! Podczas składania zamówienia upewnij się, którą wersję zamawiasz! (UK, A, A+, B lub C). *http://sklep.avt.pl*

Tabela 1. Rozkład wyprowadzeń wyświetlacza MI160128AO z opisem ich funkcjonalności

Pin	Symbol	Opis
1	NC	Niewykorzystane
2	VSDH	Masa sterownika kolumn
3	VDDH	Napięcie zasilania sterownika kolumn i wierszy (14 V)
4	VSSH	Masa sterownika wierszy
5	IREF	Rezystor referencyjny źródła prądowego (regulacja jasności). Pomiędzy to wejście a masę należy dołączyć rezystor o wartości 68 kΩ.
6	OSCA2	Rezystor referencyjny wewnętrznego oscylatora. Jeśli wewnętrzny oscylator jest wykorzystywany pomiędzy OSCA2 a OSCA1 należy dołączyć rezystor o wartości 10 kΩ. W przeciwnym wypadku OSCA1 jest wejściem zewnętrznego sygnału zegarowego.
7	OSCA1	
8	VDDIO	Napięcie zasilania interfejsów I/O sterownika (1.6÷3.3 V)
9	VSYNCO	Wyjście sygnału synchronizacji pionowej (aktywny interfejs RGB)
10	VSYNC	Wejście sygnału synchronizacji pionowej (aktywny interfejs RGB)
11	HSYNC	Wejście sygnału synchronizacji poziomej (aktywny interfejs RGB)
12	DOTCLK	Wejście sygnału taktującego (aktywny interfejs RGB)
13	ENABLE	Wejście aktywacji dla sygnału video RGB
14	CPU	Wybór typu magistrali danych: 0: magistrala zgodna z interfejsem 80XX, 1: magistrala zgodna z interfejsem 68XX
15	PS	Wybór rodzaju magistrali danych: 0: szeregowa, 1: równoległa
16	D17	Dwukierunkowa, 9-bitowa magistrala danych. Znaczenie poszczególnych wyprowadzeń zależy od ustawienia wejścia PS: dla magistrali równoległej 8-bitowej wykorzystywane są wejścia D17…D10 a dla 9-bitowej D17…D9. Dla magistrali szeregowej SPI wybrane wejścia mają następującą funkcjonalność: D17->SCL, D16->SDI, D15->SDO. Wejścia niewykorzystane należy podłączyć do masy.
17	D16	
18	D15	
19	D14	
20	D13	
21	D12	
22	D11	
23	D10	
24	D9	
25	RS	Sygnał wyboru rodzaju danych: 0: rozkaz sterujący, 1: argument lub bajt pamięci obrazu DDRAM
26	CS	Sygnał wyboru układu (aktywacji chipsetu)
27	RD	Sygnał żądania operacji odczytu
28	WR	Sygnał żądania operacji zapisu
29	RES	Sygnał zerowania chipsetu
30	VSS	Masa logiki sterownika
31	VDD	Napięcie zasilania logiki sterownika (2.6÷3.3 V)
32	VSSH	Masa sterownika wierszy
33	VDDH	Napięcie zasilania sterownika kolumn i wierszy (14 V)
34	VSDH	Masa sterownika kolumn
35	NC	Niewykorzystane

nabrać pewności, iż postawiona wyżej teza graniczy z pewnością! Idąc, więc z duchem czasu wykorzystajmy w naszym projekcie kolorowy panel OLED, dla którego to dotychczasowe obszary zastosowań zarezerwowane były wyłącznie dla nowoczesnych smartfonów. Nie będzie to, co prawda, element o tak wysublimowanych parametrach, jakie można znaleźć we wspomnianych powyżej rozwiązaniach profesjonalnych, ale i cena tego peryferium jest, nazwijmy to, „amatorska", a więc dostępna dla przeciętnego elektronika.

Artykuł postanowiłem opracować w nieco inny sposób, niż miało to miejsce dotychczas. Tym razem przedstawię niezbędne minimum informacji dotyczących samego wyświetlacza skupiając się bardziej na aspektach programistycznych a na „deser" pokażę projekt ramki cyfrowej wyposażonej w tenże panel OLED (niespotykane w handlu rozwiązanie) oraz zintegrowaną obsługę kart typu microSD. Przejdźmy, zatem, do interesującego nas panelu będącego przedmiotem niniejszego artykułu.

Przeglądając rozwiązania dostępne w handlu dość szybko zorientowałem się, że lista ich nie będzie zbyt długa, zwłaszcza mając na uwadze kryteria cenowe. W ten prosty sposób, jedynym panelem, jaki mogłem z powodzeniem zastosować, okazał się kolorowy, pasywny wyświetlacz OLED produkcji Multi-Inno Technology Co., Ltd. o oznaczeniu MI160128AO charakteryzujący się następującymi parametrami:

- rozdzielczość ekranu 160×128 pikseli,
- możliwość wyświetlania 262 tys. kolorów (18 bitów) lub 65 tys. kolorów (16 bitów),
- rozmiar obszaru aktywnego ekranu 34 mm×28 mm,

- wbudowany, zaawansowany sterownik ekranu SEPS525 wyposażony w pamięć ekranu DDRAM o wielkości 160×18(RGB)×128=368640 bitów,
- 3 dostępne interfejsy sterujące: równoległy (68/80 w konfiguracji 8/9/16/18-bitów), SPI i RGB,
- niewielkie wymiary zewnętrzne 40 mm×34 mm×**1,7** mm (imponująco mała grubość!).

Listing 1. Listing funkcji zapisu rozkazu sterującego
```
static void WriteCommand(uint8_t Command)
{
  RESET_RS;
  RESET_WR;
  RESET_CS;
  DATA_PORT = Command;
  SET_CS;
  SET_WR;
  SET_RS;
}
```

Listing 2. Funkcja zapisu argumentu rozkazu sterującego lub danej pamięci ekranu DDRAM
```
static void WriteData(uint8_t Data)
{
  RESET_WR;
  RESET_CS;
  DATA_PORT = Data;
  SET_CS;
  SET_WR;
}
```

Listing 3. Listing funkcji zapisu rozkazu sterującego wraz z argumentem
```
static void WriteRegister(uint8_t Register, uint8_t Value)
{
  WriteCommand(Register);
  WriteData(Value);
}
```

