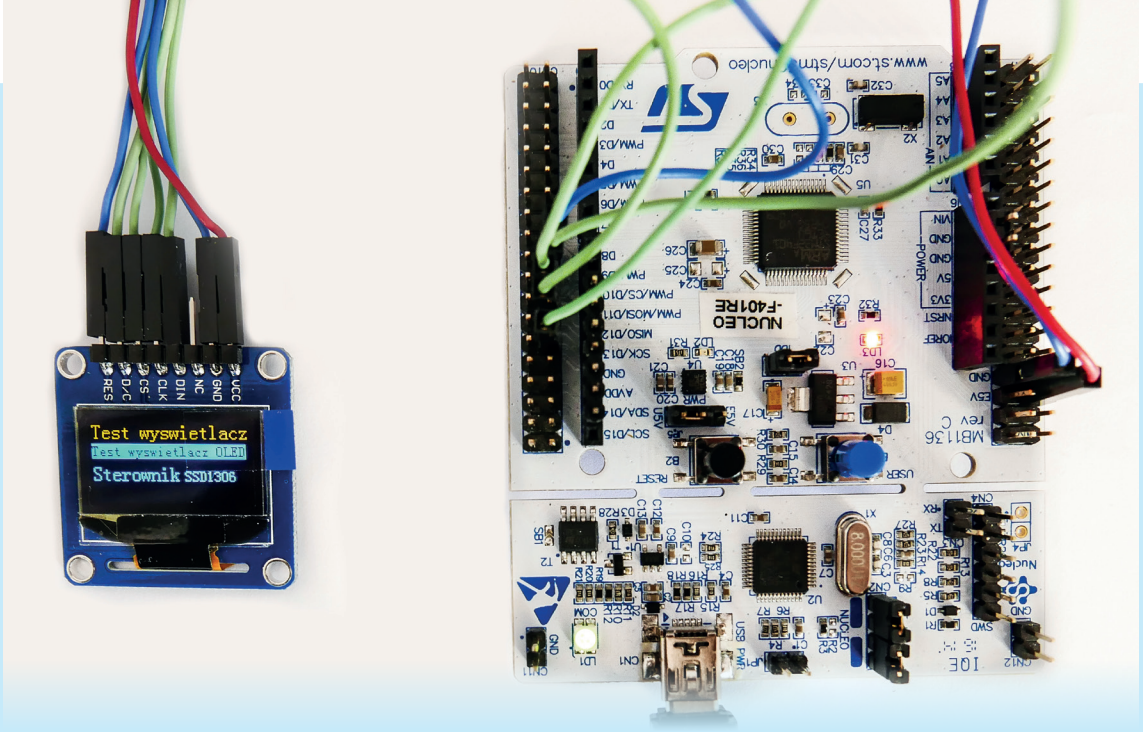


Obsługa wyświetlacza ze sterownikiem SSD1306

Niewielkie wyświetlacze z matrycami OLED cieszą się dużą popularnością wśród konstruktorów projektujących interfejsy użytkownika. Ich zaletami są: bardzo dobra czytelność w różnych warunkach oświetlenia, brak konieczności podświetlania tła, mały pobór prądu oraz stosunkowo niska cena. Niewielkie wymiary mogą być zaletą w aplikacjach wymagających małych w obudów. Jednym z takich elementów jest opisywany wyświetlacz z dwukolorową matrycą OLED o przekątnej 0,96 cala. Jego matryca ma rozdzielczość 128×64 piksele i jest sterowana przez specjalizowany sterownik SSD1306 produkowany przez firmę Solomon. 16 linii umieszczonych na górze ma kolor żółty, a pozostałe są niebieskie.

Podstawowe właściwości sterownika zostały pokazane w **tabeli 1**, a schemat blokowy sterownika pokazano na **rysunku 1**. Układ SSD1306 ma możliwość komunikowania się z systemem nadrzędnym za pomocą magistrali równoległej w standardzie Intel 8080 lub Motorola 6800 oraz za pomocą interfejsu szeregowego SPI lub I²C. Dla aplikacji, w których jest ważna prędkość transmisji danych jest wybierana któraś z magistrali równoległych, a w pozostałych wypadkach – magistrale szeregowe. Dla wielu typowych aplikacji jest wystarczająca prędkość transmisji dostępna za pomocą interfejsu szeregowego. Aktywny interfejs jest wybierany sprzętowo przez wymuszenie poziomów logicznych na wejściach konfiguracyjnych BS0…BS2 (**tabela 2**).

Poziom na wejściu BS2 określa czy będzie użyty interfejs szeregowy (BS2=0), czy równoległy (BS2=1). W tym wyświetlaczu BS2 jest na stałe dołączony do masy i mamy możliwość wyboru tylko jednego z trzech interfejsów szeregowych. Na płytce drukowanej wyświetlacza są umieszczone dwa pola lutownicze umożliwiające ustawianie poziomów na BS0 i BS1 (**fotografia 2**). Fabrycznie są tam wymuszone poziomy BS0=0 i BS1=0, czyli 4-przewodowy interfejs SPI. Zgodnie z tym ustawieniem są opisane wyprowadzenia płytki wyświetlacza. Oczywiście, można zmienić interfejs, ale nie widziałem powodu, aby to robić w układzie testowym, więc pozostawiłem ustawienia fabryczne

Ponieważ nie przewidziano odczytywania jakichkolwiek danych ze sterownika, to SPI ma tylko linię danych DIN (MOSI z punktu widzenia hosta) oraz linię zegarową CLK. Oprócz tych linii, do przesyłania danych jest używana linia CS i D/C.

Wejście CS jest standardowym sygnałem SPI uaktywniającym układ, do którego są wysyłane dane. Wykorzystując CS można do jednej magistrali dołączyć kilka wyświetlaczy – każdy wybierany za pomocą odrębnej linii CS. Linia D/C jest wykorzystywana do kierowania danych albo do rejestru komend (D/C=0), albo do pamięci obrazu (D/C=1). Przy 3-przewodowym interfejsie SPI brak jest sygnału D/C i jest konieczne przesłanie dodatkowego bitu informującego sterownik, gdzie maja trafić dane z magistrali. Komplikuje to obsługę transmisji, bo dane są 9-bitowe i nie można wykorzystać standardowych, sprzętowych interfejsów SPI wysyłających słowa 1-bajtowe. Używając interfejs I²C trzeba dodatkowo wysłać po adresie *slave* bajt kontrolny. Z jednej

strony dodatkowa linia D/C wymaga linii portu sterownika, a z drugiej strony jej użycie przyśpiesza i upraszcza transfer danych. Przebiegi czasowe na liniach magistrali 4-przewodowej pokazano na **rysunku 3**.

