Programowanie STM32F4 (6)

W artykule weźmiemy na warsztat popularny, monochromatyczny wyświetlacz graficzny LCD będący klonem wyświetlacza używanego w telefonach Nokia 5110 oraz Nokia 3310.

Wyświetlacz graficzny opisywany w artykule ma rozdzielczość 84×48 piksele oraz 1-bitową „skalę szarości" – dany punkt może być zaczerniony lub przezroczysty. Te wyświetlacze są również dostępne w formie płytek – modułów ułatwiających ich aplikację, gotowych do współpracy z zestawami ewaluacyjnymi, takimi jak Arduino lub Nucleo. Moduł można zasilać oraz sterować nim pomocą napięcia 3,3 V lub 5 V. Moduły są oferowane w dwóch wariantach – z podświetleniem jasnoszarym lub niebieskim. Można je kupić w wielu sklepach internetowych. W omawianym module zastosowano kontroler LCD typu PCD8544.

Kontroler LCD typu PCD8544

Układ PCD8544 komunikuje się z systemem nadrzędnym za pomocą interfejsu SPI. Przesyłamy nim polecenia sterujące i konfigurujące sterownik oraz dane, które chcemy wyświetlić na wyświetlaczu. Sterownik ma pamięć RAM o pojemności odpowiadającej liczbie pikseli matrycy LCD – 84×48 bitów. Jest w niej przechowywany stan każdego punktu – „zaświecony" bądź „zgaszony". Sterownik nie ma wbudowanych czcionek, więc jest jedynie prostym buforem ramki – przesyłamy do niego mapę bitową pikseli matrycy. Jest ona zapisywana w pamięci RAM, a następnie ustawiana na matrycy poprzez wygenerowanie odpowiednich sygnałów sterujących przez sterownik.

Wyprowadzenia

Sterownik ma wyprowadzenia o następujących oznaczeniach: RST, DC, CE, DIN oraz CLK, do których dochodzi jeszcze wyprowadzenie modułu – pin BL oraz piny zasilania i masy:

- Wyprowadzenie RST odpowiada za zerowanie układu. Podanie na ten pin napięcia odpowiadającego poziomowi niskiemu, a następnie wysokiemu powoduje wyzerowanie zawartości wszystkich rejestrów sterownika i przywrócenie konfiguracji domyślnej. Nie powoduje to jednak wyczyszczenia zawartości pamięci RAM – ta po uruchomieniu może być wypełniona niezerowymi wartościami. Jest to obowiązkowa operacja po uruchomieniu układu, przed przystąpieniem do jego wykorzystania.
- Poziom logiczny na wejściu DC decyduje, czy transmitowany sygnał jest instrukcją sterującą, czy danymi do umieszczenia w pamięci RAM i w konsekwencji – wyświetlenia na ekranie. Do transmisji poleceń zerujemy wejście DC, a do transmisji danych – ustawiamy.
- Pin CE – Clock Enable, blokuje i odblokowuje możliwość transmisji danych interfejsem SPI. Aby transmisja mogła się odbyć, na czas jej trwania potrzebujemy ustawić wejście CE.
- DIN oraz CLK to piny interfejsu SPI. Pinem DIN odbywa się transmisja danych i poleceń sterujących. CLK to wejście zegarowe.
- Za pomocą pinu BL sterujemy podświetleniem wyświetlacza – ustawienie wejścia powoduje włączenie podświetlenia.

Oprócz wymienionych powyżej, układ sterownika ma jeszcze inne wyprowadzenia – testowe oraz sterujące. Nie są one wyprowadzone z modułu wyświetlacza, więc nie będziemy ich omawiać.

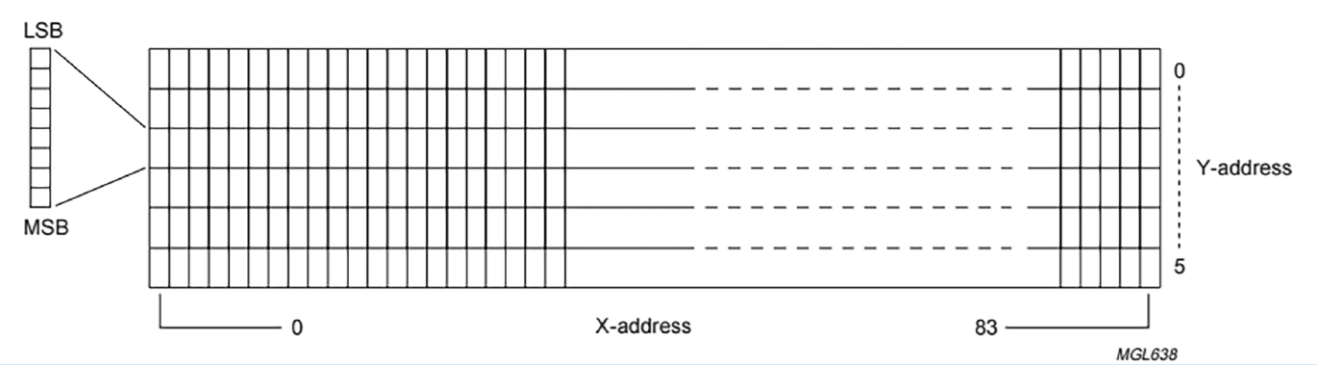
Polecenia sterujące i konfiguracja sterownika

