Mały wyświetlacz OLED

Wyświetlacze OLED w porównaniu z wyświetlaczami LCD mają kilka zalet. Po pierwsze, nie wymagają podświetlania, przez co są mniej energochłonne. Po drugie, mają bardzo dobry kontrast i dużą jasność świecenia. Producenci też uporali się z ograniczoną trwałością tych elementów. Z tych powodów panele OLED są coraz chętniej stosowane w interfejsach użytkownika. Opisywany tu wyświetlacz ma matrycę o przekątnej 1,3 cala (aktywna część wyświetlacza 29,4 mm × 14,7 mm) i rozdzielczości 128×64 piksele.

Zazwyczaj wyświetlacze są wykonane jako płytka szklana z przyklejonym do niej płaskim złączem taśmowym. Z punktu widzenia konstruktora chcącego zastosować taki element w produkcji małoseryjnej nie jest to wygodne, ponieważ wymaga wykonania mocowania szklanego panelu, zastosowania złącza dla taśmy itp. To również praktycznie dyskwalifikuje taki wyświetlacz, jeżeli chcemy go po prostu przetestować.

Wyświetlacz opisywany w artykule nie ma tych wad, bo został przymocowany na stałe do płytki drukowanej, na której umieszczono wszystkie niezbędne elementy dodatkowe: stabilizator LDO napięcia +3,3 V, goldpiny do podłączenia sygnałów magistrali przeznaczonej mikrokontrolera–hosta, pola lutownicze do przylutowania taśmy wyświetlacza (**fotografia 1**).

Sterownik SH1106

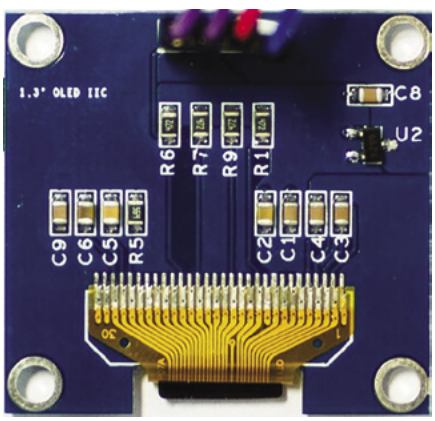
Do sterowania matrycą i komunikacji z hostem zastosowano sterownik SH1106 (**rysunek 2**). Jest to specjalizowany układ przeznaczony do sterowania matryc OLED/PLED o rozdzielczości maksymalnej 132×64 piksele. Wbudowany driver steruje 132 segmentami i 64 kolumnami matrycy typu *Common Cathode*. Sterownik ma tryb oszczędzania energii *Sleep Mode* z poborem prądu nieprzekraczającym 5 µA. Nie bez znaczenia jest też szeroki zakres temperatury pracy – od -40°C do +85°C.

Magistrala sterująca

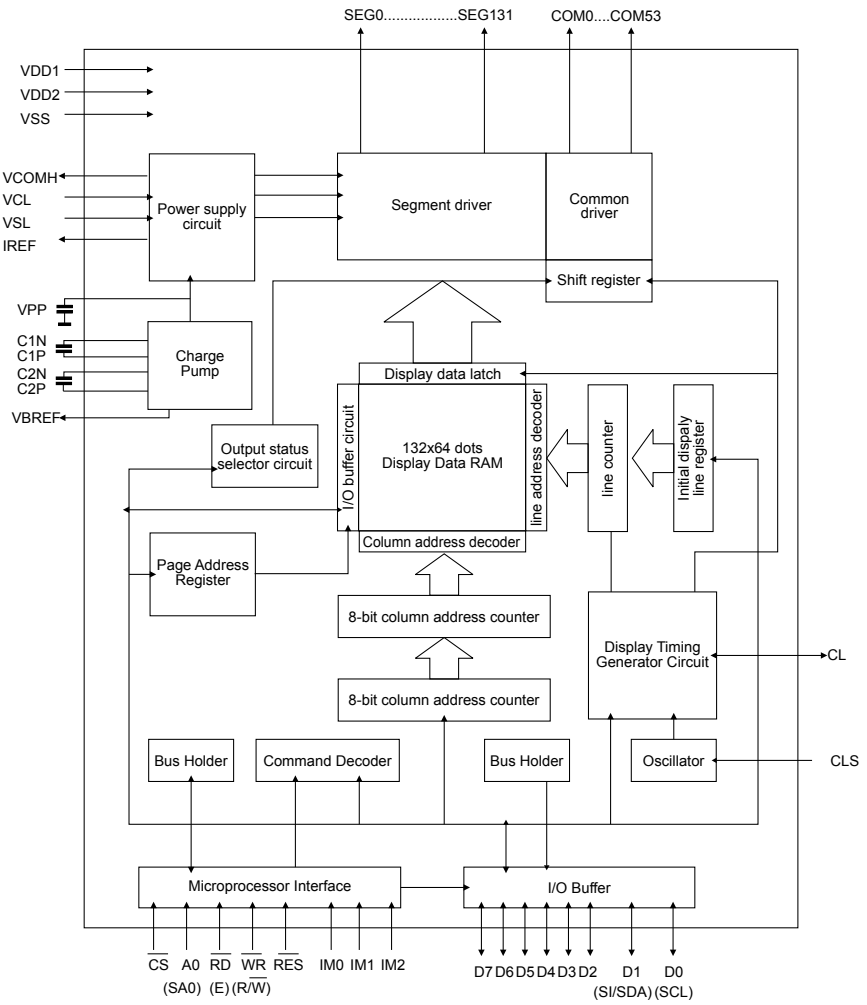
Sterownik komunikuje się z hostem za pomocą magistrali. Jej budowa jest determinowana przez ilość przesyłanych danych. Dla wyświetlaczy kolorowych z głębią 24-bitową ilość przesyłanych danych jest stosunkowo duża. Wtedy najlepszym rozwiązaniem jest zastosowanie magistrali równoległej, 8- lub 16- bitowej charakteryzującej się dużą przepływnością danych. W małych wyświetlaczach monochromatycznych ilość przesyłanych danych jest stosunkowo nieduża i dlatego można stosować magistrale szeregowe, zajmujące mniej miejsca i wymagające mniejszej liczby portów procesora–hosta. Sterownik SH1106 ma 8-bitową magistralę równoległą mogącą pracować w trybach zgodnych z Intel 8080 lub Motorola 6800 oraz 2 interfejsy szeregowe: SPI i I²C. Wyboru interfejsu aktywnego dokonuje się przez wymuszenie odpowiednich poziomów logicznych na wejściach konfiguracyjnych IM0…IM2.

