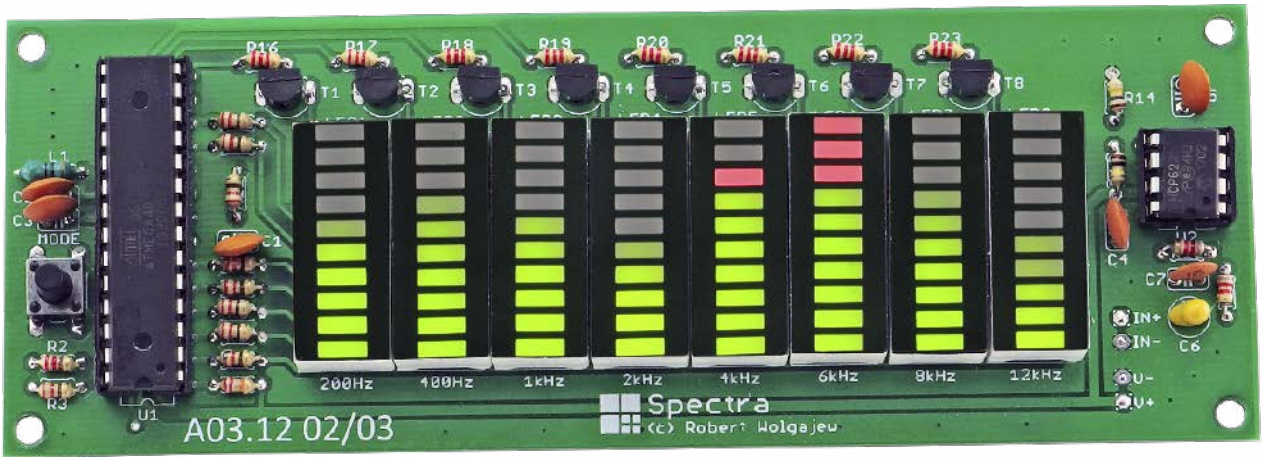


Spectra – analizator widma sygnału audio

Matematyka jest królową wszystkich nauk. Z tym cytatem spotkał się prawie każdy i nie sposób się z nim nie zgodzić, głównie wtedy, gdy zajmujemy się szeroko rozumianą elektroniką. Również i ja przekonałem się o tym wielokrotnie, zwłaszcza od czasu, gdy zacząłem zajmować się zagadnieniami z zakresu DSP, a zacząłem się nimi zajmować po natrafieniu na bardzo interesujący artykuł autorstwa Łukasza Podkalickiego (EP 12/2019) prezentujący arcyciekawe zagadnienia z zakresu DSP w odniesieniu do transformacji Fouriera w ujęciu realizacji na prostych, 8-bitowych mikrokontrolerach o ograniczonej mocy obliczeniowej i niewielkiej ilości pamięci RAM.

We wspomnianym artykule autor pokazuje, jak przy użyciu arytmetyki stałoprzecinkowej i współczynników wektora rotującego (tzw. *twiddle factors*) w prosty sposób jesteśmy w stanie wykonać dyskretną transformatę Fouriera (DFT) sygnału audio, otrzymując widmo jego mocy. Jakby tego było mało, autor prezentuje również praktyczną realizację 3-punktowego „kolorofonu" bazującego na niewielkim mikrokontrolerze firmy Atmel typu ATtiny13 i programowej realizacji transformaty DFT.

Posiłkując się tym unikalnym materiałem, postanowiłem odświeżyć swój wcześniejszy projekt analizatora widma sygnału akustycznego pod nazwą „Spectrum" (EP 9/2019), który cieszył się sporym zainteresowaniem wśród Czytelników, lecz tym razem z zastosowaniem wspomnianej wcześniej techniki DSP. Warto przypomnieć, że w budowie urządzenia zastosowałem bardzo ciekawy układ analizatora widma pod postacią układu scalonego MSGEQ7. Integruje on 7 filtrów pasmowo-przepustowych o częstotliwościach środkowych 63 Hz, 160 Hz, 400 Hz, 1 kHz, 2,5 kHz, 6,25 kHz oraz 16 kHz, 7 detektorów wartości szczytowej i analogowy multiplekser wyjściowy. Okazało się jednak, że układ MSGEQ7 jest peryferium trudno dostępnym, a nawet jeśli uda się go pozyskać (głównie za sprawą azjatyckiego portalu aukcyjnego), to nazbyt często okazuje się, że tak pozyskane peryferia nie są do końca sprawne lub, co gorsza, są to niedziałające podróbki.

Biorąc to wszystko pod uwagę, postanowiłem zbudować urządzenie o zbliżonej funkcjonalności, lecz pozbawione wcześniejszych rozwiązań sprzętowych, a realizujące całe skomplikowane zagadnienie filtracji sygnałów na drodze programowej. Co więcej, chciałem, by w odróżnieniu od pierwowzoru nowo projektowane urządzenie zbudowane było wyłącznie z popularnych i łatwo dostępnych elementów przeznaczonych do montażu przewlekanego, co w zamyśle miało ułatwić montaż układu nawet przez mało doświadczonych elektroników. Jako pewne rozszerzenie funkcjonalności urządzenia źródłowego w niniejszym projekcie dodałem jeszcze jedną linijkę LED, przez co zwiększyłem liczbę analizowanych pasm częstotliwościowych do ośmiu. Tak oto narodził się projekt Spectra.

