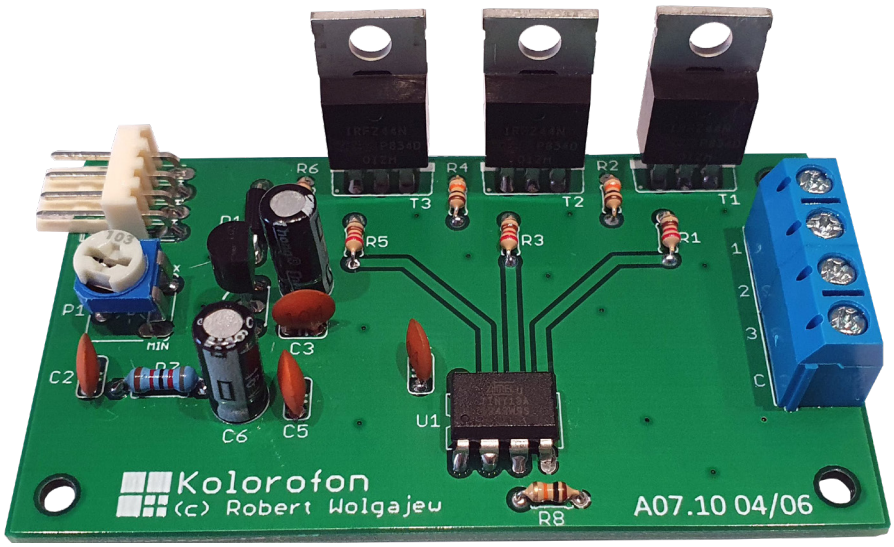
# Kolorofon

*Tym razem wracam do lat 80., gdy powszechnie pożądanym zestawom wieżowym (audio) towarzyszył sprzęt dostarczający efektów wizualnych, nazywany kolorofonem. Był to 3-kanałowy sterownik oświetlenia, którego każdy z kanałów sterowany był sygnałem audio z innego zakresu częstotliwości, dając ciekawy efekt wizualny pogłębiający doznania dźwiękowe. Oczywiście, jak na ten czas przystało, było to urządzenie w pełni analogowe. Zapatrzony w ówczesne czasy i zaopatrzony we współczesną wiedzę, postanowiłem skonstruować podobne urządzenie, lecz tym razem z użyciem techniki mikroprocesorowej.*




Zadaniem urządzenia miała być realizacja funkcji kolorofonu z cyfrowym torem przetwarzania sygnałów audio, bez potrzeby stosowania analogowych filtrów wejściowych. Co więcej, chciałem, żeby docelowa implementacja bazowała na bardzo prostym mikrokontrolerze, o ograniczonych zasobach sprzętowych. W ten sposób postanowiłem udowodnić, że nawet skomplikowane, na pozór, zadania programowe można zrealizować przy użyciu minimalnych zasobów sprzętowych.

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

**W ofercie AVT* AVT-5833**

**Podstawowe parametry:**
- częstotliwości środkowe filtrów kanałów LED: 400 Hz, 2000 Hz, 4000 Hz,
- maksymalny prąd gniazd wyjściowych LED1…3: ok. 10 A,
- napięcie zasilania: 7…12 V,
- pobór prądu samego urządzenia: ok. 10 mA.

**Projekty pokrewne na www.media.avt.pl:**

| | |
|---|---|
| AVT-5662 | Kolorofon z Wi-Fi (EP 2/2019) |
| AVT-5530 | Regulator natężenia oświetlenia z Wi-Fi (EdW 10/2017) |
| AVT-1853 | Iluminofonia LED RGB (EP 5/2015) |
| AVT-2980 | Kolorofon – sterownik RGB (EdW 6/2011) |
| AVT-2742 | Uniwersalna iluminofonia – kolorofon (EdW 2/2005) |

