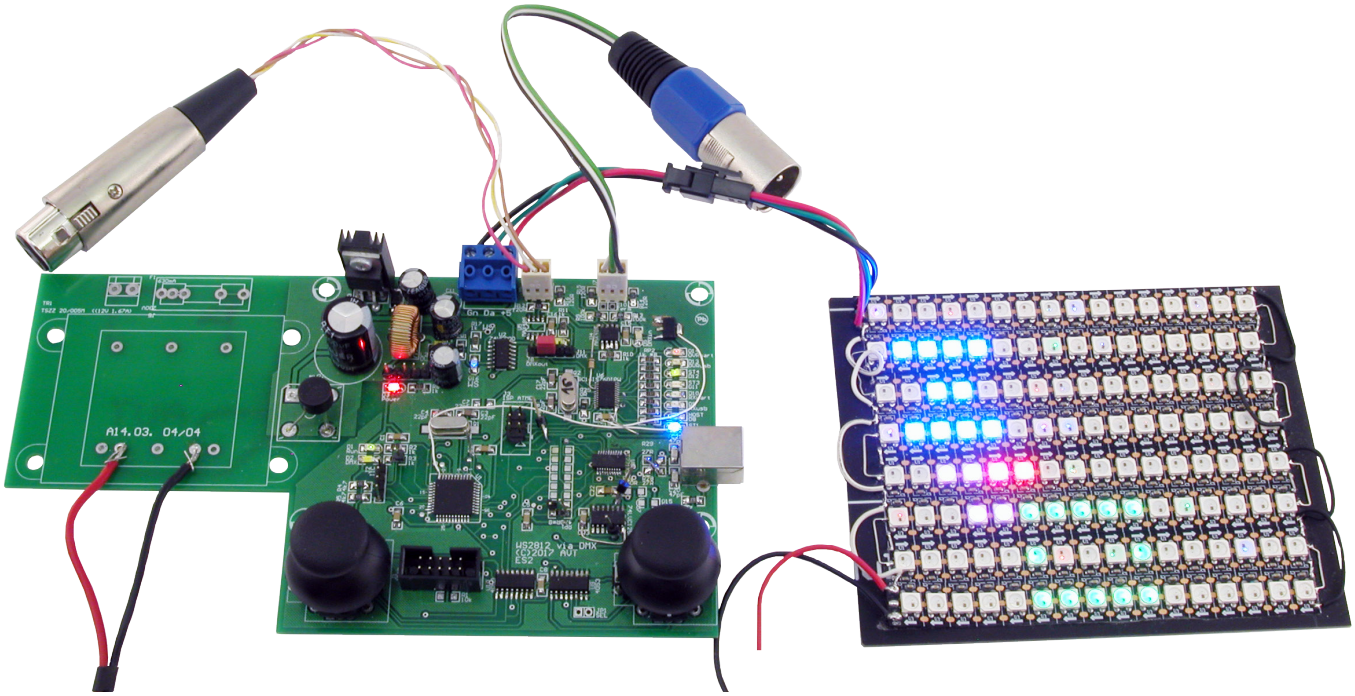


Sterowanie diod WS2812 poprzez DMX

Sterowanie diodami WS2812 za pośrednictwem DMX nie jest pomysłem wyjątkowym, jednak zrealizowanie tego zadania z użyciem interfejsu UART i systemu przerwań na bazie mikrokontrolera AVR nie jest zadaniem łatwym. Natomiast praca w tym samym czasie jako urządzenie DMX oraz odbiór sygnału z USB, z prędkością 1 Mb/s, wydaje się niemożliwe. Opracowanie takiego systemu wymagało zastosowania nieszablonowych rozwiązań.

Głównym zadaniem urządzenia jest sterowanie diodami WS2812 poprzez interfejs DMX. W ten sposób można kontrolować 170 diod z 24-bitową paletą kolorów lub 512 diod z paletą 8-bitową. Sterownik może obsługiwać kilka wirtualnych ekranów (warstw) niezależnych od siebie. Warstwy można użyć do połączenia dwu strumieni DMX (jedynym ograniczeniem może być tylko wydajność mikrokontrolera) lub naniesienia na obraz z DMX innych, niezależnie animowanych obiektów. Ekrany mają priorytety tak jak duszki (sprite's) w C-64 czy Atari. Warstwa o najniższym numerze ma najwyższy priorytet i zasłania wszystko, co się pod nią znajduje. Aby zademonstrować możliwości sterownika, diody ułożono w prostokąt 16×8 LED.

Budowa i działanie

Schemat urządzenia podzielono na bloki, których schematy pokazane zostały na **rysunkach 1...5**. Mikrokontroler taktowany zegarem 20 MHz (ATmega1284 to nieliczny układ z rodziny AVR Mega, który może być taktowany zegarem większym niż 16 MHz) steruje diodami LED oraz odbiera transmisję DMX. Układ został wyposażony w 16 kB pamięci RAM. Tak duża pamięć jest wymagana dlatego, że obsługa jednej diody z użyciem UART wymaga 8 bajtów pamięci (z użyciem SPI 9 bajtów). Przy 512 diodach daje to 4 kB pamięci, a trzeba gdzieś jeszcze umieścić dane, stos, bufor DMX. Większość dużych mikrokontrolerów AVR ma 4 kB, nieliczne 8 kB lub więcej. Pamięć RAM można zaoszczędzić, dekodując dane dla diod metodą „w locie", niestety zastosowany układ jest zbyt wolny. Co prawda diodami dałoby się sterować, ale na dekodowanie DMX pewnie nie starczyłoby już czasu.

Zasilacz urządzenia zrealizowano w typowym układzie aplikacyjnym stabilizatora impulsowego LM2576. Może dostarczyć prąd o wartości do 3 A, do zasilania diod LED oraz układów sterujących. Można zastosować

Dodatkowe materiały do pobrania ze strony www.media.avt.pl

W ofercie AVT* AVT5607

Podstawowe parametry:
- sterowanie diodami WS2812 poprzez interfejs DMX,
- umożliwia kontrolowanie 170 diod z 24-bitową paletą kolorów lub 512 diod z paletą 8-bitową,
- obsługa kilku wirtualnych ekranów (warstw),
- sterowanie uwzględnia korektę gamma,
- sterowanie zarówno w trybie konwertera DMX (odbiór danych z USB), jak i poprzez odbieranie z RS485/422,
- zaimplementowane dwie proste gry: Tenis i Galaga, dla zaprezentowania możliwości urządzenia.