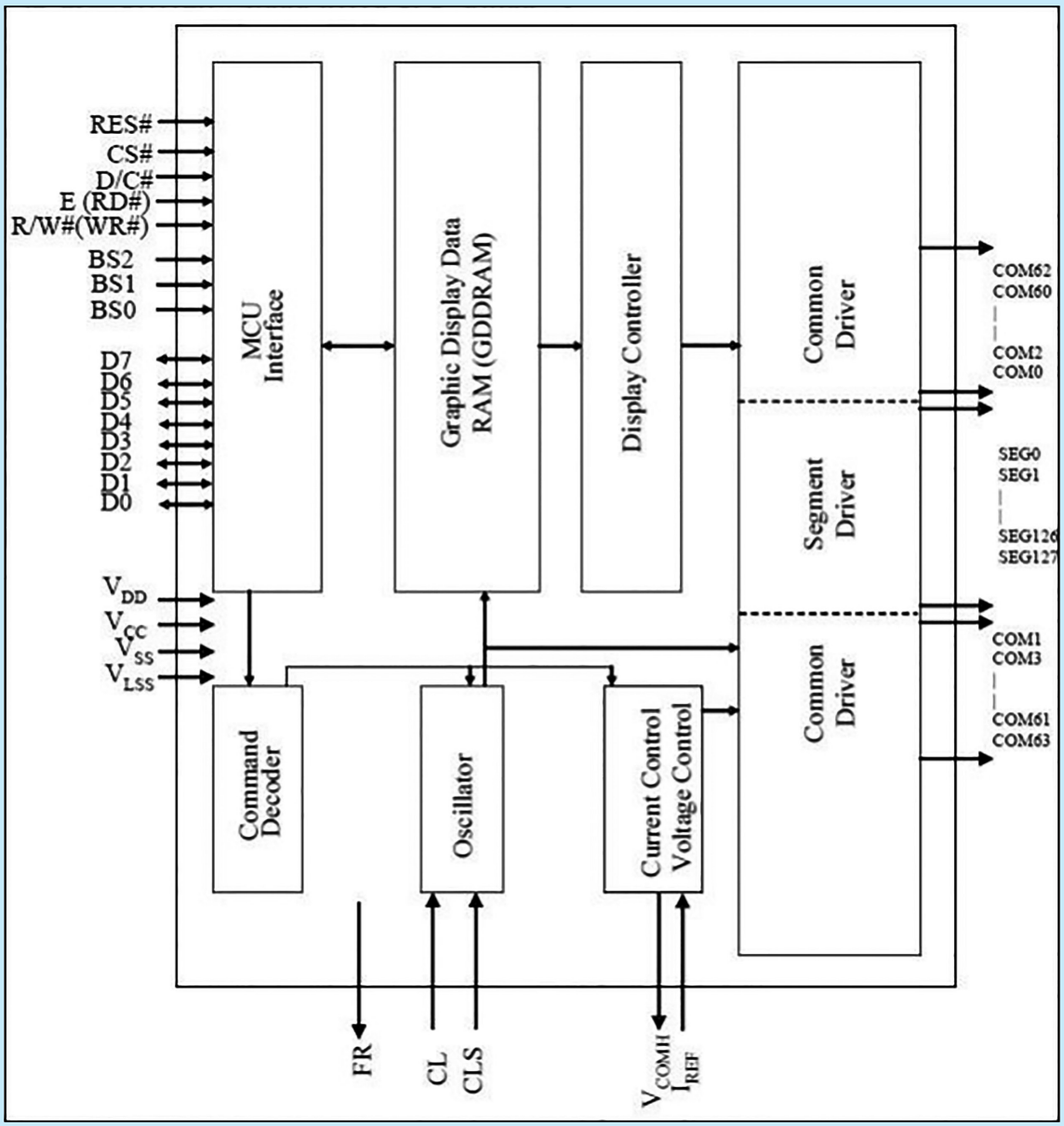
Tabela 1. Podstawowe właściwości sterownika SSD1306

Rozdzielczość	Matryca 128×64 piksele
Napięcie zasilania	Układy logiczne +1,65...+3,3 V
	Drivery panelu matrycy +7...+15 V
Maksymalny prąd segmentu	100 µA
Maksymalny całkowity prąd matrycy	15 mA
Kontrast matrycy	256 poziomów (programowany)
Wbudowana pamięć obrazu	SRAM 128×64 bity
Interfejsy komunikacyjne	Równoległy, 8-bitowy 8080/6800
	Szeregowy SPI 3- lub 4-liniowy
	Szeregowy I²C
Funkcje akceleratora graficznego	Ciągłe skrolowanie w poziomie
	Ciągłe skrolowanie w pionie
Remapowanie	Kolumn i wierszy
Oscylator	Wbudowany w układ
Zakres temperatury pracy	-40...+85°C

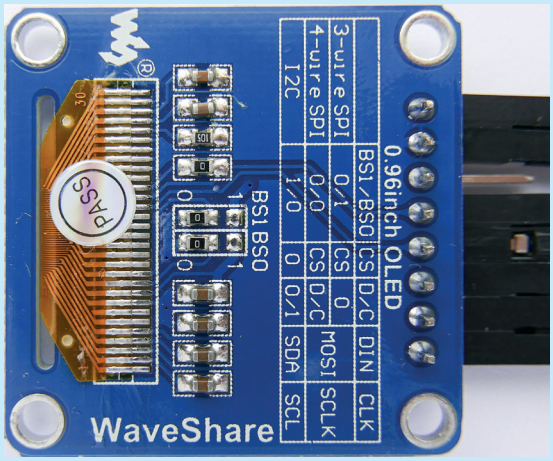
Pamięć obrazu GDDRAM (Graphic Dispaly Data)

Pamięć obrazu jest statyczną pamięcią RAM o organizacji 128×64 bity. Każdemu bitowi pamięci odpowiada pojedynczy piksel na matrycy OLED. Pamięć o takiej organizacji nie może być bezpośrednio zapisywana przez mikrokontroler, dlatego została logicznie podzielona na 8 stron (Page0….Page7, **rysunek 4**). Jedna strona zawiera 128 bajtów. Zapisanie bajtu do pamięci odpowiada pionowej linijce o długości 8 pikseli (**rysunek 5**). Aby zapewnić elastyczność przy mechanicznym mocowaniu wyświetlacza jest możliwe remapowanie segmentów: są one wybierane licząc od lewej do prawej, od 127 do 0. Podobnie są remapowane linie matrycy. Remapowaniem sterują dwie komendy: *Set Segment Re-map (A0/A1)* i *Set COM Output Scan Direction (C0/C8)*.

Zapisywanie danych do pamięci RAM jest adresowane przez dwa liczniki adresowe: licznik kolumn zmieniający się w zakresie 0…127 i licznik stron zmieniający się w zakresie 0…7. Dostęp do pamięci RAM powoduje automatyczne zwiększenie licznika kolumn i/lub licznika stron. Zakres zmiany adresów i sposób



Rysunek 1. Schemat blokowy sterownika SSD1306



Fotografia 2. Płytka z ustawieniami interfejsu

inkrementacji jest określany przez tryb adresowania ustawiany komendą Set Memory Addressing (20h).

Pierwszym z tych trybów jest **Page Addresing Mode**. Zasadę adresowania w tym trybie zilustrowano na **rysunku 6**. Początkowy adres w pamięci jest ustalany przez komendy:

- **Set Lower Column Start Address for Page Addressing Mode** (00…0x0F). Ta komenda zapisuje 4 młodsze bity licznika startowego adresu w pamięci RAM dla trybu adresowania stron.
- **Set Higer Column Start Address for Page Addressing Mode** (0x10…0x1F). Ta komenda zapisuje 4 starsze bity licznika startowego adresu w pamięci RAM dla trybu adresowania stron.
- **Set Page Address for Page Addressing Mode** (0xB0…0xB7). Ta komenda ustala licznik adresu stron w pamięci RAM dla trybu adresowania stron.

