Piotr Knutel (156476), ET3-DI

Systemy Mikroprocesorowe

Projekt 2

Temat 13:

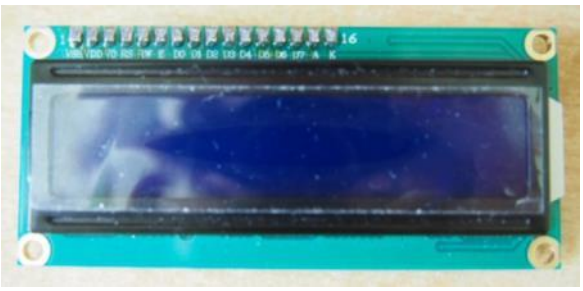
Instrukcja laboratoryjna dla zestawu NUCLEO-64 STM32F103: wyświetlacz znakowy LCD ze sterownikiem HD44780. Dołączenie do zestawu w sposób estetyczny wyświetlacza i zaproponowanie ćwiczenia laboratoryjnego.

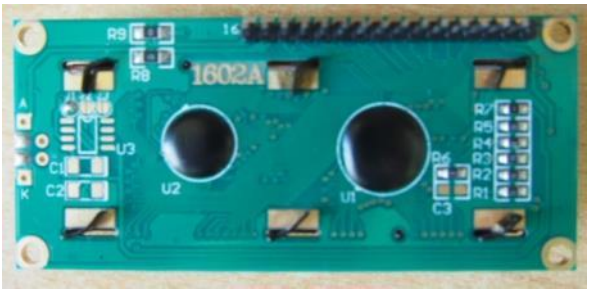
Wstęp

Wyświetlacze alfanumeryczne LCD, dzięki niskim cenom i dużym możliwościom, zyskały popularność w projektach różnych urządzeń elektronicznych. Sterowanie samym wyświetlaczem jest złożone, dlatego powszechnie używa się sterowników, które odbierają rozkazy i dane od głównego mikroprocesora, a następnie zapalają lub gaszą poszczególne piksele/segmenty. Sterowniki te mogą posiadać pamięć, a także wysłać określone dane do samego mikroprocesora. Takim sterownikiem jest HD44780.

Wyświetlacza LCD ze sterownikiem HD44780

Układ HD44780 może sterować pracą różnych wyświetlaczy alfanumerycznych. Wybrano wyświetlacz 2-liniowy, z 16-stoma znakami w jednej linii. Wyświetlacz, razem ze sterownikiem, wyprowadzeniami i innymi elementami, tworzy przedstawiony poniżej moduł.

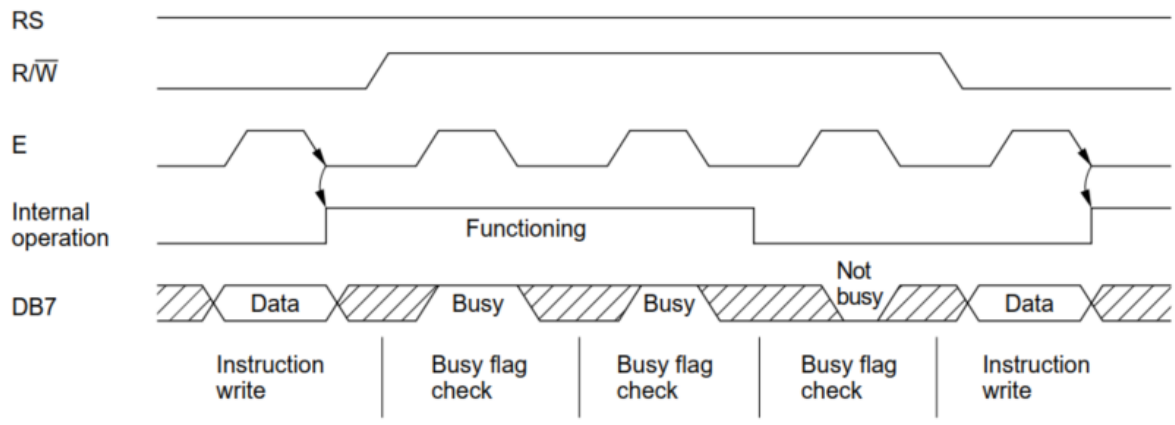




Komunikacja ze sterownikiem odbywa się za pomocą następujących wyprowadzeń:

Nazwa	Funkcja
RS	*Register Select*, wybór rejestru; 0 – rejestr instrukcji, 1 – rejestr danych
R/W	*Read/Write*, czytaj/zapisz; 0 – wysłanie do wyświetlacza, 1 – obiór z wyświetlacza
E	*Enable*, taktowanie transmisji, aktywne jest zbocze opadające
D7, B6, D5, D4	*Data Bits*, starsza połówka bajta danych
D3, D2, D1, D0	młodsza połówka bajta danych (nie używana w trybie 4-bitowym)

Po wysłaniu instrukcji, a przed wysłaniem kolejnej, należy poczekać, aż sterownik skończy wewnętrzne operacje. Zazwyczaj jest to co najmniej 37µs, a dla polecenia *Return Home* – 1,52ms. Inną metodą jest odczytanie flagi zajętości *Busy Flag*. Odczyt jest możliwy przy RS=0 i R/W=1. Jeśli na D7 będzie stan wysoki, sterownik pracuje, jeśli niski – można wysyłać następne dane/rozkazy.



Komunikacja ze sterownikiem jest możliwa za pomocą 8 lub 4 linii danych. W trybie 4-bitowych najpierw jest wysłana starsza połówka bajta, a następnie młodsza za pomocą linii D7 – D4. W tej instrukcji jest wykorzystana metoda 8-bitowa.

Sterownik może odbierać rozkazy, które go konfigurują lub powodują wykonanie konkretnych czynności. Rozkazy są wysyłane, gdy na RS i R/W jest stan niski. Poniżej zaprezentowano kilka z nich.

Nazwa rozkazu	D7	D6	D5	D4	D3	D2	D1	D0
Function Set	0	0	1	DL	N	F	*	*
Display On/Off Control	0	0	0	0	1	D	C	B
Entry Mode Set	0	0	0	0	0	1	I/D	S
Clear Display	0	0	0	0	0	0	0	1
Return Home	0	0	0	0	0	0	1	*
Cursor or Display Shift	0	0	0	1	S/C	R/L	*	*
Set CGRAM Adress	0	1	A5	A4	A3	A2	A1	A0
Set DDRAM Adress	1	A6	A5	A4	A3	A2	A1	A0
Read Busy Flag and Adress (RS=0, R/W=1)	BF	A6	A5	A4	A3	A2	A1	A0
Write Data to CG or DDRAM (RS=1, R/W=0)	D7	D6	D5	D4	D3	D2	D1	D0
Read Data from CG or DDRAM (RS=1, R/W=1)	D7	D6	D5	D4	D3	D2	D1	D0

„*” obojętne, 0 lub 1

Function Set – niezbędna na początku sterowania.
DL=1 tryb 8-bitowy, DL=0 tryb 4-bitowy.
N=0 jedna linia znaków, N=1 dwie linie znaków.
F=0 rozmiar znaków to 5 x 8 px, F=1 to 5 x 10 px.

Display On/Off Control
D=1 wyświetlacz wł., D=0 wył.
C=1 pokazywanie kursora wł., C=0 pokazywanie kursowa wył.
B=1 znak pokazany przez kursor miga, B=0 nie miga.

Entry Mode Set
I/D=1 automatyczne przesuwanie kursora w prawo, po wysłaniu znaku, I/D=0 w lewo
S=1 przesuwa wyświetlane znaki, S=0 nie przesuwa.

Clear Display – czyści wyświetlacz, zapełnia go spacjami.

Return Home – adres aktualnego pola znaku jest ustawiany na 0x00, czyli kursor wraca na początek.