Często zdarza się, że producent wyświetlacza nie wyprowadza wejść konfiguracyjnych i magistrala jest ustawiona na stałe, bez możliwości zamiany przez użytkownika. Tak jest i w tym przypadku. Uznano, że wyświetlacz będzie sterowany za pomocą I²C i innej możliwości nie ma.

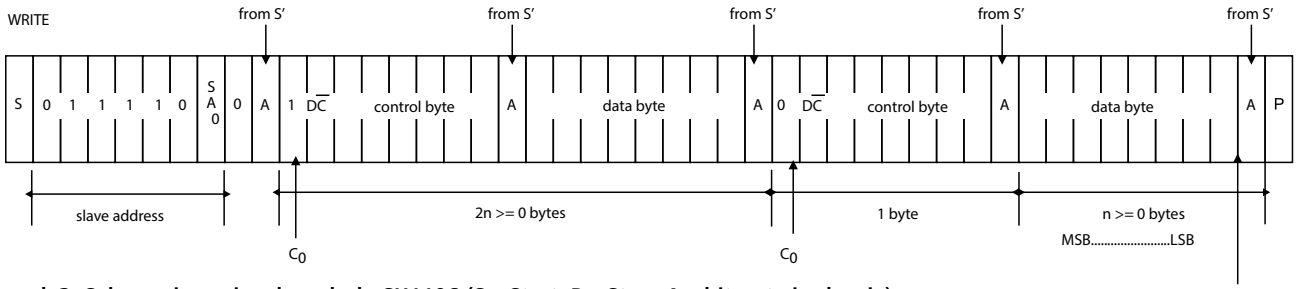
Pod sposobu wykonania połączeń magistrala I²C jest jedną z mniej skomplikowanych magistral szeregowych, ponieważ wymaga tylko 2 linii (SDA i SCL) oraz 2 rezystorów pod-



Fotografia 1. Płytka bazowa wyświetlacza



Rysunek 2. Schemat blokowy sterownika SH1106



Rysunek 3. Sekwencja zapisu danych do SH1106 (S – Start, P – Stop, A – bit potwierdzenia)

ciągających te linie do plusa zasilania. Prosty interfejs jest za to bardziej skomplikowany w obsłudze programowej w porównaniu np. z SPI lub interfejsem równoległym. Sterownik jest układem Slave mającym swój własny adres, a nadrzędny system sterujący jest układem Master.

Każdy transfer danych na magistrali I²C rozpoczyna się od wysłania sekwencji *Start*, a następnie bajta z adresem Slave (7 starszych bitów) i najmłodszego bitu R/W. Adres Slave ma wartość 0111100b, czyli kompletny bajt zapisu danych do SH1106 to 0x78 (R/W=0) i 0x79 dla odczytu danych (R/W=1). Po przesłaniu adresu protokół transmisji SH1106 wymaga przesłania bajta kontrolnego. Mają w nim znaczenie 2 najstarsze bity: *Co* i *D/C*. Pozostałych 6 młodszych bitów musi być zerami. Jeżeli bit *Co* jest wyzerowany, to następne bajty są bajtami danych. Po wpisaniu do *Co* jedynki sterownik interpretuje dwa następne bajty jako dane i kolejny bajt jako bajt kontrolny. Wyzerowania bitu R/W oznacza, że następny bajt jest komendą. Dla R/W=1 kolejne bajty są bajtami danych wpisywanych do pamięci obrazu sterownika. Sekwencję zapisu pokazano na **rysunku 3**.

STM32F100 – interfejs I²C

Do sterowania wyświetlaczem użyłem modułu *STM32F100 Discovery* z mikrokontrolerem *STM32F100RB8T6*. Mikrokontroler ma wbudowany sprzętowy interfejs I²C, który zostanie wykorzystany do komunikacji z wyświetlaczem. Na początku programu skonfigurujemy ten interfejs.