Po zapisaniu liczników każdy dostęp do pamięci RAM powoduje inkrementowanie licznika kolumn. Jeżeli licznik osiągnie wartość 127, to następny dostęp (zapis) do pamięci RAM powoduje wyzerowanie licznika kolumn, a licznik stron pozostanie bez zmian. Żeby go zmienić trzeba użyć komendy *Set Page Address for Page Addressing Mode*.

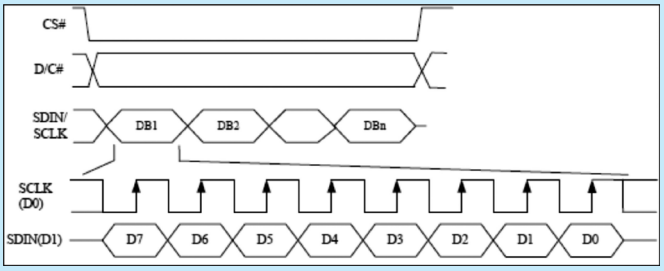
Kolejnym trybem jest tryb adresowania horyzontalnego **Horizontal Addressing Mode** pokazany na **rysunku 7**. Po każdym zapisaniu danej do pamięci RAM jest zwiększany licznik kolumn. Jeżeli licznik osiągnie wartość końcową, to jest ustawiany na wartość początkową i jednocześnie licznik stron jest automatycznie inkrementowany. Jeżeli licznik stron i licznik kolumn osiągną wartość końcową, to są one automatycznie ustawiane na wartość początkową.

Ostatni tryb, to tryb adresowania wertykalnego **Vertical Addresing Mode** (**rysunek 8**). Po każdym zapisaniu danej do pamięci RAM jest inkrementowany licznik stron. Jeżeli licznik osiągnie wartość końcową, to jest ustawiany na wartość początkową i jednocześnie jest automatycznie inkrementowany licznik kolumn. Jeżeli licznik kolumn i licznik stron osiągną wartość końcową, to są one automatycznie ustawiane na wartość początkową.

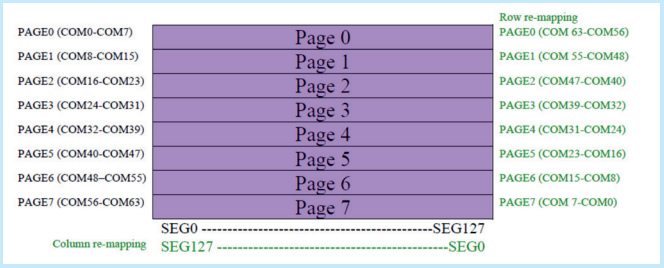
Wartości początkowe i końcowe liczników kolumn i stron nie muszą mieć wartości maksymalnych adresujących całą pamięć wyświetlacza. Zakresy adresowania dla trybu horyzontalnego i wertykalnego są ustawiane przez dwie komendy:

Tabela 2. Ustawianie interfejsu komunikacyjnego

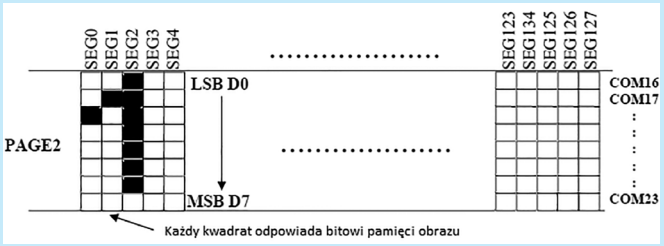
Interfejs Wejście	I²C	Motorola 6800	Intel 8080	SPI 4-wire	SPI 3-wire
BS0	0	0	0	0	1
BS1	1	0	1	0	0
BS2	0	1	1	0	0



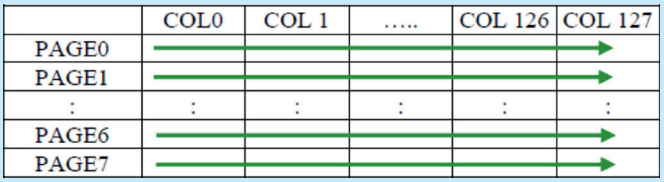
Rysunek 3. Przebiegi czasowe na 4-przewodowej magistrali SPI



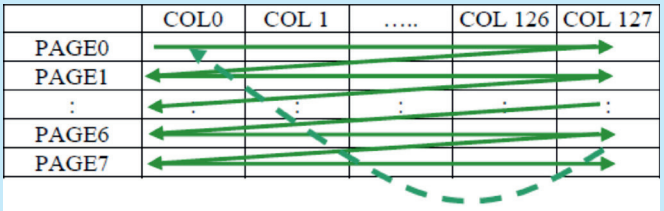
Rysunek 4. Podział pamięci obrazu na strony



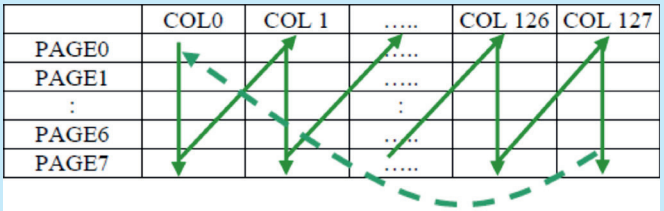
Rysunek 5. Organizacja strony pamięci RAM



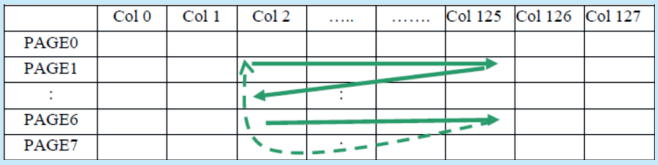
Rysunek 6. Adresowanie w trybie *Page Addressing Mode*



Rysunek 7. Tryb adresowania horyzontalnego



Rysunek 8. Tryb adresowania wertykalnego



Rysunek 9. Adresowanie horyzontalne z ograniczonym zakresem zmian adresów

Listing 1. Konfigurowanie interfejsu SPI i linii dodatkowych

```
SPI dev(SPI_MOSI, SPI_MISO, SPI_SCK);
DigitalOut CS(PB_6);
DigitalOut DC(PC_7);
DigitalOut RES(PA_9);
```

Listing 2. Wysłanie kodu komendy do sterownika

```
//***********************************************
//wysłanie danej przez SPI
//***********************************************
void WriteSpi(unsigned char data)
{
    dev.write(data);
}

//***********************************************
//zapisanie kodu komendy do sterownika SSD1306
//***********************************************
void WriteCmd(unsigned char cmd)
{
    CS=1;
    DC=0;  //DC=0 przesłanie komendy
    CS=0;
    WriteSpi(cmd);  //zapisanie kodu komendy
    CS=1;
}
```

Listing 3. Zapisanie danej do pamięci RAM

```
//***********************************************
//zapisanie danej do pamięci obrazu sterownika SSD1306
//***********************************************
void WriteData(unsigned char data)
{
    CS=1;
    DC=1;  //DC=1 przesłanie danej do pamięci RAM
    CS=0;
    WriteSpi(data);  //zapisanie danej
    CS=1;
}
```