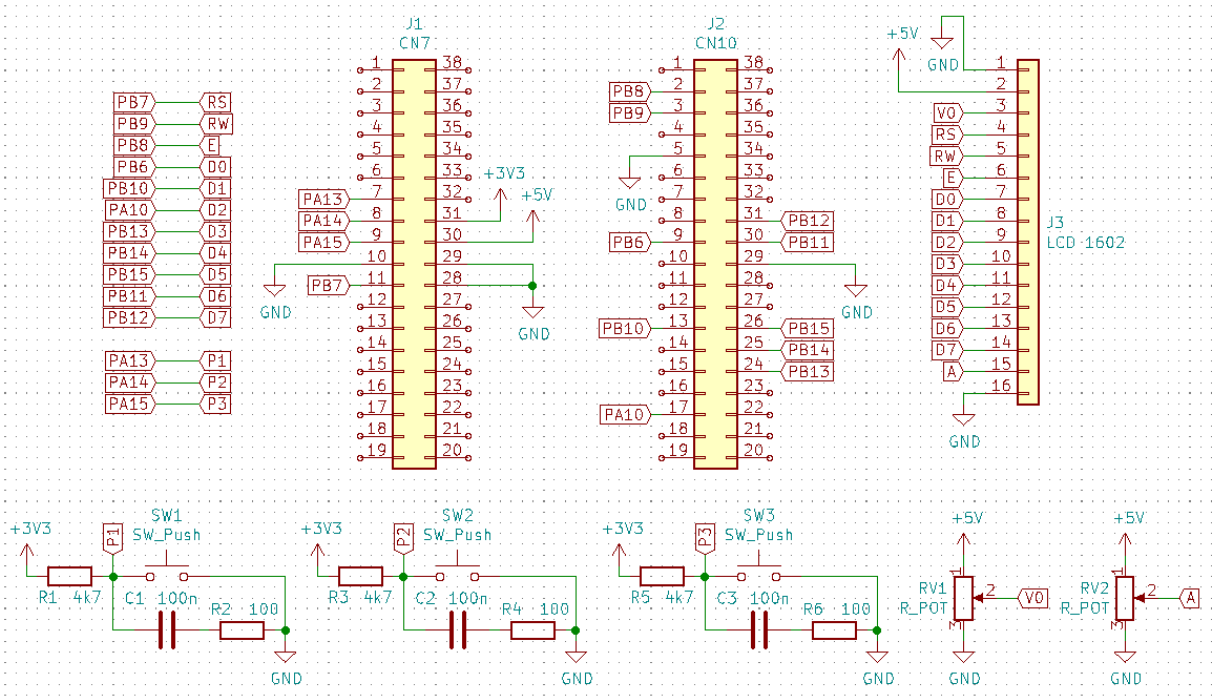
Znaki wysyła się przy RS=1 i R/W=0, poprzez odpowiednie ustawienie D7 – D0. Wpisuje się na nie kod znaku. Tablica kodów jest w dodatku na końcu tej instrukcji. Kody znaków odpowiadają kodom ASCII. Tak więc, 0 to 48 (0x30), 1 to 49 (0x31), A to 65 (0x41), a to 97 (0x61). Należy pamiętać o wyzwalaniu odczytu przez sterownik, za pomocą zbocza opadającego na linii E.

Pamięć DDRAM zawiera 2 x 40 komórek, a używany wyświetlacz ma tylko 2 wiersze po 16 znaków. Oznacza to, że do pamięci można wczytać więcej znaków, niż się aktualnie wyświetla, i je przesuwać. **Dla wybranego wyświetlacza adresy pierwszej linii to 0x00 – 0x27, a drugiej to 0x40 – 0x67.**

Moduł rozszerzeń

W celu łatwego, szybkiego i pewnego połączenia płytki rozwojowej *Nucleo64* (STM32F103RB) z wyświetlaczem LCD wykonano moduł rozszerzeń na bazie laminatu miedzianego. Moduł wyposażony jest dodatkowo w 3 przyciski, które mogą służyć, jako urządzenia wejścia mikrokontrolera. Dwa potencjometry 10kΩ służą do ręcznej regulacji podświetlenia i napięcia kontrastu.

Na schemacie: po prawej jest gniazdo na wyświetlacz, na środku – gniazdo na *Nucleo64*, po lewej – sposób połączenia mikrokontrolera z płytką wyświetlacza, a na dole – przyciski i potencjometry.



