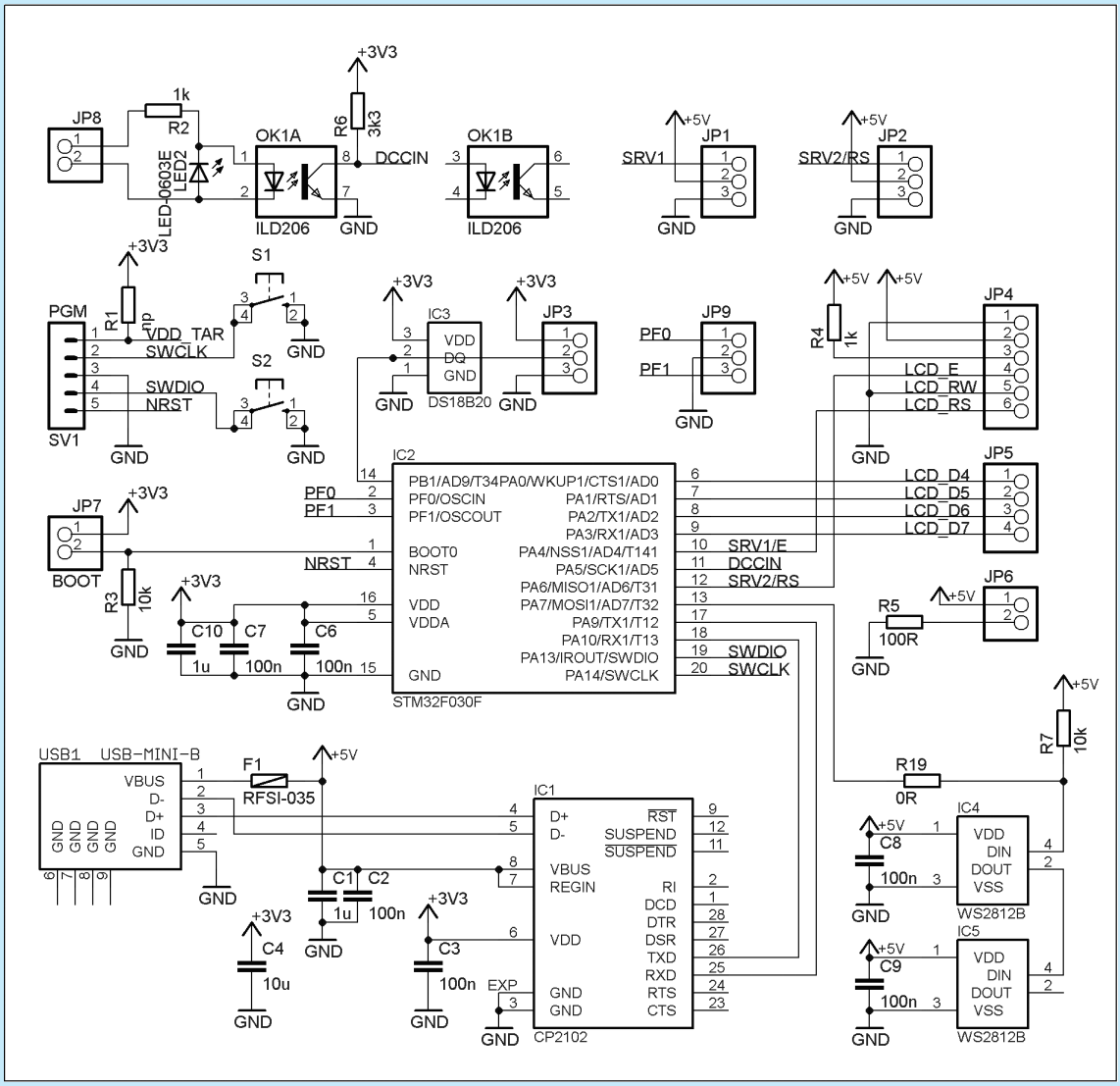
32 bity jak najprościej (3)

STM32F0 – płytka eksperymentalna z mikrokontrolerem STM32F030F4

Latem 2013 roku pojawiła się na rynku nowa seria mikrokontrolerów – STM32F030. Są one bardzo podobne do STM32F050, lecz mają nieco uboższy zestaw bloków funkcjonalnych i niższą cenę. W ramach dostępnych peryferiali STM32F030 zachowują całkowitą zgodność z STM32F05x, więc zaprezentowane wcześniej przykłady mogą być bez żadnych zmian uruchomione na płytce STM32F0308DISCOVERY, wyposażonej w mikrokontroler STM32F030R8.

Ponieważ prezentowane dotychczas w tej serii artykułów przykłady niemal wyczerpały możliwości samej płytki STM32F0DISCOVERY, do dalszych ćwiczeń z programowania mikrokontrolerów powstała płytka f030exp1, wyposażona w najmniejszy układ serii STM32F030 – model STM32F030F4 w20-końcówkowej obudowie TSSOP. Tematem kilku następnych projektów będzie nieblokująca i nietypowa obsługa popularnych składników urządzeń z mikrokontrolerami, dlatego na płytce przewidziano możliwość dołączenia do mikrokontrolera m.in:

- wyświetlacza LCD ze sterownikiem serii HD44780,
- czujników temperatury serii DS18x20,
- do 2 szt. LED ze zintegrowanym sterownikiem typu WS2812 lub taśmy zawierającej dowolną liczbę tych elementów,
- interfejsu USB-UART typu CP2102



Rysunek 1. Schemat ideowy płytki eksperymentalnej

- dwóch serwomechanizmów
- izolowanego wejścia cyfrowego, np. dla sygnału DCC.