List.4. Zawartość pliku nagłówkowego SEPS525drv.h
```
#define HORIZ_RESOLUTION 160  //Rozdzielczość pozioma ekranu
#define VERT_RESOLUTION 128  //Rozdzielczość pionowa ekranu
//Ustawienia liczby obsługiwanych kolorów (1->65K, 0->262K)
#define COLOR_DEPTH_65K 1
//Informacja dla kompilatora czy korzystamy z możliwości odczytu obrazków z plików na karcie SD (używamy PetitFS)
#define USE_SD_PICTURE_SUPPORT 1
//Definicje rozkazów sterujących (przedrostek SET) i dostępnej listy ich argumentów (bez przedrostka SET)
#define SET_OSC_CTRL 0x02  //Konfiguracja sygnału zegarowego/oscylatora
#define OSC_WITH_EXT_RESISTOR (0<<6)
#define OSC_WITH_INT_RESISTOR (1<<6)
#define CLOCK_SCR_CLOCK_OFF 0b00
#define CLOCK_SCR_INT_OSC 0b01
#define CLOCK_SCR_EXT_CLOCK 0b11
//Konfiguracja częstotliwości ramki: częstotliwość oscylatora oraz wartość podzielnika (wybrane wartości)
#define SET_CLOCK_DIV 0x03
#define FRAME_RATE_75HZ (0b0000<<4)
#define FRAME_RATE_80HZ (0b0001<<4)
#define FRAME_RATE_85HZ (0b0010<<4)
#define FRAME_RATE_90HZ (0b0011<<4)
#define FRAME_RATE_95HZ (0b0100<<4)
#define FRAME_RATE_100HZ (0b0101<<4)
#define FRAME_RATE_105HZ (0b0110<<4)
#define FRAME_RATE_110HZ (0b0111<<4)
#define FRAME_RATE_115HZ (0b1000<<4)
#define FRAME_RATE_120HZ (0b1001<<4)
#define FREQ_DIVIDEDBY_1 0b0000
#define FREQ_DIVIDEDBY_2 0b0010
#define FREQ_DIVIDEDBY_3 0b0011
#define FREQ_DIVIDEDBY_4 0b0100
#define FREQ_DIVIDEDBY_5 0b0101
#define FREQ_DIVIDEDBY_6 0b0110
#define FREQ_DIVIDEDBY_7 0b0111
#define FREQ_DIVIDEDBY_8 0b1000
#define SET_REDUCE_CURRENT 0x04  //Konfiguracja poboru mocy przez peryferia sterownika
#define DRIVING_CURRENT_NORMAL (0<<2)
#define DRIVING_CURRENT_HALF (1<<2)
#define OSC_POWER_ON (0<<1)
#define OSC_POWER_OFF (1<<1)
#define POWER_SAVE_MODE_DISABLE (0<<0)
#define POWER_SAVE_MODE_ENABLE (1<<0)
#define SET_SOFT_RESET 0x05  //Reset softwareowy sterownika SEPS525
#define SOFT_RESET_OFF (0<<0)
#define SOFT_RESET_ON (1<<0)
#define SET_DISPLAY_ON_OFF 0x06  //Programowe sterowanie załączaniem ekranu
#define DISPLAY_OFF (0<<0)
#define DISPLAY_ON (1<<0)
#define SET_PRECHARGE_TIME_RED 0x08  //Czas wstępnego ładowania diod OLED (w cyklach oscylatora)
#define SET_PRECHARGE_TIME_GREEN 0x09
#define SET_PRECHARGE_TIME_BLUE 0x0A
#define SET_PRECHARGE_CURRENT_RED 0x0B  //Prąd wstępnego ładowania diod OLED (wartosc * 8uA)
#define SET_PRECHARGE_CURRENT_GREEN 0x0C
#define SET_PRECHARGE_CURRENT_BLUE 0x0D
#define SET_DRIVING_CURRENT_RED 0x10  //Prąd diod OLED (wartosc * 1uA)
#define SET_DRIVING_CURRENT_GREEN 0x11
#define SET_DRIVING_CURRENT_BLUE 0x12
#define SET_DISPLAY_MODE_SET 0x13  //Konfiguracja sposobu odzwierciedlenia danych obrazu w pamięci DDRAM sterownika
#define RGB_SWAP_OFF (0<<7)  //RGB->RGB
#define RGB_SWAP_ON (1<<7)  //RGB->BGR
#define ROW_SCAN_DIR_NORMAL (0<<5)  //0, 1, 2...127
#define ROW_SCAN_DIR_REVERSE (1<<5)  //127, 126, 125...0
#define COLUMN_SCAN_DIR_NORMAL (0<<4)  //0, 1, 2...159
#define COLUMN_SCAN_DIR_REVERSE (1<<4)  //159, 158, 157...0
#define ONE_SCREEN_MODE (0<<2)
#define TWO_SCREEN_MODE (1<<2)
#define COLUMN_DATA_DISP_CTRL_NORMAL 0b00
#define COLUMN_DATA_DISP_CTRL_ALL_LOW 0b01
#define COLUMN_DATA_DISP_CTRL_ALL_HIGH 0b10
#define SET_RGB_INTERFACE 0x14  //Konfiguruje rodzaj interfejsu RGB jak i magistrali sterującej
#define RGB_INTERFACE_18BIT (0b00<<4)
#define RGB_INTERFACE_16BIT (0b01<<4)
#define RGB_INTERFACE_6BIT (0b10<<4)
#define INTERFACE_MODE_MPU (1<<0)  //Magistrala sterująca MPU
#define INTERFACE_MODE_RGB (0<<0)  //Aktywny interfejs RGB sterownika
#define SET_RGB_POLARITY 0x15  //Konfiguruje polaryzację sygnałów interfejsu RGB
#define DOT_CLOCK_POL_0 (0<<4)  //Dane zatrzaskiwane przy zboczu rosnącym
#define DOT_CLOCK_POL_1 (1<<4)  //Dane zatrzaskiwane przy zboczu opadającym
#define VSYNC_OUT_DISABLE (0<<3)  //Dezaktywacja wyjścia sygnałów synchronizacji pionowej VSYNCO
#define VSYNC_OUT_ENABLE (1<<3)  //Aktywacja wyjścia sygnałów synchronizacji pionowej VSYNCO
#define POLARITY_ENABLE (0<<5)
#define POLARITY_DISABLE (1<<5)
```

List.4. c.d.
```
#define SET_MEM_WRITE_MODE 0x16  //Konfiguruje tryb zapisu do pamięci obrazu oraz zachowanie wewn. licznika adresów
#define SING_TRANS_18BIT_262K (0b000<<4)
#define SING_TRANS_16BIT_65K (0b010<<4)
#define DUAL_TRANS_9BIT_262K (0b100<<4)
#define DUAL_TRANS_8BIT_65K (0b110<<4)
#define TRIPLE_TRANS_8BIT_262K (0b111<<4)
#define HORIZ_ADDR_INCR (1<<2)  //Wewnętrzny licznik adresów DDRAM jest inkrementowany w pionie
#define HORIZ_ADDR_DECR (0<<2)  //Wewnętrzny licznik adresów DDRAM jest dekrementowany w pionie
#define VERT_ADDR_INCR (1<<1)  //Wewnętrzny licznik adresów DDRAM jest inkrementowany w poziomie
#define VERT_ADDR_DECR (0<<1)  //Wewnętrzny licznik adresów DDRAM jest dekrementowany w poziomie
#define DDRAM_WRITTEN_HORIZ (0<<0)  //Dane do DDRAM zapisywanie wierszami
#define DDRAM_WRITTEN_VERT (1<<0)  //Dane do DDRAM zapisywanie kolumnami
#define SET_MEM_X1_ADDR 0x17  //Ustawienia aktywnego obszaru pamięci ekranu DDRAM (X1-Y1 , X2-Y2)
#define SET_MEM_X2_ADDR 0x18
#define SET_MEM_Y1_ADDR 0x19
#define SET_MEM_Y2_ADDR 0x1A
#define SET_MEM_ACCESS_POINTER_X 0x20  //Współrzędna X początkowego adresu pamięci DDRAM przeznaczonego do zapisu
#define SET_MEM_ACCESS_POINTER_Y 0x21  //Współrzędna Y początkowego adresu pamięci DDRAM przeznaczonego do zapisu
#define SET_DDRAM_DATA_ACCESS_PORT 0x22  //Rozkaz dostępu do pamięci ekranu DDRAM (rozpoczyna zapis ekranu)
#define SET_DISP_DUTY_RATIO 0x28  //Wyznacza zakres wierszy dla mechanizmu odświeżania ekranu
#define SET_DISP_START_LINE 0x29  //Określa adres początkowego wiersza ekranu
#define SET_IREF 0x80  //Konfiguruje źródło napięcia odniesienia
#define REF_VOLT_CTRL_BY_EXT_RESISTOR (0<<0)
#define REF_VOLT_CTRL_BY_INT_RESISTOR (1<<0)
```

