Wyświetlacze z serii HCMS-29xx firmy Avago Technologies

W artykule zaprezentowano niezbyt popularne, głównie ze względu na cenę, wyświetlacze z serii HCMS. Moduł wyświetlacza zawiera nie tylko matryce LED 5×7, ale również wszystko to, co jest potrzebne do sterowania wyświetlaniem, czyli rejestry przesuwne, wewnętrzny oscylator multipleksowania, źródła prądowe oraz obwody kontroli parametrów pozwalające, na przykład, na zmianę jasności świecenia. W ofercie firmy są moduły o różnej wielkości, począwszy od najmniejszego 1×4 piksele. Są one dostępne w podstawowych kolorach – zielonym, czerwonym, żółtym lub pomarańczowym. Niezależnie od barwy świecenia i liczby matryc zasada sterowania jest taka sama. Najmniejszy moduł składa się z czterech matryc i taki omówię w artykule.

Electrical Description

Pin Function	Description
RESET ($\overline{RST}$)	Sets Control Register bits to logic low. The Dot Register contents are unaffected by the Reset pin. (logic low = reset; logic high = normal operation).
DATA IN (D_{IN})	Serial Data input for Dot or Control Register data. Data is entered on the rising edge of the Clock input.
DATA OUT (D_{OUT})	Serial Data output for Dot or Control Register data. This pin is used for cascading multiple displays.
CLOCK (CLK)	Clock input for writing Dot or Control Register data. When $\overline{Chip\ Enable}$ is logic low, data is entered on the rising Clock edge.
REGISTER SELECT (RS)	Selects Dot Register (RS = logic low) or Control Register (RS = logic high) as the destination for serial data entry. The logic level of RS is latched on the falling edge of the $\overline{Chip\ Enable}$ input.
$\overline{CHIP\ ENABLE}$ ($\overline{CE}$)	This input must be a logic low to write data to the display. When $\overline{CE}$ returns to logic high and CLK is logic low, data is latched to either the LED output drivers or a Control Register.
OSCILLATOR SELECT (SEL)	Selects either an internal or external display oscillator source. (logic low = External Display Oscillator; logic high = Internal Display Oscillator).
OSCILLATOR (OSC)	Output for the Internal Display Oscillator (SEL = logic high) or input for an External Display Oscillator (SEL = logic low).
BLANK (BL)	Blanks the display when logic high. May be modulated for brightness control.
GND_{LED}	Ground for LED drivers.
GND_{LOGIC}	Ground for logic.
V_{LED}	Positive supply for LED drivers.
V_{LOGIC}	Positive supply for logic.

Rysunek 1. Wyprowadzenia układu wyświetlacza (źródło: nota katalogowa)

Każdy, kto zajmuje się konstruowaniem i budową urządzeń, wcześniej czy później jest zmuszony zaprezentować parametry pracy urządzenia na wyświetlaczu. Mogą to być ustawienia menu, wskazania parametrów pracy lub logo firmy. Na rynku mamy bardzo duży wybór różnych wyświetlaczy. Wszystkie do prawidłowej pracy potrzebują algorytmów inicjalizacji i obsługi w kodzie programu. Zajmuje to dość dużo cennej pamięci, co ma szczególnie znaczenie dla tanich mikrokontrolerów wyposażonych w skromne zasoby. Obsługa interfejsu fizycznego wyświetlacza może wiązać się z koniecznością użycia dużej liczby wyprowadzeń lub zastosowania układów pośredniczących. Wyświetlacze Avago są bardzo ciekawą alternatywą, ponieważ prezentują informacje w czytelny sposób, również przy jasnym oświetleniu słonecznym.