Schemat płytki przedstawiono poniżej. Płytka jest wyposażona w złącze do programowania i debugowania zgodne ze złączem udostępnianym na płytkach serii DISCOVERY.

Oczywiście wszystkie projekty zrealizowane przy użyciu płytki mogą być również wykonane na bazie płytki DISCOVERY poprzez dołączenie odpowiednich elementów zewnętrznych do złącz za pomocą przewodów. Schemat ideowy płytki pokazano na **rysunku 1**.

Na potrzeby programów działających na płytce został stworzony plik nagłówkowy, definiujący przypisania linii portów mikrokontrolera. Plik ten jest umieszczony wraz z innymi plikami pomocniczymi w folderze Common, który musi być włączony do ścieżki poszukiwania plików nagłówkowych.

Sterowanie układów WS2812B przy użyciu interfejsu SPI

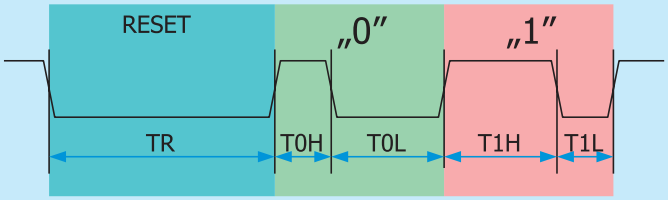
W kolejnym przykładzie zajmiemy się dwoma zagadnieniami – programowanie modułu PLL służącego do powielania częstotliwości zegara mikrokontrolera oraz nietypowym użyciem interfejsu SPI. Tematem przykładu będzie sterowanie łańcuchem diod LED-RGB z wbudowanymi sterownikami PWM. Do uruchomienia przykładów użyto płytki F0exp1a z jednym układem WS2812B, połączonej z taśmą zawierającą 144 układy tego samego typu. Układ hybrydowy WS2812B chińskiego producenta WorldSemi zawiera w jednej czterokońcówkowej obudowie typu 5050 scalony sterownik LED serii WS2811 i trzy diody – czerwoną, zieloną i niebieską. Układ ma cztery wyprowadzenia – zasilanie 5 V oraz jedną linię wejściową i jedną linię wyjściową danych. Przy pełnej jasności diod pojedynczy układ pobiera prąd o natężeniu do 60 mA. Zasilenie taśmy zawierającej 144 układy przy dopuszczeniu wysterowania wszystkich diod na pełną jasność wymaga więc zastosowania zasilacza o odpowiednio dużej wydajności prądowej – ok. 10 A.

Układ WS2812B – format danych i parametry czasowe

Układ WS2812 ma cztery wyprowadzenia – masa, zasilanie 5 V oraz wejście i wyjście danych. W celu wysterowania diod do układu przesyła się słowo 24-bitowe określające wypełnienia przebiegów sterujących poszczególnymi diodami (**rysunek 2**). Układy mogą być łączone w łańcuchy – każdy układ odbiera, zapamiętuje i usuwa z wejściowego strumienia danych pierwsze 24 bity (emitując w ich miejsce na wyjściu poziom niski), a pozostałe dane po regeneracji przepuszcza na wyjście, połączone z wejściem kolejnego układu w łańcuchu. Zastosowany format danych umożliwia transmisję danych po jednej linii sygnałowej, bez oddzielnego przebiegu zegarowego. Układ rozróżnia trzy symbole wejściowe:

- reset (start danych) – poziom niski linii wejściowej trwający min. 50 µs,
- transmisja bitu 0 – poziom wysoki trwający 400 ns, stan niski przez 850 ns,
- transmisja bitu1 – poziom wysoki przez 800 ns, stan niski przez 450 ns.

Producent określa tolerancję czasów trwania poszczególnych stanów dla transmisji zera i jedynki logicznej na 150 ns. Nominalna częstotliwość transmisji bitów wynosi 800 kHz.



Rysunek 2. Przebiegi czasowe układu WS2812B

Podawane na wejście układu słowo 24-bitowe zawiera 8-bitowe wartości trzech składowych w kolejności: zielony, czerwony, niebieski., przy czym poszczególne składowe są transmitowane w kolejności od bitu najbardziej znaczącego do najmniej znaczącego. Pomiędzy danymi transmitowanymi do kolejnych elementów w łańcuchu nie mogą występować przerwy. Gdy dane nie są transmitowane, linia pozostaje w stanie niskim (RESET).

Należy zauważyć, że współczesne mikrokontrolery nie są wyposażone w moduły interfejsów umożliwiające wprost transmisję danych w formacie wymaganym przez układ WS2812. Przebieg taki można byłoby wygenerować przy użyciu wyjścia timera działającego w trybie PWM, wymagałoby to jednak modyfikacji wypełnienia z częstotliwością transmisji bitów – 800 kHz, co leży poza możliwościami obsługi przerwań większości mikrokontrolerów.