Jak wspomniano, wyświetlacz MI160128AO wyposażono w rozbudowany, scalony sterownik ekranu SEPS525 produkcji Synocoam Co. Ltd. Układ wyposażono w szereg zaawansowanych, sprzętowych funkcji sterujących czyniąc obsługę panelu OLED bardzo łatwą. Układ integruje w sobie 480 źródeł prądowych (po 160 dla każdej składowej RGB) przeznaczonych do sterowania pracą kolumn matrycy OLED (wyprowadzeń wspólnych anod diod OLED) z możliwością programowej zmiany prądu (w 256 krokach, maksymalnie do wartości 255 μA), 128 kluczy tranzystorowych przeznaczonych do sterowania pracą wierszy (wspólnych katod diod OLED), blok formujący napięcia referencyjne dla całej architektury sterownika, zintegrowany, stabilizowany oscylator dostarczający przebiegi zegarowe do sterowania pracą wszystkich, wewnętrznych modułów oraz pamięć ekranu DDRAM

Listing 5. Listing funkcji inicjalizacji sterownika SEPS525 i włączenia panela OLED.
```
void SEPS525init(void)
{
  //Port danych OLED jako port wyjsciowy ze stanem 0x00
  DATA_DDR = 0xFF;
  //Porty sterujące OLEDa jako wyjsciowe ze stanami: RST=1, RD=1 (nie korzystamy z operacji odczytu), pozostałe=0
  CONTROL_PORT |= (1<<RD_PIN) | (1<<RS_PIN);
  CONTROL_DDR |= (1<<WR_PIN) | (1<<RD_PIN) | (1<<CS_PIN) | (1<<RS_PIN) | (1<<RST_PIN);
  //Port sterujący zasilaniem OLEDa jako port wyjsciowy ze stanem „0” czyli zasilanie WYŁĄCZONE
  VCC_DDR |= (1<<VCC_PIN);
  RESET_RST;  //Zerowanie sprzętowe sterownika SEPS525
  _delay_ms(1);
  SET_RST;
//Konfiguracja poboru mocy przez peryferia sterownika SEPS525 - sekwencja wymagana przez dokumentację
  WriteRegister(SET_REDUCE_CURRENT, DRIVING_CURRENT_NORMAL | OSC_POWER_ON | POWER_SAVE_MODE_ENABLE);
  _delay_ms(1);
  WriteRegister(SET_REDUCE_CURRENT, DRIVING_CURRENT_NORMAL | OSC_POWER_ON | POWER_SAVE_MODE_DISABLE);
//Programowe wyłączenie panela OLED
  WriteRegister(SET_DISPLAY_ON_OFF, DISPLAY_OFF);
//Programowe zerowanie sterownika SEPS525
  WriteRegister(SET_SOFT_RESET, SOFT_RESET_ON);
  WriteRegister(SET_SOFT_RESET, SOFT_RESET_OFF);
//Konfiguracja sygnału zegarowego/oscylatora
  WriteRegister(SET_OSC_CTRL, OSC_WITH_EXT_RESISTOR | CLOCK_SCR_INT_OSC);
//Konfiguracja czestotliwosci ramki: czestotliwosc oscylatora oraz wartosc podzielnika
  WriteRegister(SET_CLOCK_DIV, FRAME_RATE_90HZ | FREQ_DIVIDEDBY_1);
//Konfiguracja czasu wstępnego ładowania dla sterowników diod OLED (w cyklach oscylatora)
  WriteRegister(SET_PRECHARGE_TIME_RED, 0x04);
  WriteRegister(SET_PRECHARGE_TIME_GREEN, 0x05);
  WriteRegister(SET_PRECHARGE_TIME_BLUE, 0x05);
//Konfiguracja prądu wstępnego ładowania dla sterowników diod OLED (wartosc * 8uA)
  WriteRegister(SET_PRECHARGE_CURRENT_RED, 0x9D);  //ewentualnie: 0x56
  WriteRegister(SET_PRECHARGE_CURRENT_GREEN, 0x8C);  //ewentualnie: 0x4D
  WriteRegister(SET_PRECHARGE_CURRENT_BLUE, 0x57);  //ewentualnie: 0x46
//Konfiguracja prądu sterowników diod OLED (wartosc * 1uA)
  WriteRegister(SET_DRIVING_CURRENT_RED, 0x45);  //ewentualnie: 0x0A
  WriteRegister(SET_DRIVING_CURRENT_BLUE, 0x34);  //ewentualnie: 0x0A
  WriteRegister(SET_DRIVING_CURRENT_GREEN, 0x33);  //ewentualnie: 0x0A
  //Konfiguracja sposobu odzwierciedlenia danych obrazu w pamięci DDRAM sterownika
  WriteRegister(SET_DISPLAY_MODE_SET, RGB_SWAP_OFF | ROW_SCAN_DIR_NORMAL | COLUMN_SCAN_DIR_NORMAL | ONE_SCREEN_MODE
|COLUMN_DATA_DISP_CTRL_NORMAL);
//Konfiguracja typu interfejsu RGB jak i aktywnej magistrali sterującej
  WriteRegister(SET_RGB_INTERFACE, RGB_INTERFACE_16BIT | INTERFACE_MODE_MPU);
//Konfiguracja polaryzacji sygnałów interfejsu RGB
  WriteRegister(SET_RGB_POLARITY, DOT_CLOCK_POL_0 | VSYNC_OUT_DISABLE  | POLARITY_ENABLE);
//Konfiguracja trybu zapisu do pamięci obrazu, liczby kolorów oraz zachowania wewn. licznika adresów DDRAM
  #if COLOR_DEPTH_65K == 1
    WriteRegister(SET_MEM_WRITE_MODE, DUAL_TRANS_8BIT_65K | HORIZ_ADDR_INCR | VERT_ADDR_INCR | DDRAM_WRITTEN_HORIZ);
  #else
    WriteRegister(SET_MEM_WRITE_MODE, TRIPLE_TRANS_8BIT_262K | HORIZ_ADDR_INCR | VERT_ADDR_INCR | DDRAM_WRITTEN_HORIZ);
  #endif
//Ustawienie aktywnego obszaru pamięci DDRAM: X1/Y1 i X2/Y2 oraz początkowego adresu tejże pamięci przeznaczonego do
zapisu
  SetActiveWindow(0, HORIZ_RESOLUTION-1, 0, VERT_RESOLUTION-1);
//Konfiguracja zakresu wierszy dla mechanizmu odswierzania ekranu
  WriteRegister(SET_DISP_DUTY_RATIO, VERT_RESOLUTION-1);
//Konfiguracja adresu pierwszego wiersza ekranu
  WriteRegister(SET_DISP_START_LINE, 0x00);
//Konfiguracja źródła napięcia odniesienia
  WriteRegister(SET_IREF, REF_VOLT_CTRL_BY_EXT_RESISTOR);
  OLEDcls();  //Wyczyszczenie pamięci obrazu DDRAM
  OLEDon();  //Załączenie napięcia zasilającego matrycę
}
```