Listing 4. Inicjalizacja sterownika wyświetlacza

```
//***********************************************
//inicjalizacja sterownika SSD1306
//***********************************************
void InitSSD1306 (void)
{
    CS=1;
    DC=0;
    RES=1;  //_reset = 1;
    wait(0.01);
    RES=0;  //_reset = 0;
    wait(0.10);
    RES=1;  //_reset = 1;
    WriteCmd(0xAE);  //wyłącz panel OLED
    WriteCmd(0x00);  //adres koluny LOW
    WriteCmd(0x10);  //adres koluny HIGH
    WriteCmd(0x40);  //adres startu linii
    WriteCmd(0x20);  //tryb adresowania strony
    WriteCmd(0x02);
    WriteCmd(0x81);  //ustaw kontrast
    WriteCmd(0xCF);
    WriteCmd(0xA1);  //ustaw remapowanie
    WriteCmd(0xC0);  //kierunek skanowania
    WriteCmd(0xA6);  //wyświetlanie bez inwersji
    WriteCmd(0xA8);  //ustaw multiplex ratio
    WriteCmd(0x3F);  //1/64
    WriteCmd(0xD3);  //ustaw display offset
    WriteCmd(0x00);  //bez offsetu
    WriteCmd(0xD5);  //ustaw divide ratio/częstotliwość
oscylatora
    WriteCmd(0x80);  //100ramek/sec
    WriteCmd(0xD9);  //ustaw okres pre charge
    WriteCmd(0xF1);  //pre charge 15 cykli, discharge 1
cykl
    WriteCmd(0xDA);  //konfiguracja wyprowadzeń
sterownika
    WriteCmd(0x12);
    WriteCmd(0xDB);  //ustawienie vcomh
    WriteCmd(0x40);
    WriteCmd(0x8D);  //ustawienie Charge Pump
    WriteCmd(0x14);
    WriteCmd(0xA4);  //"podłączenie" zawartości RAM
do panelu OLED
    WriteCmd(0xA6);  //wyłączenie inwersji wyświetlania
    WriteCmd(0xAF);  //włącza wyświetlacz
    DisplayCls(0);
}
```

- **Set Column Address** (0x21) komenda z dwoma argumentami z zakresu 0…127 określającymi adres kolumny początkowej i adres kolumny końcowej dla trybów adresowania horyzontalnego i wertykalnego.
- **Set Page Address** (0x22) komenda z dwoma argumentami z zakresu 0…7 określającymi adres strony początkowej i adres strony końcowej dla trybów adresowania horyzontalnego i wertykalnego.

Oba te tryby w połączeniu z komendami ustawiania adresów początku i końca obszaru umożliwiają łatwe wyświetlanie fontów mających wysokość większą niż 8 pikseli oraz małych bitmap (ikonek). Przykład takiego adresowania został pokazany na **rysunku 9**.

Komendy sterownika SSD1306

Przy okazji omawiania adresowania pamięci RAM obrazu opisałem kilka komend przeznaczonych do wyboru trybu adresowania i ustawiania liczników adresu. Sterownik SSD1306 ma spory zestaw komend sterujących. Dokładne opisywanie wszystkich znacznie wykracza poza zakres tego artykułu i nie jest też konieczne, bo dane te można znaleźć w dokumentacji. W **tabeli 3** umieszczono zestawienie wszystkich komend.

Programowa obsługa wyświetlacza

Moduł wyświetlacza jest skonfigurowany do pracy z 4-przewodową magistralą SPI. Jako sterownik użyłem modułu Nucleo z mikrokontrolerem z rodziny STM-32F401RE produkowanym przez firmę STMicroelectronics. Główne powody tego wyboru to wbudowany programator i wsparcie w systemie bezpłatnego środowiska projektowego mbed. Wsparcie Nucleo przez mbed zapewnia łatwą konfigurację układów peryferyjnych i pozwala skupić się na głównym problemie, czyli na obsłudze wyświetlacza.

Na początek zajmiemy się interfejsem komunikacyjnym SPI. W środowisku mbed trzeba zainicjalizować interfejs SPI podając linie, na których będą dostępne sygnały: MOSI (dane wyjściowe), MISO (dane wejściowe) i SCK (zegar taktujący przesyłaniem danych). Standardowo te sygnały są dostępne na złączu zgodnym ze standardem Arduino Uno modułu Nucleo. Jeżeli chcemy z nich skorzystać, wystarczy zadeklarować interfejs z predefiniowanymi definicjami. Interfejs SPI, oprócz linii MOSI, MISO i SCK, wymaga linii CS (standardowa linii interfejsu SPI) i linii DC określającej miejsce docelowe przesyłanych danych. Poza tym, sterownik SSD1306 musi zostać sprzętowo wyzerowany przez wyzerowanie linii RESET. Linie interfejsu SPI, łącznie z linią RESET, należy również zdefiniować. Sposób wykonania konfiguracji sprzętowej pokazano na **listingu 1**.

Do sterowania wyświetlaczem będą nam potrzebne dwie użyteczne procedury: zapisania rejestrów stepujących, czyli kodu komendy i ewentualnie jej argumentów, oraz zapisania danej do pamięci obrazu wyświetlacza – zamieszczono je na **listingach 2** i **3**.

Inicjalizowanie sterownika

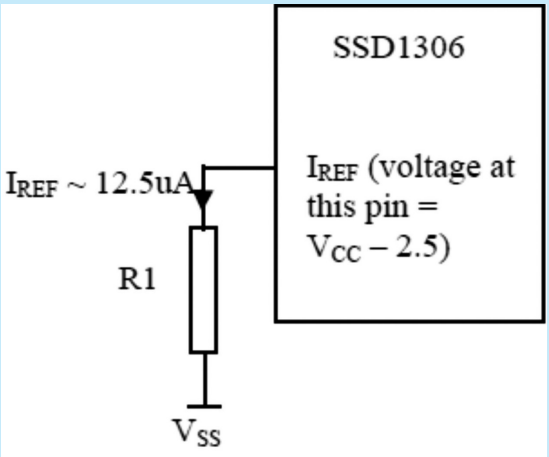
Jak większość sterowników wyświetlaczy, układ SSD1306 po włączeniu zasilania ustawia w rejestrach konfiguracyjnych wartości domyślne. To, jakie wartości