Do generowania potrzebnego przebiegu można jednak użyć dostępnego w większości mikrokontrolerów modułu SPI, dokonując programowego kodowania strumienia bitów. Łatwo zauważyć, że wymagany przez układ stosunek czasów trwania poziomów wysokiego i niskiego jest bliski 1:2 przy transmisji bitu o wartości 0 i 2:1 przy transmisji bitu o wartości 1. Oznacza to, że korzystając z interfejsu SPI, zapewniającego z zasady swego działania stały okres transmisji bitu, można przesyłać do WS2812 bit o wartości 0 jako sekwencję jedynki i dwóch zer, a bit o wartości 1 – jako sekwencję dwóch jedynek i zera. Ponieważ jeden bit będzie przesyłany przez SPI jako trzy bity, częstotliwość transmisji SPI powinna wynosić 2,4 MHz. Biorąc pod uwagę dopuszczalne tolerancje długości impulsów układu WS2812B, może się ona mieścić w zakresie od ok. 2,2 do 2,6 MHz. Sekwencja startowa może być wygenerowana poprzez transmisję odpowiedniej liczby bitów o wartości 0 (min. 120 bitów przy częstotliwości 2,4 MHz, min. 125 bitów przy 2,5 MHz).

Istotnym problemem w realizacji transmisji w opisany sposób jest zapewnienie jej ciągłości – w transmitowanym strumieniu danych nie mogą wystąpić jakiekolwiek przerwy, wynikające np. z opóźnienia w zapisie kolejnego słowa danych do interfejsu SPI.

Użycie modułu SPI mikrokontrolera STM32F0 do formatowania danych dla WS2812B

Ponieważ wartości składowych RGB poszczególnych diod są reprezentowane wyjściowo przez liczby 8-bitowe, kodowanie danych powinno być przeprowadzone w sposób minimalizujący nakłady czasowe oprogramowania. Moduł SPI w STM32F0 może transmitować słowa o długości od 4 do 16 bitów. W naszym zastosowaniu

Listing 1. Plik f030exp1.h

```
/*
    F030exp1 board defs
*/

#include „stm32f0yy.h”
#include „stm32futil.h”
//================================================
#define BUTTON_PORT                 GPIOA
#define BUTTON_BIT          0

#define SRV1_PWM   TIM14->CCR1
#define SRV2_PWM   TIM3->CCR1
//================================================
#define LCD_PORT   GPIOA
#define LCD_E_BIT    4
#define LCD_RS_BIT   6
#define LCD_E_SET          LCD_PORT->BSRR = 1 << LCD_E_BIT
#define LCD_E_CLR          LCD_PORT->BRR = 1 << LCD_E_BIT
#define LCD_RS_SET         LCD_PORT->BSRR = 1 << LCD_RS_BIT
#define LCD_RS_CLR         LCD_PORT->BRR = 1 << LCD_RS_BIT
#define LCD_DATA_SET(v)    LCD_PORT->BSRR = ((v) & 0xf) |
(0xf << 16)
```

wygodne będzie użycie formatu 12-bitowego, w którym jedna ramka danych SPI odpowiada czterem bitom danych transmitowanych do WS2812B. Każdy przesyłany bajt zostanie zakodowany w postaci dwóch słów 12-bitowych.

Częstotliwość transmisji interfejsu SPI w STM32F0 jest uzyskiwana przez podział głównego zegara modułów peryferyjnych (PCLK) przez potęgę dwójki z zakresu od 1 do 256. Korzystając z oscylatora o częstotliwości 8 MHz lub generatora PLL o częstotliwości wejściowej 4 MHz nie da się więc uzyskać częstotliwości 2,4 MHz. Uwzględniając dopuszczalną tolerancję parametrów czasowych układu WS2812B można użyć częstotliwości 2,5 MHz, która może być uzyskana przez podział częstotliwości 40 MHz przez 16 lub 20 MHz przez 8. Jeżeli decydujemy się na użycie wewnętrznego oscylatora RC mikrokontrolera oznacza to konieczność użycia modułu PLL w celu podwyższenia częstotliwości taktowania z domyślnych 8 MHz do wielokrotności 20 MHz.

Należy zauważyć, że przy takiej częstotliwości transmisji zapis kolejnych słów danych do rejestru nadawczego SPI musi następować z częstotliwością ponad 200 kHz, a, jak już wcześniej wspomniano, niedopuszczalne są tu jakiekolwiek opóźnienia, które mogłyby spowodować przerwę w transmisji strumienia bitów. Przy tak dużej częstotliwości transmisji słów warto byłoby do obsługi SPI użyć modułu DMA. Powodowałoby to jednak znaczny wzrost zajętości pamięci RAM, gdyż transmitowany dany musiałyby być w niej przechowywane w postaci gotowej do transmisji – jeden oktet zajmowałby więc w pamięci dwa słowa 16-bitowe o 12 bitach znaczących. Z tego powodu w prezentowanym rozwiązaniu zastosowano obsługę transmisji z użyciem przerwań. Ponieważ sam narzut czasu wynikający z rozpoczęcia i zakończenia obsługi przerwania wynosi ok. 30 cykli zegara procesora, poprawna obsługa przerwań SPI wymaga świadomego i starannego zaprojektowania całego oprogramowania. Należy zwrócić uwagę na wymaganą wydajność procesora – częstotliwość taktowani musi być znacznie wyższa od domyślnych 8 MHz, a inne używane w projekcie przerwania nie mogą zakłócać obsługi przerwania SPI.