Rysunek 2. Wygląd okna aplikacji Bitmap2RGBconverter

o organizacji 160×18(RGB)×128 bitów. Uproszczony schemat funkcjonalny układu SEPS525 w zakresie sterownika matrycy OLED przedstawiono na rysunku **rysunku 1**.

Szczegóły konstrukcyjne sterowników paneli OLED nie są tematem niniejszego artykułu, więc nie będę się wdawał w niuanse implementacyjne. Przypomnę tylko, że sterowanie pracą matrycy diod OLED (jak zresztą typowej diody LED) odbywa się dzięki zastosowaniu źródeł prądowych sterowanych cyfrowo, które zapewniają wysoki prąd ładowania wstępnego (pozwalający na szybkie naładowanie pojemności pasożytniczej diody OLED) oraz właściwy prąd sterujący, przy czym sama kontrola jasności pojedynczego piksela jest dokonywana poprzez zmianę wypełnienia impulsu sterującego generowanego przez sterownik kolumn. Czytelników szczególnie zainteresowanych tą tematyką odsyłam do bardzo ciekawego opracowania firmy Solomon Systech Ltd. dostępnego w Internecie pod tytułem „OLED Driver IC Optimizes Display Performance" lub do artykułu mojego autorstwa opublikowanego w EP 01/2011 i 02/2011 pt. „Obsługa monochromatycznych wyświetlaczy OLED ze sterownikiem SSD1325".

Wróćmy, zatem do naszego wyświetlacza, ponieważ obiecałem skupić się na zagadnieniach programistycznych. Moduł ten wyposażony został w 35-stykowe złącze typu ZIF, którego rozkład wyprowadzeń wraz z opisem ich funkcjonalności zamieszczono w **tabeli 1**.

Już z pobieżnej analizy opisów wyprowadzeń złącza możemy wnioskować, że wprowadzono typowe sygnały sterujące, z którymi można się spotkać przy każdego rodzaju modułach wyświetlaczy graficz-

Listing 6. Listing funkcji SetActiveWindow

```
static void SetActiveWindow(uint8_t X1, uint8_t Y1, uint8_t X2, uint8_t Y2)
{
  WriteRegister(SET_MEM_X1_ADDR, X1);  //Adres startowy aktywnego obszaru pamięci ekranu dla osi X
  WriteRegister(SET_MEM_X2_ADDR, X2);  //Adres końcowy aktywnego obszaru pamięci ekranu dla osi X
  WriteRegister(SET_MEM_Y1_ADDR, Y1);  //Adres startowy aktywnego obszaru pamięci ekranu dla osi Y
  WriteRegister(SET_MEM_Y2_ADDR, Y2);  //Adres końcowy aktywnego obszaru pamięci ekranu dla osi Y
  WriteRegister(SET_MEM_ACCESS_POINTER_X, X1);  //Współrzędna X początkowego adresu pamięci DDRAM przeznaczonego do zapisu
  WriteRegister(SET_MEM_ACCESS_POINTER_Y, Y1);  //Współrzędna Y początkowego adresu pamięci DDRAM przeznaczonego do zapisu
}
```

Listing 7. Listing funkcji umożliwiającej wyświetlenie kolorowego obrazu zapisanego w tablicy typu uint8_t.

```
void DrawPicture(uint8_t X1, uint8_t Y1, const uint8_t *Picture)
{
  uint16_t Bytes, i = 2;
/* Obliczamy liczbę bajtów niezbędnych do wysłania do pamięci ekranu DDRAM w zależnosci od wymiarów obrazka (okreslają
je dwa, pierwsze bajty tablicy Picture) jak i aktulanej głębi kolorów. Wspomniana tablica, co oczywiste, musi miec
odpowiednią budowę wynikającą z oczekiwanej liczby kolorów, za co odpowiada dedykowana aplikacja konwertera obrazów.
W wersji 262K głębi kolorów wykorzystujemy pełną, 18-bitową paletę kolorów (RGB typu 666) , inaczej niż to ma miejsce
w pozostałych funkcjach. */
  #if COLOR_DEPTH_65K == 1
    Bytes = 2*(pgm_read_byte(&Picture[0]))*pgm_read_byte(&Picture[1]);
  #else
    Bytes = 3*(pgm_read_byte(&Picture[0]))*pgm_read_byte(&Picture[1]);
  #endif
//Ustawiamy aktywne okno pamięci obrazu DDRAM sterownika SEPS525 by uprościć zapis danych
  SetActiveWindow(X1, Y1, X1+pgm_read_byte(&Picture[0])-1, Y1+pgm_read_byte(&Picture[1])-1);
//Komenda inicjująca zapis do pamięci ekranu DDRAM
  WriteCommand(SET_DDRAM_DATA_ACCESS_PORT);
  while(i < Bytes+2) WriteData( pgm_read_byte(&Picture[i++]) );
}
```

Listing 8. Listing funkcji umożliwiającej wyświetlenie kolorowego obrazu zapisanego na karcie SD w pliku *.rgb.

```
void DrawSDPicture(uint8_t X1, uint8_t Y1, char *fileName)
{
  register uint8_t Width, Height;
  uint16_t  Bytes, i;
  WORD readBytes;
  if(pf_open(fileName) != FR_OK) return;  //Jesli brak pliku na karcie SD to kończymy działanie funkcji
//Odczyt pierwszego sektora o wielkosci 512 bajtów
  pf_read(SDbuffer, 512, &readBytes);  //Zmienna readBytes informuje nas o rzeczywistej liczbie odczytanych bajtów
//Wyznaczenie szerokosci i wysokosci obrazka (pierwsze 2 bajty, gdyż obrazek jest mniejszy jest niż 256x256px)
  Width = SDbuffer[0]; Height = SDbuffer[1];
//Obliczenie ilości bajtów reprezentujących obrazek w zależnosci od formatu (565 lub 666)
  #if COLOR_DEPTH_65K == 1
    Bytes = 2*Width*Height;
  #else
    Bytes = 3*Width*Height;
  #endif
//Ustawiamy aktywne okno pamięci obrazu DDRAM sterownika SEPS525 by uprościć zapis danych
  SetActiveWindow(X1, Y1, X1+Width-1, Y1+Height-1);
//Komenda inicjująca zapis do pamięci ekranu DDRAM
  WriteCommand(SET_DDRAM_DATA_ACCESS_PORT);
//Wysyłamy pozostałą część bajtów z otwartego wczeniej bufora pomijając 2 pierwsze bajty definiujące rozmiar obrazka
  for(i=2; i<readBytes; i++)
  {
    WriteData(SDbuffer[i]);
    Bytes--;
  }
//Jeśli są jeszcze jakiekolwiek dane do wyświetlenia odczytujemy je diu bufora SD i wyświetlamy
  while(Bytes)
  {
    pf_read(SDbuffer, 512, &readBytes);  //Zmienna readBytes informuje nas o rzeczywistej liczbie odczytanych bajtów
    for(i=0; i<readBytes; i++)
    {
      WriteData(SDbuffer[i]);
      Bytes--;
    }
  }
}
```

