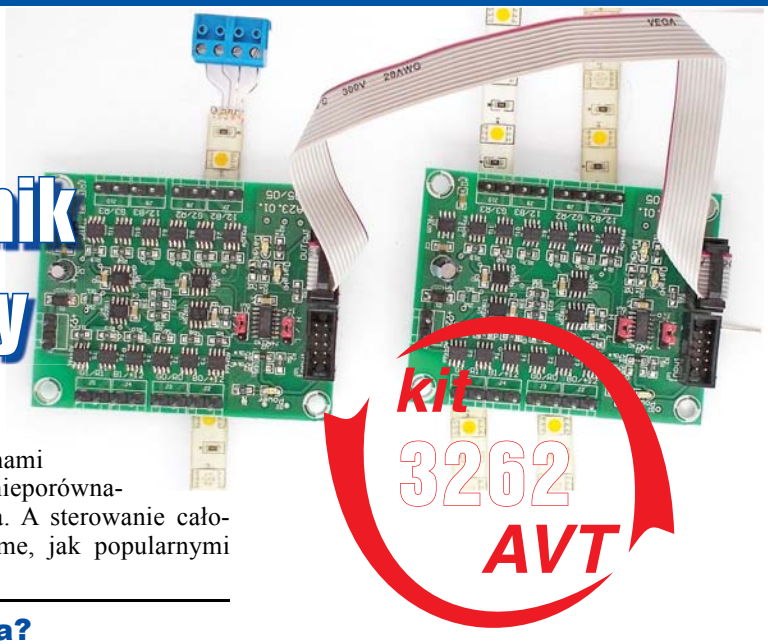
WS2811

Uniwersalny sterownik taśm LED dużej mocy



Taśmy LED są używane do oświetlenia użytecznego i efektowego. Ostatnio popularne stało się podświetlanie nimi schodów w różnorodny atrakcyjny sposób z wykorzystaniem diod o różnych kolorach. Gdy niezależnie sterowanych segmentów taśm jest dużo, problemem staje się zarówno liczba przewodów, jak i wielkość prądu zasilającego. Skutecznym rozwiązaniem jest sterownik – driver oparty na układzie WS2811 i tranzystorach MOSFET. Moduł jest więc wersją turbo popularnych diod WS2812 i ma być sterowany jak WS2812, na przykład za pomocą Arduino.

Do czego to służy?

Opisywany moduł służy do sterowania za pomocą praktycznie dowolnego mikrokontrolera 12-woltowych taśm LED o dowolnych kolorach, w tym taśm RGB i RGBW. Pojedynczy moduł ma 12 kanałów – wyjść, więc może wysterować 12 jednokolorowych taśm LED, cztery RGB lub 3 RGBW. Liczbę kanałów można niemal dowolnie zwiększać, teoretycznie nawet do tysiąca niezależnych odcinków taśm LED, dołączając szeregowo kolejne takie same moduły. A wszystko za pomocą układu scalonego WS2811, blisko spokrewnionego z popularnymi diodami LED WS2812.

Wszystkie drivery są sterowane za pomocą tylko jednego wyprowadzenia mikrokontrolera, co znakomicie upraszcza okablowanie. Przykładowo co 3...4 schody umieszczamy płytkę driverów, a od niej krótkie przewody połączeniowe do taśm LED. Do modułu prowadzą tylko trzy przewody: masa, sygnał sterujący z mikrokontrolera i zasilanie +12V. W proponowanym systemie moduły łączone są pomiędzy sobą popularną taśmą FLAT10 z żyłami połączonymi równolegle, a gdyby prąd zasilania miał być bardzo duży, ponad 10 amperów, taśma będzie się nadmiernie grzać i należy poprowadzić grube przewody zasilające LED-y.

Przedstawiona koncepcja polega więc w sumie na zastąpieniu malutkich diod WS2812(B) opisywanym sterownikiem i taśmami LED, dającymi nieporównanie więcej światła. A sterowanie całością jest takie same, jak popularnymi diodami WS2812.

Jak to działa?