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

W ofercie AVT* AVT5866

Podstawowe parametry:
- zasilanie: 5 VDC, ok. 170 mA,
- maksymalna amplituda analogowego sygnału wejściowego: 2,5 V (5 Vpp),
- analizowane pasma częstotliwościowe: 200 Hz, 400 Hz, 1 kHz, 2 kHz, 4 kHz, 6 kHz, 8 kHz, 12 kHz,
- liczba sposobów prezentacji widma sygnału: 5.

Projekty pokrewne na www.media.avt.pl:

AVT-5767	Stereofoniczny wskaźnik poziomu wysterowania z funkcją Peak-Hold (EP 5/2020)
AVT-5748	SpectrumDFT – analizator widma sygnału akustycznego (EP 3/2020)
AVT-5712	Spectrum – prosty analizator widma sygnału akustycznego (EP 9/2019)
AVT-5678	Stereofoniczny wskaźnik wysterowania (EP 6/2019)
AVT-5585	Sterownik wskaźnika wychyłowego do wzmacniacza (EP 1/2018)
AVT-1716	Wskaźnik wysterowania z pamięcią (EP 12/2012)
AVT-1517	Wskaźnik nie tylko wysterowania (EP 9/2012)
AVT-5219	Wizualizator do Winampa na USB (EP 1/2010)
AVT-5210	Analizator widma sygnału audio (EP 11/2009)
AVT-2864	Analogowo-cyfrowy analizator widma (EdW 5/2008)
AVT-580	Procesor audio z equalizerem i analizatorem widma (EP 6-7/2004)
AVT-2375	Wskaźnik wysterowania 2×5 LED (EdW 9/1999)
AVT-2353	Pseudoanalogowy VU-metr (EdW 4/1999)
AVT-1190	Wskaźnik wysterowania (EP 8/1998)

*** 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:

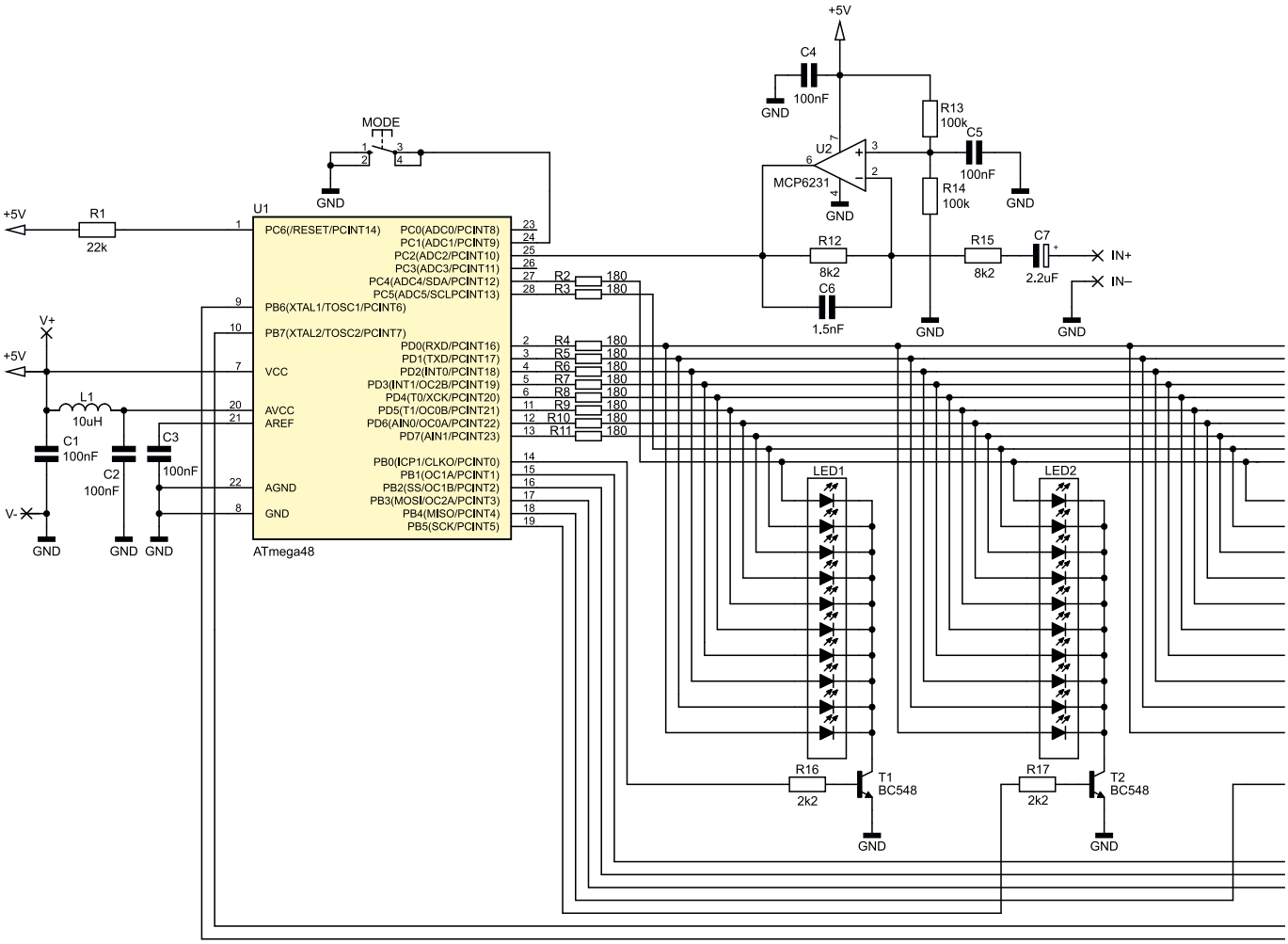
- wersja [**C**] – zmontowany, uruchomiony i przetestowany zestaw [**B**] (elementy wlutowane w płytkę PCB)
- wersja [**A**] – płytka drukowana bez elementów i dokumentacji