Programy przykładowe

Poniżej przedstawiono dwa przykłady programów sterujących zespołem układów WS2812B. Pierwszy umożliwia zadawanie kolorów świecenia poszczególnych elementów z komputera PC przez interfejs VCOM – np. przy użyciu programu terminala. Drugi program steruje prostą animacją. Oba programy zostały uruchomione i przetestowane przy użyciu taśmy zawierającej 144 układy WS2812B.

Najpierw zostaną omówione fragmenty oprogramowania wspólne dla obu programów, a następnie fragmenty specyficzne dla każdego z nich.

Programowanie generatora zegara – PLL

We wcześniejszych projektach używaliśmy domyślnej początkowej częstotliwości taktowania mikrokontrolera, wynoszącej 8 MHz. Ponieważ do sterowania WS2812B potrzebujemy taktować moduł SPI częstotliwością stanowiącą wielokrotność 20 MHz, musimy odpowiednio podnieść częstotliwość taktowania mikrokontrolera.

Źródłem częstotliwości wejściowej PLL będzie oscylator HSI mikrokontrolera, o nominalnej częstotliwości 8 MHz. Zgodnie z dokumentacją modułu RCC częstotliwość wejściowa dla PLL wynosi w tym przypadku 4 MHz i może być ona mnożona przez liczbę całkowitą z zakresu od 1 do 12. Z wymagań czasowych układu WS2812B wynika, że częstotliwość PCLK (domyślnie równa częstotliwości głównego zegara mikrokontrolera) musi wynosić 20 lub 40 MHz, tak, aby po jej podzieleniu przez potęgę dwójki można było uzyskać częstotliwość 2,5 MHz. W prezentowanym projekcie wybrano częstotliwość 40 MHz.

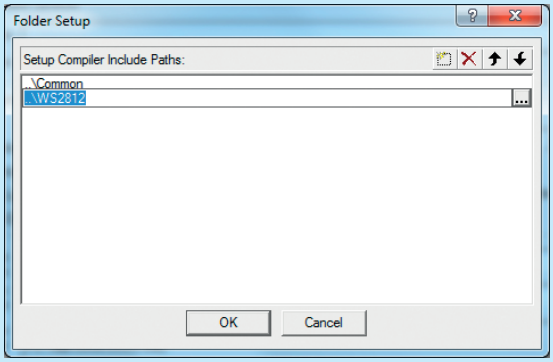
Zaprogramowanie generatora składa się z dwóch faz. Pierwsza – to konfiguracja i włączenie PLL, druga – to przełączenie źródła przebiegu zegarowego mikrokontrolera, które może nastąpić po ustabilizowaniu działania PLL, co według danych producenta może zająć do 200 μs. Aby nie marnować całego tego czasu na oczekiwanie, w przykładzie rozsunięto obie fazy w czasie – uruchomienie PLL następuje w procedurze SystemInit(), wywoływanej bezpośrednio po starcie procesora. Funkcja main() zostaje wywołana po zainicjowaniu zmiennych, co zajmuje czas zbliżony do czasu stabilizacji PLL. Na początku funkcji main() jest umieszczona pętla, w której następuje oczekiwanie na gotowość PLL, po czym zostaje wywołana procedura inicjująca moduły mikrokontrolera. Ponieważ zastosowana w projekcie docelowa częstotliwość zegara wymaga wprowadzenia taktu oczekiwania przy dostępie do pamięci Flash, na samym początku sekwencji inicjującej następuje ustawienie wydłużenia czasu dostępu pamięci (rejestr FLASH>ACR), a zaraz potem przełączenie taktowania mikrokontrolera.

Sekwencja inicjująca generator przebiegu zegarowego jest umieszczona w plik main.c każdego z projektów, podobnie jak sekwencja inicjująca moduł SPI.

Inicjowanie interfejsu SPI

Spośród czterech linii tworzących interfejs SPI, do sterowania łańcucha WS2812B jest potrzebna tylko jedna – linia MOSI, dlatego też tylko ona zostaje uaktywniona w ustawieniach funkcji linii portów. Ponieważ częstotliwość transmisji przekracza 2 MHz, niezbędne jest zaprogramowanie podwyższonej szybkości bufora wyjściowego tej linii w rejestrze OSPEEDR.

Samo zainicjowanie modułu SPI sprowadza się do dwóch zapisów rejestrów sterujących modułu SPI. Najpierw w rejestrze CR2 ustala się długość słowa danych i włącza zgłaszanie przerwania przy gotowości nadajnika. Następnie zapis do rejestru CR1 powoduje ustalenie



Rysunek 3. Dialog ścieżek poszukiwania plików nagłówkowych

szybkości transmisji i włączenie modułu SPI w trybie master.