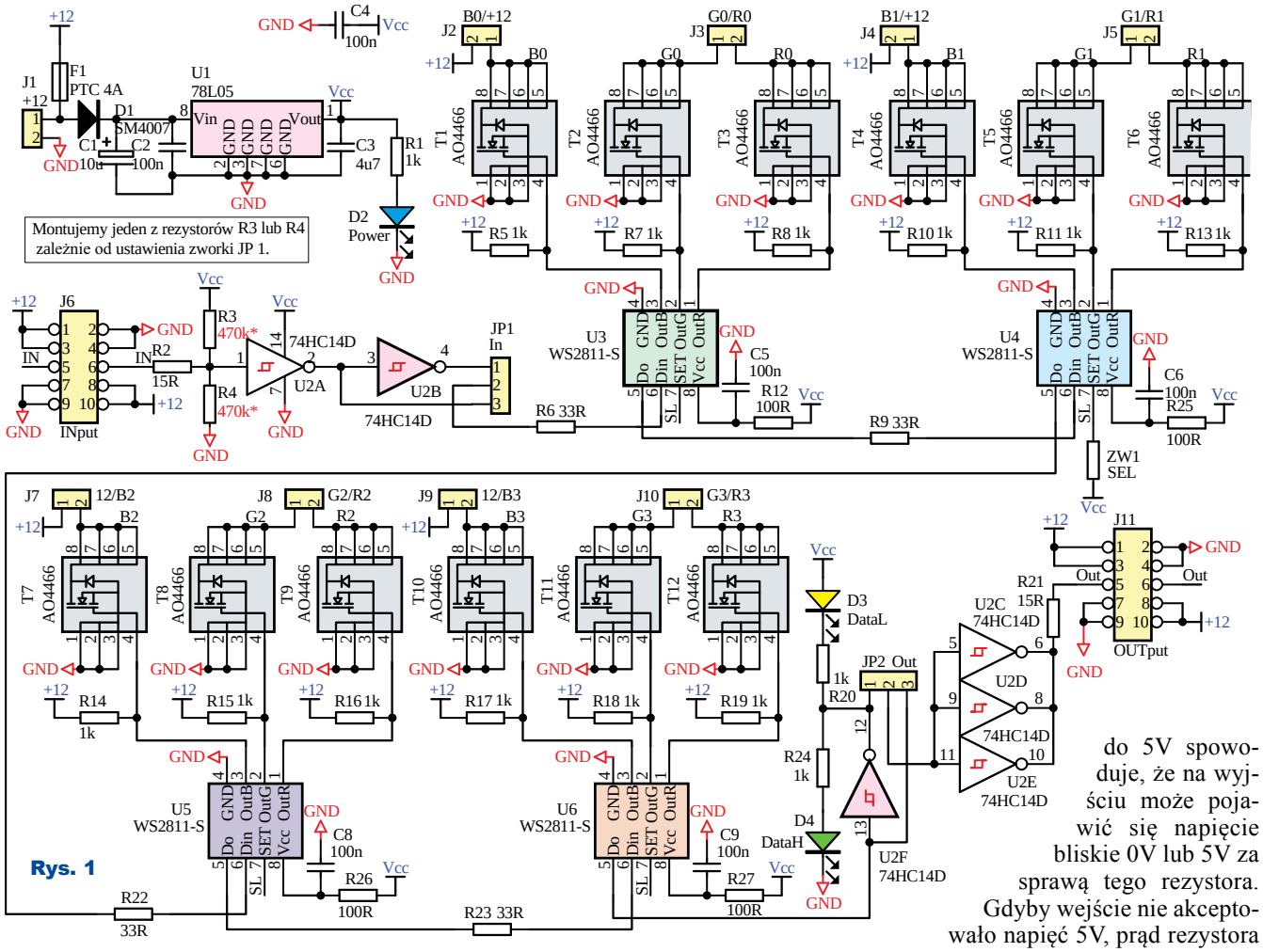
Warto poznać rodzinę układów WS28xx. Obecnie najpopularniejsze są WS2812B – diody zintegrowane ze sterownikiem, które zdobyły dużą popularność za sprawą niskiej ceny, dużych możliwości oraz prostego interfejsu komunikacyjnego. Zaletą WS2812 jest zintegrowanie sterownika z diodą w niewielkiej obudowie SMD5050, co czasem jest jej wadą, bo niemożliwe jest sterowanie innymi diodami (zestawem diod połączonych szeregowo) i napięciem innym niż 5V ±10%. W handlu dostępne są diody PL8923 o średnicy 5 i 8mm, ale na razie większych nie ma.

Protoplastą WS2812 był układ WS2801 dostępny w obudowie DIP-14 oraz SO-14. Ma on 3 wyjścia PWM o rozdzielczości 8-bit, których prąd ustala rezystor (osobny dla każdego wyjścia). Prąd można ustawić w zakresie 5...150mA. Napięcie zasilania układu może wahać się od 3,3V do 5,5V. Interfejs układu jest synchroniczny, 2-przewodowy (zegar i dane), maksymalna częstotliwość taktująca to 25MHz.