Rysunek 1: Schemat modułu rozszerzeń

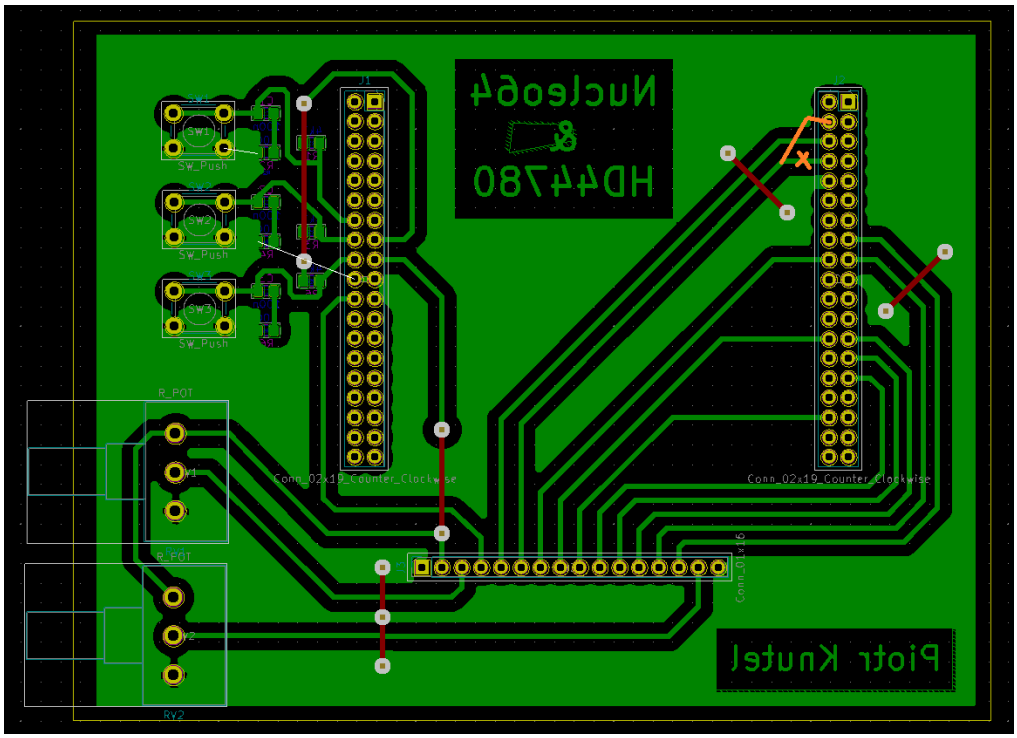
Jako napięcie zasilania wyświetlacza wybrano 5V, pomimo możliwości zasilania 3,3V, ponieważ dało się wtedy poprawnie dobrać napięcie kontrastu. Wymogło to podłączenie pinów danych (D0 – D) do wyprowadzeń mikrokontrolera, które posiadają tolerancję 5V (FT), ponieważ mogą one być używane jako wejścia.

Projekt PCB wykonano w programie KiCAD. Wymiary płytki to 84 x 112 mm. Płytka jest jednostronnie pokryta miedzią. Gniazda, przyciski i potencjometry są w obudowach do montażu przewlekanego (THT), a rezystory i kondensatory – do montażu powierzchniowego (SMD). Rozmiar obudowy elementów powierzchniowych to 0805.