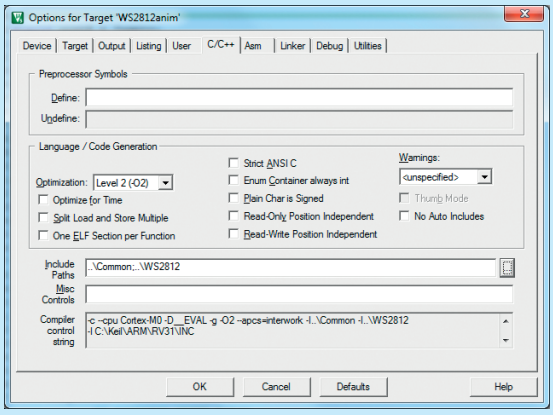
W dwóch przedstawionych programach pokazano dwa nieco odmienne sposoby inicjowania modułu SPI, wykorzystujące różne wartości rejestrów CR2 i CR1 i różniące się definicją funkcji wewnętrznego sygnału SS interfejsu. Działanie modułu przy obu możliwych sposobach zaprogramowania funkcji SS jest identyczne.

Przerwanie od modułu SPI musi również zostać włączone w sterowniku przerwań, co dzieje się pod koniec sekwencji inicjującej mikrokontroler.

Obsługa interfejsu SPI

Procedurę obsługi przerwania SPI i procedurę inicjowania transmisji wydzielono w module WS2812.c, a potrzebne deklaracje zawarto w pliku nagłówkowym WS2812.h. Oba pliki zostały umieszczone w folderze WS2812. Aby włączyć je do projektu należy, poza dodaniem do projektu pliku WS2812.c, dodać w opcjach projektu folder WS2812 do ścieżki poszukiwania plików nagłówkowych. W tym celu otwieramy dialog opcji projektu (**rysunek 3** i **rysunek 4**) i w zakładce C/C++ dodajemy folder WS2812 do ścieżki.

Do ładowania danych transmitowanych przez interfejs SPI użyto przerwania nadawania danych SPI. Jest ono zgłaszane z częstotliwością ponad 200 kHz, a jego obsługa nie może zostać opóźniona o czas dłuższy od ok. 5 µs, co odpowiada nieco mniej niż dwustu instrukcjom procesora. To wymaganie powoduje konieczność starannego zaprojektowania pozostałych części oprogramowania, w tym przyjrzenia się procedurom obsługi innych przerwań wykorzystywanych w projekcie.



Rysunek 4. Dialog opcji projektu z aktywną zakładką opcji kompilatora

Listing 2. Plik WS2812.h

```
/*  WS2812b - related declarations
    gbm, 12'13  */

#include <stdint.h>
// data order: green, red, blue

struct wspix_ {
    uint8_t green, red, blue;
};

void WS2812_start(struct wspix_ *ptr, uint32_t nleds);
```

Ponieważ gotowość nadajnika jest jedynym uaktywnionym źródłem przerwania SPI, nie ma potrzeby sprawdzania w procedurze obsługi przerwania innych znaczników stanu interfejsu poza bitem TXE w rejestrze stanu SR. Obsługę przerwania zrealizowano w konwencji automatu trójstanowego. Ma ona postać pętli, gdyż w chwili rozpoczęcia obsługi przerwania w buforze danych interfejsu SPI może być miejsce na więcej niż jedno słowo.

Plik nagłówkowy i przykładowy program obsługi zamieszczono na **listingu 2** i **listingu 3**.

Stan WS_RESET odpowiada generowaniu stanu startu transmisji dla układów WS2812. W stanie tym są transmitowane słowa zawierające same zera, a więc linia MOSI pozostaje w stanie niskim. Warunkiem przejścia z tego stanu do stanu WS_DATAH jest gotowość nowych danych do transmisji i osiągnięcie odpowiedniego czasu trwania poziomu niskiego.

W stanie WS_DATAH do bufora danych jest ładowane słowo 12-bitowe reprezentujące wartość bardziej

Listing 3. Plik WS2812.c

```
/*  WS2812b service routines
    gbm, 12'13  */

#include „stm32f0yy.h”
#include „WS2812.h”

static uint8_t *wsptr;
static uint32_t wscnt;

void WS2812_start(struct wspix_ *ptr, uint32_t nleds)
{
    wsptr = (uint8_t *)ptr;
    wscnt = nleds * 3;
}

// WS2812 reset pulse width
// (50 us -> 120 bits @2.4 MHz, 125 bits @2.5 MHz)
#define WS_RST_FRM11 // no. of 12-bit frames

void SPI1_IRQHandler(void)
{
    static enum ws_state_ {WS_RESET, WS_DATAH, WS_DATAL}
ws_state = WS_RESET;
    static uint8_t rstcnt = WS_RST_FRM;    // reset frame
count
    static const uint16_t encode[] = {
        04444, 04446, 04464, 04466, 04644, 04646, 04664,
04666,
        06444, 06446, 06464, 06466, 06644, 06646, 06664,
06666
    };  // bit-to triple encoding table

    while (SPI1->SR & SPI_SR_TXE)
    switch (ws_state)
    {
        case WS_RESET:  // reset pulse
            if (rstcnt || wscnt == 0)
            {
            // continue reset pulse
                if (rstcnt) rstcnt --;
                SPI1->DR = 0;
                break;
            }
            // ready to start data xfer
            rstcnt = WS_RST_FRM;
        case WS_DATAH:  // high nibble
            SPI1->DR = encode[*wsptr >> 4];
            ws_state = WS_DATAL;
            break;
        case WS_DATAL:    // low nibble
            SPI1->DR = encode[*wsptr ++ & 0xf];
            ws_state = -- wscnt ? WS_DATAH : WS_RESET;
    }
}
```