Po restarcie wyświetlacza musimy go skonfigurować, tj. wybrać napięcie sygnału sterującego matrycą (co pośrednio powoduje ustalenie kontrastu), współczynnik bias (liczba progów napięcia sygnału sterującego), współczynnik kompensacji temperaturowej oraz tryb pracy wyświetlacza. Odbywa się to poprzez przesyłanie rozkazów, mieszczących się wraz z ustawianymi wartościami w poszczególnych bajtach danych przesyłanych

INSTRUCTION	D/$\overline{C}$	COMMAND BYTE								DESCRIPTION
		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
(H = 0 or 1)										
NOP	0	0	0	0	0	0	0	0	0	no operation
Function set	0	0	0	1	0	0	PD	V	H	power down control; entry mode; extended instruction set control (H)
Write data	1	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	writes data to display RAM
(H = 0)										
Reserved	0	0	0	0	0	0	1	X	X	do not use
Display control	0	0	0	0	0	1	D	0	E	sets display configuration
Reserved	0	0	0	0	1	X	X	X	X	do not use
Set Y address of RAM	0	0	1	0	0	0	Y_2	Y_1	Y_0	sets Y-address of RAM; $0 \le Y \le 5$
Set X address of RAM	0	1	X_6	X_5	X_4	X_3	X_2	X_1	X_0	sets X-address part of RAM; $0 \le X \le 83$
(H = 1)										
Reserved	0	0	0	0	0	0	0	0	1	do not use
	0	0	0	0	0	0	0	1	X	do not use
Temperature control	0	0	0	0	0	0	1	TC_1	TC_0	set Temperature Coefficient (TC_x)
Reserved	0	0	0	0	0	1	X	X	X	do not use
Bias system	0	0	0	0	1	0	BS_2	BS_1	BS_0	set Bias System (BS_x)
Reserved	0	0	1	X	X	X	X	X	X	do not use
Set V_{OP}	0	1	V_{OP6}	V_{OP5}	V_{OP4}	V_{OP3}	V_{OP2}	V_{OP1}	V_{OP0}	write V_{OP} to register

Rysunek 1. Zestaw instrukcji obsługiwanych przez sterownik wyświetlacza

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 39483, pass: 5ikr2a2ku



Rysunek 2. Struktura pamięci RAM sterownika Philips PCD8544

interfejsem SPI. Dane te transmitujemy z maksymalną szybkością do 4.0 Mbit/s.

Na **rysunku 1** pokazano tabelę zaczerpniętą z dokumentacji sterownika. Znajduje się w niej spis rozkazów obsługiwanych przez sterownik wraz z ich strukturą binarną. Polecenia podzielono na dwie grupy – zestaw poleceń rozszerzonych (H=1) oraz podstawowych (H=0). Za pomocą poleceń rozszerzonych konfigurujemy sterownik do pracy z konkretną matrycą. Polecenia podstawowe dotyczą trybu wyświetlania, sposobu transmisji danych do wyświetlacza oraz położenia „kursora". Są też częściej wykorzystywane w trakcie pracy z wyświetlaczem.

Aby wejść w tryb poleceń rozszerzonych i skonfigurować wyświetlacz, musimy nadać interfejsem SPI ciąg bitów „00100001" (0x21), aby później przejść do trybu poleceń podstawowych – „00100000" (0x20). Wyzerowane parametry PD oraz V występujące odpowiednio na bicie nr 3 i 2 (licząc od prawej) odpowiadają odpowiednio za włączenie chipu oraz ustawienie kierunku adresacji (o tym później).

Będąc w trybie poleceń rozszerzonych, ustawiamy kolejno:

- **Napięcie sygnału sterującego (kontrast).** Napięcie sygnału sterującego jest zależne od wykorzystywanej matrycy i powinno być odpowiednio do niej dobrane. Im wyższa wartość, tym wyższy będzie kontrast wyświetlanych elementów, nie możemy jednak zwiększyć jej za bardzo – mogłoby to spowodować uszkodzenie matrycy. Napięcie ustawiamy poleceniem „1XXXXXXX", gdzie „XXXXXXX" to liczba binarna, którą podstawiamy do wzoru „3,06+XXXXXXX*0,06 V". Dysponujemy zakresem od 3,06 V do 10,74 V. Optymalna wartość dla opisywanego modułu to 5,34 V, a ustawiające ją polecenie sterujące to: „11001000" (hex 0xB8).
- **Współczynnik kompensacji temperaturowej.** W zależności od temperatury matrycy zmienia się kontrast wyświetlanych na niej elementów. W niższej temperaturze kontrast jest niższy – ciekłe kryształy nie poruszają się tak „żwawo", jak w wyższej temperaturze. Aby to skompensować, sterownik wyświetlacza może zwiększyć napięcie sterujące i zarazem kontrast przy niższych temperaturach oraz zmniejszyć napięcie sterujące przy wyższych. Do ustawienia współczynnika kompensacji temperaturowej służy polecenie „000001XX", gdzie pod „XX", podstawiamy wartość współczynnika z zakresu od „00" do „11" binarnie. Dla omawianego modułu efekt ten nie jest zauważalny, możemy więc spokojnie wybrać wartość „00".
- **Współczynnik bias.** Aby pobudzić ciekły kryształ do skręcenia, potrzebujemy przepuścić przez niego prąd przemienny. Im więcej pikseli/obszarów obsługiwanych jest przez pojedynczą elektrodę sterującą ciekłym kryształem (w parze z innymi elektrodami po przeciwnej stronie matrycy), tym więcej poziomów napięć powinien mieć nasz sygnał sterujący. Współczynnik bias decyduje o liczbie tych poziomów. Zmieniamy go poleceniem „00010XXX". Jego optymalna wartość dla tej liczby pikseli przypadających na elektrodę to 4 (dokładniej 1/8, bo wartość ta podstawiana jest do wzoru „1/n+4"), czyli „100" binarnie. Gotowy rozkaz konfiguracyjny przyjmie postać: „00010100" (0x14).

Po powrocie do trybu poleceń podstawowych, poleceniem „00001D0E" musimy jeszcze wybrać tryb wyświetlania elementów:

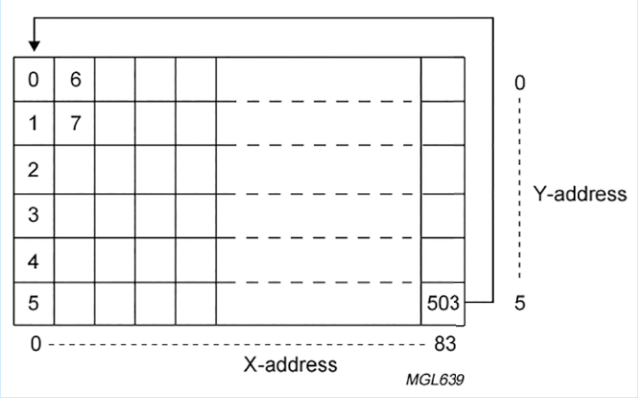
- D=0, E=0, polecenie: „00001000" (hex 0x08) – w tym trybie wszystkie piksele wyświetlacza będą zgaszone, tj. będą przepuszczały światło.
- D=1, E=0, polecenie: „00001100" (hex 0x0C) – tryb normalny – jedynka w mapie bitowej odpowiadać będzie czarnemu punktowi na wyświetlaczu, a zero spowoduje, że ten punkt będzie przepuszczał światło.
- D=1, E=1, polecenie: „00001101" (hex 0x0D) – tryb odwrócony – jedynka odpowiada punktowi przepuszczającemu światło, a zero – blokującemu je.
- D=0, E=1, polecenie: „00001001" (hex 0x09) – w tym trybie wszystkie piksele wyświetlacza będą ustawione, tj. będą blokowały światło.

Adresowanie i transmisja danych

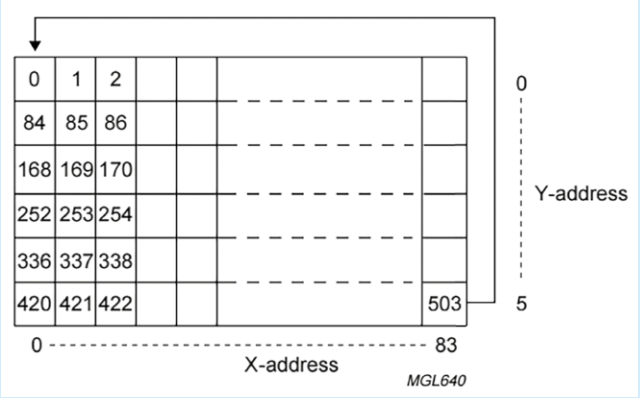
Dane przesyłane do sterownika trafiają do pamięci RAM (**rysunek 2**), a następnie „na matrycę". Pamięć podzielono na 6 segmentów, które wybieramy, ustawiając adres poziomy (liczbę z zakresu 0…5) oraz pionowy, tj. liczbę z zakresu 0…83. Transmitując bajt danych (8 bitów), przesyłamy mapę bitową paska o wysokości 8 pikseli, umieszczanego pod wskazanym adresem oraz rysowanego w odpowiadającym mu miejscu na wyświetlaczu. Każdy bit, odpowiada jednemu pikselowi – logiczna „1" powoduje, że piksel będzie zaczerniony, a logiczne „0", że przezroczysty. Najbardziej znaczący bit tego bajtu danych opisuje skrajnie dolny punkt paska.