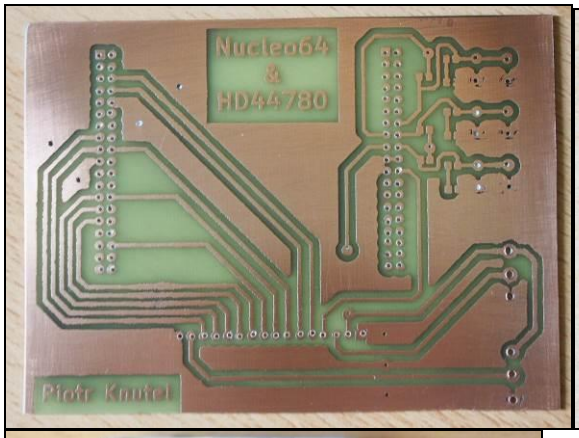
Podczas projektowania doszło do pomyłki przy połączeniu PB8 – RW. Wyprowadzenie RW modułu wyświetlacza zostało podciągnięte do wyprowadzenia AVDD *Nucleo64*, zamiast do PB8. Problem rozwiązano uszkadzając istniejącą ścieżkę i dodając niewielką zworkę (fioletowy przewód). Poprawkę zaznaczono kolorem pomarańczowym na rysunku projektu PCB.

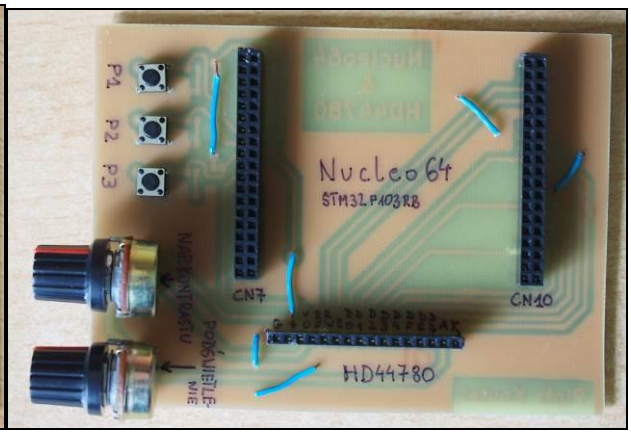
Ułożenie ścieżek zostało wydrukowane na papierze kredowym, a następnie, za pomocą metody termotranseru, przeniesione na laminat. Płytka została wytrawiona w roztworze nadsiarczanu sodowego.

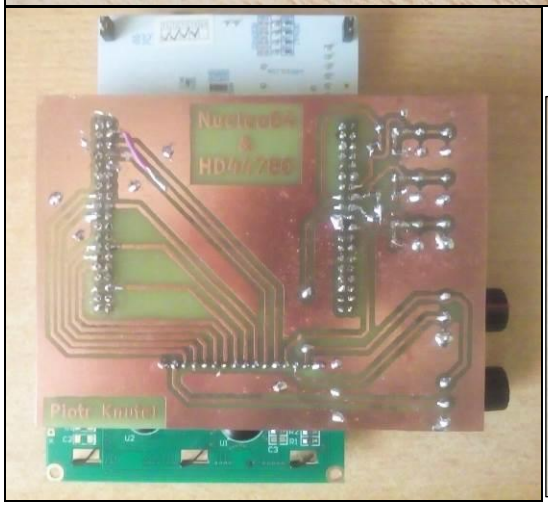
Jakość połączeń linii danych, prowadzących do gniazda wyświetlacza, sprawdzono za pomocą prostego programu (okresowo zmieniającego stany wyprowadzeń) i diody LED (podłączanej do poszczególnych pinów gniazda wyświetlacza.