Listing 1. Definicja taktowania mikrokontrolera z częstotliwością 24 MHz

```
#if defined (STM32F10X_LD_VL) || (defined STM32F10X_MD_VL)
/* #define SYSCLK_FREQ_HSE    HSE_Value */
 #define SYSCLK_FREQ_24MHz  24000000
#else
/* #define SYSCLK_FREQ_HSE    HSE_Value */
 #define SYSCLK_FREQ_24MHz  24000000
/* #define SYSCLK_FREQ_36MHz  36000000 */
/* #define SYSCLK_FREQ_48MHz  48000000 */
/* #define SYSCLK_FREQ_56MHz  56000000 */
//#define SYSCLK_FREQ_72MHz  72000000
#endif
```

Listing 2. Konfiguracja interfejsu I2C I kanału DMA

```
    // konfiguracja portów
    GPIO_InitTypeDef  GPIO_InitStructure;
    I2C_InitTypeDef  I2C_InitStructure;
    /* GPIOB clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    /* Enable the DMA1 clock */
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
    //konfigurowanie interfejsu I2C
    /* I2C1 clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
    /* I2C1 SDA and SCL configuration */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    /* Enable I2C1 reset state */
    RCC_APB1PeriphResetCmd(RCC_APB1Periph_I2C1, ENABLE);
    /* Release I2C1 from reset state */
    RCC_APB1PeriphResetCmd(RCC_APB1Periph_I2C1, DISABLE);
//konfigurowanie DMA
    DMA_DeInit(I2C1_DMA_CHANNEL_TX);
    I2CDMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)I2C1_DR_Address;
    I2CDMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)0;    /* This parameter will be configured durig communication */
    I2CDMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;     /* This parameter will be configured durig communication */
    I2CDMA_InitStructure.DMA_BufferSize = 0xFFFF;             /* This parameter will be configured durig communication */
    I2CDMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    I2CDMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    I2CDMA_InitStructure.DMA_PeripheralDataSize = DMA_MemoryDataSize_Byte;
    I2CDMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
    I2CDMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
    I2CDMA_InitStructure.DMA_Priority = DMA_Priority_VeryHigh;
    I2CDMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
    DMA_Init(I2C1_DMA_CHANNEL_TX, &I2CDMA_InitStructure);

    /* I2C1 RX DMA Channel configuration */
    DMA_DeInit(I2C1_DMA_CHANNEL_RX);
    DMA_Init(I2C1_DMA_CHANNEL_RX, &I2CDMA_InitStructure);
}
```

Sprzętowy interfejs STM32 może pracować w trybie z wieloma masterami i obsługuje adresowanie 7 i 10 bitowe. Dostępne są standardowe prędkości transmisji (100 kHz) i szybkie prędkości (Fast Speed – 400 kHz). Do obsługi transmisji można wykorzystać mechanizm przerwań lub odpytywanie (pooling). Możliwe jest też przesyłanie danych z wykorzystaniem kanału DMA.

W naszym przypadku będziemy wykorzystywać wyłącznie tryb Master (z jednym masterem na magistrali), a dane będą wysyłane tylko przez Mastera – ze sterownika wyświetlacza nic nie będziemy odczytywali.

Oprogramowanie skomplikowanego interfejsu, mimo że nie jest specjalnie trudne, to jest zadaniem żmudnym, na które trzeba poświęcić sporo czasu. Aby tego uniknąć można skorzystać z gotowego rozwiązania. Firma STM udostępnia notę aplikacyjną **AN 2824** opisującą sposób obsługi I²C oraz dołączony do niej program demonstracyjny. Po krótkim przetestowaniu procedur z tej noty postanowiłem zastosować je do sterowania wyświetlaczem. Projekt przykładowy jest skonfigurowany dla taktowania z częstotliwością 72 MHz, natomiast rdzeń zastosowanego mikrokontrolera może pracować z maksymalną częstotliwością taktowania 24 MHz. Jeżeli tego nie zmienimy, to mikrokontroler nie będzie działał. Predefiniowane ustawienia taktowania są umieszone w pliku **system_stm32f10x.c**, którego zawartość pokazano na **listingu 1**.

Przed użyciem moduł I²C musi być zainicjowany. Pierwsza cześć inicjacji dotyczy konfigurowania linii portów dla sygnałów SDA i SCL, druga część, to konfiguracja samego modułu I²C, a trzecia opcjonalnie konfiguruje kanał DMA – **listing 2**.

Na **rysunku 4** pokazano sekwencja zapisywania danych przez Mastera do układu Slave.

Z zadaniami wykonywanymi przez mastera na magistrali są związane zdarzenia (*events*, EV):

- EW5: SB1, zerowana po odczytaniu rejestru SR1 . Po tym jest zapisywany rejestr DR z bajtem adresu Slave uzupełnionym o bit R/W=0.
- EV6: ADDR=1, ustawiany w rejestrze SR1, zerowany przez odczytanie rejestru SR2.
- EV8_1: TxE=1, rejestr przesuwny jest pusty, rejestr danych jest pusty, zapisanie danej do rejestru DR.
- EV8: TxE=1, rejestr przesuwny nie jest pusty, rejestr danych jest pusty, zerowany przez zapis rejestru DR.
- EV8_2:, TxE=1, BTF=1 – żądanie sekwencji STOP. TxE i BTF zerowany sprzętowo przez sekwencję *Stop*.

