Odbiornik DMX512 z mikrokontrolerem STM32F030

Nieskomplikowany układ sterujący oświetleniem LED, wyposażony w wejście w standardzie DMX512, został wykonany i zaprojektowany na bazie mikrokontrolera STM32F0. Projekt ma charakter demonstracyjny – urządzenie może równocześnie sterować trzema kanałami PWM o napięciu zasilania do 24 V i prądzie obciążenia do 2 A, które mogą służyć np. do zasilania taśmy z diodami RGB oraz łańcuchem diod RGB wyposażonych we wbudowane sterowniki, zgodne z WS2812 lub PL9823 – własność ta prawdopodobnie nie jest zbyt przydatna w praktyce, ale umożliwia skuteczne testowanie działania sterownika.

Prezentowane rozwiązanie może zostać łatwo przeniesione na inny mikrokontroler rodziny STM32F lub zmodyfikowane np. poprzez zwiększenie liczby kanałów PWM, użycie kanałów PWM do sterowania przetwornicami dla diod LED dużej mocy lub usunięcie sterowania łańcuchem WS2812B.

Standard DMX512

DMX512 w warstwie fizycznej korzysta z łącza w standardzie elektrycznym zgodnym z RS485. Dane są transmitowane przy użyciu linii różnicowej, z szybkością 250 kb/s, w typowym formacie transmisji asynchronicznej, z 8 bitowym słowem i jednym bitem stopu, bez kontroli parzystości.

Transmisja DMX512 następuje w pakietach. Pakiet rozpoczyna się symbolem BREAK i zawiera jednobajtowy kod startu oraz dane o długości nieprzekraczającej 512 bajtów. Stan BREAK powinien trwać nie krócej niż 88 μs. Odstępy czasowe pomiędzy elementami ramki nie mogą przekraczać 1 sekundy. Dłuższy odstęp może być traktowany jako koniec ramki i świadczy o zaniku transmisji, która zgodnie z wymaganiami standardu powinna być ciągle powtarzana.

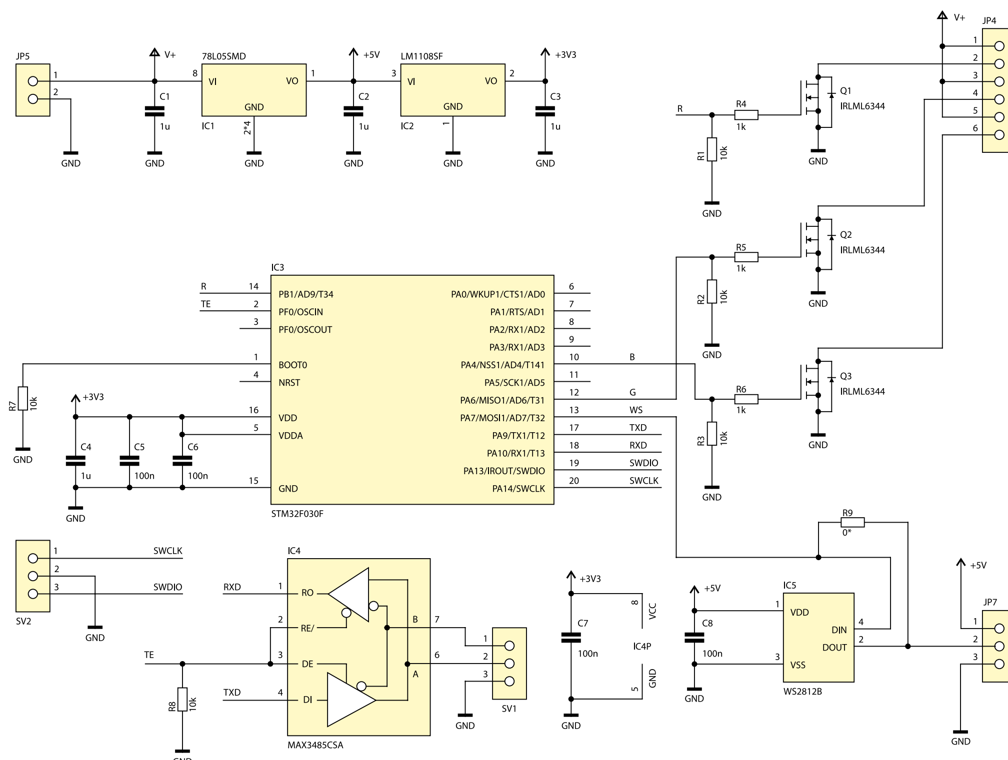
Kod startu identyfikuje zawartość pakietu. Najprostsze systemy DMX512 używają tylko pakietów z kodem 0, które służą do zwykłego sterowania urządzeniami wykonawczymi.