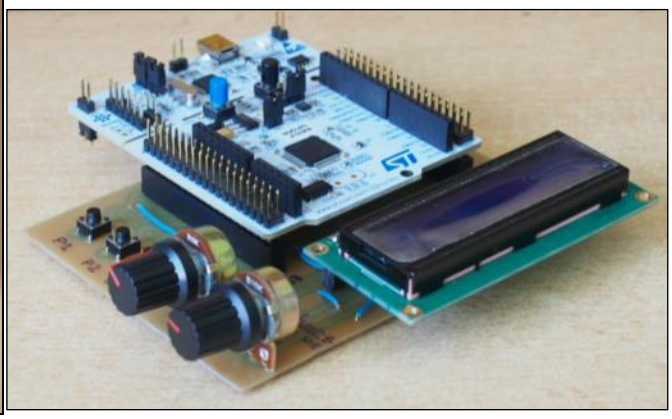


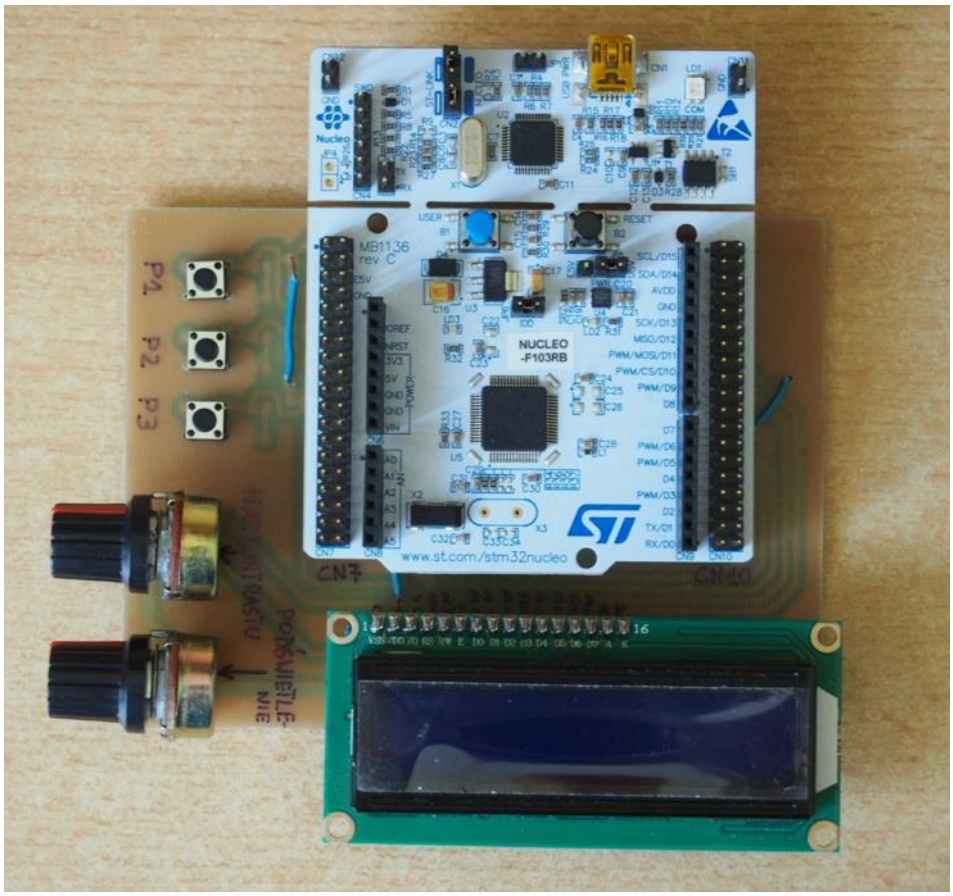
Rysunek 2: Projekt PCB











Rysunek 3: Moduł rozszerzeń

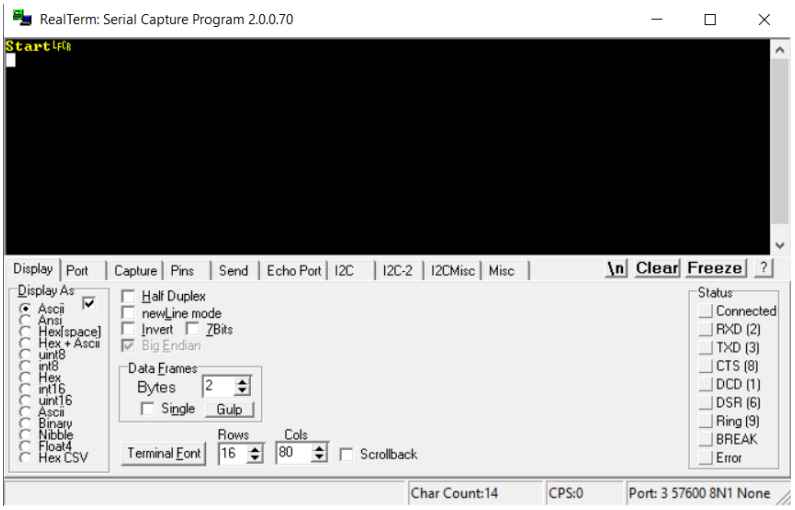
Zadania

Początek

Należy ustawić wszystkie piny jako wyjścia. Opcjonalna jest też napisanie funkcji ustawiających D7 - D0 jako wejścia lub wyjścia. Przykładowy fragment:

```
GPIO_InitTypeDef rs_Struct;
rs_Struct.GPIO_Mode = GPIO_Mode_Out_PP;
rs_Struct.GPIO_Pin = pinRS;
rs_Struct.GPIO_Speed = GPIO_Speed_10MHz;
GPIO_Init(portRS, &rs_Struct);
```

Potrzebna jest też funkcja opóźniająca. Można zbudować 2 funkcje, jedną z argumentem podawanym w ms, a drugą z argumentem w dziesiątkach µs. Dołączony plik zawiera funkcje *opoznienie_ms()* i *opoznienie_10us()*. Do ich działania niezbędna jest zmienna globalna *unsigned int licznik_opoznienia* i konfiguracja *SysTick* w *SysTick_Konfiguracja(90ul)*. W main() należy wpisać też: *NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0)*. Do testowania działania swojego kodu student może użyć też wysyłania komunikatów przez USART2, który jest domyślnie podłączony do programatora i da się z niego korzystać poprzez przewód USB. Funkcje *usart2_Konfiguracja(), usart2WyslijBajt(char c), usart2WyslijCiag(char* s)* są dołączone do projektu, ale ich używani nie jest konieczne. Mogą służyć jako pomoc.



Rysunek 4: Odbieranie komunikatów przez USART2, pomocny w testowaniu programu

Konfiguracja

Początkowa konfiguracja sterownika wymaga:

1. Ustawienie wszystkich linii jako wyjścia i stanu wysokiego na E
2. Rozkaz Function Set
3. Rozkaz Display on
4. Rozkaz Entry Mode Set