Projekty pokrewne na www.media.avt.pl:

AVT-5816	Regulator balansu tonów (EP 10/2020)
AVT-5637	Wielokanałowy regulator głośności VCA (EP 8/2018)
AVT-5629	Cyfrowy regulator głośności z układem PT2257 (EP 6/2018)
AVT-3222	Sterowany dowolnym pilotem potencjometr audio z przekaźnikiem (EdW 5/2018)

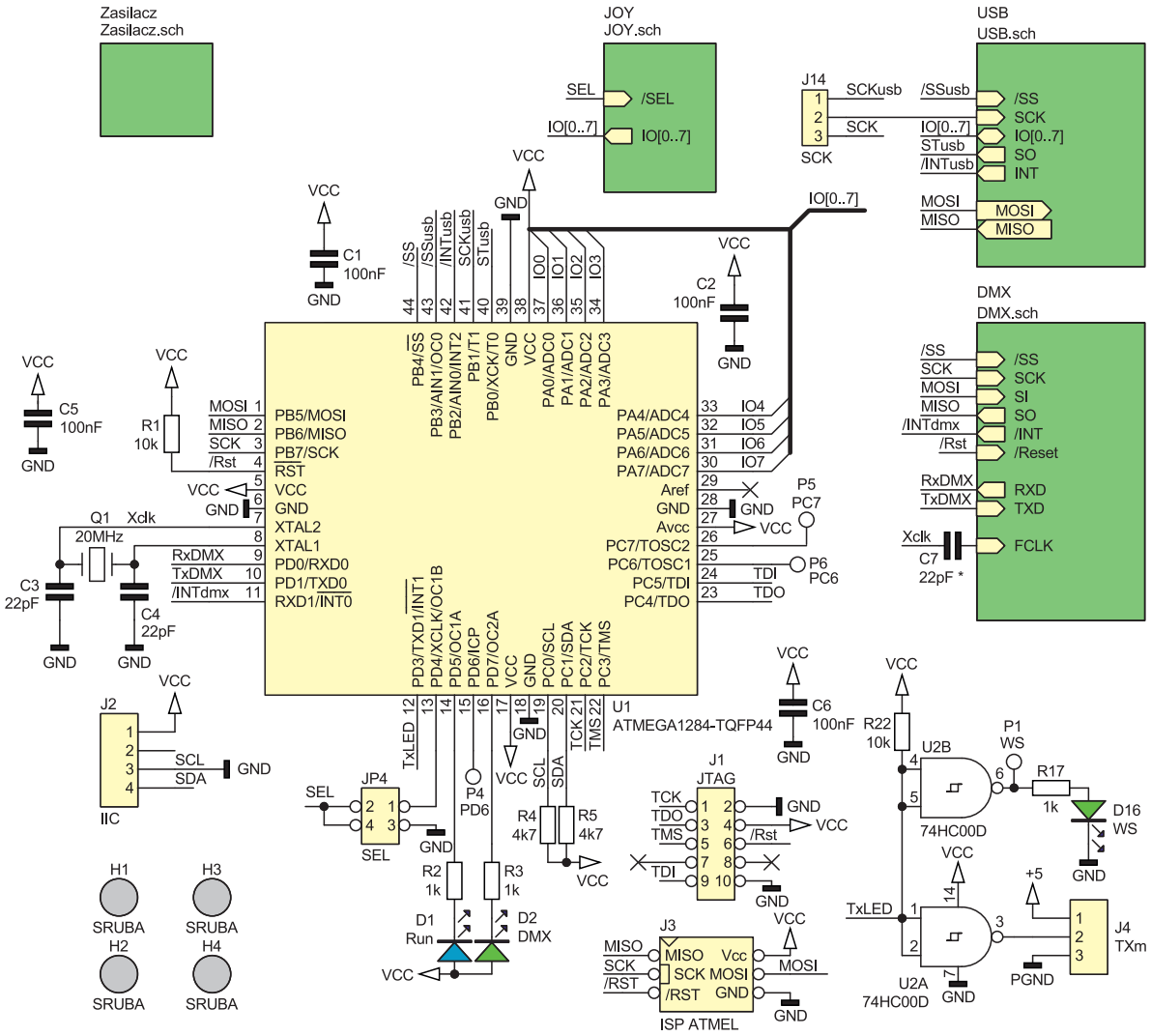
*** Uwaga! Elektroniczne zestawy do samodzielnego montażu. Wymagana umiejętność lutowania!**

Podstawową wersją zestawu jest wersja [**B**] nazywana potocznie KIT-em (z ang. zestaw). Zestaw w wersji [**B**] zawiera elementy elektroniczne (w tym [**UK**] – jeśli występuje w projekcie), które należy samodzielnie wlutować w dołączoną płytkę drukowaną (PCB). Wykaz elementów znajduje się w dokumentacji, która jest podlinkowana w opisie kitu.

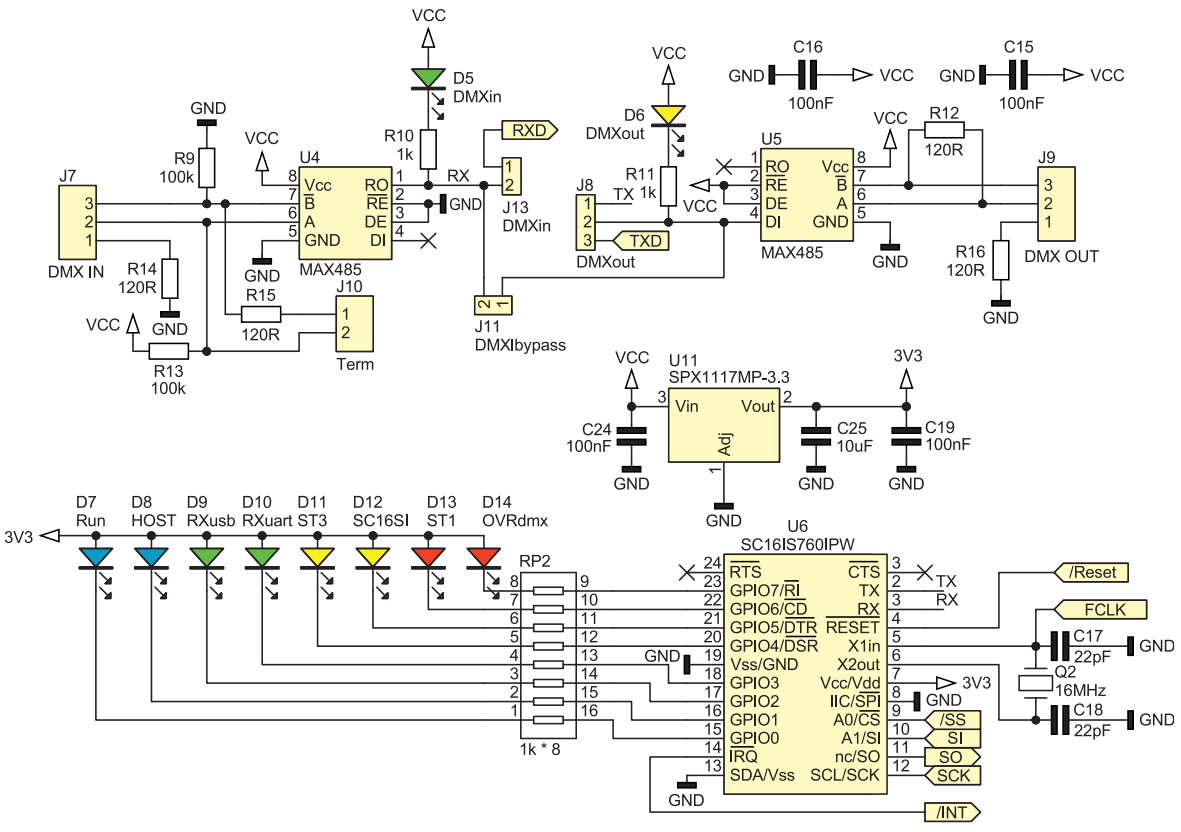
Mając na uwadze różne potrzeby naszych klientów, oferujemy dodatkowe wersje:
- wersja [**C**] – zmontowany, uruchomiony i przetestowany zestaw [**B**] (elementy wlutowane w płytkę PCB)
- wersja [**A**] – płytka drukowana bez elementów i dokumentacji

Kity w których występuje układ scalony wymagający zaprogramowania, mają następujące dodatkowe wersje:
- wersja [**A+**] – płytka drukowana [**A**] + zaprogramowany układ [**UK**] i dokumentacja
- wersja [**UK**] – zaprogramowany układ

Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik pdf! Podczas składania zamówienia upewnij się, którą wersję zamawiasz! *http://sklep.avt.pl*. W przypadku braku dostępności na *http://sklep.avt.pl*, osoby zainteresowane zakupem płytek drukowanych (PCB) prosimy o kontakt via e-mail: kity@avt.pl.



