# STM32 – sprzętowe sterowanie multipleksowanego wyświetlacza LED

**Multipleksowane wyświetlacze 7-segmentowe należą do najbardziej popularnych elementów urządzeń mikroprocesorowych służących do komunikacji z użytkownikiem. Istnieje wiele sposobów przyłączania takich wyświetlaczy do mikrokontrolera. Może do tego posłużyć np. gotowy układ sterownika wyświetlacza, dołączony do interfejsu SPI lub I²C mikrokontrolera. Najtańszym sposobem jest sterowanie wyświetlacza przez mikrokontroler. Zaletą takiego rozwiązania, oprócz niskiego kosztu, jest również możliwość uzyskania efektów optycznych niedostępnych w standardowych układach sterowników, w tym płynnej zmiany jasności stosownie do natężenia oświetlenia zewnętrznego oraz „miękkiej" zmiany zawartości wyświetlacza.**

W zależności od wymaganej jasności świecenia i natężenia prądów sterujących wyświetlacz, może być niezbędne użycie układów wzmacniających w postaci kluczy tranzystorowych bipolarnych lub MOSFET albo specjalizowanych układów scalonych. Zazwyczaj stosuje się klucze dla wspólnych elektrod cyfr. Jeśli wymagane natężenie prądu poszczególnych segmentów przekracza dopuszczalną obciążalność prądową wyjść mikrokontrolera, należy również zastosować wzmacniacze segmentów.

Wyświetlanie multipleksowane zazwyczaj polega na kolejnym, naprzemiennym wyświetlaniu poszczególnych cyfr z taką częstotliwością, aby obserwator nie zauważył migotania. Poprawnie zaprojektowane wyświetlacze są odświeżane z częstotliwością od ok. 120 Hz do 2 kHz. Większa częstotliwość odświeżania ogranicza niepożądane efekty wizualne, zwłaszcza, gdy obserwator przemieszcza się względem wyświetlacza. Zbyt mała częstotliwość odświeżania powoduje, że obserwator ma wrażenie falowania lub drżenia obrazu. Jeżeli dopuszczamy możliwość ruchu obserwatora względem wyświetlacza, częstotliwość odświeżania nie powinna być mniejsza od 400 Hz. Górna wartość częstotliwości odświeżania jest uwarunkowana parametrami czasowymi elementów przełączających oraz – przy programowym sterowaniu wyświetlaniem – zajętością czasu procesora.

## Zasady programowego sterowania wyświetlacza

Programowa obsługa odświeżania wyświetlacza jest powszechnie znana i stosowana, jednak warto zwrócić uwagę na zasady jej poprawnej realizacji.

W celu utrzymania stałej i jednakowej jasności cyfr przełączanie cyfr musi zachodzić w stałych odstępach czasu, wyznaczanych przez sprzętowy timer dostępny w mikrokontrolerze. W typowych rozwiązaniach zmianami sterowania wyświetlacza zajmuje się fragment programu stanowiący część obsługi przerwania timera. Należy zauważyć, że wymagana częstotliwość przerwań timera jest iloczynem częstotliwości odświeżania wyświetlacza i liczby cyfr. Proste odświeżanie 4-cyfrowego wyświetlacza z częstotliwością 400 Hz wymaga zgłaszania przerwań z częstotliwością 1600 Hz.

Zawartość wyświetlacza powinna być przygotowana przez oprogramowanie i przechowywana w wektorze danych, którego poszczególne elementy odpowiadają wyświetlanym cyfrom. Informacja ta może mieć postać obrazu bitowego segmentów poszczególnych cyfr lub stanów wszystkich sygnałów sterujących wyświetlaczem (sterowania cyfr i segmentów). W ten sposób unikamy zbędnego przekodowania symboli na ich obrazy przy każdym wyświetleniu cyfry.

Przy odświeżaniu programowym linie sterujące segmentami i cyframi nie muszą należeć do jednego portu GPIO, chociaż takie rozwiązanie jest najwygodniejsze do obsługi programowej. Jeżeli sterowania segmentów nie są zgrupowane w jednym porcie, wyświetlany obraz może być przechowywany w postaci wektora struktur, których poszczególne pola zawierają stany wyjść portów, z których są wyprowadzone sygnały sterujące.