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 ideowy analizatora Spectra

Budowa i działanie

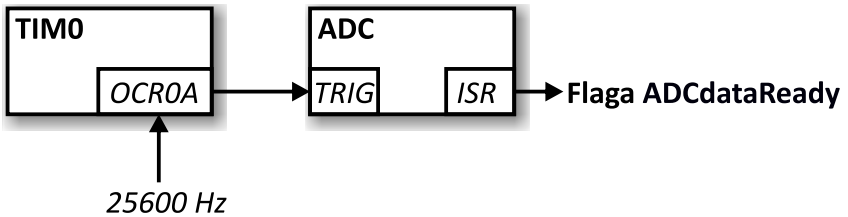
Schemat ideowy urządzenia został pokazany na **rysunku 1**. Jak widać, zaprojektowano bardzo prosty układ mikroprocesorowy, którego sercem jest niewielki mikrokontroler firmy Microchip (dawniej Atmel) o oznaczeniu ATmega48 taktowany wewnętrznym, wysokostabilnym sygnałem zegarowym o częstotliwości 8 MHz. Bardziej doświadczonych konstruktorów dziwić może fakt tak niskiej częstotliwości taktowania mikrokontrolera, biorąc pod uwagę rodzaj zadań, jakie będzie miał do wykonania (DSP). Uwierzcie mi na słowo, że jest to częstotliwość zupełnie wystarczająca. Przy okazji pozbywamy się dodatkowych elementów, które należałoby zastosować w celu zwiększenia częstotliwości taktowania poza wartości dostępne wewnątrz struktury naszego układu (mowa o zewnętrznym rezonatorze kwarcowym).

Głównym zadaniem naszego mikrokontrolera jest próbkowanie sygnału audio dostarczanego na wyprowadzenie ADC2 (PC2), co wykonywane jest przy użyciu wbudowanego przetwornika ADC pracującego w trybie 8-bitowym (tak naprawdę w trybie 10-bitowym, lecz z uwagi na wyrównanie wyniku konwersji do lewej łatwiej jest czytać 8-bitową wartość będącą wynikiem przetwarzania), zasilanego sygnałem zegarowym o częstotliwości 1 MHz, co zapewnia odpowiednio krótki czas przetwarzania sygnału (ok. 15,5 µs). Ten wymagany krótki czas konwersji wynika z dość dużej częstotliwości próbkowania wejściowego sygnału audio, a mianowicie z wartości 25600 próbek na sekundę. To z kolei wynika z pożądanego pasma przetwarzanego sygnału, które w tym przypadku, zgodnie z twierdzeniem Nyquista, wyniesie 12,8 kHz.

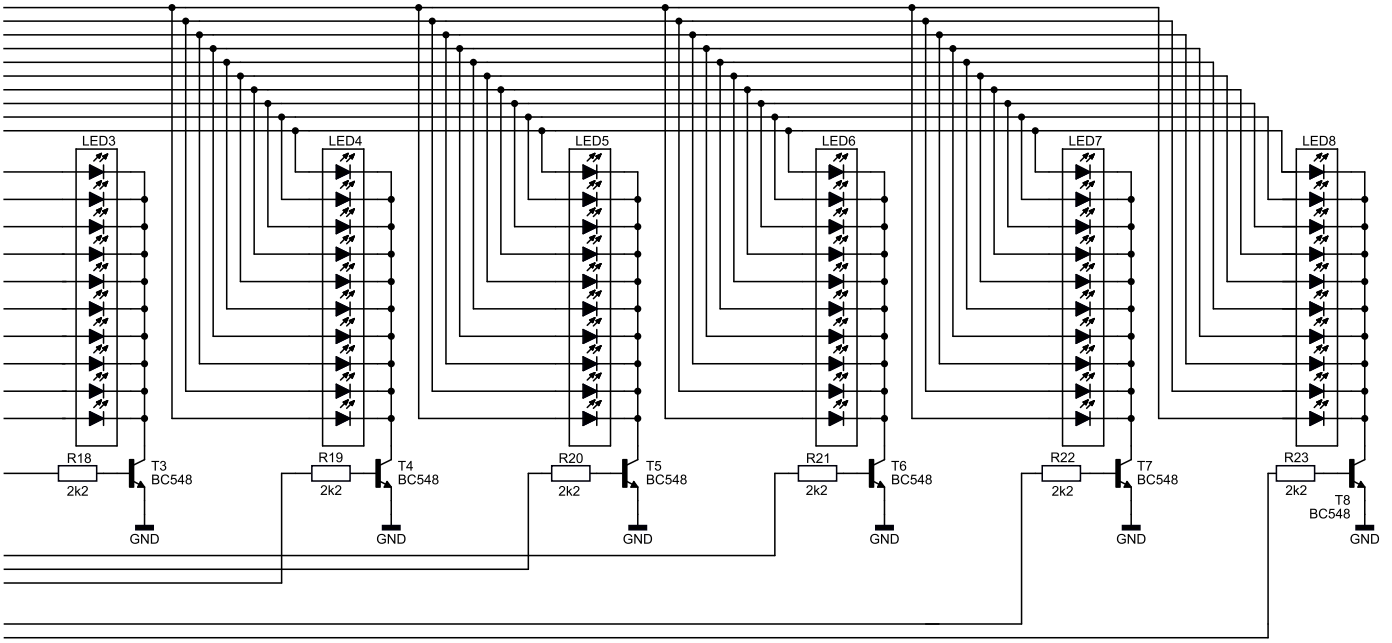
Zanim jednak sygnał audio trafi na wejście przetwornika ADC wbudowanego w strukturę mikrokontrolera, podlega filtracji w obwodzie prostego filtra dolnoprzepustowego pierwszego rzędu o częstotliwości granicznej około 12,8 kHz zbudowanego z użyciem wzmacniacza operacyjnego *rail-to-rail* pod postacią układu scalonego U2 (MCP6231) oraz kilku elementów biernych. Wzmocnienie w obwodzie filtra ustawiono na wartość 1, lecz można je skalować w górę, zmieniając stosunek rezystancji R12/R15, jednak cały czas pamiętając o częstotliwości granicznej wyznaczonej wzorem $fc=1/(2\cdot\pi\cdot R12\cdot C6)$. Filtr w takiej konfiguracji odwraca fazę sygnału, lecz z uwagi na spolaryzowanie wejścia nieodwracającego wzmacniacza operacyjnego połową napięcia zasilania oraz obecność kondensatora C7 to odwrócenie fazy nie ma znaczenia, jeśli chodzi o dalszą analizę DSP, gdyż sygnały przetwarzane przez przetwornik ADC znajdą się w zakresie napięć 0…5 V.