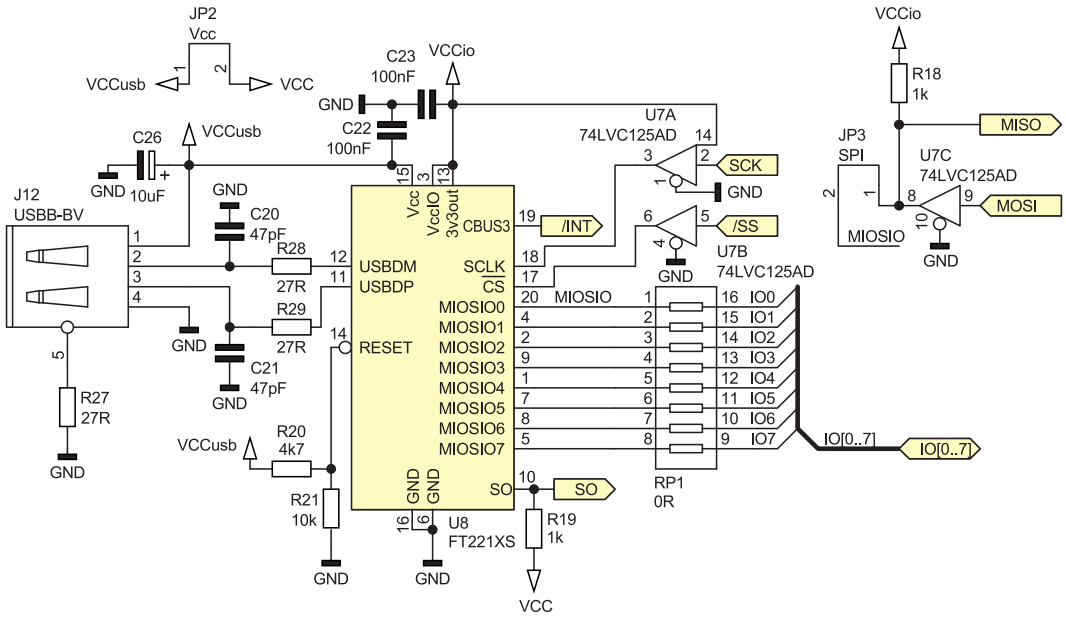
Rysunek 1. Schemat bloku głównego z mikrokontrolerem sterującym



Rysunek 2. Schemat bloku interfejsu DMX

układ z ustalonym napięciem (wersja oznaczona -5.0), jak i regulowanym (oznaczenie -ADJ). Zależnie od typu układu trzeba zastosować odpowiednie rezystory R6 i R7. Maksymalne napięcie wejściowe wynosi 40 V. Trzeba pamiętać, aby wejściowy kondensator filtrujący miał odpowiednie napięcie znamionowe.

Prąd pobierany przez sterownik bez dołączonych diod LED wynosi ok. 100 mA. Maksymalny pobór prądu przez jedną diodę to blisko 50 mA (dla diod w obudowie 5050 przypada 16 mA na każdą z diod RGB). Wydajność zasilacza (3 A) wystarczy na zaświecenie na biało z maksymalną jasnością około 60 diod. Jednak w praktyce rzadko się zdarza, aby wszystkie diody świeciły jednocześnie na biało.



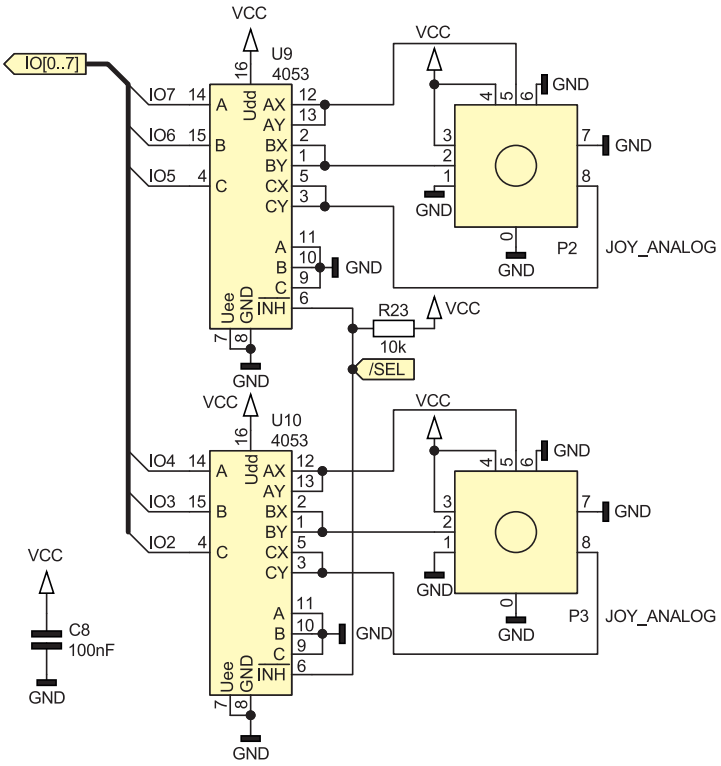
Rysunek 3. Schemat bloku komunikacji USB

Sterowanie diodami za pomocą UART

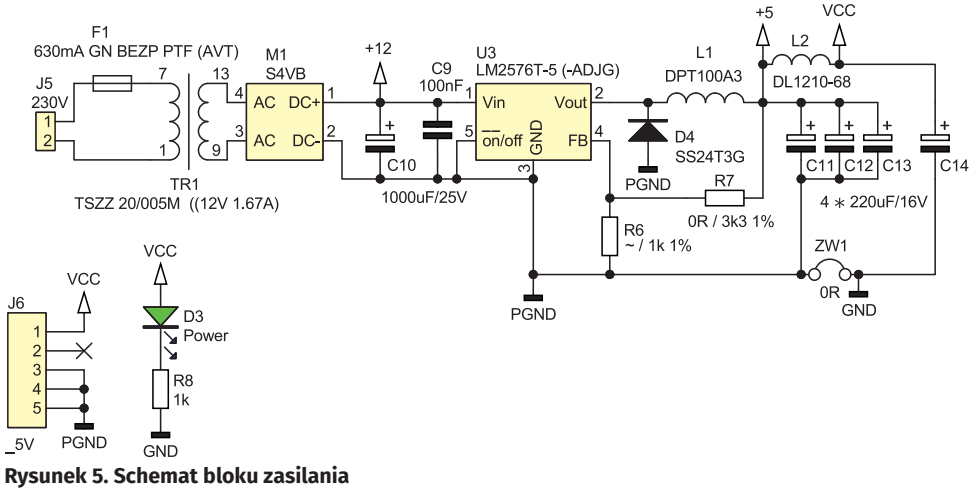
Diody sterowane są impulsami o okresie 1,25 µs ±600 ns (**rysunek 6**), co daje przepływność na poziomie 800 kb/s. Transmisja każdego bitu jest podzielona na trzy odcinki czasowe, z których pierwszy zawsze jest stanem wysokim, drugi decyduje o wartości przesyłanej informacji a trzeci zawsze ma stan niski (**rysunek 7**). Zatem przepływność UART powinna wynosić 2,4 Mb/s (1,6...4,6 Mb). Ramka transmisyjna UART 7N1 składa się z dziewięciu odcinków czasowych – bit startu, 7 bitów danych i 1 bit stopu. W jednym bajcie można przesłać 3 bity sterujące diodą typu WS2812. Bit startu i stopu ma z góry ustaloną wartość. Niestety bit startu to zero, a bit stopu jeden. Po zanegowaniu sygnału UART otrzymujemy sygnał wymagany przez sterownik diod LED typu WS2812. Aby wysłać trzy bity o wartości 0 poprzez UART, należy wpisać wartość $26 (binarnie 100110). Działanie tego rozwiązania zostało zobrazowane za pomocą **tabeli 1**.