Aby po przesłaniu każdego bajta nie była konieczna zmiana adresu, zdefiniowano dwa tryby adresacji – wertykalny (pionowy) i horyzontalny (poziomy). Po przetransmitowaniu każdego bajtu jest inkrementowany adres poziomy lub pionowy. W trybie horyzontalnym kolejne „paski" będą umieszczane po prawej względem uprzednio przesłanego. Gdy skończy się linia, to „pasek" zostanie umieszczony w pierwszej kolumnie następnego segmentu lub jeśli był to koniec ostatniego segmentu, w pierwszej kolumnie pierwszego segmentu. W trybie wertykalnym kolejne paski umieszczane są pod poprzednimi. Gdy skończy się kolumna, kolejny pasek trafi do pierwszego wiersza kolejnej kolumny lub pierwszego wiersza pierwszej kolumny. Dokładniej zobrazowano to na **rysunkach 3** i **4**.

Adresację pionową, będąc w trybie poleceń podstawowych, wybieramy poleceniem „00100010" (hex 0x22), poziomą – „00100000" (hex 0x20). Poleceniami „1XXXXXXX" oraz



Rysunek 3. Kolejność wypełniania komórek pamięci RAM i pikseli na ekranie przy adresacji wertykalnej



Rysunek 4. Kolejność wypełniania komórek pamięci RAM i pikseli na ekranie przy adresacji horyzontalnej

„01000YYY”, podstawiając za „XXXXXXX” oraz „YYY” odpowiednio nr kolumny i numer segmentu, wybieramy konkretny adres. Aby teraz wyświetlić na wyświetlaczu mapę bitową, przesyłamy do niego kolejne bajty, w identyczny sposób jak polecenia, jednak wcześniej ustawiamy pin DC.

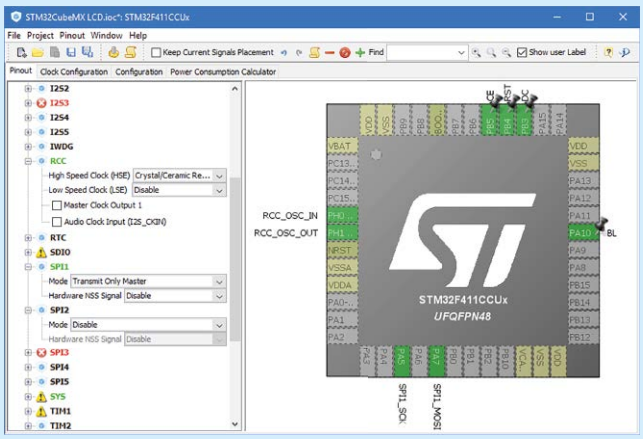
Pierwszy projekt – szachownica

Po wyjaśnieniu teorii działania i sterowania wyświetlaczem, przejdźmy do części praktycznej i spróbujemy wyświetlić na nim jakiś obraz, np. szachownicę. Uruchamiamy narzędzie STM32CubeMX i w kreatorze wyboru mikrokontrolera wybieramy posiadany przez nas układ. Dla przypomnienia, na używanej przeze mnie, podczas tworzenia tego kursu, płytce rozwojowej KA–NUCLEO–F411CE jest mikrokontroler STM32F411CEU6 (**rysunek 5**).

Następnie, na pierwszej planszy kreatora, zdecydować musimy, jakie wyprowadzenia procesora połączone zostaną z jakimi wyprowadzeniami modułu wyświetlacza. Potrzebujemy także uruchomić interfejs SPI oraz zdefiniować typ zastosowanego źródła częstotliwości taktowania układu. Nie możemy też oczywiście zapomnieć o podłączeniu modułu wyświetlacza do mikroprocesora.

Na **fotografii 6** pokazano wyprowadzenia modułu wyświetlacza. Ich znaczenie zostało już wyjaśnione wcześniej. Teraz potrzebujemy jedynie pamiętać, że wyprowadzenia procesora przyłączone do pinów BL, CE, RST i DC powinny być skonfigurowane jako „GPIO Output”. Piny DIN i CLK to wyprowadzenia interfejsu SPI – odpowiednio MISO oraz SCK.