Dla porządku dodam, że wyzwalaczem konwersji wbudowanego przetwornika ADC jest układ czasowo-licznikowy Timer0 pracujący w trybie CTC, dla którego parametry dobrano w taki sposób, by porównanie zachodziło 25600 razy na sekundę.

Schemat blokowy systemu akwizycji danych urządzenia Spectra został pokazany na **rysunku 2**. W tym miejscu warto również



Rysunek 2. Schemat blokowy systemu akwizycji danych urządzenia Spectra



zaznaczyć, że producent mikrokontrolera, firma Microchip, podaje w stosownej dokumentacji dotyczącej przetwornika ADC dość niejednoznaczne, a przynajmniej niepełne informacje. Z jednej strony zaleca się, by częstotliwość taktowania przetwornika ADC nie była większa niż 200 kHz, zaś z drugiej podaje, że jeśli nie zależy nam na pełnej rozdzielczości peryferium (10 bitów), to częstotliwość tę możemy zwiększyć aż do 1 MHz (wartości, skądinąd wziętej z jeszcze innej części dokumentacji). Brak jednak jakichkolwiek wzorów czy informacji, w jaki sposób powiązać oczekiwaną rozdzielczość przetwornika z dopuszczalną częstotliwością taktowania tegoż peryferium. Na szczęście z pomocą przychodzi nam Internet i niezmiernie ciekawy, praktyczny artykuł umieszczony pod adresem https://bit.ly/3oWFoJZ, gdzie empirycznie dowiedziono, jakie są limity tejże konstrukcji. Szczerze polecam lekturę tego artykułu, gdyż w sposób bardzo prosty tłumaczy zawiłe mechanizmy wpływające na proces akwizycji danych.

Wróćmy do schematu naszego urządzenia. Mikrokontroler realizuje (poza procesem akwizycji danych i DSP) obsługę wyświetlacza LED zbudowanego z ośmiu 10-punktowych linijek LED połączonych w konfiguracji wspólnej katody oraz realizuje obsługę przycisku MODE przeznaczonego do zmiany sposobu prezentacji widma sygnału. Sterowanie pracą wyświetlacza LED jest wykonywane z użyciem mechanizmu multipleksowania, za który odpowiedzialny jest wbudowany w mikrokontroler układ czasowo-licznikowy Timer1 skonfigurowany w taki sposób, by generował przerwanie od porównania (tryb CTC układu) 480 razy na sekundę, czyli 60 razy na każdą z linijek LED. W przerwaniu takim w pierwszej kolejności wygaszane są wszystkie wspólne katody wyświetlaczy LED (porty kolumn), następnie na porty PD0…PD7 oraz PC4 i PC5 (porty wierszy) wystawiana jest wartość odpowiadająca kolejnemu pasmu częstotliwościowemu. Następnie włączana jest kolejna wspólna katoda sterowana z portów PB0…PB7 mikrokontrolera (poprzez tranzystory sterujące T1…T8). Jest to typowe rozwiązanie mechanizmu multipleksowania stosowane w wielu systemach mikroprocesorowych, dzięki któremu możliwe jest sterowanie dużą liczbą diod LED przy ograniczonej liczbie portów mikrokontrolera. Dzięki temu ograniczamy przy okazji sumaryczny prąd pobierany przez wyświetlacz LED, który w tym przypadku wynosi maksymalnie 10·16 mA. Tyle w kwestiach sprzętowych, w związku z czym przejdźmy do najważniejszych zagadnień programowych.

```
Ustawienia Fusebitów (ważniejszych):
CKSEL3…0: 0010
SUT1…0: 10
CKDIV8: 1
EESAVE: 1
```