Prosty sterownik świateł powinien rozpoznawać pakiety DMX z kodem startu o wartości 0. Pakiety z kodami startu o innych wartościach są ignorowane. Dane pakietów o kodzie startowym 0 są traktowane jako wartości wysterowania dla poszczególnych urządzeń (np. jasności świateł). Urządzenie interpretuje jeden lub kilka kolejnych bajtów danych, począwszy od bajtu, którego pozycja w bloku danych odpowiada adresowi bazowemu urządzenia.

Schemat układu

Schemat zaprojektowanego sterownika przedstawiono na **rysunku 1**. Układ zasilania składa się z dwóch stabilizatorów liniowych wytwarzających napięcia 5 V i 3,3 V. Układ może być zasilany napięciem od 8 do 24 V albo 5 V (doprowadzonego za stabilizatorem 78L05, np. z taśmy z układami WS2812B). Napięcie 5 V jest na płytce używane do zasilania opcjonalnego elementu WS2812B. Może ono również być użyte do zasilania układu odbiornika RS485 oraz opcjonalnych czujników współpracujących z mikrokontrolerem. Napięcie 3,3 V służy do zasilania mikrokontrolera oraz układu odbiornika RS485 w wersji niskonapięciowej (MAX3485).

W sterowniku zastosowano najmniejszy i najtańszy mikrokontroler rodziny STM32



Rysunek 1. Schemat ideowy odbiornika DMX512 z STM32F030

– STM32F030F4, umieszczony w obudowie TSSOP-20. W projekcie użyto następujących bloków funkcjonalnych mikrokontrolera:

- USART1 – do odbioru sygnału DMX.
- SPI1 – do przysyłania danych sterujących układami WS2812B.
- TIM3 i TIM14 – do generowania przebiegów PWM sterujących jasnością trzech grup diod LED.
- TIM16 – do odliczania maksymalnych odstępów transmisji DMX512.

Programowanie i debugowanie umożliwia złącze trójstykowe, udostępniające tylko sygnały SWDIO i SWCLK. Brak wyprowadzenia sygnału RESET nie przeszkadza w programowaniu ani debugowaniu, pod warunkiem, że oprogramowanie nie zmienia domyślnych funkcji linii SWDIO i SWCLK mikrokontrolera.

Do sterowania taśmy LED użyto tranzystorów NMOS IRLML6344, w obudowach SOT-23, charakteryzujących się małą rezystancją w stanie włączenia i możliwością sterowania z wyjść logicznych w standardzie napięciowym 3,3 V. Rezystory szeregowe R4...6 ograniczają chwilową wartość natężenia prądu bramki podczas przełączania tranzystora, a rezystory ściągające wyjścia mikrokontrolera do masy R1…R3 zabezpieczają przed krótkotrwałym włączeniem świateł podczas włączania zasilania układu, zanim oprogramowanie ustawi na wyjściach mikrokontrolera stan nieaktywny.

Umieszczony na płytce pojedynczy układ WS2812B umożliwia łatwe monitorowanie pracy urządzenia. Jeżeli nie jest on potrzebny, zamiast niego można zamontować rezystor R9, który zapewni dostarczenie sygnału sterującego do zewnętrznego łańcucha WS2812B.

Jako odbiornika sygnału interfejsu DMX512 użyto popularnego układu MAX3485. Podłączenie wejść DE i -RE umożliwia przełączenie z odbioru na nadawanie; możliwość ta nie jest wykorzystywana w prezentowanej wersji oprogramowania. Rezystor R8 zapewnia początkowe ustawienie układu w tryb odbioru danych. W wykonaniu przemysłowym można zastosować izolowany odbiornik RS485, np. typu ADM2587.