*** 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.

## Budowa i działanie

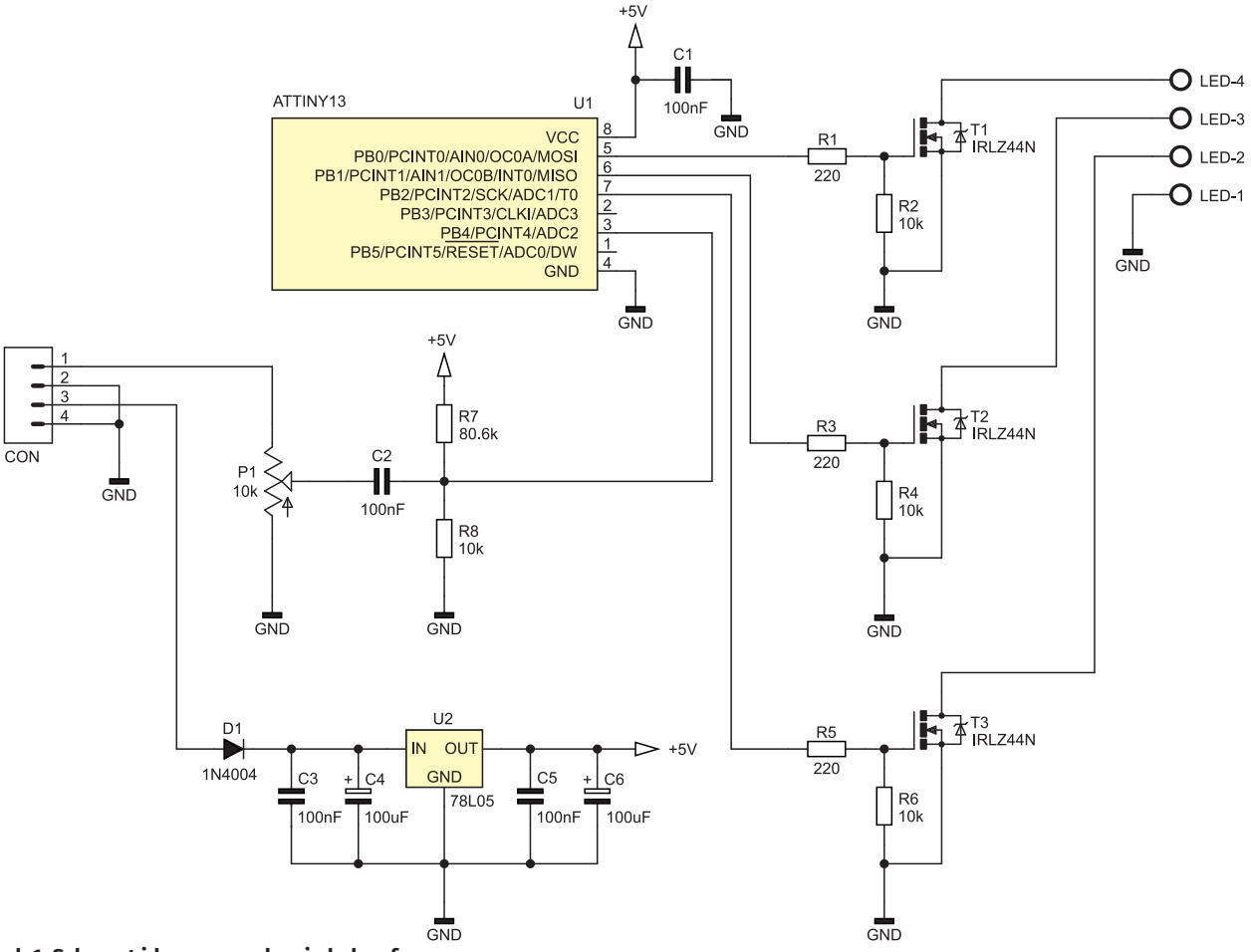
Zaprojektowano bardzo prosty system mikroprocesorowy, którego schemat ideowy został pokazany na **rysunku 1**. Jego sercem jest jeden z najmniejszych mikrokontrolerów firmy Microchip (dawniej Atmel) o oznaczeniu ATtiny13, taktowany wewnętrznym, wysokostabilnym generatorem RC o częstotliwości 9,6 MHz. Jest on odpowiedzialny za realizację pełnej funkcjonalności urządzenia. Próbkuje wejściowy sygnał audio doprowadzony do wyprowadzenia ADC2 (PB4) układu, z częstotliwością 8 kHz, używając przy tym układu czasowo-licznikowego Timer0, który jest w tym wypadku wyzwalaczem konwersji wbudowanego przetwornika ADC pracującego w trybie 8-bitowym.