Podczas zmiany wyświetlanej cyfry musimy zadbać o zachowanie właściwych zależności czasowych pomiędzy sygnałami sterującymi. Jeżeli wszystkie sygnały sterujące wyświetlaczem nie pochodzą z jednego portu lub nie jest możliwa równoczesna zmiana stanu wszystkich sygnałów przez pojedynczy zapis danej do portu, należy zwrócić uwagę na poprawną kolejność poszczególnych zmian. W takim przypadku oprogramowanie realizujące odświeżanie musi wykonać kolejno trzy czynności:

- wyłączenie bieżącej cyfry,
- ustawienie obrazu nowej cyfry na wyjściach sterujących segmentami,
- włączenie nowej cyfry.

Częstym błędem popełnianym przez początkujących programistów jest pominięcie pierwszej z tych czynności lub niewłaściwa kolejność czynności. Skutkuje to błędnym wyświetlaniem „cienia" cyfry na sąsiedniej pozycji wyświetlacza.

Poszczególne czynności związane z wysterowaniem wyświetlacza wymagają selektywnej zmiany stanu wyjść, co w prostszych mikrokontrolerach realizuje się poprzez operacje logiczne na rejestrach wyjściowych portów. W przypadku rodziny STM32 można i należy użyć mechanizmu modyfikacji wybranych linii portów, dostępnego poprzez rejestry BSRR i BRR portów GPIO. W ten sposób możemy dowolnie zmienić stan grupy wyjść pojedynczą operacją zapisu bez narażania się na problemy wynikające z operacji logicznych na portach.

Poprzednie części kursu i dodatkowe materiały dostępne są na stronie www.media.avt.pl

Przy takim rozwiązaniu sterowania wyświetlaczem wygodnie jest, gdy oprogramowanie przygotowuje dane do wyświetlenia w postaci wartości przesyłanych do rejestru BSRR. Dane odpowiadające poszczególnym cyfrom są umieszczone w wektorze, którego zawartość jest cyklicznie przesyłana do rejestru BSRR portu używanego do sterowania wyświetlaczem.

## Przykładowy program

Do zademonstrowania technik sterowania wyświetlacza posłuży prosty program odliczający czas w sekundach i wyświetlający go w postaci liczby 4-cyfrowej na multipleksowanym wyświetlaczu LED. Program został napisany w dwóch wersjach, różniących się sposobem realizacji odświeżania.

Uruchomiony go na płytce L476RG-Nucleo, zawierającej mikrokontroler STM32L476RGT. Do płytki dołączono moduł KA-NUCLEO-MULTISENSOR zawierający m.in. multipleksowany, 4-cyfrowy wyświetlacz LED. Ze względu na planowane użycie modułu w warunkach laboratoryjnych i wynikający stąd brak wymagania na dużą jasność, wyświetlacz jest sterowany z wyjść mikrokontrolera, bez użycia wzmacniaczy segmentów ani kluczy tranzystorowych cyfr. Wspólne elektrody cyfr są sterowane z linii portów mikrokontrolera bezpośrednio, a segmenty – poprzez rezystory ograniczające natężenie prądu do wartości ok. 5 mA. Wyświetlacz podłączono w taki sposób, że wszystkie linie sterujące nim należą do portu GPIOC; linie PC0…PC7 sterują świeceniem segmentów, a PC8…PC11 – wyborem cyfr.

Każda wersja programu składa się z dwóch plików źródłowych. Korzystają one z kilku plików nagłówkowych. Plik 7seg_common.c, wspólny dla obu wersji zawiera dwie procedury. Pierwsza z nich, *common_init()* jest wywoływana przy starcie i służy do inicjowania zegara mikrokontrolera i portu sterującego wyświetlaczem. Druga, *run_every_10ms()* ma za zadanie aktualizację wyświetlanego czasu i reagowanie na naciśnięcie przycisku. Zgodnie z nazwą powinna ona być wywoływana co 10 ms przez procedurę obsługi przerwania timera.

Pliki *7seg-dma.c-soft.c* i *7seg-dma.c* zawierają części oprogramowania specyficzne dla obu wersji projektu. Zostały one opisane w dalszej części artykułu.