Aby ustawić wyprowadzenie procesora w tryb pracy „GPIO Output”, klikamy prawym przyciskiem na jego rysunek i z listy wybieramy stosowną pozycję. Polecam przy okazji nadać pinom nazwy, takie jak na rysunku 5. Ułatwi to ich późniejszą identyfikację oraz użycie w kodzie programu. Interfejs SPI



Rysunek 5. Konfiguracja wyprowadzeń w STM32CubeMX

Operacje logiczne w C

W kodzie funkcji wysyłających do wyświetlacza polecenia sterujące wykorzystana została operacja logicznej sumy – „LICZBA_NR_1 | LICZBA_NR_2”. Co powoduje wykonanie takiego kodu? Po „zamianie”* obu liczb na postać binarną operacja ta wykonywana jest osobno na każdej parze bitów, pochodzących z obu liczb i będących na tych samych pozycjach, licząc od prawej. Jeśli choć jeden z nich jest jedynką, w wyniku, na tej samej pozycji, zapiana zostanie jedynka. Istnieje również podobna operacja logicznej koniunkcji. Jeśli oba bity będą takie same (zero i zero lub jedynka i jedynka), wynik przyjmie wartość jedynki. Definiujemy ją symbolem „LICZBA_NR_1 & LICZBA_NR_1”. Możemy także zanegować wszystkie bity w danej licznie, tj. zamienić jedynki na zera i zera na jedynki, poprzedzając liczbę symbolem „~”.

konfigurujemy, rozwijając dowolny interfejs „SPIx” na liście po lewej stronie okna programu i z listy rozwijanej „Mode” wybierając opcję „Transmit Only Master”. Dokładne znaczenie tej opcji przedstawione zostało w poprzedniej części serii. Jeśli nasza płytka rozwojowa lub prototypowa ma oscylator kwarcowy (tak jak KA–NUCLEO–F411CE), włączamy jego użycie poprzez rozwinięcie, na liście po lewej stronie okna, pozycji „RCC” i ustawienie, w polu „High Speed Clock (HSE)”, wartości „Crystal/Ceramic Resonator”.

Przedstawiony na fotografii 6 moduł ma inne oznaczenia pinów niż posiadany przeze mnie – moduły te występują w wielu różnych wersjach, produkowanych przez różnych producentów. Pin BL oznaczono LCD, CE – SCE, a CLK jako SCLK.

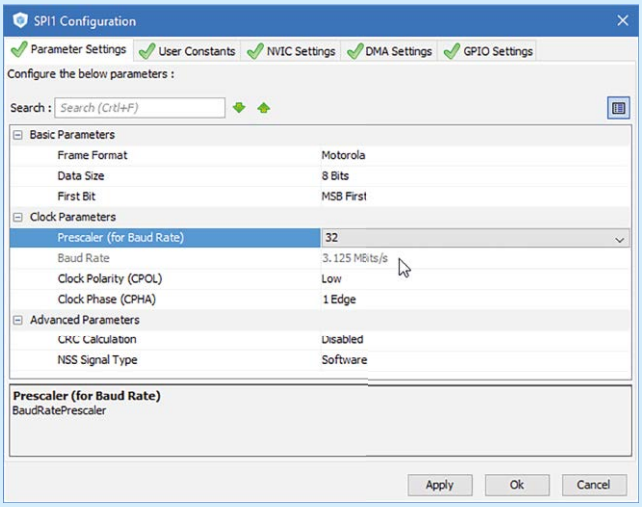
Jeśli korzystamy z płytki rozwojowej KA–NUCLEO, możemy użyć tych samych wyprowadzeń procesora, które przedstawiono na rysunku 5 oraz tego sam interfejsu SPI1. Należy wtedy połączyć piny modułu wyświetlacza: RST, CE, DC, DIN, CLK, VCC, BL, GND, odpowiednio do pinów płytki: D5, D4, D3, D11, D13, 3,3V , D2, GND.

Na kolejnej planszy kreatora – „Clock Configuration”, podobnie jak w poprzednich częściach, konfigurujemy „rozchodzenie się” po układzie mikrokontrolera sygnału taktującego. Dokładny opis schematu przedstawiony został w pierwszej części kursu. Jeśli używana przez nas płytka ewaluacyjna ma oscylator kwarcowy, ustawiamy jego częstotliwość oraz wybieramy go jako źródło sygnału częstotliwości – w polu „Input frequency” podajemy jego częstotliwość (na płytce KA–NUCLEO jest to 8 MHz), w polu „PLL Source Mux” wybieramy opcję

Fotografia 6. Wyprowadzenia modułu wyświetlacza (źródło: sklep internetowy Kamami)

Listing 1. Plik nagłówkowy *display.h*

```
#ifndef display_header
#define display_header

#include „stm32f4xx_hal.h”
#include „spi.h”
#include „gpio.h”

enum display_direction {horizontal, vertical};

struct display_config
{
    SPI_HandleTypeDef * spi;
    GPIO_TypeDef * reset_port;  uint16_t reset_pin;
    GPIO_TypeDef * bl_port;     uint16_t bl_pin;
    GPIO_TypeDef * dc_port;     uint16_t ce_pin;
    GPIO_TypeDef * ce_port;     uint16_t dc_pin;
};

void display_setup(struct display_config * cfg);
void display_set_dxy(struct display_config * cfg,
                     enum display_direction direction,
                     uint8_t column, uint8_t row);
void display_write_cmd(struct display_config * cfg, uint8_t cmd);
void display_write_data(struct display_config * cfg, uint8_t data);

#endif
```



Rysunek 7. Konfigurowanie szybkości pracy interfejsu SPI

„HSE”, a w polu „System Clock Mux” – „PLLCLK”. W pole „HCLK” wpisujemy natomiast maksymalną dozwoloną wartość.