Następnie, po zebraniu N liczby próbek (20), wykonuje zoptymalizowany algorytm dyskretnej transformaty Fouriera korzystający ze współczynników wektora rotującego (*twiddle factors*), aby wyznaczyć moce sygnału dla trzech zakresów częstotliwościowych: 400 Hz, 2000 Hz i 4000 Hz. Wyznaczone wartości są z kolei przeliczane (zakres 0…127) na współczynnik wypełnienia dla trzech wyjściowych kanałów PWM, za pomocą których urządzenie steruje pracą diod powerLED (z wykorzystaniem wbudowanych tranzystorów MOSFET) podłączonych do złącza LED.

Zastosowano tutaj programowy, 3-kanałowy, 7-bitowy generator PWM, bazujący na tym samym układzie czasowo-licznikowym Timer0 i jego przerwaniu od porównania zawartości licznika z wartością rejestru OCR0A (TIM0_COMPA_vect). Takie rozwiązanie było konieczne, ponieważ mikrokontroler ATtiny13 dysponuje wyłącznie dwoma kanałami sprzętowego generatora PWM, co jest w tym wypadku niewystarczające.

Uważny Czytelnik zastanowi się również nad zagadnieniem, w jaki sposób tak prosty mikrokontroler o pamięci programu wielkości 1 kB i 64-bajtowej pamięci RAM jest w stanie zrealizować skomplikowane obliczenia dyskretnej transformaty Fourier'a DFT. Zastosowałem tutaj wiedzę zawartą w interesującym artykule autorstwa Łukasza Podkalickiego (EP 12/2019) prezentującym arcyciekawe zagadnienia z zakresu DSP w odniesieniu do transformaty Fouriera w ujęciu realizacji na prostych, 8-bitowych mikrokontrolerach o ograniczonej mocy obliczeniowej i niewielkiej ilości pamięci RAM. Autor opisuje tam, jak przy użyciu arytmetyki stałoprzecinkowej i współczynników wektora rotującego w prosty sposób jesteśmy w stanie wykonać dyskretną transformatę Fouriera (DFT) sygnału audio, otrzymując widmo jego mocy. Na pierwszy rzut oka wydaje się to niewiarygodne, ale naprawdę działa. Oczywiście stosuję tutaj pewną sztuczkę polegającą na tym, że moc sygnału obliczana jest wyłącznie dla wybranych prążków częstotliwościowych, ale w niczym nie umniejsza to możliwości liczenia transformaty Fouriera na tak niewielkim mikrokontrolerze AVR.

Wróćmy jeszcze na chwilę do głównego zadania naszego, układu jakim jest akwizycja i przetwarzanie danych próbek sygnału audio. Proces ten możliwy jest do wykonania dzięki sprzęgnięciu ze sobą kilku podsystemów mikrokontrolera AVR, co pokazano na **rysunku 2**. Jak widać, podstawowym podsystemem mikrokontrolera