Plik *board.h* zawiera definicje zasobów płytki, w tym linii portów, do których jest podłączony wyświetlacz. Zastosowany w pliku sposób zdefiniowania sterowań wyświetlacza umożliwia użycie tego samego kodu źródłowego niezależnie od polaryzacji sygnałów sterujących segmentów i cyfr. Ponadto, w skład projektu wchodzą trzy własne pliki zawierające definicje przydatne przy programowaniu mikrokontrolerów rodziny STM32. Stanowią one uzupełnienie pliku definicji zasobów mikrokontrolera dostarczonego przez producenta. Oba projekty są dostępne w pliku ep_l476_7seg.zip w materiałach dodatkowych do projektu.

## Inicjowanie mikrokontrolera

Mikrokontroler STM32L476 może być taktowany z jednego z kilku dostępnych źródeł przebiegu zegarowego, w tym trzech wewnętrznych generatorów RC o niskiej dokładności oraz generatorów wewnętrznych lub zewnętrznych synchronizowanych rezonatorami kwarcowymi. Ciekawą i rzadko spotykaną cechą dostępną w L476 jest możliwość synchronizacji wewnętrznego generatora RC o dużej częstotliwości przez generator używający „zegarkowego" rezonatora kwarcowego 32768 Hz. Taki właśnie sposób generowania przebiegu zegarowego wybrano dla projektów przykładowych. Jest to jedyny dostępny sposób na uzyskanie dokładnej częstotliwości roboczej mikrokontrolera na płytce Nucleo bez konieczności modyfikacji sprzętowej konfiguracji płytki.

Domyślnym źródłem przebiegu zegarowego aktywnym po zainicjowaniu mikrokontrolera jest generator wewnętrzny MSI o częstotliwości około 4 MHz. W celu uzyskania maksymalnej dozwolonej częstotliwości taktowania równej 80 MHz i synchronizacji generatora częstotliwością 32768 Hz należy kolejno:

- włączyć moduł PWR poprzez zapis do rejestru RCC → AHB2ENR i odblokować możliwość zapisu rejestru RCC → BDCR,
- włączyć generator LSE 32 kHz i poczekać na jego uruchomienie,
- włączyć synchronizację generatora MSI przebiegiem z generatora LSE,
- skonfigurować parametry głównej pętli PLL dla uzyskania częstotliwości 80 MHz z częstotliwości wejściowej 4 MHz, pochodzącej z MSI,
- skonfigurować parametry dostępu do pamięci Flash odpowiednie dla docelowej częstotliwości pracy
- poczekać na synchronizację PLL i włączyć taktowanie mikrokontrolera z PLL.

Wymienione powyżej czynności są wykonywane przez procedurę *common_setup()* z pliku *7seg-common.c*.

Ponieważ maksymalne dozwolone częstotliwości pracy wszystkich szyn wewnętrznych mikrokontrolera L476 wynoszą 80 MHz, nie ma potrzeby konfigurowania dzielników częstotliwości poszczególnych szyn – wszystkie peryferiale, w tym timery będą taktowane częstotliwością 80 MHz.

## Obsługa wyświetlacza w przerwaniu timera

Pierwsza wersja programu, zawarta w projekcie 7seg-soft (**listing 1**), obsługuje wyświetlacz programowo, w przerwaniu timera. Aby ograniczyć liczbę źródeł przerwań, w wersji tej użyto jednego przerwania timera zarówno do obsługi wyświetlania, jak i do odliczania czasu.

**Listing 1. Plik 7seg-soft.c - Obsługa wyświetlacza multipleksowanego w przerwaniu timera**

```
/* STM32L476 + Nucleo Companion Board
7-segment mpx display, soft refresh, brightness control
gbm, 01'2017 */

#include "board.h"
#include "7seg-common.h"

int main(void)
{
    common_setup();  // setup clock & GPIOC
    // LED multiplexing timer
    LEDMpx_TIM->PSC = SYSCLK_FREQ / MPX_FREQ / MPX_STEPS - 1;
    LEDMpx_TIM->ARR = MPX_STEPS - 1;
    LEDMpx_TIM->DIER = TIM_DIER_CC1IE | TIM_DIER_UIE;
    LEDMpx_TIM->CR1 = TIM_CR1_CEN;
    // interrupts and sleep control
    NVIC_EnableIRQ(LEDMpx_IRQn);
    SCB->SCR = SCB_SCR_SLEEPONEXIT_Msk; // sleep while not in handler
    __WFI();  // go to sleep
}

void LEDMpx_IRQHandler(void)
{
    uint32_t tim_sr = LEDMpx_TIM->SR;
    LEDMpx_TIM->SR = ~(TIM_SR_UIF | TIM_SR_CC1IF);

    if (tim_sr & TIM_SR_UIF)
    {
        static uint8_t dig;
        dig = (dig + 1) % NDIGITS;
        LEDMpx_PORT->BSRR = display[dig];
        static uint8_t tdiv;
        if (++ tdiv == MPX_FREQ / 100)
        {
            tdiv = 0;
            run_every_10ms();
        }
    }
    if (tim_sr & TIM_SR_CC1IF) LEDMpx_PORT->BSRR = DigAct(0) | SegAct(0);
}
```