5. Wyczyszczenie wyświetlacza i powrót kursora na początek (rozkaz Clear i Home)

Ułatwieniem w ustawianiu stanów na wyjściach mikrokontrolera może być napisanie funkcji takiej jak *lcd_ustaw_bajt(uint8_t bajt)*.

```
void lcd_ustaw_bajt(uint8_t bajt)
{
	bool d0, d1, d2, d3, d4, d5, d6, d7;
	d0 = (bool)(bajt&0b00000001);
	d1 = (bool)((bajt&0b00000010)>>1);
	d2 = (bool)((bajt&0b00000100)>>2);
	d3 = (bool)((bajt&0b00001000)>>3);
	d4 = (bool)((bajt&0b00010000)>>4);
	d5 = (bool)((bajt&0b00100000)>>5);
	d6 = (bool)((bajt&0b01000000)>>6);
	d7 = (bool)((bajt&0b10000000)>>7);

	if(d0)
		GPIO_SetBits(portD0, pinD0);
	else
		GPIO_ResetBits(portD0, pinD0);
	if(d1)
		GPIO_SetBits(portD1, pinD1);
	else
		GPIO_ResetBits(portD1, pinD1);
	Dalej podobnie dla d2, d3, d4, d5, d6 i d7.
}
```

Poniższe funkcje ustawiają odpowiednio RS i R/W, w zależności od tego, czy wysłany jest rozkaz, czy dane. Wywołują funkcję *wyzwolenie()*, odpowiedzialną za pojawienie się zbocza opadającego na linii E, które to wyzwala odczyt przez sterownik. Wprowadzają też opóźnienia 0,1ms, które umożliwiają

ustabilizowanie się stanów na liniach, oraz poczekanie na koniec operacji wewnętrznych sterownika. Dzięki temu nie jest konieczne używanie flagi *Busy Flag*.

```
void lcd_rozkaz(uint8_t rozkaz)
{
	GPIO_ResetBits(portRW, pinRW);
	GPIO_ResetBits(portRS, pinRS);
	opoznienie_10us(10);
	lcd_ustaw_bajt(rozkaz); //function set
	wyzwolenie();
	opoznienie_10us(10);
}

void lcd_wyslij_dane(uint8_t dane)
{
	GPIO_ResetBits(portRW, pinRW);
	GPIO_SetBits(portRS, pinRS);
	opoznienie_10us(10);
	lcd_ustaw_bajt(dane);
	wyzwolenie();
	opoznienie_10us(10);
}
```