Dalej przechodzimy do zakładki „Configuration” i z pola „Connectivity” wybieramy pozycję „SPIx”. Opcja, która nas tutaj interesuje, to „Prescaler”. Musimy dobrać go tak, aby wartość w polu „Baud Rate” była możliwie wysoka, ale nie przekraczała 4 MBit/s – maksymalnej częstotliwości obsługiwanej przez układ wyświetlacza (**rysunek 7**).

Teraz już możemy wygenerować kod projektu i zaimportować go w środowisku IDE. Klikamy w ikonę zębatki – podajemy nazwę projektu i ścieżkę dostępu do miejsca, gdzie ma on zostać zapisany, z pola „Toolchain / IDE”, wybieramy pozycję „SW4STM32”, w zakładce „Code Generator” zaznaczamy opcję „Generate peripherial initialization as pair of ‚.c/.h’...” i klikamy „OK”. Otwieramy środowisko System Workbench for STM32, zamykamy w nim planszę powitalną i w panelu po lewej stronie okna klikamy prawym przyciskiem myszy, z menu kontekstowego wybieramy opcję „Import...” –> „Existing Projects into Workspace” –> „Browse...”, nawigujemy do folderu projektu i klikamy „Finish”.

Dodajemy do nowego projektu 2 pliki – do folderu „Inc”: „display.h” oraz do folderu „Src”: „display.c”. Aby to zrobić, klikamy PPM na folder nadrzędny i z menu wybieramy opcje „New” –> „File”, podajemy nazwę tworzonego pliku oraz klikamy przycisk „Finish”. Następnie oba pliki wypełniamy zawartością, odpowiednio z **listingów 1** i **2**, modyfikujemy plik main.c zgodnie z **listingiem 3** oraz kompilujemy kod i uruchamiamy go na mikrokontrolerze – ikony młotka i robaka na pasku narzędziowym.

Podobnie jak w poprzedniej części, funkcje służące do obsługi wyświetlacza wydzielone zostały do osobnych plików. Plik „.h” jest plikiem nagłówkowym – przechowuje on definicje typów danych i struktur, a także prototypy funkcji. Zawartość funkcji umieszczona jest w pliku „.c”. Struktura „display_config” przechowuje informacje nt. używanych wyprowadzeń i ich portów oraz interfejsu SPI. Musimy ją utworzyć i wypełnić przed wykorzystaniem jakiejkolwiek funkcji „display_...()”. Utworzenie i wypełnienie struktury ma miejsce