Nota aplikacyjna zawiera procedurę wysyłania przez I²C zawartości bufora z danymi. Jej argumentami są: numer interfejsu sprzętowego (I2C1 lub I2C2), wskaźnik do bufora z danymi, liczba bajtów do wysłania, tryb pracy i adres Slave.

Możliwe są 3 tryby pracy: *polling*, *interrupt* i *DMA*. Sprawdziłem działanie wszystkich. Z jakichś powodów nie pracował poprawnie tryb *interrupt*. Nie sprawdzałem, dlaczego tak było – zapewne przez różnice pomiędzy rdzeniami STM32F100 i STM32F103. Za to tryby *polling* i *DMA* działały bardzo dobrze.

Na **listingu 3** pokazano procedurę wysyłania zawartości bufora w trybie *polling*. W pierwszej kolejności jest inicjowana sekwencja *Start*. Zakończenie tej sekwencji jest sygnalizowane przez wyzerowanie bitu SB – zdarzenie EV5 (**rysunek 4**). Jeżeli sekwencja *Start* nie zakończy się w określonym czasie, to procedura kończy działanie i zwraca koda błędu. Po sekwencji *Start* jest wysyłany adres *slave* przez zapisanie jego wartości do rejestru DR. Wysłanie adresu i odebranie bitu potwierdzenia A jest sygnalizowane ustawieniem bitu ADDR. Jeżeli wszystko przebiega poprawnie, to można zapisywać w pętli kolejne dane do rejestru DR. Wysłanie kompletnej danej jest sygnalizowane ustawieniem bitu TxE. Na **listingu 4** pokazano procedurę wysyłania zawartości bufora z wykorzystaniem kanału DMA.

Mając do dyspozycji procedurę wysyłania bajtów można napisać dwie funkcje podstawowe – do wysyłania komendy i do wysyłania jednej danej. Pokazano je na **listingu 5** i **listingu 6**.

Funkcja *OledCmd* jest przeznaczona tylko do wysyłania komend bez argumentu. Zgodnie z tym, co napisano w bajcie kontrolnym przesyłanym po adresie *slave* bit *Co=1* i bit *D/C=0*. Kiedy jest przesyłana dana przez funkcję *OledData*, to w bajcie kontrolnym *Co=0* i *D/C=1*. Każde przesłanie danej z wykorzystaniem funkcji *OledData* wymaga wysłania adresu *slave*, bajta kontrolnego i właściwego

Listing 3. Wysyłanie zawartości bufora tryb polling

```
else if (Mode == Polling) /* I2Cx Master Transmission using Polling */
{
    Timeout = 0xFFFF;
    /* Send START condition */
    I2Cx->CR1 |= CR1_START_Set;
    /* Wait until SB flag is set: EV5 */
    while ((I2Cx->SR1&0x0001) != 0x0001)
    {
        if (Timeout-- == 0)
        return Error;
    }
    /* Send slave address */
    /* Reset the address bit0 for write*/
    //SlaveAddress &= OAR1_ADD0_Reset;
    Address = SlaveAddress;
    /* Send the slave address */
    I2Cx->DR = Address;
    Timeout = 0xFFFF;
    /* Wait until ADDR is set: EV6 */
    while ((I2Cx->SR1 &0x0002) != 0x0002) if (Timeout-- == 0) return Error;
    /* Clear ADDR flag by reading SR2 register */
    temp = I2Cx->SR2;
    /* Write the first data in DR register (EV8_1) */
    I2Cx->DR = *pBuffer;
    /* Increment */
    pBuffer++;
    /* Decrement the number of bytes to be written */
    NumByteToWrite--;
    /* While there is data to be written */
    while (NumByteToWrite--)
    {
/* Poll on BTF to receive data because in polling mode we can not guarantee
the EV8 software sequence is managed before the current byte transfer
completes */
        while ((I2Cx->SR1 & 0x00004) != 0x000004);
        /* Send the current byte */
        I2Cx->DR = *pBuffer;
        /* Point to the next byte to be written */
        pBuffer++;
    }
    /* EV8_2: Wait until BTF is set before programming the STOP */
    while ((I2Cx->SR1 & 0x00004) != 0x000004);
    /* Send STOP condition */
    I2Cx->CR1 |= CR1_STOP_Set;
    /* Make sure that the STOP bit is cleared by Hardware */
    while ((I2Cx->CR1&0x200) == 0x200);
}
```