## Konfiguracja timera

Do obsługi wyświetlacza użyto timera TIM4. Okres timera odpowiada czasowi świecenia pojedynczej cyfry. Dodatkowo kanał porównania CCR1 został użyty do regulacji jasności wyświetlacza. W celu zaprogramowania timera należy kolejno:

- włączyć taktowanie modułu timera w rejestrze APB1ENR1,
- ustawić preskaler (rejestr PSC) i okres timera (rejestr ARR),
- ustawić początkową jasność wyświetlacza (rejestr CCR1),
- włączyć generowanie przerwać na końcu okresu i przy porównaniu z CCR1 (rejestr DIER),
- uruchomić timer (rejestr CR1).

Wartości wszystkich parametrów potrzebnych do zaprogramowania timera zostały zdefiniowane w pliku *7seg-common.h* jako wyrażenia stałe rozwijane przez preprocesor języka C na podstawie dwóch stałych zdefiniowanych przez programistę: częstotliwości odświeżania wyświetlacza MPX_FREQ oraz liczby dostępnych poziomów jasności MPX_STEPS. Dla zapewnienia dokładnego odliczania czasu iloczyn tych stałych powinien być równocześnie wielokrotnością częstotliwości testowania stanu przycisku (50 Hz) i dzielnikiem częstotliwości zegara mikrokontrolera. Do sprawdzenia tego warunku można użyć stosownych dyrektyw preprocesora.

Po zainicjowaniu mikrokontrolera następuje jego uśpienie. Cała funkcjonalność urządzenia została zrealizowana w przerwaniu timera TIM4.

Procedura obsługi przerwania timera składa się z dwóch bloków, wykonywanych odpowiednio przy końcu okresu timera i przy osiągnięciu przez timer wartości zapisanej w rejestrze CCR1.

## Wyświetlenie cyfry

Obsługa przerwania końca okresu timera składa się z kilku kolejnych czynności. Rozpoczyna się ona od wyświetlenia kolejnej cyfry, co następuje przy każdym przerwaniu. Wyświetlenie polega na zapisie wcześniej przygotowanej wartości do rejestru BSRR portu GPIOC. Wartość ta jest pobierana z wektora *display[]*, którego każdy element odpowiada jednej cyfrze wyświetlacza. Zawartość wektora jest modyfikowana przy zmianie danych, które mają być wyświetlane. Ponieważ pojedynczy zapis rejestr BSRR powoduje równoczesną zmianę stanu wszystkich sygnałów sterujących wyświetlaczem, nie ma potrzeby wykonywania