Listing 1. Plik nagłówkowy hcms.h

```
#ifndef HCMS_H_
  #define HCMS_H_
  //definicje wyprowadzeń
  #define MOSI      PB5
  #define SCK       PB7
  #define CE_PIN    PB4
  #define RS_PIN    PD5
  #define RESET_PIN PB3
  #define LED_PIN   PD3
  //definicje makr
  #define write PORTB &=~(1<<CE_PIN)
  #define latch PORTB  |=(1<<CE_PIN)
  #define dot_reg PORTD &=~(1<<RS_PIN)
  #define control_reg PORTD  |=(1<<RS_PIN)
  #define reset_on PORTB &=~(1<<RESET_PIN)
  #define reset_off PORTB  |=(1<<RESET_PIN)
  #define led_on PORTD  |=(1<<LED_PIN)
  #define led_off PORTD &=~(1<<LED_PIN)
  #define led_tog PORTD  ^=(1<<LED_PIN)
  /* ustalenie liczby modułów -jeden moduł
   to wyświetlacz z 4 matrycami */
  #define display_len 2
  //deklaracje funkcji
  void Display_init(void);
  void Reset_display(void);
  void Clear_display(void);
  void SPI_init(void);
  void Send_SPI(uint8_t a);
  void putChar( char c, uint8_t  *buf);
  void putStringP(const char * s, uint8_t * buf);
  void putString(char * s, uint8_t * buf);
  void putInt(uint8_t x, int8_t a);
  uint8_t textBufor[20];
#endif /* HCMS_H_ */
```

Zaglądając do noty katalogowej układu, widzimy, że wyświetlacz komunikuje się z mikrokontrolerem za pomocą interfejsu SPI. W dalszej kolejności musimy zwrócić uwagę na to, jakie wyprowadzenia układu będą nam jeszcze potrzebne do sterowania. Na **rysunku 1** zamieszczono zaczerpnięty z dokumentacji opis wyprowadzeń. W podstawowej konfiguracji nie wszystkie są używane:

- **RESET** powinno być dołączone do mikrokontrolera.
- **DATA IN** to wejście danych do układu (do MOSI mikrokontrolera).
- **DATA OUT** jest wyjściem dołączanym danych do kolejnego układu. Służy do szeregowego łączenia układów wyświetlaczy.
- **CLOCK** wejście sygnału zegarowego interfejsu danych (do SCK mikrokontrolera).
- **RS** doprowadzenie sterujące zapisem danych do rejestru diod lub rejestrów kontrolnych.

Listing 2. Funkcje służące do obsługi wyświetlacza

```
//Inicjalizacja wyświetlacza
void Display_init(void)
{
    DDRD |= (1<<LED_PIN)|(1<<RS_PIN);
    DDRB |= (1<<RESET_PIN);
    Reset_display();
}

//Zerowanie wyświetlacza
void Reset_display(void)
{
    reset_on;
    _delay_ms(20);
    reset_off;
}

//Kasowanie wyświetlacza
void Clear_display(void)
{
    for(uint8_t i=0; i<(display_len)*20; i++) Send_SPI(0x00);
}

//Inicjalizowanie interfejsu SPI
void SPI_init(void)
{
    DDRB |= (1<<MOSI)|(1<<SCK)|(1<<CE_PIN);
    SPCR |= (1<<SPE)|(1<<MSTR);
    SPSR |= (1<<SPI2X);
}

//Wysłanie słowa za pomocą SPI
void Send_SPI(uint8_t a)
{
    write;
    SPDR=a;
    while(!(SPSR&(1<<SPIF)));
    latch;
}

//Przesłanie znaku do wyświetlacza
void putChar(char c, uint8_t  *buf)
{
    for(uint8_t col=0; col<5; col++)
    *(buf+col)=pgm_read_byte(font5x8+(c-' ')*5 + col);
}

//Wyświetlanie napisu z pamięci programu
void putStringP(const char * s, uint8_t * buf)
{
    char c;

    while (( c = pgm_read_byte (s++)))
    {
        for(uint8_t i=0; i<5; i++) putChar(c, buf);
        for(uint8_t i=0;i<5;i++) Send_SPI(textBufor[i]);
    }
}

//Wyświetlenie napisu z pamięci danych
void putString(char * s, uint8_t * buf)
{
    char c;

    while (( c = *(s++)))
    {
        for(uint8_t i=0; i<5; i++) putChar(c, buf);
        for(uint8_t i=0;i<5;i++) Send_SPI(textBufor[i]);
    }
}

//Funkcja wyświetlająca wartość liczbową
void putInt(uint8_t x, int8_t a)
{
    char bufor[5];

    itoa(a, bufor, 10);
    putString(bufor, textBufor);
    for(uint8_t j=0; j<(x-1); j++)
    {
        putChar(32,textBufor);
        for(uint8_t i=0;i<5;i++) Send_SPI(textBufor[i]);
    }
}
```