Listing 4. Plik main.c dla przykładu 1

```
/*  STM32F0 tutorial
    WS2812B control w/SPI, PLL in use, UART
    gbm, 12'2013    */

#include „f030exp1.h"
#include „WS2812.h"
#define SYSCLK_FREQ        (HSI_VALUE * 10/2)
#define BAUD_RATE 115200
void SystemInit(void)
{
    FLASH->ACR = FLASH_ACR_PRFTBE;  // enable prefetch
    RCC->CFGR = RCC_CFGR_PLLMULL10;  // set PLL multiplier to 4 for 16 MHz clock
    RCC->CR |= RCC_CR_PLLON;  // turn PLL on
}

static const struct init_entry_ init_table[] =
{
    {&FLASH->ACR, FLASH_ACR_PRFTBE | 1},  // enable prefetch, 1 wait state
    {&RCC->CFGR, RCC_CFGR_PLLMULL10 | RCC_CFGR_SW_PLL},  // switch to PLL clock
    // enable peripherals
    {&RCC->APB2ENR, RCC_APB2ENR_USART1EN | RCC_APB2ENR_SPI1EN},
    {&RCC->AHBENR, RCC_AHBENR_GPIOAEN | RCC_AHBENR_SRAMEN},
    // port setup
    // GPIOA AFR[0]: 7..4 - SPI is fun 0 (default)
    {&GPIOA->AFR[1], BF4(10, 1) | BF4(9, 1)},  // USART pins 10 - RX, 9 - TX
    {&GPIOA->OSPEEDR, BF2(7, GPIO_OSPEEDR_MED)},  // MOSI - medium speed
    {&GPIOA->MODER, GPIOA_MODER_SWD | BF2(10, GPIO_MODER_AF)
        | BF2(9, GPIO_MODER_AF) | BF2(7, GPIO_MODER_AF)},  // UART pins & MOSI as AF
    // USART1 setup
    {(__IO32p)&USART1->BRR, (SYSCLK_FREQ + BAUD_RATE / 2) / BAUD_RATE},
    {&USART1->CR1, USART_CR1_RXNEIE | USART_CR1_TE | USART_CR1_RE | USART_CR1_UE},  // enable
    // SPI setup
    {(__IO32p)&SPI1->CR2, SPI_CR2_DSIZE(12) | SPI_CR2_TXEIE | SPI_CR2_SSOE},
    {(__IO32p)&SPI1->CR1, SPI_CR1_SPE | SPI_CR1_BRDIV16 | SPI_CR1_MSTR},  // enable
    // interrupts and sleep
    {&NVIC->ISER[0], 1 << USART1_IRQn | 1 << SPI1_IRQn},  // enable interrupts
    {&SCB->SCR, SCB_SCR_SLEEPONEXIT_Msk},  // sleep while not in handler
    {0, 0}
};

int main(void)
{
    while (!(RCC->CR & RCC_CR_PLLRDY));  // wait for PLL lock
    writeregs(init_table);
    __WFI();  // go to sleep
}

// WS281x data
#define NLEDS 145
static struct wspix_ wsdata[NLEDS];

void USART1_IRQHandler(void)
{
    static uint8_t val = 0, idx = 0;
    uint32_t c;
    if (USART1->ISR & USART_ISR_RXNE)  // data received
    {
        c = USART1->RDR;
        if (c >= ,a') c -= ,a' - ,A';
        if ((c >= ,0' && c <= ,9') || (c >= ,A' && c <= ,F'))
        {
            USART1->TDR = c;
            if (c >= ,A') c -= ,A' - ,9' - 1;
            val = (val << 4) + c - ,0';
            if ((idx & 1) && (idx >> 1) < sizeof(wsdata))
                ((uint8_t *)wsdata)[idx >> 1] = val;  // store byte
            idx ++;
        }
        else if (c == ,\r')
        {
            USART1->TDR = c;
            USART1->CR1 |= USART_CR1_TXEIE;  // will send ,\n'
            WS2812_start(wsdata, idx / 6);
            idx = 0;
        }
    }
    if (USART1->ISR & USART1->CR1 & USART_ISR_TXE)
    {
        USART1->CR1 &= ~USART_CR1_TXEIE;
        USART1->TDR = ,\n';
    }
}
```

znaczącej tetrady oktetu danych przesyłanych do układów WS2812. Jednocześnie następuje przejście do stanu WS_DATAL.