Na pomarańczowo zaznaczono bity startu i stopu, na żółto stałą wartość jaka musi być wysyłana poprzez UART, litery A, B, C ustalają wartości bitów 1, 2, 3, przesyłanych do diody LED.

Jaką maksymalną przepływność można uzyskać za pomocą UART w AVR? Według noty katalogowej: Fosc/8·UBRR+1. Dla 20 MHz i UBRR=1 otrzymamy 1,25 Mb/s, czyli dwa razy za wolno. Okazało się, że wpisując zero dla UBRR, otrzymujemy przepływność 2,5 Mb/s (**rysunek 8**).



Rysunek 4. Schemat bloku z joystickami



Rysunek 5. Schemat bloku zasilania

Data transfer time(TH+TL=1.25μs±600ns)

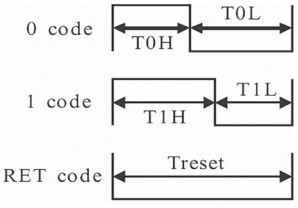
T0H	0 code ,high voltage time	0.4us	±150ns
T1H	1 code ,high voltage time	0.8us	±150ns
T0L	0 code , low voltage time	0.85us	±150ns
T1L	1 code ,low voltage time	0.45us	±150ns
RES	low voltage time	Above 50μs	

Rysunek 6. Fragment dokumentacji diod LED typu WS2812 pokazujący czasy impulsów sterujących

Tabela 1. Transmisja danych poprzez UART przekształcona na sygnał sterujący diodami typu WS2812

	Bit startu	7	6	5	4	3	2	1	Bit stopu
Transmisja z UART	0	A	1	0	B	1	0	C	1
Zanegowane wyjście UART	1	/A	0	1	/B	0	1	/C	0

Transmisję da się zrealizować na przerwaniach nawet bez wstawki w asemblerze, co pokazuje **listing 1**. Wynik działania programu obsługi przerwania został pokazany na **rysunku 9**. Czas obsługi przerwania wynosi 4,44 μs, więc na pewno nie będzie zinterpretowany przez diody jako reset. Po optymalizacji kodu, której wynik pokazano na **listingu 2** (niepotrzebne operacje zakomentowano), czas przerwania zmniejszył się do 4,08 μs. Łączna oszczędność 7 rozkazów/cykli ma dość duże znaczenie z racji bardzo częstego wywoływania przerwania. Efekt działania zoptymalizowanej procedury został pokazany na **rysunku 10**. Niebieski przebieg obrazuje działanie pętli głównej programu, która po skompilowaniu wykonuje się w czterech cyklach maszynowych (**listing 3**). Jak łatwo zauważyć, pomiędzy przerwaniami program główny wykonuje około 16 rozkazów. Dość mało i niewiele da się zrobić w takim czasie. Jeśli jednak weźmiemy pod uwagę to, że transmisja nie musi trwać bez przerwy i pomiędzy paczkami wstawimy pauzę o długości 20 ms, co da odświeżanie zawartości LED na poziomie 50 Hz, to przy 60 diodach LED otrzymamy: 60·1,2 μs·24 bity ok. 1,7 ms, czyli zajętość CPU na poziomie 8,5%. Przykładowe obliczenia obciążenia CPU dla innych parametrów pokazano w **tabeli 2**.



Rysunek 7. Fragment dokumentacji diod LED typu WS2812 pokazujący podstawowe przebiegi sterujące

Sygnał DMX

Sygnał przesyłany poprzez DMX jest konwertowany w układzie U4, natomiast układ U6 typu SC16IS760 odbiera dane. Można je z niego odczytać poprzez interfejs I²C lub SPI. W urządzeniu wybrano tryb SPI (nóżka 8 zwarta z masą), który jest dużo szybszy. Może pracować z szybkościa do 15 Mb/s (1,5 MB/s) i nie ma potrzeby adresowania. Układ SC16IS760 to interesujący kontroler UART, ma 64-bajtowy bufor FIFO do nadawania i odbioru, a na poziomie rejestrów jest kompatybilny z 16C55x/16C45x. Ponadto może automatycznie sterować/reagować na linie sprzętowego przesyłu danych RTS/CTS. Taką funkcję miały tylko dwa układy: intelowski 8251 i Z-80 SIO. Inną zaletą kontrolera

Listing 1. Kod realizujący transmisję za pomocą przerwania

```
SIGNAL(USART1_TX_vect){
    if (pozScr){
        UDR1 = *(scr+pozScr++);
        pozScr &=SCRLEN-1;
    }
    if (pozScr >= SCRLEN) FL_TransEnd=true;
}
```

Listing 2. Kod obsługi przerwania po optymalizacji

```
ISR(USART1_TX_vect, ISR_NAKED){
    asm volatile (
    // usunięta operacja na nieużywanym  R1 i R0
    //  "push   r1      \n\t"
    //  "push   r0      \n\t"
    //  "in r0, 0x3f    \n\t"   // SREG
    //  "push   r0      \n\t"
    //  "eor    r1, r1  \n\t"
        "push   r18     \n\t"
    //SREG na stos przy użyciu używanego później R18 (+3)
        "in r18, 0x3f   \n\t"   // SREG
        "push   r18     \n\t"
    //Dalej tak jak zrobił kopilator
        "push   r24     \n\t"
        "push   r25     \n\t"
        "push   r30     \n\t"
        "push   r31     \n\t"
    );
    //I dotychczasowy kod:
        if ( pozScr ){
            UDR1 = *(scr+pozScr++);
            pozScr &=SCRLEN-1;
        }
        if (pozScr >= SCRLEN) FL_TransEnd=true;
    asm volatile (
        "pop    r31     \n\t"
        "pop    r30     \n\t"
        "pop    r25     \n\t"
        "pop    r24     \n\t"
    //Dodane SREG przy użyciu R18
        "pop    r18     \n\t"
        "out    0x3f, r18\n\t"
    //Dalej normalnie R18
        "pop    r18     \n\t"
    //Usunięte operacje na R0 i R1
    //  "pop    r0      \n\t"
    //  "out    0x3f, r0\n\t"
    //  "pop    r0      \n\t"
    //  "pop    r1      \n\t"
    );
    reti();
}
```

Wykaz elementów:

Rezystory: (SMD1206, o ile nie zaznaczono inaczej)
R1, R21, R22, R23: 10 kΩ
R2, R3, R8, R10, R11, R17…R19: 1 kΩ
R4, R5, R20: 4,7 kΩ
R6: nie montować/1 kΩ 1%, opis w tekście
R7: 0 Ω/3,3 kΩ 1%, opis w tekście
R9, R13: 100 kΩ
R12, R14…R16: 120 Ω
R27…R29: 27 Ω
RP1: 8×0 Ω opis w tekście
RP2: 8×1 kΩ
ZW1: 0 Ω
P2, P3: joystick typu JV1603-B10K/SW11 + gałka