Listing 9. Konstrukcja struktury opisującej zestaw znaków
```
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 przypadających na definicję 1 znaku
  uint8_t FirstCharCode;  //Kod ASCII pierwszego znaku w tablicy wzorców
  const uint8_t *Bitmap;  //Wskaźnik to tablicy zawierającej wzorce poszczególnych znaków
} fontDescription;
```

nych, co z kolei przekłada się na podobieństwo procedur sterujących. Przypomnę już po raz kolejny, iż sterowanie pracą wyświetlaczy tego typu polega, po pierwsze, na wysyłaniu predefiniowanych rozkazów sterujących i towarzyszących im argumentów, za pomocą których ustawiamy niezbędne parametry konfiguracyjne sterownika ekranu zaś po drugie, na zapisie (lub też odczycie) danych do/z pamięci ekranu DDRAM powodującej ich natychmiastowe wyświetlenie. Interpretacja rodzaju danych, które mają zostać odebrane przez chipset SEPS525, jest zdeterminowana poziomem wyprowadzenia RS. Jest to typowe rozwiązanie w sterowaniu wyświetlaczami różnego typu. Poziom niski na tym wyprowadzeniu decyduje o tym czy przesyłana wartość zostanie zinterpretowana jako rozkaz sterujący, zaś poziom wysoki powoduje, że dana ta zinterpretowana zostanie jako argument przesłanego wcześniej rozkazu sterującego lub słowo pamięci ekranu DDRAM. W związku z powyższą organizacją procedur sterujących można wyróżnić dwa rodzaje sekwencji operacji zapisu do układu SEPS525 (operacje odczytu przebiegają analogicznie, lecz nie będą tematem naszych rozważań):

- sekwencja zapisu rozkazu sterującego,
- sekwencja zapisu argumentu rozkazu sterującego lub danych do pamięci ekranu DDRAM.

Wspomnianym sekwencjom sterującym odpowiadają funkcje, które zamieszczono na **listingu 1** i **listingu 2**.

Dla uproszczenia kolejnej procedury, odpowiedzialnej za konfigurację sterownika SEPS525, wprowadzimy jeszcze jedną, prostą funkcję narzędziową, za pomocą której prześlemy do sterownika ekranu rozkaz sterujący wraz z towarzyszącym mu argumentem. Ciało tej funkcji narzędziowej pokazano na **listingu 3**.

Posiłkując się tymi elementarnymi funkcjami narzędziowymi umożliwiającymi komunikację ze sterownikiem wyświetlacza możemy przejść do funkcji służącej inicjalizacji sterownika SEPS525 i włączeniu zasilania panelu OLED. Zanim jednak przejdę do omawiania wspomnianej funkcji inicjalizacyjnej pokażę zawartość pliku nagłówkowego *SEPS525drv.h*, w którym zdefiniowano nazwy (oraz odpowiadające im wartości) interesujących nas rozkazów sterujących jak i możliwych wartości ich argumentów.

Proszę o zwrócenie uwagi na sposób zapisu poszczególnych definicji. Preferuję tego typu rozwiązanie, gdyż umożliwia mi ono szybkie zorientowanie się w możliwościach danego peryferium bez jakiegokolwiek zaglądania do jego dokumentacji, gdyż każdy definiowany parametr (nazwa rozkazu, dostępne wartości argumentów) został bardzo jednoznacznie zidentyfikowany. Wspomniany plik nagłówkowy pokazano na **listingu 4**.

Znając definicje pokazane na **listingu 4** bez trudu i słowa komentarza zorientujemy się, jakiego rodzaju konfiguracje przeprowadzane są w zakresie funkcji inicjalizacyjnej, której ciało przedstawiono na **listingu 5**.

Prawda, że łatwe? Mając tak skonstruowany plik nagłówkowy, bez trudu zorientujemy się, jakiego rodzaju konfiguracji poddawany jest sterownik SEPS525 w trakcie procedury inicjalizacji. Krótkiej wzmianki wymaga funkcja *SetActiveWindow*. Za jej pomocą definiujemy aktywny obszar ekranu ograniczony współrzędnymi jego górnego, lewego wierzchołka (X1/Y1) oraz dolnego, prawego wierzchołka (X2/Y2) w ramach, którego w trakcie operacji zapisu/odczytu sterownik SEPS525 automatycznie inkrementuje wewnętrzny licznik adresów. Jest to także dość typowe rozwiązanie stosowane powszechnie w implementacji wszelkiego rodzajów sterowników wyświetlaczy graficznych, które znacznie upraszcza procedury wyświetlające znaki czy obrazy. Ciało funkcji *SetActiveWindow* pokazano na **listingu 6**.

W tym momencie, potrafimy już zainicjować interesujący nas panel OLED w związku, z czym przejdziemy do opisu funkcji odpowiedzialnych za wyświetlanie obrazów (zapisanych zarówno w pamięci Flash mikrokontrolera jak i na karcie pamięci SD) jak i czcionek ekranowych. Pierwszym z zadań, jakie przede mną stanęły było opracowanie efektywnej funkcji pozwalającej na wyświetlanie obrazów na ekranie naszego panela OLED. Jako, że dla 8-bitowych mikrokontrolerów AVR obsługa plików BMP a w szczególności JPG w czasie rzeczywistym byłaby dość dużym wyzwaniem postanowiłem napisać prostą aplikację umożliwiającą konwersję wspomnianych typów plików do prostego formatu zawierającego surowe dane poszczególnych pixeli obrazu. Taka oto powstała prosta aplikacja *Bitmap2RGBconverter*, której wygląd pokazano na **rysunku 2**.

Za pomocą tej prostej aplikacji każdy plik typu BMP czy JPG może zostać skonwertowany do dwóch typów danych wyjściowych:

- pliku typu *.c zawierającego definicję tablicy bajtów w pamięci Flash opisujących kolory poszczególnych pixeli obrazu,
- pliku typu *.rgb zawierającego dane binarne j.w.