Listing 2. Plik biblioteki funkcji *display.c*

```
#include „display.h”

void display_setup(struct display_config * cfg)
{
    // Sprzętowy reset wyświetlacza
    HAL_GPIO_WritePin(cfg->reset_port, cfg->reset_pin, GPIO_PIN_RESET);
    HAL_Delay(100);
    HAL_GPIO_WritePin(cfg->reset_port, cfg->reset_pin, GPIO_PIN_SET);
    // Włączenie podświetlenia
    HAL_GPIO_WritePin(cfg->bl_port, cfg->bl_pin, GPIO_PIN_SET);
    // Przejście w tryb poleceń rozszerzonych
    // 0x00100001 (0x21) - polecenie
    display_write_cmd(cfg, 0x21);
    // Temperature Coefficient
    // 0b00000100 (0x04) - polecenie | 0b00000000 (0x00) - wartość
    // = 0b00000100 (0x04)
    display_write_cmd(cfg, 0x04);
    // Bias System
    // 0b00010000 (0x10) - polecenie | 0x00000100 (0x04) - wartość
    // = 0b00010100 (0x14)
    display_write_cmd(cfg, 0x10 | 0x04);
    // Ustawienie napięcia sygnału sterującego matrycą LCD / kontrastu
    // 0b10000000 (0x80) - polecenie | 0x01001000 (0x38) - wartość
    // = 0x11001000 (0xB8)
    display_write_cmd(cfg, 0x80 | 0x38);
    // Powrót do trybu poleceń podstawowych
    // 0x00100001 (0x20) - polecenie
    display_write_cmd(cfg, 0x20);
    // Ustawienie trybu pracy wyświetlacza - normalnego
    // 0x00001000 (0x08) - polecenie | 0b00000100 (0x04) - wartość
    // = 0b00001100 (0x0C)
    // Pozostałe tryby pracy: cały wyłączony - 0b00000000 (0x00);
    // cały zapalony - 0x00000001 (0x01); odwrócony - 0x00000101 (0x05)
    display_write_cmd(cfg, 0x08 | 0x04);
    // Czyszczenie wyświetlacza
    for (int i = 0; i < 504; i++)
        display_write_data(cfg, 0x00);
}

void display_set_dxy(struct display_config * cfg,
                     enum display_direction direction,
                     uint8_t column, uint8_t row)
{
    // Przejście w tryb poleceń podstawowych i ustawienie kierunku rysowania
    if(direction == vertical)
        display_write_cmd(cfg, 0x20 | 0x02);
    else
        display_write_cmd(cfg, 0x20 | 0x00);
    display_write_cmd(cfg, 0x80 | column); // Wybór kolumny
    display_write_cmd(cfg, 0x40 | row); // Wybór wiersza
}

void display_write_cmd(struct display_config * cfg, uint8_t cmd) {
    // Wybór tryby transmisji poleceń
    HAL_GPIO_WritePin(cfg->dc_port, cfg->dc_pin, GPIO_PIN_RESET);
    // Odblokowanie wejścia zegarowego
    HAL_GPIO_WritePin(cfg->ce_port, cfg->ce_pin, GPIO_PIN_RESET);
    // Transmisja danych interfejsem SPI
    HAL_SPI_Transmit(cfg->spi, &cmd, 1, 100);
    // Blokada wejścia zegarowego
    HAL_GPIO_WritePin(cfg->ce_port, cfg->ce_pin, GPIO_PIN_SET);
}

void display_write_data(struct display_config * cfg, uint8_t data) {
    // Wybór tryby transmisji poleceń
    HAL_GPIO_WritePin(cfg->dc_port, cfg->dc_pin, GPIO_PIN_SET);
    // Odblokowanie wejścia zegarowego
    HAL_GPIO_WritePin(cfg->ce_port, cfg->ce_pin, GPIO_PIN_RESET);
    // Transmisja danych interfejsem SPI
    HAL_SPI_Transmit(cfg->spi, &data, 1, 100);
    // Blokada wejścia zegarowego
    HAL_GPIO_WritePin(cfg->ce_port, cfg->ce_pin, GPIO_PIN_SET);
}
```

Listing 3. Plik programu głównego *main.c*

```
[...]
/* USER CODE BEGIN Includes */
#include „display.h”
/* USER CODE END Includes */
[...]

int main(void)
{
    [...]
    /* USER CODE BEGIN 2 */
    struct display_config cfg;
    cfg.spi = &hspi1;
    cfg.reset_port = RST_GPIO_Port;
    cfg.reset_pin = RST_Pin;
    cfg.bl_port = BL_GPIO_Port;
    cfg.bl_pin = BL_Pin;
    cfg.dc_port = DC_GPIO_Port;
    cfg.dc_pin = DC_Pin;
    cfg.ce_port = CE_GPIO_Port;
    cfg.ce_pin = CE_Pin;

    display_setup(&cfg);
    display_set_dxy(&cfg, horizontal, 0, 0);
    for (uint8_t row = 0; row < 6; row++) {
        for (uint8_t col = 0; col < 10; col++) {
            for (uint8_t j = 0; j < 4; j++) display_write_data(&cfg, 0x0F);
                for (uint8_t j = 0; j < 4; j++) display_write_data(&cfg, 0xF0);
        }
        for (uint8_t j = 0; j < 4; j++) display_write_data(&cfg, 0x0F);
    }
    /* USER CODE END 2 */
    [...]
}
[...]
```

w pliku „main.c”, w sekcji „USER CODE 2”. Następnie wywoływane są funkcje display_setup() i display_set_dxy(). Pierwsza z nich, konfiguruje sterownik do pracy zgodnie z sekwencją przedstawioną powyżej. Druga ustala kierunek rysowania oraz pozycje „kursora”. Dalej, w pętli, rysowany jest wzór szachownicy, poprzez umieszczanie w pamięci RAM wyświetlacza czterech pionowych pasków „11110000” (hex 0xF0) na przemian z czterema paskami „00001111” (hex 0x0F). W podobny sposób możliwe byłoby wyświetlenie pliku graficznego – należałoby zamienić go na postać 1-bitowej bitmapy o wymiarach 84×48 pikseli i za pomocą darmowego programu LCDAssistant zamienić na postać tablicy bajtów pionowych pasków, następnie można by przekopiować w pętli całą tablicę do pamięci RAM wyświetlacza, korzystając z funkcji display_write_data().