Kondensatory: (SMD1206, o ile nie zaznaczono inaczej)
C1, C2, C5, C6, C8, C9, C15, C16, C19, C22…C24: 100 nF
C3, C4: 22 pF
C10: 1000 μF/25 V (lub 63 V) opis w tekście
C11, C12, C13, C14: 220 μF/16 V
C17, C18: 22 pF opis w tekście
C20 C21: 47 pF
C23, C26: 10 μF/16 V

Półprzewodniki:
D1 (Run): dioda LED SMD1206 niebieska
D2 (DMX): dioda LED SMD1206 zielona
D3 (Power): dioda LED SMD1206 zielona
D4: SS24
D5 (DMXin): dioda LED SMD1206 zielona
D6 (DMXout): dioda LED SMD1206 żółta
D7 (Run): dioda LED SMD1206 niebieska
D8 (HOST): dioda LED SMD1206 niebieska
D9 (RXusb): dioda LED SMD1206 zielona
D10 (RXuart): dioda LED SMD1206 zielona
D12 (SC16SI): dioda LED SMD1206 żółta
D11 (ST3): dioda LED SMD1206 żółta
D13 (ST1): dioda LED SMD1206 czerwona
D14 (OVRdmx): dioda LED SMD1206 czerwona
D16 (WS): dioda LED SMD1206 zielona
M1: mostek prostowniczy S4VB
U1: ATMEGA1284 (TQFP44)
U2: 74HC00D (SO-14)
U3: LM2576T-5/-ADJG (TO-220V)
U4, U5: MAX485 (SO-08)
U6: SC16IS760IPW (SSOP-24)
U7: 74LVC125AD (SO-14)
U8: FT221XS (SSOP-20)
U9, U10: 4053 (SO-16)
U11: SPX1117MP-3.3 (SOT-223)

Pozostałe:
Q2: rezonator kwarcowy SMD 16 MHz
Q1: rezonator kwarcowy SMD 20 MHz
L1: 100 μH 3 A SMD DPT100A3
L2: 68 μH SMD1210 DL1210-68
TR1: TSZZ 20/005M (12 V, 1,67 A)
F1: bezpiecznik 630 mA + GN BEZP PTF
J5 (230V): ARK2
J7 (DMX IN): XRL3 wtyk
J9: (DMX OUT): XRL3 gniazdo
J13: (DMXin): goldpin 1×2
J11: (DMXbypass): goldpin 1×2
J8 (DMXout): goldpin 1×3
J2 (IIC): goldpin 1×4
J3 (ISP): T821-1-06-S1
J1 (JTAG): WS10
J14 (SCK): goldpin 1×3
JP4 (SEL): goldpin 2×2
JP3 (SPI): goldpin 1×2
J4 (Txm): ARK3
J10 (Term): goldpin 1×2
J12 (USBB-BV): gniazdo USB
JP2 (Vcc): goldpin 1×2

Listing 3. Kod testowej pętli głównej

```
loop:
    wdg();
    PORTA |= _BV(PA3);
    PORTA &= ~_BV(PA3);
```

jest automatyczna zmiana kierunku transmisji dla RS485/422. Układ może sprzętowo obsługiwać Xon/Xoff ,składające się z jednego lub dwu bajtów. Dostępne są jeden lub dwa interfejsy UART w jednej obudowie. Dodatkowe 8 (10) portów IO z możliwością generowania przerwań też jest zaletą. Często umożliwi to rezygnację z popularnego PCF8574, a pracuje w trybie FAST (400 kHz), a nie standardowym (100 kHz). Dodatkowe rejestry kontrolera (23 rejestry na 16 adresach, a nie 10 na 8, jak w przypadku 16C55x) umożliwiają ustawienie progu sygnalizacji zapełniania FIFO w zakresie 4...64 bajty z rozdzielczością 4 bajtów. Inne rejestry przechowują informacje o liczbie znaków w FIFO nadawczym i odbiorczym. W urządzeniu skorzystano z tej funkcji.

Po wykryciu pojawienia się znaku w kontrolerze UART wykonywany jest program z **listingu 4.** Podczas nadawania także używany jest bufor FIFO. Cała procedura nadawania jest bardziej skomplikowana i oparta na maszynie stanów.

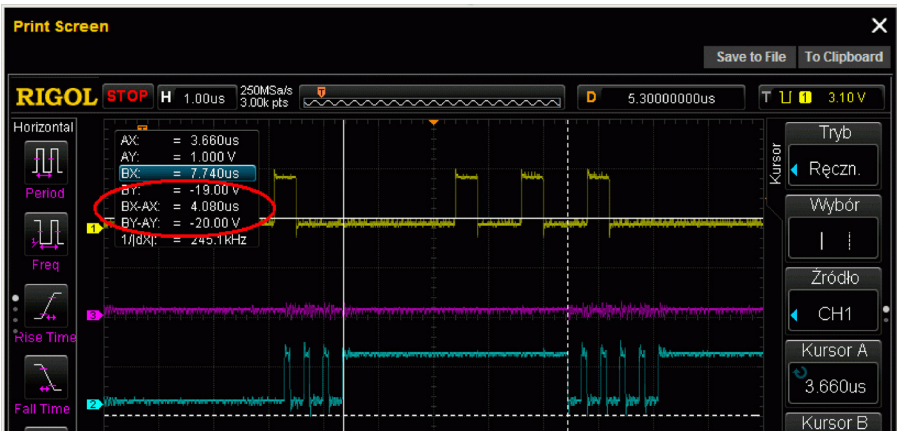
Kontroler UART wymaga jeszcze kilku słów wyjaśnienia. Aby zapewnić funkcjonalną kompatybilność z pierwowzorem i przypadkowo nie uaktywnić dodatkowych funkcji, wprowadzono specyficzny sposób dostępu do niektórych rejestrów. Aby dostać się do TCR i TLR, należy uaktywnić między innymi bit 4 w EFR. Bit w EFR będzie dostępny natomiast po wpisaniu do LCR wartości $BF. Wszystko jest opisane w nocie katalogowej, ale, aby ułatwić zmagania z układem, procedura inicjalizacji została pokazana na **listingu 5.** Funkcje obsługi układu sprowadzają się do:

```
byte Write_SC16IS(byte slave,
byte reg, byte data)
void WriteBlock_SC16IS(byte
slave, byte reg, byte *buf,
byte len)
byte Read_SC16IS(byte slave,
byte reg)
void ReadBlock_SC16IS(byte slave,
byte reg, byte *buf, byte len)
```

Dodam jeszcze, że zanim zrealizowałem projekt konwertera DMX, wykonałem próby dekodowania DMX z użyciem interfejsu UART mikrokontrolera podczas transmisji do WS2812. Ze względu na to, że niezalecane jest używanie innych przerwań blokowanych (*SIGNAL* lub *ISR* bez atrybutu *NOBLOCK*) poza tymi do obsługi diod LED, a takimi przerwaniami muszą być przerwania odbiorcze od UART, w przerwaniach nieblokowanych (*INTERRUPT* lub *ISR* z atrybutem *BLOCK*) od timera wywoływanymi

Rysunek 8. Zmierzona maksymalna przepływność interfejsu UART

Rysunek 9. Efekt działania transmisji z kodem bez optymalizacji



Rysunek 10. Efekt działania transmisji ze zoptymalizowanym kodem

Rysunek 11. Pomiary czasów realizacji funkcji w trakcie działania pętli *main*

Tabela 2. Przykładowe obliczenia obciążenia CPU

Liczba LED	Odświeżanie	Częstotliwość odświeżania	Czas transmisji	Zajętość CPU
400	20 ms	50 Hz	11,5 ms	57%
400	50 ms	20 Hz	11,5 ms	23,00%
60	50 ms	20 Hz	1,7 ms	3,40%

co 33 µs (co 3 przerwania 11 µs) sprawdzałem UART.