**Listing 2. Plik 7seg-common.c – inicjowanie mikrokontrolera, odliczanie czasu i sterowanie jasnością**

```
#include "board.h"
#include "7seg-common.h"

uint32_t display[NDIGITS];
// encode table for digits 0..9
static const uint8_t encode_digit[] = {
    SEG_A_MSK | SEG_B_MSK | SEG_C_MSK | SEG_D_MSK | SEG_E_MSK | SEG_F_MSK,  // 0
    SEG_B_MSK | SEG_C_MSK,  // 1
    SEG_A_MSK | SEG_B_MSK | SEG_D_MSK | SEG_E_MSK | SEG_G_MSK,  // 2
    SEG_A_MSK | SEG_B_MSK | SEG_C_MSK | SEG_D_MSK | SEG_G_MSK,  // 3
    SEG_B_MSK | SEG_C_MSK | SEG_F_MSK | SEG_G_MSK,  // 4
    SEG_A_MSK | SEG_C_MSK | SEG_D_MSK | SEG_F_MSK | SEG_G_MSK,  // 5
    SEG_A_MSK | SEG_C_MSK | SEG_D_MSK | SEG_E_MSK | SEG_F_MSK | SEG_G_MSK,  // 6
    SEG_A_MSK | SEG_B_MSK | SEG_C_MSK,  // 7
    SEG_A_MSK | SEG_B_MSK | SEG_C_MSK | SEG_D_MSK | SEG_E_MSK | SEG_F_MSK | SEG_G_MSK,  // 8
    SEG_A_MSK | SEG_B_MSK | SEG_C_MSK | SEG_D_MSK | SEG_F_MSK | SEG_G_MSK  // 9
};
static const uint16_t digmask[] = {DIG0_MSK, DIG1_MSK, DIG2_MSK, DIG3_MSK};

// set clock to 80 MHz, PLL MSI-fed, sychronized to LSE
// setup GPIOC
void common_setup(void)
{
    RCC->APB1ENR1 = RCC_APB1ENR1_PWREN | RCC_APB1ENR1_TIM4EN;
    RCC->AHB2ENR = RCC_AHB2ENR_GPIOCEN;
    PWR->CR1 |= PWR_CR1_DBP;  // Enable access to BDCR
    RCC->BDCR = RCC_BDCR_LSEON;
    // GPIOC - display and button
    GPIOC->MODER = BF2A(0, GPIO_MODER_OUT) & BF2A(1, GPIO_MODER_OUT) & BF2A(2, GPIO_MODER_OUT) & BF2A(3, GPIO_MODER_OUT) & BF2A(4,
GPIO_MODER_OUT) & BF2A(5, GPIO_MODER_OUT) & BF2A(6, GPIO_MODER_OUT) & BF2A(7, GPIO_MODER_OUT) & BF2A(8, GPIO_MODER_OUT) & BF2A(9,
GPIO_MODER_OUT) & BF2A(10, GPIO_MODER_OUT) & BF2A(11, GPIO_MODER_OUT) & BF2A(13, GPIO_MODER_IN);
    GPIOC->PUPDR = BF2(13, GPIO_PUPDR_PU);  // Nucleo64 button
    while (~RCC->BDCR & RCC_BDCR_LSERDY);
    RCC->CR |= RCC_CR_MSIPLLEN;  // Sync MSI to LSE
    // PLL - 80 MHz
    RCC->PLLCFGR = RCC_PLLCFGR_PLLREN | RCC_PLLCFGR_PLLNV(40) | RCC_PLLCFGR_PLLMV(1) | RCC_PLLCFGR_PLLSRC_MSI;
    RCC->CR |= RCC_CR_PLLON;
    // set Flash speed
    FLASH->ACR = FLASH_ACR_DCEN | FLASH_ACR_ICEN | FLASH_ACR_PRFTEN | FLASH_ACR_LATENCY_4WS;  // 4ws 64..80
    while (!(RCC->CR & RCC_CR_PLLRDY));
    RCC->CFGR |= RCC_CFGR_SW_PLL;
}

// key handling, brightness control, time increment
void run_every_10ms(void)
{
    static uint8_t k_nuc_hist;
    static uint8_t brightness_target = MPX_BRIGHT;
    // key handling
    if ((k_nuc_hist = (k_nuc_hist << 1 | BTN_DOWN) & 3) == 1) brightness_target ^= MPX_BRIGHT ^ MPX_DIM;
    // brightness adjust
    uint32_t d = LEDMpx_TIM->CCR1;
    if (d != brightness_target) LEDMpx_TIM->CCR1 = d < brightness_target ? d + 1 : d - 1;
    static uint16_t tdiv = 99;
    if (++ tdiv == 100)
    {
        tdiv = 0;
        // 1s time increment
        static uint16_t seccnt = 9999;
        if (++ seccnt == 10000) seccnt = 0;
        // prepare image
        uint32_t v = seccnt;
        for (uint32_t i = 0; i < NDIGITS; i ++)
        {
            display[i] = DigAct(digmask[i]) | SegAct(v || !i ? encode_digit[v % 10] : 0);
            v /= 10;
        }
    }
}
```

opisanej wcześniej sekwencji czynności związanych z przejściem do kolejnej cyfry.

## Regulacja jasności

Zawartość rejestru CCR1 określa wypełnienie, a tym samym jasność wyświetlacza. Obraz cyfry jest wyświetlany przez przerwanie końca okresu, a wygaszany w przerwaniu porównania CCR1. Jedyną akcją wykonywaną przy obsłudze tego przerwania jest wyłączenie wszystkich cyfr i segmentów, które następuje w wyniku zapisu odpowiedniej stałej wartości do rejestru BSRR portu sterującego wyświetlaczem.