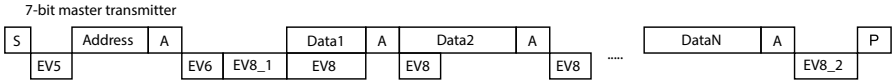
Listing 4. Wysyłanie zawartości bufora tryb DMA

```
if (Mode == DMA)  /* I2Cx Master Transmission using DMA */
{
    Timeout = 0xFFFF;
    /* Configure the DMA channel for I2Cx transmission */
    I2C_DMAConfig (I2Cx, pBuffer, NumByteToWrite, I2C_DIRECTION_TX);
    /* Enable the I2Cx DMA requests */
    I2Cx->CR2 |= CR2_DMAEN_Set;
    /* Send START condition */
    I2Cx->CR1 |= CR1_START_Set;
    /* Wait until SB flag is set: EV5 */
    while ((I2Cx->SR1&0x0001) != 0x0001)
    {
        if (Timeout-- == 0)
        return Error;
    }
    Timeout = 0xFFFF;
    /* Send slave address */
    /* Reset the address bit0 for write */
    SlaveAddress &= OAR1_ADD0_Reset;
    Address = SlaveAddress;
    /* Send the slave address */
    I2Cx->DR = Address;
    /* Wait until ADDR is set: EV6 */
    while ((I2Cx->SR1&0x0002) != 0x0002)
    {
        if (Timeout-- == 0)
        return Error;
    }
    /* Clear ADDR flag by reading SR2 register */
    temp = I2Cx->SR2;
    if (I2Cx == I2C1)
    {
        /* Wait until DMA end of transfer */
        while (!DMA_GetFlagStatus(DMA1_FLAG_TC6));
        /* Disable the DMA1 Channel 6 */
        DMA_Cmd(I2C1_DMA_CHANNEL_TX, DISABLE);
        /* Clear the DMA Transfer complete flag */
        DMA_ClearFlag(DMA1_FLAG_TC6);
    }
    else  /* I2Cx = I2C2 */
    {
        /* Wait until DMA end of transfer */
        while (!DMA_GetFlagStatus(DMA1_FLAG_TC4));
        /* Disable the DMA1 Channel 4 */
        DMA_Cmd(I2C2_DMA_CHANNEL_TX, DISABLE);
        /* Clear the DMA Transfer complete flag */
        DMA_ClearFlag(DMA1_FLAG_TC4);
    }
    /* EV8_2: Wait until BTF is set before programming the STOP */
    while ((I2Cx->SR1 & 0x00004) != 0x000004);
    /* Program the STOP */
    I2Cx->CR1 |= CR1_STOP_Set;
    /* Make sure that the STOP bit is cleared by Hardware */
    while ((I2Cx->CR1&0x200) == 0x200);
}
```

bajta danej. W przypadku przesyłania dużej ilości danych, na przykład przy wyświetlaniu bitmap, można skorzystać z procedury *I2C_Master_BufferWrite*, tylko należy pamiętać, że pierwszy bajt bufora wysyłany po adresie *slave* jest bajtem kontrolnym o wartości 0x40.

Inicjalizacja sterownika

Każdy sterownik wyświetlacza graficznego wymaga zainicjowania. Konieczność inicjalizacji wynika na przykład z możliwości ustalania współrzędnej początkowej (0,0) zależnie od mocowania mechanicznego panelu, różnych źródeł zasileń driverów matrycy (wewnętrzna przetwornica vs napięcie podawane zewnętrznie), poziomu kontrastu, częstotliwości odświeżania itp. Inicjalizacja polega na wysłaniu szeregu komend ustalających początkowe parametry pracy.



Rysunek 4. Sekwencja zapisania danych na magistrali I²C

Page Address	Data		Line Address
D3D2D1D0	D0		00H
	D1		01H
	D2		02H
0 0 0 0	D3	PAGE 0	03H
	D4		04H
	D5		05H
	D6		06H
	D7		07H
D3D2D1D0	D0		08H
	D1		09H
	D2		0AH
0 0 0 1	D3	PAGE1	0BH
	D4		0CH
	D5		0DH
	D6		0EH
	D7		0FH
D3D2D1D0	D0		10H
	D1		11H
	D2		12H
0 0 1 0	D3	PAGE2	13H
	D4		14H
	D5		15H
	D6		16H
	D7		17H
D3D2D1D0	D0		18H
	D1		19H
	D2		1AH
0 0 1 1	D3	PAGE3	1BH
	D4		1CH
	D5		1DH
	D6		1EH
	D7		1FH
D3D2D1D0	D0		20H
	D1		21H
	D2		22H
0 1 0 0	D3	PAGE4	23H
	D4		24H
	D5		25H
	D6		26H
	D7		27H
D3D2D1D0	D0		28H
	D1		29H
	D2		2AH
0 1 0 1	D3	PAGE5	2BH
	D4		2CH
	D5		2DH
	D6		2EH
	D7		2FH
D3D2D1D0	D0		30H
	D1		31H
	D2		32H
0 1 1 0	D3	PAGE6	33H
	D4		34H
	D5		35H
	D6		36H
	D7		37H
D3D2D1D0	D0		38H
	D1		39H
	D2		3AH
0 1 1 1	D3	PAGE7	3BH
	D4		3CH
	D5		3DH
	D6		3EH
	D7		3FH

Column Address												
ADC	D0="0"	00H	01H	02H						81H	82H	83H
	D0="1"	83H	82H	81H						02H	01H	00H
LCD OUT		SEG0	SEG1	SEG2						SEG129	SEG130	SEG131

Rysunek 5. Adresowanie pamięci obrazu

W **tabeli 1** umieszczono wykaz ważniejszych komend sterownika SH1106. Wszystkie komendy są dokładnie opisane w dokumentacji sterownika i powielanie tego opisu nie ma sensu. Dla nas najważniejsze jest takie zainicjowanie sterownika, aby pracował poprawnie z panelem wyświetlacza. Inicjalizacja powinna być przeprowadzana na podstawie danych technicznych producenta matrycy OLED. Niestety, w wypadku tego wyświetlacza nie mamy takich danych i trzeba niektóre ustawienia dobrać eksperymentalnie.