WS2812 zawiera w swej strukturze zintegrowany sterownik i diodę LED. Nie ma źródeł prądowych, dlatego powinien być zasilany napięciem 5V ±10%. W Internecie, na forach, pojawiają się dyskusje na temat tego, czy WS2812 ma źródła prądowe, czy nie ma. Moim zdaniem nie ma, ponieważ gdyby były, to zmiana napięcia zasilania w dopuszczanym zakresie 4,5...5,5V nie powodowałaby zmiany jasności świecenia LED, a powoduje. WS2812B jest zmodyfikowaną wersją WS2812. Główne modyfikacje to zabezpieczenie przed odwrotnym podłączeniem zasilania oraz zasilenie LED i sterownika jednym wyprowadzeniem układu, dzięki czemu WS2812B umieszczono w obudowie PLCC4, a nie PLCC6, jak WS2812. Warto wspomnieć o WS2813, które potrafią „pomijać" uszkodzone piksele.

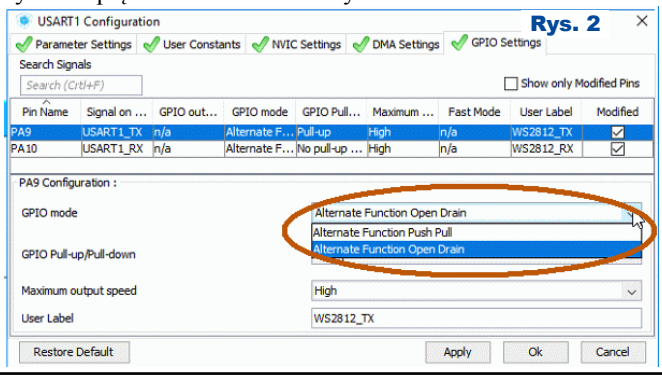
Wracając do tematu, aby sterować prawie dowolnymi zewnętrznymi LED-ami czy zespołem LED, należałoby użyć układu WS2801. Ma on niestety inny interfejs niż popularne WS2812 i jest montowany w stosunkowo dużej obudowie 14 pin. ***Problem rozwiązuje WS2811, do którego można podłączyć różnorodne diody LED. Ma on trzy wyjścia PWM z otwartym drenem. Prąd wypływający z pojedynczego wyjścia może osiągnąć wartość 18,5mA. Napięcie zasilania WS2811 może mieć wartość do 12V.*** W takiej sytuacji szeregowy rezystor na linii Vdd powinien mieć wartość 2,7kΩ (100Ω przy zasilaniu 5V). Niewątpliwą zaletą WS2811 jest możliwość pracy z sygnałem zegarowym 400kHz i 800kHz. Praca z częstotliwością 400kHz może być ułatwieniem przy pisaniu oprogramowania na wolne mikrokontrolery, zwłaszcza gdy nie wykorzystuje się ich sprzętowych możliwości (sterowanie WS2812 przez UART, SPI lub I2C), tylko bezpośrednio steruje pinem GPIO – tak zwane „machanie pinem".

Schemat ideowy proponowanego modułu pokazany jest na **rysunku 1**. Układ zasilany jest ze złącza J1 lub JP1, ewentualnie JP2. Zasilanie układów scalonych zapewnia stabilizator U1. D1 zabezpiecza stabilizator przed skutkami odwrotnego podłączenia zasilania. Tranzystory wykonawcze są zabezpieczone bezpiecznikiem polimerowym