Program sterujący

Pora na odrobinę kodu związanego z systemem akwizycji danych. Kod odpowiedzialny za inicjalizację układu czasowo-licznikowego Timer0 będącego wyzwalaczem akwizycji oraz konfigurację przetwornika ADC pokazano na **listingu 1**. Dalej, na **listingu 2** pokazano kod odpowiedzialny za akwizycję danych przetwornika ADC. Jak widać, zebraniu kompletnej porcji danych towarzyszy zatrzymanie akwizycji danych (wyłączenie timera Timer0) oraz ustawienie flagi *ADCdataReady*, dzięki czemu możliwe jest przetworzenie danych przez program główny aplikacji.

Kilka słów uwagi wymaga zapis ADCdata[Idx++]=ADCH–128. Odjęcie wartości 128 od wyniku przetwarzania przetwornika ADC (ADCH) wynika z faktu, że wejście ADC2 przetwornika spolaryzowano połową wartości napięcia zasilania, a więc jednocześnie napięcia referencyjnego po to, aby układ mógł przetwarzać rzeczywiste sygnały audio, które jak wiemy, przyjmują również wartości poniżej 0.

Listing 1. Kod odpowiedzialny za inicjalizację akwizycji danych ADC

```
void initADC(void){
    //Konfiguracja Timera0 w trybie CTC, by zdarzenie porównania występowało 25600 razy na sekundę
    TCCR0A = (1<<WGM01);  //Tryb CTC
    TCCR0B = (1<<CS01);  //Preskaler = 8 (1MHz)
    OCR0A = 38;  //Zdarzenie porównania 25600 razy na sekundę

    //Vref = AVcc (5V), wyrównanie wyniku do lewej (rozdzielczość 8-bitów), MUX = ADC2 (PC2)
    ADMUX = (1<<REFS0)|(1<<ADLAR)|(1<<MUX1);
    //Włączenie ADC, automatyczne wyzwalanie konwersji, zezwolenie na przerwanie ADC, Preskaler = 8 (1MHz)
    ADCSRA = (1<<ADEN)|(1<<ADATE)|(1<<ADIE)|(1<<ADPS1)|(1<<ADPS0);
    //Wybór źródła wyzwalania konwersji ADC: Timer/counter0 compare match A
    ADCSRB = (1<<ADTS1)|(1<<ADTS0);
    //Wyłączenie cyfrowych obwodów wejściowych na pinie ADC2 (PC2)
    DIDR0 = (1<<ADC2D);
}
```

Listing 2. Kod odpowiedzialny za akwizycję danych przetwornika ADC

```
ISR(ADC_vect){
    static uint8_t Idx;

    ADCdata[Idx++] = ADCH - 128;

    if(Idx == SAMPLES){
        Idx = 0;
        ADCdataReady = 1;
        STOP_ACQUISITION;
    }

    //Kasujemy flagę Timer0
    //Output Compare A Match
    //aby umożliwić wyzwalanie ADC
    // timerem
    TIFR0 = (1<<OCF0A);
}
```

Listing 3. Kod funkcji odpowiedzialnej za obliczenie dyskretnej transformaty Fouriera

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

//Czas wykonania 1.45ms
uint16_t DFT(uint8_t i, uint8_t dftSensivity){
    uint16_t a, b, Power;
    int32_t Re, Im;

    //Obliczamy moc sygnału dla szukanego
    //prążka częstotliwościowego i < (SAMPLES/2)+1

    Re = Im = a = 0;
    b = 3*SAMPLES/4;

    for (uint8_t j=0; j<SAMPLES; ++j){
        Re += (ADCdata[j] * Twiddle[a % SAMPLES])/MULF;
        Im -= (ADCdata[j] * Twiddle[b % SAMPLES])/MULF;
        a += i;
        b += i;
    }

    if(dftSensivity == SENSIVITY_HI){
        Power = (Re*Re + Im*Im)/
        2048UL; else  Power = (Re*Re + Im*Im)/
        16384UL;
    }