- **CE** wejście sterujące zapisem danych. Zapis następuje, gdy wejście CE jest wyzerowane. Narastające zbocze CE powoduje zapamiętanie danych w rejestrach.
- **SEL** jest wejściem sygnału służącego do wyboru wewnętrznego lub zewnętrznego oscylatora. Dołączenie do napięcia +5 V (poziomu wysokiego) spowoduje wybór oscylatora wewnętrznego.
- **BLANK** wejście służące do wyłączenia wyświetlacza. Aktywne, gdy jest ustawione. Podczas normalnej pracy powinno być wyzerowane.

Pozostałe piny to masa i zasilanie, przy czym możemy korzystać z oddzielnych źródeł napięcia dla logiki i driverów diodo.

Wiemy już, co i jak przyłączyć – czas na realizację. Na **listingu 1** zamieszczono plik **hcms.h**, w którym dla wygody przygotowano makra służące do obsługi układu wyświetlacza. Oprócz makr dla poszczególnych pinów widzimy deklarację potrzebnych funkcji, bufora tekstowego oraz definiujemy liczbę połączonych modułów. Najważniejsze funkcje, oprócz inicjalizacji modułu SPI, to funkcje narzędziowe do inicjalizacji, zerowania i czyszczenia wyświetlaczy.

Na **listingu 2** pokazano funkcje użyteczne przy sterowaniu wyświetlaczem. W funkcji **Display_init** oprócz ustawienia potrzebnych pinów dokonujemy zerowania wyświetlacza. Sama funkcja **Reset_display** sprowadza się do zmiany poziomu na dedykowanym wyprowadzeniu mikrokontrolera. Funkcja **Clear_display** za pomocą pętli *for* wpisuje do rejestru diod same 0, co powoduje wyłączenie wyświetlacza. Liczba powtórzeń pętli jest wyznaczana makrem *display_len*. Oczywiście, wpisanie do wysłania wartości 0xFF spowoduje zaświecenie wszystkich diod w matrycach.

Funkcje **SPI_init** oraz **Send_SPI** nie wymagają omawiania. Interfejs SPI działa w trybie *MODE 0*. Można zrezygnować z włączenia bitu SPI2X. W testach zegar kontrolera pracował z zewnętrznym oscylatorem kwarcowym o częstotliwości 16 MHz, więc szybkość zegara SPI to 8 MHz przy włączonym bicie SPI2X. Jedyne, na co pragnę zwrócić uwagę w funkcji *Send_SPI*, to kolejność ustawiania pinu CE inaczej niż w typowych rejestrach przesuwnych.

Mamy zainicjalizowany wyświetlacz. Czas na funkcje pozwalające na wyświetlenie komunikatu lub zmiennej. Podstawową będzie funkcja wyświetlająca pojedynczy znak. Można by było napisać funkcję wyświetlającą pojedynczy piksel, ale ze względu na specyfikę tych wyświetlaczy nie będziemy na nich tworzyć grafik, więc ograniczymy się do wyświetlenia znaku zdefiniowanego w pliku **fonty.h**. Funkcja **putChar** wypełnia bufor odpowiednimi bitami, odwołując się do wspomnianego pliku z definicjami fontów. Funkcja niczego nie wyświetla, a jedynie wypełnia bufor odpowiednimi kombinacjami bitów. Jeżeli chcielibyśmy za jej pomocą wyświetlić pojedynczy znak, to musimy dodać pętlę, która prześle dane 5 kolumn z bufora na wyświetlacz.