F1. Sygnał sterujący diodami trafia na bramkę Schmitta U2A. Rezystor R3 lub R4 zapobiega odbieraniu zakłóceń, gdy wejście bramki U2A „wisi" w powietrzu, co może mieć miejsce, gdy nie jest ono podłączone lub gdy wyjście mikrokontrolera nie jest skonfigurowane, przez co znajduje się w stanie trzecim, na przykład w czasie resetu, który może trwać bardzo długo, gdy mikrokontroler jest w trybie debugowania. To, czy montowany jest R3, czy R4, zależy od ustawienia zworki na JP1. Gdy sygnał z mikrokontrolera nie jest negowany (na JP1 zwarte piny 2–3), montowany jest R4, gdy musi zostać zanegowany (na JP1 zwarte piny 1–2), montujemy R3. Negowanie sygnału jest konieczne, gdy korzystamy z UART lub I2C do sterowania driverami WS2811, a mikrokontroler nie ma możliwości zanegowania sygnału, co ma miejsce na przykład w przypadku popularnych AVRmega/tiny czy też starszych rodzin STM32. Negowanie sygnału nie jest jedyną funkcją układu U2. Poprawia on zbocza sygnału, który może być doprowadzony długimi przewodami. Inną jego funkcją może być konwertowanie sygnału ze standardu 3,3V na 5V, choć w tej roli, ze względu na gwarantowane poziomy napięć, lepiej sprawdziłby się układ 74HCT04 o rozkładzie wyprowadzeń zgodnym z 74HC14. Przy okazji konwersji napięć z 3,3V na 5V warto wiedzieć, że wiele mikrokontrolerów ma możliwość prawie sprzętowej konwersji poziomów. W tym celu należy skonfigurować wyjście jako otwarty dren – **rysunek 2**. Ważne, aby wyjście to, gdy pracuje w roli wejścia, mogło pracować z napięciami 5V (5V tolerant I/O pin). W przypadku STM32 wejścia takie są oznaczone jako FT. Wejście takie ma nieco inną budowę (**rysunek 3**) niż wejście akceptujące tylko napięcie zasilania (**rysunek 4**). Drobną różnicę zaznaczono na rysunkach 3, 4 czerwoną elipsą. W wejściach akceptujących 5V diody zabezpieczające nie są podłączone do linii zasilania (Vdd), tylko do napięcia V_{DD_FT} (w uproszczeniu do diody Zenera 5V). Dodanie zewnętrznego rezystora podciągającego do 5V spowoduje, że na wyjściu może pojawić się napięcie bliskie 0V lub 5V za sprawą tego rezystora. Gdyby wejście nie akceptowało napięć 5V, prąd rezystora zostałby skierowany przez diodę zabezpieczającą (**rysunek 5**) do linii zasilania Vdd. W takiej sytuacji rezystor podciągający pozwalałby uzyskać napięcie Vdd + 0,6V, co w przypadku WS281x byłoby wystarczające, gdy mikrokontroler jest zasilany z 3,3V, a WS281x z 5V. Według katalogu, gwarantowany poziom wysoki WS2811 przy zasilaniu 5V to 3,5V. 3,3V zasilające mikrokontroler + 0,6V daje 3,9V. Nawet zasilenie mikrokontrolera napięciem 3V daje 100mV rezerwy.

Wyjścia PWM z otwartym drenem układów WS2811 są podciągane do +12V rezystorami 1kΩ, które sterują bramkami tranzystorów MOSFET. Ze



Rys. 2

względu na to, że częstotliwość pracy PWM wynosi tylko około 2kHz, nie zdecydowano się na dodatkowe drivery, pozostawiając jedynie rezystor. Dodatkowe drivery MOSFET-ów byłyby konieczne, gdyby zastosować tranzystory bardzo dużej mocy, które mają dużą pojemność wejściową bramki. Niekorzystnym efektem ubocznym braku driverów jest zanegowanie składowych RGB, gdzie minimalna wartość PWM to wyjście rozwarte, maksymalna – zwarcie z masą. Zanegowanie danych w programie nie stanowi kłopotu. To, że po włączeniu zasilania wszystkie wyjścia będę ustawione na 100% mocy, może, ale nie musi być problemem. W przypadku awarii sterownika, oświetlenie można załączyć przez podanie napięcia zasilającego. Wszystko zależy od tego, do czego urządzenie będzie wykorzystane. Jeśli zanegowanie sygnału jest wymagane, konieczna jest zmiana konstrukcji urządzenia i dodanie driverów dla tranzystorów MOS lub, w najprostszej, niezbyt doskonałej wersji, dwóch układów 74HC(LS)07.

Sygnał z ostatniego układu WS2811 jest transmitowany na wyście dostępne na złączu JP2. Dodatkowo jedna z bramek układu U2 steruje diodami LED D3, D4. Dzięki nim można łatwo ocenić, czy polaryzacja sygnału ustawiona na JP1 jest poprawna i czy transmisja danych ma miejsce. JP2 umożliwia udostępnienie na wyjściu JP2 sygnału prostego lub zanegowanego. Rezystory R2, R21, podobnie jak R6, R9, R22, R23, poprawiają dopasowanie wejść i wyjść. Częstotliwość pracy układów cyfrowych 800kHz nie jest (w dzisiejszych czasach) duża. Rezystory te nie zaszkodzą, a w razie problemów można dobrać ich wartość do warunków pracy.