    return Power;
}
```

Za przetworzenie zebranych danych, czyli obliczenie dyskretnej transformaty Fouriera z próbek sygnału zebranych w tablicy *ADCdata[]*, odpowiedzialna jest funkcja pokazana na **listingu 3**. Wprowadzono możliwość regulacji „czułości" funkcji DFT, a to z uwagi na przewagę tonów o niskich częstotliwościach w rzeczywistym sygnale audio oraz potrzebę zmniejszenia „czułości" funkcji w tym zakresie. Wynikiem działania funkcji *DFT()* jest obliczenie mocy poszczególnych prążków częstotliwości (specyfikowanych wartością argumentu *i*). I właśnie to zadanie stanowi główny problem obliczeniowy, o którym wspomniano na wstępie artykułu. Wynika to z liczby obliczeń stałoprzecinkowych wykonywanych w ramach pętli, z jakiej składa się wspomniana funkcja. Jak widać, liczba tych obliczeń zależy bezpośrednio od liczby punktów transformaty Fouriera (*SAMPLES*), która w naszym przypadku wynosi 128. Z kolei liczba punktów transformaty determinuje odległość kolejnych prążków mocy (tzw. *BIN*), a wynika z zależności: BIN=częstotliwość próbkowania sygnału/liczba punktów transformaty (SAMPLES). Dla naszego przypadku BIN=200·(25,6 kHz/128), co oznacza, że kolejne wartości częstotliwości, dla których liczona jest moc sygnału, są wielokrotnością wartości 200 Hz. Wartość ta jest z kolei kompromisem pomiędzy rozdzielczością mocy (BIN) a czasem niezbędnym na wykonanie funkcji *DFT()* przy przyjętej liczbie punktów transformaty (SAMPLES). W naszym przypadku czas ten wynosi około 1,45 ms·8 (gdyż liczymy DFT wyłącznie dla wybranych 8 punktów). To determinuje częstotliwość odświeżania wykresu widma (tzw. framerate), która w tym przypadku wynosi około 80 Hz, a więc bardzo dużo, jak na nasze potrzeby. Dalsze zwiększanie liczby punktów transformaty (SAMPLES), choć pożądane, zmniejszyłoby częstotliwość odświeżania wykresu widma 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 ATmega i maksymalnej, dostępnej częstotliwości taktowania przetwornika ADC (1 MHz). Zresztą nie bez powodu jako liczbę punktów transformaty wybrano wartość 128 będącą potęgą liczby 2. Wynikało to z faktu dzielenia, jakie wykonywane jest we wspomnianej wcześniej pętli, a jak doskonale wiemy dzielenie przez liczbę będącą całkowitą potęgą liczby 2 jest o rząd wielkości szybsze, aniżeli dzielenie przez jakąkolwiek inną liczbę, gdyż tak naprawdę jest to zwykłe przesuwanie wartości dzielonej w prawo.

Tyle w kwestii ograniczeń implementacji. Wróćmy zatem do naszej funkcji *DFT()*. Jak widać, i o czym powiedziano na wstępie artykułu, funkcja korzysta z tablicy *Twiddle[]* współczynników wektora rotującego, której wyznaczeniem zajmuje się funkcja *calculateTwiddleFactors()* pokazana na **listingu 4**. Oczywiście funkcja z listingu jest niejako nadmiarowa, gdyż współczynniki takie moglibyśmy wyznaczyć sobie w arkuszu kalkulacyjnym i zapisać na stałe w pamięci programu, co zmniejszyłoby zajętość pamięci RAM mikrokontrolera oraz kod obsługi aplikacji. Ja zdecydowałem się na wyznaczenie ich w trakcie działania programu, gdyż po pierwsze, jest to szybsze, zaś po drugie i nie mniej ważne, mikrokontroler nasz wyposażono w dużą ilość pamięci RAM, która jest nieużywana.

Warto podkreślić, że przed wykonaniem funkcji *DFT()* zebrana tablica danych wejściowych *ADCdata[]* poddawana jest okienkowaniu, które ma na celu ograniczenie tak zwanych wycieków widma sygnału, które samo w sobie wpływa niekorzystanie na wynikowe widmo sygnału. Zastosowana funkcja okna jest typu Hanna (Hanninga), zaś wyznaczeniem stosownych współczynników okna (*Window[]*) zajmuje się funkcja

Wykaz elementów:

Rezystory: (miniaturowe 1/4 W, raster 0,2")
R1: 22 kΩ
R2…R11: 180 Ω
R12, R15: 8,2 kΩ
R13, R14: 100 kΩ
R16…R23: 2,2 kΩ

Kondensatory:
C1…C5: 100 nF (raster 0,1")
C6: 1,5 nF (raster 0,1")
C7: 2,2 µF/10 V elektrolityczny (ø 4 mm, raster 2 mm)

Półprzewodniki:
LED1…LED8: linijka diodowa zielono-czerwona typu OSX10201-GGR1
T1…T8: BC548 (TO-92)
U1: ATmega48 (DIL-28)
U2: MCP6231 (DIL-08)

Pozostałe:
L1: dławik osiowy 10 µH typu DLA10-MN (miniaturowy, raster 0,2")
MODE: mikroswitch TACT 9 mm

Listing 4. Kod funkcji odpowiedzialnej za wyznaczenie współczynników wektora rotującego

```
#define PI2 6.2832  //2*Pi

void calculateTwiddleFactors(void){
    for(uint8_t i=0; i<SAMPLES; ++i)
        Twiddle[i] = (int8_t) (MULF*cos(i*PI2/SAMPLES));
}
```

Listing 5. Kod funkcji odpowiedzialnej za wyznaczenie współczynników okna Hanna (Hanninga)

```
void calculateWindowFactors(void){  //Okno Hanninga
    for(uint8_t i=0; i<SAMPLES; ++i)
        Window[i] = (int8_t) (MULF*(0.5-0.5*cos(i*PI2/(SAMPLES-1))));
}
```

Listing 7. Funkcja inicjalizacyjna mechanizmu multipleksowania

```
void initMultiplex(void){
    //Porty wierszy, jako porty wyjściowe ze stanem nieaktywnym "0"
    ROW_LSB_DDR = 0xFF;
    ROW_MSB_DDR |= (1<<ROW_MSB_1)|(1<<ROW_MSB_0);
    //Port kolumn, jako port wyjściowy ze stanem nieaktywnym "0"
    COLUMN_DDR = 0xFF;