Odbiór danych z interfejsu DMX512

Do podstawowej obsługi odbioru danych DMX512 potrzebny jest interfejs UART i timer odmierzający maksymalne czasy charakterystyczne protokołu – odstępy w obrębie ramki i pomiędzy ramkami, wszystkie o długości 1 sekundy. Opisywana implementacja korzysta w tym celu z timera sprzętowego TIM16. W przypadku niedostępności timera sprzętowego można zastosować timer programowy korzystający ze stałych odcinków czasu odmierzanych przez sprzętowe timery generujące przebiegi PWM. Obsługa UART została zrealizowana z użyciem przerwań.

Protokół DMX512 wymaga wykrywania symbolu BREAK, który jest transmitowany poprzez uaktywnienie linii danych (ustawienie jej w stan logiczny 0) na czas dłuższy, niż czas transmisji bajtu. Blok USART mikrokontrolera STM32F0 nie zapewnia jawnego wykrywania tego symbolu, ale jego detekcja może być zrealizowana poprzez sprawdzenie stanu znacznika FE w rejestrze ISR, który jest ustawiany przy wykryciu stanu aktywnego linii danych w czasie transmisji bitu stopu. Wykrycie symbolu BREAK zachodzi, gdy znacznik FE jest ustawiony i odebrany bajt danych ma wartość 0.

Oprogramowanie obsługujące interfejs DMX512 składa się z kodu inicjującego i procedury obsługi przerwania UART. Obsługa przerwania zapewnia reakcję na znacznik początku ramki i odbiór danych. Została ona zrealizowana w konwencji automatu o trzech stanach:

1. Oczekiwania na znacznik początku ramki.
2. Oczekiwania na identyfikator ramki.
3. Odbioru danych.

Przejścia pomiędzy stanami automatu następują w wyniku zdarzeń sygnalizowanych przez UART (poprawny lub błędny odbiór danych, wykrycie symbolu BREAK) oraz odmierzenia przez timer maksymalnej dopuszczonej przez protokół DMX512 wartości czasu pomiędzy tymi zdarzeniami.

Stanem początkowym jest oczekiwanie na znacznik początku ramki – symbol BREAK. Niezależnie od bieżącego stanu automatu, odebranie symbolu BREAK powoduje przejście do stanu oczekiwania na identyfikator ramki, a odebranie błędnych danych (z aktywnym bitem stopu) lub przekroczenie dopuszczalnej przerwy wewnątrz ramki – przejście do stanu początkowego.

Listing 1. Oprogramowanie odbiornika DMX512

```
/*
Tdmx1a-ws - STM32F030 RGB LED strip & WS2812B control via DMX512
gbm, 02'2015
*/

#define PLLMUL 10  //40 MHz
#include „board.h"
#include „ws2812.h"

#define PWM_STEPS  256
#define PWM_FREQ  400
#define PWM_PRE  (SYSCLK_FREQ / PWM_FREQ / PWM_STEPS)
#define BAUD_RATE  250000
#define DMX_TOUT  3000  // ms
#define DMX_CHANNELS 512

//==================================================
void SystemInit(void)
{
    RCC->CFGR = RCC_CFGR_PLLMULV(PLLMUL);
    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_PLLMULV(PLLMUL) | RCC_CFGR_SW_PLL},  // switch to PLL clock
// enable peripherals
    {&RCC->APB1ENR, RCC_APB1ENR_TIM14EN | RCC_APB1ENR_TIM3EN},
    {&RCC->APB2ENR, RCC_APB2ENR_SPI1EN | RCC_APB2ENR_USART1EN | RCC_APB2ENR_TIM16EN},
    {&RCC->AHBENR, RCC_AHBENR_RSTVAL | RCC_AHBENR_GPIOAEN | RCC_AHBENR_GPIOBEN},
// port setup
    // GPIOA AFR[0]: 7..4 - SPI is fun 0
    {&GPIOA->AFR[0], BF4(6, 1) | BF4(4, 4)},  // 6: T3_1, 4:T14_1
    {&GPIOA->AFR[1], BF4(10, 1) | BF4(9, 1)},  // set USART pins 10 - RX, 9 - TX
    {&GPIOA->OSPEEDR, BF2(7, GPIO_OSPEEDR_MED)},  // MOSI - medium speed
    {&GPIOA->MODER, GPIOA_MODER_SWD | BF2(7, GPIO_MODER_AF)  // MOSI
        | BF2(10, GPIO_MODER_AF) | BF2(9, GPIO_MODER_AF)  // UART pins as AF
        | BF2(6, GPIO_MODER_AF) | BF2(4, GPIO_MODER_AF)},  // TIM
    {&GPIOB->AFR[0], BF2(1, 1)},  // 1: T3_4
    {&GPIOB->MODER, BF2(1, GPIO_MODER_AF)},  // MOSI as AF
// SPI setup for WS2812 TX
    {(__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
// USART1 setup
    {(__IO32p)&USART1->BRR, (SYSCLK_FREQ + BAUD_RATE / 2) / BAUD_RATE},
    {&USART1->CR1, USART_CR1_RXNEIE | USART_CR1_RE | USART_CR1_UE},  // enable
// PWM timer setup - TIM14
    {(__IO32p)&TIM14->PSC, PWM_PRE - 1},  // prescaler
    {(__IO32p)&TIM14->ARR, PWM_STEPS - 1},  // period
// blue - CH1
    {(__IO32p)&TIM14->CCMR1, TIM_CCMR1_OC1M_PWM1 | TIM_CCMR1_OC1PE},  // PWM mode 1, buffered preload
    {(__IO32p)&TIM14->CCER, TIM_CCER_CC1E},  // enable CH4, 1 output
    {(__IO32p)&TIM14->CR1, TIM_CR1_ARPE | TIM_CR1_CEN},  // auto reload, enable
```