Test obciążenia CPU

Na **rysunku 11** zostały pokazane pomiary czasów realizacji funkcji w trakcie działania pętli *main*. Przebiegi obrazują następujące funkcje:

- żółty – linia danych WS2812,
- błękitny – przerwanie TX UART obsługujące SPI,
- fioletowy – linia UART DMX512 (bajt $FF),
- niebieski – pętla *main*.

Widać tam bardzo silne obciążenie CPU (dochodzą jeszcze przerwania od timera0, ADC, których nie pokazano na oscylogramie), w pętli głównej wykonuje się po 12 czasem tylko 2 rozkazy pomiędzy przerwaniami. Wydaje się, że taki program nie ma prawa działać, ale program główny „nadrabia" w dwu przypadkach:

- pomiędzy transmisjami do LED występuje przerwa 50 ms,
- sygnał DMX ma stosunkowo długą przerwę po wysłaniu ramki DMX.

Niestety w praktyce okazało się, że taki mechanizm czasami powoduje gubienie danych DMX. Problemu nie byłoby, gdyby UART mikrokontrolera miał np. 16, a nie 2 bajty FIFO. Na szczęście przewidziałem miejsce na zewnętrzny UART. Oczywiście możliwa jest próba dopracowania procedur obsługi DMX przez UART mikrokontrolera, ponieważ linia DMX jest do niego doprowadzona. W kodzie źródłowym znajdują się linie *#define DMX_IN* i *#define DMX_OUT* umożliwiające wyłączenie zewnętrznego UART.

Listing 4. Kod wykonywany po wykryciu znaku w kontrolerze UART

```
// Jeśli BREAK lub RX
while( (stat=Read_SC16IS(0, SC16IS_LSR)) & 0b1001){
    //==================== Obiór ======================//
    // Czy znak w odbiorniku ?
    if (stat & 1){
        // Jeśli błąd FIFO (w danych znajduje się BREAK)
        if (stat & 0x80){
        // Czytaj bajt po bajcie
            if(!errOv) DekodujDMX(Read_SC16IS(0, SC16IS_RHR));
            block = false;
        // Jeśli nie ma błędu
        } else {
            // Czytaj blokowo
            byte static buf[64];
            // Liczba znaków w FIFO
            byte rxlvl = Read_SC16IS(0, SC16IS_RXLVL);
            // Czytaj blok
            ReadBlock_SC16IS(0, SC16IS_RXLVL, &buf, rxlvl);
            if(!errOvr) for(byte x=0; x<rxlvl; x++) DekodujDMX(buf[x]);
        }
        // Jeśli BREAK
        if (stat & 0x10){
            errOvr = false;
            TimeOutUsbLoock = 1000;
            TimLedRxUart = TIM_FLASH_LED; SetLed(LED_RX_UART);
            DekodujDMX(0xFFFF);
        // Sygnalizacja błędu przepełnienia bufora
        } else if (stat & 2+8){
            TimLedOvrUart = TIM_FLASH_LED; SetLed(LED_OVR_DMX);
        }
    }
}
```

Listing 5. Procedura inicjalizacji kontrolera UART

```
//=====================================================================//
// Inicjalizuje SC16IS. Jeśli nie wykryje układu zwraca false
//=====================================================================//
byte Init_SC16IS(byte slave){
byte hi, lo;

    slave &= SC16IS_MAX-1;
    SPI_UART_SSh();
    // Speed=max, Master, MSB First, SpiMode=0
    SpiInit(0, true, true, 0);

#define BAUD_SC16IS_VAL      CLK_SC16IS / 1UL / ( BAUD_SC16IS * 16UL)
#define  BAUD_SC16IS_VAL_L       ( BAUD_SC16IS_VAL & 0xFF )
#define  BAUD_SC16IS_VAL_H       ( BAUD_SC16IS_VAL >> 8 )

    // Udostępnienie EFR
    Write_SC16IS(slave, SC16IS_LCR, 0xBF);
    // Udostępnienie funkcji rozszerzonych
    Write_SC16IS(slave, SC16IS_EFR, 0x10);

    // Wybór rejestru prędkości
    Write_SC16IS(slave, SC16IS_LCR, 0x80 );
    Write_SC16IS(slave, SC16IS_DLL, lo=BAUD_SC16IS_VAL_L);
    Write_SC16IS(slave, SC16IS_DLH, hi=BAUD_SC16IS_VAL_H);

    //----- Weryfikacja-----
    if (Read_SC16IS(slave, SC16IS_DLL) != lo) return(sc16is_st[slave]=false);
    if (Read_SC16IS(slave, SC16IS_DLH) != hi) return(sc16is_st[slave]=false);

    //B7-latch, B6-BREAK, SetParity, EvenParity, ParityEn, B2-Stop, D1,B0 - Lenght
    // Wybór formatu ramki 8N2
    Write_SC16IS(slave, SC16IS_LCR, 0b00000111);
    // Włączenie FIFO, RX gdy 16 bajtów
    Write_SC16IS(slave, SC16IS_FCR, 0b01000001);
    // Przerwania odbiorcze
    Write_SC16IS(slave, SC16IS_IER, 0b00000001);

    return (sc16is_st[slave]=true);
}
```

USB

Konwersję komunikacji USB zapewnia układ FT221. Jego zaletą jest szybka komunikacja po SPI oraz dostęp do pamięci konfiguracji. Możliwości układów były już opisywane na łamach EP (pierwszy odcinek kursu w EP 7/2017). Układ FT221 nie wymaga konwersji poziomów na liniach. Rezystory o wartości 0 Ω zastosowano w innym celu. Przy komunikacji z FT221 przez programowe SPI montujemy tylko rezystor 0 Ω pomiędzy pinami 8 i 9 drabinki RP1. Jeśli zaszłaby potrzeba komunikacji równoległej, należy zamontować wszystkie rezystory, a podczas komunikacji z układem odłączać klucze 4053 odczytujące stan joysticków (sygnał SEL), co z kolei spowoduje konieczność modyfikacji odczytu stanów joysticków i prawdopodobnie przeniesienie tej funkcjonalności z przerwań do pętli głównej.

Z układem można komunikować się także przez sprzętowe SPI. W tym celu używa się bramki trójstanowej realizującej funkcję bramki OC (U7C). Poza modyfikacją programu należy zamontować zworę JP3. Trzeba pamiętać, że SPI jest używane do komunikacji z kontrolerem UART, który może wykorzystywać mechanizm przerwań. Trzeba o tym pamiętać i zastosować mechanizm semaforów. Z tego właśnie powodu zdecydowano się na programowe SPI.

Dane DMX przychodzące przez RS485/422 mają większy priorytet niż DMX przez USB. Jeśli więc dane będą przychodzić po RS, informacja po USB będzie ignorowana.