Kolejne dwie funkcje to **putString** i **putStringP**. Obydwie umożliwiają wyświetlenie łańcucha znaków zapisanego w pamięci danych RAM lub programu Flash. Funkcje różnią się tylko odwołaniem do miejsca pobierania znaku. Następnie wypełniamy bufor i za pomocą krótkiej pętli *for* przesyłamy dane z bufora do wyświetlacza. Oczywiście – każdy znak osobno.

Pozostała jeszcze do omówienia funkcja wyświetlająca wartość liczbową, to jest **putInt()**. Dodano w niej parametr pozwalający na pozycjonowanie wyświetlanej wartości. Wyświetlając łańcuch, możemy do pozycjonowania użyć znaków odstępu, natomiast w wypadku wyświetlania wartości temperatury takiej możliwości nie mamy. Dlatego dodałem argument pozwalający na taką czynność. Pierwszy argument to właśnie pozycja na wyświetlaczu. Drugi to zmienna do wyświetlenia, przy czym mamy tu dowolność typu. Podstawowy, użyty w funkcji to 8-bitowy ze znakiem, ale w zależności od potrzeb możemy to zmienić. Funkcja zamienia wartość liczbową na znak ASCII, a następnie za pomocą *putString* wypełnia bufor tekstowy, który potem jest wysyłany do wyświetlacza. W funkcji widać jeszcze jedną pętlę, która ma za zadanie pozycjonowanie łańcucha na wyświetlaczu. Bufor tekstowy jest wypełniany znakiem spacji w liczbie

Listing 3. Przykładowy program testujący wyświetlacz

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/pgmspace.h>
#include „hcms.h”
#include „fonty.h”

int licznik; // zmienna tylko do testów w pętli głównej
const char PROGMEM Napis1[]={„HCMS2903”};

int main(void)
{
    Display_init();
    SPI_init();
    dot_reg;
    Clear_display();

    putStringP(Napis1, textBufor);
    control_reg;
    //rejestr „0” - wyłączenie bitu „sleep”, ustawienie jasności
    Send_SPI(0x73);
    //rejestr „1” - włączenie bitu „Data out”
    Send_SPI(0x81);
    _delay_ms(5000);
    // sei();
    while(1)
    {
        led_tog;
        _delay_ms(1000);
        dot_reg;
        Clear_display();
        putInt(3,licznik);
        licznik++;