Listing 1. c.d.

```
// PWM timer setup - TIM3
    {(__IO32p)&TIM3->PSC, PWM_PRE - 1},  // prescaler
    {(__IO32p)&TIM3->ARR, PWM_STEPS - 1},  // period
// green - CH1, red - CH4
    {(__IO32p)&TIM3->CCMR1, TIM_CCMR1_OC1M_PWM1 | TIM_CCMR1_OC1PE},  // PWM mode 1, buffered preload
    {(__IO32p)&TIM3->CCMR2, TIM_CCMR2_OC4M_PWM1 | TIM_CCMR2_OC4PE},  // PWM mode 1, buffered preload
    {(__IO32p)&TIM3->CCER, TIM_CCER_CC4E | TIM_CCER_CC1E},  // enable CH4, 1 output
    {(__IO32p)&TIM3->CR1, TIM_CR1_ARPE | TIM_CR1_CEN},  // auto reload, enable
// DMX timeout timer setup
    {(__IO32p)&TIM16->PSC, SYSCLK_FREQ / 1000 - 1},  // prescaler for 1 ms clock
    {(__IO32p)&TIM16->ARR, DMX_TOUT},       // period
    {(__IO32p)&TIM16->CR1, TIM_CR1_ARPE | TIM_CR1_CEN},  // auto reload, enable
// interrupts and sleep
    {&NVIC->IP[USART1_IRQn / 4], BF8(USART1_IRQn, 0x80)},  // lower priority
    {&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
    return 0;  // suppress warning
}
//================================================
void USART1_IRQHandler(void)
{
    static enum {DS_BREAK, DS_STARTCODE, DS_CHANNELDATA} dmx_state = DS_BREAK;
    static uint16_t cd_addr = 0;
    static uint8_t channel_data[DMX_CHANNELS];

    uint32_t s = USART1->ISR;  // status
    uint8_t c = USART1->RDR;  // received byte

    if (s & USART_ISR_FE)
    {
        // break or error
        USART1->ICR = USART_ICR_FECF;
        dmx_state = c ? DS_BREAK : DS_STARTCODE;
    }
    else
    {
        // byte received
        if (TIM16->SR & TIM_SR_UIF) dmx_state = DS_BREAK;  // timeout
        switch (dmx_state)
        {
        case DS_STARTCODE:
            dmx_state = c ? DS_BREAK : DS_CHANNELDATA;
            break;
        case DS_CHANNELDATA:
            channel_data[cd_addr] = c;
            if (++ cd_addr == DMX_CHANNELS)
            dmx_state = DS_BREAK;
        default: ;
        }
    }
    TIM16->EGR = TIM_EGR_UG;  // restart DMX timer
    if (dmx_state != DS_CHANNELDATA && cd_addr)
    {
        // process data
        TIM3->CCR1 = channel_data[0];
        TIM3->CCR4 = channel_data[1];
        TIM14->CCR1 = channel_data[2];
        WS2812_start((struct wspix_ *) channel_data, cd_addr);
        cd_addr = 0;
    }
    TIM16->SR = ~TIM_SR_UIF;  // clear overflow flag
}
```

Przejście ze stanu oczekiwania na identyfikator ramki do stanu odbioru danych następuje, gdy odebrany identyfikator ma wartość 0.