Ponieważ część komend wymaga oprócz samej komendy dodatkowego argumentu, to nie użyłem do inicjowania funkcji *OledCmd*, tylko wszystkie bajty komend inicjalizacji łącznie z niezbędnymi argumentami umieściłem w buforze *Buffer_Init*. Zawartość bufora z pierwszym bajtem kontrolnym 0x80 jest wysyłana do sterownika wyświetlacza za pomocą funkcji *I2C_Master_BufferWrite*. Zawartość tego bufora pokazano na **listingu 7**.

Inicjalizacje można podzielić na kilka etapów:

- Wyzerowanie liczników: kolumn, stron pamięci i linii początkowej.
- Ustawienie powiązania zawartości pamięci RAM obrazu z pozycją na panelu OLED (orientacja wyświetlania).
- Ustawienie kontrastu, przetwornicy DC/DC zasilającej drivery, i częstotliwości taktowania, oraz włączenie wyświetlenia.

Żeby prawidłowo zapisywać dane do zainicjalizowanego sterownika trzeba znać powiązanie zawartości pamięci obrazu z wyświetlanymi pikselami na matrycy OLED. Matryca jest monochromatyczna, a w takim wypadku jednemu pikselowi odpowiada jeden bit w pamięci obrazu. Pomimo że pamięć ma wielkość 132×64 bity, to w rzeczywistości ma ona organizacje bajtową. Host wysyła kolejne bajty za pomocą I²C pod lokacje określone przez liczniki kolumn i stron. Przyporządkowanie tak zaadresowanej pamięci obrazu do pikseli na matrycy pokazano na **rysunku 5**. Jeden bajt w pamięci obrazu odpowiada pionowej linijce o długości 8 pikseli . Najmłodszy bit tego bajta jest pikselem położonym najwyżej w linijce, a bit najstarszy pikselem położonym najniżej. Położenie linijki w poziomie określa licznik kolumn zmieniający się w zakresie 0…131, a położenie w pionie określa licznik stron zmienia się w zakresie 0…7. Po ustaleniu numeru strony kolejne zapisywane bajty tworzą pasek o szerokości 8 pikseli. Każdy zapis danej powoduje inkrementację licznika kolumn i nowa dana jest zapisywana po kolejna lokację. Kiedy licznik kolumn osiągnie wartość 131, to po następnym

Tabela 1. Najważniejsze komendy sterownika SH1106

Komenda	B7	B6	B5	B4	B3	B2	B1	B0	Funkcja
Adres kolumny 4 młodsze bity	0	0	0	0	Młodsza część licznika kolumn				Ustawienie 4 młodszych bitów licznika kolumn
Adres kolumny 4 starsze bity	0	0	0	1	Starsza część licznika kolumn				Ustawienie 4 starszych bitów licznika kolumn
Napięcie wyjściowe wbudowanej przetwornicy DC/DC	0	0	1	1	0	0	napięcie		Ustawienie napięcia wyjściowego wbudowanej przetwornicy DC/DC
Linia początkowa wyświetlacza	0	1		Licznik linii początkowej wyświetlacza					Przypisanie pozycji w pamięci obrazu do COM0
Ustawienie kontrastu	1	0	0	0	0	0	0	1	Kod komendy
	Poziom kontrastu 0…255								Wartość kontrastu
Remapowanie segmentów	1	0	1	0	0	0	0	ADC	ADC=0
Włączenie całego wyświetlacza	1	0	1	0	0	1	0	D	D=0 wyświetlacz wyłączony D=1 wyświetlacz włączony
Wyświetlanie w negatywie	1	0	1	0	0	1	1	D	D=0 wyświetlanie normalne D=1 negatyw
Multiplex ration	1	0	1	0	1	0	0	0	Ilość aktywnych kolumn skanowanych w jednej ramce odświeżania
	*	*	Ilość kolumn 1…64 (domyślnie 64)						
Sterownie wewnętrzną przetwornicą DC/DC	1	0	1	0	1	1	0	1	D=0 DC/DC wyłączona D=1 DC/DC wyłączona
	1	0	0	0	1	0	1	D	
Włączenie/wyłączenie wyświetlacza	1	0	1	0	1	1	1	D	D=0 wyświetlacz wyłączony D=1 wyświetlacz włączony
Ustawienie licznika stron	1	0	1	1	Licznik stron				Ustawienie licznika stron
Kierunek skanowania kolumn	1	1	0	0	D	*	*	*	D=0 COM0->COM63 D=1 COM63->COM0
Offset wyświetlania	1	1	0	1	0	0	1	1	Ustawia pierwszą linię przypisaną początkowi pamięci obrazu
	*	*	COMx						
Częstotliwość wewnętrznego zegara	1	1	0	1	0	1	0	1	B3..B0 – dzielnik czestotliwości 1…16 B7…B4 regulacja częstotliwości -25%..._50%
	Regulacja częstotliwości					Dzielnik			
Sprzętowa konfiguracja wyprowadzeń	1	1	0	1	1	0	1	0	Ustawia 2 sekwencje sterowania wyświetlaczem
	0	0	0	D	0	0	1	0	
Ustawienie napięcia VCOM	1	1	0	1	1	0	1	1	Ustawienie napięcia VCOM
	VCOM=(B[7:0]Xv ref)								