Listing 4. Dodanie bufora znaków w pliku *display.h*

```
#ifndef display_header
#define display_header

[...]
static const uint8_t display_font[][5] =
{
    { 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x5f, 0x00, 0x00 },
    { 0x00, 0x07, 0x00, 0x07, 0x00 },
    [...] // Pełny kod dostępny do pobrania z serwera FTP
    { 0x00, 0x41, 0x36, 0x08, 0x00 },
    { 0x10, 0x08, 0x08, 0x10, 0x08 },
    { 0x78, 0x46, 0x41, 0x46, 0x78 }
};
// Powyższa czcionka, autorstwa Pascala Stanga, pochodzi
// z biblioteki GLCD, dostępnej na licencji BSD. Źródło:
// https://github.com/andygock/glcd/blob/master/fonts/font5x7.h
[...]
void display_write_char(struct display_config * cfg, char chr);
void display_rewrite_buffer(struct display_config * cfg);
void display_clear_buffer(struct display_config * cfg);
[...]
#endif
```

Listing 5. Definicje funkcji wyświetlających znaki w *display.c*

```
[...]
void display_write_char(struct display_config * cfg, char chr)
{
    for (uint8_t i = 0; i < 5; i++)
        display_write_data(cfg, display_font[chr - 0x20][i]);
    display_write_data(cfg, 0x00);
}

void display_rewrite_buffer(struct display_config * cfg)
{
    display_set_dxy(cfg, horizontal, 0, 0);
    for(uint8_t i=0; i<5; i++)
        for(uint8_t j=0; j<14; j++)
            display_write_char(cfg, cfg->buffer[i][j]);
}

void display_clear_buffer(struct display_config * cfg)
{
    for(uint8_t i=0; i<6; i++)
        for(uint8_t j=0; j<14; j++)
            cfg->buffer[i][j] = ' ';
}
```

Drugi projekt – wyświetlanie tekstu

Oprócz wyświetlania bitmap, bardzo często potrzebujemy prezentować na wyświetlaczu litery i cyfry. Rozbudujmy więc nasz projekt, aby to umożliwić. Musimy dodać czcionkę i funkcję wypisującą na wyświetlaczu kolejne znaki. Dodatkowo wyposażymy mikrokontroler w bufor znaków i to do niego zapisywać będziemy tekst przed wyświetleniem.

Do plików „display.h” i „display.c” dopisujemy kod umieszczony na **listingach 4** i **5**. Plik „main.c” modyfikujemy zgodnie z **listingiem 6**.

W pliku „display.h”, do definicji struktury display_config, dodajemy dwuwymiarową tablicę – bufor znaków. Umieszczamy tam także: tablicę map bitowych poszczególnych znaków – czcionkę oraz prototypy dodatkowych funkcji rysujących zawartość bufora znaków oraz poszczególne znaki. Do pliku „display.c” dodajemy definicje tych funkcji. Modyfikujemy także sekcję „USER CODE 2”, dodając do niej wywołanie nowych funkcji. Od teraz tekst, który chcemy wyświetlić na wyświetlaczu, umieszczamy w dwuwymiarowej tablicy „cfg.buffer[y][x]” – gdzie y i x to współrzędne, odpowiednio pionowe i poziome, znaku na wyświetlaczu. Najłatwiej zrobić to za pomocą funkcji memcpy() – kopiującej fragmenty pamięci z podanego adresu pod podany adres. Kolejne jej parametry to docelowy adres miejsca w pamięci, źródłowy adres oraz długość, w bajtach, kopiowanych danych. Aby wydobyć adres danej zmiennej lub komórki tablicy z pamięci, poprzedzamy jej nazwę znakiem „&”.

Powyższy przykład możemy rozbudować i np., korzystając z funkcji sprintf() i snprintf(), zapisywać do bufora i wyświetlać na wyświetlaczu odczytywane parametry stało i zmiennoprzecinkowe z różnych czujników czy też zbudować proste menu obsługiwane przyciskami. Przydatne może być umieszczenie wywoływania funkcji przerysowującej zawartość wyświetlacza, na podstawie bufora znaków, w funkcji obsługi przerwania zegara.

Aleksander Kurczyk

Listing 6. Modyfikacja programu głównego *main.c*

```
[...]
/* USER CODE BEGIN Includes */
#include „string.h”
#include „display.h”
/* USER CODE END Includes */
[...]
int main(void)
{
    [...]
    /* USER CODE BEGIN 2 */
    struct display_config cfg;
    cfg.spi = &hspi1;
    cfg.reset_port = RST_GPIO_Port;
    cfg.reset_pin = RST_Pin;
    cfg.bl_port = BL_GPIO_Port;
    cfg.bl_pin = BL_Pin;
    cfg.dc_port = DC_GPIO_Port;
    cfg.dc_pin = DC_Pin;
    cfg.ce_port = CE_GPIO_Port;
    cfg.ce_pin = CE_Pin;

    display_setup(&cfg);
    display_clear_buffer(&cfg);
    char hw[] = „HELLO WORLD!”;
    memcpy(&(cfg.buffer[2][1]), hw, strlen(hw));
    display_rewrite_buffer(&cfg);
    /* USER CODE END 2 */
    [...]
}
[...]
```

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 39483, pass: 5ikr2a2ku