Na końcu obsługi przerwania USART następuje restart timera TIM16. Ważne jest tu zapewnienie odpowiedniego odstępu czasowego pomiędzy restartem (ustawienie bitu UG w rejestrze EGR) i wyzerowaniem znacznika przeładowania timera (zapis zera do bitu UIF w rejestrze SR). Instrukcje realizujące te dwie akcje zostały celowo rozsunięte.

W stanie odbioru danych następuje gromadzenie danych w buforze. Po odebraniu 512 bajtów danych następuje przejście do stanu oczekiwania na znacznik początku ramki. Licznik/indeks odebranych danych – zmienna cd_addr – służy równocześnie jako znacznik obecności danych.

Dane z ramki są interpretowane po wykryciu końca ramki, a więc wtedy, gdy nastąpiło wyjście ze stanu odbioru danych i wartość licznika danych jest różna od zera.

W przykładowym programie interpretacja danych polega na ustawieniu wypełnień trzech sygnałów PWM według wartości danych trzech pierwszych kanałów oraz przesłaniu danych wszystkich kanałów do łańcucha układów WS2812B. Po interpretacji danych zmienna cd_addr jest zerowana.

Dane odbierane przez interfejs DMX512 są gromadzone w buforze channel_data[]. Po zakończeniu odbioru ramki danych następuje ich interpretacja, która w prezentowanym przykładzie polega na ustawieniu wartości wypełnień trzech kanałów PWM sterujących świeceniem taśmy RGB na podstawie danych o adresach 0, 1 i 2 oraz zainicjowaniu transmisji całej zawartości bufora danych do łańcucha WS2812B. Akcja ta zostaje zainicjowana przy przejściu ze stanu odbioru danych do innego stanu, o ile zostały odebrane jakiekolwiek dane. Ponieważ interpretacja danych nie wymaga długiego czasu, zachodzi ona w procedurze obsługi przerwania USART.

Transmisja danych do łańcucha WS2812B została zrealizowana w sposób opisany w EP 03’2014. Wykorzystano w tym celu linię MOSI interfejsu SPI, a do obsługi SPI użyto przerwań. Ze względu na wymagania czasowe protokołu transmisji stosowanego w układach WS2812B, przerwanie SPI musi mieć priorytet główny wyższy od wszystkich innych przerwań.

Oprogramowanie sterownika nie weryfikuje wymaganego przez standard DMX512 minimalnego ani maksymalnego czasu trwania symbolu BREAK

Inicjowanie mikrokontrolera

Podczas inicjowana mikrokontrolera są kolejno wykonywane następujące czynności:

- Ustawienie taktowania mikrokontrolera na 40 MHz (potrzebne do transmisji danych do WS2812B).
- Włączenie modułów SPI1, USART1, TIM3, TIM14, TIM16.
- Wybór funkcji dla linii portów.
- Zaprogramowanie interfejsów SPI1 i USART1
- Zaprogramowanie timerów TIM3 i TIM14, generujących przebiegi PWM.
- Zaprogramowanie timera TIM16, służącego do wykrywania przekroczenia dopuszczalnych odstępów w ramce DMX.
- Obniżenie priorytetu przerwania USART1.
- Włączenie przerwań z USART1 i SPI1 i ustawienie trybu usypiania procesora przy wyjściu z obsługi przerwania.

Ciekawą cechą prezentowanego rozwiązania jest całkowity brak przerwań od timerów. Podczas pracy urządzenia działają tylko dwie procedury obsługi przerwań.

Oprogramowanie zostało napisane w środowisku Keil MDK-ARM 5.13 z zainstalowanymi pakietami ARM::CMSIS i Keil::STM32F0xx_DFP. Projekt jest zawarty w pliku archiwum tdmx1a.zip Całkowity rozmiar obrazu binarnego programu nie przekracza 1.5 kB.

Grzegorz Mazur