    //Konfiguracja licznika Timer1 w celu generowania przerwania
    //do obsługi multipleksowania wyświetlacza LED (480 Hz)
    TCCR1B = (1<<WGM12)|(1<<CS12);  //Tryb CTC, Preskaler = 256 @ 8MHz
    //480 Hz (przerwanie 480 razy na sekundę,
    //60 razy na sekundę dla każdego wyświetlacza LED)
    OCR1A = 64;
    //Uruchomienie przerwania Output Compare Match A (od porównania)
    TIMSK1 = (1<<OCIE1A);
}
```

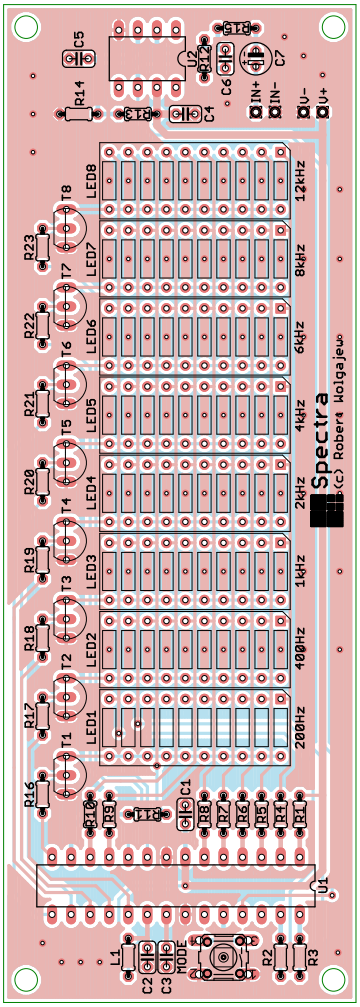
Listing 6. Plik nagłówkowy mechanizmu multipleksowania

```
#define ROW_LSB_PORT PORTD
#define ROW_LSB_DDR DDRD
#define ROW_MSB_PORT PORTC
#define ROW_MSB_DDR DDRC
#define ROW_MSB_0 PC5
#define ROW_MSB_1 PC4

#define COLUMN_PORT PORTB
#define COLUMN_DDR DDRB
#define COLUMN_0 PB0
#define COLUMN_1 PB5
#define COLUMN_2 PB4
#define COLUMN_3 PB3
#define COLUMN_4 PB2
#define COLUMN_5 PB1
#define COLUMN_6 PB7
#define COLUMN_7 PB6
#define COLUMN_BLANK COLUMN_PORT = 0x00

//Zmienna przechowująca wartość
//wyświetlaną na kolejnych słupkach
//wyświetlacza
extern uint16_t Led[8];
```

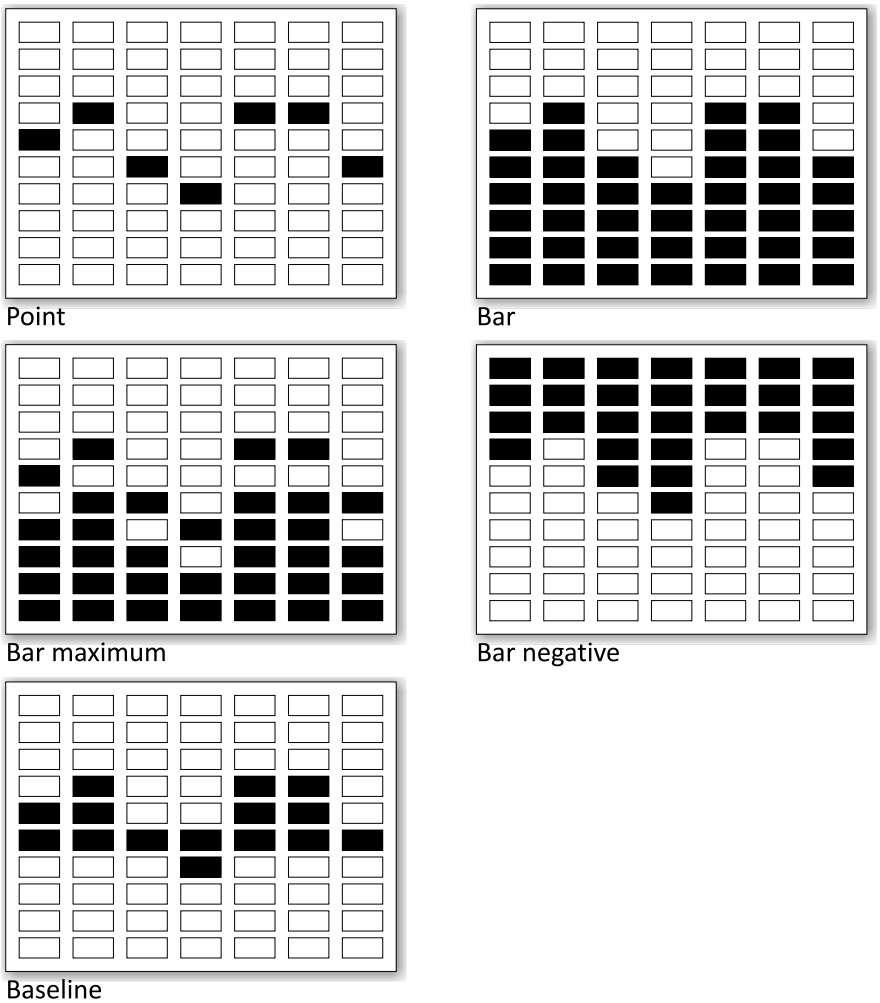
calculateWindowFactors(), której ciało pokazano na **listingu 5**.

Spróbkowany przebieg wejściowy zebrany w tablicy *ADCdata[]* jest przemnażany przez funkcję okna *Window[]* przed wykonaniem transformaty Fouriera w ramach funkcji *DFT()*. Na sam koniec, po wykonaniu funkcji *DFT()*, a przed wyświetleniem widma mocy sygnału, stosowne moce przeliczane są do skali logarytmicznej (dB), co wynika głównie z dużej dynamiki sygnału *Power[]* i konieczności pokazania go na ograniczonej rozdzielczością pionową wyświetlacza LED (10 diod) skali sygnału. Przeliczenie, o którym mowa, wykonywane jest według następującej zależności: *Power[i]*=3,333·log10(*Power[i]*).

Tyle, jeśli chodzi o zagadnienia przetwarzania sygnałów DSP. Na koniec omówię w skrócie stosowne funkcje obsługi



Rysunek 3. Schemat płytki PCB wraz z rozmieszczeniem elementów



Rysunek 4. Wizualizacje przedstawiające 5 trybów wyświetlania informacji o widmie sygnału audio

mechanizmu multipleksowania odpowiadające za wyświetlanie danych naszego ekranu LED. Zacznijmy od pliku nagłówkowego, którego zawartość pokazano na **listingu 6**. Dalej, na **listingu 7** pokazano funkcję inicjalizacyjną mechanizmu multipleksowania, która ustawia odpowiednie stany logiczne na portach wspólnych katod i anod wyświetlacza LED oraz inicjuje stosowny Timer sprzętowy.

Kolejną, prezentowaną funkcją jest funkcja obsługi przerwania realizująca mechanizm multipleksowania, której zawartość pokazano na **listingu 8**. Wspomniana funkcja korzysta z deklaracji zmiennej umieszczonej w pamięci Flash mikrokontrolera:

```
const uint8_t Columns[8] PROGMEM
= {
  (1<<COLUMN_0), (1<<COLUMN_1),
  (1<<COLUMN_2), (1<<COLUMN_3),
  (1<<COLUMN_4), (1<<COLUMN_5),
  (1<<COLUMN_6), (1<<COLUMN_7)};
```

Wprowadzenie tej zmiennej upraszcza dostęp do portów kolumn. Tyle w kwestiach programowych, przejdźmy zatem do zagadnień montażowych.

Listing 8. Funkcja obsługi przerwania realizująca mechanizm multipleksowania

```
//Zmienna przechowująca wartość
//wyświetlaną na kolejnych słupkach wyświetlacza
uint16_t Led[8];

ISR(TIMER1_COMPA_vect){
    //Numer kolejnego słupka do wyświetlenia
    static uint8_t Nr;

    //Wygaszenie wszystkich kolumn
    COLUMN_BLANK;

    //Wystawienie właściwych stanów na portach wierszy
    ROW_LSB_PORT = Led[Nr] & 0xFF;
    if(Led[Nr] & 0b100000000)
        ROW_MSB_PORT |= (1<<ROW_MSB_0);
    else
        ROW_MSB_PORT &= ~(1<<ROW_MSB_0);

    if(Led[Nr] & 0b1000000000)
        ROW_MSB_PORT |= (1<<ROW_MSB_1);
    else
        ROW_MSB_PORT &= ~(1<<ROW_MSB_1);

    //Załączenie odpowiedniej kolumny
    COLUMN_PORT = pgm_read_byte(&Columns[Nr]);
    //Wybranie kolejnej kolumny (słupka LED)
    Nr = (Nr+1) & 0x07;
}
```

Montaż i uruchomienie

Schemat montażowy urządzenia Spectra został pokazany na **rysunku 3**. Zaprojektowano bardzo zwarty obwód drukowany z zastosowanie wyłącznie elementów THT. Montaż urządzenia rozpoczynamy od przylutowania wszystkich rezystorów, następnie lutujemy dławik L1, kondensatory, potem układy scalone (które warto wyposażyć w stosowne podstawki) a na końcu wyświetlacze LED1… LED8 oraz wyłącznik MODE.

Poprawnie zmontowany układ nie wymaga żadnych regulacji i powinien działać tuż po włączeniu zasilania. Dla dociekliwych warto wspomnieć, iż program obsługi aplikacji urządzenia Spectrum przewiduje 5 trybów wyświetlania informacji o widmie sygnału audio, których przykładowe wizualizacje obrazuje **rysunek 4**. Co warto podkreślić, tryb trzeci (Bar maximum) integruje dodatkową funkcjonalność w postaci pokazywania wartości szczytowej w każdym z pasm częstotliwościowych.

Podsumowanie

Testy końcowe urządzenia z użyciem generatora arbitralnego wykazały bardzo dużą selektywność układu, jeśli chodzi o reakcję na wybrane częstotliwości analizowane przez nasze urządzenie oraz nieco mniejszą „odpowiedź" układu w przypadku „zwykłych" sygnałów audio (zwłaszcza tych o wysokich częstotliwościach), co skłoniło mnie do zwiększenia czułości funkcji liczącej dyskretną transformatę Fouriera DFT oraz do zmiany skali wynikowej tej funkcji na skalę logarytmiczną, aby poprawnie odwzorować bardzo dużą dynamikę sygnału.

Robert Wołgajew, EP