## Odliczanie czasu i sterowanie jasnością

Sto razy na sekundę procedura obsługi przerwania timera wywołuje procedurę *run_every_10ms()* służącą do sterowania jasnością wyświetlacza i odliczania czasu. Okres 10 ms zapewnia poprawne ignorowanie drgań styków przycisku.

Wykrycie naciśnięcia przycisku następuje, gdy przy poprzednim teście przycisk był zwolniony, a obecnie jest wciśnięty. W takim przypadku następuje zmiana zadanej jasności wyświetlacza poprzez przełączenie pomiędzy dwiema dostępnymi jasnościami. W celu uzyskania płynnej zmiany jasności kolejny fragment kodu stopniowo modyfikuje bieżące wypełnienie przebiegów sterujących wyświetlaniem tak, by osiągnęło ono wartość zadaną.

Raz na 100 wejść do procedury, a więc jeden raz na sekundę, następuje wejście do bloku aktualizacji czasu. Licznik sekund jest w nim inkrementowany, a następnie jego stan jest zamieniany na reprezentację odpowiednią do sterowania wyświetlacza. Informacja dla każdej cyfry przyjmuje postać słowa 32-bitowego, przeznaczonego do przesłania do rejestru BSRR portu GPIOC. Do wyznaczenia zawartości tego słowa służą makrodefinicje umieszczone w pliku *board.h*. Zostały one skonstruowane w taki sposób, by łatwo można było zdefiniować polaryzację sygnałów sterujących aktywacją cyfr i segmentów, zależną od typu wyświetlacza (wspólna anoda/katoda) i sposobu jego podłączenia (wzmacniacze odwracające lub ich brak). Służą do tego dwa symbole preprocesora: *DigActLevel* i *SegActLevel*, określające poziom logiczny wyjścia, przy którym następuje odpowiednio załączenie cyfry lub segmentu (**listing 2**).

**Listing 3. Obsługa wyświetlacza multipleksowanego przy użyciu DMA**

```
/* STM32L476 + Companion Board
7-segment mpx display, DMA referesh, brightness control
gbm, 01'2017 */

#include "board.h"
#include "7seg-common.h"
static const uint32_t displayoff = DigAct(0) | SegAct(0);
int main(void)
{
    // enable peripherals
    RCC->AHB1ENR |= RCC_AHB1ENR_DMA1EN;
    // setup clock & GPIOC
    common_setup();
    // LED multiplexing DMA
    DMA1_CSELR->CSELR = BF4(0, LEDMpx_DMARq) | BF4(3, LEDMpx_
DMARq) | BF4(6, LEDMpx_DMARq);
    LEDMpx_DMACh->CPAR = (uint32_t)&LEDMpx_PORT->BSRR;
    LEDMpx_DMACh->CMAR = (uint32_t)display;
    LEDMpx_DMACh->CNDTR = NDIGITS;
    LEDMpx_DMACh->CCR = DMA_CCR_DIR_M2P | DMA_CCR_MSIZE32 | DMA_
CCR_PSIZE32 | DMA_CCR_MINC | DMA_CCR_CIRC | DMA_CCR_EN;
    LEDMpxOff_DMACh->CPAR = (uint32_t)&LEDMpx_PORT->BSRR;
    LEDMpxOff_DMACh->CMAR = (uint32_t)&displayoff;
    LEDMpxOff_DMACh->CNDTR = 1;
    LEDMpxOff_DMACh->CCR = DMA_CCR_DIR_M2P | DMA_CCR_MSIZE32 |
DMA_CCR_PSIZE32 | DMA_CCR_MINC | DMA_CCR_CIRC | DMA_CCR_EN;
    // LED multiplexing timer
    LEDMpx_TIM->PSC = SYSCLK_FREQ / MPX_FREQ / MPX_STEPS - 1;
    LEDMpx_TIM->ARR = MPX_STEPS - 1;
    LEDMpx_TIM->DIER = TIM_DIER_CC1DE | TIM_DIER_UDE;
    LEDMpx_TIM->EGR = TIM_EGR_UG;
    LEDMpx_TIM->CR1 = TIM_CR1_CEN;
    SysTick_Config(SYSCLK_FREQ / SYSTICK_FREQ);
    SCB->SCR = SCB_SCR_SLEEPONEXIT_Msk;  // sleep while not in
handler
    __WFI();  // go to sleep
}

void SysTick_Handler(void)
{
    run_every_10ms();
}
```

## Sprzętowa obsługa wyświetlacza przy użyciu DMA

Dostępny we wszystkich modelach mikrokontrolerów rodziny STM32 moduł bezpośredniego dostępu do pamięci umożliwia obsługę odświeżania wyświetlacza bez udziału oprogramowania, pod warunkiem, że wszystkie sygnały sterujące pochodzą z jednego portu GPIO.