Tabela 3. Komendy sterownika SSD1306

Kod (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Komenda	Opis
81	1	0	0	0	0	0	0	1	Set Contrast Control	Ustawienie kontrastu w zakresie od 1 do 256
	A7	A6	A5	A4	A3	A2	A1	A0		
A4/A5	1	0	1	0	0	1	0	X	Entire Display ON	X=1 dane z RAM są wyświetlane na matrycy X=0 dane z RAM nie są wyświetlane
A6/A7	1	0	1	0	0	1	1	X	Set Normal/Inverse Display	X=0 – wyświetlanie normalne X=1 – wyświetlanie w negatywie
AE/AF	1	0	1	0	0	1	1	X	Set Display ON/OFF	X=0 sterownie wyświetlaczem wyłączone (SLEEP) X=1 wyświetlacz włączony
00-0F	0	0	0	0	X3	X2	X1	X0	Set Lower Column Address for Page Addr.	Młodsze cztery bity adresu kolumn
10-1F	0	0	0	1	X3	X2	X1	X0	Set Higer Column Address for Page Addr.	Starsze cztery bity adresu kolumn
20	0	0	1	0	0	0	0	0	Set Memory Addressing Mode	A[1:0]=00 Horizontal Addr. Mode A[1:0]=01 Vertical Adde. Mode A[1:0]=10 Page Addr. Mode A[1:0]==11 invalid
	*	*	*	*	*	*	A1	A0		
21	0	0	1	0	0	0	0	1	Set Column Address	A[6:0] adres początku 0...127 B[6:0] adres końca 0...127 Tylko dla trybów Vertical i Horizontal
	*	A6	A5	A4	A3	A2	A1	A0		
	*	B6	B5	B4	B3	B2	B1	B0		
22	0	0	1	0	0	0	0	1	Set Page Address	A[2:0] adres początku 0...7 B[2:0] adres końca 0...7 Tylko dla trybów Vertical i Horizontal
	*	*	*	*	*	A2	A1	A0		
	*	*	*	*	*	B2	B1	B0		
B0-B7	1	0	1	1	0	X2	X1	X0	Set Page Address for Page Addr.	X[2:0] adres strony w trybie adresowania Page Addr.
40-7F	0	1	X5	X4	X3	X2	X1	X0	Set display Start Line	Ustawienie linii początkowej – po zerowaniu równe 0
A0/A1	1	0	1	0	0	0	0	X	Set Segment Re-map	X=0 adres kolumny 0 odpowiada segmentowi SEG0 X=1 adres kolumny 0 odpowiada segmentowi SEG127
A8	1	0	1	0	1	0	0	0	Set Multiplex Ratio	A[5:0] =16MUX....64MUX A[5:0] =0...14 błędna wartość
	*	*	A5	A4	A3	A2	A1	A0		
C0/C8	1	1	0	0	X3	0	0	0	Set COM Output Scan Direction	X=0 skanowanie COM0...COM[N-1] X=1 skanowanie COM[N-1]...COM0
D3	1	1	0	1	0	0	1	1	Set Display Offset	Ustawienie przesunięcia w pionie W zakresie 0...63
	*	*	A5	A4	A3	A2	A1	A0		
DA	1	1	0	1	1	0	1	0	Set COM Pins Hardware Configurations	A4=0 konfiguracja Sequential COM A4=1 konfiguracja AlternativeCOM A5=0 blokada rempowania L/R A5=1 odblokowanie remap. L/R
	0	0	A5	A4	0	0	1	0		
D5	1	1	0	1	0	1	0	1	Set Display Clock Divide Ratio/Oscillator Frequency	A[3:0]+1= divide ratio A[7:4]=oscillator frequency
	A7	A6	A5	A4	A3	A2	A1	A0		
D9	1	1	0	1	1	0	0	1	Set Pre-charge period	A[3:0]phase1 period od 1 do 15CLK A[7:4] phase2 period od 1 do15CLK
	A7	A6	A5	A4	A3	A2	A1	A0		
DB	1	1	0	1	1	0	1	1	Set Vcomh Deselect Level	A[6:4]=00 ~0,65xVcc A[6:4]=20h ~0,77xVcc A[6:4]=30h~0,83Vcc
	0	A6	A5	A4	0	0	0	0		
26/27	0	0	1	0	0	1	1	X	Continous Horizontal Scroll Setup	X=0 przesuwanie w poziomie X=1 przesuwanie w pionie B[2:0]adres strony startowej C[2:0] interwał pomiędzy przesunięciami D[2:0] adres strony końcowej
	0	0	0	0	0	0	0	0		
	*	*	*	*	*	B2	B1	B0		
	*	*	*	*	*	C2	C1	C0		
	*	*	*	*	*	D2	D1	D0		
	0	0	0	0	0	0	0	0		
	1	1	1	1	1	1	1	1		
29/2A	0	0	1	0	1	0	X1	X0	Continous Vertical and Horizontal Scroll Setup	X[1:0]=01 przesuwanie w pionie i w prawo X[1:0]=10 przesuwanie w pionie i w lewo B[2:0]adres strony startowej C[2:0] interwał pomiędzy przesunięciami D[2:0] adres strony końcowej E[5:0] offset przesunięcia w liniach
	0	0	0	0	0	0	0	0		
	*	*	*	*	*	B2	B1	B0		
	*	*	*	*	*	C2	C1	C0		
	*	*	*	*	*	D2	D1	D0		
	*	*	E5	E4	E3	E2	E1	E0		
2E	0	0	1	0	1	1	1	0	Deactivate Scroll	Wyłączenia skrolowania
2F	0	0	1	0	1	1	1	1	Activate Scroll	Włączenia skrolowania

Tabela 3. cd.

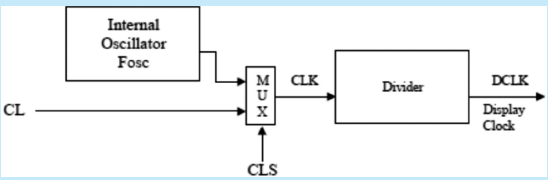
Kod (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Komenda	Opis
A3	1	0	1	0	0	0	1	1	Set Verital Scroll Area	A[5:0] ilość linii w ustalonej górnej przestrzeni B[6:0] ilość linii w skrolowanej przestrzeni
	*	*	A5	A4	A3	A2	A1	A0		
	*	B6	B5	B4	B3	B2	B1	B0		
8D	1	0	0	0	1	1	0	1	Charge Pump Setting	A2=0 pompa ładunkowa wyłączaona A2=1 pompa ładunkowa włączona
	*	*	0	1	0	A2	0	0		



Rysunek 10. Określenie napięcia zasalania matrycy

trzeba wpisać do tych rejestrów zależy od rodzaju użytej matrycy OLED i zwykle program użytkownika musi je zapisać w procesie inicjalizacji. Kompletną funkcję inicjalizacji *void InitSSD1306()* pokazano na **listingu 4**. Najpierw jest przeprowadzana sekwencja sprzętowa generująca impuls zerujący o długości 10 ms, ustawiająca bit CS i zerująca DC. Właściwa programowa inicjalizacja zaczyna się od wysłania komendy AE wyłączającej sterowanie matrycą OLED. Potem są inicjowane (zerowane) liczniki adresowe kolumn i stron oraz jest wprowadzany tryb adresowania *Page Addressing Mode*. Zakładamy, że wyświetlacz będzie pracował w położeniu normalnym to znaczy, że dół wyświetlacza będzie przy krawędzi płytki z wyprowadzeniami. Do tej orientacji mechanicznej jest ustawiane remapowanie, kierunek skanowania i konfiguracja wyprowadzeń sterownika.

Sterownik może współpracować z różnymi matrycami, więc ma możliwość ustawienia wielu parametrów, takich jak: *divide ratio*, częstotliwość odświeżania oraz napięcie zasilające (prąd segmentów) matrycy OLED. Z tymi ustawieniami jest bardzo często problem. Albo mamy wyświetlacz, o którym wiemy tylko, jaki ma sterownik albo w danych katalogowych nie są podane istotne informacje tego typu. Pozostaje inicjowanie pracy układu sterującego metodą prób i błędów. W opisywanym wyświetlaczu istnieje coś w rodzaju dokumentacji technicznej, ale niestety, opisuje ona głównie komunikację ze sterownikiem, a te informacje są doskonale opisane w dokumentacji sterownika SSD1306.