Joysticki

Aby zademonstrować możliwości diod, płytkę wyposażono w dwa joysticki analogowe z przyciskiem. Można je odłączyć, ustawiając linie SEL w stanie wysokim, co zdezaktywuje klucze 4053. Jeśli opcja sterowania kluczami nie jest potrzebna, można zewrzeć piny 3–4 JP4, a port PD4

Tabela 3. Opis znaczenia zworek znajdujących się na płytce urządzenia (* – ustawienie domyślne)

Oznaczenie/ nazwa	Funkcja	Uwagi
JP2/Vcc	Zasilanie konwertera z USB	Rozwarty* – zasilanie zewnętrzne Zwarty – zasilanie z USB
JP3/SPI	Sprzętowe SPI dla FT221	Rozwarty* – SPI programowe Zwarty – SPI sprzętowe
J13/SCK	Interfejs SPI dla FT221	1-2* – programowe SPI dla FT221 2-3 – sprzętowe SPI dla FT221
JP4/SEL	Przyłączenie joysticków do wejścia multipleksera analogowego mikrokontrolera	1-2* – sterowanie portem PD4 2-3 – joysticki zawsze podłączone do ADC
J8/DmxOut	Wybór nadajnika DMX	1-2* – zewnętrzny UART 2-3 – UART mikrokontrolera
J10/Term	Terminowanie linii	Rozwarty* – terminator wyłączony Zwarty – terminator włączony
J11/DMXbypass	Przesłanie sygnału odbieranego z DMX bezpośrednio do nadajnika	Rozwarty* – sygnał nie jest retransmitowany Zwarty – sygnał jest retransmitowany
J13/DMXin	Odłączenie odbiornika DMX od mikrokontrolera	Rozwarty* – odbiornik odłączony, PD0 nieużywany Zwarty – odbiornik podłączony do mikrokontrolera

wykorzystać w innym celu. Przy braku transmisji DMX uruchamia się jedna z wybranych gier. Opcje tę można, wyłączyć konfigurując urządzenie przez USB.

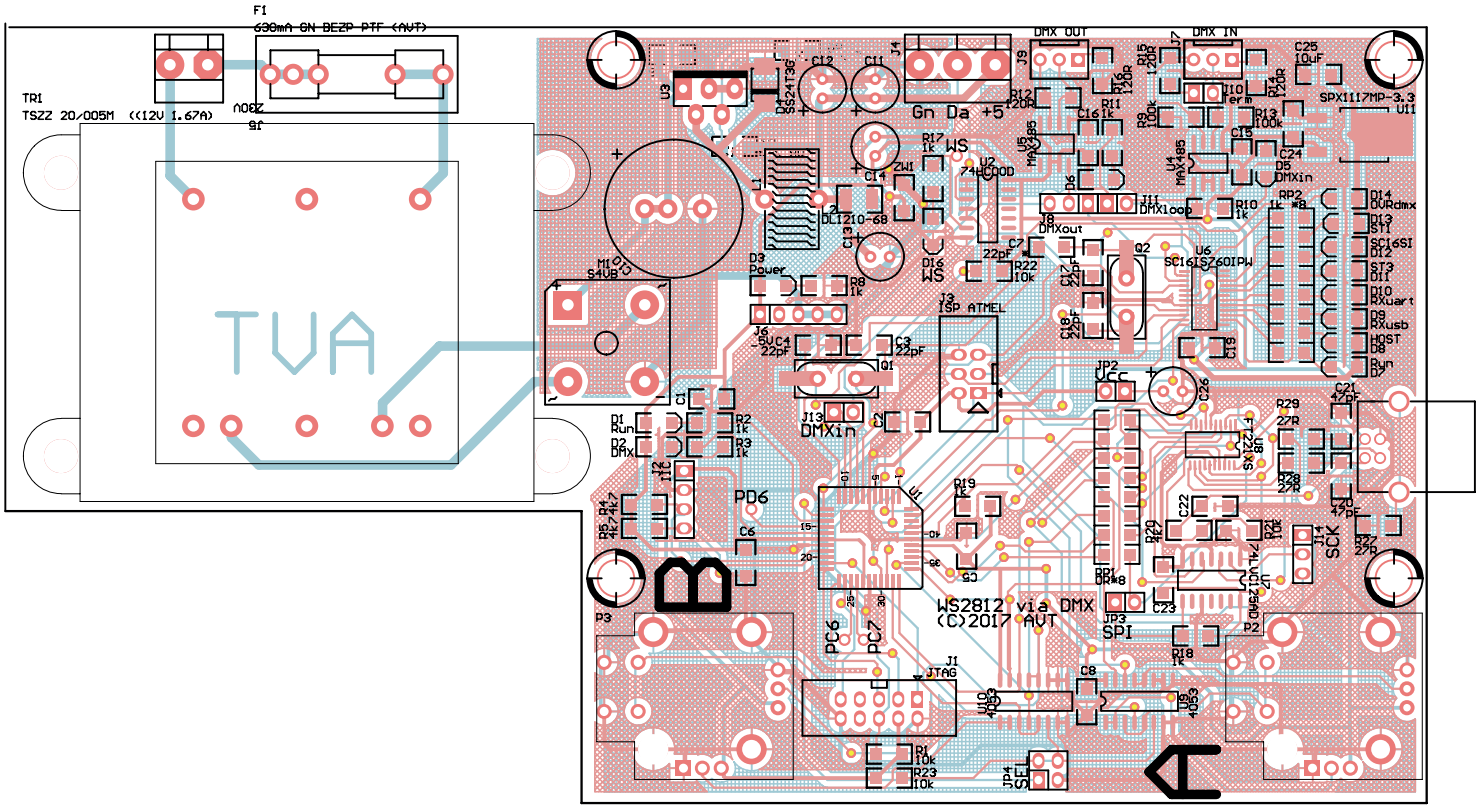
Praca jako konwerter USB-DMX

Urządzenie może pracować jako konwerter USB-DMX. Zdekodowany sygnał, poza sterowaniem diodami WS2812, dostępny jest na złączu J9. Niestety FT221 nie potrafi wykryć sygnału BREAK synchronizującego początek ramki DMX. Z tego powodu realizowane jest to programowo na zasadzie wykrycia długiego sygnału MARK. W konsekwencji praca w tym trybie nie jest gwarantowana z każdym programem. Prototyp bez większych problemów działał z DMX512 Light Control, ale współpraca z USB2DMX512DEMO napisanym w Delphi była niemożliwa. Dane na wyjście DMX są transmitowane zarówno w trybie konwerter DMX (odbiór danych z USB), jak i gdy są odbierane z RS485/422.

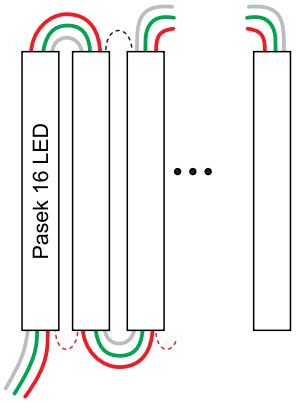
Montaż i uruchomienie

Schemat płytki drukowanej został pokazany na **rysunku 12**. Montaż jest typowy i nie wymaga szczegółowego omawiania. Zwrócę tylko uwagę na kilka istotnych szczegółów. Uruchomienie urządzenia najlepiej rozpocząć od zasilacza. Jeśli użyjemy LM2576T-5, nie montujemy R6, a R7 zastępujemy zworą. Gdy użyjemy LM2576T-ADJG, wartości rezystorów są następujące: R6=1 kΩ, R7=3,3 kΩ. Wskazane jest użycie rezystorów 1%. Gdy wydajność zasilacza jest mała (przy obciążeniu prądem 1,5...2 A napięcie spada do 4 V lub mniej), winny jest dławik (wchodzi w nasycenie) lub kondensatory C11, C14 mają dużą wartość ESR. Zła jakość C9 też może powodować takie efekty. W takiej sytuacji pomaga dołączenie kondensatora 100 µF jak najbliżej wyprowadzeń 1 i 3 układu U3.