zapisie danych jest zerowany. Przepełnienia licznika kolumn nie powoduje inkrementacji licznika stron i jeżeli nie zmodyfikujemy go wykorzystując do tego celu komendę „ustawienie licznika stron", to kolejne wpisy do pamięci będą nadpisywać dane począwszy od pozycji zerowej (wyzerowanie licznika kolumn). To ważna właściwość, o której trzeba pamiętać, tym bardziej, że w podobnych rozwiązaniach (na przykład SSD1306) przepełnienie licznika kolumn może powodować inkrementację licznika stron.

Tryb tekstowy

Wyświetlanie tekstu jest bardzo ważnym elementem tworzenia graficznego interfejsu użytkownika. W odróżnieniu od klasycznych wyświetlaczy, alfanumerycznych można tu definiować czcionki o różnej wielkości i stosować je zależnie od potrzeb. Przypuszczalnie im mniejszy wyświetlacz, tym większe znaki trzeba będzie definiować, aby były dobrze widoczne. Tablice wzorców dużych znaków zajmują sporo miejsca w pamięci programu mikrokontrolera i dlatego warto definiować tylko te, które są niezbędne. Typowym przykładem takiego podejścia jest definiowanie tylko wzorców cyfr 0…9, plus parę znaków dodatkowych potrzebnych do wyświetlania wartości cyfrowych.

W czasie testów wyświetlacza najpierw wykorzystałem tablicę z wzorcami znaków 8×6 pikseli stosowaną przeze mnie przy wyświetlaniu tekstów na monochromatycznym wyświetlaczu LCD od telefonu komórkowego Nokia 3310. Przy używanym w sterowniku trybie adresowania wystarczy ustawić adres kolumny i numer strony (**listing 8**) i wysłać do sterownika kolejnych 6 bajtów (**listing 9**).

Definiowanie tablicy wzorców znaków jest zajęciem pracochłonnym i dlatego warto skorzystać ze specjalnych programów, które zrobią to za nas. W testach wyświetlacza

Listing 5. Wysłanie komendy

```
//wysłanie  komendy
Status OledCmd(uint8_t cmd)
{
    uint8_t BufCmd[3];
    BufCmd[0]=0x80;//bajt kontrolny
    BufCmd[1]=cmd;
    return(I2C_Master_BufferWrite(I2C1, BufCmd,2,DMA, 0x78));
}
```

Listing 6. Wysłanie jednej danej

```
Status OledData(uint8_t data)
{
    uint8_t BufData[3];
    BufData[0]=0x40;
    BufData[1]=data;
    return(I2C_Master_BufferWrite(I2C1, BufData,2,DMA, 0x78));
}
```

Listing 7. Bufor Buffer_Init z komendami inicjalizacji

```
uint8_t Buffer_Init[255]=
{
    0x80,  //control byte
    0xae,//Display OFF
    0x00,//Low Column
    0x10,//High Column
    0xB0,//Page
    0x40,//Start line
    0xA1,//remap
    0xDA,//com pins
    0x12,
    0xD3, //display offset
    0x00,//NO offset
    0xc0,//scan direction
    0xc8,
    0xA6,//normal display
    0xA4,//display ON
    0x81,//set contrast
    0x50,//contrast DATA
    0xa8,//multiplex ratio
    0x3f,//1/64 duty
    0xD5,//Display clock divide
    0x80,
    0xd9,//precharge period
    0xF1,
    0xDB,//VCOM deselect
    0x40,
    0x8d,//charge pump
    0x14,
    0xAF,//display ON
};
```

Listing 8. ustalenie pozycji na ekranie
```
void WriteChar(char code)
{
    int code_point;
    code_point=((int)code*6);
    OledData(rom_gen[code_point++]);
    OledData(rom_gen[code_point++]);
    OledData(rom_gen[code_point++]);
    OledData(rom_gen[code_point++]);
    OledData(rom_gen[code_point++]);
    OledData(rom_gen[code_point]);
}
```

Listing 9. Wyświetlanie znaku 8×6 pikseli
```
void SetPos(uint8_t x, uint8_t y)
{
    SetColumn(x);
    SetPage(y);
}
```

użyłem darmowego programu **TheDotFactory** autorstwa **Erana Duchana**. Najnowsza wersja programu generuje wzorce w konwencji adresowania pamięci stosowanej w sterowniku SH1106 (rys. 5). Dodatkowo, można zdefiniować tablicę wzorca tylko dla wybranych znaków. Ze względów praktycznych (zajętość pamięci Flash) zdefiniowałem tablicę znaków (głównie cyfr) dla symboli „średnich" o wielkości będącej wielokrotnością 8 pikseli (1 bajta), wysokości 16 pikseli (2 bajty) i szerokości 10 pikseli oraz „dużych" o wysokości 24 pikseli (3 bajty) i szerokości 15 pikseli. Procedurę wyświetlania symboli „średnich" pokazano na **listingu 10**, a „dużych" na **listingu 11**.