Rysunek 11. Taktowanie sterownika

Zacznijmy inicjowanie sterownika od ustawienia kontrastu. Prąd segmentu OLED można wyliczyć z równania $I_{SEG}=Contrast/256\times I_{REF}\times$scale factor, w którym *Contrast* to wartość argumentu komendy 0x81 zmieniająca się w zakresie 0…255, a *Scale Factor* jest wartością stałą, równą 8.

Aby móc zasilić matrycę OLED napięciem wyższym od napięcia zasilającego układy cyfrowe, w układ SSD1306 wbudowano przetwornicę podwyższająca napięcie działającą na zasadzie pompy ładunkowej. Do działania takiego układu potrzebne są tylko dwa kondensatory zewnętrzne dołączone do wyprowadzeń C1N, C1P i C2N oraz C2P. Wartość napięcia wyjściowego określa się za pomocą rezystora zewnętrznego dołączonego do wyprowadzenia IREF. Napięcie na tym wyjściu jest równe *Vcc-2,5*, gdzie *Vcc* jest napięciem zasilającym matrycę (**rysunek 10**). W opisywanym wyświetlaczu rezystor R1 ma wartość 1 MΩ i powinien wymuszać prąd o wartości ok 12,5 μA ±2 μA. Powoduje on spadek na rezystorze R1 równy 12,5 V, zatem matryca jest zasilana napięciem 12,5 V+2,5 V=15 V. Po włączeniu zasilania sterownika przetwornica jest domyślnie wyłączona. Dlatego w procedurze inicjalizacji należy ją włączyć wysyłając komendę 8D z argumentem 0x14 (A2=1). Po wyliczeniu prądu jednego segmentu można optymalnie dobrać wartość kontrastu dla matrycy. Ja wpisałem 0xDF, ale eksperymentowałem również z innymi wartościami.

Sterownik SSD1306 ma wbudowany układ taktowania zbudowany z wewnętrznego oscylatora RC i programowanego dzielnika (**rysunek 11**). Do taktowania można tez użyć zewnętrznego zegara podawanego na wejście CL. Stopnień podziału częstotliwości programuje się w zakresie od 1 do 16 za pomocą komendy **D5 Set Display Clock Divide Ratio/Oscillator Frequency**. Najmłodsze 4 bity zawierają wartość D, a $DCLK=F_{OSC}/D+1$. Jak można zobaczyć w procedurze inicjalizacji argument komendy D5 ma wartość 0x80, czyli D+1=1. Sterownik jest taktowany częstotliwością równą częstotliwości F_{OSC}. Cztery starsze bity argumentu zawierają współczynnik K określający ile cykli zegara jest przeznaczonych na wyświetlenie jednego rzędu (linijki). Częstotliwość, z którą są wyświetlane ramki można wyliczyć z równania

$$F_{FRM}=\frac{F_{osc}}{D\times K\times No.of\ Mux}$$

Mamy D=1, K=8, współczynnik multipleksowania *Mux* jest ustawiany komendą A8 i wynosi 64. Zatem $F_{FRM}=F_{OSC}/(8\times64)=F_{OSC}/512$. Częstotliwość oscylatora to około 370 kHz, a F_{FRM} wynosi ok. 720 Hz.

Większość ustawień inicjalizacji została dobrana na podstawie informacji zawartych w dokumentacji sterownika. Ostatnią czynnością inicjalizacji jest wyczyszczenie (wyzerowanie) pamięci obrazu RAM, aby

po inicjalizacji na ekranie nie wyświetlały się przypadkowe wartości. Funkcję czyszczenia wyświetlacza pokazano na **listingu 5**. Działa ona poprawnie dla trybu adresowania *Page Addressing Mode* ustawionego w czasie inicjalizacji sterownika. Funkcja *void DisplayCls()* najpierw zeruje bufor w pamięci mikrokontrolera (dwuwymiarowa tablica), a potem zawartość tego bufora przepisuje do pamięci obrazu RAM sterownika – funkcja *RefreshRAM()*. Na **listingu 6** pokazano procedurę *void SetColStart()* ustawiająca adres kolumny na początek (0x00).

Praca w trybie tekstowym

Wyświetlacz graficzny doskonale nadaje się do wyświetlania tekstu. W typowych wyświetlaczach alfanumerycznych czcionki mają jednakową wielkość zależną od wielkości wyświetlacza i mogą być wyświetlane w ustalonych wierszach. W wyświetlaczu graficznym można definiować czcionki o różnych rozmiarach (zależnie od potrzeb) i umieszczać je w dowolnym miejscu wyświetlacza, o ile tylko tam się zmieszczą.

Przy adresowaniu *Page Addressing Mode* jest łatwo zdefiniować znaki o wysokości 8 pikseli lub wielokrotności 8. Najłatwiej jest definiować znaki np. 8×6 pikseli umieszczając w pamięci generatora znaków 6 kolejnych bajtów – łatwo to sobie wyobrazić patrząc na **rysunek 7**. W programowych bibliotekach obsługujących wyświetlacze graficzne stosowałem z powodzeniem tę metodę. Mam zdefiniowaną tablice generatora znaków i funkcje służące do wyświetlania znaku na podstawie jego kodu ASCII. Niestety w tym wyświetlaczu znaki o wysokości 8 pikseli są nieczytelne. Wynika to po prostu z małych wymiarów – w tym wyświetlaczu wartością graniczną jest wysokość 12 pikseli, a najlepiej, gdyby znaki miały 16 lub więcej pikseli.

Do rysowania znaków o dowolnej wielkości najlepiej nadaje się funkcja, która potrafi zaświecić lub zgasić pojedynczy piksel o dowolnych współrzędnych *x*, *y* niezależnych od trybu adresowania. Taką funkcję pokazano na **listingu 7**. Jak łatwo zauważyć, funkcja *void DrawPoint()* z list. 7 modyfikuje zawartość dwuwymiarowego bufora *DispBuff* umieszczonego w pamięci RAM mikrokontrolera hosta i nie zapisuje modyfikacji do sterownika SSD1306. Jak zobaczymy dalej, wszystkie procedury wyświetlania modyfikują tylko ten bufor i aby zobaczyć efekt tych modyfikacji trzeba przepisać całą zawartość *DispBuff* do pamięci sterownika wywołując funkcję *void RefreshRAM()* pokazaną na list. 5.

Mając do dyspozycji procedurę umożliwiającą zaświecenie/zgaszenie piksela o konkretnej współrzędnej możemy rysować proste, figury, okręgi, ale też rysować znaki alfanumeryczne o dowolnej wielkości. Żeby to robić, potrzebne są wzorce znaków umieszczone w tablicy – generatorze znaków. Ręczne tworzenie wzorców znaków jest możliwe, ale to żmudne zajęcie. Istnieje szereg programów tworzących wzorce o zadanej wielkości znaku. W Internecie można też znaleźć gotowe tablice z wzorcami. Ja skorzystałem z gotowych tablic ze zdefiniowanymi znakami o wielkości 12×6 pikseli i 16×8 pikseli. Tablice są tak zbudowane, że kod ASCII znaku adresuje grupę bajtów definiujących ten znak. To znacznie upraszcza procedury wyświetlające łańcuchy znaków (napisy). Na przykład, tablica dla znaków o wysokości 12 i szerokości 6 pikseli jest dwuwymiarową tablicą *const char c_chFont1206[95][12]* i zawiera 95 wzorców znaków. Każdy element wzorca znaków ma 12 bajtów i definiuje znak o szerokości 6 pikseli, ale wykorzystuje przestrzeń 12×8 pikseli (**listing 8**). Dla znaków 16×8 pikseli jest zdefiniowana druga tablica *const uint8_t c_chFont1608[95][16]*.