Ponadto, program ten umożliwia wybór docelowej palety kolorów spośród wartości RGB565 (dwa bajty opisujące kolor pojedynczego piksela obrazu) lub RGB666 (3 bajty opisujące kolor pojedynczego piksela obrazu) oraz nadanie nazwy generowanej tablicy, dla przypadku pliku *.c. Należy podkreślić, iż niezależnie od wybranego formatu danych wyjściowych, dwa (a dla szerokości>255px, trzy) pierwsze bajty pliku docelowego zawierają szerokość i wysokość obrazka. Posiłkując się wspomnianą wyżej aplikacją oraz funkcjami przedstawionymi na **listingu 7** i **listingu 8** otrzymujemy możliwość wyświetlenia obrazu na naszym panelu OLED, którego dane umieszczone są albo w tablicy, którą należy dołączyć do właściwej aplikacji lub też na karcie SD (użyto funkcji biblioteki PetitFS autorstwa Elm Chan).

Jak widać, wyświetlanie obrazów wygenerowanych przez aplikację *Bitmap2RGB-*

Listing 10. Konstrukcja funkcji SetFont
```
void SetFont(const fontDescription *Font)
{
  CurrentFont.Width = pgm_read_byte(&Font->Width);
  CurrentFont.Height = pgm_read_byte(&Font->Height);
  CurrentFont.Interspace = pgm_read_byte(&Font->Interspace);
  CurrentFont.BytesPerChar = pgm_read_byte(&Font->BytesPerChar);
  CurrentFont.FirstCharCode = pgm_read_byte(&Font->FirstCharCode);
  CurrentFont.Bitmap = (uint8_t*)pgm_read_word(&Font->Bitmap);
}
```

Listing 11. Konstrukcja funkcji odpowiedzialnej za wyświetlanie napisów na wyświetlaczu OLED

```
void DrawText(uint8_t X1, uint8_t Y1, char *Text, uint16_t Color, uint16_t Background)
{
  register char Character;
  register uint8_t widthIndex, heightIndex, widthByteNr, readByte, pixelsNr, i;
  uint16_t offset;
//Sprawdzamy kody ASCII kolejnych znaków napisu wejsciowego aż do wystąpienia terminatora kończącego C-string
  while ((Character = *Text++))
  {
    if(Character == ' ') DrawFilledBox(X1, Y1, X1+CurrentFont.Width-1, Y1+CurrentFont.Height-1, Background);
    else
    {
//Ustawiamy aktywne okno pamięci obrazu DDRAM sterownika SEPS525 by uprościć zapis danych
      SetActiveWindow(X1, Y1, X1+CurrentFont.Width-1, Y1+CurrentFont.Height-1);
//Komenda inicjująca zapis do pamięci ekranu DDRAM
      WriteCommand(SET_DDRAM_DATA_ACCESS_PORT);
//Obliczamy offset położenia wzroca bieżącego znaku ASCII w tablicy wzorców znaków
      offset = (Character-CurrentFont.FirstCharCode)*CurrentFont.BytesPerChar;
      for(heightIndex = 0; heightIndex < CurrentFont.Height; heightIndex++)
      {
        for(widthIndex = 0, widthByteNr = 0; widthIndex < CurrentFont.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 nie bedacej wielokrotnoscia liczby 8 każdy, ostatni w wierszu bajt definicji wzorca
//nie jest w pełni wykorzystany, jesli chodzi o poszczególne bity w związku z czym w takim przypadku musimy
//ustalic użyteczną liczbę pixeli przeznaczonych do przesłania do sterownika SEPS525
pixelsNr = ((widthByteNr+1)*8) <= CurrentFont.Width ? 8 : CurrentFont.Width - (widthByteNr *8);
          for(i=0; i<pixelsNr; i++)
          {
            if(readByte&0x80)  Draw_Pixel(Color);  //Pixel aktywny w kolorze Color
            else Draw_Pixel(Background);  //Piksel nieaktywny w kolorze tła Background
            readByte<<=1;
          }
        }
      }
    }
    X1+=(CurrentFont.Width+CurrentFont.Interspace);  //Przesuwamy współrzędną X1 o szerokość znaku plus zdefiniowany
odstęp
  }
  SetActiveWindow(0, HORIZ_RESOLUTION-1, 0, VERT_RESOLUTION-1);
}
```

Wykaz elementów

Rezystory: (obudowy SMD 0603)
R1: 187 kΩ (1%)
R2: 18 kΩ (1%)
R3: 47 kΩ
R4: 68 kΩ
R5: 20 kΩ
R6: 10 kΩ
R7: 2,2 kΩ

Kondensatory:
C1: 10 µF/16 V (ceramiczny X5R, SMD 0805)
C2:1 µF/16 V (ceramiczny X5R, SMD 0603)
C3, C8…C10, C13…C15:100 nF (ceramiczny X5R, SMD 0603)
C4: 3,3 nF (ceramiczny X5R, SMD 0603)
C5: 1 µF/20 V (SMD A, EIA 3216-18W)
C5A, C5B: 10 µF/25 V (ceramiczny X5R, SMD 0805)
C6, C7, C11, C12: 4,7 µF/20 V (SMD A, EIA 3216-18W)

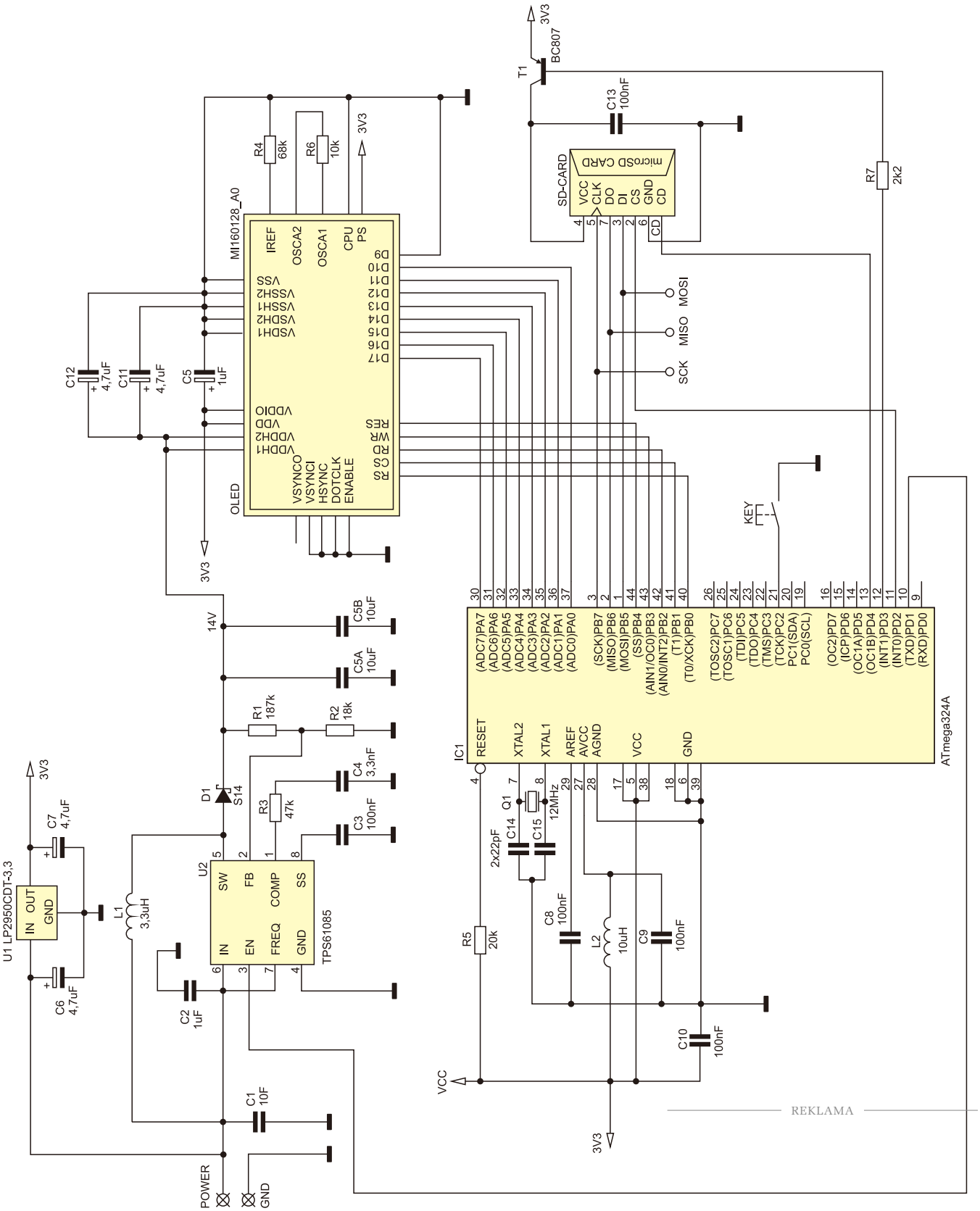
Półprzewodniki:
U1: LP2950CDT-3.3 (obudowa DPACK)
U2: TPS61085 (obudowa TSSOP8)
U3: ATmega324A (obudowa TQFP44)
D1: SS14 (obudowa SMA)
T1: BC807 (obudowa SOT23-BEC)