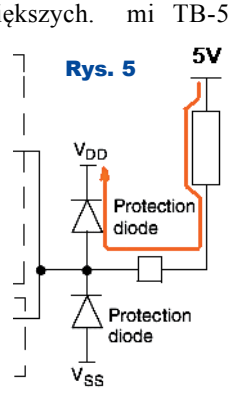
Montaż i uruchomienie

Układ można zmontować na płytce drukowanej, której projekt pokazany jest na **rysunku 6**. Standardowo montujemy układ, zaczynając od elementów najmniejszych, a kończąc na największych. Zaleca się jednak, w pierwszej kolejności, uruchomić stabilizator 5V. Warto zdecydować, czy sygnał do WS2811 będzie prosty, czy zanegowany i wlutować odpowiednio R4 lub R3. Gdy stabilizator pracuje poprawnie, można zamontować pozostałe elementy. Zależnie od tego, czy układy mają pracować z częstotliwością 400kHz, czy 800kHz, wlutowujemy ZW1 (400kHz) lub nie (800kHz). ZW1 to rezystor 0R w obudowie 1206. Podczas montażu należy zwrócić uwagę na poprawne przylutowanie tranzystorów. Ze względu na grube ścieżki, które skutecznie odprowadzają ciepło od grota, podobnie jak i wyprowadzenia tranzystorów, lutowanie najlepiej wykonać lutownicą na gorące powietrze. Gdy lutowanie wykonywane jest lutownicą z grotem, należy chwilę grzać wyprowadzenie tranzystora i ścieżkę, po czym przyłożyć stop lutowniczy, koniecznie z topnikiem. Stop po chwili rozpłynie się, skutecznie łącząc nóżkę tranzystora z miedzianą ścieżką. Pierwsza nóżka, zwłaszcza drenu, który jest podłożem, będzie sprawiała najwięcej kłopotów, z kolejnymi będzie łatwiej, bo zarówno ścieżka, jak i tranzystor będzie już rozgrzany. Ze względu na to, że trudno opisać ten proces, w materiałach dodatkowych można znaleźć filmy z poprawnego, jak i niepoprawnego lutowania. Niepoprawne przylutowanie wyprowadzeń może uszkodzić tranzystory przy dużym prądzie, ponieważ obudowa nie będzie w stanie rozproszyć wymaganej mocy. Układ zmontowany poprawnie nie wymaga uruchamiania. Jedyną czynnością jest założenie lub wlutowanie zworek JP1 i JP2 zależnie od potrzeb.

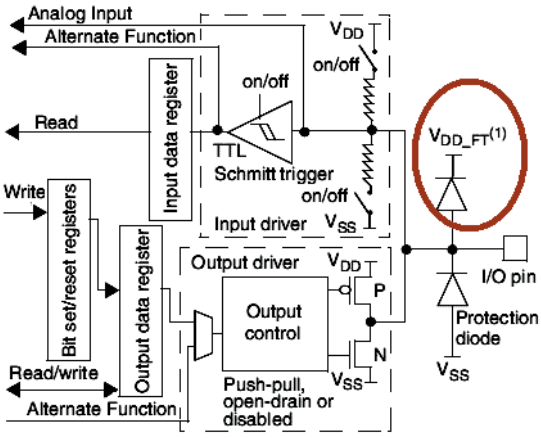
Prąd pojedynczego wyjścia może osiągnąć wartość 1...2A. Duży prąd może wymagać dolutowania dodatkowego drutu do ścieżek zasilających LED-y, ponieważ przy prądzie 2A na jedno wyjście sumaryczny prąd może osiągnąć 24A. Jeszcze większe prądy wyjść (do 7A) wymagałyby zmodyfikowania rysunku płytki, aby wyposażyć tranzystory w radiatory na warstwie miedzi.

Na koniec wspomnę o zastosowanych złączach. Można użyć złączy ARK, ale wygodniejsze są TB-5.0-PP-2P wraz z listwami TB-5.0-PIN24. Dzięki temu, płytkę można szybko odłączyć, np. w celu wymiany. Nie ma konieczności odkręcania dziesiątek zacisków, a co ważne, luźnie przewody zasilające nie spowodują zwarcia.

Rys. 5



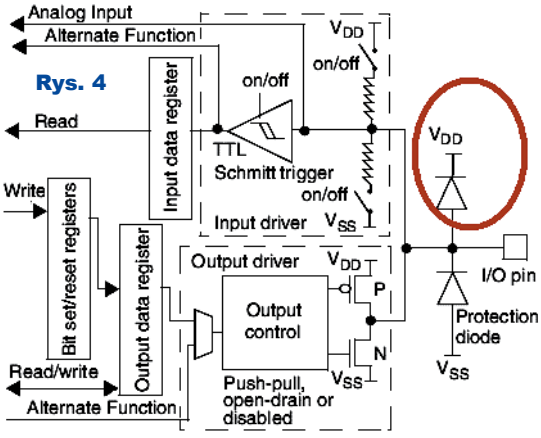
Basic structure of a 5-Volt tolerant I/O port bit