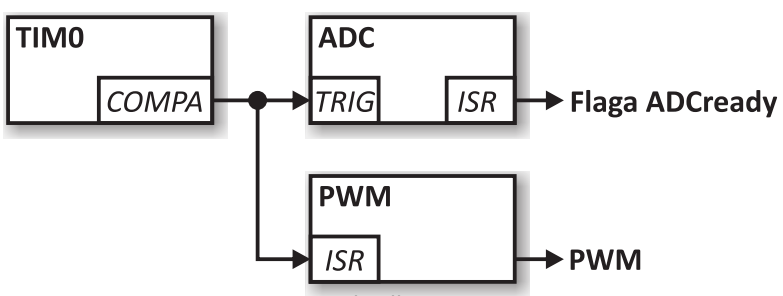

**Rysunek 1. Schemat ideowy urządzenia kolorofon**

ATtiny13 inicjującym pomiar przetwornika ADC jest układ czasowo-licznikowy Timer0 taktowany przebiegiem zegarowym o częstotliwości 1,2 MHz (otrzymanym poprzez podzielenie przebiegu zegarowego mikrokontrolera o częstotliwości 9,6 MHz przez preskaler równy 8) generuje zdarzenie porównania licznika (COMPA) co 150 taktów zegara (OCR0A=149), czyli 8000 razy na sekundę. Zdarzenie porównania jest wyzwalaczem (TRIG) dla wbudowanego w strukturę mikrokontrolera przetwornika ADC, dzięki czemu możliwe jest próbkowanie wejściowego sygnału audio w równych odstępach czasu (8 kHz).

Dalej przerwanie od zakończenia konwersji przetwornika ADC (ADC_vect) odpowiedzialne jest za właściwą akwizycję danych oraz ustawianie stosownych flag dla programu obsługi aplikacji. Co więcej, wspomniane wcześniej zdarzenie porównania stanowi również wyzwalacz dla procedury obsługi przerwania od porównania, które jest odpowiedzialne za programową obsługę 3 kanałów PWM. W ten prosty sposób Timer0 steruje zarówno akwizycją danych przetwornika ADC, jak i stanem pracy wyjściowych kanałów PWM.

```
Listing 1. Kod odpowiedzialny za inicjalizację przetwornika ADC

void ADCinit(void) {
    //Internal Voltage Reference (1.1V), ADC Left Adjust Result (8-bit), MUX = ADC2 (PB4)
    ADMUX = (1<<REFS0)|(1<<ADLAR)|(1<<MUX1);
    //ADC Enable, ADC Auto Trigger Enable, ADC Interrupt Enable , Division Factor = 64 (150kHz)
    ADCSRA = (1<<ADEN)|(1<<ADATE)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1);
    //ADC Auto Trigger Source: Timer/Counter Compare Match A
    ADCSRB = (1<<ADTS1)|(1<<ADTS0);
    //Digital Input Disable on ADC2 (PB4) pin
    DIDR0 = (1<<ADC2D);
}
```




**Rysunek 2. Schemat blokowy systemu akwizycji danych**

## Program sterujący

Kod odpowiedzialny za inicjalizację przetwornika ADC pokazano na **listingu 1**, zaś kod odpowiedzialny za inicjalizację układu czasowo-licznikowego Timer0 pokazano na **listingu 2**. Dalej, na **listingu 3** pokazano kod obsługi przerwania odpowiedzialny za realizację 3-kanałowego, 7-bitowego, programowego generatora PWM.

Na **listingu 4** pokazano kod obsługi przerwania po konwersji przetwornika ADC odpowiedzialny za akwizycję danych wejściowych ADC. Zebraniu kompletnej porcji danych towarzyszy zatrzymanie akwizycji danych oraz ustawienie flagi *ADCdataReady*, dzięki czemu możliwe jest przetworzenie danych przez program główny aplikacji. Za przetworzenie danych, czyli obliczenie dyskretnej transformaty Fouriera z próbek sygnału zebranych w tablicy *ADCbuffer[]*, odpowiedzialna jest funkcja pokazana na **listingu 5**. Wynikiem działania funkcji *DFT()* jest moc szukanego prążka częstotliwości

Listing 2. Kod odpowiedzialny za inicjalizację układu czasowo-licznikowego Timer0

```
void PWMinit(void) {
    PWM_DDR = 0b111;        //PWM port as output
    TCCR0A = (1<<WGM01);    //Timer0 CTC mode
    OCR0A = 149;            //ISR TIM0_COMPA_vect triggered 8000 times per second
    TCCR0B = (1<<CS01);     //Prescaler = 8
    TIMSK0 = (1<<OCIE0A);   //Timer/Counter0 Output Compare Match A Interrupt Enable
}
```