Nie należy ignorować roli rezystora R1 podciągającego linie reset mikrokontrolera do zasilania. Ponieważ linia ta jest podłączona także do kontrolera UART, jego brak powodować będzie przypadkowe resety.



Rysunek 12. Schemat płytki PCB wraz z rozmieszczeniem elementów



Rysunek 13. Sposób połączenia pasków diodowych dla zademonstrowania możliwości urządzenia

Rezystor R22 można pominąć, jego rola sprowadza się do ustalenia stabilnego poziomu niskiego na wejściu danych diod WS2812 w czasie wgrywania programu do mikrokontrolera. Jego brak powodował przypadkowe zaświecanie się diod w czasie transmisji danych do układu. Podczas pisania oprogramowania bywało, że wszystkie zaświecały się na biało, co powodowało duży pobór prądu, na tyle duży, że przy 264 diodach (dwa paski 60 LED + jeden 144 LED) włączało się ograniczenie prądowe. To powodowało, że mikrokontoroler wychodził z trybu *debug*.

W interfejsie USB montujemy tylko rezystor 0 Ω. Zwora JP2 umożliwia zasilanie układu z USB. Jeśli także diody LED mają być zasilane z tego źródła, należy być świadomym ograniczenia prądowego wnoszonego przez USB.

Moduł UART jest taktowany kwarcem przyłączonym do niego. Możliwe jest taktowanie zegarem CPU. W takiej sytuacji należy usunąć kwarc Q2 oraz elementy C17 i C18. Konieczne jest wlutowanie C7 oraz zmiana w kodzie źródłowym (zmiana *#define CLK_SC16IS* na *20000000*). Takie użycie sygnału zegarowego CPU nie jest zalecane. Wskazane jest ustawienie bitu konfiguracyjnego CKOUT i pobranie sygnału z wyprowadzenia CPU udostępniającego taki sygnał. Dla M1284 jest to PB1 użyty w roli programowego portu SPI. Dla lepszej przejrzystości znaczenie wszystkich zworek konfiguracyjnych znajdujących się na PCB zestawiono w **tabeli 3**, natomiast w **tabeli 4** zostały opisane funkcje diod sygnalizacyjnych.

Jeśli chcemy zobaczyć demonstracje możliwości sterownika w postaci gier, należy zmontować ekran według **rysunku 13.** Liniami przerywanymi oznaczono mostki, które zaleca się wykonać. Zmniejszają one spadek napięcia na ścieżkach doprowadzających zasilanie.

Aby zwiększyć kontrast wyświetlacza, należy zastosować filtr. W przypadku wyświetlacza o jednym kolorze sprawa jest prosta, natomiast w przypadku wielobarwnego nie można zastosować filtru jednokolorowego. Filtr powinien być szary lub brązowy. Zakup odpowiedniego filtra nie jest łatwy, ale w tej roli doskonale sprawdza się folia do przyciemniania szyb samochodowych, którą należy nakleić na kawałek szyby lub pleksi.

Tabela 4. Opis znaczenia diod LED

Oznaczenie/nazwa	Funkcja/uwagi
D3/Power	Sygnalizuje zasilanie
D1/Run	Pulsuje w czasie pracy mikrokontrolera
D2/DMX	Świeci, jeśli wykryto sygnał DMX z RS485/422 lub USB
D5/DMXin	Fizyczny stan linii odbiorczej DMX
D6/DMXout	Fizyczny stan linii nadawczej DMX
D7/Run	Miga w czasie pracy mikrokontrolera
D8/Host	Świeci, jeśli wykryto sygnał DMX z RS485/422 lub USB
D9/RXusb	Rozbłyska na 100 ms po odebraniu bajtu z USB
D10/TXuart	Rozbłyska na 100 ms po odebraniu bajtu z RS485/422
D11/ST3	Status/nieużywane
D12/SC16SI	Poprawne zainicjowanie UART
D13/ST1	Status/nieużywane
D14/OVRdmx	Przepełnienie bufora odbiorczego UART
D16/WS	Świeci w czasie przesyłania „1” do diod WS2812

Komendy sterujące

Komendy są interpretowane, gdy nie trwa transmisja DMX po RS485/422 lub USB. Transmisja jest sygnalizowana cykliczną zmianą stanu sygnału CD (*Carier Detect*) w terminalu. Komendy można wydawać z terminalu przez USB (wirtualny port COM dla FT221). Parametry transmisji (prędkość, format ramki) nie są istotne. Komendę rozpoczyna znak „:” kończy CR lub CR+LF. Każda komenda jest potwierdzana komunikatem na ekranie (najczęściej wyświetlenie wprowadzonych parametrów). W **tabeli 5** znajduje się spis komend (komendy wprowadzamy bez spacji, dodano je dla lepszej czytelności).

Możliwość zmian

W urządzeniu nie przewidziano możliwości wyboru adresu bazowego, który aktualnie jest ustawiony na pierwszy kanał DMX. Jeśli zajdzie taka potrzeba, można zmodyfikować kody źródłowe celem wprowadzenia takiej funkcjonalności. Istnieje także możliwość udostępnienia konfigurowania urządzenia przez RS485/422. Niestety w takiej sytuacji nie ma informacji zwrotnej, ponieważ DMX512 w podstawowej wersji jest jednokierunkowy.

Można zastanowić się nad konfigurowaniem niektórych parametrów, takich jak timeout rozpoznający koniec ramki DMX przy dekodowaniu jej przez USB. Podobnie jest z reakcją na nadawanie DMX w przypadku zaniku sygnału z komputera. Tych funkcji nie wprowadzono, uznając je za mało przydatne, a przedłużyłyby znacznie prace nad projektem. W przypadku dużego zainteresowania takimi funkcjonalnościami autor wprowadzi stosowne modyfikacje. Proszę też pisać o innych pomysłach związanych z konwerterem czy zastosowaniem diod WS2812.

W materiałach dodatkowych, poza kodami źródłowymi, znajdują się oscylogramy i logi z SaleAE robione na różnych etapach pracy urządzenia.

Sas
sas@elportal.pl

Tabela 5.

Komenda	Opis
: h	Wyświetla pomoc, przykład: Help: :R – Restart :c – Colors [4, 8, 24-bit] :g – Demo games [0, 1] :o – Off-line: [b]lank, [m]emory, [g]ames
: ?	Wyświetla status, np.: ?: c=24-bit colors o=Game g=0 Tenis gdzie: c – tryb wyświetlania barw (24-, 8- lub 4-bitowy) o – zachowanie w trybie off-line g – tryb gry w trybie off-line
: R	Wywołuje restart konwertera
: c x	Liczba bitów palety barw, gdzie x: 24 – 24-bity 4 – 4-bity, 16-barw 8 – 8-bitów, 216 „bezpiecznych” barw *
: o x	Zachowanie w trybie off-line, gdzie x: b – wygaszenie wyświetlacza** m – wyświetlenie ostatnio odebranej ramki DMX** g – wyświetlenie gry ustawionej komendą g
: g x	Wybór gry w trybie off-line, gdzie x: 0 – Tenis 1 – Galaga

* Paleta bezpiecznych barw miała na celu ujednolicenie kolorów na różnych komputerach
** Ustawienie tego trybu wyłącza demo w trybie off-line oraz ekran powitalny