1. V_{DD_FT} is a potential specific to 5-Volt tolerant I/Os, and different from V_{DD}.

Rys. 3

Basic structure of a standard I/O port bit

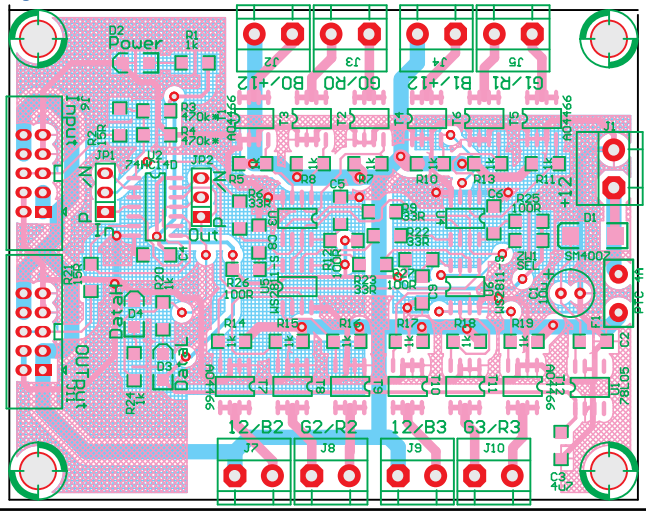


Rys. 4

Tylko dla dociekliwych

Opisany moduł nie zawiera mikrokontrolera czy układu programowalnego, a jest jedynie układem wykonawczym dużej mocy („TurboWS2812"), dlatego musi być sterowany tak samo, jak diody WS2812(B). To sterowanie można zrealizować w różny sposób, nawet za pomocą Arduino. Chciałbym jednak przekazać bardziej dociekliwym trochę informacji na temat sterowania WS281x. Najczęstszym, często wystarczającym, sposobem

Rys. 6



jest programowe sterowanie układami. Nie jest ono zalecane, ponieważ w tym czasie muszą być zawieszone przerwania. Jeśli nawet mikrokontroler jest bardzo szybki i przerwań nie trzeba zawieszać, to nie da się skorzystać z dobrodziejstw przerwań czy DMA. CPU jest absorbowany przez cały czas transmisji danych do LED, co przy 1000 diod LED trwa ok. 30ms, a 60ms w przypadku taktowania 400kHz. Jeśli konieczna jest płynna animacja, LED trzeba odświeżać ok. 30 razy na sekundę. W takiej sytuacji CPU nie robi nic poza ciągłą transmisją danych i na obliczenia związane z animacją nie ma już czasu. Ponadto łatwo policzyć, ile znaków może przyjąć UART w czasie 30ms czy tym bardziej 60ms przy dużych prędkościach, co wymusza stosowanie dużych buforów, a w przypadku AVR, gdzie trzeba zawiesić przerwania, ile znaków przepadnie? Przy popularnej prędkości 115200 8N1, w ciągu 30ms UART może odebrać blisko 450 znaków, przy 921600 – ponad 2,7kB, przy dość niskiej prędkości 9600 w ciągu 30ms transmitowane może być prawie 29 znaków. Ta sama sytuacja ma miejsce w przypadku sprzętowego interfejsu USB, tyle że tu w ciągu 1ms może zostać przesłane 1500 bajtów lub więcej. Należy też odrzucić mechanizm sterowania GPIO w przerwaniach. Nawet dla ARM, przełączanie GPIO co ok. 420ns (2,4MHz) jest dużym wyzwaniem. Jeśli nawet się uda, to CPU będzie bardzo obciążony. Problem utraty znaków rozwiązuje wykorzystanie sprzętowego interfejsu, np. UART. Jest on o tyle wygodny, że aby wysłać 24 bity danych do WS2812, wystarczy osiem bajtów w pamięci RAM, jeśli danych nie transkodujemy „w locie”, na co w przypadku AVR przeważnie zabraknie czasu CPU, a w przypadku DMA możliwości transkodowania danych nie ma.

W przypadku SPI na jedną diodę potrzeba dziewięciu bajtów. Kolejną zaletą USART jest to, że WS281x są bardziej liberalne na przedłużenie poziomu niskiego pomiędzy bitami (dane z UART muszą być negowane!) niż wysokiego. W przypadku SPI wymagane jest, aby pomiędzy transmisjami nie było przerw, bo transmisja może kończyć się zarówno poziomem H, jak i L. O ile używa się DMA, to nie problem, ale w przypadku AVRmega/tiny bez DMA, przerwania wywoływane ok. 300000 razy na sekundę są dużym obciążeniem. W przypadku UART transmisja zawsze kończy się (po zanegowaniu) poziomem niskim. Kolejną zaletą, ujawniającą się w przypadku AVR, jest możliwość wykorzystania 2-bajtowego FIFO. Dzięki temu można w jednym przerwaniu wysłać dwa znaki, co zmniejszy częstotliwość przerwań do ok. 150000 razy na sekundę.