Funkcje obsługi wyzwalania za pomocą linii E:

```
void e(bool stan)
{
	if(stan)
		GPIO_SetBits(portE, pinE);
	else
		GPIO_ResetBits(portE, pinE);
}

void wyzwolenie(void)
{
	e(1);
	opoznienie_10us(10);
	e(0);
}
```

Całą konfigurację można teraz zrobić następująco.

```
void lcd_Konfiguracja()
{
(w tym miejscu konfiguracja pinów RS, RW, E i D7-D0 jako wyjścia)
	opoznienie_ms(30);
	GPIO_SetBits(portE, pinE);
	lcd_rozkaz(function_set);
	lcd_rozkaz(display_on_off);
	lcd_rozkaz(entry_mode_set);
	lcd_rozkaz(clear_display);
	lcd_rozkaz(return_home);
	opoznienie_ms(2);		//rozkaz Home wymaga więcej czasu (1,52ms)
}
```

Użyte tutaj nazwy zamiast wartości w bajtach są zdefiniowane na początku pliku main.c. Dotyczą one używanego wyświetlacza i konkretnych ustawień, które student może zmienić.

```
#define function_set	0b00111000
#define display_on_off	0b00001100
#define entry_mode_set	0b00000110
```

```
#define clear_display     0b00000001
#define return_home       0b00000010

#define kursor_prawo      0b00010100
#define kursor_lewo       0b00010100
#define tekst_prawo       0b00011100
#define tekst_lewo        0b00011000
```

Wyświetlenie znaku

Po konfiguracji można wyświetlać znaki. Można to zrobić wywołując funkcję *void lcd_wyslij_dane(uint8_t dane)*. Jej definicja znajduje się wcześniej.

Przykład użycia (wyświetli 0): *lcd_wyslij_dane(48)*;
(wyświetli 2): *lcd_wyslij_dane(0x32)*;
(wyświetli duża literę B): *lcd_wyslij_dane('B');*

Wyświetlenie tekstu

Wiedząc już jak wyświetlić znak, należy napisać funkcję do wyświetlania ciągu zanków.

Przykładowe rozwiązanie:

```
void lcd_wyslij_ciag(char *s)
{
       while (*s)
              lcd_wyslij_dane(*s++);
}
```

Przykładowe wywołanie: *lcd_wyslij_ciag("HD44780 jest ok!");*

Wybór adresu pamięci wyświetlacza

Należy napisać funkcję która będzie ustawiała kursor na konkretnym polu.

Przykładowe rozwiązanie:

```
void lcd_ustaw_adres_DD(uint8_t adres)
{
       if(adres <= 0x27 || (adres >= 0x40 && adres <= 0x67))
              lcd_rozkaz(0b10000000 | adres);
}
```

Przykładowe użycie: *lcd_ustaw_adres_DD(adres_poczatku_dolnej_lini);*

Jeśli jest wcześniej napisane: *#define adres_poczatku_dolnej_lini 0x40*

Zadania dodatkowe

Spowodować wyświetlanie kursora. Wyświetlić znak % i &. Przesunąć wpisane znaki o 1 miejsce w prawo.

Zadanie zaawansowane

Należy skonfigurować sterownik w trybie 4-bitowym.

Dodatek: tablica znaku znajdujących się w pamięci sterownika i ich adresy.

Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)																
xxxx0001	(2)																
xxxx0010	(3)																
xxxx0011	(4)																
xxxx0100	(5)																
xxxx0101	(6)																
xxxx0110	(7)																
xxxx0111	(8)																
xxxx1000	(1)																
xxxx1001	(2)																
xxxx1010	(3)																
xxxx1011	(4)																
xxxx1100	(5)																
xxxx1101	(6)																
xxxx1110	(7)																
xxxx1111	(8)																