W stanie WS_DATAL do bufora danych SPI jest ładowane słowo 12-bitowe reprezentujące wartość mniej znaczącej tetrady bajtu danych. Następnie jest inkrementowany wskaźnik transmitowanych danych i dekrementowany ich licznik. Przy wyzerowaniu licznika (czyli zakończeniu transmisji) następuje przejście do stanu WS_RESET; w przeciwnym razie przechodzimy do stanu WS_DATAH.

Do inicjowania transmisji bloku danych po ich przygotowaniu służy procedura WS2812_start(), przy wywołaniu, której należy przekazać dwa argumenty: adres początkowy bufora danych i liczbę układów w sterowanym łańcuchu.

Przykład 1 – ustawianie kolorów świecenia przez interfejs szeregowy

Pierwszy z dwóch przykładowych programów (**listing 4**) umożliwia sterowanie zespołem WS2812 przez interfejs szeregowy mikrokontrolera. Pliki projektu są umieszczo-

Listing 5. Plik main.c dla przykładu 2

```
/*  STM32F0 tutorial
    WS2812B control w/SPI, PLL in use, SysTick-driven animation
    gbm, 12'2013    */

#include „f030exp1.h"
#include „WS2812.h"
#define SYSCLK_FREQ        (HSI_VALUE * 10/2)
#define SYSTICK_FREQ       100      // 100 Hz -> 10 ms
#define BAUD_RATE 115200

void SystemInit(void)
{
    FLASH->ACR = FLASH_ACR_PRFTBE;  // enable prefetch
    RCC->CFGR = RCC_CFGR_PLLMULL10;  // set PLL multiplier to 10 for 40 MHz clock
    RCC->CR |= RCC_CR_PLLON;       // turn PLL on
}

static const struct init_entry_ init_table[] =
{
    {&FLASH->ACR, FLASH_ACR_PRFTBE | 1},  // enable prefetch, 1 wait state
    {&RCC->CFGR, RCC_CFGR_PLLMULL10 | RCC_CFGR_SW_PLL},  // switch to PLL clock
    // enable peripherals
    {&RCC->APB2ENR, RCC_APB2ENR_SPI1EN},
    {&RCC->AHBENR, RCC_AHBENR_GPIOAEN | RCC_AHBENR_SRAMEN},
    // port setup
    // GPIOA AFR[0]: 7..4 - SPI is fun 0 (default)
    {&GPIOA->OSPEEDR, BF2(7, GPIO_OSPEEDR_MED)},  // MOSI - medium speed
    {&GPIOA->MODER, GPIOA_MODER_SWD | BF2(7, GPIO_MODER_AF)},  // MOSI as AF
    // SPI setup
    {(__IO32p)&SPI1->CR2, SPI_CR2_DSIZE(12) | SPI_CR2_TXEIE},
    {(__IO32p)&SPI1->CR1, SPI_CR1_SSM | SPI_CR1_SSI | SPI_CR1_SPE | SPI_CR1_BRDIV16 | SPI_CR1_MSTR},   //
enable
    //SysTick setup
    {&SCB->SHP[1], 0x80c00000},  // PendSV lower priority, Systick higher
    {&SysTick->LOAD, SYSCLK_FREQ / SYSTICK_FREQ - 1},
    {&SysTick->VAL, 0},
    {&SysTick->CTRL, SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_TICKINT_Msk | SysTick_CTRL_ENABLE_Msk},
    // interrupts and sleep
    {&NVIC->ISER[0], 1 << SPI1_IRQn},  // enable interrupts
    {&SCB->SCR, SCB_SCR_SLEEPONEXIT_Msk},  // sleep while not in handler
    {0, 0}
};

int main(void)
{
    while (!(RCC->CR & RCC_CR_PLLRDY));  // wait for PLL lock
    writeregs(init_table);
    __WFI();  // go to sleep
}

// WS281x data
#define NLEDS 145
static struct wspix_ wsdata[NLEDS];

// animation data
#define CHANGE_PERIOD  7
#define GEN_PERIOD  24
#define MAX  16
static const struct wspix_ colors[] = {
    {0, MAX, 0}, {MAX, MAX, 0}, {MAX, 0, 0},
    {MAX, 0, MAX}, {0, 0, MAX}, {0, MAX, MAX},
    {MAX, MAX, MAX}
};

void SysTick_Handler(void)
{
    static uint8_t change_timer = CHANGE_PERIOD;
    static uint8_t newpix_timer = GEN_PERIOD;
    static uint8_t nxtcol = 0;
    if (-- change_timer == 0)
    {
        uint32_t i;
        change_timer = CHANGE_PERIOD;
        for (i = NLEDS - 1; i > 0; i --)
            wsdata[i] = wsdata[i - 1];  // move forward
        if (-- newpix_timer == 0)
        {
            newpix_timer = GEN_PERIOD;
            wsdata[0] = colors[nxtcol]; // create new worm
            if (++ nxtcol == sizeof(colors) / sizeof(struct wspix_))
            nxtcol = 0;
        }
        else
        {
            wsdata[0].green /= 2;  // fade
            wsdata[0].red /= 2;
            wsdata[0].blue /= 2;
        }
        WS2812_start(wsdata, NLEDS);
    }
}
```