Mając tablicę generatora znaków i procedurę *DrawPoint()* potrafiącą „zapalać i gasić" bit w buforze pamięci mikrokontrolera odpowiadający zawartości pamięci RAM sterownika, a tym samym odpowiadający pikselowi na matrycy OLED można napisać procedurę „rysującą" znak w pamięci RAM mikrokontrolera. Pamiętamy, że aby ten znak się wyświetlił, trzeba przepisać zawartość *DispBuff* do pamięci RAM wywołując funkcję *void RefreshRAM()*. Na **listingu 9** pokazano procedurę *void DisplayChar()* z argumentami:

- **X** i **Y** – współrzędne początku znaku na ekranie.
- **Chr** – kod ASCII wyświetlanego znaku.
- **Size** – wielkość znaku 12 lub 16 pikseli.
- **Mode= 1** wyświetlanie normalne, **mode=0** wyświetlanie w negatywie.

Zależnie od wartości argumentu *size*, bajty wzorca są pobierane z tablicy *c_chFont1206[95][12]* lub z tablicy *c_chFont1608[95][16]*. Kiedy argument *mode* jest

Listing 5. Czyszczenie zawartości wyświetlacza

```
//***********************************************
//zeruj bufor wyświetlacza
//i zapisz jego zwartość do RAM obrazu
//***********************************************
void DisplayCls(unsigned char fill)
{
    unsigned char  i, j;

    for (i = 0; i < 8; i ++) {
        for (j = 0; j < 128; j ++) {
            DispBuff[j][i] = fill;
        }
    }

    RefreshRAM();//zawartość bufora do RAM obrazu
}

//***************************************************
//zapisanie zawartości bufora do RAM wyświetlacza
//***************************************************
void RefreshRAM(void)
{
    uint8_t i, j;

    for (i = 0; i < 8; i ++) {
        WriteCmd(0xB0 + i);
        SetColStart();
        for (j = 0; j < 128; j ++) {
            WriteData(DispBuff[j][i]);
        }
    }
}
```

Listing 6. Zerowanie licznika kolumn

```
void SetColStart(void){
    WriteCmd(0x00); //low
    WriteCmd(0x10); //high
}
```

Listing 7. Rysowanie punktu o ustalonych współrzędnych

```
//*****************************************************
*****
//"rysowanie" punktu w buforze RAM mikrokontrolera Hosta
//*****************************************************
*****
void DrawPoint(unsigned char x ,unsigned char y, unsigned
char p)
{
    uint8_t chPos, chBx, chTemp = 0;

    if (x > 127 || y > 63) return;
    chPos = 7 - y / 8;
    chBx = y % 8;
    chTemp = 1 << (7 - chBx);
    if (p) {
        DispBuff[x][chPos] |= chTemp;
    } else {
        DispBuff[x][chPos] &= ~chTemp;
    }
}
```

Listing 8. Fragment tablicy generatora znaków 12×6 pikseli

```
const char c_chFont1206[95][12] = {
    {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},  //" ",0
    {0x00,0x00,0x00,0x00,0x3F,0x40,0x00,0x00,0x00,0x00,0x00,0x00},  //"!",1
    {0x00,0x00,0x30,0x00,0x40,0x00,0x30,0x00,0x40,0x00,0x00,0x00},  //"",2
    {0x09,0x00,0x0B,0xC0,0x3D,0x00,0x0B,0xC0,0x3D,0x00,0x09,0x00},  //"#",3
    {0x18,0xC0,0x24,0x40,0x7F,0xE0,0x22,0x40,0x31,0x80,0x00,0x00},  //"$",4
    {0x18,0x00,0x24,0xC0,0x1B,0x00,0x0D,0x80,0x32,0x40,0x01,0x80},  //"%",5
    {0x03,0x80,0x1C,0x40,0x27,0x40,0x1C,0x80,0x07,0x40,0x00,0x40},  //"&",6
    {0x10,0x00,0x60,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},  //"'",7
    {0x00,0x00,0x00,0x00,0x00,0x00,0x1F,0x80,0x20,0x40,0x40,0x20},  //"(",8
    {0x00,0x00,0x40,0x20,0x20,0x40,0x1F,0x80,0x00,0x00,0x00,0x00},  //")",9
    {0x09,0x00,0x06,0x00,0x1F,0x80,0x06,0x00,0x09,0x00,0x00,0x00},  //"*",10
    {0x04,0x00,0x04,0x00,0x3F,0x80,0x04,0x00,0x04,0x00,0x00,0x00},  //"+",11
    {0x00,0x10,0x00,0x60,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},  //",",12
    {0x04,0x00,0x04,0x00,0x04,0x00,0x04,0x00,0x04,0x00,0x00,0x00},  //"-",13
    {0x00,0x00,0x00,0x40,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},  //".",14
    {0x00,0x20,0x01,0xC0,0x06,0x00,0x38,0x00,0x40,0x00,0x00,0x00},  //"/",15
    {0x1F,0x80,0x20,0x40,0x20,0x40,0x20,0x40,0x1F,0x80,0x00,0x00},  //"0",16
    {0x00,0x00,0x10,0x40,0x3F,0xC0,0x00,0x40,0x00,0x00,0x00,0x00},  //"1",17
    {0x18,0xC0,0x21,0x40,0x22,0x40,0x24,0x40,0x18,0x40,0x00,0x00},  //"2",18
    {0x10,0x80,0x20,0x40,0x24,0x40,0x24,0x40,0x1B,0x80,0x00,0x00},  //"3",19
    {0x02,0x00,0x0D,0x00,0x11,0x00,0x3F,0xC0,0x01,0x40,0x00,0x00},  //"4",20
    {0x3C,0x80,0x24,0x40,0x24,0x40,0x24,0x40,0x23,0x80,0x00,0x00},  //"5",21
    {0x1F,0x80,0x24,0x40,0x24,0x40,0x34,0x40,0x03,0x80,0x00,0x00},  //"6",22
    {0x30,0x00,0x20,0x00,0x27,0xC0,0x38,0x00,0x20,0x00,0x00,0x00},  //"7",23
    {0x1B,0x80,0x24,0x40,0x24,0x40,0x24,0x40,0x1B,0x80,0x00,0x00},  //"8",24
    {0x1C,0x00,0x22,0xC0,0x22,0x40,0x22,0x40,0x1F,0x80,0x00,0x00},  //"9",25
}
```