Listing 3. Kod odpowiedzialny za obsługę programowego generatora PWM

```
ISR(TIM0_COMPA_vect) {
    static uint8_t Counter;
    Counter = (Counter +1) & 0x7F;  //7 bits
    for(uint8_t i=0; i<3; ++i){
        if(Counter < PWM[i]) PWM_PORT |= (1<< i);
        else PWM_PORT &= ~(1<< i);
    }
}
```

**Ustawienia Fuse-bitów:**
SUT1: 1
SUT0: 0
CKSEL1: 1
CKSEL0: 0
CKDIV8: 1

Listing 4. Kod obsługi przerwania po konwersji przetwornika ADC odpowiedzialny za akwizycję danych wejściowych

```
ISR(ADC_vect){
    static uint8_t Idx;
    if(ADCdataReady == 0) {
        ADCbuffer[Idx++] = ADCH - 128;
        if(Idx == N) {
            Idx = 0;
            ADCdataReady = 1;
        }
    }
}
```

(w zakresie 0…127 dostosowanym do rozdzielczości kanałów PWM).

Właśnie to zadanie stanowi główny problem obliczeniowy, o którym wspomniano na wstępie artykułu. Wynika to z liczby obliczeń stałoprzecinkowych wykonywanych w ramach pętli, z których składa się wspomniana funkcja. Jak widać, liczba tych obliczeń zależy bezpośrednio od liczby punktów transformaty Fouriera (N), która w naszym przypadku wynosi 20. Z kolei liczba punktów transformaty determinuje odległość kolejnych prążków mocy (tzw. BIN) a wynika z zależności:

BIN = częstotliwość próbkowania sygnału/liczba punktów transformaty (N).

Dla naszego przypadku BIN=400 Hz (8 kHz/20), co oznacza, że kolejne wartości częstotliwości, dla których liczona jest moc sygnału, są wielokrotnością wartości 400 Hz. Wartość ta jest kompromisem pomiędzy rozdzielczością mocy (BIN) a czasem niezbędnym na wykonanie funkcji *DFT()* przy przyjętej liczbie punktów transformaty (N). W naszym przypadku czas ten wynosi około 5,4 ms (zmierzony empirycznie), co determinuje częstotliwość odświeżania (*framerate*), która w tym przypadku wynosi około 185 Hz, a więc doskonale, jak na nasz niewielki mikrokontroler. Dalsze zwiększanie liczby punktów transformaty (N), choć pożądane, zmniejszyłoby częstotliwość odświeżania kanałów PWM do wartości nieakceptowalnych i praktycznie nieużytecznych. To jest główne ograniczenie software'owe naszej implementacji, o którym wspomniano wcześniej i wynika w głównej mierze z 8-bitowej architektury mikrokontrolera AVR i maksymalnej, dostępnej częstotliwości taktowania rdzenia.

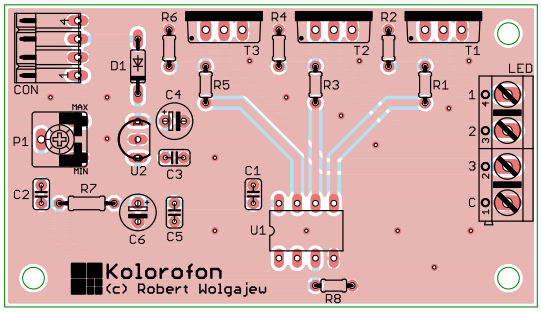
Warto również podkreślić, że przed wykonaniem funkcji *DFT()* (w pętli głównej programu aplikacji) zebrana tablica danych wejściowych *ADCbuffer[]* poddawana jest okienkowaniu, które ma na celu ograniczenie tak zwanych wycieków widma sygnału. Zastosowana funkcja okna jest typu Hanna (Hanninga), zaś stosowne współczynniki zebrano w tabeli *Window[]* umieszczonej w pamięci Flash mikrokontrolera, której treść pokazano na **listingu 6**.