To, że użycie DMA nie jest trudne, pokażę na przykładzie ARM STM32F072. W pierwszej kolejności należy skonfigurować UART do pracy z częstotliwością 2,4...2,5MHz 7N1. W przypadku F072 można zanegować sygnał TX – **rysunek 7**. Ponadto można ustawić wyjście w konfiguracji otwarty dren (rysunek 3).

Program wysyłający dane jest banalnie prosty, gdy dane są gotowe do wysłania, wystarczy wywołać funkcję:

*HAL_UART_Transmit_DMA(&huart1, scr, WSLED*8);*

gdzie „WSLED” to liczba LED w łańcuchu. Prawda, że proste?

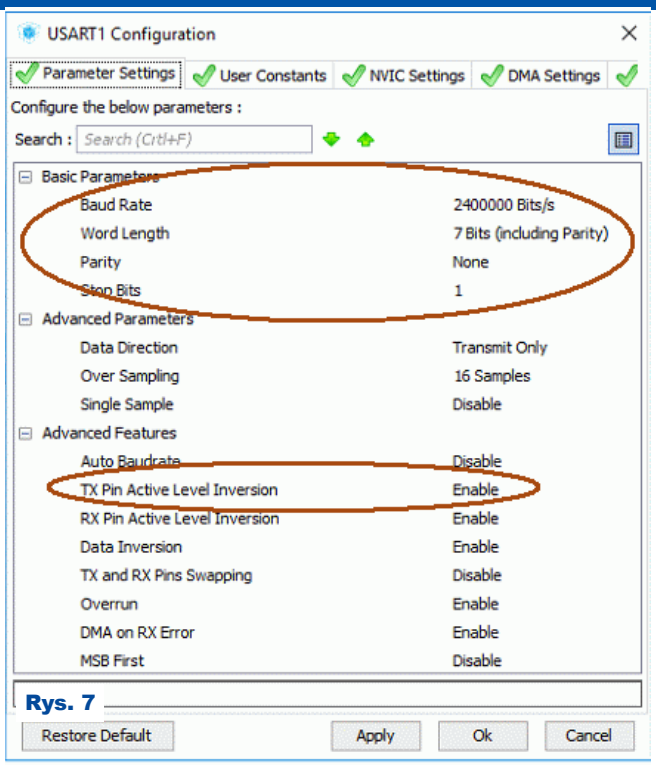
Jeśli dane mają być wysyłane z maksymalnym odświeżaniem, trzeba określić, kiedy DMA skończyło transmisję. W tym celu w pliku „stm32f0xx_it.c” w funkcji „DMA1_Channel2_3_IRQHandler(void)”, należy pomiędzy:

/ USER CODE BEGIN DMA1_Channel2_3_IRQn 0 */*

a

/ USER CODE END DMA1_Channel2_3_IRQn 0 */*

dopisać funkcję określającą źródło przerwania, która za pomocą timera wygeneruje pauzę co najmniej 50µs, po czym zainicjuje kolejną transmisję. Opis, jak określić, czy przerwanie pochodzi od DMA, które wysyła dane, czy też przerwanie dotyczy wysłania wszystkich danych, czy tylko ich części, wykracza poza ramy tego artykułu. Opis zająłby pewnie kilka stron, ponieważ trzeba wiedzieć, który kanał DMA odpowiada za wysyłanie, który za odbieranie danych, jakie kanały przypisane są do poszczególnych układów peryferyjnych. W tym pomocna może być analiza funkcji „HAL_DMA_IRQHandler”. Naturalnie, jeśli będzie zainteresowanie Czytelników, napiszę artykuł szczegółowo opisujący transmisję przez DMA z wykorzystaniem przerwań od tego jakże pożytecznego układu (proszę o e-maile w tej sprawie).