Listing 9. Wyświetlenie jednego znaku

```
//*************************************************************
//wyświetlenie jednego znaku, argumenty:
//x,y - współrzędne na ekranie
//Chr - kod ASCII znaku
//size - rozmiar 12, lub 16
//mode=1 - tryb normalny, mode=0 znak wyświetlany w negatywie
//*************************************************************
void DisplayChar(uint8_t x, uint8_t y, uint8_t Chr, uint8_t size, uint8_t mode)
{
    uint8_t i, j;
    uint8_t chTemp, chYpos0 = y;

    Chr =Chr - ' ';
    for (i = 0; i < size; i ++) {
        if (size == 12) {
            if (mode) {
                chTemp = c_chFont1206[Chr][i];
            } else {
                chTemp = ~c_chFont1206[Chr][i];
            }
        } else {
            if (mode) {
                chTemp = c_chFont1608[Chr][i];
            } else {
                chTemp = ~c_chFont1608[Chr][i];
            }
        }
        for (j = 0; j < 8; j ++) {
            if (chTemp & 0x80) {
                DrawPoint(x, y, 1);
            } else {
                DrawPoint(x, y, 0);
            }
            chTemp <<= 1;
            y ++;
            if ((y - chYpos0) == size) {
                y = chYpos0;
                x ++;
                break;
            }
        }
    }
}
```

Listing 10. Procedura wyświetlania komunikatów

```
//*************************************************************************
//wyświetlenie łańcucha znaków - napisu
//argumenty
//x,y - współrzędne na ekranie
//*txt - wskaźnik na początek bufora zwierającego łańcuch znaków  ASCII do wyświetlenia
//size - wysokość znaków 12, lub 16 pikseli
//mode=1 znaki wyświetlane normalnie, mode=0 znaki wyświetlane w negatywie
//*************************************************************************
void DispTxt(uint8_t x, uint8_t y, const uint8_t *txt, uint8_t size, uint8_t mode)
{
    while (*txt != '\0') {
        if (x > (SSD1306_WIDTH - size / 2)) {
            x = 0;
            y += size;
            if (y > (SSD1306_HEIGHT - size)) {
                y = x = 0;
                DisplayCls(0x00);
            }
        }
        DisplayChar(x, y, *txt, size, mode);
        x += size / 2;
        txt ++;
    }
}
```

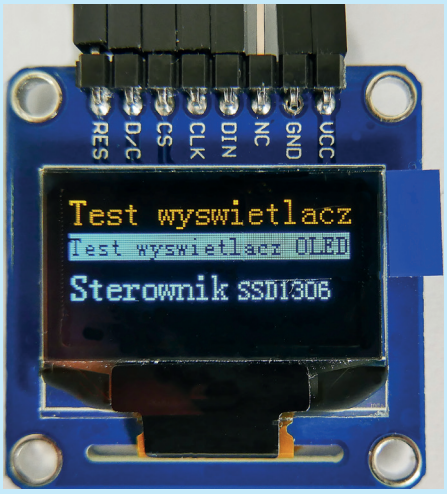
Listing 11. Wyświetlanie w trybie tekstowym

```
int main (void)
{
    InitSSD1306 ();
    DispTxt(0,0,"Test wyswietlacz",16,1);  //znaki 16pikseli - obszar żółty
    DispTxt(0,16,"Test wyswietlacz OLED",12,0);  //znaki 12 pikseli - negatyw
    DispTxt(0,32,"Sterownik „,16,1);  //znaki 16 pikseli
    DispTxt(76,36,"SSD1306",12,1);  //znaki 12 pikseli
    RefreshRAM();  //przepisz do sterownika SSD1306
    while(1);
}
```

Listing 12. Wyświetlenie bitmapy

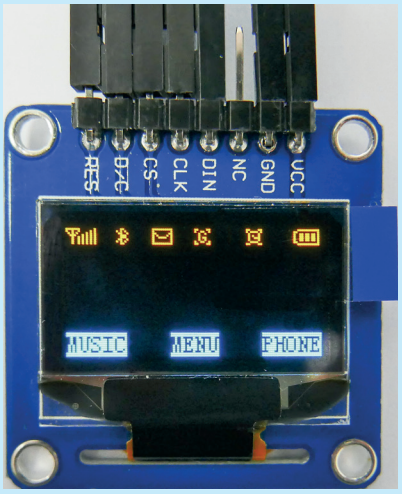
```
//*************************************************
//rysuj bitmapę
//argumenty
//x,y współrzędne początku bitmapy
//*bmp wskaźnik na tablicę z przekonwertowaną  bitmapą
//width  - szerokość bitmapy
//height - wysokość bitmapy
//*************************************************
void DrawBmp(uint8_t x, uint8_t y, const uint8_t *bmp, uint8_t width, uint8_t height)
{
    uint16_t i, j, byteWidth = (width + 7) / 8;

    for(j = 0; j < height; j ++){
        for(i = 0; i < width; i ++ ) {
            if(*(bmp + j * byteWidth + i / 8) & (128 >> (i & 7))) {
                DrawPoint(x + i, y + j, 1);
            }
        }
    }
}
```



Fotografia 12. Wyświetlanie testowych napisów o wysokości 16 i 12 pikseli

Fotografia 13. Pełnowymiarowa bitmapa 128×64 piksele



Fotografia 14. Przykład wyświetlenia ikonek

wyzerowany, to dodatkowo wartość pobranego bajtu jest negowana. Potem jest analizowany każdy bit bajtu wzorca i zależnie od jego wartości *DrawPoint()* zapisuje do bufora *DispBuff* odpowiednią wartość.

Mając procedurę wyświetlania jednego znaku o dowolnych współrzędnych można napisać procedurę wyświetlającą łańcuch znaków od określonej pozycji. Na **listingu 10** pokazano taką procedurę – *void DispTxt()* przyjmującą następujące argumenty:

- **X** i **Y** – współrzędne początku znaku na ekranie.
- ***txt** – wskaźnik na początek bufora z łańcuchem znaków.
- **Size** – wielkość znaku 12, lub 16 pikseli.
- **Mode=1** wyświetlanie normalne, **mode=0** wyświetlanie w negatywie.

Po każdym wyświetleniu jednego znaku na podstawie informacji o maksymalnych współrzędnych ekranu wyświetlacza (*SSD1306_WIDTH i SSD1306_HEIGHT*) oraz wartości argumentu *size* są wyliczane współrzędne kolejnego znaku. Jeżeli kolejny znak nie zmieści się w całości w linijce, to tekst jest automatycznie przenoszony na początek kolejnej linijki (CR, LF). Efekt działania krótkiego fragmentu programu z **listingu 11** jest pokazany na **rysunku 12**.

Wyświetlanie bitmap

Każda czcionka wyświetlana w trybie tekstowym jest bitmapą rysowaną na ekranie. Te bitmapy – wzorce znaków umieszczane są w tablicy generatora znaków i mają jednakową wielkość. Jednak często zachodzi konieczność wyświetlania bitmap o różnych rozmiarach. Pewnym problemem jest konwertowanie monochromatycznych – ja używam programu *bmp2.exe* służącego do konwertowania bitmap monochromatycznych o niewielkich rozmiarach na tablicę w języku C. Kiedy już mamy skonwertowana bitmapę, to procedura jej wyświetlania jest stosunkowo prosta – pokazano ją na **fotografii 12**. Na **fotografii 13** pokazano pełnowymiarową bitmapę 128×64 piksele, a na **fotografii 14** kilka małych bitmap – ikon wyświetlanych w żółtym obszarze matrycy.

Tomasz Jabłoński, EP