    }
}
```

wskazanej przez pierwszy argument funkcji pomniejszony o jeden, ponieważ procesor liczy od 0.

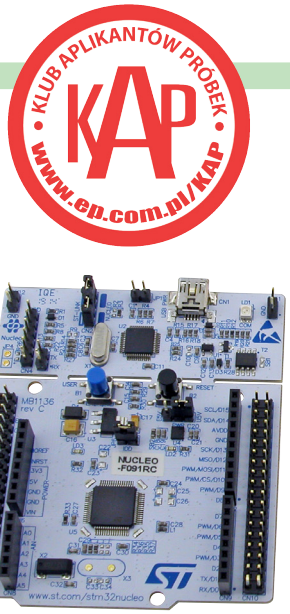
Testując wyświetlacze, ograniczyłem się do kilku podstawowych funkcji, mając na uwadze to, że najważniejsze jest uruchomienie układu, a dalsze zabawy, na przykład z przewijaniem czy miganiem napisu pozostawiam czytelnikom. Przykładowy test wyświetlacza pokazano na **listingu 3**.

Na początku są dołączane potrzebne biblioteki. Definiujemy zmienną *licznik* do testów oraz komunikat w pamięci Flash. Inicjalizujemy wyświetlacz oraz interfejs SPI, czyścimy wyświetlacz i za pomocą funkcji *putStringP* wyświetlamy wcześniej zdefiniowany napis. Następnie musimy zająć się rejestrami kontrolnymi wyświetlacza, ponieważ bez tego nic nie zobaczymy.

Wyświetlacz ma dwa rejestry kontrolne indeksowane 0 i 1. W pierwszym bezwzględnie musimy ustawić bit na pozycji szóstej. Odpowiada on za tryb sleep wyświetlacza i jeżeli jest wyzerowany, to wyświetlacz pozostaje ciemny (wyłączony). Należy wspomnieć o tym, że w trybie uśpienia wyświetlacz pobiera prąd o natężeniu jedynie 50 µA, co można wykorzystać przy zasilaniu bateryjnym. Pozostałe bity 0…5 rejestru decydują o jasności świecenia. Po szczegóły odsyłam do noty aplikacyjnej. Rejestr o indeksie 1 wykorzystujemy tylko przy kaskadowym połączeniu wyświetlaczy. Ustawienie bitu 0 spowoduje, że na pinie DATAOUT wyświetlacza otrzymamy dane dla kolejnych modułów. W pętli głównej migamy diodą oraz wyświetlamy co sekundę zmienną licznik.

Na koniec zachęcam czytelników do dzielenia się pomysłami na stronach forum „Elektroniki Praktycznej”, gdzie można zastosować te wyświetlacze.

Marek Rębecki
marekrebecki@wp.pl



Nucleo-F091RC – zestaw startowy z mikrokontrolerem z rodziny STM32 (STM32F091)

Dzięki uprzejmości firmy Kamami, mamy dla czytelników EP zestaw startowy z serii Nucleo wyposażony w mikrokontroler STM32F091RCT6 (256 kB Flash), z wbudowanym programatorem-debugerem ST-Link/v2, wyposażony w złącza zgodne z Arduino.

Płytki Nucleo są jedną z najłatwiejszych, najtańszych i najbardziej przystępnych metod zapoznania się z mikrokontrolerami STM32. Stanowią bazę, którą można obudować układami zewnętrznymi wykonując w ten sposób prototyp dowolnego urządzenia z układem procesorowym.

Dzięki złączom płytki nadają się do wielokrotnego użytku. Wbudowany programator-debugger zwalnia ich użytkownika z konieczności zakupu drogiego wyposażenia, a szeroka gama dostępnych procesorów przy jednoczesnej kompatybilności płytek pin-to-pin pozwalają na łatwe dobranie odpowiedniego układu do aplikacji oraz jego zmianę w razie konieczności.

Dla płytek są dostępne kompilatory oferowane przez różnych producentów (w tym darmowe), biblioteka HAL, oprogramowanie ułatwiające konfigurację oraz wyczerpująca dokumentacja, przykłady programów oraz gotowych aplikacji.

Podstawowe parametry płytki oferowanej w ramach KAP:

- Mikrokontroler STM32 w obudowie LQFP64.
- Dostępność dwóch rodzajów płytek rozszerzeń: moduły Arduino Uno rev. 3 oraz STMicroelectronics Morpho.
- Wbudowany programator/debugger ST-Link/v2-1 z interfejsem SWD. Programator można wyłamać i używać do programowania innych procesorów STM32.
- Możliwość zasilania za pomocą USB lub zewnętrznego zasilacza o napięciu z zakresu 3,3...5 V lub 7...12 V.
- Trzy diody LED: sygnalizacja komunikacji USB LD1, do wykorzystania w aplikacji użytkownika LD2, sygnalizująca załączenie zasilania LD3.
- Dwa przyciski: USER oraz RESET.
- Trzy tryby pracy USB: Virtual COM, pamięć masowa, port programatora/debugera.