ne w folderze WS2812uart. Projekt zawiera dwa pliki źródłowe – własny plik główny main.c i plik WS2812.c z foldera WS2812, zawierający kod obsługi transmisji danych do układów WS2812. Wartości składowych mogą być zadawane np. z terminala w postaci szesnastkowej. Dane mają postać ciągu cyfr szesnastkowych. Każda para cyfr reprezentuje wartość jednej składowej światła jednego układu WS2812, w kolejności odpowiadającej porządkowi strumienia danych – zielony, czerwony, niebieski. Maksymalna liczba sterowanych diod została zdefinio-

wana jako symbol preprocesora NLEDS. Interfejs UART pracuje z szybkością 115200 b/s. Odbiór danych zrealizowano w przerwaniu UART. Oprogramowanie reaguje tylko na cyfry szesnastkowe i kod końca wiersza (CR) – są one odsyłane zwrotnie, a pozostałe znaki są ignorowane. Wartości kolejnych składowych są wpisywane do bufora, a po wprowadzeniu CR następuje przesłanie zawartości bufora do łańcucha WS2812. Do łańcucha WS2812 są transmitowane tylko wartości wprowadzone w ostatnim wierszu – diody, których wartości nie podano, zachowują poprzedni kolor świecenia. Dla podniesienia czytelności tekstu na terminalu, po kodzie CR mikrokontroler odsyła sekwencję CR+LF. Jest to osiągane poprzez odblokowanie przerwania nadajnika UART po wpisaniu do niego kodu CR. Obsługa tego przerwania spowoduje wysłanie kodu LF i zablokowanie przerwania.

Ponieważ obsługa przerwania UART jest bardzo prosta i zajmuje krótki czas, nie zachodzi niebezpieczeństwo zakłócenia ciągłości transmisji danych przez interfejs SPI.

Przykład 2 – animacja

Drugi przykład pokazuje prostą, samoczynną animację efektu świetlnego za pomocą łańcucha układów WS2812 (**listing 5**). Pliki projektu są umieszczone w folderze WS2812anim.

Animacja polega na generowaniu poruszających się ruchem jednostajnym wzdłuż taśmy „robaczków" o różnych kolorach. Sposób realizacji nie jest optymalny dla tak prostych obiektów, ale za to umożliwia on łatwą modyfikację animowanych obiektów i algorytmu animacji. Do wyznaczania okresu kroku animacji użyto timera SysTick, zaprogramowanego na 100 Hz. Parametry animacji są określone przez stałe zdefiniowane jako symbole preprocesora oraz wektor kolorów sekwencji:

- stała CHANGE_PERIOD określa szybkość poruszania się świateł wzdłuż taśmy,
- stała GEN_PERIOD określa okres, co jaki jest generowana nowa sekwencja świateł,
- stała MAX określa maksymalną intensywność składowych;
- wektor colors[] zawiera wartości składowych świateł kolejnych generowanych sekwencji.

Zastosowane w programie wartości parametrów i algorytm animacji powodują, że natężenie prądu zasilania taśmy w czasie pracy nie przekracza 1 A, gdyż większość diod pozostaje wygaszona, a wypełnienie diod, które świecą, jest niewielkie,

Analizę parametrów czasowych urządzenia należy zacząć od oszacowania czasu transmisji całego bloku danych do 145 układów wynosi około 4,3 ms. Stąd wynika minimalny możliwy odstęp czasowy pomiędzy zainicjowaniem transmisji i rozpoczęciem kolejnej modyfikacji wyświetlanego obrazu, jeśli dane będą modyfikowane w każdym przerwaniu timera kroków animacji, to czas potrzebny na modyfikację danych nie może przekroczyć 5,7 ms. Przy zastosowanym prostym algorytmie animacji wykonanie kroku animacji będzie znacznie szybsze – zajmie ono czas rzędu jednej milisekundy. Nie ma również potrzeby modyfikacji danych co 10 ms.

Ponieważ modyfikacja kolorów świateł jest wykonywana w procedurze obsługi przerwania timera, musimy zagwarantować, że wykonanie tej procedury nie zablokuje przerwań niezbędnych do obsługi interfejsu SPI. Łatwo zauważyć, że podczas wykonywania kroku animacji interfejs SPI zgłosi kilkaset przerwań. W związku z tym niezbędne jest skorzystanie z wielopoziomowego systemu przerwań procesora Cortex-M0 poprzez zróżnicowanie poziomów wywłaszczania przerwań – tak, aby przerwanie SPI przerywało programową obsługę przerwania timera. Ponieważ domyślnie wszystkie przerwania mają ten sam, najwyższy priorytet wywłaszczania (0), niezbędne jest obniżenie priorytetu przerwania SysTick podczas inicjowania mikrokontrolera, przed uruchomieniem timera. Rdzeń Cortex-M0 ma cztery poziomy wywłaszczania – 0…3; w naszym programie obniżymy priorytet wywłaszczania SysTick z 0 do 2 –. Obniżenie priorytetu przerwania SysTick następuje poprzez zapis wartości bitów 31..30 rejestru SHP[1], umieszczonego w module SCB.

Grzegorz Mazur