Rys. 7

Wracając do głównego tematu, jak stworzyć „surowe” dane do wysłana? WS281x mają interfejs synchroniczny, w którym zero reprezentuje sekwencja bitów HLL, jedynkę HHL (odsyłam do noty katalogowej WS281x). Wymaga to, aby pojedynczy bit danych koloru rozbić na trzy bity dla UART. Wykorzystany jest fakt, że transmisja 7N1 zawiera dziewięć bitów, na które składają się:
1 bit startu = 0
7 bitów danych

Wykaz elementów

Rezystory 1206:	
R1,R5,R7,R8,R10,R11,R13,R14,R15,R16,R17,R18,R19,R20,R24	1kΩ
R2,R21	15Ω
R6,R9,R22,R23	33Ω
R3,R4	470kΩ (patrz tekst)
R12,R25,R26,R27	100Ω
C1	10u ce6.3/2.5
C3	4u7 ceram. 1206
C2,C4,C5,C6,C8,C9	100nF ceram. 1206
U1	78L05 SO-08
U2	74HC14D SO-14
U3,U4,U5,U6	WS2811-S SO-08
T1-T12	AO4466 SO-08
D1	SM4007
D2	Dioda LED niebieska 1206
D3	Dioda LED żółta 1206
D4	Dioda LED zielona 1206
J1-J5,J7,J9	TB-5.0-PP-2P + TB-5.0-PIN24
J6,J11	ZL231-10PG (KG) + AVP-10 + taśma FLAT10
JP1,JP2	Goldpin 3×1 2,54mm + jumper
F1	PTC 4A

Komplet podzespołów z płytką jest dostępny w Sklepie AVT jako zestaw AVT3262

1 bit stopu = 1
Można to zapisać tak:
0ddddddd1
Po zanegowaniu sekwencja wygląda jak poniżej:
1ddddddd**0**
Wysłanie trzech bitów zero:
100100100
Wysłanie trzech bitów jeden:
110110111**0**
Można więc zauważyć, że sekwencja:
1x**01**x**01**x**0**
jest stała. Wystarczy w miejsca „x" wstawić kolejno transmitowane bity. Realizuje to funkcja pokazana w nastepującym **listingu**:

```
void  WS2812B_transcodeRGB(uint8_t  red,  uint8_t  green,
uint8_t  blue,  byte  *buffer  )     //Przekonwertuj  podany
kolor w formacie GRB na ciag 8 bajtów dla USART
{
  for (byte x = 0; x < 8; x++) *(buffer + x) = b00100100;
// Czyścimy bufor
  if ( green & 0x80 ) *(buffer + 0) |= ~(b01111110);
  if ( green & 0x40 ) *(buffer + 0) |= ~(b01110011);
  if ( green & 0x20 ) *(buffer + 0) |= ~(b00011111);
  if ( green & 0x10 ) *(buffer + 1) |= ~(b01111110);
  if ( green & 0x08 ) *(buffer + 1) |= ~(b01110011);
  if ( green & 0x04 ) *(buffer + 1) |= ~(b00011111);
  if ( green & 0x02 ) *(buffer + 2) |= ~(b01111110);
  if ( green & 0x01 ) *(buffer + 2) |= ~(b01110011);

  if ( red   & 0x80 ) *(buffer + 2) |= ~(b00011111);
  if ( red   & 0x40 ) *(buffer + 3) |= ~(b01111110);
  if ( red   & 0x20 ) *(buffer + 3) |= ~(b01110011);
  if ( red   & 0x10 ) *(buffer + 3) |= ~(b00011111);
  if ( red   & 0x08 ) *(buffer + 4) |= ~(b01111110);
  if ( red   & 0x04 ) *(buffer + 4) |= ~(b01110011);
  if ( red   & 0x02 ) *(buffer + 4) |= ~(b00011111);
  if ( red   & 0x01 ) *(buffer + 5) |= ~(b01111110);

  if ( blue  & 0x80 ) *(buffer + 5) |= ~(b01110011);
  if ( blue  & 0x40 ) *(buffer + 5) |= ~(b00011111);
  if ( blue  & 0x20 ) *(buffer + 6) |= ~(b01111110);
  if ( blue  & 0x10 ) *(buffer + 6) |= ~(b01110011);
  if ( blue  & 0x08 ) *(buffer + 6) |= ~(b00011111);
  if ( blue  & 0x04 ) *(buffer + 7) |= ~(b01111110);
  if ( blue  & 0x02 ) *(buffer + 7) |= ~(b01110011);
  if ( blue  & 0x01 ) *(buffer + 7) |= ~(b00011111);
}
```

Tak naprawdę to nie są jeszcze wszystkie funkcje potrzebne do poprawnej obsługi LED. Charakterystyka świecenia diody LED i oka ludzkiego nie jest liniowa. Aż prosi się zastosowanie odpowiedniej korekty. Kolejną przydatną funkcją jest interpolacja, umożliwiająca płynne przejście z jednej barwy w drugą. Tak jak w przypadku DMA i przerwań, tak i w przypadku korekty i interpolacji jest to temat na osobny artykuł. Jeśli Czytelnicy wykażą zainteresowanie poparte e-mailami, stosowny artykuł pojawi się na łamach EdW.

SaS
sas@elportal.pl