## Montaż i uruchomienie

Schemat płytki PCB wraz z rozmieszczeniem elementów urządzenia kolorofon pokazano na **rysunku 3**. Zaprojektowano niewielką, dwustronną płytkę drukowaną z zastosowaniem wyłacznie elementów przewlekanych, co w założeniu miało uprościć implementację urządzenia. Montaż rozpoczynamy od przylutowania wszystkich rezystorów oraz diody półprzewodnikowej D1. Dalej lutujemy wszystkie kondensatory ceramiczne, następnie kondensatory elektrolityczne, po czym przystępujemy do przylutowania wszystkich elementów półprzewodnikowych. Na samym końcu montujemy potencjometr montażowy P1 oraz złącza przyłączeniowe CON i LED.

Warto zauważyć, że tranzystory sterujące T1…T3 mogą wymagać zastosowania niewielkich radiatorów, a sama potrzeba wynika z mocy, jaką te elementy przełączają. Z kolei potencjometr P1 służy do regulacji czułości układy wejściowego. Zmontowaną płytkę urządzenia pokazuje fotografia tytułowa. Na koniec warto wspomnieć o tym, że zamiast diod power-LED możemy również zastosować żarówki na napięcie 12 V.

**Robert Wołgajew, EP**




Rysunek 3. Schemat montażowy kolorofonu

**Wykaz elementów:**

**Rezystory:**
R1, R3, R5: 220 Ω (miniaturowy)
R2, R4, R6, R8: 10 kΩ (miniaturowy)
R7: 80,6 kΩ 0,1%

**Kondensatory:**
C1…C3, C5: 100 nF ceramiczny
C4, C6: 100 µF/16 V

**Półprzewodniki:**
U1: ATtiny13 (DIP08)
U2: 78L05 (TO92)
T1…T3: IRLZ44N (TO220)
D1: 1N4007 (DO41)

**Pozostałe:**
P1: potencjometr montażowy leżący 10 kΩ
LED: złącze śrubowe AK500/4
CON: gniazdo męskie 4 pin (NSL25-4W)

Listing 5. Kod funkcji odpowiedzialnej za obliczenie dyskretnej transformaty Fouriera

```
#define MULF 64  //Multiplication factor, max 128

//Tablica współczynników Twiddle Factors: Twiddle[i] = (int16_t) (MULF*cos(i*PI2/N)), dla i<N
const int16_t Twiddle[N] PROGMEM = {64, 61, 52, 38, 20, 0, -20, -38, -52, -61, -64, -61, -52,
-38, -20, 0, 20, 38, 52, 61};

uint8_t DFT(uint8_t i){
    uint16_t a, b;
    int32_t Re, Im;
    //Obliczamy moc sygnału (0...127) dla szukanego prążka częstotliwościowego i < (N/2)+1
    Re = Im = a = 0;
    b = 3*N/4;

    for (uint8_t j=0; j<N; ++j){
        Re += (ADCbuffer[j] * (int8_t) pgm_read_byte(&Twiddle[a % N]))/MULF;
        Im -= (ADCbuffer[j] * (int8_t) pgm_read_byte(&Twiddle[b % N]))/MULF;
        a += i;
        b += i;
    }
    return (Re*Re + Im*Im)/16384;
}
```

Listing 6. Tablica współczynników okna Hanna (Hanninga)

```
//Tablica współczynników okna Hanninga: Window[i] = (int8_t) (MULF*(0.5-0.5*cos(i*PI2/(N-1)))), dla i<N
const int8_t Window[N] PROGMEM = {0, 2, 7, 14, 24, 35, 45, 54, 60, 64, 64, 60, 54, 45, 35, 24, 14, 7, 2, 0};
```