Inne:
OLED: wyświetlacz kolorowy OLED MI160128_A0 (MULTI-INNO TECHNOLOGY)
L1: dławik mocy 3,3 µH typu DLG-0504-3R3
L2: dławik 10 µH (obudowa SMD 1206)
Q1: rezonator kwarcowy SMD 12 MHz typ 7M-12.000MAAJ-T (TXC, obudowa 3,2 mm×2,5 mm)
SD-CARD: gniazdo karty microSD push-push typ MEM2051-00-195-00-A z wyprowadzeniem Card Detect (GLOBAL CONNECTOR TECHNOLOGY, obudowa SMT)
KEY: microswitch SMD DTSM31
ZIF: złącze typu ZIF do montażu powierzchniowego (raster 0.5 mm, 35-pin, dolny kontakt)

converter jest łatwe i efektywne. Na koniec tej tematyki chciałbym w dużym skrócie omówić zagadnienie generowania jak i wyświetlania własnych czcionek ekranowych. Tym razem, do utworzenia stosownych tablic i struktur zawierających używane w programie obsługi czcionki wykorzystano doskonały program firmy Atnel (www.atnel.pl) o nazwie PixelFactory, którego autorem jest Mirosław Kardaś. Program ten, oprócz dostępnej innej funkcjonalności, umożliwia utworzenie wzorców czcionek na podstawie wybranych przez użytkownika fontów dostępnych w systemie Windows, przy czym możliwe jest w zasadzie dowolne skalowanie elementów wyjściowych jak i wybór zakresu dostępnych znaków. Wynikiem działania programu jest utworzenie dwóch plików: plik C zawierający tablice wzorców wygenerowanych znaków jak i 2 struktury opisujące parametry fizyczne zestawu tychże znaków oraz plik H zawierający deklaracje używanych tablic/struktur jak i nowych typów danych. Nie będę w tym miejscu rozpisywał się na temat obsługi programu PixelFactory, gdyż po pierwsze jest ona niezmiernie intuicyjna a po drugie, stosowne artykuły znaleźć można w Internecie, lecz skupię się na kwestii konstrukcji wspomnianych struktur w świetle ich implementacji dla wyświetlaczy OLED zaznaczając od razu, iż w naszym programie obsługi zmodyfikujemy nieco sposób dostępu do wzorców znaków, jako że nie jest nam potrzebna tak duża elastyczność użytkowa, jaką przewidział autor tego, skądinąd, unikalnego programu. Najważniejszą strukturą z punktu widzenia programu obsługi urządzenia jest ta, która niesie informację na temat cech fizycznych zestawu znaków jak i samego znaku. Konstrukcję takiej struktury pokazano na **listingu 9**.

Tablica wzorców poszczególnych znaków, do której wskaźnik umieszczony zostanie we wspomnianej strukturze jest typową tablicą *uint8_t Tablica[] PROGMEM* zawierającą wygenerowane przez program PixelFactory wzorce znaków. Nie korzystamy zarazem z trzeciej struktury, którą tworzy oprogramowanie, a która to zawiera informację o kodach znaków (i stosownym offsecie) umieszczonych w tablicy wzorców, gdyż moim zdaniem zwykle korzystamy z ciągłego przedziału dostępnych znaków standardu ASCII w odpowiednio ograniczonym zakresie (np. same cyfry lub małe litery). Dla wygody przewidziano dodatkową funkcję SetFont, której zadaniem jest odczytanie z pamięci Flash parametrów bieżącej czcionki do zmiennej umieszczonej w pamięci RAM, co upraszcza nieco konstrukcję właściwej funkcji odpowiedzialnej za wyświetlanie tekstu. Konstrukcję wspomnianej funkcji jak i docelowej funkcji odpowiedzialnej za wyświetlanie napisów zamieszczono na **listingu 10** i **listingu 11**.

Dla porządku należy dodać, iż użyta w ciele funkcji *DrawText* funkcja *Draw_Pixel* powoduje wysłanie do pamięci ekranu dwóch lub trzech bajtów danych opisujących wybrany kolor (w zależności od wybranej głębi kolorów), zaś funkcja *DrawFilledBox* powoduje wyświetlenie na ekranie wyświetlacz OLED prostokąta wypełnionego wybranym kolorem o współrzędnych określonych argumentami wywołania.

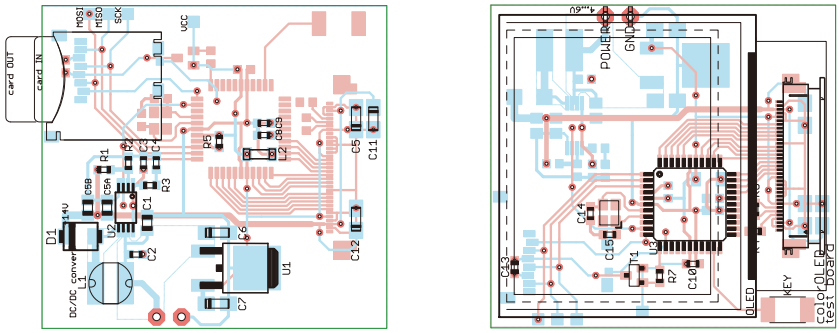


Rysunek 3. Schemat ideowy cyfrowej ramki zdjęciowej z kolorowym wyświetlaczem OLED

W tym miejscu znamy już wszystkie szczegóły dotyczące sposobu obsługi wyświetlaczy OLED wyposażonych w sterownik ekranu SEPS525 w związku, z czym przejdźmy do obiecanego wcześniej rozwiązania praktycznego, jakim jest cyfrowa ramka zdjęciowa, której chemat ideowy pokazano na **rysunku 3**.