W przypadku znaku mającego 16×10 pikseli najpierw jest wyświetlana górna połowa znaku, czyli linijka o wysokości 8 pikseli i szerokości 10 pikseli, przez wysłanie do sterownika 10 bajtów z tablicy wzorca. Potem jest ustawiana wartość początkowa licznika kolumn, licznik stron jest inkrementowany i do sterownika jest wysłanych kolejnych 10 bajtów z tablicy wzorca. Zasada wyświetlania dużych znaków jest identyczna z tym, że wysyłane są bajty wzorca w trzech porcjach po 15 bajtów. Na **fotografii 6** pokazano ekran z wyświetlonymi znakami o 3 różnych wielkościach, od największej do najmniejszej. Nawet przy tak małym wyświetlaczu i ustawieniu kontrastu na średnią wartość duże znaki są dobrze widoczne i czytelne z odległości około 2 metrów. To zapewne zasługa technologii OLED, bo trudno byłoby uzyskać taki efekt z wyświetlaczem LCD o identycznych wymiarach.

Wyświetlanie bitmap

Żeby wyświetlić bitmapę trzeba ją najpierw przekształcić do postaci monochromatycznej oraz dostosować jej wielkości do rozdzielczości matrycy. W naszym wypadku bitmapa musi mieć 128×64 piksele. Do zmiany wielkości obrazu można użyć programów graficznych, na przykład Paint, Microsoft Office Picture Manager, Paint.net lub innych. Tak przygotowaną bitmapę trzeba następnie przekonwertować do tablicy w języku C. Tutaj jest niezbędne zastosowanie specjalnego programu, który potrafi to zrobić zgodnie z zasadą powiązania zawartości pamięci programu z wyświetlanymi pikselami na matrycy – ja

Fotografia 6. Wyświetlenie znaków alfanumerycznych

użyłem **bmp2c.exe**. Po wczytaniu bitmapy, zaznaczeniu opcji V-8 i kliknięciu na przycisk „C", program generuje tablicę w języku C i umieszcza ją w schowku systemu Windows. We własnym programie wystarczy wkleić tę tablicę i pobierać z niej dane w porcjach po 128 bajtów, a po każdej porcji inkrementować licznik stron. Procedurę wyświetlająca pełnowymiarową bitmapę pokazano na **listingu 12**, a efekt jej działania na **fotografii 8**.

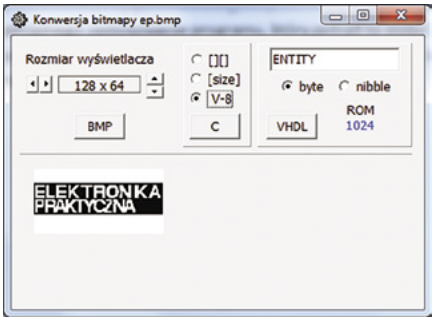
Funkcję *OledBmp* można zoptymalizować czasowo. Każdy bajt z tablicy bitmapy jest wysyłany do sterownika wyświetlacza przez funkcję *OledData*, Jak już wspomniałem, wymaga to wysłania adresu *slave*, bajta kontrolnego i bajta danej. Nie jest zbyt optymalne czasowo, ale w trakcie testów okazało się, że użyty mikrokontroler i jego interfejs I²C wysyła dane na tyle szybko, że bitmapa ładuje się bez widocznego migotania.

Tomasz Jabłoński, EP

Listing 10. Wyświetlanie znaku 16×10 pikseli
```
void CharMedium(uint8_t cyfra, uint8_t x,uint8_t y )
{
    int i;
    SetPos(x*10,y*2);
    for(i=cyfra*20;i<cyfra*20+10;i++)
    OledData (dig_mid[i]);
    SetPos(x*10,(y*2)+1);
    for(i=cyfra*20+10;i<cyfra*20+20;i++)
    OledData (dig_mid[i]);
}
```

Listing 11. Wyświetlanie znaku 24×15 pikseli
```
void CharBig(uint8_t cyfra, uint8_t x,uint8_t y )
{
    int i;
    SetPos(x*15,y*3);
    for(i=cyfra*45;i<cyfra*45+15;i++)
    OledData (dig_big[i]);
    SetPos(x*15,(y*3)+1);
    for(i=cyfra*45+15;i<cyfra*45+30;i++)
    OledData (dig_big[i]);
    SetPos(x*15,(y*3)+2);
    for(i=cyfra*45+30;i<cyfra*45+45;i++)
    OledData (dig_big[i]);
}
```

Listing 12. Wyświetlenie pełnowymiarowej bitmapy
```
void OledBmp(void)
{
    uint16_t i,j,k;
    SetColumn(0);
    SetLine(0);
    SetPage(0);
    k=0;
    for(j=0;j<8;j++)
    {
        SetPage(j);
        SetColumn(0);
        for (i=0; i<128; i++) OledData(bmp[k++]);
    }
}
```



Rysunek 7. Okno programu bmp2c.exe

Fotografia 8. Pełnowymiarowa bitmapa