Druga wersja programu demonstracyjnego, zawarta w projekcie *7seg-dma* i korzystająca z modułu bezpośredniego dostępu do pamięci, realizuje sprzętowo funkcje, które w poprzednim przykładzie były wykonywane w procedurze obsługi przerwania timera. Eliminujemy w ten sposób konieczność zgłaszania przerwań przez timer sterujący wyświetlaniem i zmniejszamy obciążenie procesora obsługą przerwań. Dzięki temu możemy zmniejszyć częstotliwość przerwań timera do niezbędnej dla uzyskania pozostałej funkcjonalności urządzenia. W naszym przypadku będzie to częstotliwość 100 Hz, potrzebna do sprawdzania stanu przycisku i odmierzania czasu. Do zgłaszania przerwań z tą częstotliwością użyjemy timera SysTick. Akcje związane z obsługą przerwania realizuje opisana wcześniej procedura *run_every_10ms()*. Aby uniknąć dodatkowego narzutu czasu potrzebnego na jej wywołanie z procedury obsługi przerwania SysTick można zmienić jej nazwę na *SysTick_Handler* przy użyciu odpowiedniej definicji dla preprocesora.

W celu zapewnienia odświeżania wyświetlacza należy najpierw zaprogramować moduł DMA. Potrzebne do tego symbole – kanały i numery żądań DMA – zostały zdefiniowane w pliku *board.h*. W projekcie użyto dwóch kanałów DMA, z których jeden służy do wyświetlania kolejnych cyfr, a drugi – do regulacji jasności poprzez ich wygaszania po upłynięciu określonego czasu od wyświetlenia. Podobnie jak przy realizacji programowej, obie te czynności będą wyzwalane przez odpowiednie zdarzenia generowane przez timer. Inicjowanie modułu DMA umieszczono w funkcji main() (**listing 3**).

W celu zaprogramowania DMA wykonujemy kolejno następujące czynności:

- Wybieramy źródła zgłoszeń dla obu używanych kanałów DMA (rejestr DMA1_CSELR).
- Ustawiamy adres docelowy danych dla kanału wyświetlania – rejestr BSRR portu wyświetlacza.
- Ustawiamy adres źródłowy danych – wektor sterowań dla poszczególnych cyfr.
- Ustawiamy długość bufora równą 4.
- Ustawiamy tryb pracy kanału DMA (transmisja powtarzana danych 32-bitowych z inkrementacją adresu źródła) i włączamy kanał.
- Ustawiamy adres docelowy danych dla kanału wygaszania – rejestr BSRR portu wyświetlacza.
- Ustawiamy adres źródłowy danej – stałej powodującej wygaszenie całego wyświetlacza.
- Ustawiamy długość bufora równą 1.
- Ustawiamy tryb pracy kanału DMA (transmisja powtarzana danych 32-bitowych) i włączamy kanał.

Po zaprogramowaniu moduł DMA czeka na żądania transmisji, które będą generowane przez timer po jego zaprogramowaniu. Timer odświeżania wyświetlacza programujemy podobnie jak dla odświeżania programowego, jednak nie włączamy przerwań końca okresu i porównania CCR1. Zamiast tego włączamy w rejestrze TIM4 → DIER żądania transmisji DMA wynikające z obu tych zdarzeń. Transmisja danych jest wykonywana przez moduł DMA na podstawie żądań zgłaszanych przez timer.

**Uwaga:** w mikrokontrolerach serii STM32F4 i STM32F7, z powodu struktury połączeń wewnętrznych, dostęp do portów GPIO ma jedynie moduł DMA2. W przypadku użycia mikrokontrolerów z tych serii żądanie odświeżania musi być zgłaszane przez timer połączony z modułem DMA2.

**Grzegorz Mazur**