Jak widać, i co może okazać się zdumiewające, jest to niezmiernie prosty system mikroprocesorowy składający się zaledwie z czterech, głównych elementów: mikrokontrolera ATmega324A, wyświetlacza OLED o rozdzielczości 160×128 pikseli, gniazda kart microSD typu push-push oraz przetwornicy typu step-up o oznaczeniu TPS61085

Ustawienia Fuse-bitów (ważniejszych):
CKSEL3...0: 1111
SUT1...0: 11
CKDIV: 1
CKOUT: 1
JTAGEN: 1
BODEN: 1
SPIEN: 0
OCDEN: 1
BOOTRST: 1



Rysunek 4. Schemat montażowy cyfrowej ramki zdjęciowej z kolorowym wyświetlaczem OLED

produkcji firmy Texas Instruments w swej, podstawowej aplikacji pracy. Wybór ostatnio wspomnianego elementu nie był przypadkowy! Przetwornica ta odznacza się doskonałymi parametrami elektrycznymi, z których najważniejsze to wysoka sprawność dochodząca do 93%, szeroki zakres napięcia wejściowego (2,3 do 6 V), opcja miękkiego startu, szereg wbudowanych zabezpieczeń (termiczne, od niskiego napięcia wejściowego) oraz duża częstotliwość przełączania 1,2 MHz, dzięki czemu można stosować elementy o mniejszych wymiarach (mniejsza indukcyjność i pojemność). Dodatkowo, układ TPS61085 wyposażono w wejście EN (*Enable*) przy pomocy, którego możemy wyłączyć przetwornicę, jeśli zajdzie taka potrzeba (np. wygaszenie wyświetlacza) i tym samym ograniczyć pobór prądu ze źródła napięcia zasilającego.

Kilka słów uwagi należy się obsłudze karty pamięci SD, jako że jest to zwykle dość skomplikowane zagadnienie z punktu widzenia programisty. Uważny Czytelnik z pewnością już zauważył, że zasoby sprzętowe mikrokontrolera zostały użyte w naszym systemie tak naprawdę w niewielkim stopniu, jednak należy mieć na uwadze, iż urządzenie to musi zapewnić obsługę karty pamięci SD sformatowanej dla systemu plików FAT (16/32), co pociąga za sobą konieczność stosowania dość „pamięciożernych" bibliotek, a więc i zaopatrzonych w odpowiednią ilość pamięci RAM mikrokontrolerów. Co należy szczególnie podkreślić, prosta implementacja tej funkcjonalności możliwa stała się dzięki wykorzystaniu gotowej biblioteki PetitFS autorstwa Elm Chan, dzięki której tak skomplikowane zagadnienie stało się niezmiernie proste a poza tym niezbyt wymagające, jeśli chodzi o obciążenie mikrokontrolera (w porównaniu np. do biblioteki AVR-DOS dostarczanej wraz z kompilatorem Bascom, której to autorem jest Franz Vögel). Co oczywiste, używana karta pamięci powinna być odpowiednio sformatowana. Dodatkowo, w konstrukcji urządzenia wykorzystano specjalne wyprowadzenie złącza karty SD oznaczone jako CD a służące do sprawdzenia jej obecności w gnieździe, co powinno zapobiec potencjalnym problemom programistycznym. Co więcej, w projekcie naszej ramki zdjęciowej skorzystano z mośliwości programowego sterowania zasilaniem wspomnianej karty pamięci dzięki zastosowaniu pary tranzystor T1/rezystor R7, co przydaje się szczególnie w przypadku niepoprawnej inicjalizacji sterownika karty lub też problemów ze stosowną komunikacją. W takich przypadkach, program główny, wykonuje restart zasilania karty SD i rozpoczyna jej ponowną inicjalizację.

Montaż

Projekt płytki drukowanej ramki zdjęciowej pokazano na **rysunku 4**. Jak widać zaprojektowano bardzo zwartą i niewielką konstrukcję z zastosowaniem wyłącznie montażu SMD umieszczając, na co należy zwrócić szczególną uwagę, poszczególne elementy po obu stronach obwodu drukowanego. Jak zwykle w przypadku tego typu systemów, rozpoczynamy od wlutowania elementów o najbardziej kłopotliwej konstrukcji, czyli układów scalonych SMD oraz złącza ZIF. Następnie przechodzimy do przylutowania wszystkich elementów biernych zaś na sam koniec pozostawiamy wszelkiego rodzaju elementy mechaniczne. Każdorazowo należy uważać by nie uszkodzić termicznie poszczególnych podzespołów, zwłaszcza półprzewodników. Z uwagi na zagęszczenie wyprowadzeń złącza ZIF jak i układów scalonych przed pierwszym podłączeniem układu należy jeszcze raz sprawdzić jakość wykonanych połączeń by nie dopuścić do ewentualnych zwarć. Wspomniana kontrola będzie znacznie łatwiejsza, jeśli zmontowana płytkę sterownika przemyjemy alkoholem izopropylowym w celu wypłukania nadmiaru kalafonii lutowniczej. Z uwagi na fakt, iż zastosowany typ wyświetlacza nie posiada jakichkolwiek elementów) ułatwiających jego montaż, moduł ten najlepiej jest przytwierdzić do obwodu drukowanego urządzenia za pomocą dwustronnej taśmy klejącej. Poprawnie zmontowany układ powinien działać od razu po dołączeniu zasilania. Ewentualnego sprawdzenia może wymagać wartość napięcia wyjściowego przetwornicy, która to powinna wynosić 14±0.5 V. Warto podkreślić, iż należy zwrócić szczególną uwagę na polaryzację źródła zasilania, jako że urządzenie nie zostało zabezpieczone przed przypadkowym jej odwróceniem.

Obsługa

Na temat samej obsługi urządzenia można tak naprawdę napisać niewiele, gdyż jest ona niezmiernie prosta. Funkcja *int main(void)* przeszukuje w nieskończonej pętli *while(1)* główny katalog karty SD w poszukiwaniu plików *.rgb po czy sekwencyjnie, w cyklach 2-sekundowych, wyświetla znalezione obrazki. Przejście do kolejnego pliku z obrazem może zostać przyspieszone poprzez naciśnięcie przycisku *KEY*. Po wyświetleniu ostatniego, dostępnego w katalogu głównym pliku typu *rgb, algorytm funckji głównej powraca na początek katalogu. Wszelkie, potencjalne problemy związane z inicjalizacją sterownika karty SD, jej nieobecnością w gnieździe czy też dotyczące systemu plików zostaną każdorazowo zasygnalizowane poprzez wyświetlenie odpowiedniego komunikatu.

Robert Wołgajew, EP